

BUKU KHUSUS · PYTHON

PYTHON UNTUK GIS

Dari Nol sampai Bisa Berimprovisasi

PyQGIS dan ArcPy (Technical GIS Textbook)
untuk Kehutanan dan Pertanian

Badar Mubarok Yogaswara

DAFTAR ISI

Pendahuluan: Selamat Datang di Dunia Skrip **1**
Cara membaca buku ini 1
Dari mana skrip di buku ini berasal 1
Kenalan dengan KPH Contoh 2
Perangkat dan data 2
Hak cipta dan penggunaan 3

BAB 1: Kenalan dengan Skrip: Menyuruh Komputer dengan Tulisan **4**
Studi kasus: "Tolong buatkan ringkasan dua puluh petak" 4
Konsep: skrip dalam tiga kalimat 4
Mengapa Python untuk GIS? 4
Apa yang akan Anda bisa setelah buku ini? 5
Peta buku ini 6
Tiga kebiasaan belajar 6

BAB 2: Menyiapkan Peralatan: Memasang Python dan QGIS **8**
Studi kasus: "Komputer baru untuk Analis Muda" 8
Konsep: dua peralatan, dua jalan 8
Langkah: memasang di tiga sistem operasi 9
Memeriksa pemasangan 9

BAB 3: Tempat Menulis dan Menjalankan: Konsol, Editor, dan Terminal **11**
Studi kasus: "Di mana saya mengetiknya?" 11
Konsep: tiga tempat dalam tiga kalimat 11
Tempat 1: Konsol Python QGIS 11
Tempat 2: Editor skrip di dalam konsol 12
Tempat 3: Terminal 12
Skrip mandiri butuh persiapan 13

BAB 4: Kotak Bernama: Variabel, Angka, dan Teks **16**
Studi kasus: "Luas KPH Alpha berapa?" 16
Konsep: variabel dalam tiga kalimat 16
Tiga jenis nilai yang paling sering dipakai 16
Dua perintah penolong: print dan komentar 17
Langkah: mencoba di tempat mengetik Python 17
Skrip yang dibedah: lima hal Python untuk memulai 17

BAB 5: Rak dan Daftar Isi: Daftar, Tuple, Set, dan Kamus **20**
Studi kasus: "Dua puluh petak, dua puluh kotak?" 20
Konsep: daftar dan kamus dalam tiga kalimat 20
Daftar: rak bernomor 21
Tuple dan set: dua sepupu daftar 21
Kamus: daftar isi buku 22
Skrip yang dibedah: daftar dan kamus di b1_00 23

BAB 6: Persimpangan dan Putaran: Keputusan dan Perulangan **26**
Studi kasus: "Hanya petak yang besar" 26
Konsep: keputusan dan perulangan dalam tiga kalimat 26

Keputusan: if, elif, else 27
Perulangan: for 27
Alat bantu perulangan: enumerate, break, range 28
Comprehension: perulangan satu baris 28
Skrip yang dibedah: perulangan di b1_00 29

BAB 7: Resep Bernama: Fungsi 32
Studi kasus: "Meter persegi ke hektare, lagi dan lagi" 32
Konsep: fungsi dalam tiga kalimat 32
Membuat dan memanggil fungsi 33
Parameter bawaan dan docstring 33
Menerima argumen bernama sebanyak apa pun: kwargs 34
Fungsi tanpa nama: lambda 34
Skrip yang dibedah: fungsi di b1_00 35

BAB 8: Gudang Perkakas: Modul, Import, dan Jalur Berkas 38
Studi kasus: "Skrip jalan di komputer saya, tapi tidak di komputer kantor" 38
Konsep: modul dan jalur dalam tiga kalimat 38
Import: membuka laci perkakas 39
Perkakas berkas: os dan shutil 39
Menaruh alamat di satu tempat 40
Tempat Python mencari modul: sys.path 41
Skrip yang dibedah: pembuka b2_03_digitasi.py 41

BAB 9: Catatan dan Laporan: Berkas dan Merakit Teks 45
Studi kasus: "Kirim ringkasannya dalam bentuk tabel" 45
Konsep: berkas dan teks dalam tiga kalimat 45
Merakit teks: tiga cara 46
Menulis dan membaca berkas teks 46
Tabel dalam berkas: CSV 47
Data bersusun: JSON 47
Skrip yang dibedah: ekspor ke CSV di b2_07 48

BAB 10: Membaca Petunjuk Komputer: Galat dan Cara Menanganinya 51
Studi kasus: "Lima berkas kiriman tim lapangan" 51
Konsep: galat dalam tiga kalimat 51
Membaca pesan galat 52
Jenis galat yang paling sering muncul 52
Menangani galat: try dan except 53
Menimbulkan galat sendiri: raise dan assert 53
Skrip yang dibedah: memeriksa masukan sebelum dipakai 54

BAB 11: Cetakan dan Pembungkus: Objek, Kelas, dan Dekorator 59
Studi kasus: "Kok perintah QGIS ditulis dengan titik di tengah?" 59
Konsep: kelas dan objek dalam tiga kalimat 59
Membuat kelas sendiri 60
Pewarisan: kelas turunan 60
Dekorator: pembungkus fungsi 61
Skrip yang dibedah: dua gaya algoritma Processing 62

BAB 12: Kotak Perkakas Tambahan: Pustaka Ilmiah 66
Studi kasus: "Rata-rata dari sembilan puluh ribu piksel" 66

Konsep: pustaka dalam tiga kalimat....66
numpy: kalkulator untuk deretan angka....67
osgeo (GDAL): pembaca raster....67
matplotlib: pembuat grafik....69
scipy dan scikit-learn: statistik dan pembelajaran mesin....70
Bagaimana mengenali pustaka di skrip orang lain....70

BAB 13: Belajar Mandiri: Mencari, Mencoba, Mengubah....73
Studi kasus: "Tidak ada contohnya di buku!"....73
Konsep: siklus belajar dalam tiga kalimat....73
Langkah 1: Tanya objeknya sendiri....74
Langkah 2: Tanya objek QGIS....74
Langkah 3: Baca dokumentasi resmi....75
Langkah 4: Coba satu baris, lalu ubah....75
Langkah 5: Membaca kode orang lain dengan tenang....76
Skrip yang dibedah: satu baris, lalu jadi pola....76

BAB 14: Mengetuk Pintu QGIS: Memuat, Membaca, dan Menyimpan dengan Skrip....80
Studi kasus: "Peta saya bisa dibuka lagi besok?"....80
Konsep: PyQGIS dalam tiga kalimat....80
Konsep: proyek, layer, dan fitur....82
Dua cara menjalankan: di dalam QGIS dan mandiri....83
Skrip 14.1: Lima hal Python sebelum masuk QGIS....84
Skrip 14.2: Memuat layer dan mengambilnya dengan nama....86
Skrip 14.3: Membaca fitur dan atribut, lalu menulis CSV....90
Skrip 14.4: Menyimpan proyek dan membukanya lagi....94

BAB 15: Lemari Data yang Patuh: Atribut, Seleksi, Digitasi, dan Laporan dengan Skrip....99
Studi kasus: "Ada berapa petak, dan jenis apa saja?"....99
Konsep: tabel atribut dalam tiga kalimat....99
Skrip 15.1: Membaca tabel atribut....100
Skrip 15.2: Memilih fitur dengan seleksi....104
Skrip 15.3: Menggambar dan membelah dengan koordinat....108
Skrip 15.4: Memeriksa dan memperbaiki geometri....113
Skrip 15.5: Menambah kolom dengan rumus....117
Skrip 15.6: Menggabungkan tabel dengan join....121
Skrip 15.7: Membuat rekap untuk Kepala Seksi....126

BAB 16: Kisi Piksel yang Bisa Dihitung: Raster dan Citra dengan Skrip....133
Studi kasus: "Seberapa hijau KPH kita, dan berapa luas lereng curamnya?"....133
Konsep: raster dalam tiga kalimat....133
Skrip 16.1: Memeriksa isi sebuah citra....135
Skrip 16.2: Menumpuk empat band menjadi satu citra....138
Skrip 16.3: Dari satu peta tinggi menjadi empat turunan....141
Skrip 16.4: Memotong raster dengan poligon....143
Skrip 16.5: Mengelompokkan lereng menjadi tiga kelas....146
Skrip 16.6: Menghitung NDVI dan merata-ratakan per KPH....149
Skrip 16.7: Satu tabel ringkasan untuk Kepala Seksi....153

BAB 17: Dari Data ke Lembar Peta: Mewarnai, Memberi Label, dan Mencetak dengan Skrip....159
Studi kasus: "Peta volume kayu untuk rapat besok"....159
Konsep: peta jadi dalam tiga kalimat....159

Sebelum mulai: urutan menjalankan dan bahan....160
Skrip 17.1: Muat layer dan beri warna per jenis....160
Skrip 17.2: Klasifikasi angka menjadi kelas warna....165
Skrip 17.3: Warna untuk peta tinggi dan bayangan bukit....171
Skrip 17.4: Label berisi kode petak dan volume....175
Skrip 17.5: Menyusun lembar peta A4....179
Skrip 17.6: Ekspor ke PDF dan PNG tiga resolusi....186
Skrip 17.7: Proyek mini dari data mentah sampai PDF....189

BAB 18: Dari Peta ke Jawaban: Buffer, Overlay, Join, dan Model dengan Skrip....198
Studi kasus: "Berapa hektare yang benar-benar bisa dipanen?"....198
Konsep: geoprocessing dalam tiga kalimat....198
Sebelum mulai: bahan, jalur, dan urutan....199
Skrip 18.1: Memeriksa data sebelum dianalisis....199
Blok pembuka: baris 1-28....201
Isi khusus Skrip 18.1: baris 29-46....202
Skrip 18.2: Zona aman di sekeliling sungai....204
Skrip 18.3: Dua peta bertumpuk, tiga cara membaca tumpukannya....208
Skrip 18.4: Melebur batas dan merekap luas....212
Skrip 18.5: Menghubungkan data dari letaknya....217
Jebakan nomor predikat: dua alat, dua daftar....222
Skrip 18.6: Merangkai alat menjadi model....223
Skrip 18.7: Laporan areal efektif dalam satu skrip....229

BAB 19: Piksel Bicara Angka: Analisis Raster dan Klasifikasi Citra dengan Skrip....235
Studi kasus: "Seberapa tinggi, seberapa curam, lahan apa?"....235
Konsep: raster dalam skrip, dalam tiga kalimat....235
Dua cara bekerja, dan urutan ketujuh skrip....236
Siapkan dulu: data, folder, dan tempat menjalankan....236
Skrip 19.1: Berhitung dengan piksel....237
Skrip 19.2: Membaca bentuk bumi dari DEM....242
Skrip 19.3: Dari angka ke kelas lereng....246
Skrip 19.4: Biarkan citra yang bicara....251
Skrip 19.5: Ajari citra mengenal lahan....259
Skrip 19.6: Seberapa benar peta ini?....265
Skrip 19.7: Lahan terbuka di lereng curam....273

BAB 20: Titik yang Bisa Dipercaya: Survei Lapangan dan Basis Data dengan Skrip....280
Studi kasus: "Titik ini benar-benar di sini?"....280
Konsep: survei yang bisa dipercaya, dalam tiga kalimat....280
Siapkan dulu: folder, data, dan urutan skrip....281
Skrip 20.1: Seberapa tepat titik GNSS?....282
Skrip 20.2: Menyiapkan lemari survei....286
Tiga lapis penjagaan: peta untuk Skrip 20.3 sampai 20.5....292
Skrip 20.3: Menanam domain nilai di dalam berkas....292
Skrip 20.4: Merancang formulir isian....297
Skrip 20.5: Pagar di dalam berkas....305
Skrip 20.6: Memeriksa mutu data survei....309
Skrip 20.7: Bertanya ke basis data dengan SQL....315
Skrip 20.8: Aturan kualitas sebagai ekspresi QGIS....321
Skrip 20.9: Dua surveyor, satu nomor yang sama....327

BAB 21: Memanggil Alat dari Skrip: Satu Perintah, Satu Analisis, Satu Resep untuk Banyak Berkas 334
Studi kasus: "Sempadan sungai, tiga puluh ubin, dan sore yang pendek" 334
Konsep: memanggil alat dengan formulir pesanan 334
Dua cara menjalankan, dan satu tambahan untuk Processing 335
Skrip 21.1: Memanggil satu alat, Buffer 336
Skrip 21.2: Merangkai alat vektor 340
Skrip 21.3: Alat raster, NDVI dan tinggi tajuk 343
Konsep: batch, satu resep untuk banyak berkas 347
Skrip 21.4: Batch sederhana, NDVI untuk semua ubin 347
Skrip 21.5: Satu putaran untuk tiap fitur 351
Konsep: latar belakang dan fungsi selesai 354
Skrip 21.6: Batch di latar belakang dengan QgsTask 354

BAB 22: Skrip yang Tidak Mudah Menyerah: Galat, Batch Tahan Galat, dan Ekspresi QGIS 361
Studi kasus: "Enam kiriman, satu berkas rusak, dan laporan yang menulis dirinya sendiri" 361
Konsep: galat dalam tiga kalimat 361
Skrip 22.1: Memeriksa masukan sebelum dipakai 362
Skrip 22.2: Batch tahan galat dengan laporan 366
Skrip 22.3: Pembungkus untuk alat Processing 372
Konsep: ekspresi QGIS dalam tiga kalimat 376
Skrip 22.4: Menghitung ekspresi dari Python 377
Skrip 22.5: Ekspresi di dalam alat Processing 384

BAB 23: Menyatukan Semuanya: Model, Proyek Terpadu, dan Ceklis Masuk M1 389
Studi kasus: "Sempadan 20, 30, atau 50 meter, lalu laporan awal bulan" 389
Konsep: model sebagai berkas, dalam tiga kalimat 389
Skrip 23.1: Membangun model lewat Python 390
Skrip 23.2: Menjalankan model dari Python 397
Konsep: proyek terpadu dan ceklis 400
Skrip 23.3: Proyek mini terpadu 400
Skrip 23.4: Ringkasan sebagai tugas latar belakang 410
Skrip 23.5: Ceklis masuk M1 414
Uji negatif: membuktikan ceklis bisa berkata GAGAL 420

BAB 24: Satu Perintah, Banyak Pekerjaan: NDVI, Validasi, dan Batch dengan Alat Processing 425
Studi kasus: "Hijaukah KPH kita, dan benarkah petanya?" 425
Konsep: Processing dalam tiga kalimat 425
Skrip 24.1: NDVI dari citra cloud (versi naskah) 426
Skrip 24.2: NDVI cloud, dengan layer bernama 431
Skrip 24.3: Akurasi peta dari titik survei 435
Skrip 24.4: NDVI banyak ubin di latar belakang 438
Skrip 24.5: Batch NDVI, versi dengan layer bernama 443
Ekspresi 24.6: teks laporan untuk Atlas (bukan Python) 445

BAB 25: Dari Lemari Berkas ke Gudang Data: PostGIS, Jaringan, Hidrologi, dan Kesesuaian Lahan dengan Skrip 450
Studi kasus: "Datanya ada di mana-mana, hasilnya beda-beda" 450
Konsep: gudang data dalam tiga kalimat 450
Sebelum mulai: tiga persiapan 451
Berkas pembantu: _inisialisasi.py 452

Skrip 25.1: Memasukkan berkas ke PostGIS dengan GDAL....454
Skrip 25.2: Memuat tabel PostGIS di QGIS dengan skrip....457
Skrip 25.3: Kueri yang sama di GeoPackage dan PostGIS....460
Skrip 25.4: Hitungan yang sama tanpa basis data....464
Skrip 25.5: Rute terpendek dan tercepat di QGIS....466
Skrip 25.6: Rute dan area layanan, QGIS lawan pgRouting....471
Skrip 25.7: Hidrologi dari peta ketinggian....474
Skrip 25.8: Peta curah hujan dari 24 stasiun....480
Skrip 25.9: Peta kesesuaian lahan dengan tumpang tindih berbobot....489
Skrip 25.10: Cadangan dan pemulihan dengan Bash....497
Sembilan berkas SQL yang hanya disebut....500

BAB 26: Hutan dalam Film Pendek: Deret Waktu, Perubahan, dan Klasifikasi dengan Skrip....503

Studi kasus: "Apa yang terjadi sejak dua tahun lalu?"....503
Konsep: deret waktu dalam tiga kalimat....503
Persiapan: folder, data, dan cara menjalankan....504
Kotak perkakas: m3_umum.py....506
Kelompok A: Menyusun deret, dari 24 berkas menjadi satu tumpukan....511
Konsep: tumpukan bertanggal dalam tiga kalimat....511
Skrip 26.1: Menyusun NDVI menjadi tumpukan 24 band....512
Skrip 26.2: Menggambar profil waktu empat titik....515
Skrip 26.3: Memberi tanggal agar bisa diputar seperti film....519
Kelompok B: Membersihkan awan....522
Konsep: tiga cara menyingkirkan awan....522
Skrip 26.4: Membuang awan dengan band SCL....523
Skrip 26.5: Komposit median dari beberapa tanggal....527
Skrip 26.6: Menambal lubang awan, lalu menguji tambalannya....530
Kelompok C: Mengukur irama tiap tutupan lahan....535
Konsep: irama dalam tiga kalimat....535
Skrip 26.7: Metrik deret waktu per piksel....535
Kelompok D: Mendeteksi hutan yang hilang....540
Konsep: dua foto lawan seluruh film....540
Skrip 26.8: Deforestasi dari dua waktu....540
Skrip 26.9: Titik patah, kapan hutan mulai hilang....546
Kelompok E: Jejak api....551
Konsep: arang yang menyerap hijau....551
Skrip 26.10: Bekas kebakaran dengan dNBR....552
Skrip 26.11: Mencocokkan titik panas dan memperkirakan tanggal terbakar....557
Kelompok F: Mesin yang belajar dari contoh....562
Konsep: panitia juri dalam tiga kalimat....562
Skrip 26.12: Menyusun dua belas fitur....563
Kotak perkakas kedua: hutan_mini.py....566
Skrip 26.13: Melatih hutan acak dan memetakan seluruh area....571
Kelompok G: Seberapa boleh peta ini dipercaya?....578
Konsep: ujian dengan soal baru....578
Skrip 26.14: Validasi dua peta dengan sampel berstrata....578

BAB 27: Dari Skrip ke Produk: Dasbor, Peta Web, Model 3D, Alat Processing, dan Plugin....588

Studi kasus: "Hasilnya harus bisa dipakai orang lain"....588
Konsep: skrip sebagai pabrik kecil....588
Menyiapkan dulu: folder kerja dengan struktur penulis....589

Skrip 27.1: Dasbor satu halaman....590
Skrip 27.2: Peta web dari data QGIS....596
Skrip 27.3: Model 3D dari DSM....600
Skrip 27.4: Alat Processing buatan sendiri, gaya kelas....605
Skrip 27.5: Alat Processing gaya dekorator....611
Dua gaya, satu pekerjaan: kapan memakai yang mana....614
Skrip 27.6: Plugin sendiri di QGIS....615
Skrip 27.7: Memeriksa dan mengemas plugin....620

BAB 28: Pola dan Improvisasi: Kerangka Siap Salin dan Cara Mengubahnya....626
Studi kasus: "Datanya berubah. Skripnya bagaimana?"....626
Konsep: improvisasi dalam tiga kalimat....626
Menyiapkan dulu: data dan cara menjalankan....627
Pola 28.0: Kepala skrip untuk terminal....627
Pola 28.1: Muat, ubah, simpan....628
Pola 28.2: Loop fitur....630
Pola 28.3: Memanggil alat Processing....632
Pola 28.4: Batch, banyak berkas dengan satu putaran....634
Pola 28.5: Tahan galat, satu berkas rusak tidak menghentikan semua....636
Pola 28.6: Ekspresi QGIS dari Python....638
Pola 28.7: Baca dan tulis berkas CSV dan JSON....640
Pola 28.8: Membuat laporan dari beberapa sumber....642
Tabel: bila data saya berubah, bagian mana yang saya ganti?....644
Jebakan lintas bab....645
Cara mencari fungsi sendiri....648
Cara membaca dokumentasi API QGIS....650
Latihan akhir: rekap petak dari tiga sumber....651

BAB 29: Kenalan dengan ArcPy: Dua Generasi, Satu Sistem Windows....654
Studi kasus: "Skrip itu jalan di kantor pusat, mati di kantor cabang"....654
Konsep: ArcPy dalam tiga kalimat....655
Dua generasi: Pro lawan ArcMap....656
Hanya Windows: apa artinya bagi Anda....656
Di mana mengetik dan menjalankan skrip....657
Percobaan pertama: cek versi....658
Beda penulisan yang paling sering menjebak....658
Jalur Windows dan huruf r....659
Kebiasaan kerja yang aman....660

BAB 30: Empat Pekerjaan Klasik di ArcGIS: Skrip Masterclass dalam Bahasa ArcPy....662
Studi kasus: "Empat tugas rutin, satu kantor cabang"....662
Konsep: pekerjaan sama, bahasa beda....662
Skrip 30.1: Memotong citra dan menghitung NDVI....663
Versi ArcGIS Pro (Python 3)....664
Versi ArcMap 10.8 (Python 2.7)....665
Skrip 30.2: Akurasi titik survei lawan peta klasifikasi....666
Versi ArcGIS Pro (Python 3)....667
Versi ArcMap 10.8 (Python 2.7)....669
Skrip 30.3: NDVI untuk semua ubin foto drone....670
Versi ArcGIS Pro (Python 3)....671
Versi ArcMap 10.8 (Python 2.7)....672

Skrip 30.4: Satu PDF laporan untuk setiap KPH....674
Versi ArcGIS Pro (Python 3)....675
Versi ArcMap 10.8 (Python 2.7)....676
Teks dinamis untuk tata letak ArcMap....677

BAB 31: Dua Gaya, Satu Pekerjaan: ArcPy di ArcGIS Pro dan ArcMap....680
Studi kasus: "Kantor cabang memakai ArcGIS"....680
Konsep: ArcPy dalam tiga kalimat....680
Skrip 31.1: Membaca data, atribut, dan proyek....681
Versi ArcGIS Pro (Python 3)....681
Versi ArcMap 10.8 (Python 2.7)....683
Skrip 31.2: Rantai alat, dari sempadan sungai sampai NDVI per KPH....685
Versi ArcGIS Pro (Python 3)....686
Versi ArcMap 10.8 (Python 2.7)....688
Skrip 31.3: Batch NDVI semua ubin dan potongan CHM per KPH....690
Versi ArcGIS Pro (Python 3)....691
Versi ArcMap 10.8 (Python 2.7)....693
Skrip 31.4: Menangkap galat dengan ArcPy....695
Versi ArcGIS Pro (Python 3)....696
Versi ArcMap 10.8 (Python 2.7)....698
Skrip 31.5: Proyek mini terpadu, ringkasan per KPH ke CSV....700
Versi ArcGIS Pro (Python 3)....701
Versi ArcMap 10.8 (Python 2.7)....702

BAB 32: Dari QGIS ke ArcPy dan Kembali: Pola, Padanan, dan Cara Menerjemahkan....707
Studi kasus: "Skrip ini jalan di QGIS. Bisakah dibuat versi ArcGIS-nya?"....707
Konsep: menerjemahkan skrip dalam tiga kalimat....707
Tabel padanan: membaca data dan atribut....708
Tabel padanan: raster....709
Tabel padanan: alat vektor....709
Tabel padanan: peta, galat, dan berkas....710
Cara menerjemahkan skrip: enam langkah....711
Contoh terjemahan 1: membaca luas tiap KPH....712
Contoh terjemahan 2: penyangga sungai....712
Contoh terjemahan 3: dari ArcGIS Pro ke ArcMap....713
Jebakan lintas alat....713
Latihan menerjemahkan....714

Glosarium....717

Daftar Pustaka....727
Dokumentasi QGIS dan PyQGIS....727
Dokumentasi Python....727
Dokumentasi ArcGIS dan ArcPy....727

Tabel Semua Skrip....729

Pendahuluan: Selamat Datang di Dunia Skrip

Buku ini mengajak Anda menulis perintah untuk peta. Anda tidak perlu pernah memrogram sebelumnya. Anda juga tidak perlu jago matematika.

Setelah selesai, Anda bisa membaca skrip GIS milik orang lain. Anda bisa mengubahnya untuk data Anda sendiri. Anda juga bisa menulis skrip baru dari kerangka yang sudah Anda kenal. Itulah tujuan buku ini: bukan menghafal, melainkan **paham polanya**.

Cara membaca buku ini

Buku ini terdiri atas tiga bagian:

- **Bagian A, Python dari nol.** Cara memasang Python dan semua dasar bahasanya.
- **Bagian B, PyQGIS.** Skrip untuk QGIS, dibahas satu per satu.
- **Bagian C, ArcPy.** Skrip untuk ArcGIS Pro dan ArcMap 10.8. Bagian ini hanya untuk komputer Windows.

Tiap bab memakai pola yang sama:

- **Studi kasus**: masalah kecil dari "KPH Contoh", sebuah kantor pengelola hutan fiktif.
- **Konsep**: inti gagasan dalam tiga kalimat, dengan analogi.
- **Langkah** atau **Skrip**: praktik, dibedah baris demi baris.
- **Cek paham**: tiga pertanyaan singkat beserta jawabannya.
- **Kesalahan umum**: jebakan yang sering menimpa pemula.
- **Ringkasan dan latihan**.

Setiap skrip dibedah dengan urutan tetap: tujuan, bahan, skrip utuh, penjelasan baris demi baris, hasil, **bagian yang boleh diubah**, jebakan, dan latihan variasi. Urutan ini sengaja dibuat sama. Dengan begitu Anda tahu persis di mana mencari jawaban saat data Anda berbeda dari contoh.

Dari mana skrip di buku ini berasal

Skrip di buku ini bukan karangan baru. Semuanya diambil dari seri **buku tutorial GIS** karya penulis yang sama. Di awal setiap bab yang memakai skrip, sebuah kotak biru menyebut **seri asal** skrip itu. Dengan begitu Anda bisa membuka buku seri tersebut untuk membandingkan langkah lewat menu dengan skrip di sini.

Seri	Judul buku	Kelas	Dipakai di
B2	Data Vektor: Atribut, Digitasi, Seleksi	Basic	Bagian B
B3	Raster dan Citra Dasar	Basic	Bagian B
B4	Kartografi dan Layout Peta	Basic	Bagian B
I1	Analisis Vektor dan Geoprocessing	Intermediate	Bagian B
I2	Analisis Raster dan Pengindraan Jauh Dasar	Intermediate	Bagian B
I3	Survei Lapangan dan Basis Data	Intermediate	Bagian B

Seri	Judul buku	Kelas	Dipakai di
I4	Otomasi Dasar dengan Python	Intermediate	Bagian A, B, C
M1	Masterclass QGIS Modern	Master	Bagian B, C
M2	Basis Data Spasial dan Analisis Lanjut	Master	Bagian B
M3	Deret Waktu dan Pemodelan	Master	Bagian B
M4	Pengembangan dan Publikasi	Master	Bagian B

Anda tidak wajib memiliki buku-buku itu. Buku ini bisa dibaca sendiri.

Kenalan dengan KPH Contoh

Cerita di buku ini terjadi di **KPH Contoh**. Semua nama, tempat, dan peristiwa di dalamnya fiktif. Tokohnya hanya dikenal lewat peran: **Analis Muda** (itu Anda) dan **Kepala Seksi** (atasan yang sering memberi tugas mendadak).

Data latihannya dibuat buatan, jadi aman dipakai tanpa soal izin. Hasil dari data ini **bukan** kondisi lapangan sebenarnya.

Perangkat dan data

- **Python 3**: dasar semua skrip di buku ini.
- **QGIS**: buku ini menguji skrip pada QGIS 4.0.2. Tampilan versi lain bisa sedikit berbeda.
- **ArcGIS Pro** dan **ArcMap 10.8**: hanya untuk Bagian C, dan hanya berjalan di Windows. Langkah bertanda **[CEK]** perlu Anda cocokkan dengan perangkat Anda.
- **Paket data**: folder `data` dari penulis. Isinya data buatan per bab.

> **Catatan tentang ilustrasi:** Gambar di buku ini adalah **skema buatan penulis**, bukan tangkapan layar. Tampilan komputer Anda bisa sedikit berbeda. Ikuti nama menu dan isi layar yang tertulis.

Kontak penulis: badarmubarok@gmail.com.

Hak cipta dan penggunaan

Hak cipta © 2026 Badar Mubarok Yogaswara. All Rights Reserved.

Hak cipta dilindungi undang-undang. Dilarang mengutip atau memperbanyak sebagian atau seluruh isi buku ini tanpa izin tertulis dari penulis. Untuk penggunaan lain, misalnya bahan ajar berbayar atau penerbitan ulang, hubungi penulis di badarmubarok@gmail.com.

Pengecualian untuk skrip: kode di dalam kotak kode buku ini, tombol Salin dan Unduh di versi web, serta berkas di paket skrip boleh Anda pakai dan ubah secara bebas untuk belajar dan untuk pekerjaan Anda sendiri. Skrip, utuh atau yang sudah diubah, tidak boleh dijual ulang atau diterbitkan ulang sebagai produk, buku, kursus berbayar, atau paket skrip. Bila Anda membagikan skrip kepada orang lain, sebutkan penulisnya: Badar Mubarok Yogaswara. Penjelasan, gambar, dan teks di sekitar skrip tetap dilindungi seperti paragraf di atas.

Data latihan di buku ini buatan (sintetis), bukan data lapangan. Nama produk dan merek, misalnya QGIS dan ArcGIS, milik pemiliknya masing-masing.

BAGIAN A | PYTHON DARI NOL

BAB 1: Kenalan dengan Skrip: Menyuruh Komputer dengan Tulisan

Studi kasus: "Tolong buatkan ringkasan dua puluh petak"

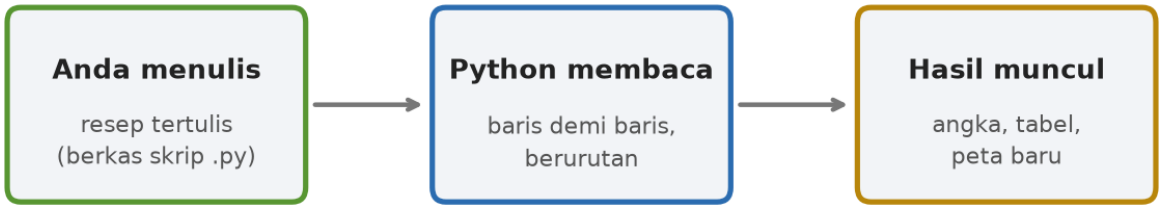
Kepala Seksi datang membawa daftar baru: dua puluh petak, masing-masing butuh ringkasan luas dan jenis tanaman. Anda bisa membuka tabel dan menghitungnya satu per satu. Itu memakan satu hari kerja. Besok daftarnya berubah lagi. Di titik inilah skrip menolong. Anda menulis urutan pekerjaannya sekali, lalu komputer mengulanginya dalam hitungan detik, berapa kali pun diminta.

Konsep: skrip dalam tiga kalimat

Sebuah **skrip** adalah berkas teks berisi urutan perintah untuk komputer. Bayangkan resep masakan: bahan, langkah, dan hasil, ditulis rapi supaya siapa pun bisa mengulanginya. **Python** adalah bahasa yang dipakai untuk menulis resep itu.

Ilustrasi 1.1: Skrip seperti resep masakan

Skema; bukan tangkapan layar



Skrip sama dengan resep masakan: urutan langkah yang bisa diulang kapan saja.

Empat istilah yang perlu Anda kenal sekarang:

- **Perintah**: satu langkah dalam resep, misalnya "hitung luas".
- **Skrip**: berkas yang menyimpan banyak perintah berurutan. Berkas skrip Python berakhiran `.py`.
- **Python**: bahasa tulisan yang dipahami komputer, dengan aturan yang sengaja dibuat mudah dibaca.
- **Menjalankan**: menyuruh komputer membaca skrip dari atas ke bawah dan mengerjakan tiap perintah.

Mengapa Python untuk GIS?

Alasannya sederhana: dua aplikasi GIS yang dibahas buku ini memakai Python. QGIS menyediakan Python yang dikenal sebagai **PyQGIS**. ArcGIS Pro dan ArcMap menyediakan **ArcPy**. Jadi satu bahasa bisa Anda pakai di kedua aplikasi, dengan sedikit perbedaan penulisan.

Ada tiga keuntungan yang akan Anda rasakan:

- **Cepat.** Pekerjaan yang berulang selesai dalam sekali jalan.
- **Bisa diulang.** Skrip menyimpan langkah Anda, sehingga hasilnya sama setiap kali dijalankan.
- **Bisa dibagikan.** Rekan kerja cukup menjalankan berkas yang sama.

Berikut tiga baris pertama yang bisa Anda baca sekarang, walaupun belum belajar apa pun:

PYTHON

```
luas_ha = [4.0, 4.0, 8.0]
print("Total luas:", sum(luas_ha), "ha")
```

Baris pertama menyimpan tiga luas petak. Baris kedua menjumlahkan lalu menampilkannya. Layar menampilkan:

HASIL

```
Total luas: 16.0 ha
```

Jangan khawatir bila ada bagian yang belum jelas. Daftar dibahas di Bab 5, dan `print` di Bab 4.

Apa yang akan Anda bisa setelah buku ini?

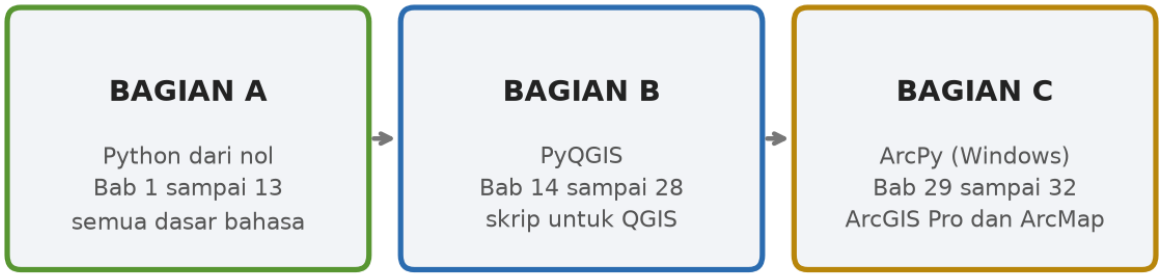
- Memasang Python dan menjalankan skrip di tiga tempat: konsol, editor, dan terminal.
- Menulis perintah Python dasar: menyimpan nilai, mengambil keputusan, dan mengulang pekerjaan.
- Membaca skrip PyQGIS dan ArcPy milik orang lain, lalu memahami tiap barisnya.
- **Mengubah skrip itu untuk data Anda sendiri.** Inilah tujuan utama buku ini.
- Mengenali pesan galat dan memperbaikinya dengan tenang.

Peta buku ini

Buku ini terdiri atas tiga bagian. Bagian A mengajarkan Python dari nol. Bagian B membahas skrip untuk QGIS, satu per satu. Bagian C membahas skrip untuk ArcGIS, dan hanya berlaku di komputer Windows.

Ilustrasi 1.2: Peta tiga bagian buku

Skema; bukan tangkapan layar



Tiap bagian bisa dibaca mandiri. Bagian A paling bermanfaat bila Anda baru mulai.

Bila Anda belum pernah menulis skrip, ikuti Bagian A dari awal sampai akhir. Bila Anda sudah akrab dengan Python, Anda boleh langsung melompat ke Bagian B atau C.

Tiga kebiasaan belajar

1. **Ketik sendiri, jangan salin-tempel.** Jari yang mengetik membuat Anda memperhatikan tiap tanda baca.
2. **Ubah satu hal, lalu lihat akibatnya.** Ganti satu angka atau satu nama, jalankan lagi, dan amati bedanya. Dari situlah pola mulai terlihat.
3. **Baca pesan galat.** Galat bukan tanda Anda gagal. Galat adalah petunjuk dari komputer tentang baris mana yang perlu diperiksa.

Cek paham

1. Apa itu skrip, dengan kata-kata Anda sendiri?
2. Sebutkan dua nama Python untuk dua aplikasi GIS di buku ini.
3. Mengapa skrip lebih cocok daripada klik menu untuk pekerjaan yang berulang?

Jawaban.

1. Berkas teks berisi urutan perintah yang dijalankan komputer dari atas ke bawah, seperti resep masakan.
2. PyQGIS untuk QGIS, dan ArcPy untuk ArcGIS Pro dan ArcMap.
3. Skrip menyimpan langkah-langkahnya, sehingga pekerjaan yang sama bisa diulang cepat dengan hasil yang sama.

Kesalahan umum

- **Mengira harus jago matematika atau komputer lebih dulu.** Perbaikan: mulailah dari contoh kecil. Buku ini memperkenalkan satu hal baru pada satu waktu.
- **Takut merusak komputer.** Perbaikan: skrip latihan di buku ini hanya membaca data contoh atau menulis ke folder hasil. Bagian yang berisiko diberi peringatan jelas.
- **Hanya menyalin tanpa memahami.** Perbaikan: sebelum menjalankan, bacalah penjelasan baris demi baris, lalu ubah satu bagian sendiri.

Ringkasan dan latihan

- **Skrip** adalah resep tertulis untuk komputer, dan **Python** adalah bahasa untuk menulisnya.
- QGIS memakai PyQGIS, ArcGIS memakai ArcPy. Keduanya berbasis Python.
- Cara belajar: ketik sendiri, ubah satu hal, baca pesan galat.

Latihan. Pilih satu pekerjaan di kantor yang Anda ulang tiap bulan. Tuliskan langkah-langkahnya dengan bahasa sehari-hari, satu langkah per baris. Itulah bentuk awal sebuah skrip.

Tabel perbandingan: lewat menu dan lewat skrip

Hal	Lewat menu	Lewat skrip
Pekerjaan sekali jalan	Cepat dan mudah dipelajari	Perlu waktu menulis
Pekerjaan berulang	Klik yang sama berulang kali	Sekali tulis, jalankan kapan saja
Hasil yang sama setiap kali	Bergantung pada ketelitian Anda	Selalu sama bila datanya sama
Dibagikan ke rekan	Perlu petunjuk panjang	Cukup kirim berkasnya

BAB 2: Menyiapkan Peralatan: Memasang Python dan QGIS

Studi kasus: "Komputer baru untuk Analis Muda"

Kepala Seksi menyerahkan sebuah komputer baru. Isinya kosong, tanpa Python dan tanpa QGIS. Sebelum bisa menulis skrip, Anda perlu menyiapkan dua peralatan itu. Bab ini menuntun Anda memasangnya di tiga sistem operasi: Windows, macOS, dan Ubuntu.

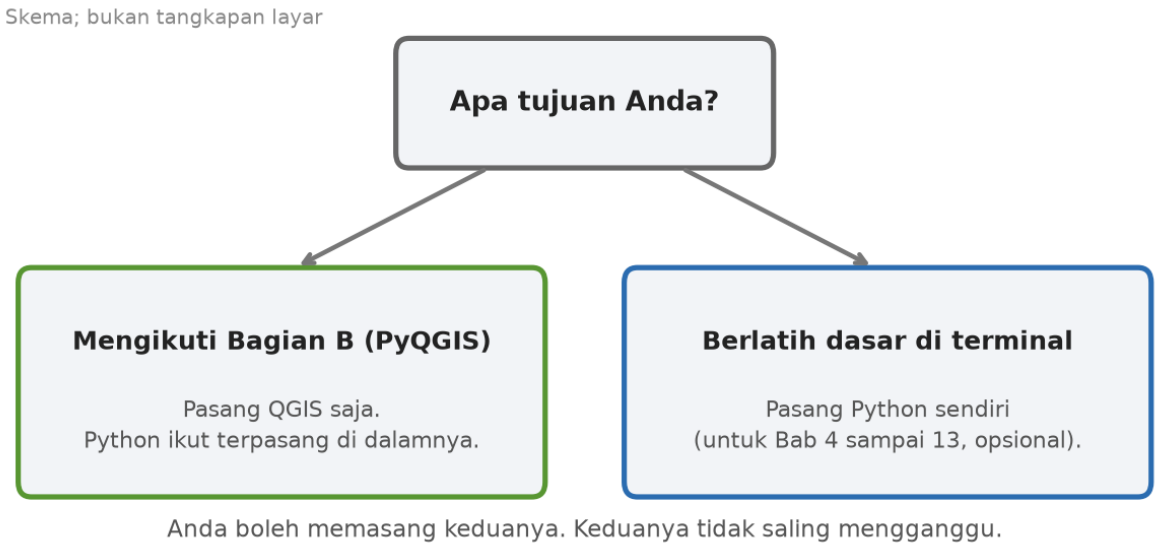
> **Langkah macOS dan Ubuntu bertanda [CEK]:** Penulis menguji langkah di Windows. Langkah macOS dan Ubuntu disusun dari dokumentasi resmi, tetapi belum dicoba langsung di komputer penulis. Cocokkan dengan layar Anda, dan laporkan bila ada yang berbeda.

Konsep: dua peralatan, dua jalan

Bayangkan dapur. **Python** adalah kompor dan peralatan masaknya. **QGIS** adalah dapur lengkap yang sudah membawa kompornya sendiri. Anda boleh memasak di dapur lengkap itu saja. Anda juga boleh memasang kompor sendiri di rumah untuk berlatih.

Hal ini penting: **QGIS membawa Python di dalamnya.** Di komputer penulis, QGIS 4.0.2 membawa Python 3.12.13. Karena itu, untuk mengikuti Bagian B, memasang QGIS sudah cukup. Memasang Python sendiri hanya perlu bila Anda ingin berlatih dasar di terminal (Bab 4 sampai 13).

Ilustrasi 2.1: Dua jalan memasang peralatan



Istilah baru di bab ini:

- **Terminal**: jendela untuk mengetik perintah teks. Di Windows namanya Command Prompt atau PowerShell, di macOS bernama Terminal, di Ubuntu juga Terminal.
- **Versi**: nomor edisi sebuah perangkat lunak, misalnya `3.12.13`.

Langkah: memasang di tiga sistem operasi

Langkah	Windows	macOS	Ubuntu
Python (opsional)	Unduh **Python Install Manager** dari python.org/downloads, lalu jalankan.	Unduh pemasang `.pkg` dari python.org/downloads/macos, klik dua kali, ikuti petunjuk. [CEK]	Biasanya sudah terpasang. [CEK]
Cek Python	Di terminal: `python --version`	Di Terminal: `python3 --version`	Di Terminal: `python3 --version`
QGIS	Unduh pemasang MSI dari qgis.org, atau pasang lewat OSGeo4W.	Unduh berkas `.dmg` dari qgis.org. [CEK]	Tambahkan repositori QGIS sesuai petunjuk di qgis.org, lalu jalankan perintah pemasangan di bawah. [CEK]

Berikut perintah pemasangan QGIS di Ubuntu, setelah repositori QGIS ditambahkan sesuai petunjuk di situs resmi:

TERMINAL
```
sudo apt install qgis qgis-plugin-grass
```

Beberapa catatan dari halaman resmi, yang dibaca penulis pada 7 Oktober 2026:

- Python Install Manager adalah pemasang Python yang kini dianjurkan untuk Windows. Pemasang lama (full installer) dinyatakan usang mulai Python 3.14.
- Pemasang Python untuk macOS bersifat *universal2*, jadi berjalan di Mac Intel maupun Apple Silicon. Setelah memasang, jalankan `Install Certificates.command` di folder Python di dalam Applications.
- Halaman unduhan QGIS menawarkan versi terbaru **4.2.3** dan versi *Long Term Release* **3.44.15**. Angka versi bisa berubah sewaktu-waktu.

> **Versi apa yang dipakai buku ini?:** Seluruh skrip PyQGIS di buku ini diuji pada **QGIS 4.0.2**. Bila Anda memasang versi lain, sebagian tampilan dan penulisan bisa sedikit berbeda. Skrip pada versi 3.x mungkin perlu penyesuaian kecil. [CEK]

Memeriksa pemasangan

Setelah memasang Python, ketik perintah ini di terminal untuk memastikannya terpasang. Di Windows, ketik `python --version`. Di macOS dan Ubuntu, ketik `python3 --version`.

Hasilnya satu baris yang menyebut versi, misalnya `Python 3.12.10`. Angka di komputer Anda boleh berbeda. Selama berawalan `Python 3`, Anda siap melanjutkan.

> **Jangan pakai Python 2:** Buku ini memakai **Python 3**. Bila terminal menjawab `Python 2.7`, itu versi lama. Gunakan `python3`, atau pasang Python 3. Pengecualian hanya untuk ArcMap 10.8 di Bagian C, yang memakai Python 2.7 bawaannya sendiri.

Cek paham

1. Mengapa memasang QGIS sudah cukup untuk mengikuti Bagian B?
2. Perintah apa yang Anda ketik di macOS atau Ubuntu untuk mengecek versi Python?
3. Versi QGIS apa yang dipakai untuk menguji skrip di buku ini?

Jawaban.

1. Karena QGIS membawa Python di dalamnya.
2. `python3 --version`.
3. QGIS 4.0.2.

Kesalahan umum

- **Terminal menjawab "python tidak dikenal" di Windows.** Perbaikan: tutup lalu buka lagi terminalnya setelah memasang. Bila masih gagal, ulangi pemasangan dan pastikan opsi penambahan ke PATH (daftar lokasi program) dicentang bila ditawarkan. [CEK]
- Memakai `python` di macOS atau Ubuntu padahal yang ada hanya `python3`. Perbaikan: ketik `python3`.
- **Memasang banyak versi tanpa tahu mana yang dipakai.** Perbaikan: cek dengan `--version` sebelum mulai berlatih.

Ringkasan dan latihan

- QGIS membawa Python sendiri. Untuk Bagian B, pasang QGIS saja.
- Untuk berlatih dasar di terminal, pasang Python 3 secara terpisah.
- Cek pemasangan dengan `python --version` (Windows) atau `python3 --version` (macOS dan Ubuntu).

Latihan. Buka terminal di komputer Anda, lalu cek versi Python. Catat nomor versinya di selembar kertas. Anda akan membutuhkannya nanti saat membandingkan dengan penjelasan di buku.

Tabel perbandingan: perintah di tiga sistem operasi

Pekerjaan	Windows	macOS	Ubuntu
Mengecek versi Python	`python --version`	`python3 --version`	`python3 --version`
Menjalankan berkas skrip	`python halo.py`	`python3 halo.py`	`python3 halo.py`
Membuka terminal	Command Prompt atau PowerShell	Terminal	Terminal

BAB 3: Tempat Menulis dan Menjalankan: Konsol, Editor, dan Terminal

Studi kasus: "Di mana saya mengetiknya?"

Python sudah terpasang. Analis Muda membuka QGIS dan bingung: tulisan Python itu diketik di mana? Ternyata ada tiga tempat, dan tiap tempat punya kegunaannya. Bab ini membantu Anda memilih tempat yang tepat dan menjalankan skrip pertama Anda di ketiganya.

Konsep: tiga tempat dalam tiga kalimat

Konsol Python adalah kotak untuk mencoba satu perintah dan langsung melihat hasilnya, seperti kertas coret-coret. **Editor skrip** adalah lembar kerja untuk menulis banyak perintah lalu menyimpannya sebagai berkas, seperti buku resep. **Terminal** adalah tempat menjalankan berkas skrip dari luar QGIS, seperti dapur cadangan.

Ilustrasi 3.1: Tiga tempat menulis dan menjalankan skrip

Skema; bukan tangkapan layar



Istilah baru di bab ini:

- **Konsol**: kotak ketik yang menjalankan perintah satu per satu.
- **Editor**: tempat menulis dan menyimpan berkas skrip.
- `iface`: kotak bernama khusus di dalam QGIS, tempat Anda mengendalikan jendela QGIS dari skrip. Kotak ini hanya ada di dalam QGIS.
- **Skrip mandiri**: skrip yang dijalankan dari terminal, di luar QGIS.

Tempat 1: Konsol Python QGIS

1. Buka QGIS. Pilih menu **Plugins ► Python Console**. Pintasannya **Ctrl+Alt+P**. [terbukti: manual resmi QGIS]
2. Panel konsol terbuka. Klik di kolom ketik paling bawah.

3. Ketik `print("Halo, KPH Contoh!")`, lalu tekan **Enter**.
4. Kolom hasil menampilkan: `Halo, KPH Contoh!` Letak panel bisa berbeda di versi Anda. [CEK]

Hasilnya sama dengan yang Anda lihat bila Anda menjalankan perintah itu di terminal. Konsol cocok untuk mencoba-coba: ketik, lihat hasil, ketik lagi.

Beberapa tombol di bilah alat konsol, menurut manual resmi: **Clear Console** untuk membersihkan layar, **Run Command** untuk menjalankan isi kolom ketik (sama dengan Enter), dan **Show Editor** untuk membuka editor.

Tempat 2: Editor skrip di dalam konsol

1. Di bilah alat konsol, klik **Show Editor**. Sebuah area tulis terbuka.
2. Ketik di editor: `print("Halo dari editor").`
3. Simpan dengan **Ctrl+S**. Pilih folder kerja Anda, lalu beri nama `halo_editor.py`.
4. Jalankan seluruh skrip dengan tombol **Run script** atau pintasan **Ctrl+Shift+E**. [terbukti: manual resmi QGIS]
5. Hasilnya muncul di kolom hasil konsol: `Halo dari editor`.

Editor juga punya **Open script** untuk membuka skrip dari berkas, **Run selected** (Ctrl+E) untuk menjalankan baris yang disorot, dan **Check syntax** (Ctrl+4) untuk mencari kesalahan penulisan.

> **Siapkan satu folder khusus skrip:** Buat satu folder di tempat yang mudah Anda temukan, misalnya `Dokumen/skrip-saya`. Simpan semua skrip latihan di sana, bukan di folder sementara. Folder sementara bisa terhapus sendiri.

Tempat 3: Terminal

Terminal menjalankan berkas skrip dari luar QGIS. Ada dua cara, tergantung pada skrip.

Skrip Python murni. Simpan baris `print("Halo, KPH Contoh!")` dalam berkas `halo.py`. Lalu jalankan dari terminal, di folder tempat berkas itu berada:

```
TERMINAL
python halo.py
```

Di macOS dan Ubuntu, ketik `python3 halo.py`. Hasilnya:

```
HASIL
Halo, KPH Contoh!
```

Skrip yang memakai QGIS. Python biasa tidak mengenal QGIS. Di Windows, QGIS menyediakan penjalan khusus bernama `python-qgis.bat` di folder `bin` instalasinya. Di komputer penulis, lokasinya `C:\Program Files\QGIS 4.0.2\bin\python-qgis.bat` [terbukti]. Simpan skrip berikut sebagai `versi.py`:

```
PYTHON
import sys
from qgis.core import Qgis

print(sys.version.split()[0])
print("QGIS", Qgis.version())
```

Lalu jalankan dengan penjalan itu:

```
TERMINAL
"C:\Program Files\QGIS 4.0.2\bin\python-qgis.bat" versi.py
```

Hasil di komputer penulis:

```
HASIL
3.12.13
QGIS 4.0.2-Norrköping
```

Dua baris itu menjawab dua pertanyaan: Python versi berapa yang dibawa QGIS, dan QGIS versi berapa yang terpasang. Di Ubuntu, modul QGIS untuk Python berasal dari paket `python3-qgis`, lalu skripnya dijalankan dengan `python3`. [CEK]

Skrip mandiri butuh persiapan

Di dalam QGIS, semua sudah siap. Di terminal, sebagian perkakas QGIS harus Anda nyalakan sendiri lebih dulu. Dokumentasi resmi QGIS menyebut urutannya: buat objek aplikasi, nyalakan dengan `initQgis()`, dan matikan dengan `exitQgis()`. Skrip kecil berikut menyalakan dan mematikan QGIS, lalu mencetak satu kalimat:

```
PYTHON
from qgis.core import QgsApplication

app = QgsApplication([], False)
app.initQgis()
print("QGIS siap dipakai")
app.exitQgis()
```

Hasilnya, bila dijalankan dengan `python-qgis.bat`:

HASIL
```
QGIS siap dipakai
```

Alat-alat Processing (misalnya Buffer) butuh persiapan tambahan. Skrip berikut menyiapkannya, lalu mencari alat Buffer:

PYTHON
```
import sys
from qgis.core import QgsApplication

app = QgsApplication([], False)
app.initQgis()
sys.path.append(r"C:\Program Files\QGIS 4.0.2\apps\qgis\python\plugins")
from processing.core.Processing import Processing
Processing.initialize()
from qgis.analysis import QgsNativeAlgorithms
QgsApplication.processingRegistry().addProvider(QgsNativeAlgorithms())
import processing

alat = QgsApplication.processingRegistry().algorithmById("native:buffer")
print("Alat ditemukan:", alat.displayName())
app.exitQgis()
```

Hasil:

HASIL
```
Alat ditemukan: Buffer
```

Baris `sys.path.append(...)` menunjuk lokasi modul Processing di komputer penulis. Lokasinya di komputer Anda bisa berbeda, terutama di macOS dan Ubuntu. [CEK]

> **Bagian B memakai Konsol dan Editor:** Hampir semua skrip di Bagian B dirancang untuk **dijalankan di dalam QGIS**, lewat Konsol dan Editor. Di sana `processing` sudah siap. Cara terminal dipakai untuk pengujian dan pekerjaan otomatis, dan dibahas lagi di Bab 21.

Cek paham

1. Tempat mana yang cocok untuk mencoba satu perintah dan langsung melihat hasilnya?
2. Mengapa skrip yang memakai QGIS tidak dijalankan dengan `python` biasa di terminal?
3. Apa pintasan untuk membuka Konsol Python di QGIS?

Jawaban.

1. Konsol Python.
2. Python biasa tidak mengenal modul QGIS. Gunakan penjalan khusus QGIS, misalnya `python-qgis.bat` di Windows.

3. **Ctrl+Alt+P**, atau lewat menu **Plugins ► Python Console**.

Kesalahan umum

- Menjalankan skrip QGIS dengan `python` biasa. Pesan galat: `ModuleNotFoundError: No module named 'qgis'`. Perbaikan: jalankan dengan penjalan QGIS, atau jalankan dari editor di dalam QGIS.
- **Lupa menyimpan skrip di editor sebelum menjalankan.** Perbaikan: simpan dengan Ctrl+S dulu, dan beri nama berakhiran `.py`.
- **Menyimpan skrip di folder sementara.** Perbaikan: pakai folder khusus skrip, seperti di kotak tip di atas.

Ringkasan dan latihan

- **Konsol** untuk mencoba satu baris, **editor** untuk berkas skrip, **terminal** untuk menjalankan dari luar QGIS.
- Di Windows, skrip yang memakai QGIS dijalankan dengan `python-qgis.bat`.
- Skrip mandiri harus menyalakan QGIS lebih dulu dengan `initQgis()`. Di dalam QGIS, itu sudah otomatis.

Latihan. Tulis skrip dua baris yang mencetak nama Anda dan nama kantor fiktif Anda. Simpan sebagai `perkenalan.py`, lalu jalankan di tiga tempat: konsol, editor, dan terminal. Catat tempat mana yang paling mudah menurut Anda.

Tabel perbandingan: tiga tempat menjalankan

Hal	Konsol	Editor skrip	Terminal
Isi	Satu baris tiap kali	Banyak baris, disimpan	Berkas yang sudah ada
Tempat	Dalam QGIS	Dalam QGIS	Luar QGIS
Cocok untuk	Mencoba-coba	Skrip yang dipakai ulang	Pekerjaan otomatis dan uji

BAB 4: Kotak Bernama: Variabel, Angka, dan Teks

> **Berdasarkan Seri I4:** Skrip bab ini berasal dari Buku I4, *Otomasi Dasar dengan Python* (Intermediate Class), berkas `b1_00_python_dasar.py`. Di buku itu, skrip ini dipakai sebagai pemanasan sebelum PyQGIS.

Studi kasus: "Luas KPH Alpha berapa?"

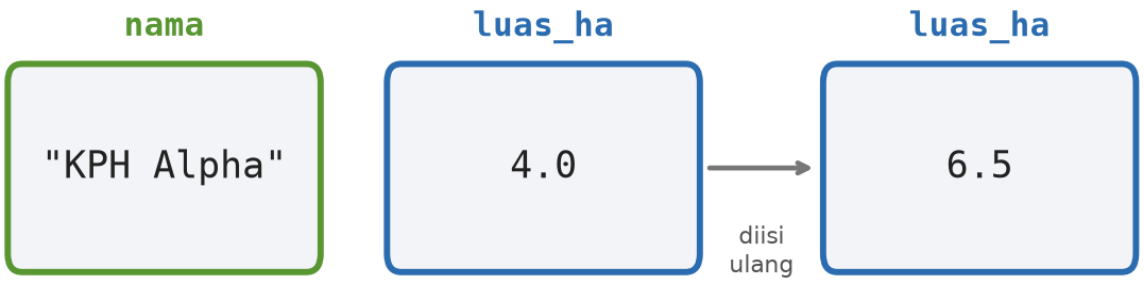
Kepala Seksi mengirim pesan singkat: "Luas KPH Alpha berapa hektare?" Anda bisa membuka peta dan membaca jawabannya. Tetapi besok ia bertanya lagi, kali ini untuk dua puluh petak. Dengan Python, Anda menulis jawabannya sekali. Komputer lalu mengulanginya sebanyak yang Anda mau. Langkah pertamanya sederhana: menyimpan nilai dalam sebuah kotak bernama.

Konsep: variabel dalam tiga kalimat

Sebuah **variabel** adalah kotak bernama yang menyimpan satu **nilai**. Bayangkan kotak arsip dengan label di depannya: label itu nama variabelnya, dan isi kotak itu nilainya. Saat Anda menyebut labelnya, Python mengeluarkan isi kotaknya.

Ilustrasi 4.1: Variabel seperti kotak arsip bernama

Skema; bukan tangkapan layar



Label di atas kotak = nama variabel. Tulisan di dalam kotak = nilai.

Isi kotak boleh diganti kapan saja. Kotaknya tetap sama, hanya isinya yang berubah.

Tiga jenis nilai yang paling sering dipakai

Python membedakan nilai menurut jenisnya. Tiga jenis ini cukup untuk memulai:

- **Teks** (di Python disebut *string*): rangkaian huruf, angka, dan tanda baca. Selalu diapit tanda kutip, misalnya `"KPH Alpha"`.
- **Bilangan bulat** (*integer*): angka tanpa desimal, misalnya `7` atau `2008`.
- **Bilangan desimal** (*float*): angka dengan desimal, misalnya `4.0` atau `6.5`. Python memakai **titik**, bukan koma.

Teks itu seperti tulisan di label. Angka itu seperti jumlah yang bisa dihitung. Anda bisa menjumlahkan dua angka. Anda juga bisa menyambung dua teks. Tetapi Anda tidak bisa mencampur keduanya begitu saja.

Dua perintah penolong: print dan komentar

- `print`: perintah untuk menampilkan sesuatu di layar. Isi yang ingin dilihat ditulis di dalam tanda kurung.
- **Komentar**: catatan untuk manusia. Python mengabaikan semua tulisan setelah tanda pagar `#`.

Langkah: mencoba di tempat mengetik Python

Pakai tempat mengetik yang Anda siapkan di Bab 3, yaitu Konsol Python QGIS atau terminal. Ketik satu baris, lalu tekan **Enter**.

1. Ketik `nama = "KPH Alpha"`. Layar tidak menampilkan apa pun. Itu normal: kotak bernama `nama` sudah dibuat.
2. Ketik `luas_ha = 4.0`. Kotak kedua, bernama `luas_ha`, sudah terisi.
3. Ketik `print(nama, "luasnya", luas_ha, "ha")`. Layar menampilkan `KPH Alpha luasnya 4.0 ha`.
4. Ketik `luas_ha = 6.5`, lalu ketik ulang baris `print` tadi. Layar menampilkan `KPH Alpha luasnya 6.5 ha`. Kotak yang sama kini berisi nilai baru.

Hasilnya: Anda sudah membuat dua variabel, menampilkannya, dan mengganti isi salah satunya. Itu kerangka hampir semua skrip di buku ini.

> **Tanda sama dengan bukan "sama dengan":** Di Python, tanda `=` berarti "isi kotak ini dengan". Baca `luas_ha = 4.0` sebagai: "isi kotak luas_ha dengan 4.0".

Skrip yang dibedah: lima hal Python untuk memulai

Berkas `b1_00_python_dasar.py` memuat lima hal dasar dalam 33 baris. Bab ini membedah bagian pertamanya. Empat bagian sisanya muncul di Bab 5, 6, dan 7.

Tujuan. Menyimpan nama dan luas petak, lalu menampilkannya dalam satu kalimat.

Bahan. Tidak ada. Skrip ini tidak memakai QGIS dan tidak membaca berkas data.

Skrip (bagian pertama):

```python
# 1. Variabel: kotak bernama yang menyimpan nilai
nama = "KPH Alpha"
luas_ha = 4.0
print(nama, "luasnya", luas_ha, "ha")
```

Penjelasan baris demi baris. Nomor baris dihitung dari potongan di atas.

Baris	Tulisan	Artinya
1	`# 1. Variabel: ...`	Komentar. Python melewatinya. Isinya catatan untuk Anda.
2	`nama = "KPH Alpha"`	Buat kotak bernama `nama`, isi dengan teks `KPH Alpha`. Tanda kutip menandakan ini teks.
3	`luas_ha = 4.0`	Buat kotak bernama `luas_ha`, isi dengan angka desimal `4.0`. Akhiran "ha" hanya bagian dari nama. Python tidak tahu artinya.
4	`print(nama, "luasnya", luas_ha, "ha")`	Tampilkan empat hal berurutan, dipisah spasi: isi `nama`, teks `luasnya`, isi `luas_ha`, dan teks `ha`.

Hasil. Satu baris di layar:

```
HASIL
KPH Alpha luasnya 4.0 ha
```

Bagian yang boleh diubah.

- Isi `nama`: ganti menjadi `"KPH Beta"`. Layar menampilkan nama baru.
- Isi `luas_ha`: ganti menjadi `8.0` atau `12.5`. Layar menampilkan luas baru.
- **Nama kotak**: `nama` boleh diganti `nama_petak`. Ganti di semua baris yang memakainya, termasuk baris `print`.
- Kata di dalam kutip pada `print`: ganti `"luasnya"` menjadi `"memiliki luas"`.

Aturan nama kotak: pakai huruf kecil, boleh ada angka (tetapi bukan di awal), dan pakai garis bawah `_` sebagai pengganti spasi. Python membedakan huruf besar dan kecil.

Jebakan.

- **Huruf besar dan kecil.** Menulis `print(Nama)` menghasilkan `NameError: name 'Nama' is not defined`. Artinya Python tidak menemukan kotak bernama `Nama`.
- **Lupa tanda kutip.** Menulis `teks = KPH Alpha` menghasilkan `SyntaxError: invalid syntax`. Python mengira `KPH` dan `Alpha` adalah nama kotak.
- **Koma sebagai desimal.** Menulis `luas_ha = 4,0` tidak menimbulkan galat. Tetapi isinya menjadi pasangan `(4, 0)`, bukan angka `4.0`. Python membaca koma sebagai pemisah.

Latihan variasi. Ubah skrip untuk KPH Gamma, luas 8.0 hektare, dan tambahkan variabel ketiga bernama `jenis` berisi `"Jati"`. Tampilkan ketiganya dalam satu kalimat.

Contoh jawaban:

```
PYTHON
nama = "KPH Gamma"
luas_ha = 8.0
jenis = "Jati"
print(nama, "luasnya", luas_ha, "ha, jenis", jenis)
```

Layar menampilkan `KPH Gamma luasnya 8.0 ha, jenis Jati.`

> **Pola yang bisa dipakai ulang:** Hampir semua skrip dimulai dari dua baris ini. Ganti huruf besar dengan nama dan nilai Anda sendiri: `NAMA_KOTAK = NILAI print(NAMA_KOTAK)`

Cek paham

1. Apa isi kotak `luas_ha` setelah baris `luas_ha = 6.5` dijalankan?
2. Mengapa `"KPH Alpha"` memakai tanda kutip, sedangkan `nama` tidak?
3. Apa yang terjadi bila Anda menulis `print(Nama)`, padahal kotaknya bernama `nama`?

Jawaban.

1. Isinya `6.5`. Nilai lama `4.0` tergantikan dan tidak tersimpan lagi.
2. Tanda kutip menandakan teks. Tanpa kutip, Python mengira itu nama sebuah kotak.
3. Python menampilkan `NameError`, karena huruf besar dan kecil dianggap berbeda. Kotak `Nama` tidak ada.

Kesalahan umum

- **Menulis desimal dengan koma.** Perbaikan: pakai titik, misalnya `4.0`.
- **Lupa kutip pada teks.** Perbaikan: apit teks dengan tanda kutip ganda di awal dan akhir.
- **Memberi nama kotak dengan spasi**, misalnya `luas ha = 4.0`. Python menolaknya. Perbaikan: pakai garis bawah, yaitu `luas_ha`.

Ringkasan dan latihan

- **Variabel** adalah kotak bernama yang berisi satu nilai. Isinya boleh diganti.
- Tiga jenis nilai awal: **teks**, **bilangan bulat**, dan **bilangan desimal**.
- Perintah `print` menampilkan nilai di layar. Tanda `#` membuat komentar.
- Huruf besar dan kecil dibedakan. Desimal memakai titik.

Latihan. Buat tiga variabel untuk KPH Beta: nama, luas dalam hektare, dan jenis tanaman. Tampilkan ketiganya dalam satu kalimat dengan `print`. Lalu ubah luasnya dan jalankan lagi.

Tabel perbandingan: tiga jenis nilai

Jenis	Contoh	Nama di Python	Bisa dihitung?
Teks	`"KPH Alpha"`	`str`	Tidak dihitung, hanya bisa disambung dengan `+`
Bilangan bulat	`2008`	`int`	Ya
Bilangan desimal	`4.0`	`float`	Ya

BAB 5: Rak dan Daftar Isi: Daftar, Tuple, Set, dan Kamus

> **Berdasarkan Seri I4:** Bagian skrip di bab ini berasal dari Buku I4, *Otomasi Dasar dengan Python* (Intermediate Class), berkas `b1_00_python_dasar.py`, bagian 2 dan 3. Contoh tuple dan koordinat memakai angka dari skrip `b2_03_digitasi.py` di Buku B2, *Data Vektor: Atribut, Digitasi, Seleksi* (Basic Class).

Studi kasus: "Dua puluh petak, dua puluh kotak?"

Kepala Seksi memberi Anda daftar dua puluh nama petak. Anda mulai mengetik: `petak1 = ...`, `petak2 = ...`, sampai `petak20`. Baru tiga baris, Anda sudah lelah. Pasti ada cara yang lebih rapi. Ada: satu kotak yang memuat banyak nilai sekaligus. Bab ini memperkenalkan empat jenis kotak seperti itu.

Konsep: daftar dan kamus dalam tiga kalimat

Sebuah **daftar** adalah rak berurutan yang menyimpan banyak nilai, dan tiap nilai punya nomor urut yang dimulai dari **0**. Sebuah **kamus** adalah daftar isi buku: Anda mencari sebuah **kunci** (judul), lalu mendapat **nilai** (nomor halaman). Daftar dicari lewat nomor, kamus dicari lewat kunci.

Ilustrasi 5.1: Daftar dan kamus



Istilah baru di bagian ini:

- **Daftar** (*list*): kumpulan nilai berurutan, ditulis di dalam kurung siku `[ ]`.
- **Indeks**: nomor urut sebuah nilai di dalam daftar. Indeks pertama adalah 0.

Daftar: rak bernomor

Daftar ditulis dengan kurung siku. Nilainya dipisah koma.

```python
petak = ["KPH Alpha", "KPH Beta", "KPH Gamma"]
print(petak[0])
print(len(petak))
petak.append("KPH Delta")
print(petak)
print(petak[-1])
print(petak[1:3])
```

Baris demi baris:

- `petak[0]` mengambil nilai bernomor 0, yaitu yang **pertama**.
- `len(petak)` menghitung banyak isi daftar.
- `petak.append(...)` menambahkan satu nilai di ujung daftar.
- `petak[-1]` mengambil nilai **terakhir**. Angka minus berarti menghitung dari belakang.
- `petak[1:3]` mengambil irisan: dari nomor 1 sampai **sebelum** nomor 3.

```
KPH Alpha
3
['KPH Alpha', 'KPH Beta', 'KPH Gamma', 'KPH Delta']
KPH Delta
['KPH Beta', 'KPH Gamma']
```

> **Mengapa mulai dari 0?:** Anggap nomor sebagai jarak dari ujung rak. Nilai pertama berjarak nol langkah dari ujung. Jadi nilai ketiga bernomor 2. Ini kebiasaan hampir semua bahasa pemrograman, dan tidak perlu dihafal mati: cukup ingat "hitung dari nol".

Tuple dan set: dua sepupu daftar

Dua jenis ini jarang berdiri sendiri. Tetapi skrip GIS sering memakainya, jadi kenali sekarang.

- **Tuple**: daftar yang **tidak bisa diubah** setelah dibuat. Ditulis dengan kurung biasa `( )`. Cocok untuk pasangan yang tetap, misalnya koordinat.
- **Set** (himpunan): kumpulan nilai **tanpa kembar** dan tanpa urutan.

```python
titik = (312000.0, 9996000.0)
x, y = titik
print(x, y)

jenis = ["Jati", "Mahoni", "Jati", "Sengon", "Mahoni"]
unik = sorted(set(jenis))
print(unik)
print(len(jenis), len(set(jenis)))
```

Penjelasannya:

- Baris `x, y = titik` membongkar tuple: nilai pertama masuk ke `x`, nilai kedua ke `y`. Teknik ini bernama *unpacking*.
- `set(jenis)` membuang nilai yang kembar. `sorted(...)` mengurutkan hasilnya menurut abjad.
- Dua angka terakhir menunjukkan: lima nama jenis, tetapi hanya tiga yang berbeda.

HASIL

```
312000.0 9996000.0
['Jati', 'Mahoni', 'Sengon']
5 3
```

Pola `sorted(set(...))` Anda temui di skrip Buku B2 untuk mendaftar jenis tanaman tanpa pengulangan.

Kamus: daftar isi buku

Kamus ditulis dengan kurung kurawal `{ }`. Tiap pasangan berbentuk `kunci: nilai`.

PYTHON

```
luas = {"KPH Alpha": 4.0, "KPH Beta": 4.0, "KPH Gamma": 8.0}
print(luas["KPH Gamma"])
luas["KPH Delta"] = 6.5
print(len(luas))
print(luas.get("KPH Omega", 0))
for nama, ha in luas.items():
    print(nama, ha)
```

Penjelasannya:

- `luas["KPH Gamma"]` mencari nilai lewat kunci `"KPH Gamma"`.
- `luas["KPH Delta"] = 6.5` menambah pasangan baru. Bila kuncinya sudah ada, nilainya diganti.
- `luas.get("KPH Omega", 0)` mencari kunci yang **belum tentu ada**. Bila tidak ada, hasilnya `0` (nilai cadangan), bukan galat. Pola ini dipakai di skrip penjumlahan luas per KPH di Buku B2.
- Baris `for ... in luas.items()` mengulang tiap pasangan. Perulangan dibahas di Bab 6.

HASIL

```
8.0
4
0
KPH Alpha 4.0
KPH Beta 4.0
KPH Gamma 8.0
KPH Delta 6.5
```

Skrip yang dibedah: daftar dan kamus di b1_00

Berkas `b1_00_python_dasar.py` memuat lima hal dasar. Bab 4 membedah bagian pertamanya. Di sini kita bedah bagian kedua dan ketiga.

Tujuan. Menyimpan nama tiga petak dalam daftar, menyimpan luasnya dalam kamus, lalu mengambil masing-masing satu nilai.

Bahan. Tidak ada. Skrip ini tidak memakai QGIS dan tidak membaca berkas.

Skrip (bagian kedua dan ketiga):

```
PYTHON
# 2. Daftar: kumpulan nilai berurutan. Nomor urut dimulai dari 0, bukan 1.
petak = ["KPH Alpha", "KPH Beta", "KPH Gamma"]
print("Petak pertama:", petak[0])
print("Jumlah petak:", len(petak))

# 3. Kamus: pasangan kunci dan nilai, seperti daftar isi dengan nomor halaman
luas = {"KPH Alpha": 4.0, "KPH Beta": 4.0, "KPH Gamma": 8.0}
print("Luas KPH Gamma:", luas["KPH Gamma"])
```

Penjelasan baris demi baris. Nomor baris dihitung dari potongan di atas.

Baris	Tulisan	Artinya
1	`# 2. Daftar: ...`	Komentar. Python melewatinya.
2	`petak = [...]`	Buat kotak `petak` berisi daftar tiga teks. Tanda kurung siku menandakan daftar.
3	`print("Petak pertama:", petak[0])`	Tampilkan teks, lalu isi nomor 0 dari daftar, yaitu `KPH Alpha`.
4	`print("Jumlah petak:", len(petak))`	Tampilkan banyak isi daftar.
6	`# 3. Kamus: ...`	Komentar.
7	`luas = {...}`	Buat kotak `luas` berisi kamus. Tiap pasangan: nama petak sebagai kunci, luasnya sebagai nilai.
8	`print("Luas KPH Gamma:", luas["KPH Gamma"])`	Cari kunci `"KPH Gamma"`, lalu tampilkan nilainya.

Hasil.

```
HASIL
Petak pertama: KPH Alpha
Jumlah petak: 3
Luas KPH Gamma: 8.0
```

Bagian yang boleh diubah.

- **Isi daftar (baris 2)**: tambah nama petak, misalnya `"KPH Delta"`. Angka `Jumlah petak` ikut berubah.

- Nomor di `petak[0]` (baris 3): ganti menjadi `1` untuk petak kedua, atau `-1` untuk petak terakhir.
- **Isi kamus (baris 7)**: ubah luas, atau tambah pasangan baru.
- Kunci di `luas["KPH Gamma"]` (baris 8): ganti dengan kunci lain yang ada di kamus.

Jebakan.

- **Nomor melebihi isi daftar.** Daftar `petak` hanya punya nomor 0, 1, dan 2. Menulis `petak[3]` menghasilkan `IndexError: list index out of range`. Artinya: nomor itu di luar rak.
- **Kunci yang tidak ada.** Menulis `luas["KPH Omega"]` menghasilkan `KeyError: 'KPH Omega'`. Pakai `luas.get("KPH Omega", 0)` bila kunci belum tentu ada.
- **Mengubah tuple.** Menulis `titik[0] = 0` menghasilkan `TypeError: 'tuple' object does not support item assignment`. Tuple memang tidak bisa diubah. Buat tuple baru bila perlu.
- **Huruf besar dan kecil pada kunci.** `"kph gamma"` dan `"KPH Gamma"` dianggap dua kunci berbeda.

Latihan variasi. Tambahkan satu petak baru, `"KPH Delta"` seluas 6.5 hektare, ke daftar dan kamus. Lalu tampilkan jumlah petak dan luas KPH Delta.

Contoh jawaban:

PYTHON

```
petak = ["KPH Alpha", "KPH Beta", "KPH Gamma"]
luas = {"KPH Alpha": 4.0, "KPH Beta": 4.0, "KPH Gamma": 8.0}
petak.append("KPH Delta")
luas["KPH Delta"] = 6.5
print("Jumlah petak:", len(petak))
print("Luas KPH Delta:", luas["KPH Delta"])
```

HASIL

```
Jumlah petak: 4
Luas KPH Delta: 6.5
```

> **Pola yang bisa dipakai ulang: Mencari lewat nomor atau lewat nama?** Bila urutan penting, pakai daftar dan ambil dengan `daftar[nomor]`. Bila Anda ingin mencari lewat nama, pakai kamus dan ambil dengan `kamus["kunci"]`. Untuk kunci yang belum tentu ada, pakai `kamus.get("kunci", nilai_cadangan)`.

Cek paham

1. Apa nomor nilai pertama dalam sebuah daftar?
2. Apa beda daftar dan kamus dalam cara mencari nilai?
3. Mengapa `luas.get("KPH Omega", 0)` lebih aman daripada `luas["KPH Omega"]` untuk kunci yang belum tentu ada?

Jawaban.

1. Nomor 0.

2. Daftar dicari lewat nomor urut, kamus dicari lewat kunci.
3. `get` mengembalikan nilai cadangan bila kunci tidak ada, sedangkan tanda kurung siku langsung menimbulkan `KeyError`.

Kesalahan umum

- **Menghitung dari 1.** Nilai ketiga bernomor 2, bukan 3. Perbaikan: hitung dari nol.
- **Salah kurung.** Kurung siku untuk daftar, kurung biasa untuk tuple, kurung kurawal untuk kamus dan set. Perbaikan: lihat bentuk kurung di awal tiap contoh.
- **Mengira set berurutan.** Set tidak menjamin urutan. Perbaikan: bungkus dengan `sorted(...)` bila butuh urutan.

Ringkasan dan latihan

- **Daftar** `[ ]`: berurutan, bernomor mulai 0, boleh diubah.
- **Tuple** `( )`: berurutan, tidak boleh diubah.
- **Set**: tanpa kembar, tanpa urutan.
- **Kamus** `{ }`: pasangan kunci dan nilai, dicari lewat kunci.

Latihan. Buat daftar berisi lima jenis tanaman (boleh ada yang kembar). Tampilkan jumlah seluruhnya, lalu jumlah jenis yang berbeda.

Tabel perbandingan: empat jenis kumpulan nilai

Jenis	Tulisan	Cara mengambil	Cocok untuk
Daftar	`["a", "b"]`	`daftar[0]`	Urutan yang boleh berubah
Tuple	`("a", "b")`	`tuple[0]`	Pasangan tetap, misalnya koordinat
Set	`{"a", "b"}`	`set` tidak bernomor	Menyaring nilai kembar
Kamus	`{"a": 1}`	`kamus["a"]`	Mencari lewat nama

BAB 6: Persimpangan dan Putaran: Keputusan dan Perulangan

> **Berdasarkan Seri I4 dan B2:** Bagian skrip di bab ini berasal dari Buku I4, *Otomasi Dasar dengan Python* (Intermediate Class), berkas `b1_00_python_dasar.py` bagian 4. Contoh `enumerate`, `break`, dan comprehension berasal dari Buku B2, *Data Vektor: Atribut, Digitasi, Seleksi* (Basic Class), berkas `b2_01_atribut.py`.

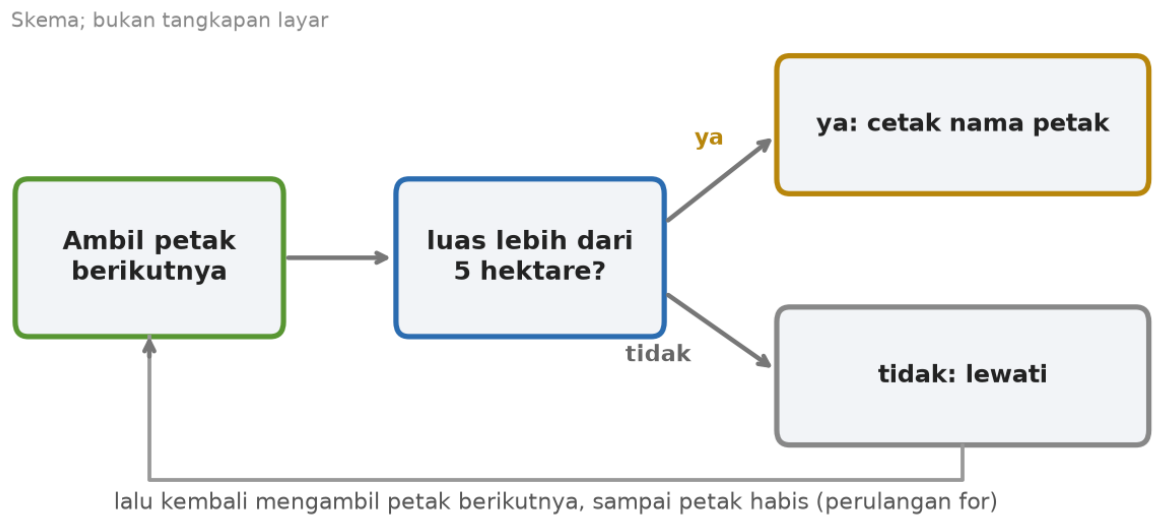
Studi kasus: "Hanya petak yang besar"

Kepala Seksi bertanya: "Dari semua petak, mana yang lebih dari lima hektare?" Anda punya daftar luas. Memeriksanya satu per satu dengan mata memang bisa, tetapi lambat dan rawan salah. Python punya dua kemampuan yang menjawab ini: **mengambil keputusan** dan **mengulang pekerjaan**.

Konsep: keputusan dan perulangan dalam tiga kalimat

Keputusan (`if`) adalah persimpangan jalan: bila syaratnya terpenuhi, Python belok kanan, bila tidak, ia lanjut lurus. **Perulangan** (`for`) adalah petugas yang mengambil satu berkas dari tumpukan, mengerjakannya, lalu mengambil berkas berikutnya sampai tumpukan habis. Keduanya sering dipakai bersama: ambil satu petak, periksa syaratnya, lalu lanjut ke petak berikutnya.

Ilustrasi 6.1: Alur keputusan di dalam perulangan



Istilah baru di bagian ini:

- **Syarat**: pertanyaan yang jawabannya hanya "benar" atau "salah", misalnya `luas > 5`.
- **Blok**: kelompok baris yang **menjorok ke dalam**. Baris yang menjorok adalah isi dari `if` atau `for` di atasnya.
- **Indentasi**: jarak menjorok itu. Python memakai empat spasi.

> **Menjorok itu penting:** Python memakai jarak menjorok sebagai tanda "baris ini termasuk blok di atasnya". Tanpa menjorok, program tidak jalan. Gunakan empat spasi, dan jangan campur spasi dengan tombol Tab.

Keputusan: if, elif, else

PYTHON
```
luas_ha = 8.0
if luas_ha > 5:
    print("petak besar")
elif luas_ha > 2:
    print("petak sedang")
else:
    print("petak kecil")
```

Penjelasannya:

- `if luas_ha > 5:` bertanya: apakah luas lebih dari 5? Tanda titik dua di ujung wajib ada.
- `elif` (singkatan "else if") hanya diperiksa bila pertanyaan di atasnya salah.
- `else` menampung semua yang tidak cocok dengan pertanyaan di atasnya.
- Python berhenti di pilihan pertama yang benar dan melewati sisanya.

HASIL
```
petak besar
```

Tanda perbandingan yang sering dipakai: `>` lebih dari, `<` kurang dari, `>=` lebih dari atau sama dengan, `==` sama dengan, `!=` tidak sama dengan. Perhatikan: **dua** tanda sama (`==`) untuk bertanya, **satu** tanda sama (`=`) untuk mengisi kotak.

Perulangan: for

PYTHON
```
luas = {"KPH Alpha": 4.0, "KPH Beta": 4.0, "KPH Gamma": 8.0}
total = 0
for nama, ha in luas.items():
    total = total + ha
    if ha > 5:
        print(nama, "termasuk petak besar")
print("Total:", total, "ha")
```

Penjelasannya:

- `total = 0` menyiapkan kotak penampung jumlah, mulai dari nol.
- `for nama, ha in luas.items():` mengulang satu putaran untuk tiap pasangan di kamus. Di tiap putaran, kotak `nama` dan `ha` berisi pasangan yang sedang dibaca.

- `total = total + ha` menambahkan luas petak ke penampung. Baca: "isi kotak total dengan isi total yang lama ditambah ha".
- `if ha > 5:` berada **di dalam** perulangan, karena menjorok dua tingkat. Ia diperiksa tiap putaran.
- `print("Total:" ...)` tidak menjorok. Ia berjalan **sekali**, setelah perulangan selesai.

HASIL
```
KPH Gamma termasuk petak besar
Total: 16.0 ha
```

Alat bantu perulangan: enumerate, break, range

- `range(n)` menghasilkan angka 0 sampai n dikurangi 1. Cocok untuk mengulang sejumlah tertentu.
- `enumerate` menambahkan nomor urut (mulai 0) pada tiap putaran.
- `break` menghentikan perulangan lebih awal.

PYTHON
```
petak = ["P-01", "P-02", "P-03", "P-04", "P-05"]
for i, kode in enumerate(petak):
    print(i, kode)
    if i == 2:
        break
print("selesai")
for n in range(3):
    print("putaran", n)
```

HASIL
```
0 P-01
1 P-02
2 P-03
selesai
putaran 0
putaran 1
putaran 2
```

Pola `enumerate` dengan `break` persis dipakai di skrip Buku B2 untuk menampilkan hanya tiga baris pertama tabel. Angka 2 di sana berarti "berhenti setelah nomor urut 2", yaitu baris **ketiga**.

Comprehension: perulangan satu baris

Kadang Anda hanya ingin mengumpulkan hasil ke dalam daftar baru. **Comprehension** menulisnya dalam satu baris.

PYTHON
```
luas = [4.0, 4.0, 8.0, 2.5]
```

```
besar = [ha for ha in luas if ha > 3]
print(besar)
dobel = [ha * 2 for ha in luas]
print(dobel)
jenis = ["Jati", "Mahoni", "Jati"]
print(sorted({j for j in jenis}))
```

Cara membacanya: "buat daftar berisi `ha`, untuk setiap `ha` di `luas`, bila `ha > 3`". Tanda kurung kurawal tanpa titik dua membuat **set comprehension**.

HASIL

```
[4.0, 4.0, 8.0]
[8.0, 8.0, 16.0, 5.0]
['Jati', 'Mahoni']
```

Skrip yang dibedah: perulangan di b1_00

Tujuan. Menjumlahkan luas tiga petak, dan menandai petak yang lebih dari lima hektare.

Bahan. Tidak ada. Skrip ini tidak memakai QGIS.

Skrip (bagian keempat):

PYTHON

```
luas = {"KPH Alpha": 4.0, "KPH Beta": 4.0, "KPH Gamma": 8.0}

# 4. Perulangan for dan percabangan if. Baris yang menjorok ke dalam ikut dalam blok.
total = 0
for nama, ha in luas.items():
    total = total + ha
    if ha > 5:
        print(nama, "termasuk petak besar")
print("Total:", total, "ha")
```

Baris pertama (`luas = ...`) berasal dari bagian ketiga skrip aslinya. Saya sertakan supaya potongan ini bisa Anda jalankan sendiri.

Penjelasan baris demi baris. Nomor baris dihitung dari potongan di atas.

Baris	Tulisan	Artinya
1	`luas = {...}`	Kamus luas tiga petak (dari bagian ketiga).
3	`# 4. Perulangan ...`	Komentar yang mengingatkan arti menjorok.
4	`total = 0`	Siapkan penampung jumlah, mulai dari nol.
5	`for nama, ha in luas.items():`	Mulai perulangan: satu putaran untuk tiap pasangan nama dan luas.
6	`total = total + ha`	Tambahkan luas petak ke penampung.
7	`if ha > 5:`	Periksa: apakah luas petak ini lebih dari 5?

Baris	Tulisan	Artinya
8	`print(nama, "termasuk petak besar")`	Bila ya, tampilkan nama petak. Menjorok dua tingkat: ia bagian dari `if`.
9	`print("Total:", total, "ha")`	Tampilkan jumlah. Tidak menjorok, jadi berjalan sekali setelah perulangan.

Hasil.

```
HASIL
KPH Gamma termasuk petak besar
Total: 16.0 ha
```

Bagian yang boleh diubah.

- **Angka 5 (baris 7)**: ganti dengan batas lain, misalnya `3`. Petak yang tercetak ikut berubah.
- **Isi kamus (baris 1)**: tambah petak. Total dan daftar petak besar menyesuaikan sendiri.
- Isi blok `if` (baris 8): ganti `print` dengan perintah lain, misalnya menyimpan nama ke daftar.

Jebakan.

- **Salah menjorok.** Bila baris 8 sejajar dengan `if`, Python menolak dengan `IndentationError: expected an indented block after 'if' statement`. Bila jarak menjorok tidak sama, pesannya `IndentationError: unindent does not match any outer indentation level`.
- **Lupa titik dua.** Menulis `for nama, ha in luas.items()` tanpa titik dua di ujung menghasilkan `SyntaxError`.
- Salah tempat `print` terakhir. Bila baris 9 ikut menjorok, jumlah dicetak tiap putaran, bukan sekali.
- **Mengubah daftar sambil mengulanginya.** Menambah atau membuang isi daftar di dalam perulangan atas daftar yang sama bisa menghasilkan urutan yang aneh. Buat daftar baru bila perlu.

Latihan variasi. Ubah skrip supaya mencetak semua petak yang **kurang dari atau sama dengan** 4 hektare, dan menghitung berapa banyak jumlahnya.

Contoh jawaban:

```
PYTHON
luas = {"KPH Alpha": 4.0, "KPH Beta": 4.0, "KPH Gamma": 8.0}
kecil = 0
for nama, ha in luas.items():
    if ha <= 4:
        print(nama, "petak kecil")
        kecil = kecil + 1
print("Jumlah petak kecil:", kecil)
```

```
HASIL
KPH Alpha petak kecil
KPH Beta petak kecil
```

```
Jumlah petak kecil: 2
```

> **Pola yang bisa dipakai ulang:** Hampir semua skrip pengolah data berbentuk ini: **siapkan penampung**, **ulangi tiap data**, **periksa syarat di dalamnya**, **tampilkan hasil setelah perulangan**. Ganti isi data, syarat, dan aksi di dalam `if`, lalu polanya tetap sama.

Cek paham

1. Apa fungsi tanda titik dua di ujung baris `if` dan `for`?
2. Apa beda `=` dan `==`?
3. Bagaimana cara Python tahu bahwa suatu baris termasuk isi dari `for`?

Jawaban.

1. Titik dua menandai bahwa baris-baris yang menjorok sesudahnya adalah isi blok.
2. Satu `=` mengisi kotak. Dua `==` bertanya apakah dua nilai sama.
3. Lewat jarak menjorok (indentasi): baris yang menjorok adalah isi.

Kesalahan umum

- Memakai `=` pada syarat. `if ha = 5` salah. Perbaikan: `if ha == 5`.
- **Menjorok tidak seragam.** Perbaikan: selalu empat spasi, dan atur editor agar Tab menjadi spasi.
- Mengira `range(3)` berisi 1, 2, 3. Isinya 0, 1, 2. Perbaikan: ingat "mulai dari nol".

Ringkasan dan latihan

- `if`, `elif`, `else` mengambil keputusan. `for` mengulang.
- Baris yang menjorok adalah isi blok. Empat spasi.
- `enumerate` menambah nomor urut, `break` menghentikan perulangan, `range` menghasilkan deretan angka.
- Comprehension adalah perulangan satu baris untuk membuat daftar atau set.

Latihan. Dari daftar `[4.0, 4.0, 8.0, 2.5, 12.0]`, cetak hanya luas yang lebih dari 3, lalu cetak jumlahnya. Kerjakan dua cara: dengan `for` dan dengan comprehension.

Tabel perbandingan: perulangan biasa dan comprehension

Hal	Perulangan `for`	Comprehension
Panjang	Beberapa baris	Satu baris
Cocok untuk	Pekerjaan banyak langkah	Menyaring atau mengubah daftar
Contoh	`for ha in luas:` lalu `if` dan `append`	`[ha for ha in luas if ha > 3]`

BAB 7: Resep Bernama: Fungsi

> **Berdasarkan Seri I4 dan B2:** Bagian skrip di bab ini berasal dari Buku I4, *Otomasi Dasar dengan Python* (Intermediate Class), berkas `b1_00_python_dasar.py` bagian 5. Contoh parameter bawaan dan docstring berasal dari Buku B2, *Data Vektor: Atribut, Digitasi, Seleksi* (Basic Class), berkas `b2_05_kalkulasi.py`. Contoh `**kwargs` dari `b2_04_topologi.py`, dan contoh `lambda` dari `b2_06_join.py` dan `b2_07_laporan.py`.

Studi kasus: "Meter persegi ke hektare, lagi dan lagi"

Data dari lapangan selalu dalam meter persegi, tetapi Kepala Seksi hanya mau melihat hektare. Anda menulis `luas / 10000` di belasan tempat. Lalu Kepala Seksi mengubah aturan: pembulatan dua desimal. Anda harus mencari dan mengubah belasan baris itu. Ada cara yang lebih cerdas: tulis satu **resep bernama**, lalu panggil namanya di mana pun dibutuhkan.

Konsep: fungsi dalam tiga kalimat

Sebuah **fungsi** adalah resep yang diberi nama: Anda memberinya bahan, ia mengerjakan langkah-langkahnya, lalu mengembalikan hasil. Bayangkan mesin pengubah satuan: masukkan 40.000 meter persegi, keluar 4,0 hektare. Anda menulis resepnya satu kali, memanggilnya berkali-kali, dan bila aturan berubah, Anda hanya mengubah satu tempat.

Ilustrasi 7.1: Fungsi sebagai resep bernama



Satu resep bernama, dipakai berulang kali dengan masukan berbeda.

Istilah baru di bagian ini:

- **Fungsi**: resep bernama, dibuat dengan kata `def`.
- **Parameter**: nama bahan yang diterima fungsi, ditulis di dalam kurung saat fungsi dibuat.
- **Argumen**: bahan sungguhan yang Anda berikan saat memanggil fungsi.
- `return`: perintah untuk mengembalikan hasil ke pemanggil.

Membuat dan memanggil fungsi

```python
def ke_hektar(meter_persegi):
    return meter_persegi / 10000


print("40000 m2 =", ke_hektar(40000), "ha")
print("125000 m2 =", ke_hektar(125000), "ha")
```

Penjelasannya:

- `def ke_hektar(meter_persegi):` membuat fungsi bernama `ke_hektar` dengan satu parameter. Titik dua dan baris yang menjorok mengikuti aturan yang sama seperti `if` dan `for`.
- `return meter_persegi / 10000` menghitung hasil dan mengirimkannya kembali.
- `ke_hektar(40000)` memanggil fungsi. Angka `40000` adalah argumen. Hasilnya langsung dipakai oleh `print`.

```
40000 m2 = 4.0 ha
125000 m2 = 12.5 ha
```

> **Membuat fungsi tidak menjalankannya:** Baris `def` hanya menyimpan resep. Resep baru dikerjakan saat Anda memanggilnya dengan namanya dan tanda kurung. Pada contoh di atas, pemanggilan ada di dua baris `print`.

Parameter bawaan dan docstring

Parameter boleh punya **nilai bawaan**. Bila pemanggil tidak memberi nilainya, nilai bawaan dipakai. Tepat di bawah `def`, Anda boleh menulis **docstring**: catatan tiga tanda kutip yang menjelaskan fungsi.

```python
def ke_hektar(meter_persegi, desimal=2):
    """Mengubah meter persegi menjadi hektare, dibulatkan sesuai desimal."""
    return round(meter_persegi / 10000, desimal)


print(ke_hektar(125437))
print(ke_hektar(125437, 4))
print(ke_hektar(meter_persegi=125437, desimal=0))
```

Penjelasannya:

- `desimal=2` memberi nilai bawaan dua desimal.
- Pemanggilan pertama tidak memberi `desimal`, jadi dipakai nilai bawaan.
- Pemanggilan kedua memberi `4`, dan menggantikan nilai bawaan.

- Pemanggilan ketiga menyebut nama parameter (`meter_persegi=...`). Cara ini disebut **argumen bernama**, membuat panggilan lebih mudah dibaca.

```
HASIL
12.54
12.5437
13.0
```

Skrip Buku B2 `b2_05_kalkulasi.py` memakai pola yang sama: fungsi `tambah(lyr, nama, tipe, rumus, panjang=0, presisi=0)` punya dua parameter bawaan dan satu docstring.

Menerima argumen bernama sebanyak apa pun: kwargs

Kadang fungsi harus meneruskan banyak pilihan yang berbeda-beda ke perkakas lain. Awalan `**` pada parameter menampung semua argumen bernama ke dalam sebuah kamus.

```
PYTHON
def jalankan(nama_alat, **param):
    print("Alat:", nama_alat)
    print("Pilihan:", param)


jalankan("buffer", INPUT="sungai", JARAK=50)
```

Penjelasannya: `**param` menangkap `INPUT="sungai"` dan `JARAK=50` ke dalam kamus `param`. Nama `param` bebas, tetapi kebiasaannya `kwargs` (singkatan *keyword arguments*).

```
HASIL
Alat: buffer
Pilihan: {'INPUT': 'sungai', 'JARAK': 50}
```

Skrip Buku B2 `b2_04_topologi.py` memakai pola ini: `def jalankan(alg, **param): return processing.run(alg, param)`. Fungsi kecil itu hanya meneruskan pilihan ke alat QGIS.

Fungsi tanpa nama: lambda

Kadang Anda butuh fungsi sangat pendek, dipakai sekali saja. **Lambda** adalah fungsi satu baris tanpa nama.

```
PYTHON
petak = [("P-01", 4.0), ("P-02", 8.0), ("P-03", 2.5)]
terbesar = max(petak, key=lambda p: p[1])
print(terbesar)
urut = sorted(petak, key=lambda p: -p[1])
print([kode for kode, ha in urut])
```

Penjelasannya:

- `lambda p: p[1]` berarti "diberi `p`, kembalikan isi nomor 1 dari `p`", yaitu luasnya.
- `key=` memberi tahu `max` atau `sorted` kriteria apa yang dipakai untuk membandingkan.
- Tanda minus pada `-p[1]` membalik urutan, sehingga yang terbesar di depan.

```
HASIL
('P-02', 8.0)
['P-02', 'P-01', 'P-03']
```

Pola `max(..., key=lambda x: x["VOL_HA"])` dan `sorted(..., key=lambda x: -x["VOL_HA"])` Anda temui di skrip Buku B2 untuk mencari petak bervolume tertinggi.

Skrip yang dibedah: fungsi di b1_00

Tujuan. Mengubah satuan luas lewat sebuah fungsi.

Bahan. Tidak ada. Skrip ini tidak memakai QGIS.

Skrip (bagian kelima):

```
PYTHON
# 5. Fungsi: resep yang diberi nama supaya bisa dipakai berulang kali
def ke_hektar(meter_persegi):
    return meter_persegi / 10000


print("40000 m2 =", ke_hektar(40000), "ha")
```

Penjelasan baris demi baris. Nomor baris dihitung dari potongan di atas.

Baris	Tulisan	Artinya
1	`# 5. Fungsi: ...`	Komentar.
2	`def ke_hektar(meter_persegi):`	Buat fungsi bernama `ke_hektar`. Ia menerima satu bahan bernama `meter_persegi`.
3	`return meter_persegi / 10000`	Bagi bahan dengan 10000, lalu kembalikan hasilnya.
6	`print("40000 m2 =", ke_hektar(40000), "ha")`	Panggil fungsi dengan argumen `40000`, lalu tampilkan hasilnya di tengah kalimat.

Hasil.

```
HASIL
40000 m2 = 4.0 ha
```

Bagian yang boleh diubah.

- **Isi hitungan (baris 3)**: ganti `10000` untuk satuan lain. Contoh: `/ 1000000` untuk kilometer persegi.
- **Nama fungsi dan parameter (baris 2)**: boleh diganti, asal semua pemanggilnya ikut diganti.
- **Argumen (baris 6)**: ganti `40000` dengan luas lain, atau panggil berkali-kali.

Jebakan.

- **Argumen kurang.** Memanggil `ke_hektar()` tanpa isi menghasilkan `TypeError: ke_hektar() missing 1 required positional argument: 'meter_persegi'`.
- Lupa `return`. Fungsi tanpa `return` mengembalikan `None` (artinya "tidak ada nilai"). Hasil `print` akan berbunyi `None`.
- **Memanggil sebelum membuat.** Fungsi harus dibuat (`def`) di baris yang lebih atas daripada tempat dipanggil.
- **Mencampur teks dengan angka.** `"luas: " + ke_hektar(40000)` menghasilkan `TypeError: can only concatenate str (not "float") to str`. Pakai koma di `print`, atau ubah angka ke teks dengan `str(...)`.

Latihan variasi. Tulis fungsi `ke_meter_persegi(hektare)` yang kebalikannya, lalu coba dengan 4 hektare.

Contoh jawaban:

```python
def ke_meter_persegi(hektare):
    return hektare * 10000


print("4 ha =", ke_meter_persegi(4), "m2")
```

```
4 ha = 40000 m2
```

> **Pola yang bisa dipakai ulang:** Bila Anda menulis rumus yang sama lebih dari dua kali, jadikan fungsi: `def nama_fungsi(bahan): return hitungan_dengan_bahan` Lalu panggil `nama_fungsi(isi_bahan)`. Perubahan aturan cukup dilakukan di satu tempat.

Cek paham

1. Apa beda parameter dan argumen?
2. Mengapa fungsi dengan `return` lebih berguna daripada yang hanya `print`?
3. Apa yang ditampung `**param`?

Jawaban.

1. Parameter adalah nama bahan saat fungsi dibuat. Argumen adalah nilai sungguhan saat fungsi dipanggil.
2. Hasil yang dikembalikan bisa dipakai lagi untuk perhitungan atau disimpan. Hasil `print` hanya tampil di layar.
3. Semua argumen bernama yang tidak punya parameter sendiri, dalam bentuk kamus.

Kesalahan umum

- Menulis `def` tetapi tidak pernah memanggilnya. Perbaikan: panggil fungsi dengan nama dan tanda kurung.
- Menaruh `print` padahal butuh `return`. Perbaikan: pakai `return` bila hasilnya mau dipakai lagi.
- **Lupa menjorok isi fungsi.** Perbaikan: empat spasi, sama seperti `if` dan `for`.

Ringkasan dan latihan

- **Fungsi** adalah resep bernama: `def nama(parameter):` lalu isi yang menjorok.
- `return` mengembalikan hasil. Parameter boleh punya nilai bawaan.
- `**param` menampung argumen bernama dalam kamus. `lambda` adalah fungsi satu baris.

Latihan. Tulis fungsi `kategori(luas_ha)` yang mengembalikan `"besar"` bila lebih dari 5 hektare, `"sedang"` bila lebih dari 2, dan `"kecil"` bila tidak. Uji dengan tiga angka berbeda.

Tabel perbandingan: fungsi biasa dan lambda

Hal	Fungsi `def`	`lambda`
Nama	Ada	Tidak ada
Panjang	Beberapa baris	Satu baris
Cocok untuk	Resep yang dipakai berulang	Kriteria pendek, misalnya `key=`

BAB 8: Gudang Perkakas: Modul, Import, dan Jalur Berkas

> **Berdasarkan Seri B2:** Bagian skrip di bab ini berasal dari Buku B2, *Data Vektor: Atribut, Digitasi, Seleksi* (Basic Class), berkas `b2_03_digitasi.py`, bagian pembukanya. Skrip ini dijalankan di dalam QGIS.

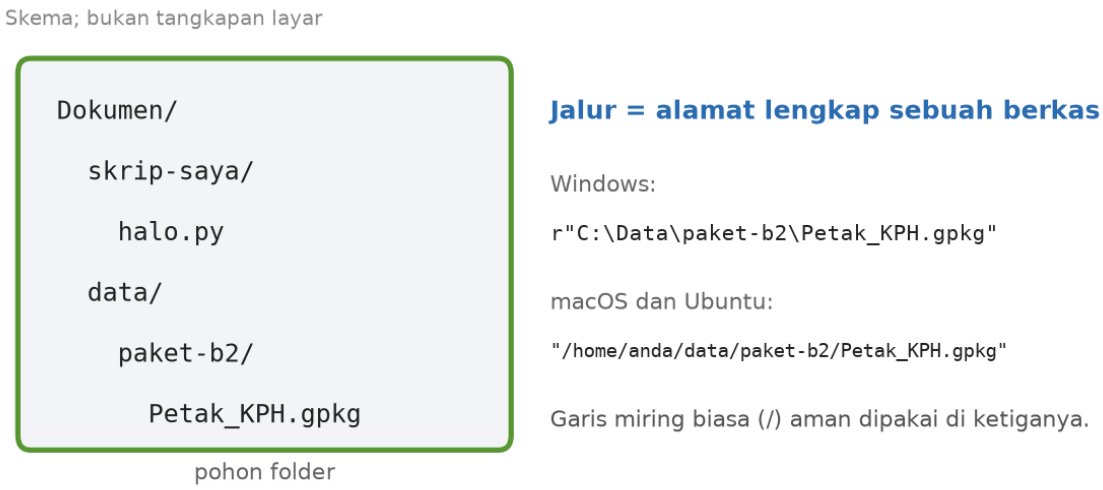
Studi kasus: "Skrip jalan di komputer saya, tapi tidak di komputer kantor"

Analis Muda mengirim skrip ke rekan di kantor cabang. Balasannya singkat: "Galat. Berkas tidak ditemukan." Ternyata skrip itu menyebut folder `C:/Data/...`, dan folder itu hanya ada di komputer Anda. Pelajaran hari ini: cara memanggil perkakas siap pakai, dan cara menulis alamat berkas supaya skrip bisa berpindah komputer.

Konsep: modul dan jalur dalam tiga kalimat

Sebuah **modul** adalah laci perkakas yang sudah disiapkan orang lain, dan `import` adalah perintah untuk membuka laci itu. **Jalur** adalah alamat lengkap sebuah berkas, seperti alamat rumah yang ditulis dari kota sampai nomor rumah. Skrip yang baik tidak menulis alamat mati di tengah kode, melainkan menaruhnya di satu tempat di bagian atas supaya mudah diganti.

Ilustrasi 8.1: Jalur dan pohon folder



Istilah baru di bagian ini:

- **Modul**: berkas Python berisi perkakas siap pakai. Contoh: `os`, `shutil`, `csv`.
- `import`: perintah memanggil sebuah modul.
- **Jalur** (*path*): alamat lengkap berkas atau folder.

- **Variabel lingkungan**: pengaturan bernama yang disimpan oleh sistem operasi, dan bisa dibaca skrip.

Import: membuka laci perkakas

```
PYTHON
import os

print(os.name)
print(os.path.join("data", "paket-b2", "Petak_KPH.gpkg"))
```

Penjelasannya:

- `import os` membuka modul `os`, perkakas untuk urusan sistem operasi: folder, berkas, dan pengaturan.
- `os.name` memberi nama jenis sistem operasi. Di Windows jawabannya `nt`, di macOS dan Ubuntu `posix`. [CEK untuk macOS dan Ubuntu]
- `os.path.join` merangkai potongan alamat dengan pemisah yang **sesuai sistem operasi**. Anda tidak perlu mengingat apakah harus `\` atau `/`.

```
HASIL
nt
data\paket-b2\Petak_KPH.gpkg
```

Di macOS dan Ubuntu, baris kedua tertulis dengan garis miring biasa: `data/paket-b2/Petak_KPH.gpkg`.

Ada tiga cara membuka modul:

- `import os`, lalu pakai `os.makedirs(...)`. Awalan `os.` menunjukkan asal perkakas.
- `from os import makedirs`, lalu pakai `makedirs(...)` langsung.
- `from qgis.core import QgsVectorLayer`: bentuk kedua yang paling sering Anda temui di skrip GIS.

Perkakas berkas: os dan shutil

```
PYTHON
import os
import shutil

os.makedirs("keluaran", exist_ok=True)
with open("asli.txt", "w") as f:
    f.write("Petak P-01")
shutil.copy("asli.txt", os.path.join("keluaran", "salinan.txt"))
print(os.listdir("keluaran"))
print(os.path.exists(os.path.join("keluaran", "salinan.txt")))
print(os.path.basename("data/paket-b2/Petak_KPH.gpkg"))
```

Penjelasannya:

- `os.makedirs("keluaran", exist_ok=True)` membuat folder. `exist_ok=True` berarti "jangan galat bila folder sudah ada".
- Baris `with open(...)` menulis berkas kecil. Cara kerjanya dibahas di Bab 9.
- `shutil.copy(asal, tujuan)` menyalin berkas.
- `os.listdir` mendaftar isi folder, dan `os.path.exists` bertanya apakah sebuah berkas ada.
- `os.path.basename` mengambil nama berkas dari jalur lengkap.

HASIL

```
['salinan.txt']
True
Petak_KPH.gpkg
```

Menaruh alamat di satu tempat

Perhatikan cara skrip GIS menyimpan alamat. Alamat mati tidak diselipkan di tengah kode. Ia ditaruh di baris atas, sebagai kotak bernama berhuruf besar. Cara yang lebih baik lagi: alamat boleh **diganti dari luar** lewat variabel lingkungan.

PYTHON

```
import os

DATA = os.environ.get("B2_DATA", r"C:/KPH_Contoh/paket-b2")
print(DATA)
```

`os.environ.get("B2_DATA", ...)` bertanya kepada sistem: "Adakah pengaturan bernama `B2_DATA`?" Bila ada, nilainya dipakai. Bila tidak, dipakai alamat cadangan di ujung.

HASIL

```
C:/KPH_Contoh/paket-b2
```

Dua hal penting tentang penulisan alamat:

- Huruf `r` di depan kutip (`r"C:\Data"`) membuat garis miring terbalik dibaca apa adanya. Tanpa huruf `r`, garis miring terbalik dipakai Python untuk tanda khusus, misalnya `\n` berarti baris baru, dan alamat jadi salah.
- Garis miring biasa `/` aman dipakai di Windows, macOS, dan Ubuntu. Karena itu banyak skrip GIS menulis `"C:/KPH_Contoh/paket-b2"` dengan garis miring biasa.

Tempat Python mencari modul: sys.path

Saat Anda menulis `import namamodul`, Python mencari modul itu di sejumlah folder. Daftar folder itu tersimpan di `sys.path`. Pada Bab 3 Anda sudah melihat baris `sys.path.append(...)`: baris itu menambah satu folder ke daftar.

```
PYTHON
import sys

print(type(sys.path).__name__)
sys.path.append("modul-saya")
print(sys.path[-1])
```

```
HASIL
list
modul-saya
```

Jadi `sys.path` adalah daftar biasa, dan `append` menambah satu folder di ujung.

Skrip yang dibedah: pembuka b2_03_digitasi.py

Tujuan. Menyiapkan folder hasil dan menyalin tiga berkas data agar berkas asli tidak berubah saat Anda menggambar.

Bahan. Paket data `paket-b2`. Variabel lingkungan `B2_DATA` dan `B2_KELUAR` boleh diisi, atau ganti alamat cadangannya.

Skrip (bagian pembuka):

```
PYTHON
import os
import shutil
from qgis.core import (QgsVectorLayer, QgsRasterLayer, QgsFeature, QgsGeometry, QgsPointXY, QgsPoint,
QgsLineString)

DATA = os.environ.get("B2_DATA", r"C:/KPH_Contoh/paket-b2")
KELUAR = os.environ.get("B2_KELUAR", r"C:/KPH_Contoh/keluaran")
os.makedirs(KELUAR, exist_ok=True)
X0, Y0 = 312000.0, 9996000.0          # sudut barat-daya area contoh

# Salin lapisan kosong agar berkas asli di paket data tidak berubah
for nama in ("Jalan_Baru.gpkg", "Persemaian_Baru.gpkg", "Petak_KPH.gpkg"):
    shutil.copy(os.path.join(DATA, nama), os.path.join(KELUAR, nama))
```

Penjelasan baris demi baris. Nomor baris dihitung dari potongan di atas.

Baris	Tulisan	Artinya
1-2	`import os`, `import shutil`	Buka dua laci perkakas: urusan sistem dan penyalinan berkas.

Baris	Tulisan	Artinya
3	`from qgis.core import (...)`	Ambil delapan perkakas khusus dari QGIS. Kurung panjang boleh dipecah ke beberapa baris.
5	`DATA = os.environ.get(...)`	Isi kotak `DATA` dengan lokasi folder data. Bila pengaturan `B2_DATA` ada, nilainya dipakai.
6	`KELUAR = os.environ.get(...)`	Sama, untuk folder hasil.
7	`os.makedirs(KELUAR, exist_ok=True)`	Buat folder hasil bila belum ada.
8	`X0, Y0 = 312000.0, 9996000.0`	Isi dua kotak sekaligus dengan sepasang angka. Itu koordinat sudut barat-daya area contoh.
10	`# Salin lapisan kosong ...`	Komentar yang menjelaskan alasan penyalinan.
11	`for nama in (...):`	Ulangi untuk tiga nama berkas. Tanda kurung biasa menandakan tuple (Bab 5).
12	`shutil.copy(os.path.join(DATA, nama), os.path.join(KELUAR, nama))`	Rangkai alamat asal dan tujuan dengan `os.path.join`, lalu salin berkas.

Hasil. Skrip bagian ini tidak mencetak apa-apa. Untuk membuktikan salinannya, tambahkan satu baris di ujung:

```
PYTHON
import os
import shutil

DATA = os.environ.get("B2_DATA", r"C:/KPH_Contoh/paket-b2")
KELUAR = os.environ.get("B2_KELUAR", r"C:/KPH_Contoh/keluaran")
os.makedirs(KELUAR, exist_ok=True)
for nama in ("Jalan_Baru.gpkg", "Persemaian_Baru.gpkg", "Petak_KPH.gpkg"):
    shutil.copy(os.path.join(DATA, nama), os.path.join(KELUAR, nama))
print(sorted(os.listdir(KELUAR)))
```

Dengan `B2_DATA` menunjuk folder `paket-b2` dan `B2_KELUAR` menunjuk folder hasil yang kosong, layar menampilkan:

```
TERMINAL
['Jalan_Baru.gpkg', 'Persemaian_Baru.gpkg', 'Petak_KPH.gpkg']
```

Bagian yang boleh diubah.

- **Baris 5, lokasi data.** Ganti alamat cadangan dengan lokasi `paket-b2` di komputer Anda. Atau isi variabel lingkungan `B2_DATA`.
- **Baris 6, lokasi hasil.** Pilih folder hasil di tempat yang Anda kuasai, misalnya `Dokumen/keluaran`.
- **Baris 11, daftar berkas.** Tambah atau kurangi nama berkas yang disalin.

Jebakan.

- **Alamat contoh tidak ada di komputer Anda.** Bila `B2_DATA` tidak diisi dan folder `C:/KPH_Contoh/paket-b2` tidak ada, `shutil.copy` berhenti dengan `FileNotFoundError`. Ganti baris 5.
- Huruf `r` hilang pada alamat Windows. Menulis `"C:\data\nilai"` tanpa huruf `r` membuat `\n` dibaca sebagai baris baru. Alamatnya rusak tanpa pesan yang jelas.
- **Mengubah berkas asli.** Skrip ini sengaja bekerja pada **salinan**. Bila Anda menggambar langsung di berkas asli, data contoh ikut berubah.
- **Sistem operasi berbeda.** Alamat dengan huruf drive (`C:/`) hanya ada di Windows. Di macOS dan Ubuntu, pakai alamat seperti `/home/anda/...`. [CEK]

Latihan variasi. Ubah skrip supaya menyalin **hanya** `Petak_KPH.gpkg`, dan mencetak jalur lengkap salinannya.

Contoh jawaban, hanya bagian yang berubah:

```python
import os
import shutil

DATA = os.environ.get("B2_DATA", r"C:/KPH_Contoh/paket-b2")
KELUAR = os.environ.get("B2_KELUAR", r"C:/KPH_Contoh/keluaran")
os.makedirs(KELUAR, exist_ok=True)
tujuan = os.path.join(KELUAR, "Petak_KPH.gpkg")
shutil.copy(os.path.join(DATA, "Petak_KPH.gpkg"), tujuan)
print("Tersalin ke:", tujuan)
```

> **Pola yang bisa dipakai ulang:** Awali skrip dengan blok alamat di bagian atas: `DATA = os.environ.get("NAMA_PENGATURAN", r"alamat-cadangan")` Semua alamat lain dirangkai dari `DATA` dengan `os.path.join(DATA, "nama berkas")`. Untuk memindahkan skrip ke komputer lain, Anda hanya mengganti satu baris.

Cek paham

1. Apa fungsi `os.path.join`?
2. Mengapa alamat Windows sering diawali huruf `r`?
3. Apa yang terjadi bila variabel lingkungan `B2_DATA` tidak ada?

Jawaban.

1. Merangkai potongan alamat dengan pemisah yang sesuai sistem operasi.
2. Agar garis miring terbalik dibaca apa adanya, bukan sebagai tanda khusus.
3. Skrip memakai alamat cadangan yang tertulis di ujung baris `os.environ.get(...)`.

Kesalahan umum

- **Menyelipkan alamat mati di banyak tempat.** Perbaikan: taruh satu kali di bagian atas.

- **Memakai satu garis miring terbalik dalam teks biasa.** Perbaikan: tambah huruf `r` di depan kutip, atau pakai garis miring biasa.
- **Lupa membuat folder hasil.** Perbaikan: panggil `os.makedirs(..., exist_ok=True)` sebelum menulis ke folder itu.

Ringkasan dan latihan

- `import` membuka laci perkakas. `os` mengurus folder dan berkas, `shutil` menyalin.
- `os.path.join` merangkai alamat sesuai sistem operasi. Garis miring biasa aman di tiga sistem.
- Taruh alamat di satu tempat, dan bila perlu bisa diganti lewat variabel lingkungan.

Latihan. Tulis skrip yang membuat folder `hasil_latihan`, membuat berkas `catatan.txt` di dalamnya, lalu mencetak daftar isi folder itu.

Tabel perbandingan: alamat di tiga sistem operasi

Hal	Windows	macOS	Ubuntu
Contoh alamat	`C:/Data/paket-b2`	`/Users/anda/data/paket-b2`	`/home/anda/data/paket-b2`
Pemisah folder biasa	`\`	`/`	`/`
`os.name`	`nt`	`posix`	`posix`

BAB 9: Catatan dan Laporan: Berkas dan Merakit Teks

> **Berdasarkan Seri B2:** Bagian skrip di bab ini berasal dari Buku B2, *Data Vektor: Atribut, Digitasi, Seleksi* (Basic Class), berkas `b2_07_laporan.py`, bagian ekspor ke CSV. Contoh f-string tidak ada di skrip Buku B2. Format `%` dipakai di Buku I4 dan di skrip ArcMap (Bagian C).

Studi kasus: "Kirim ringkasannya dalam bentuk tabel"

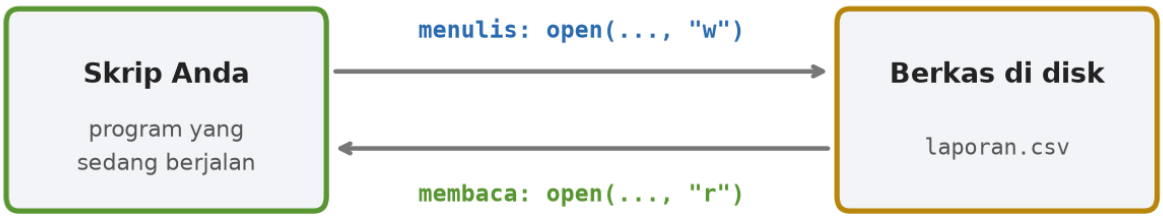
Kepala Seksi meminta daftar petak jati berumur tua dalam berkas yang bisa dibuka di Excel. Hasil skrip Anda sejauh ini hanya tampil di layar, dan hilang saat jendela ditutup. Anda perlu dua kemampuan: **menulis ke berkas**, dan **merakit teks** dengan rapi supaya laporannya enak dibaca.

Konsep: berkas dan teks dalam tiga kalimat

Berkas adalah buku catatan di disk: skrip Anda bisa menulisnya, menutupnya, lalu membacanya lagi besok. `with open(...)` adalah cara aman membuka buku itu, karena Python menutupnya sendiri setelah selesai. **Merakit teks** adalah menempelkan nilai ke dalam kalimat, seperti mengisi titik-titik pada surat.

Ilustrasi 9.1: Skrip dan berkas

Skema; bukan tangkapan layar



with open(...) menutup berkas otomatis setelah selesai.

Istilah baru di bagian ini:

- `open`: perintah membuka berkas. Mode `"w"` untuk menulis, `"r"` untuk membaca.
- `with`: bungkus yang menutup berkas otomatis.
- **CSV**: berkas teks berisi tabel, kolom dipisah koma. Bisa dibuka di Excel.
- **JSON**: berkas teks berisi data bersusun, mirip kamus.
- **f-string**: teks bertanda `f` di depan, yang boleh memuat `{nilai}` di dalamnya.

Merakit teks: tiga cara

```
PYTHON
nama = "KPH Gamma"
luas = 8.0
print("%s luasnya %.2f ha" % (nama, luas))
print(f"{nama} luasnya {luas:.2f} ha")
print("{} luasnya {:.2f} ha".format(nama, luas))
```

Tiga baris itu menghasilkan teks yang sama.

- Cara `%`: `%s` menandai tempat teks, `%.2f` menandai angka dengan dua desimal. Nilainya menyusul setelah tanda `%` dalam kurung. Cara ini dipakai di Buku I4 dan skrip ArcMap.
- **Cara f-string**: awali dengan huruf `f`, lalu tulis nama kotak di dalam `{ }`. Bagian `:.2f` mengatur dua desimal. Paling mudah dibaca, tetapi hanya ada di Python 3.6 ke atas, jadi tidak ada di ArcMap.
- Cara `.format`: `{}` menandai tempat nilai. Nilainya menyusul di `.format(...)`.

```
HASIL
KPH Gamma luasnya 8.00 ha
KPH Gamma luasnya 8.00 ha
KPH Gamma luasnya 8.00 ha
```

> **Pilih yang mana?:** Di Python 3 (QGIS dan ArcGIS Pro), pakai f-string: paling jelas. Di ArcMap (Python 2.7), pakai `%`. Anda akan sering menemui ketiganya di skrip orang lain, jadi kenali semuanya.

Menulis dan membaca berkas teks

```
PYTHON
with open("catatan.txt", "w", encoding="utf-8") as f:
    f.write("Petak P-01\n")
    f.write("Petak P-02\n")

with open("catatan.txt", "r", encoding="utf-8") as f:
    for baris in f:
        print(baris.strip())
```

Penjelasannya:

- `open("catatan.txt", "w", ...)` membuka berkas untuk **ditulis**. Berkas lama dengan nama itu ditimpa.
- `encoding="utf-8"` menjaga huruf Indonesia dan simbol tidak rusak. Biasakan menulisnya.
- `f.write(...)` menulis teks. `\n` berarti pindah baris.
- Blok kedua membuka berkas untuk **dibaca**. Perulangan `for baris in f` membaca satu baris tiap putaran.

- `.strip()` membuang spasi dan pindah baris di ujung teks.

HASIL
```
Petak P-01
Petak P-02
```

Tabel dalam berkas: CSV

Modul `csv` mengurus koma dan tanda kutip untuk Anda.

PYTHON
```
import csv

baris = [["KODE_PETAK", "NAMA_KPH", "LUAS_HA"], ["P-01", "KPH Alpha", 4.0], ["P-02", "KPH Beta", 4.0]]
with open("jati.csv", "w", newline="", encoding="utf-8") as fh:
    w = csv.writer(fh)
    for b in baris:
        w.writerow(b)

with open("jati.csv", encoding="utf-8") as fh:
    for b in csv.reader(fh):
        print(b)
```

Penjelasannya:

- `newline=""` mencegah baris kosong ekstra di Windows.
- `csv.writer(fh)` membuat penulis. `w.writerow(...)` menulis satu baris tabel.
- `csv.reader(fh)` membaca berkas itu, baris demi baris, dalam bentuk daftar.
- Perhatikan: saat dibaca kembali, **semua isi berupa teks**, termasuk angka `4.0` menjadi `'4.0'`. Ubah dengan `float(...)` bila perlu menghitung.

HASIL
```
['KODE_PETAK', 'NAMA_KPH', 'LUAS_HA']
['P-01', 'KPH Alpha', '4.0']
['P-02', 'KPH Beta', '4.0']
```

Data bersusun: JSON

JSON menyimpan kamus dan daftar sebagai teks. Cocok untuk pengaturan atau data yang berlapis.

PYTHON
```
import json

data = {"kph": "KPH Alpha", "petak": ["P-01", "P-02"], "luas_ha": 4.0}
teks = json.dumps(data, ensure_ascii=False)
print(teks)
kembali = json.loads(teks)
print(kembali["petak"][1])
```

`json.dumps` mengubah kamus menjadi teks JSON. `json.loads` mengubah teks itu kembali menjadi kamus. `ensure_ascii=False` menjaga huruf non-Latin tetap terbaca.

```
HASIL
{"kph": "KPH Alpha", "petak": ["P-01", "P-02"], "luas_ha": 4.0}
P-02
```

Skrip yang dibedah: ekspor ke CSV di b2_07

Skrip `b2_07_laporan.py` membuat rekap akhir dari sebuah layer QGIS. Di bab ini kita bedah bagian akhirnya, yang mengekspor petak jati berumur tua ke berkas CSV. Skrip lengkapnya dibahas di Bab 15.

Tujuan. Menulis daftar petak terpilih ke berkas CSV yang bisa dibuka di Excel.

Bahan. Layer `lap` yang sudah ada dan berisi seleksi (dari bagian atas skrip), serta folder `KELUAR`. Potongan ini hanya bisa dijalankan sebagai bagian dari skrip lengkap, di dalam QGIS.

Skrip (potongan):

```
PYTHON
with open(KELUAR + "/Jati_Tua.csv", "w", newline="", encoding="utf-8") as fh:
    w = csv.writer(fh)
    w.writerow(["KODE_PETAK", "NAMA_KPH", "LUAS_HA", "UMUR", "POHON_HA", "VOL_HA"])
    for f in lap.selectedFeatures():
        w.writerow([f["KODE_PETAK"], f["NAMA_KPH"], f["LUAS_HA"], f["UMUR"], f["POHON_HA"], f["VOL_HA"]])
```

Penjelasan baris demi baris. Nomor baris dihitung dari potongan di atas.

Baris	Tulisan	Artinya
1	`with open(KELUAR + "/Jati_Tua.csv", "w", ...) as fh:`	Buka berkas CSV untuk ditulis di folder hasil. Sambungan teks dengan + merangkai alamatnya. Berkas diberi nama pendek `fh`.
2	`w = csv.writer(fh)`	Buat penulis CSV yang menulis ke berkas itu.
3	`w.writerow([...])`	Tulis baris judul kolom.
4	`for f in lap.selectedFeatures():`	Ulangi untuk tiap petak yang sedang terpilih di layer.
5	`w.writerow([f["KODE_PETAK"], ...])`	Tulis satu baris tabel: isi enam kolom dari petak yang sedang dibaca.

Hasil. Berkas `Jati_Tua.csv` berisi satu baris judul dan satu baris untuk tiap petak terpilih. Bentuk isinya mengikuti contoh `jati.csv` di atas.

Bagian yang boleh diubah.

- **Nama berkas (baris 1)**: ganti `Jati_Tua.csv` menjadi nama lain.

- **Baris judul (baris 3)**: ganti nama kolom, dan sesuaikan isi di baris 5 dengan urutan yang sama.
- **Kolom yang ditulis (baris 5)**: tambah atau kurangi kolom. Jumlah dan urutannya harus sama dengan baris judul.

Jebakan.

- Lupa `newline=""`. Di Windows, berkas CSV punya baris kosong di antara isi.
- **Menimpa berkas lama tanpa sadar.** Mode `"w"` menimpa. Pakai nama berbeda, atau mode `"a"` untuk menambah di ujung.
- **Nilai kosong.** Kolom yang kosong di data (nilai `None`) ditulis sebagai teks kosong. Cek dulu bila hasilnya akan dihitung.
- **Huruf rusak di Excel.** Bila nama berisi huruf khusus, Excel di beberapa versi lebih suka `utf-8-sig`. [CEK]

Latihan variasi. Tulis skrip mandiri yang menulis daftar tiga petak (kode dan luas) ke `petak.csv`, lalu membacanya kembali dan mencetak jumlah luas seluruhnya.

Contoh jawaban:

PYTHON
```
import csv

petak = [["P-01", 4.0], ["P-02", 8.0], ["P-03", 2.5]]
with open("petak.csv", "w", newline="", encoding="utf-8") as fh:
    w = csv.writer(fh)
    w.writerow(["KODE", "LUAS_HA"])
    for p in petak:
        w.writerow(p)

total = 0
with open("petak.csv", encoding="utf-8") as fh:
    pembaca = csv.reader(fh)
    next(pembaca)
    for baris in pembaca:
        total = total + float(baris[1])
print("Total luas:", total, "ha")
```

HASIL
```
Total luas: 14.5 ha
```

Baris `next(pembaca)` melewati satu baris pertama, yaitu baris judul. Baris `float(baris[1])` mengubah teks kembali menjadi angka.

> **Pola yang bisa dipakai ulang: Menulis:** `with open(nama, "w", newline="", encoding="utf-8") as fh:` lalu `csv.writer(fh)`, `writerow(judul)`, dan perulangan `writerow(isi)`. **Membaca:** `with open(nama, encoding="utf-8") as fh:` lalu `for baris in csv.reader(fh):`, ingat semua isi berupa teks.

Cek paham

1. Apa keuntungan `with open(...)` dibanding `open(...)` saja?
2. Mengapa angka yang dibaca dari CSV perlu diubah dengan `float(...)`?
3. Apa bedanya mode `"w"` dan `"r"`?

Jawaban.

1. Berkas ditutup otomatis setelah blok selesai, bahkan bila terjadi galat.
2. Karena semua isi CSV terbaca sebagai teks, dan teks tidak bisa dijumlahkan sebagai angka.
3. `"w"` untuk menulis dan menimpa isi lama. `"r"` untuk membaca.

Kesalahan umum

- **Memakai alamat berkas yang folder induknya belum ada.** Perbaikan: buat dulu dengan `os.makedirs(..., exist_ok=True)`.
- **Membuka berkas CSV di Excel saat skrip masih menulis.** Perbaikan: tutup Excel dulu, jalankan skrip, lalu buka hasilnya.
- **Mencampur cara merakit teks dalam satu skrip.** Perbaikan: pilih satu cara dan konsisten.

Ringkasan dan latihan

- Berkas ditulis dan dibaca dengan `with open(...)`. Mode `"w"` menulis, `"r"` membaca.
- Rakit teks dengan `%`, `.format`, atau f-string.
- `csv` untuk tabel, `json` untuk data bersusun.

Latihan. Buat berkas `rekap.csv` yang berisi daftar tiga KPH dan luasnya. Lalu tulis skrip yang membacanya dan mencetak satu kalimat untuk tiap KPH dengan f-string.

Tabel perbandingan: tiga cara merakit teks

Cara	Contoh	Catatan
`%`	`"%s %.2f" % (nama, luas)`	Tersedia di Python 2.7 dan 3
`.format`	`"{} {:.2f}".format(nama, luas)`	Tersedia di Python 2.7 dan 3
f-string	`f"{nama} {luas:.2f}"`	Hanya Python 3.6 ke atas

BAB 10: Membaca Petunjuk Komputer: Galat dan Cara Menanganinya

> **Berdasarkan Seri I4:** Skrip bab ini berasal dari Buku I4, *Otomasi Dasar dengan Python* (Intermediate Class), berkas `b4_01_galat_dasar.py`. Skrip ini dijalankan di dalam QGIS.

Studi kasus: "Lima berkas kiriman tim lapangan"

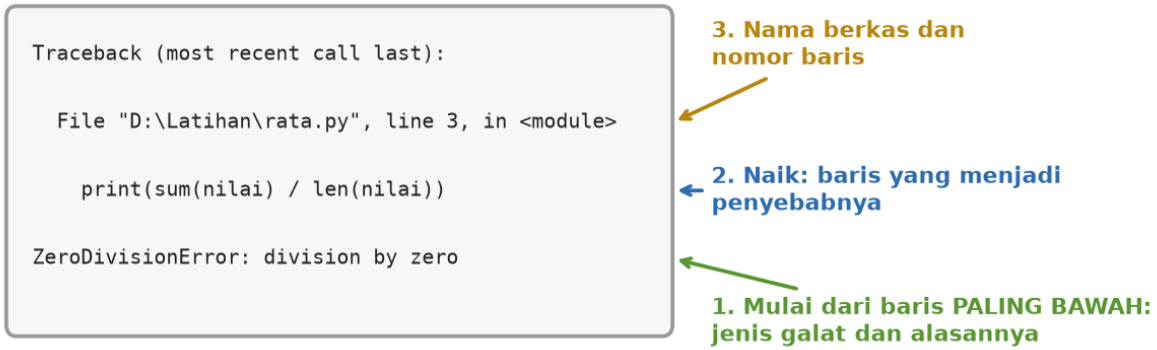
Tim lapangan mengirim lima berkas data. Anda menjalankan skrip pembaca, dan skrip berhenti di berkas ketiga dengan tulisan merah panjang. Dua berkas sisanya tidak sempat dibaca. Kepala Seksi bertanya: "Mengapa satu berkas rusak menghentikan semuanya?" Jawabannya: skrip Anda belum **siap menghadapi galat**. Bab ini mengajarkan cara membaca galat dan cara menanganinya dengan tenang.

Konsep: galat dalam tiga kalimat

Galat (*error*) adalah pesan dari Python bahwa ia tidak bisa melanjutkan, lengkap dengan petunjuk letak dan sebabnya. Bayangkan petugas yang menolak berkas dengan catatan: "halaman 3, baris 5: tanda tangan kurang". Dengan **try/except**, Anda memberi petugas itu instruksi: "bila ada berkas bermasalah, catat lalu lanjutkan ke berkas berikutnya".

Ilustrasi 10.1: Cara membaca pesan galat

Skema; bukan tangkapan layar



Galat bukan tanda Anda gagal. Galat adalah petunjuk arah.

Istilah baru di bagian ini:

- **Galat** (*exception*): peristiwa yang menghentikan skrip.
- **Traceback**: tulisan jejak galat. Dibaca dari baris paling bawah ke atas.
- `try` dan `except`: coba kerjakan, dan bila galat, tangkap lalu tangani.
- `raise`: perintah untuk **menimbulkan** galat sendiri dengan pesan yang jelas.
- `assert`: perintah memastikan sebuah syarat terpenuhi.

Membaca pesan galat

Simpan tiga baris ini sebagai `rata.py`, lalu jalankan:

```
PYTHON
nilai = []
print("Menghitung rata-rata")
print(sum(nilai) / len(nilai))
```

Python menampilkan:

```
TERMINAL
Menghitung rata-rata
Traceback (most recent call last):
  File "D:\Latihan\rata.py", line 3, in <module>
    print(sum(nilai) / len(nilai))
          ~~~~~~~~~~~^~~~~~~~~~~~
ZeroDivisionError: division by zero
```

Baris pertama (`Menghitung rata-rata`) adalah hasil `print` sebelum galat. Alamat berkas di komputer Anda akan berbeda. Cara membacanya, **dari bawah ke atas**:

1. **Baris paling bawah** menyebut jenis galat dan alasannya: `ZeroDivisionError: division by zero`. Artinya: ada pembagian dengan nol.
2. Baris bergaris `~~~^~~~` menandai bagian kode yang bermasalah, tepat di bawah baris penyebab: `print(sum(nilai) / len(nilai))`.
3. Baris `File ...` menyebut nama berkas dan nomor baris: berkas `rata.py`, baris 3.

Kini Anda bisa langsung ke baris 3 dan bertanya: mengapa `len(nilai)` bernilai nol? Karena daftar `nilai` kosong.

Jenis galat yang paling sering muncul

Jenis galat	Contoh pesan	Penyebab umum
`IndexError`	`list index out of range`	Nomor daftar melebihi isi
`KeyError`	`'B'`	Kunci kamus tidak ada
`NameError`	`name 'Nama' is not defined`	Salah ketik atau huruf besar-kecil
`TypeError`	`can only concatenate str (not "float") to str`	Mencampur teks dengan angka
`ValueError`	`invalid literal for int() with base 10: 'empat'`	Isi tidak bisa diubah ke jenis yang diminta
`FileNotFoundError`	`No such file or directory`	Alamat berkas salah
`ModuleNotFoundError`	`No module named 'qgis'`	Modul belum terpasang atau salah penjalan
`IndentationError`	`expected an indented block`	Jarak menjorok salah

Jenis galat	Contoh pesan	Penyebab umum
`ZeroDivisionError`	`division by zero`	Pembagi bernilai nol

Menangani galat: try dan except

PYTHON
```
def persen(bagian, total):
    try:
        return 100 * bagian / total
    except ZeroDivisionError:
        return None


print("Persen 3 dari 12:", persen(3, 12))
print("Persen 3 dari 0:", persen(3, 0))
```

Penjelasannya:

- Python mencoba isi `try` lebih dulu.
- Bila **tidak ada** galat, hasil dikembalikan seperti biasa.
- Bila terjadi `ZeroDivisionError`, Python melompat ke blok `except` yang cocok. Di sini jawabannya `None`, artinya "tidak ada jawaban yang masuk akal".
- Skrip **tidak berhenti**. Baris `print` berikutnya tetap berjalan.

HASIL
```
Persen 3 dari 12: 25.0
Persen 3 dari 0: None
```

Blok lengkapnya punya empat bagian:

- `try`: yang dicoba.
- `except JenisGalat`: ditangani bila galat itu muncul. Boleh ada beberapa `except`.
- `else`: berjalan hanya bila **tidak** ada galat.
- `finally`: **selalu** berjalan, galat atau tidak. Cocok untuk merapikan.

Menimbulkan galat sendiri: raise dan assert

PYTHON
```
def luas_ke_hektar(m2):
    if m2 < 0:
        raise ValueError("luas tidak boleh negatif: " + str(m2))
    return m2 / 10000


try:
    luas_ke_hektar(-5)
except ValueError as e:
    print("Galat tertangkap:", e)
```

```
luas = 3
assert luas > 0, "luas harus positif"
print("lolos")
```

Penjelasannya:

- `raise ValueError("...")` menghentikan fungsi dengan pesan buatan Anda yang jelas.
- `except ValueError as e` menangkap galat itu, dan `e` berisi pesannya.
- `assert syarat, "pesan"` memastikan syarat benar. Bila salah, skrip berhenti dengan `AssertionError` dan pesan Anda.

HASIL

```
Galat tertangkap: luas tidak boleh negatif: -5
lolos
```

Skrip yang dibedah: memeriksa masukan sebelum dipakai

Tujuan. Membuka empat berkas kiriman lapangan satu per satu. Berkas yang bermasalah dilewati dengan pesan yang jelas, dan berkas yang baik tetap dibaca.

Bahan. Paket data `paket-i4`, folder `Kiriman_Lapangan` berisi `tim_a.gpkg` (baik), `tim_e.gpkg` (rusak), dan `tim_f.gpkg` (kolom `Kondisi` tidak ada). Berkas `tim_x.gpkg` memang tidak ada. Ganti dua baris awal `DATA` dan `HASIL` sesuai komputer Anda. Dijalankan di Konsol atau Editor QGIS.

Skrip utuh:

PYTHON

```
# [SKRIP 4.1: Memeriksa masukan sebelum dipakai dan menangkap galat dengan try-except]
# Penulis: Badar Mubarok Yogaswara
import os
from qgis.core import QgsVectorLayer

DATA = "D:/Latihan/paket-i4"
HASIL = "D:/Latihan/hasil_i4"


def muat_vektor(jalur, nama, field_wajib=()):
    """Membuka layer vektor. Bila ada yang tidak beres, berhenti dengan pesan yang jelas."""
    if not os.path.exists(jalur):
        raise FileNotFoundError("berkas tidak ada: " + jalur)
    layer = QgsVectorLayer(jalur, nama, "ogr")
    if not layer.isValid():
        raise ValueError("berkas tidak bisa dibuka (mungkin rusak): " + os.path.basename(jalur))
    kurang = [f for f in field_wajib if layer.fields().indexOf(f) < 0]
    if kurang:
        raise ValueError("field wajib tidak ada: " + ", ".join(kurang))
    if layer.featureCount() == 0:
        raise ValueError("layer kosong, tidak ada fitur")
    return layer


# Empat berkas dicoba satu per satu. Satu gagal tidak boleh menghentikan yang lain.
kasus = [
    DATA + "/Kiriman_Lapangan/tim_a.gpkg",      # baik
```

```
    DATA + "/Kiriman_Lapangan/tim_x.gpkg",      # tidak ada
    DATA + "/Kiriman_Lapangan/tim_e.gpkg",      # rusak
    DATA + "/Kiriman_Lapangan/tim_f.gpkg",      # field Kondisi tidak ada
]
for jalur in kasus:
    nama = os.path.basename(jalur)
    try:
        layer = muat_vektor(jalur, nama, field_wajib=["Kondisi", "Tinggi_Phn"])
    except FileNotFoundError as e:
        print("[LEWATI] %s -> %s" % (nama, e))
    except ValueError as e:
        print("[LEWATI] %s -> %s" % (nama, e))
    else:
        # blok else: hanya jalan bila TIDAK ada galat
        print("[OK]     %s -> %d fitur" % (nama, layer.featureCount()))
    finally:
        # blok finally: selalu jalan, berhasil atau gagal
        print("         selesai memeriksa", nama)

# Galat hitungan yang sering terjadi: pembagian dengan nol
def persen(bagian, total):
    try:
        return 100 * bagian / total
    except ZeroDivisionError:
        return None          # tidak ada jawaban yang masuk akal

print("Persen 3 dari 12:", persen(3, 12))
print("Persen 3 dari 0:", persen(3, 0))
```

Penjelasan baris demi baris. Skrip ini panjang, jadi dibahas dalam empat blok. Nomor baris mengikuti skrip di atas.

Blok 1: pembuka (baris 1-7).

Baris	Tulisan	Artinya
1-2	`# [SKRIP 4.1 ...`	Judul dan nama penulis.
3-4	`import os`, `from qgis.core import QgsVectorLayer`	Buka perkakas sistem dan perkakas pembuka layer vektor.
6-7	`DATA = ...`, `HASIL = ...`	Dua alamat di satu tempat (Bab 8). Ganti sesuai komputer Anda.

Blok 2: fungsi penjaga `muat_vektor` (baris 10-22). Fungsi ini memeriksa empat hal secara berurutan. Begitu satu pemeriksaan gagal, ia menimbulkan galat dengan pesan yang jelas, dan berhenti.

Baris	Tulisan	Artinya
10	`def muat_vektor(jalur, nama, field_wajib=()):`	Buat fungsi dengan tiga bahan. Bahan ketiga boleh dikosongkan karena punya nilai bawaan: tuple kosong.
11	`"""Membuka layer ..."""`	Docstring: catatan penjelasan fungsi (Bab 7).
12-13	`if not os.path.exists(jalur): raise FileNotFoundError(...)`	Pemeriksaan 1: berkas harus ada. Bila tidak, timbulkan galat jenis `FileNotFoundError` dengan pesan buatan sendiri.

Baris	Tulisan	Artinya
14-16	`layer = QgsVectorLayer(...)` lalu `if not layer.isValid(): raise ValueError(...)`	Pemeriksaan 2: berkas harus bisa dibuka. Berkas rusak menghasilkan layer tidak valid.
17-19	`kurang = [... for f in field_wajib if ... < 0]` lalu `raise`	Pemeriksaan 3: semua kolom wajib harus ada. Comprehension (Bab 6) mengumpulkan kolom yang hilang. `indexOf` memberi -1 bila kolom tidak ada.
20-21	`if layer.featureCount() == 0: raise ...`	Pemeriksaan 4: layer tidak boleh kosong.
22	`return layer`	Semua lolos, kembalikan layernya.

Blok 3: perulangan dengan try (baris 25-45).

Baris	Tulisan	Artinya
25-31	`kasus = [...]`	Daftar empat alamat berkas yang akan dicoba. Komentar menandai kondisi masing-masing.
32-33	`for jalur in kasus:` lalu `nama = os.path.basename(jalur)`	Ulangi tiap alamat. Ambil nama berkasnya saja.
34-35	`try: layer = muat_vektor(...)`	Coba buka berkas dengan fungsi penjaga. Kolom wajib: `Kondisi` dan `Tinggi_Phn`.
36-39	`except FileNotFoundError as e:` dan `except ValueError as e:`	Dua jenis galat ditangani. Tiap kali, cetak pesan `[LEWATI]` dan lanjut ke berkas berikutnya.
40-42	`else:`	Berjalan hanya bila tidak ada galat: cetak `[OK]` dan jumlah fitur.
43-45	`finally:`	Selalu berjalan: cetak "selesai memeriksa".

Blok 4: galat hitungan (baris 47-55). Fungsi `persen` memakai pola yang sama seperti contoh di awal bab. Ia mengembalikan `None` bila pembaginya nol.

Hasil. Jalankan dengan `DATA` menunjuk folder paket Anda. Hasil di komputer penulis (jalur pada baris `[LEWATI] tim_x.gpkg` mengikuti nilai `DATA` Anda):

```
HASIL
[OK]      tim_a.gpkg -> 6 fitur
          selesai memeriksa tim_a.gpkg
[LEWATI] tim_x.gpkg -> berkas tidak ada: D:/Latihan/paket-i4/Kiriman_Lapangan/tim_x.gpkg
          selesai memeriksa tim_x.gpkg
[LEWATI] tim_e.gpkg -> berkas tidak bisa dibuka (mungkin rusak): tim_e.gpkg
          selesai memeriksa tim_e.gpkg
[LEWATI] tim_f.gpkg -> field wajib tidak ada: Kondisi
          selesai memeriksa tim_f.gpkg
Persen 3 dari 12: 25.0
```

```
Persen 3 dari 0: None
```

Satu berkas baik dan tiga berkas bermasalah, tetapi **keempatnya diperiksa**. Itulah tujuan `try/except` di dalam perulangan.

Bagian yang boleh diubah.

- **Baris 6, lokasi data.** Ganti dengan lokasi `paket-i4` di komputer Anda.
- **Baris 26-31, daftar berkas.** Ganti dengan berkas Anda sendiri.
- **Baris 35, kolom wajib.** Ganti nama kolom sesuai data Anda: `field_wajib=["KODE", "LUAS"]`.
- Isi pesan di `print` dan `raise`. Ganti dengan kalimat yang paling jelas bagi rekan kerja Anda.

Jebakan.

- Urutan `except`. Python memakai `except` pertama yang cocok. Tulis galat yang spesifik di atas, dan yang umum di bawah.
- `except` kosong. Menulis `except:` tanpa jenis menangkap semuanya, termasuk salah ketik Anda sendiri, dan menyembunyikan masalah. Selalu sebut jenis galatnya.
- **Jalur contoh.** Bila `DATA` belum diganti, semua berkas akan `[LEWATI]` dengan pesan "berkas tidak ada". Itu bukan kerusakan data.
- **Menyembunyikan terlalu banyak.** Menangkap galat lalu diam saja membuat masalah baru tak terlihat. Selalu cetak atau catat sesuatu.

Latihan variasi. Tambahkan satu berkas ke daftar `kasus`, yaitu `tim_c.gpkg`, lalu jalankan lagi. Apa hasilnya?

Jawabannya bergantung pada isi `tim_c.gpkg`. Pola yang Anda pelajari memberi tahu Anda cara mencari tahu: baca baris `[OK]` atau `[LEWATI]`. Bila `[LEWATI]`, pesan setelah panah menyebut alasannya.

> **Pola yang bisa dipakai ulang: Penjaga masukan:** fungsi yang memeriksa syarat satu per satu dan melempar `raise` dengan pesan jelas. **Pemakai penjaga:** perulangan berisi `try`, `except JenisGalat as e` yang mencetak pesan, dan `else` untuk jalur sukses. Ganti isi pemeriksaan dan daftar berkas, kerangkanya tetap.

Cek paham

1. Pada pesan galat, baris mana yang dibaca pertama?
2. Apa beda `else` dan `finally` pada blok `try`?
3. Mengapa `except:` tanpa jenis galat tidak dianjurkan?

Jawaban.

1. Baris paling bawah, yang menyebut jenis dan alasan galat.
2. `else` berjalan hanya bila tidak ada galat. `finally` selalu berjalan.
3. Karena menangkap semua galat, termasuk kesalahan tulis Anda sendiri, sehingga masalah tersembunyi.

Kesalahan umum

- **Panik melihat tulisan merah panjang.** Perbaikan: baca baris paling bawah dulu, lalu naik.
- **Menangkap galat tanpa mencetak apa pun.** Perbaikan: selalu beri pesan, supaya Anda tahu apa yang dilewati.
- Membungkus seluruh skrip dalam satu `try`. Perbaikan: bungkus hanya bagian yang memang bisa gagal, misalnya membuka berkas.

Ringkasan dan latihan

- Baca galat **dari bawah ke atas**: jenis dan alasan, baris penyebab, nama berkas dan nomor baris.
- `try/except/else/finally` menangani galat tanpa menghentikan skrip.
- `raise` menimbulkan galat dengan pesan sendiri. `assert` memastikan sebuah syarat.

Latihan. Tulis fungsi `rata_rata(daftar)` yang mengembalikan `None` bila daftar kosong, dan rata-ratanya bila tidak. Uji dengan daftar kosong dan daftar berisi tiga angka.

Tabel perbandingan: empat bagian blok try

Bagian	Kapan berjalan	Kegunaan
`try`	Selalu dicoba lebih dulu	Pekerjaan yang bisa gagal
`except`	Hanya bila galat itu muncul	Menangani galat dan mencetak pesan
`else`	Hanya bila tidak ada galat	Jalur sukses
`finally`	Selalu	Merapikan, menutup, atau mencatat

BAB 11: Cetakan dan Pembungkus: Objek, Kelas, dan Dekorator

> **Berdasarkan Seri M4 dan M3:** Potongan skrip di bab ini berasal dari Buku M4, *Pengembangan dan Publikasi* (Master Class), berkas `kerapatan_hotspot.py` dan `tambah_luas_ha.py`. Kelas buatan sendiri yang lebih besar ada di Buku M3, *Deret Waktu dan Pemodelan* (Master Class), berkas `hutan_mini.py`. Keduanya dibahas lengkap di Bagian B.

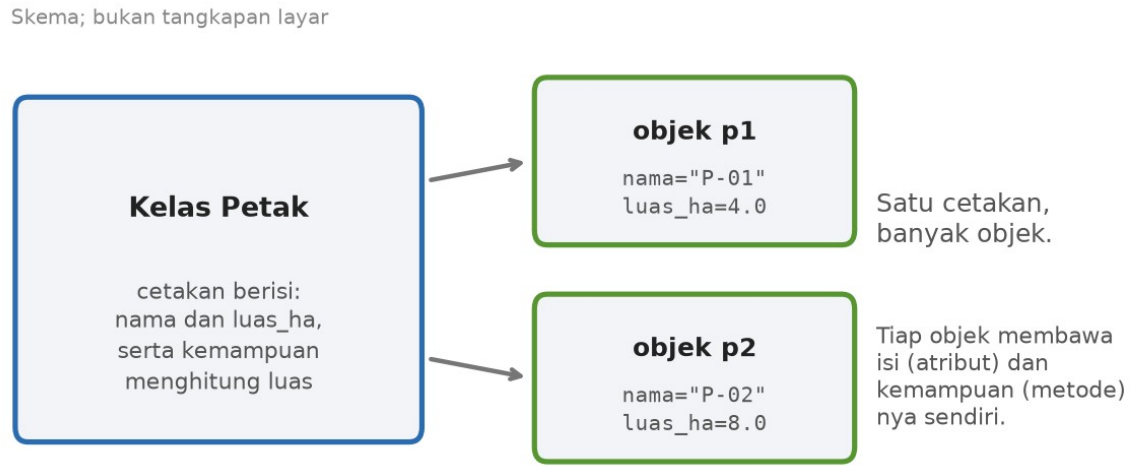
Studi kasus: "Kok perintah QGIS ditulis dengan titik di tengah?"

Analis Muda membaca skrip PyQGIS dan melihat baris seperti `petak.featureCount()` dan `layer.fields()`. Titik di tengah dan kurung di ujung itu apa artinya? Di sebagian skrip, ada pula kata `class` dan tanda `@` di atas fungsi. Semua itu membuka satu gagasan yang membuat PyQGIS dan ArcPy terasa masuk akal: **objek**.

Konsep: kelas dan objek dalam tiga kalimat

Sebuah **kelas** adalah cetakan, dan sebuah **objek** adalah barang yang dicetak darinya. Bayangkan cetakan kue: satu cetakan, banyak kue, dan tiap kue punya isinya sendiri. Objek membawa **isi** (disebut atribut) dan **kemampuan** (disebut metode), dan keduanya diakses dengan titik: `objek.isi` dan `objek.kemampuan()`.

Ilustrasi 11.1: Kelas sebagai cetakan



Istilah baru di bagian ini:

- **Kelas** (*class*): cetakan yang dibuat dengan kata `class`.
- **Objek**: hasil cetakan. Bisa banyak, dan masing-masing punya isi sendiri.
- **Atribut**: isi yang menempel pada objek, misalnya `p1.luas_ha`.
- **Metode**: fungsi yang menempel pada objek, misalnya `p1.kategori()`.

Membuat kelas sendiri

```
PYTHON
class Petak:
    def __init__(self, kode, luas_ha):
        self.kode = kode
        self.luas_ha = luas_ha

    def luas_m2(self):
        return self.luas_ha * 10000

    def kategori(self):
        if self.luas_ha > 5:
            return "besar"
        return "kecil"


p1 = Petak("P-01", 4.0)
p2 = Petak("P-02", 8.0)
print(p1.kode, p1.luas_m2(), p1.kategori())
print(p2.kode, p2.luas_m2(), p2.kategori())
```

Penjelasannya:

- `class Petak:` membuat cetakan bernama `Petak`.
- `def __init__(self, kode, luas_ha):` adalah **pembuat objek**. Python memanggilnya otomatis saat Anda menulis `Petak(...)`. Dua garis bawah di kiri dan kanan namanya itu wajib.
- `self` berarti "objek yang sedang dibuat atau dipakai". Ia selalu menjadi parameter pertama sebuah metode. Anda tidak mengisinya sendiri saat memanggil.
- `self.kode = kode` menyimpan bahan ke dalam objek sebagai atribut.
- `luas_m2` dan `kategori` adalah **metode**: fungsi biasa yang dipasang di dalam kelas.
- `p1 = Petak("P-01", 4.0)` mencetak satu objek. `p2` mencetak yang lain. Isi mereka terpisah.

```
HASIL
P-01 40000.0 kecil
P-02 80000.0 besar
```

Pewarisan: kelas turunan

Sebuah kelas boleh **mewarisi** kelas lain: ia mengambil semua isi dan kemampuannya, lalu mengganti sebagian. Tulis nama kelas induk di dalam kurung.

```
PYTHON
class Petak:
    def __init__(self, kode, luas_ha):
        self.kode = kode
        self.luas_ha = luas_ha

    def luas_m2(self):
        return self.luas_ha * 10000
```

```
    def kategori(self):
        return "biasa"


class PetakLindung(Petak):
    def kategori(self):
        return "lindung"


p3 = PetakLindung("P-03", 2.0)
print(p3.kode, p3.kategori(), p3.luas_m2())
```

`PetakLindung` tidak menulis ulang `__init__` dan `luas_m2`: ia mengambilnya dari `Petak`. Ia hanya mengganti `kategori`.

HASIL

```
P-03 lindung 20000.0
```

Pewarisan inilah yang dipakai QGIS. Skrip algoritma Processing di Buku M4 diawali `class KerapatanHotspot(QgsProcessingAlgorithm):`. Artinya: kelas baru Anda mewarisi seluruh kemampuan dasar algoritma QGIS, lalu Anda mengisi bagian yang khas, seperti nama dan parameternya.

Dekorator: pembungkus fungsi

Sebuah **dekorator** adalah pembungkus yang dipasang di atas fungsi dengan tanda `@`. Bayangkan kotak kado: fungsi di dalamnya sama, tetapi ada tambahan pada pembungkusnya.

PYTHON

```
def catat(fungsi):
    def bungkus(*args, **kwargs):
        print("memanggil", fungsi.__name__)
        return fungsi(*args, **kwargs)
    return bungkus


@catat
def ke_hektar(m2):
    return m2 / 10000


print(ke_hektar(40000))
```

Penjelasannya:

- `catat` menerima sebuah fungsi, lalu mengembalikan fungsi pembungkus yang mencetak catatan sebelum menjalankan yang asli.
- `@catat` di atas `def ke_hektar` sama dengan menulis `ke_hektar = catat(ke_hektar)`.
- `*args` dan `**kwargs` menampung semua argumen apa pun, lalu meneruskannya.

```
HASIL
memanggil ke_hektar
4.0
```

Anda hampir tidak perlu **menulis** dekorator sendiri. Yang penting, Anda bisa **mengenalinya** di skrip orang lain: tanda `@` di atas `def` berarti "fungsi ini dibungkus dengan fasilitas tertentu".

Skrip yang dibedah: dua gaya algoritma Processing

Di Buku M4 ada dua skrip yang membuat alat baru untuk QGIS. Skrip pertama bergaya **kelas**, skrip kedua bergaya **dekorator**. Di Bagian B (Bab 27) keduanya dibahas lengkap dan dijalankan. Di sini kita hanya mengenali bentuknya.

Tujuan. Mengenali kelas dan dekorator di skrip nyata.

Bahan. Tidak ada. Potongan ini tidak dijalankan, karena butuh Processing QGIS.

Potongan 1: gaya kelas (`kerapatan_hotspot.py`).

```
PYTHON
class KerapatanHotspot(QgsProcessingAlgorithm):
    PETAK = "PETAK"
    TITIK = "TITIK"
    BATAS = "BATAS"
    HASIL = "HASIL"

    def createInstance(self):
        return KerapatanHotspot()

    def name(self):
        return "kerapatan_hotspot"
```

Penjelasan baris demi baris. Nomor baris dihitung dari potongan di atas.

Baris	Tulisan	Artinya
1	`class KerapatanHotspot(QgsProcessingAlgorithm):`	Buat kelas baru yang mewarisi `QgsProcessingAlgorithm`, cetakan algoritma milik QGIS.
2-5	`PETAK = "PETAK"` dan tiga baris lain	Empat atribut tetap berupa teks. Nama berhuruf besar menandakan nilainya tidak berubah. Dipakai sebagai nama parameter.
7-8	`def createInstance(self): return KerapatanHotspot()`	Metode yang wajib ada. Ia mencetak satu salinan baru kelas ini bila QGIS memintanya.
10-11	`def name(self): return "kerapatan_hotspot"`	Metode yang memberi nama alat. QGIS memanggilnya sendiri.

Potongan 2: gaya dekorator (`tambah_luas_ha.py`).

```python
@alg(name="tambah_luas_ha", label="Tambah kolom luas (hektare)", group="kehutanan", group_label="Kehutanan")
@alg.input(type=alg.SOURCE, name="INPUT", label="Layer poligon", types=[2])
@alg.input(type=alg.SINK, name="OUTPUT", label="Hasil")
def tambah_luas_ha(instance, parameters, context, feedback, inputs):
```

Baris	Tulisan	Artinya
1	`@alg(name=..., label=..., group=...)`	Dekorator yang menjadikan fungsi di bawahnya sebuah alat QGIS, lengkap dengan nama, label, dan kelompoknya.
2	`@alg.input(type=alg.SOURCE, ...)`	Dekorator tambahan: menyatakan satu masukan berupa layer poligon.
3	`@alg.input(type=alg.SINK, ...)`	Dekorator tambahan: menyatakan satu keluaran berupa layer hasil.
4	`def tambah_luas_ha(...)`	Fungsi yang dibungkus. Isinya menghitung luas dan menulis hasilnya.

Hasil. Potongan ini tidak mencetak apa-apa. Di Bab 27 Anda akan melihat alat-alat ini muncul di kotak peralatan Processing QGIS.

Bagian yang boleh diubah. Nama, label, dan grup alat boleh Anda ganti dengan tulisan sendiri. Nama parameter (misalnya `"INPUT"`) harus sama di semua tempat yang memakainya.

Jebakan.

- Lupa `self`. Metode yang tidak menyebut `self` sebagai parameter pertama menghasilkan `TypeError` saat dipanggil, karena Python tetap menyodorkan objeknya.
- Lupa mengisi atribut lewat `self`. Menulis `kode = kode` di `__init__` hanya membuat kotak sementara. Isinya hilang. Tulis `self.kode = kode`.
- **Salah mewarisi.** Kelas turunan harus menyebut induknya di dalam kurung. Tanpa itu, ia tidak mendapat kemampuan induk.

Latihan variasi. Tambahkan metode `kerapatan(self, jumlah_titik)` ke kelas `Petak` pada contoh di atas. Metode itu mengembalikan jumlah titik dibagi luas hektare. Uji dengan petak 8 hektare dan 4 titik.

Contoh jawaban:

```python
class Petak:
    def __init__(self, kode, luas_ha):
        self.kode = kode
        self.luas_ha = luas_ha

    def kerapatan(self, jumlah_titik):
        return jumlah_titik / self.luas_ha


p = Petak("P-02", 8.0)
print(p.kerapatan(4))
```

HASIL

```
0.5
```

Pola yang bisa dipakai ulang: Saat melihat `x.sesuatu` atau `x.sesuatu()`, bacalah: `x` adalah objek, `sesuatu` adalah isi atau kemampuannya. Untuk mencari tahu daftar kemampuan sebuah objek, pakai `dir(x)` (Bab 13). Saat melihat `class Nama(Induk):`, bacalah: cetakan baru yang mewarisi `Induk`. Saat melihat `@sesuatu` di atas `def`, bacalah: **fungsi ini dibungkus**.

Cek paham

1. Apa beda kelas dan objek?
2. Apa arti `self` di dalam sebuah metode?
3. Apa yang dilakukan tanda `@` di atas sebuah fungsi?

Jawaban.

1. Kelas adalah cetakan. Objek adalah barang yang dicetak dari cetakan itu, dan tiap objek punya isi sendiri.
2. `self` berarti objek yang sedang dipakai. Lewat `self`, metode membaca dan mengubah isi objek itu.
3. Menandai bahwa fungsi di bawahnya dibungkus oleh dekorator, yaitu fungsi lain yang menambahkan fasilitas.

Kesalahan umum

- **Memanggil metode tanpa kurung.** `p1.kategori` hanya menunjuk fungsinya. Perbaikan: tulis `p1.kategori()`.
- **Mengira dua objek berbagi isi.** Tiap objek membawa isinya sendiri. Perbaikan: ubah isi lewat objek yang dimaksud.
- Takut pada `class` dan `@`. Perbaikan: pada tahap ini cukup kenali bentuknya. Menulisnya sendiri baru dibutuhkan di Bab 27.

Ringkasan dan latihan

- **Kelas** adalah cetakan, **objek** adalah barang cetakan. Objek punya **atribut** (isi) dan **metode** (kemampuan).
- `__init__` membuat objek, `self` menunjuk objek itu.
- **Pewarisan** mengambil isi kelas induk, **dekorator** membungkus fungsi.

Latihan. Buat kelas `Layer` dengan atribut `nama` dan `jumlah_fitur`, serta metode `ringkas()` yang mengembalikan kalimat `"<nama> memuat <jumlah> fitur"`. Buat dua objek, lalu cetak ringkasan keduanya.

Tabel perbandingan: tiga hal yang mirip tetapi berbeda

Hal	Fungsi	Metode
Letak	Berdiri sendiri	Di dalam kelas
Cara memanggil	`ke_hektar(40000)`	`p1.luas_m2()`
Bahan khusus	Hanya parameter	Selalu ada `self`

BAB 12: Kotak Perkakas Tambahan: Pustaka Ilmiah

> **Berdasarkan Seri I2 dan M3:** Potongan skrip di bab ini berasal dari Buku I2, *Analisis Raster dan Pengindraan Jauh Dasar* (Intermediate Class), berkas `i2_04_tak_terbimbing.py`. Pustaka `matplotlib` dan `scikit-learn` dipakai di Buku M3, *Deret Waktu dan Pemodelan* (Master Class), dan `scipy` di Buku M2, *Basis Data Spasial dan Analisis Lanjut* (Master Class).

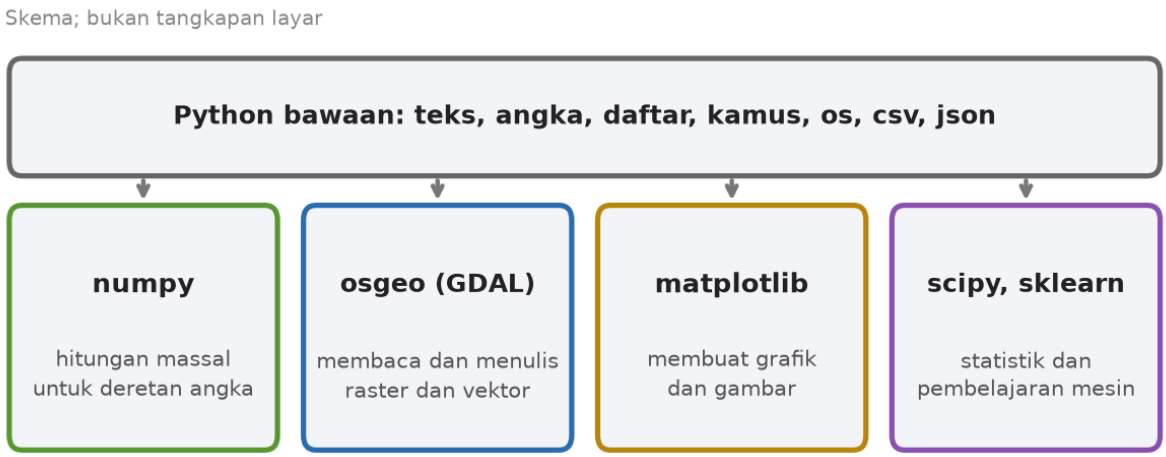
Studi kasus: "Rata-rata dari sembilan puluh ribu piksel"

Kepala Seksi meminta rata-rata nilai sebuah citra. Citra itu 300 kali 300 piksel, jadi ada 90.000 angka. Dengan perulangan `for`, Anda harus menulis dan menunggu. Ada pustaka yang menghitung semuanya sekaligus dalam satu baris. Bab ini memperkenalkan empat pustaka yang paling sering muncul di skrip GIS.

Konsep: pustaka dalam tiga kalimat

Sebuah **pustaka** adalah kotak perkakas tambahan yang ditulis orang lain dan boleh Anda pakai. Python sendiri sudah membawa perkakas dasar, dan pustaka menambahkan perkakas khusus: hitungan angka massal, pembaca raster, pembuat grafik. Anda memanggilnya dengan `import`, seperti di Bab 8.

Ilustrasi 12.1: Pustaka ilmiah



Istilah baru di bagian ini:

- **Pustaka** (*library*): kotak perkakas tambahan.
- **Array**: tabel angka berbentuk kotak, dengan baris dan kolom, dikelola oleh `numpy`.
- **Bentuk** (*shape*): ukuran sebuah array, misalnya 3 baris dan 4 kolom.

> **Pustaka mana yang sudah ada di komputer Anda?:** Python bawaan QGIS 4.0.2 di komputer penulis sudah memuat `numpy` (2.4.4) dan `osgeo` atau GDAL (3.12.4). Ia juga memuat `matplotlib` (3.10.9), `scipy` (1.17.1), dan `psycopg2` (2.9.12). `scikit-learn` **tidak ada** di sana. Python biasa di komputer penulis hanya punya `numpy` dan `matplotlib`. Karena itu skrip yang memakai `osgeo` dijalankan dengan Python bawaan QGIS (Bab 3). Pustaka lain bisa dipasang dengan `pip install nama-pustaka`. [CEK untuk caranya di QGIS Anda]

numpy: kalkulator untuk deretan angka

PYTHON

```
import numpy as np

luas = np.array([4.0, 4.0, 8.0, 2.5])
print(luas.sum(), luas.mean())
print(luas[luas > 3])
print(luas * 2)
a = np.arange(12).reshape(3, 4)
print(a.shape, a[1, 2], a.sum(axis=0))
```

Penjelasannya:

- `import numpy as np` memanggil `numpy` dan memberinya nama pendek `np`. Itu kebiasaan umum.
- `np.array([...])` membuat array dari sebuah daftar.
- `luas.sum()` dan `luas.mean()` menjumlahkan dan merata-ratakan seluruh isi sekaligus, tanpa `for`.
- `luas[luas > 3]` menyaring: hanya nilai yang lebih dari 3.
- `luas * 2` mengalikan **setiap** isi dengan 2. Daftar biasa tidak bisa melakukannya.
- `np.arange(12).reshape(3, 4)` membuat angka 0 sampai 11, lalu menatanya menjadi 3 baris dan 4 kolom.
- `a.shape` memberi bentuknya. `a[1, 2]` mengambil baris nomor 1, kolom nomor 2. `a.sum(axis=0)` menjumlahkan tiap **kolom**.

HASIL

```
18.5 4.625
[4. 4. 8.]
[ 8.  8. 16.  5.]
(3, 4) 6 [12 15 18 21]
```

osgeo (GDAL): pembaca raster

GDAL adalah pembaca dan penulis hampir semua jenis berkas peta. Di Python, ia diimpor dari paket `osgeo`. Raster dibaca menjadi array `numpy`, jadi dua pustaka ini hampir selalu berpasangan.

Skrip yang dibedah: membaca citra di `i2_04`.

Tujuan. Membuka citra empat band, mengubahnya menjadi array, lalu menyusunnya menjadi tabel satu baris per piksel.

Bahan. Paket data `paket-i2`, berkas `Citra_Lahan.tif`. Jalankan di Python bawaan QGIS.

Potongan skrip (baris 15 sampai 19 di berkas aslinya):

PYQGIS
```
ds = gdal.Open(os.path.join(PAKET_I2, "Citra_Lahan.tif"))
gt, proj = ds.GetGeoTransform(), ds.GetProjection()
citra = ds.ReadAsArray().astype("float64")                    # bentuk (4, tinggi, lebar)
_, tinggi, lebar = citra.shape
piksel = citra.reshape(4, -1).T                               # satu baris = satu piksel, empat kolom = empat
band
```

Penjelasan baris demi baris. Nomor baris dihitung dari potongan di atas.

Baris	Tulisan	Artinya
1	`ds = gdal.Open(os.path.join(...))`	Buka berkas citra. `ds` (singkatan *dataset*) adalah objek yang mewakili citra itu.
2	`gt, proj = ds.GetGeoTransform(), ds.GetProjection()`	Ambil dua keterangan letak: `gt` (titik awal dan ukuran piksel) dan `proj` (sistem koordinat). Dua isi sekaligus ke dua kotak.
3	`citra = ds.ReadAsArray().astype("float64")`	Baca seluruh piksel menjadi array `numpy`, lalu ubah jenis angkanya menjadi desimal. Bentuknya (band, tinggi, lebar).
4	`_, tinggi, lebar = citra.shape`	Ambil bentuknya. Kotak `_` menampung angka band yang tidak dipakai.
5	`piksel = citra.reshape(4, -1).T`	Ubah bentuk: empat baris (band), semua piksel dijejer. Tanda `.T` memutar tabel, jadi satu baris untuk satu piksel dan empat kolom untuk empat band. `-1` berarti "hitung sendiri".

Hasil. Untuk melihat bentuknya, tambahkan beberapa `print` di ujung potongan. Berikut hasil di komputer penulis dengan `PAKET_I2` menunjuk folder `paket-i2`:

HASIL
```
4 (4, 300, 300) (90000, 4)
(312000.0, 1.0, 0.0, 9996300.0, 0.0, -1.0)
60.1
```

- Baris pertama: citra punya 4 band, bentuk array `(4, 300, 300)`, dan tabel piksel `(90000, 4)`.
- Baris kedua: `gt` berisi koordinat sudut kiri-atas `312000.0` dan `9996300.0`, serta ukuran piksel 1 meter (`1.0` dan `-1.0`).
- Baris ketiga: rata-rata band pertama, dibulatkan dua desimal.

Layar Anda juga bisa menampilkan peringatan `FutureWarning` dari GDAL tentang `gdal.UseExceptions()`. Itu hanya peringatan versi mendatang. Skrip tetap berjalan.

Bagian yang boleh diubah.

- **Lokasi citra (baris 1).** Ganti `PAKET_I2` dan nama berkas dengan citra Anda.
- Angka 4 di `reshape(4, -1)` (baris 5). Samakan dengan jumlah band citra Anda, yaitu `ds.RasterCount`.
- `float64` (baris 3). Bila citra Anda besar, `float32` menghemat memori. [CEK]

Jebakan.

- **Salah alamat.** Bila alamat citra salah, `gdal.Open` mengembalikan `None`, dan baris berikutnya gagal dengan `AttributeError` karena `None` tidak punya `GetGeoTransform`. Periksa alamat lebih dulu.
- **Jumlah band tidak cocok.** `reshape(4, -1)` gagal bila band bukan empat. Pakai `ds.RasterCount`.
- **Memori.** Citra besar menjadi array besar di memori. Citra 300 kali 300 empat band masih kecil, tetapi citra satelit penuh bisa memenuhi memori.
- Indeks band dari 1 di GDAL, dari 0 di `numpy`. `ds.GetRasterBand(1)` adalah band pertama, tetapi `citra[0]` juga band pertama.

matplotlib: pembuat grafik

PYTHON

```
import os
import matplotlib
matplotlib.use("Agg")
import matplotlib.pyplot as plt

plt.bar(["Alpha", "Beta", "Gamma"], [4.0, 4.0, 8.0])
plt.ylabel("Luas (ha)")
plt.savefig("luas.png", dpi=100)
print(os.path.exists("luas.png"))
```

Penjelasannya:

- `matplotlib.use("Agg")` memilih mode **tanpa jendela**: grafik hanya disimpan ke berkas. Aman dijalankan di QGIS dan di terminal.
- `plt.bar(nama, nilai)` menggambar batang. `plt.ylabel` memberi judul sumbu tegak.
- `plt.savefig("luas.png", dpi=100)` menyimpan gambar. `dpi` mengatur ketajaman.

HASIL

```
True
```

Baris terakhir membuktikan berkas `luas.png` sudah ada. Buka berkas itu dengan penampil gambar untuk melihat grafiknya.

scipy dan scikit-learn: statistik dan pembelajaran mesin

`scipy` berisi banyak perkakas ilmiah. Salah satunya `curve_fit`, yang mencari garis terbaik untuk sekumpulan titik. Buku M2 memakainya pada skrip interpolasi.

PYQGIS

```
import numpy as np
from scipy.optimize import curve_fit

x = np.array([1.0, 2.0, 3.0, 4.0])
y = np.array([2.1, 3.9, 6.2, 7.8])


def garis(x, a, b):
    return a * x + b


(a, b), _ = curve_fit(garis, x, y)
print(round(float(a), 2), round(float(b), 2))
```

Penjelasannya: fungsi `garis` adalah rumus garis lurus dengan dua angka yang belum diketahui, `a` dan `b`. `curve_fit` mencari `a` dan `b` yang paling cocok dengan empat titik data. Hasilnya kemiringan dan titik potong.

HASIL

```
1.94 0.15
```

`scikit-learn` (nama modulnya `sklearn`) berisi alat pembelajaran mesin, misalnya klasifikasi hutan acak. Pustaka itu **tidak ada** di Python bawaan QGIS 4.0.2. Karena itu Buku M3 menyertakan `hutan_mini.py`, versi ringkas yang hanya memakai `numpy`, sehingga skrip tetap jalan tanpa `scikit-learn`.

Bagaimana mengenali pustaka di skrip orang lain

Lihat baris `import` di bagian atas skrip. Bacalah:

- `import numpy as np`: ada hitungan angka massal. Cari `np.`.
- `from osgeo import gdal`: ada pembacaan raster. Cari `gdal.`.
- `import matplotlib...`: ada grafik.
- `from scipy... import ...`: ada statistik atau optimisasi.
- `import psycopg2`: ada koneksi ke basis data PostGIS (Buku M2).

Anda tidak perlu menguasai semuanya. Pada skrip yang Anda pelajari di Bagian B, perkakas ini akan muncul satu per satu, dengan penjelasan seperlunya.

Latihan variasi. Dari array `[4.0, 4.0, 8.0, 2.5]`, hitung jumlah luas petak yang lebih dari 3 hektare memakai `numpy`, tanpa `for`.

Contoh jawaban:

```python
import numpy as np

luas = np.array([4.0, 4.0, 8.0, 2.5])
print(luas[luas > 3].sum())
```

```
16.0
```

> **Pola yang bisa dipakai ulang:** Raster hampir selalu diolah dengan pola tiga langkah: **baca** dengan GDAL menjadi array, **hitung** dengan `numpy`, lalu **tulis** kembali dengan GDAL. Ganti hanya nama berkas dan rumus hitungnya.

Cek paham

1. Mengapa `luas * 2` bekerja pada array `numpy`, tetapi tidak pada daftar biasa?
2. Pustaka mana yang membaca berkas raster?
3. Apa fungsi `matplotlib.use("Agg")`?

Jawaban.

1. `numpy` memperlakukan array sebagai satu kesatuan, sehingga operasi berlaku untuk setiap isinya. Daftar biasa hanya menggandakan daftarnya.
2. GDAL, yang diimpor dari paket `osgeo`.
3. Memilih mode tanpa jendela, sehingga grafik hanya disimpan ke berkas.

Kesalahan umum

- Memakai `for` untuk hitungan massal. Perbaikan: pakai `numpy` bila datanya besar.
- **Salah menjalankan.** Skrip yang memakai `osgeo` gagal dengan `ModuleNotFoundError` di Python biasa. Perbaikan: jalankan dengan Python bawaan QGIS.
- **Mencampur indeks band.** Perbaikan: ingat GDAL mulai dari 1, `numpy` mulai dari 0.

Ringkasan dan latihan

- Pustaka adalah kotak perkakas tambahan, dipanggil dengan `import`.
- `numpy` untuk hitungan massal, GDAL (`osgeo`) untuk raster, `matplotlib` untuk grafik, `scipy` dan `sklearn` untuk statistik dan pembelajaran mesin.
- Raster dibaca GDAL menjadi array `numpy`, dihitung, lalu ditulis kembali.

Latihan. Dengan `numpy`, buat array berisi lima nilai suhu, lalu cetak rata-rata, nilai terbesar, dan jumlah nilai di atas rata-rata.

Tabel perbandingan: empat pustaka

Pustaka	Dipanggil dengan	Dipakai untuk
numpy	`import numpy as np`	Hitungan angka massal
GDAL	`from osgeo import gdal`	Membaca dan menulis raster
matplotlib	`import matplotlib.pyplot as plt`	Grafik dan gambar
scipy	`from scipy... import ...`	Statistik dan optimisasi

BAB 13: Belajar Mandiri: Mencari, Mencoba, Mengubah

Studi kasus: "Tidak ada contohnya di buku!"

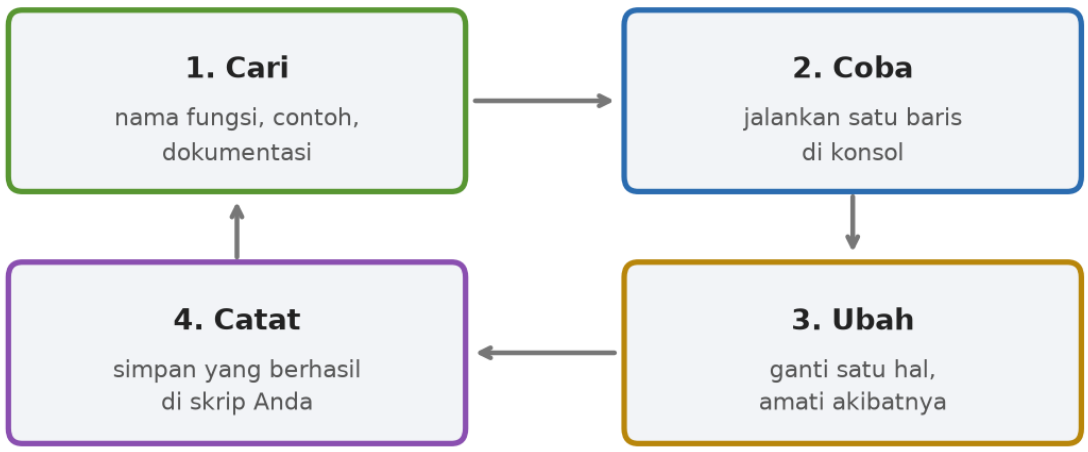
Seminggu kemudian, Kepala Seksi meminta sesuatu yang tidak ada di buku mana pun: "Tampilkan sistem koordinat sebuah layer." Analis Muda tidak menemukan skrip siap pakai. Di titik ini, pembaca yang hanya menyalin skrip akan berhenti. Pembaca yang paham **pola** belajar akan lanjut: mencari nama perintahnya, mencobanya satu baris, lalu menyusunnya. Bab ini memberi Anda perkakas untuk itu.

Konsep: siklus belajar dalam tiga kalimat

Belajar menulis skrip bukan menghafal semua perintah, melainkan **berputar dalam empat langkah**: cari, coba, ubah, catat. Bayangkan Anda belajar jalan di kota baru: bertanya arah, melangkah satu blok, menyesuaikan, lalu mencatat jalan pintas. Setelah beberapa putaran, Anda tidak lagi butuh peta.

Ilustrasi 13.1: Siklus belajar mandiri

Skema; bukan tangkapan layar



Istilah baru di bagian ini:

- **Dokumentasi**: halaman resmi yang menjelaskan tiap perintah.
- `dir()`: perintah yang menampilkan daftar isi dan kemampuan sebuah objek.
- `help()`: perintah yang menampilkan penjelasan sebuah fungsi.
- `type()`: perintah yang menyebut jenis sebuah nilai.

Langkah 1: Tanya objeknya sendiri

Python bisa menjelaskan dirinya. Tiga perintah ini adalah teman terbaik Anda.

PYTHON
```
print(type(3.5).__name__, type("teks").__name__, type([1, 2]).__name__)
print([n for n in dir(str) if not n.startswith("_")][:6])
help(str.upper)
```

Penjelasannya:

- `type(x)` menyebut jenis nilai. `.__name__` mengambil namanya saja.
- `dir(str)` menampilkan semua kemampuan sebuah teks. Daftar aslinya panjang dan penuh nama berawalan garis bawah. Perulangan di atas menyaring nama yang tidak berawalan `_`, lalu mengambil enam pertama.
- `help(str.upper)` menampilkan penjelasan fungsi `upper`.

HASIL
```
float str list
['capitalize', 'casefold', 'center', 'count', 'encode', 'endswith']
Help on method_descriptor:

upper(self, /) unbound builtins.str method
    Return a copy of the string converted to uppercase.
```

Penjelasan `help` bisa terlihat sedikit berbeda di komputer Anda, tergantung versi Python. Intinya sama.

Langkah 2: Tanya objek QGIS

Cara yang sama berlaku untuk objek QGIS. Bayangkan Anda mencari perintah untuk menghitung jumlah fitur sebuah layer. Daftar kemampuan `QgsVectorLayer` panjang, jadi saring dengan kata kunci:

PYQGIS
```
from qgis.core import QgsVectorLayer

print([n for n in dir(QgsVectorLayer) if n.startswith("featureC")])
print(len([n for n in dir(QgsVectorLayer) if "feature" in n.lower()]) > 5)
```

Baris pertama mencari nama yang berawalan `featureC`. Baris kedua memeriksa bahwa ada lebih dari lima nama yang memuat kata "feature".

HASIL
```
['featureCount']
True
```

Nama `featureCount` ketemu. Anda sudah bertemu perintah itu di Bab 15 (`petak.featureCount()`), jadi Anda tahu cara kerjanya. Untuk perintah yang belum Anda kenal, lanjutkan ke dokumentasi resmi.

> **Konsol QGIS punya pelengkap otomatis:** Di Konsol Python QGIS, ketik `petak.` lalu tekan **Ctrl+Alt+Space** untuk melihat daftar kemampuan objek `petak`. Daftar riwayat perintah muncul dengan **Ctrl+Shift+Space**. [terbukti: manual resmi QGIS]

Langkah 3: Baca dokumentasi resmi

Dokumentasi adalah sumber yang paling bisa dipercaya. Ada tiga yang perlu Anda simpan alamatnya:

Untuk	Alamat	Isi
Semua kelas PyQGIS	qgis.org/pyqgis	Dokumentasi API, dikelompokkan per modul (core, gui, analysis, processing). Tersedia untuk versi master, 4.2, dan 3.44.
Cara-cara menggunakan PyQGIS	docs.qgis.org/latest/en/docs/pyqgis_developer_cookbook	Buku resep PyQGIS: membuka konsol, menjalankan skrip, memuat layer, dan lainnya.
Semua ArcPy	doc.esri.com/en/arcgis-pro/latest/arcpy/main/arcgis-pro-arcpy-reference.html	Referensi ArcPy: fungsi, kelas, dan modul seperti `arcpy.mp`, `arcpy.da`, `arcpy.sa`.

[terbukti: tiga halaman itu dibuka penulis pada 7 Oktober 2026. Alamat halaman bisa berubah.]

Cara mencari di dokumentasi API PyQGIS:

1. Buka qgis.org/pyqgis. Pilih versi yang sesuai dengan QGIS Anda.
2. Cari nama kelas, misalnya `QgsVectorLayer`.
3. Di halaman kelas, cari nama kemampuan yang Anda temukan lewat `dir()`, misalnya `featureCount`.
4. Baca satu kalimat penjelasannya, dan lihat jenis hasil yang dikembalikan.

> **Samakan versi dokumentasi dengan versi QGIS:** Nama perintah bisa berubah antarversi. Buku ini diuji di QGIS 4.0.2. Halaman qgis.org/pyqgis menyediakan dokumentasi versi master, 4.2, dan 3.44, sementara versi lama tersedia sebagai arsip unduhan. Bila skrip dari internet gagal, cek dulu apakah ia ditulis untuk versi QGIS yang sama.

Langkah 4: Coba satu baris, lalu ubah

Jangan langsung menyusun skrip panjang. Cobalah satu baris di Konsol, lihat hasilnya, lalu ubah satu bagian.

Contoh urutan untuk soal "tampilkan sistem koordinat sebuah layer":

1. **Cari.** Di `dir(QgsVectorLayer)`, cari kata `crs`. Anda menemukan nama `crs`.
2. **Coba.** Ketik `petak.crs()` di Konsol, lalu lihat jenis hasilnya dengan `type(petak.crs())`.
3. **Ubah.** Cari kemampuan objek hasilnya dengan `dir(petak.crs())`. Anda menemukan nama yang berhubungan dengan `authid` dan `description`. Coba `petak.crs().authid()`.

4. **Catat.** Simpan baris yang berhasil ke skrip Anda, lengkap dengan komentar.

Anda menemukan jawabannya tanpa contoh siap pakai. Pada layer `Petak` dari `paket-b2`, hasilnya di komputer penulis:

```
HASIL
EPSG:32749
WGS 84 / UTM zone 49S
```

Baris pertama dari `authid()`, baris kedua dari `description()`. Jenis hasil `petak.crs()` adalah `QgsCoordinateReferenceSystem`. [terbukti: dijalankan di QGIS 4.0.2]

Langkah 5: Membaca kode orang lain dengan tenang

Saat membuka skrip asing, jangan baca dari kiri atas ke kanan bawah dengan panik. Pakai urutan ini:

1. Lihat baris `import`: perkakas apa yang dipakai (Bab 12).
2. Cari **kotak berhuruf besar** di atas: alamat dan pengaturan yang perlu Anda ganti (Bab 8).
3. Cari fungsi `def`: resep apa yang dibuat (Bab 7).
4. Baca perulangan dan `if`: alur utamanya (Bab 6).
5. Cari `print`: apa yang dihasilkan.
6. Jalankan, lalu ubah satu angka dan lihat akibatnya.

Itulah cara pembaca Bagian B akan membaca skrip puluhan baris tanpa tersesat.

Skrip yang dibedah: satu baris, lalu jadi pola

Bab ini tidak punya skrip tetap, karena skripnya adalah **kebiasaan** Anda. Tetapi berikut satu skrip kecil yang mempraktikkan keempat langkah di atas pada data tiruan.

Tujuan. Mengenali kemampuan sebuah objek lewat `dir()`, mencoba salah satunya, lalu mengubah isinya.

Bahan. Tidak ada. Skrip ini hanya memakai Python.

Skrip:

```
PYTHON
nama = "KPH Alpha"
kemampuan = [n for n in dir(nama) if n.startswith("s")]
print(kemampuan[:4])
print(nama.split())
print(nama.lower())
print(nama.replace("Alpha", "Beta"))
```

Penjelasan baris demi baris. Nomor baris dihitung dari skrip di atas.

Baris	Tulisan	Artinya
1	`nama = "KPH Alpha"`	Siapkan sebuah teks sebagai bahan percobaan.
2	`kemampuan = [n for n in dir(nama) if n.startswith("s")]`	**Cari.** Kumpulkan semua kemampuan teks yang namanya berawalan huruf `s`.
3	`print(kemampuan[:4])`	Lihat empat nama pertama.
4	`print(nama.split())`	**Coba.** Pecah teks menjadi daftar kata.
5	`print(nama.lower())`	Ubah semua huruf menjadi kecil.
6	`print(nama.replace("Alpha", "Beta"))`	**Ubah.** Ganti satu kata dengan kata lain.

Hasil.

```
HASIL
['split', 'splitlines', 'startswith', 'strip']
['KPH', 'Alpha']
kph alpha
KPH Beta
```

Dari daftar kemampuan yang berawalan `s`, Anda mencoba `split`, lalu mencoba dua kemampuan lain yang Anda ketahui. Itulah gerak cari, coba, ubah.

Bagian yang boleh diubah.

- Huruf awal di `startswith("s")` (baris 2): ganti dengan huruf lain untuk melihat daftar lain.
- Angka di `[:4]` (baris 3): ganti untuk melihat lebih banyak nama.
- Isi `nama` (baris 1): ganti dengan teks lain.

Jebakan.

- Tidak membaca `help`. Nama kemampuan saja tidak cukup. Baca penjelasannya dengan `help(nama.split)`.
- **Mengira semua nama berguna.** `dir()` menampilkan banyak nama yang jarang Anda butuhkan. Saring dengan kata kunci.
- **Mengabaikan versi.** Hasil `dir()` pada objek QGIS bisa berbeda antarversi.

Latihan variasi. Cari satu kemampuan teks yang menghitung berapa kali sebuah huruf muncul. Petunjuk: lihat daftar `dir(str)` dan cari nama yang diawali `c`.

Contoh jawaban: kemampuan itu bernama `count`.

```
PYTHON
nama = "KPH Alpha"
print(nama.count("a"))
```

```
HASIL
1
```

> **Pola yang bisa dipakai ulang:** Untuk soal baru apa pun: **(1) cari** dengan `dir()` dan dokumentasi, **(2) coba** satu baris di Konsol, **(3) ubah** satu hal dan amati, **(4) catat** yang berhasil di skrip. Ulangi sampai jadi.

Cek paham

1. Apa fungsi `dir()`?
2. Mengapa dokumentasi harus sesuai dengan versi QGIS Anda?
3. Mengapa sebaiknya mencoba satu baris lebih dulu, bukan langsung menulis skrip panjang?

Jawaban.

1. Menampilkan daftar isi dan kemampuan sebuah objek.
2. Karena nama dan cara kerja perintah bisa berubah antarversi.
3. Karena galat lebih mudah dilacak pada satu baris. Skrip panjang membuat penyebab galat sulit ditemukan.

Kesalahan umum

- **Mencari contoh siap pakai, lalu menyerah bila tidak ada.** Perbaikan: pakai siklus cari, coba, ubah, catat.
- **Menyalin dari internet tanpa membaca.** Perbaikan: jalankan baris demi baris di Konsol sebelum menempelkannya ke skrip.
- **Tidak mencatat yang berhasil.** Perbaikan: simpan setiap baris yang berhasil, lengkap dengan komentar mengapa ia dipakai.

Ringkasan dan latihan

- Belajar mandiri adalah siklus **cari, coba, ubah, catat**.
- `type()`, `dir()`, dan `help()` membuat Python menjelaskan dirinya sendiri.
- Dokumentasi resmi PyQGIS dan ArcPy adalah sumber paling dipercaya. Samakan versinya.

Latihan. Pilih satu kemampuan objek `str` yang belum Anda kenal, misalnya `title` atau `center`. Cari tahu fungsinya dengan `help()`, coba di Konsol, lalu tulis satu kalimat tentang apa yang ia lakukan.

Tabel perbandingan: tempat mencari jawaban

Pertanyaan	Alat	Contoh
Apa jenis nilai ini?	`type(x)`	`type(3.5)`
Apa kemampuan objek ini?	`dir(x)`	`dir(petak)`
Apa fungsi perintah ini?	`help(x)`	`help(str.upper)`
Bagaimana cara resmi memakainya?	Dokumentasi	qgis.org/pyqgis

BAGIAN B | PYQGIS

BAB 14: Mengetuk Pintu QGIS: Memuat, Membaca, dan Menyimpan dengan Skrip

> **Berdasarkan Seri I4:** Skrip bab ini berasal dari Buku I4, *Otomasi Dasar dengan Python* (Intermediate Class), berkas `b1_00_python_dasar.py`, `b1_01_muat_layer.py`, `b1_02_baca_atribut.py`, dan `b1_03_simpan_proyek.py`. Di dalam berkasnya tertulis nomor Skrip 1.0 sampai 1.3, mengikuti penomoran buku itu. Di bab ini keempatnya diberi nomor Skrip 14.1 sampai 14.4.

Studi kasus: "Peta saya bisa dibuka lagi besok?"

Kepala Seksi membuka rapat sepuluh menit lagi. Ia bertanya dua hal. Pertama, berapa luas tiap KPH dalam hektare? Kedua, bisakah peta yang Anda susun hari ini dibuka lagi besok pagi, utuh seperti semula? Dengan klik, Anda bisa menjawabnya. Dengan skrip, jawabannya keluar dalam sekali jalan, dan skrip yang sama bisa dipakai untuk KPH lain. Bab ini membuka pintunya: cara memerintah QGIS dari Python.

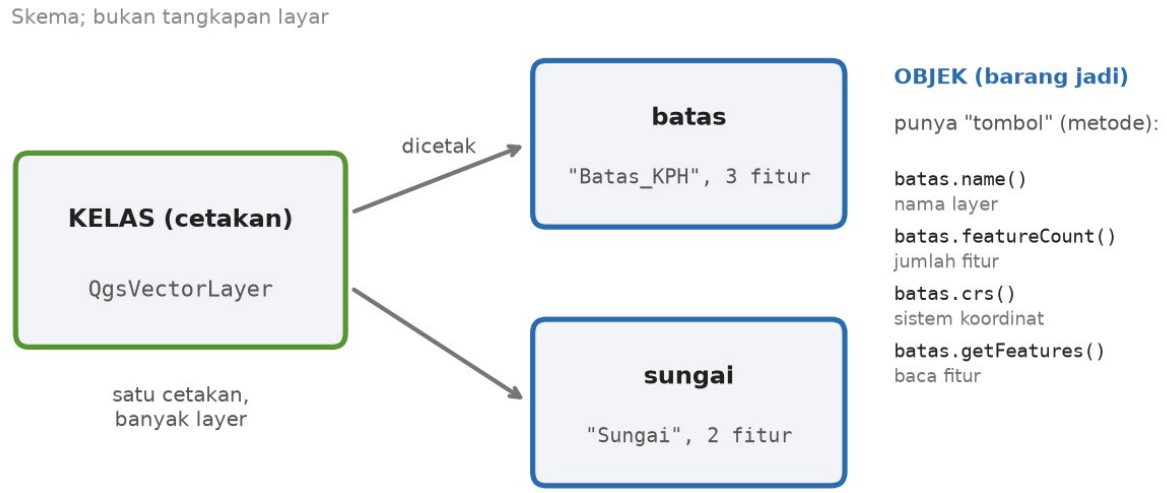
Konsep: PyQGIS dalam tiga kalimat

Bayangkan QGIS sebagai dapur restoran besar yang lengkap. Python adalah daftar pesanan tertulis yang Anda serahkan ke dapur itu. **PyQGIS** adalah buku menu yang menjelaskan pesanan apa saja yang boleh Anda tulis.

Buku menu itu punya nama resmi: **API**, yaitu daftar perintah yang disediakan sebuah program untuk dipanggil dari program lain. Hampir semua yang bisa Anda klik di QGIS punya perintah padanan di API-nya. Karena itu, Bab 13 menyarankan kebiasaan membuka dokumentasi API setiap kali Anda bertanya "perintah untuk ini apa?".

Ada tiga istilah lagi yang akan sering muncul. **Kelas** adalah cetakan, misalnya cetakan "layer vektor". **Objek** adalah barang jadi yang dicetak dari kelas itu, misalnya layer `Batas_KPH` yang sudah Anda buka. **Metode** adalah tombol pada objek itu, ditulis dengan titik dan kurung, misalnya `batas.featureCount()`. Konsep kelas dan objek dibahas lengkap di Bab 11.

Ilustrasi 14.1: Kelas, objek, dan metode



Lima istilah baru Anda sejauh ini:

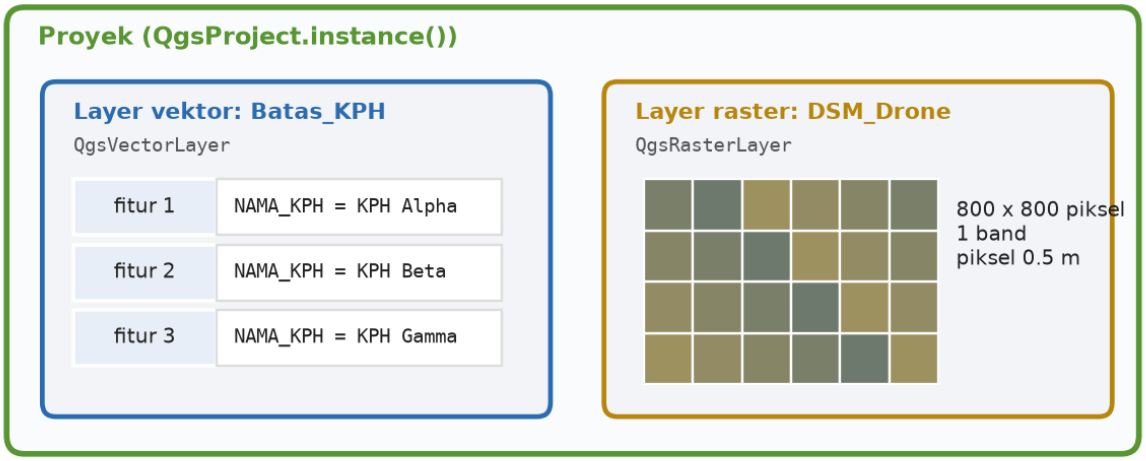
- **PyQGIS**: cara Python berbicara dengan QGIS.
- **API**: daftar perintah resmi sebuah program, seperti buku menu.
- **Kelas**: cetakan, ditulis dengan awalan `Qgs`, misalnya `QgsVectorLayer`.
- **Objek**: hasil cetakan dari sebuah kelas, misalnya `batas`.
- **Metode**: tombol pada objek, ditulis `objek.nama()`.

Konsep: proyek, layer, dan fitur

Sekarang bayangkan sebuah lemari besar bernama **proyek**. Di dalamnya ada rak-rak yang disebut **layer**. Satu layer vektor berisi baris-baris yang disebut **fitur**, sedangkan satu layer raster berisi petak-petak piksel. Skrip PyQGIS membuka lemari itu, mengambil rak yang Anda minta, lalu membaca isinya.

Ilustrasi 14.2: Isi sebuah proyek

Skema; bukan tangkapan layar



Tiga istilah baru di bagian ini:

- **Proyek**: wadah semua layer, dilihat dari Python lewat `QgsProject`.
- **Layer**: satu data di dalam proyek, vektor (`QgsVectorLayer`) atau raster (`QgsRasterLayer`).
- **Fitur**: satu baris di layer vektor, misalnya satu KPH.

Dua cara menjalankan: di dalam QGIS dan mandiri

Skrip PyQGIS bisa dijalankan dengan dua cara. Cara pertama: **di dalam QGIS**. Dapur sudah menyala dan proyek Anda sudah terbuka. Cara kedua: **mandiri**, tanpa membuka QGIS. Di cara ini skrip harus menyalakan dapurnya sendiri.

Ilustrasi 14.3: Skrip di dalam QGIS lawan skrip mandiri



Untuk cara pertama, ikuti langkah di Bab 3. Ringkasnya, menurut Buku I4 Bab 1: pilih **Plugins ▸ Python Console**, klik **Show Editor**, tempel skrip, lalu jalankan dengan Ctrl+Shift+E.

Untuk cara kedua, kerangkanya selalu sama. Anda menyalakan mesin QGIS di awal, menjalankan pekerjaan, lalu mematikannya di akhir. Kerangka ini sudah diuji dengan `python-qgis` pada QGIS 4.0.2.

```python
from qgis.core import QgsApplication

QgsApplication.setPrefixPath(r"C:/Program Files/QGIS 4.0.2/apps/qgis", True)
app = QgsApplication([], False)
app.initQgis()

# ... tempel isi skrip Anda di sini ...

app.exitQgis()
```

Baris	Artinya
`QgsApplication.setPrefixPath(...)`	Beri tahu Python di mana QGIS terpasang. Jalur ini milik komputer penulis. Di komputer Anda mungkin berbeda, dan di macOS atau Ubuntu pasti berbeda [CEK].
`QgsApplication([], False)`	Buat mesin QGIS. Isian kedua `False` berarti aplikasi tanpa layar, hanya konsol. Menurut dokumentasi API, `True` dipakai untuk aplikasi dengan antarmuka.
`app.initQgis()`	Nyalakan mesin: muat pembaca data yang dibutuhkan.
`app.exitQgis()`	Matikan mesin dan bersihkan memori.

Kerangka ini hanya untuk cara kedua. Di Konsol Python QGIS, mesinnya sudah menyala, jadi jangan menempelkannya di sana. Pada uji di bab ini, beberapa skrip kecil tetap jalan tanpa kerangka itu. Tetapi Skrip 14.4 macet pada baris pembukaan proyek. Jadi untuk skrip mandiri, selalu pakai kerangka lengkap.

Skrip 14.1: Lima hal Python sebelum masuk QGIS

Skrip pertama tidak memakai QGIS sama sekali. Ia hanya memanaskan Anda dengan lima hal Python dasar. Karena isinya dibedah di Bab 4 sampai 7, penjelasan di sini sengaja ringkas. Berkas `b1_00_python_dasar.py` berisi 33 baris, dan di dalamnya tertulis "Skrip 1.0".

Tujuan. Mengingat lima hal Python yang dipakai di seluruh bab setelah ini: variabel, daftar, kamus, perulangan dengan percabangan, dan fungsi.

Bahan. Tidak ada data. Skrip bisa dijalankan di Konsol Python QGIS, atau dengan Python biasa di terminal.

Skrip utuh:

```python
# [SKRIP 1.0: Lima hal Python yang cukup untuk memulai]
# Penulis: Badar Mubarok Yogaswara
# Tidak memakai QGIS sama sekali. Boleh dicoba di Python Console baris demi baris.

# 1. Variabel: kotak bernama yang menyimpan nilai
nama = "KPH Alpha"
luas_ha = 4.0
print(nama, "luasnya", luas_ha, "ha")

# 2. Daftar: kumpulan nilai berurutan. Nomor urut dimulai dari 0, bukan 1.
petak = ["KPH Alpha", "KPH Beta", "KPH Gamma"]
print("Petak pertama:", petak[0])
print("Jumlah petak:", len(petak))

# 3. Kamus: pasangan kunci dan nilai, seperti daftar isi dengan nomor halaman
luas = {"KPH Alpha": 4.0, "KPH Beta": 4.0, "KPH Gamma": 8.0}
print("Luas KPH Gamma:", luas["KPH Gamma"])

# 4. Perulangan for dan percabangan if. Baris yang menjorok ke dalam ikut dalam blok.
total = 0
for nama, ha in luas.items():
    total = total + ha
    if ha > 5:
        print(nama, "termasuk petak besar")
print("Total:", total, "ha")
```

```
# 5. Fungsi: resep yang diberi nama supaya bisa dipakai berulang kali
def ke_hektar(meter_persegi):
    return meter_persegi / 10000


print("40000 m2 =", ke_hektar(40000), "ha")
```

Penjelasan baris demi baris. Nomor baris mengikuti berkas di atas. Baris komentar (yang diawali `#`) dilewati Python, jadi tidak ditulis ulang di sini.

Baris	Tulisan	Artinya
6-8	`nama = "KPH Alpha"` dan seterusnya	Isi kotak `nama` dengan teks dan kotak `luas_ha` dengan angka, lalu `print` menampilkan keduanya. Dibahas di Bab 4.
11-13	`petak = [...]`	Daftar berisi tiga teks. `petak[0]` mengambil isi pertama, karena nomor urut mulai dari 0. `len(petak)` menghitung isinya. Dibahas di Bab 5.
16-17	`luas = {...}`	Kamus: pasangan kunci dan nilai. `luas["KPH Gamma"]` mengambil nilai milik kunci itu. Dibahas di Bab 5.
20-25	`for nama, ha in luas.items():`	Ulangi untuk tiap pasangan di kamus. Baris yang menjorok ke dalam ikut diulang. `if ha > 5` hanya mencetak KPH yang luasnya lebih dari 5. Dibahas di Bab 6.
29-30	`def ke_hektar(meter_persegi):`	Resep bernama. Beri angka meter persegi, ia mengembalikan (`return`) hasil bagi 10000. Dibahas di Bab 7.
33	`print("40000 m2 =", ke_hektar(40000), "ha")`	Panggil resep itu dengan angka 40000.

Hasil. Layar menampilkan (berdasarkan uji pada QGIS 4.0.2):

```
HASIL
KPH Alpha luasnya 4.0 ha
Petak pertama: KPH Alpha
Jumlah petak: 3
Luas KPH Gamma: 8.0
KPH Gamma termasuk petak besar
Total: 16.0 ha
40000 m2 = 4.0 ha
```

Hasil ini sama dengan hasil uji di Buku I4.

Bagian yang boleh diubah.

- **Baris 16, isi kamus.** Tambah pasangan baru, misalnya `"KPH Delta": 6.0`.
- **Baris 23, angka 5.** Ini batas "besar". Ganti menjadi `7`, dan hanya KPH Gamma yang lolos.
- **Baris 33, angka 40000.** Beri angka lain ke resep `ke_hektar`.

Jebakan.

- **Nama kotak terpakai ulang.** Pada baris 6, `nama` berisi `"KPH Alpha"`. Pada baris 21, perulangan memakai nama `nama` lagi. Setelah perulangan selesai, isi `nama` menjadi `KPH Gamma`. Hasil uji: perintah `print(nama)` sesudah perulangan menampilkan `KPH Gamma`. Pelajarannya: nama kotak yang sama, isi yang tertimpa.
- **Huruf besar dan kecil.** `print(Nama)` menghasilkan `NameError: name 'Nama' is not defined. Did you mean: 'nama'?`
- **Lupa menjorok.** Bila baris di bawah `for` tidak dijorokkan, muncul `IndentationError: expected an indented block after 'for' statement on line 1.`
- **Nama "petak".** Isi daftar `petak` sebenarnya nama KPH, bukan petak. Itu hanya soal nama. Python tidak peduli.

Latihan variasi. Tambahkan KPH Delta seluas 6.0 hektare ke kamus, lalu jalankan lagi.

Contoh jawaban (sudah diuji), hanya bagian yang berubah:

```python
luas = {"KPH Alpha": 4.0, "KPH Beta": 4.0, "KPH Gamma": 8.0, "KPH Delta": 6.0}
```

Layar menampilkan `KPH Gamma termasuk petak besar`, lalu `KPH Delta termasuk petak besar`, lalu `Total: 22.0 ha`.

> **Pola yang bisa dipakai ulang:** Kamus plus perulangan adalah kerangka "ringkas banyak hal": `total = 0`, lalu `for kunci, nilai in kamus.items():`, lalu `total = total + nilai`. Ganti isi kamus dengan data Anda sendiri.

Skrip 14.2: Memuat layer dan mengambilnya dengan nama

Sekarang Anda masuk ke QGIS. Bayangkan rak buku. Anda meletakkan buku di rak (memuat layer), lalu mengambilnya lagi dengan menyebut judulnya (mengambil layer dengan nama). Rak itu adalah proyek.

Tujuan. Memuat satu layer vektor dan satu raster, memeriksa apakah keduanya terbuka, memasukkannya ke proyek, lalu membaca sifat keduanya.

Bahan. Paket data `paket-i4`, berkas `Batas_KPH.gpkg` (tiga KPH) dan `DSM_Drone.tif` (raster tinggi permukaan). Berkas `b1_01_muat_layer.py` dijalankan dari Editor Skrip QGIS atau dengan kerangka mandiri di atas. Ganti dulu jalur `DATA` di baris 6 dengan lokasi `paket-i4` di komputer Anda.

Skrip ini bekerja dalam enam tahap. Gambar berikut menunjukkan urutannya, lengkap dengan nomor baris.

Ilustrasi 14.4: Alur skrip 14.2



Skrip utuh:

```python
# [SKRIP 1.1: Memuat layer lewat Python dan mengambilnya kembali dengan nama]
# Penulis: Badar Mubarok Yogaswara
# Jalankan di Plugins > Python Console. Ganti DATA dengan folder paket-i4 di komputer Anda.
from qgis.core import QgsProject, QgsVectorLayer, QgsRasterLayer, QgsWkbTypes

DATA = "D:/Latihan/paket-i4"
HASIL = "D:/Latihan/hasil_i4"

proyek = QgsProject.instance()          # proyek yang sedang terbuka
proyek.removeAllMapLayers()             # kosongkan dulu agar tidak dobel

# 1. Muat satu layer vektor dan satu raster. Argumen: jalur, nama di panel Layers, penyedia data.
batas = QgsVectorLayer(DATA + "/Batas_KPH.gpkg", "Batas_KPH", "ogr")
dsm = QgsRasterLayer(DATA + "/DSM_Drone.tif", "DSM_Drone", "gdal")

# 2. Periksa dulu: layer yang gagal dibuka tidak memicu galat, hanya isValid() bernilai False.
print("Batas_KPH sah?", batas.isValid())
print("DSM_Drone sah?", dsm.isValid())

# 3. Masukkan ke proyek supaya tampil di panel Layers.
proyek.addMapLayer(batas)
proyek.addMapLayer(dsm)

# 4. Ambil layer dengan namanya. mapLayersByName() mengembalikan DAFTAR, ambil isi pertama.
layer = proyek.mapLayersByName("Batas_KPH")[0]
print("Nama:", layer.name())
print("CRS:", layer.crs().authid())
print("Jenis geometri:", QgsWkbTypes.displayString(layer.wkbType()))
print("Jumlah fitur:", layer.featureCount())
print("Nama field:", [f.name() for f in layer.fields()])

# 5. Sifat sebuah raster
r = proyek.mapLayersByName("DSM_Drone")[0]
```

```
print("Raster:", r.width(), "x", r.height(), "piksel,", r.bandCount(), "band")
print("Ukuran piksel (m):", r.rasterUnitsPerPixelX())
print("Jumlah layer di proyek:", len(proyek.mapLayers()))
```

Penjelasan baris demi baris. Skrip ini padat, jadi dipecah menjadi tiga blok. Nomor baris mengikuti berkas di atas.

Blok A: menyiapkan dan membuka (baris 4-14).

Baris	Tulisan	Artinya
4	`from qgis.core import ...`	Panggil empat perkakas dari QGIS: `QgsProject`, `QgsVectorLayer`, `QgsRasterLayer`, dan `QgsWkbTypes`. Cara kerja `import` dibahas di Bab 8.
6-7	`DATA = ...` dan `HASIL = ...`	Dua kotak berisi jalur folder. Hanya `DATA` yang dipakai di skrip ini. `HASIL` sisa dari kerangka yang sama dengan skrip lain.
9	`proyek = QgsProject.instance()`	Ambil proyek yang sedang terbuka dan simpan di kotak `proyek`. Hanya ada satu proyek aktif, jadi `instance()` selalu mengembalikan objek yang sama.
10	`proyek.removeAllMapLayers()`	Keluarkan semua layer dari proyek, supaya tidak dobel.
13	`batas = QgsVectorLayer(...)`	Buka layer vektor. Tiga isian: sumber data (folder disambung + dengan nama berkas), nama layer di panel Layers, dan pembaca data `"ogr"` untuk vektor.
14	`dsm = QgsRasterLayer(...)`	Sama, untuk raster. Pembaca datanya `"gdal"`.

Blok B: memeriksa dan memasukkan (baris 16-22).

Baris	Tulisan	Artinya
17-18	`print("Batas_KPH sah?", batas.isValid())`	Tanya: apakah layer ini terbuka dengan benar? Jawabannya `True` atau `False`.
21-22	`proyek.addMapLayer(batas)`	Masukkan layer ke proyek supaya tampil di panel Layers. Sebelum baris ini, layer hanya ada di memori skrip.

Blok C: mengambil dan membaca sifat (baris 24-36).

Baris	Tulisan	Artinya
25	`proyek.mapLayersByName("Batas_KPH")[0]`	Cari layer bernama `Batas_KPH`. Hasilnya **daftar**, karena bisa ada beberapa layer bernama sama. `[0]` mengambil yang pertama.
26	`layer.name()`	Metode yang mengembalikan nama layer.
27	`layer.crs().authid()`	`crs()` memberi sistem koordinat layer. `authid()` memberi kodenya, misalnya `EPSG:32749` (UTM zona 49S).
28	`QgsWkbTypes.displayString(layer.wkbType())`	Jenis bentuk layer dalam kata, misalnya `Polygon`.
29	`layer.featureCount()`	Jumlah fitur.

Baris	Tulisan	Artinya
30	`[f.name() for f in layer.fields()]`	Daftar nama kolom. Cara menulis ini disebut *list comprehension*, dibahas di Bab 6.
33	`r = proyek.mapLayersByName("DSM_Drone")[0]`	Ambil raster dengan cara yang sama.
34-35	`r.width(), r.height(), r.bandCount(), r.rasterUnitsPerPixelX()`	Lebar dan tinggi dalam piksel, jumlah band, dan ukuran satu piksel dalam satuan peta.
36	`len(proyek.mapLayers())`	Jumlah layer di proyek.

Hasil. Layar menampilkan (berdasarkan uji pada QGIS 4.0.2):

```
HASIL
Batas_KPH sah? True
DSM_Drone sah? True
Nama: Batas_KPH
CRS: EPSG:32749
Jenis geometri: Polygon
Jumlah fitur: 3
Nama field: ['fid', 'NAMA_KPH']
Raster: 800 x 800 piksel, 1 band
Ukuran piksel (m): 0.5
Jumlah layer di proyek: 2
```

Dari hasil ini Anda tahu: layer batas berisi tiga KPH berbentuk poligon, dan rasternya 800 kali 800 piksel dengan piksel 0,5 meter.

Bagian yang boleh diubah.

- **Baris 6, lokasi data.** Yang paling sering Anda ubah. Pakai garis miring biasa `/`, misalnya `"D:/Latihan/paket-i4"`.
- **Baris 13-14, nama berkas.** Ganti untuk membuka data lain, misalnya `/Sungai.gpkg`.
- **Argumen kedua di baris 13.** Ini nama di panel Layers. Bila Anda menggantinya, ganti juga isi `mapLayersByName` di baris 25 dan 33.
- **Baris 25-30.** Tambah baris `print` baru dengan metode lain, misalnya `layer.source()`, yang menampilkan lokasi berkas layer.

Jebakan.

- **Jalur salah tidak menimbulkan galat.** Pada uji, `Batas_KPH.gpkg` diganti menjadi `Batas_KP.gpkg`. Python tidak berhenti. Layar menampilkan:

```
HASIL
Batas_KPH sah? False
DSM_Drone sah? True
Nama: Batas_KPH
CRS:
Jenis geometri: Unknown
Jumlah fitur: -2
Nama field: []
```

```
Raster: 800 x 800 piksel, 1 band
Ukuran piksel (m): 0.5
Jumlah layer di proyek: 2
```

Perhatikan: layer yang tidak terbuka tetap masuk proyek (jumlah layer 2). Jumlah fitur -2 artinya "tidak diketahui". Pelajarannya: **selalu baca baris pertama dan kedua hasil**. Bila ada `False`, periksa jalur di baris 6 dulu.

- **Mengambil layer yang tidak ada.** `proyek.mapLayersByName("Tidak_Ada")` menghasilkan daftar kosong `[]`. Bila Anda lalu menulis `[0]`, muncul `IndexError: list index out of range`. Itu artinya nama yang Anda cari tidak ada di proyek.
- **Baris 10 mengosongkan proyek Anda.** `removeAllMapLayers()` mengeluarkan semua layer dari proyek yang sedang terbuka di QGIS. Jalankan skrip ini di proyek kosong, bukan di proyek kerja yang penting.
- `QgsWkbTypes` (baris 4 dan 28). Skrip berjalan tanpa galat di QGIS 4.0.2. Apakah nama ini sudah dinyatakan usang di versi tertentu, belum diperiksa [CEK].

Latihan variasi.

1. Muat layer `Sungai.gpkg` (garis), lalu cetak jenis bentuk, jumlah fitur, dan nama kolomnya.
2. Dari hasil nomor 1, sebutkan apa beda jenis bentuknya dengan `Batas_KPH`.

Contoh jawaban (sudah diuji), ditulis sebagai skrip kecil sendiri:

```python
from qgis.core import QgsProject, QgsVectorLayer, QgsWkbTypes

DATA = "D:/Latihan/paket-i4"
proyek = QgsProject.instance()
sungai = QgsVectorLayer(DATA + "/Sungai.gpkg", "Sungai", "ogr")
proyek.addMapLayer(sungai)
layer = proyek.mapLayersByName("Sungai")[0]
print("Jenis geometri:", QgsWkbTypes.displayString(layer.wkbType()))
print("Jumlah fitur:", layer.featureCount())
print("Nama field:", [f.name() for f in layer.fields()])
```

Layar menampilkan `Jenis geometri: LineString`, `Jumlah fitur: 2`, dan `Nama field: ['fid', 'NAMA', 'ORDE']`. Sungai berbentuk garis, sedangkan batas KPH berbentuk poligon.

> **Pola yang bisa dipakai ulang:** Kerangka "muat, cek, tambah, ambil": `layer = QgsVectorLayer(DATA + "/BERKAS.gpkg", "NAMA", "ogr")`, lalu `print(layer.isValid())`, lalu `QgsProject.instance().addMapLayer(layer)`, lalu `QgsProject.instance().mapLayersByName("NAMA")[0]`. Untuk raster, ganti `QgsVectorLayer` dengan `QgsRasterLayer` dan `"ogr"` dengan `"gdal"`.

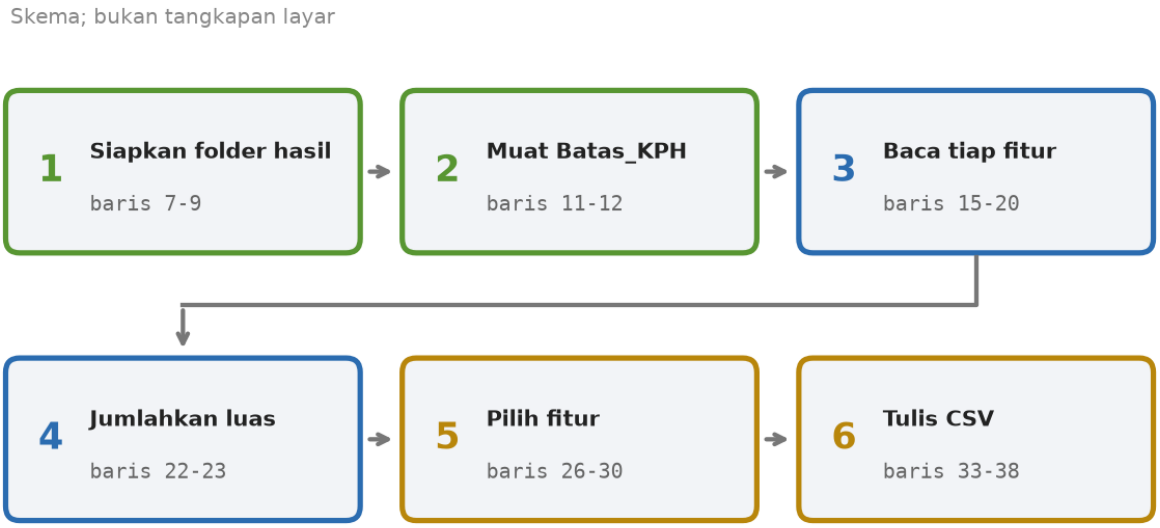
Skrip 14.3: Membaca fitur dan atribut, lalu menulis CSV

Layer vektor itu seperti tabel. Tiap baris adalah satu fitur. Skrip ini membaca tiap baris, menghitung luas, memilih satu KPH dengan ekspresi, lalu menulis hasilnya ke berkas CSV. Ini jawaban untuk pertanyaan pertama Kepala Seksi.

Tujuan. Menghitung luas tiap KPH dalam hektare, memilih KPH tertentu dengan ekspresi, dan menyimpan hasilnya sebagai berkas CSV.

Bahan. Paket data `paket-i4`, berkas `Batas_KPH.gpkg`. Berkas `b1_02_baca_atribut.py` dijalankan seperti Skrip 14.2. Ganti `DATA` di baris 7 dan `HASIL` di baris 8 (folder tempat CSV ditulis; skrip membuatnya bila belum ada).

Ilustrasi 14.5: Alur skrip 14.3



Skrip utuh:

```
PYTHON
# [SKRIP 1.2: Membaca fitur dan atribut, memilih dengan ekspresi, menulis CSV]
# Penulis: Badar Mubarok Yogaswara
import csv
import os
from qgis.core import QgsProject, QgsVectorLayer

DATA = "D:/Latihan/paket-i4"
HASIL = "D:/Latihan/hasil_i4"
os.makedirs(HASIL, exist_ok=True)

batas = QgsVectorLayer(DATA + "/Batas_KPH.gpkg", "Batas_KPH", "ogr")
QgsProject.instance().addMapLayer(batas)

# 1. Satu fitur = satu baris tabel atribut. getFeatures() membacanya satu per satu.
baris = []
for fitur in batas.getFeatures():
    nama = fitur["NAMA_KPH"]
    luas_ha = fitur.geometry().area() / 10000      # m2 menjadi hektare (CRS harus berupa meter)
    baris.append((nama, round(luas_ha, 2)))
    print(nama, round(luas_ha, 2), "ha")

total = sum(luas for _, luas in baris)
print("Total:", total, "ha")

# 2. Memilih fitur dengan ekspresi, sama seperti Select by Expression
batas.selectByExpression('"NAMA_KPH" = \'KPH Beta\'')
print("Terpilih:", batas.selectedFeatureCount(), "fitur")
for fitur in batas.selectedFeatures():
```

```
    print("  ->", fitur["NAMA_KPH"])
batas.removeSelection()

# 3. Menulis hasil ke berkas CSV
jalur = HASIL + "/luas_kph.csv"
with open(jalur, "w", newline="", encoding="utf-8") as f:
    penulis = csv.writer(f)
    penulis.writerow(["NAMA_KPH", "LUAS_HA"])
    penulis.writerows(baris)
print("CSV tersimpan:", os.path.exists(jalur))
```

Penjelasan baris demi baris. Tiga blok, sesuai komentar di dalam skrip.

Blok A: persiapan (baris 3-12).

Baris	Tulisan	Artinya
3-5	`import csv`, `import os`, `from qgis.core ...`	Panggil perkakas: `csv` untuk berkas CSV, `os` untuk urusan folder, dan dua perkakas QGIS.
9	`os.makedirs(HASIL, exist_ok=True)`	Buat folder `HASIL`. `exist_ok=True` berarti: bila sudah ada, jangan protes.
11-12	`batas = QgsVectorLayer(...)` lalu `addMapLayer`	Buka layer dan masukkan ke proyek, seperti di Skrip 14.2.

Blok B: membaca tiap fitur (baris 14-23).

Baris	Tulisan	Artinya
15	`baris = []`	Siapkan daftar kosong untuk menampung hasil.
16	`for fitur in batas.getFeatures():`	Ulangi untuk tiap fitur. `getFeatures()` membacanya satu per satu. Di tiap putaran, kotak `fitur` berisi satu KPH.
17	`nama = fitur["NAMA_KPH"]`	Ambil isi kolom `NAMA_KPH`. Nama kolom ditulis di kurung siku dan kutip.
18	`luas_ha = fitur.geometry().area() / 10000`	`geometry()` mengambil bentuk fitur, `area()` menghitung luasnya (meter persegi bila CRS berupa meter), dan `/ 10000` mengubahnya menjadi hektare.
19	`baris.append((nama, round(luas_ha, 2)))`	Tambahkan sepasang nilai (nama dan luas, dibulatkan dua desimal) ke daftar. Pasangan seperti ini disebut *tuple*.
20	`print(nama, round(luas_ha, 2), "ha")`	Tampilkan satu baris per KPH.
22-23	`total = sum(luas for _, luas in baris)`	Jumlahkan semua luas. Tanda `_` berarti "bagian ini tidak dipakai", yaitu nama.

Blok C: memilih dan menulis (baris 25-38).

Baris	Tulisan	Artinya
26	`batas.selectByExpression('"NAMA_KPH" = \'KPH Beta\'')`	Pilih fitur dengan ekspresi, sama seperti **Select by Expression** di QGIS. Teks ekspresi memakai kutip tunggal, jadi kutip tunggal di dalamnya ditulis `\'`.
27-29	`selectedFeatureCount()` dan `selectedFeatures()`	Hitung dan baca fitur yang terpilih.
30	`batas.removeSelection()`	Hapus pilihan, supaya layer kembali bersih.
33-34	`with open(jalur, "w", ...) as f:`	Buka berkas untuk ditulis (`"w"`). `with` menutup berkas otomatis. `encoding="utf-8"` mengatur kode huruf. Dibahas di Bab 9.
35-37	`csv.writer(f)`, `writerow`, `writerows`	Tulis satu baris judul, lalu semua baris data sekaligus.
38	`os.path.exists(jalur)`	Buktikan berkasnya benar-benar ada.

Hasil. Layar menampilkan (berdasarkan uji pada QGIS 4.0.2):

```
HASIL
KPH Alpha 4.0 ha
KPH Beta 4.0 ha
KPH Gamma 8.0 ha
Total: 16.0 ha
Terpilih: 1 fitur
  -> KPH Beta
CSV tersimpan: True
```

Berkas `luas_kph.csv` di folder `HASIL` berisi:

```
HASIL
NAMA_KPH,LUAS_HA
KPH Alpha,4.0
KPH Beta,4.0
KPH Gamma,8.0
```

Bagian yang boleh diubah.

- **Baris 7-8, jalur.** Sesuaikan dengan komputer Anda.
- **Baris 26, nama KPH.** Ganti `KPH Beta` dengan KPH lain. Baris ini juga bisa diganti ekspresi lain, misalnya `'"NAMA_KPH" LIKE \'KPH %\''` [CEK].
- **Baris 18, pembagi 10000.** Mengubah satuan. Hapus pembagi itu dan Anda mendapat meter persegi.
- **Baris 19 dan 20, angka 2.** Jumlah desimal.
- **Baris 33, nama berkas CSV.** Ganti sesuai kebutuhan.

Jebakan.

- **Nama kolom salah.** Pada uji, `fitur["NAMA_KPH"]` diganti menjadi `fitur["NAMA"]`. Layar menampilkan `KeyError: 'NAMA'`. Artinya kolom itu tidak ada. Cek ejaan di daftar `Nama field` pada hasil Skrip 14.2.
- **CRS berderajat.** Komentar di baris 18 mengingatkan bahwa CRS harus berupa meter. Pada uji, `Batas_KPH` diubah ke EPSG:4326 (derajat), lalu skrip dijalankan. Hasilnya `KPH Alpha 0.0 ha`, `KPH Beta 0.0 ha`, dan `KPH Gamma 0.0 ha`, tanpa galat apa pun. Angka luas tampak wajar, padahal salah. Pastikan CRS-nya metrik sebelum menghitung luas.
- **Satuan di tulisan.** Skrip mencetak "ha" di baris 20 walau Anda sudah menghapus `/ 10000`. Tulisan tidak ikut berubah otomatis.
- **Excel di Windows.** Bila nanti nama KPH memuat huruf non-ASCII, Excel mungkin menampilkannya rusak. Cara umumnya adalah `encoding="utf-8-sig"` [CEK]. Data latihan ini hanya berisi huruf biasa, jadi tidak terdampak.

Latihan variasi.

1. Pilih KPH Gamma, bukan KPH Beta.
2. Tampilkan luas dalam meter persegi, lengkap dengan tulisan "m2".

Contoh jawaban (sudah diuji). Untuk nomor 1, ganti baris 26 menjadi:

```
PYTHON
batas.selectByExpression('"NAMA_KPH" = \'KPH Gamma\'')
```

Layar menampilkan `-> KPH Gamma`. Untuk nomor 2, ubah empat baris. Nama variabelnya ikut diganti menjadi `luas_m2` karena isinya kini meter persegi:

```
PYTHON
    luas_m2 = fitur.geometry().area()
    baris.append((nama, round(luas_m2, 2)))
    print(nama, round(luas_m2, 2), "m2")
print("Total:", total, "m2")
```

Layar menampilkan `KPH Alpha 40000.0 m2`, `KPH Beta 40000.0 m2`, `KPH Gamma 80000.0 m2`, dan `Total: 160000.0 m2`. Kolom `LUAS_HA` di CSV juga ikut berisi angka meter persegi, jadi sebaiknya ganti juga judul kolomnya.

> **Pola yang bisa dipakai ulang:** Kerangka "baca tiap fitur lalu simpan": `for fitur in LAYER.getFeatures():`, lalu ambil nilai `fitur["KOLOM"]`, lalu `baris.append((...))`, dan terakhir tulis dengan `csv.writer`. Ganti `LAYER`, nama kolom, dan rumus di tengahnya.

Skrip 14.4: Menyimpan proyek dan membukanya lagi

Sekarang pertanyaan kedua Kepala Seksi: bisakah peta dibuka lagi besok? Skrip ini menyusun proyek dari nol, menyimpannya sebagai berkas `.qgz`, mengosongkannya, lalu membukanya kembali sebagai bukti isinya utuh.

Tujuan. Menyusun proyek berisi empat layer, mengatur CRS dan judulnya, menyimpannya, lalu membuktikan bahwa berkas itu bisa dibuka lagi.

Bahan. Paket data `paket-i4`: `Batas_KPH.gpkg`, `Sungai.gpkg`, `Hotspot_Nasional.gpkg`, dan `Citra_KPH.tif`. Berkas `b1_03_simpan_proyek.py` dijalankan seperti skrip sebelumnya. Ganti `DATA` dan `HASIL` di baris 6-7.

Ilustrasi 14.6: Alur skrip 14.4



Skrip utuh:

```python
# [SKRIP 1.3: Menyusun proyek lewat Python lalu menyimpan dan membukanya lagi]
# Penulis: Badar Mubarok Yogaswara
import os
from qgis.core import QgsProject, QgsVectorLayer, QgsRasterLayer, QgsCoordinateReferenceSystem

DATA = "D:/Latihan/paket-i4"
HASIL = "D:/Latihan/hasil_i4"
os.makedirs(HASIL, exist_ok=True)

proyek = QgsProject.instance()
proyek.removeAllMapLayers()
proyek.setCrs(QgsCoordinateReferenceSystem("EPSG:32749"))     # CRS proyek: UTM zona 49S
proyek.setTitle("KPH Contoh - proyek skrip")

daftar = [
    QgsVectorLayer(DATA + "/Batas_KPH.gpkg", "Batas_KPH", "ogr"),
    QgsVectorLayer(DATA + "/Sungai.gpkg", "Sungai", "ogr"),
    QgsVectorLayer(DATA + "/Hotspot_Nasional.gpkg", "Hotspot_Nasional", "ogr"),
    QgsRasterLayer(DATA + "/Citra_KPH.tif", "Citra_KPH", "gdal"),
]
for layer in daftar:
    if layer.isValid():
        proyek.addMapLayer(layer)
    else:
        print("GAGAL dimuat:", layer.name())

berkas = HASIL + "/Proyek_KPH.qgz"
```

```
print("Tersimpan?", proyek.write(berkas))          # write() mengembalikan True bila berhasil

# Buka kembali dan buktikan isinya utuh
proyek.clear()
print("Setelah dikosongkan:", len(proyek.mapLayers()), "layer")
print("Terbuka?", proyek.read(berkas))
print("Judul:", proyek.title(), "| CRS:", proyek.crs().authid())
print("Layer:", sorted(l.name() for l in proyek.mapLayers().values()))
```

Penjelasan baris demi baris. Hanya hal yang baru yang dijelaskan. Baris `import`, `DATA`, dan `HASIL` sudah dibahas di Skrip 14.2 dan 14.3.

Baris	Tulisan	Artinya
11	`proyek.removeAllMapLayers()`	Kosongkan proyek, seperti di Skrip 14.2.
12	`proyek.setCrs(QgsCoordinateReferenceSystem("EPSG:32749"))`	Atur CRS proyek ke UTM zona 49S. Ini mengatur proyek, bukan mengubah data tiap layer.
13	`proyek.setTitle("KPH Contoh - proyek skrip")`	Beri judul proyek.
15-20	`daftar = [ ... ]`	Daftar berisi empat layer: tiga vektor dan satu raster. Daftar boleh berisi objek, bukan hanya teks atau angka.
21-25	`for layer in daftar:` lalu `if layer.isValid():`	Untuk tiap layer: bila terbuka, masukkan ke proyek (baris 23). Bila tidak, cetak `GAGAL dimuat` beserta namanya (baris 25).
27-28	`proyek.write(berkas)`	Simpan proyek ke berkas `.qgz`. Metode ini mengembalikan `True` bila berhasil.
31	`proyek.clear()`	Kosongkan seluruh proyek, termasuk judul dan layer.
32	`len(proyek.mapLayers())`	Buktikan proyek kosong: hasilnya 0.
33	`proyek.read(berkas)`	Buka kembali berkas yang tadi disimpan.
34-35	`proyek.title()`, `proyek.crs().authid()`, `sorted(...)`	Cetak judul, kode CRS, dan nama semua layer yang terbaca, diurutkan menurut abjad. `.values()` mengambil isi kamus layer.

Hasil. Layar menampilkan (berdasarkan uji pada QGIS 4.0.2):

```
HASIL
Tersimpan? True
Setelah dikosongkan: 0 layer
Terbuka? True
Judul: KPH Contoh - proyek skrip | CRS: EPSG:32749
Layer: ['Batas_KPH', 'Citra_KPH', 'Hotspot_Nasional', 'Sungai']
```

Berkas `Proyek_KPH.qgz` kini ada di folder `HASIL`. Berkas ini sebenarnya paket terkompresi (zip) yang berisi berkas proyek `.qgs`. Hasil pemeriksaan: isinya `Proyek_KPH.qgs` dan satu berkas gaya.

Bagian yang boleh diubah.

- **Baris 13, judul.** Ganti sesuai proyek Anda.

- **Baris 12, kode CRS.** Ganti, misalnya, ke `"EPSG:4326"`. Yang berubah hanya CRS proyek, bukan data.
- **Baris 15-20, isi daftar.** Tambah atau hapus layer.
- **Baris 27, nama berkas.** Ganti `Proyek_KPH.qgz`.

Jebakan.

- **Jalur layer salah hanya dilaporkan.** Pada uji, `Sungai.gpkg` diganti menjadi `Sungay.gpkg`. Layar menampilkan `GAGAL dimuat: Sungai`, tetapi `Tersimpan?` tetap `True`. Proyek tersimpan tanpa layer Sungai, dan daftar layer di baris terakhir hanya berisi tiga nama. Selalu baca baris `GAGAL dimuat`.
- **Baris 31 mengosongkan proyek Anda.** Seperti baris 11, `clear()` bekerja pada proyek yang sedang terbuka di QGIS. Jangan jalankan skrip ini di proyek kerja yang belum Anda simpan.
- **Skrip mandiri tanpa kerangka.** Pada uji, skrip ini dijalankan dengan `python-qgis` tanpa kerangka dari bagian "Dua cara menjalankan". Layar berhenti setelah `Setelah dikosongkan: 0 layer`, dan tidak selesai sampai dihentikan paksa setelah 60 detik. Dengan kerangka lengkap, skrip selesai normal.

Latihan variasi. Tambahkan layer `Poligon_Sawah.gpkg` ke proyek, dan simpan dengan nama `Proyek_KPH_sawah.qgz`.

Contoh jawaban (sudah diuji). Tambahkan satu baris di dalam daftar, dan ganti satu nama berkas:

```python
    QgsVectorLayer(DATA + "/Poligon_Sawah.gpkg", "Poligon_Sawah", "ogr"),
```

Pada baris 27, ganti `Proyek_KPH.qgz` menjadi `Proyek_KPH_sawah.qgz`. Baris terakhir hasil menjadi `Layer: ['Batas_KPH', 'Citra_KPH', 'Hotspot_Nasional', 'Poligon_Sawah', 'Sungai'].`

> **Pola yang bisa dipakai ulang:** Kerangka "susun, simpan, buktikan": atur proyek (`setCrs`, `setTitle`), muat layer dalam `daftar` dengan `if layer.isValid():`, lalu `proyek.write(BERKAS)`. Untuk membuktikan, `proyek.clear()` dan `proyek.read(BERKAS)`. Ganti daftar layer dan nama berkas.

Cek paham

1. Apa beda "kelas" dan "objek"? Beri satu contoh dari bab ini.
2. Layer yang jalurnya salah tidak membuat skrip berhenti. Bagaimana Anda tahu layer itu gagal dibuka?
3. Mengapa skrip mandiri perlu `QgsApplication` dan `initQgis()`, sedangkan skrip di Konsol Python QGIS tidak?

Jawaban.

1. Kelas adalah cetakan, objek adalah hasil cetakannya. Contoh: `QgsVectorLayer` adalah kelas, dan `batas` adalah objek dari kelas itu.

2. Panggil `isValid()`. Nilai `False` berarti gagal. Tanda lain: jumlah fitur bernilai -2 dan daftar kolom kosong.
3. Di Konsol, QGIS sudah menyala. Skrip mandiri tidak punya mesin QGIS, sehingga harus menyalakannya sendiri lalu mematikannya di akhir.

Kesalahan umum

- **Membiarkan jalur contoh.** `D:/Latihan/paket-i4` adalah jalur milik penulis. Perbaikan: ganti `DATA` dan `HASIL` sesuai komputer Anda.
- Tidak membaca baris `isValid()`. Skrip tetap jalan dengan layer kosong dan hasil yang menyesatkan, seperti luas 0.0. Perbaikan: biasakan memeriksa `isValid()` dan membaca baris pertama hasil.
- **Menghitung luas pada CRS derajat.** Hasilnya salah tanpa peringatan. Perbaikan: pastikan CRS berupa meter, misalnya UTM.
- **Menjalankan skrip pembersih di proyek kerja.** `removeAllMapLayers()` dan `clear()` mengosongkan proyek yang sedang terbuka. Perbaikan: pakai proyek kosong untuk latihan.

Ringkasan dan latihan

- **PyQGIS** adalah cara Python berbicara dengan QGIS lewat **API**. Isinya **kelas** (cetakan), **objek** (hasil cetakan), dan **metode** (tombol).
- Proyek berisi layer. Layer vektor berisi fitur, layer raster berisi piksel.
- Skrip bisa jalan di dalam QGIS atau mandiri. Skrip mandiri harus menyalakan dan mematikan mesin QGIS sendiri.
- Pola dasar: muat layer, cek `isValid()`, baca fitur, simpan hasil.
- `proyek.write()` menyimpan, `proyek.read()` membuka lagi. `clear()` mengosongkan proyek yang terbuka.

Latihan. Gabungkan tiga skrip bab ini. Tulis skrip yang memuat `Batas_KPH.gpkg` dan `Hotspot_Nasional.gpkg`, mencetak jumlah fitur keduanya, lalu menyimpan proyek bernama `Proyek_Latihan.qgz`. Bagian mana dari Skrip 14.2 dan 14.4 yang Anda pinjam?

Tabel perbandingan: dari layar ke skrip

Di layar QGIS	Di skrip PyQGIS	Contoh di bab ini
Plugins ▶ Python Console	Tempat mengetik skrip	Skrip 14.2 sampai 14.4
Panel Layers	`QgsProject.instance().mapLayers()`	Jumlah layer di proyek: 2
Satu layer vektor	`QgsVectorLayer`	`batas`
Satu layer raster	`QgsRasterLayer`	`dsm`
Satu baris tabel atribut	`fitur` dari `getFeatures()`	`fitur["NAMA_KPH"]`
Select by Expression	`selectByExpression`	Memilih KPH Beta
Menyimpan proyek	`proyek.write(berkas)`	`Proyek_KPH.qgz`

BAB 15: Lemari Data yang Patuh: Atribut, Seleksi, Digitasi, dan Laporan dengan Skrip

> **Berdasarkan Seri B2:** Skrip bab ini berasal dari Buku B2, *Data Vektor: Atribut, Digitasi, Seleksi* (Basic Class), berkas `b2_01_atribut.py` sampai `b2_07_laporan.py`. Di buku itu, tiap topik dikerjakan lewat menu dan klik. Bab ini menunjukkan sisi skripnya, dan ketujuh skrip dibahas apa adanya.

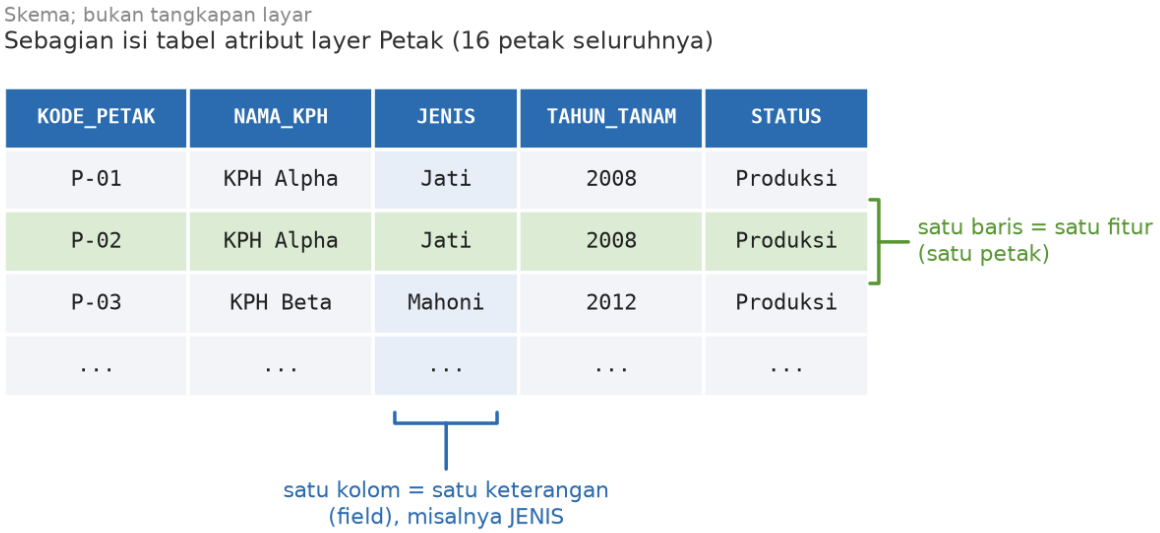
Studi kasus: "Ada berapa petak, dan jenis apa saja?"

Kepala Seksi ingin ringkasan cepat: berapa petak di KPH Contoh, dan tanaman apa saja yang tumbuh di sana? Anda bisa membuka tabel atribut lalu menghitung sendiri. Dengan skrip, jawabannya muncul dalam sekali jalan. Pekan depan permintaannya bertambah: pilih jati yang sudah tua, gambar jalan baru, gabungkan hasil inventarisasi, lalu buat rekap. Tujuh skrip di bab ini menjawab semuanya, dan Anda tahu bagian mana yang diganti untuk data lain.

Konsep: tabel atribut dalam tiga kalimat

Setiap layer vektor punya tabel di belakangnya, seperti lemari arsip. Tiap laci adalah satu **fitur** (satu petak, satu titik, atau satu jalan di peta), dan tiap lembar keterangan di dalam laci adalah satu **kolom**. Skrip PyQGIS membuka lemari itu, membaca laci satu per satu, lalu mengambil kolom yang Anda minta.

Ilustrasi 15.1: Tabel atribut: baris dan kolom



Empat istilah yang perlu Anda kenal sekarang:

- **Fitur** (*feature*): satu objek di layer, misalnya satu petak.
- **Kolom** (*field*): satu jenis keterangan, misalnya `JENIS` atau `LUAS_LAMA_HA`.
- `QgsVectorLayer`: perkakas QGIS yang membuka sebuah layer vektor dari berkas.

- `import`: perintah untuk memanggil perkakas yang tidak ikut otomatis. Cara kerjanya dibahas lengkap di Bab 8.

Setelah bisa membaca lemari, Anda akan memerintahnya. Enam istilah berikut muncul bertahap di skrip berikutnya, dan masing-masing dijelaskan di tempatnya:

- **Seleksi**: menandai sebagian fitur, seperti memberi stabilo pada beberapa baris.
- **Digitasi**: menambah atau mengubah gambar fitur, di sini lewat koordinat yang diketik.
- **Topologi**: aturan "rapi" antar bentuk, misalnya petak tidak boleh tumpang tindih.
- **Kalkulasi kolom**: mengisi kolom baru dengan rumus.
- **Join**: menempelkan kolom dari tabel lain berdasarkan kolom kunci yang sama.
- **Rekap**: ringkasan akhir untuk dibaca atasan.

Peta jalan bab ini:

Skrip	Berkas	Pekerjaannya
15.1	`b2_01_atribut.py`	Membaca tabel atribut
15.2	`b2_02_seleksi.py`	Memilih fitur dengan ekspresi dan lokasi
15.3	`b2_03_digitasi.py`	Menggambar jalan dan persemaian, membelah petak
15.4	`b2_04_topologi.py`	Memeriksa dan memperbaiki geometri
15.5	`b2_05_kalkulasi.py`	Menambah kolom hitungan
15.6	`b2_06_join.py`	Menggabungkan tabel inventarisasi
15.7	`b2_07_laporan.py`	Membuat rekap dan mengekspor

Skrip 15.1 sampai 15.4 berdiri sendiri. Skrip 15.5, 15.6, dan 15.7 harus dijalankan berurutan, karena hasil yang satu menjadi bahan yang berikutnya.

Satu catatan jalur sebelum mulai. Ketujuh skrip membaca lokasi data dari pengaturan `B2_DATA`, dan skrip yang menulis berkas membaca `B2_KELUAR`. Bila pengaturan itu tidak ada, jalur cadangan `C:/KPH_Contoh/paket-b2` dan `C:/KPH_Contoh/keluaran` yang dipakai. Anda bisa membuat folder itu, atau mengganti jalur di skripnya.

Skrip 15.1: Membaca tabel atribut

Berkas `b2_01_atribut.py` sama dengan yang dipakai di Buku B2. Di dalam berkasnya tertulis "Skrip 1.1" mengikuti penomoran buku itu.

Tujuan. Memeriksa bahwa layer terbaca, menampilkan nama kolom dan jenisnya, melihat tiga petak pertama, lalu merangkum luas dan jenis tanaman.

Bahan. Paket data `paket-b2`, berkas `Petak_KPH.gpkg`, layer `Petak` (16 petak). Skrip dijalankan dari editor skrip QGIS atau dari terminal QGIS, seperti yang Anda siapkan di Bab 3.

Skrip ini bekerja dalam enam tahap. Gambar berikut menunjukkan urutannya, lengkap dengan nomor baris.

Ilustrasi 15.2: Alur skrip 15.1



Skrip utuh:

```python
# Skrip 1.1: membaca tabel atribut lapisan Petak (QGIS Python Console)
# Penulis: Badar Mubarok Yogaswara
import os
from qgis.core import QgsVectorLayer, QgsProject, QgsDistanceArea, QgsCoordinateTransformContext

DATA = os.environ.get("B2_DATA", r"C:/KPH_Contoh/paket-b2")
petak = QgsVectorLayer(DATA + "/Petak_KPH.gpkg|layername=Petak", "Petak", "ogr")
if not petak.isValid():
    raise SystemExit("Lapisan Petak tidak terbaca. Periksa DATA: " + DATA)
print("Valid:", petak.isValid(), "| Jumlah baris:", petak.featureCount())

# Kolom (field) dan tipenya
for fld in petak.fields():
    print(fld.name(), "-", fld.typeName())

# Tiga baris pertama
for i, fitur in enumerate(petak.getFeatures()):
    print(fitur["KODE_PETAK"], fitur["NAMA_KPH"], fitur["JENIS"], fitur["TAHUN_TANAM"], fitur["STATUS"])
    if i == 2:
        break

# Ringkasan dasar: jumlah dan rata-rata LUAS_LAMA_HA
nilai = [f["LUAS_LAMA_HA"] for f in petak.getFeatures()]
print("Jumlah LUAS_LAMA_HA:", round(sum(nilai), 2), "| Rata-rata:", round(sum(nilai) / len(nilai), 4))
print("Jenis unik:", sorted(set(f["JENIS"] for f in petak.getFeatures())))
```

Penjelasan baris demi baris. Nomor baris mengikuti berkas di atas.

Baris	Tulisan	Artinya
1-2	`# Skrip 1.1 ...`	Komentar: judul dan nama penulis. Python melewatinya.

Baris	Tulisan	Artinya
3	`import os`	Panggil perkakas `os`, yang mengurus hal-hal komputer seperti pengaturan dan jalur folder.
4	`from qgis.core import ...`	Panggil empat perkakas dari QGIS. Hanya `QgsVectorLayer` yang dipakai di skrip ini. Tiga lainnya sisa dan tidak berpengaruh.
6	`DATA = os.environ.get("B2_DATA", r"C:/KPH_Contoh/paket-b2")`	Isi kotak `DATA` dengan lokasi folder data. Python mencari pengaturan bernama `B2_DATA` lebih dulu. Bila tidak ada, ia memakai jalur cadangan di ujung. Huruf `r` di depan kutip berarti teks mentah, agar garis miring pada jalur Windows tidak disalahartikan.
7	`petak = QgsVectorLayer(...)`	Buka layer dan simpan di kotak `petak`. Ada tiga isian di dalam kurung. Isian pertama adalah sumber data: lokasi folder disambung dengan `+` ke nama berkas, lalu tanda garis tegak dan nama layer di dalam berkas (`layername=Petak`). Isian kedua adalah nama layer di QGIS. Isian ketiga, `"ogr"`, adalah pembaca untuk data vektor.
8-9	`if not petak.isValid():` lalu `raise SystemExit(...)`	Pengaman. Bila layer gagal dibuka, skrip berhenti di sini dan mencetak pesan beserta isi `DATA`. `SystemExit` artinya "hentikan skrip sekarang". Tanpa baris ini, skrip terus berjalan dengan layer kosong dan baru gagal di baris 24.
10	`print("Valid:", ...)`	Tampilkan apakah layer berhasil dibuka dan berapa jumlah barisnya.
13-14	`for fld in petak.fields():`	Ulangi satu kali untuk tiap kolom. Di setiap putaran, kotak `fld` berisi satu kolom, dan baris di bawahnya (yang menjorok ke dalam) mencetak nama serta jenisnya.
17-20	`for i, fitur in enumerate(...)`	Ulangi untuk tiap petak. `enumerate` menambahkan nomor urut `i`, mulai dari 0. Baris 19-20 berarti: bila `i` sudah 2 (petak ketiga), berhenti (`break`).
18	`fitur["KODE_PETAK"]`	Ambil isi kolom `KODE_PETAK` dari petak yang sedang dibaca. Nama kolom ditulis di dalam kurung siku dan kutip.
23	`nilai = [f["LUAS_LAMA_HA"] for f in ...]`	Kumpulkan isi kolom `LUAS_LAMA_HA` dari semua petak ke dalam satu daftar, hanya dengan satu baris. Cara penulisan ini disebut ***list comprehension*** (dibahas di Bab 6).
24	`round(sum(nilai), 2)`	`sum` menjumlahkan isi daftar. `len` menghitung banyak isinya. `round(..., 4)` membulatkan hasil bagi menjadi empat desimal.
25	`sorted(set(...))`	`set` membuang isi yang kembar, dan `sorted` mengurutkannya menurut abjad.

Hasil. Layar menampilkan (berdasarkan uji pada QGIS 4.0.2):

```
HASIL
Valid: True | Jumlah baris: 16
fid - Integer64
KODE_PETAK - String
NAMA_KPH - String
JENIS - String
TAHUN_TANAM - Integer
```

```
STATUS - String
LUAS_LAMA_HA - Real
P-01 KPH Alpha Jati 2008 Produksi
P-02 KPH Alpha Jati 2008 Produksi
P-03 KPH Beta Mahoni 2012 Produksi
Jumlah LUAS_LAMA_HA: 16.0 | Rata-rata: 1.0
Jenis unik: ['Akasia', 'Jati', 'Mahoni', 'Sengon']
```

Dari hasil ini Anda tahu: ada 16 petak, ada tujuh kolom, dan empat jenis tanaman.

Bagian yang boleh diubah.

- **Baris 6, lokasi data.** Ini yang paling sering Anda ubah. Ganti jalur di ujung dengan lokasi folder `paket-b2` di komputer Anda, misalnya `r"D:/Latihan/paket-b2"`. Gunakan garis miring biasa `/` agar aman di semua sistem.
- **Baris 7, nama berkas dan layer.** Ganti `Petak_KPH.gpkg` dan `layername=Petak` untuk membuka layer lain. Contoh: `Titik_Plot.gpkg|layername=Titik_Plot`.
- **Baris 19, angka 2.** Angka ini menentukan berapa petak yang dicetak. Ganti menjadi `4` untuk menampilkan lima petak, karena nomor urut mulai dari 0.
- **Baris 18, nama kolom.** Ganti dengan kolom lain dari hasil baris 13-14. Salin nama persis seperti di daftar itu. Di QGIS 4.0.2, `fitur["jenis"]` tetap terbaca walau hurufnya kecil semua, tetapi `fields().indexOf("jenis")` mengembalikan -1. Menyalin nama persis menghindari kejutan itu.
- **Baris 23-25, kolom hitungan.** `LUAS_LAMA_HA` boleh diganti kolom angka lain, dan `JENIS` boleh diganti kolom teks lain.

Jebakan.

- **Jalur salah.** Bila baris 6 menunjuk folder yang tidak ada, baris 8-9 menghentikan skrip dengan pesan `Lapisan Petak tidak terbaca. Periksa DATA: ...`, diikuti jalur yang Anda isi. Pesan itu muncul sebelum ada hasil apa pun. Pelajarannya: **baca pesan paling atas lebih dulu**, lalu periksa jalur di baris 6. Bila baris 8-9 dihapus, layer yang gagal dibuka melaporkan jumlah baris -2 (artinya "tidak diketahui"). Skrip lalu terus berjalan dan berhenti di baris 24 dengan `ZeroDivisionError`, pesan yang tidak menunjuk ke penyebab sebenarnya.
- **Satuan tidak dicek.** Angka `16.0` dan `1.0` hanyalah isi kolom. Skrip tidak tahu apakah satuannya hektare atau meter persegi. Cek arti kolom pada dokumentasi data.
- **Nama kolom salah ketik.** Nama yang benar-benar salah, misalnya `JNS`, menimbulkan `KeyError`. Salin nama kolom langsung dari hasil baris 13-14, termasuk huruf besarnya.

Latihan variasi.

1. Tampilkan **lima** petak pertama, bukan tiga.
2. Tampilkan daftar **status** yang ada (`STATUS`), bukan jenis tanaman.

Contoh jawaban, hanya bagian yang berubah:

```PYTHON
    if i == 4:
        break
print("Status unik:", sorted(set(f["STATUS"] for f in petak.getFeatures())))
```

Layar menampilkan lima petak, lalu `Status unik: ['Lindung', 'Produksi', 'Rehabilitasi'].`

> **Pola yang bisa dipakai ulang:** Skrip ini adalah kerangka "buka lemari lalu baca isinya". Untuk layer apa pun, ganti tiga hal: **lokasi data** (baris 6), **nama berkas dan layer** (baris 7), dan **nama kolom** (baris 18 dan 23-25). Sisanya tidak perlu disentuh.

Skrip 15.2: Memilih fitur dengan seleksi

Seleksi itu seperti memberi stabilo kuning pada baris-baris tertentu di tabel. Fitur yang disorot ikut tampil terpilih di peta. Skrip ini mencoba tujuh cara menyorot, dari yang paling sederhana sampai yang memakai lokasi. Berkasnya `b2_02_seleksi.py`; di dalamnya tertulis "Skrip 2.1".

Tujuan. Memilih petak dan plot dengan syarat, membalik pilihan, memilih menurut lokasi, mempersempit pilihan, lalu menyaring layer.

Bahan. Paket data `paket-b2`: layer `Petak` (16 petak) di `Petak_KPH.gpkg` dan layer `Titik_Plot` (24 titik pengukuran) di `Titik_Plot.gpkg`. Skrip memakai `processing`, alat bawaan Konsol Python QGIS. Cara menjalankannya ada di Bab 3.

Istilah baru di skrip ini:

- **seleksi**.
- **ekspresi**: Kalimat syarat yang dipahami QGIS.
- **fungsi**: `def`, dibahas di Bab 7.
- **alat**: `processing.run`.
- **filter layer**: Menyembunyikan baris yang tidak memenuhi syarat.

Ilustrasi 15.3: Alur skrip 15.2



Skrip utuh:

```python
# Skrip 2.1: query dan seleksi pada lapisan Petak dan Titik_Plot (QGIS Python Console)
# Penulis: Badar Mubarok Yogaswara
import os
import processing
from qgis.core import Qgis, QgsVectorLayer

DATA = os.environ.get("B2_DATA", r"C:/KPH_Contoh/paket-b2")
petak = QgsVectorLayer(DATA + "/Petak_KPH.gpkg|layername=Petak", "Petak", "ogr")
plot = QgsVectorLayer(DATA + "/Titik_Plot.gpkg|layername=Titik_Plot", "Titik_Plot", "ogr")


def kode_terpilih(lyr):
    return sorted(f["KODE_PETAK"] for f in lyr.selectedFeatures())


# 1. Seleksi dengan ekspresi: jati yang ditanam sebelum 2010
petak.selectByExpression("\"JENIS\" = 'Jati' AND \"TAHUN_TANAM\" < 2010")
print("Jati sebelum 2010:", petak.selectedFeatureCount(), kode_terpilih(petak))

# 2. Seleksi gabungan: status Produksi DAN bukan Jati
petak.selectByExpression("\"STATUS\" = 'Produksi' AND \"JENIS\" <> 'Jati'")
print("Produksi bukan Jati:", petak.selectedFeatureCount())

# 3. Memakai LIKE dan ATAU
petak.selectByExpression("\"JENIS\" LIKE 'S%' OR \"STATUS\" = 'Lindung'")
print("Jenis berawalan S atau Lindung:", petak.selectedFeatureCount(), kode_terpilih(petak))

# 4. Balikkan seleksi
petak.invertSelection()
print("Setelah dibalik:", petak.selectedFeatureCount())
petak.removeSelection()

# 5. Seleksi menurut lokasi: plot yang berada di dalam petak
hasil = processing.run("native:selectbylocation", {
    "INPUT": plot, "PREDICATE": [6], "INTERSECT": petak, "METHOD": 0})   # 6 = within
print("Plot di dalam petak:", plot.selectedFeatureCount(), "dari", plot.featureCount())
plot.selectByExpression("\"ID_PLOT\" IN ('PL-23','PL-24')")
print("Plot di luar (cek):", sorted(f["ID_PLOT"] for f in plot.selectedFeatures()))

# 6. Plot sehat dan berdiameter 30 cm ke atas, di dalam petak
plot.removeSelection()
processing.run("native:selectbylocation", {"INPUT": plot, "PREDICATE": [6], "INTERSECT": petak, "METHOD": 0})
dalam = plot.selectedFeatureCount()
plot.selectByExpression("\"KONDISI\" = 'Sehat' AND \"DIAMETER_CM\" >= 30",
Qgis.SelectBehavior.IntersectSelection)   # persempit seleksi yang ada
print("Sehat, diameter >= 30, di dalam petak:", plot.selectedFeatureCount(), "(dari", dalam, "plot di dalam
petak)")

# 7. Filter lapisan (Provider Feature Filter): hanya Jati yang tampil
petak.removeSelection()
petak.setSubsetString("\"JENIS\" = 'Jati'")
print("Setelah filter Jati, baris yang tampil:", petak.featureCount())
petak.setSubsetString("")
print("Filter dilepas, baris:", petak.featureCount())
```

Penjelasan baris demi baris.

Baris	Tulisan	Artinya
1-2	`# Skrip 2.1 ...`	Komentar: judul dan penulis.

Baris	Tulisan	Artinya
3-5	`import os`, `import processing`, `from qgis.core import ...`	Panggil perkakas: `os` untuk pengaturan, `processing` untuk alat QGIS, serta `Qgis` dan `QgsVectorLayer` dari QGIS.
7	`DATA = os.environ.get(...)`	Sama dengan Skrip 15.1: lokasi data dari pengaturan `B2_DATA`, atau jalur cadangan.
8-9	`petak = ...`, `plot = ...`	Buka dua layer dan simpan di kotak `petak` dan `plot`.
12-13	`def kode_terpilih(lyr):`	Buat fungsi kecil bernama `kode_terpilih`. Ia mengambil kode petak dari fitur yang terpilih, lalu mengurutkannya. Fungsi ini dipakai di baris 18 dan 26 supaya tidak menulis hal yang sama dua kali.
17	`petak.selectByExpression(...)`	Pilih petak yang memenuhi syarat: jenis Jati **dan** tahun tanam sebelum 2010. Nama kolom diapit kutip ganda, nilai teks diapit kutip tunggal. Tanda `\"` adalah cara menulis kutip ganda di dalam teks yang juga berkutip ganda.
18	`print("Jati sebelum 2010:", ...)`	Tampilkan banyaknya petak terpilih dan kodenya.
21-22	`... <> 'Jati'`	Syarat gabungan: status Produksi dan jenis bukan Jati. Tanda `<>` berarti "tidak sama dengan".
25-26	`LIKE 'S%' OR ...`	`LIKE 'S%'` berarti "diawali huruf S" (Sengon). `OR` berarti "atau": yang berstatus Lindung ikut terpilih.
29-31	`invertSelection()`, `removeSelection()`	Balikkan pilihan (yang terpilih jadi tidak, dan sebaliknya), cetak jumlahnya, lalu lepas semua pilihan.
34-35	`processing.run("native:selectbylocation", {...})`	Jalankan alat "pilih menurut lokasi". Kamus di dalam kurung berisi isian alat: `INPUT` (yang dipilih), `PREDICATE` (hubungan lokasi), `INTERSECT` (pembanding), dan `METHOD` (cara memilih). Angka `[6]` berarti "berada di dalam" (*within*). Angka `0` berarti pilihan baru.
36	`plot.selectedFeatureCount()`	Hitung plot yang terpilih, yaitu plot yang berada di dalam petak.
37-38	`plot.selectByExpression(... IN (...))`	Pilih dua plot dengan kodenya, lalu cetak: inilah plot yang berada di luar petak.
41-43	`plot.removeSelection()`, lalu alat yang sama	Lepas pilihan, ulangi pemilihan menurut lokasi, lalu simpan jumlah plot di dalam petak ke kotak `dalam`.
44	`..., Qgis.SelectBehavior.IntersectSelection`	Pilih lagi dengan syarat sehat dan diameter 30 cm ke atas, tetapi hanya di antara yang **sudah** terpilih. Itulah arti `IntersectSelection`: irisan.
45	`print(... , "(dari", dalam, ...)`	Tampilkan hasil dan pembandingnya.
48	`petak.removeSelection()`	Lepas pilihan petak.
49	`petak.setSubsetString(...)`	Pasang filter: hanya baris Jati yang tampil. Berbeda dari seleksi, filter benar-benar menyembunyikan baris lain.
51	`petak.setSubsetString("")`	Kosongkan filter, sehingga semua baris tampil lagi.

Hasil. Layar menampilkan (uji pada QGIS 4.0.2):

```
HASIL
Jati sebelum 2010: 4 ['P-01', 'P-02', 'P-05', 'P-13']
Produksi bukan Jati: 8
Jenis berawalan S atau Lindung: 6 ['P-04', 'P-06', 'P-07', 'P-11', 'P-13', 'P-15']
Setelah dibalik: 10
Plot di dalam petak: 22 dari 24
Plot di luar (cek): ['PL-23', 'PL-24']
Sehat, diameter >= 30, di dalam petak: 5 (dari 22 plot di dalam petak)
Setelah filter Jati, baris yang tampil: 5
Filter dilepas, baris: 16
```

Cocokkan: 6 terpilih dibalik menjadi 10, dan 6 + 10 = 16 petak. Plot di dalam petak 22, ditambah 2 di luar, sama dengan 24 plot.

Bagian yang boleh diubah.

- **Baris 17, 21, 25, dan 44: syarat seleksi.** Ganti jenis, tahun, status, atau ambang diameter. Contoh: `"JENIS" = 'Mahoni'`.
- **Baris 35 dan 42, angka predikat.** Angka 6 berarti "di dalam". Dari uji di komputer ini: `0` (bersinggungan atau beririsan) memilih 22 plot, dan `2` (terpisah) memilih 2 plot, yaitu plot yang di luar petak. Nomor ini hanya berlaku untuk alat `selectbylocation`.
- Baris 35, `METHOD`. Dari uji: `0` pilihan baru, `1` tambahkan ke pilihan, `2` ambil irisan dengan pilihan sekarang, `3` buang dari pilihan. [CEK] pada versi QGIS Anda.
- **Baris 49, filter.** Ganti `'Jati'` dengan jenis lain.

Jebakan.

- **Nama kolom salah tidak menimbulkan galat.** Menulis `"TAHUN" < 2010` padahal kolomnya `TAHUN_TANAM` tidak memunculkan pesan apa pun. Hasilnya hanya `0` petak terpilih. Bila hasil seleksi 0 yang tidak Anda duga, periksa dulu ejaan kolom.
- **Huruf besar dan kecil pada nilai.** `"JENIS" = 'jati'` memilih 0 petak, karena nilainya tertulis `Jati`.
- **Kode plot tertulis tetap.** Baris 37 memakai `PL-23` dan `PL-24`, yang cocok hanya dengan data latihan ini. Untuk data lain, ambil plot di luar petak dengan alat yang sama dan predikat `2`.
- **Filter menempel.** Bila skrip berhenti sebelum baris 51, filter Jati masih terpasang pada layer (uji: `featureCount` tinggal 5 dari 16). Selalu kosongkan filter setelah selesai.
- Hasil `processing.run` baris 34 tidak dipakai. Variabel `hasil` hanya sisa. Pilihannya sudah menempel pada layer `plot`.

Latihan variasi.

1. Pilih petak berjenis Mahoni dengan status Produksi, lalu cetak kodenya.
2. Cetak kode plot yang berada di **luar** petak, tanpa menulis `PL-23` dan `PL-24`.

Contoh jawaban, hanya bagian yang berubah:

```
PYTHON
petak.selectByExpression("\"JENIS\" = 'Mahoni' AND \"STATUS\" = 'Produksi'")
```

```
print(kode_terpilih(petak))

processing.run("native:selectbylocation", {"INPUT": plot, "PREDICATE": [2], "INTERSECT": petak, "METHOD": 0})
print(sorted(f["ID_PLOT"] for f in plot.selectedFeatures()))
```

Layar menampilkan `['P-03', 'P-10', 'P-14']` dan `['PL-23', 'PL-24']`.

> **Pola yang bisa dipakai ulang:** Pilih fitur dengan satu baris: `layer.selectByExpression("SYARAT")`. Ambil hasilnya dengan `layer.selectedFeatures()`. Untuk syarat lokasi, pakai `processing.run("native:selectbylocation", {"INPUT": ..., "PREDICATE": [...], "INTERSECT": ..., "METHOD": 0})`. Ganti **syarat**, **nama layer**, dan **angka predikat**.

Skrip 15.3: Menggambar dan membelah dengan koordinat

Biasanya Anda menggambar jalan dengan mouse. Skrip ini menggambarnya dengan koordinat yang diketik, seperti memberi tahu tukang gambar titik demi titik. Berkasnya `b2_03_digitasi.py`; di dalamnya tertulis "Skrip 3.1".

Tujuan. Menggambar satu jalan (garis) dan satu persemaian (poligon), lalu membelah petak P-12 menjadi dua, pada salinan data agar berkas asli aman.

Bahan. Paket data `paket-b2`: `Jalan_Baru.gpkg`, `Persemaian_Baru.gpkg`, `Petak_KPH.gpkg`, dan citra `Citra_Sketsa.tif`. Skrip membuat folder keluaran sendiri (`B2_KELUAR`) dan menulis salinannya di sana.

Istilah baru di skrip ini:

- **digitasi**.
- **simpul**: *vertex*, titik sudut sebuah garis atau poligon.
- **geometri**: Bentuk fitur.
- **commit**: Menyimpan hasil edit ke berkas.
- **koordinat UTM**: Angka posisi dalam meter.

Ilustrasi 15.4: Siklus edit layer lewat skrip



Ilustrasi 15.5: Alur skrip 15.3



Skrip utuh:

```python
# Skrip 3.1: menggambar (digitasi) jalan dan persemaian lewat Python, lalu membelah satu petak (QGIS Python Console)
# Penulis: Badar Mubarok Yogaswara
# Hasilnya sama dengan yang Anda kerjakan dengan mouse di Bab 3, tetapi simpul diketik sebagai koordinat.
import os
import shutil
from qgis.core import (QgsVectorLayer, QgsRasterLayer, QgsFeature, QgsGeometry, QgsPointXY, QgsPoint, QgsLineString)

DATA = os.environ.get("B2_DATA", r"C:/KPH_Contoh/paket-b2")
KELUAR = os.environ.get("B2_KELUAR", r"C:/KPH_Contoh/keluaran")
os.makedirs(KELUAR, exist_ok=True)
X0, Y0 = 312000.0, 9996000.0          # sudut barat-daya area contoh

# Salin lapisan kosong agar berkas asli di paket data tidak berubah
for nama in ("Jalan_Baru.gpkg", "Persemaian_Baru.gpkg", "Petak_KPH.gpkg"):
    shutil.copy(os.path.join(DATA, nama), os.path.join(KELUAR, nama))

jalan = QgsVectorLayer(KELUAR + "/Jalan_Baru.gpkg|layername=Jalan_Baru", "Jalan_Baru", "ogr")
semai = QgsVectorLayer(KELUAR + "/Persemaian_Baru.gpkg|layername=Persemaian_Baru", "Persemaian_Baru", "ogr")
petak = QgsVectorLayer(KELUAR + "/Petak_KPH.gpkg|layername=Petak", "Petak", "ogr")
citra = QgsRasterLayer(DATA + "/Citra_Sketsa.tif", "Citra_Sketsa")


def titik(x, y):
    return QgsPointXY(X0 + x, Y0 + y)


# 1. Jalan (garis): 6 simpul, diukur dari sudut barat-daya (meter)
simpul_jalan = [(0, 150), (90, 160), (180, 210), (260, 215), (330, 260), (400, 275)]
# Pemeriksaan: warna citra di tiap simpul harus cokelat (jalan tanah), bukan hijau
for x, y in simpul_jalan:
    xs = min(x, 399)      # simpul di tepi citra digeser 1 m ke dalam agar terbaca
    nilai = [citra.dataProvider().sample(titik(xs, y), b)[0] for b in (1, 2, 3)]
    print("Citra di simpul", (x, y), "RGB =", [int(v) for v in nilai])

jalan.startEditing()
f = QgsFeature(jalan.fields())
f.setGeometry(QgsGeometry.fromPolylineXY([titik(x, y) for x, y in simpul_jalan]))
f["NAMA_JALAN"] = "Jalan Inspeksi Timur"
f["KELAS"] = "Jalan tanah"
f["PANJANG_M"] = round(f.geometry().length(), 1)
jalan.addFeature(f)
```

```
print("Commit jalan:", jalan.commitChanges(), "| panjang (m):", round(f.geometry().length(), 1))

# 2. Persemaian (poligon): 5 simpul
simpul_semai = [(220, 25), (280, 20), (290, 70), (250, 85), (215, 65)]
semai.startEditing()
g = QgsGeometry.fromPolygonXY([[titik(x, y) for x, y in simpul_semai]])
f = QgsFeature(semai.fields())
f.setGeometry(g)
f["NAMA"] = "Persemaian Utara"
f["LUAS_M2"] = round(g.area(), 1)
semai.addFeature(f)
print("Commit persemaian:", semai.commitChanges(), "| luas (m2):", round(g.area(), 1), "| ha:",
round(g.area() / 10000, 4))

# 3. Membelah petak P-12 dengan garis tegak x = 350 (pilih P-12 lebih dulu supaya petak lain tidak ikut
terbelah)
petak.selectByExpression("\"KODE_PETAK\" = 'P-12'")
if petak.selectedFeatureCount() == 0:
    raise SystemExit("P-12 tidak ditemukan; pembelahan dibatalkan agar petak lain tidak ikut terbelah.")
luas_awal = sum(ft.geometry().area() for ft in petak.selectedFeatures())
petak.startEditing()
garis_belah = QgsLineString([QgsPoint(X0 + 350, Y0 + 190), QgsPoint(X0 + 350, Y0 + 310)])
hasil, _ = petak.splitFeatures(garis_belah, False, False)
print("Hasil belah:", hasil)
print("Jumlah petak setelah belah (sebelum commit):", petak.featureCount())
print("Commit petak:", petak.commitChanges())
print("Jumlah petak setelah commit:", petak.featureCount())
for ft in petak.getFeatures():
    if ft["KODE_PETAK"] == "P-12":
        print("P-12 bagian:", ft.id(), round(ft.geometry().area(), 1), "m2")
sisa = sum(ft.geometry().area() for ft in petak.getFeatures() if ft["KODE_PETAK"] == "P-12")
print("Luas P-12 sebelum:", round(luas_awal, 1), "| jumlah dua bagian:", round(sisa, 1))
```

Penjelasan baris demi baris.

Baris	Tulisan	Artinya
1-3	`# Skrip 3.1 ...`	Komentar: judul, penulis, dan catatan bahwa hasilnya sama dengan kerja manual, hanya simpulnya diketik.
4-6	`import os`, `import shutil`, `from qgis.core import (...)`	Panggil `os` dan `shutil` (menyalin berkas), lalu tujuh perkakas dari QGIS. Tanda kurung membuat daftar impor boleh ditulis di beberapa baris.
8-9	`DATA = ...`, `KELUAR = ...`	Dua lokasi: folder data asli dan folder hasil. Keduanya dari pengaturan, dengan jalur cadangan.
10	`os.makedirs(KELUAR, exist_ok=True)`	Buat folder hasil. `exist_ok=True` berarti: bila sudah ada, tidak apa-apa.
11	`X0, Y0 = 312000.0, 9996000.0`	Dua kotak diisi sekaligus: titik acuan di sudut barat-daya area contoh. Semua simpul dihitung dari titik ini.
14-15	`for nama in (...): shutil.copy(...)`	Salin tiga berkas GeoPackage dari folder data ke folder hasil. Setelah ini, semua edit terjadi di salinan.
17-19	`jalan`, `semai`, `petak`	Buka tiga layer dari **salinan** di folder hasil.
20	`QgsRasterLayer(...)`	Buka citra sketsa. Citra adalah raster, bukan vektor. Bab 16 membahasnya.
23-24	`def titik(x, y):`	Fungsi pembantu: ubah jarak dari titik acuan (meter) menjadi titik koordinat sebenarnya.

Baris	Tulisan	Artinya
28	`simpul_jalan = [(0, 150), ...]`	Daftar enam simpul jalan. Tiap simpul adalah pasangan (jarak ke timur, jarak ke utara). Pasangan seperti ini disebut *tuple*.
30-33	`for x, y in simpul_jalan:`	Untuk tiap simpul, baca tiga angka warna citra di sana (merah, hijau, biru) lalu cetak. Baris 31 menggeser simpul di tepi paling kanan 1 meter ke dalam, supaya masih terbaca.
35	`jalan.startEditing()`	Buka mode edit pada layer jalan.
36-37	`QgsFeature(...)`, `setGeometry(...)`	Buat fitur kosong, lalu isi bentuknya dengan garis dari keenam simpul.
38-40	`f["NAMA_JALAN"] = ...`	Isi tiga kolom: nama, kelas, dan panjang (dibulatkan satu desimal).
41-42	`jalan.addFeature(f)`, `commitChanges()`	Tambahkan fitur ke layer, lalu simpan. `commitChanges()` mengembalikan `True` bila berhasil, dan nilainya dicetak.
45-53	`simpul_semai` sampai `commitChanges()`	Pola yang sama untuk persemaian, tetapi bentuknya poligon dari lima simpul. Kolom `LUAS_M2` diisi dari luas bentuk.
56	`petak.selectByExpression(...)`	Pilih petak P-12 saja. Pilihan ini penting, karena pembelahan hanya berlaku untuk fitur yang terpilih.
57-58	`if petak.selectedFeatureCount() == 0:` lalu `raise SystemExit(...)`	Pengaman. Bila tidak ada petak terpilih (misalnya kode P-12 tidak ada di data), skrip berhenti dengan pesan. Tanpa baris ini, garis belah bisa membelah petak lain yang dilewatinya. `SystemExit` artinya "hentikan skrip sekarang" (lihat Skrip 15.1).
59	`luas_awal = sum(...)`	Jumlahkan luas fitur terpilih sebelum dibelah.
60-61	`petak.startEditing()`, `garis_belah = ...`	Buka mode edit petak. Siapkan garis belah tegak lurus pada jarak 350 meter ke timur.
62	`hasil, _ = petak.splitFeatures(...)`	Belah fitur yang terpilih dengan garis itu. Alat ini mengembalikan dua nilai. Nilai pertama `0` berarti berhasil. Garis bawah `_` menampung nilai kedua yang tidak dipakai.
63-66	`print(...)`, `commitChanges()`	Cetak hasil belah dan jumlah petak sebelum simpan. Lalu simpan (baris 65) dan cetak hasilnya, `True` bila berhasil. Terakhir, cetak jumlah petak sesudah simpan.
67-69	`for ft in petak.getFeatures():`	Cetak nomor dan luas dua bagian P-12.
70-71	`sisa = ...`, `print(...)`	Jumlahkan kedua bagian dan bandingkan dengan luas awal.

Hasil. Layar menampilkan (uji pada QGIS 4.0.2):

```
HASIL
Citra di simpul (0, 150) RGB = [150, 118, 78]
Citra di simpul (90, 160) RGB = [150, 118, 78]
Citra di simpul (180, 210) RGB = [150, 118, 78]
Citra di simpul (260, 215) RGB = [150, 118, 78]
Citra di simpul (330, 260) RGB = [150, 118, 78]
Citra di simpul (400, 275) RGB = [150, 118, 78]
Commit jalan: True | panjang (m): 428.5
Commit persemaian: True | luas (m2): 3700.0 | ha: 0.37
Hasil belah: 0
```

```
Jumlah petak setelah belah (sebelum commit): 17
Commit petak: True
Jumlah petak setelah commit: 17
P-12 bagian: 12 4906.9 m2
P-12 bagian: 17 4443.1 m2
Luas P-12 sebelum: 9350.0 | jumlah dua bagian: 9350.0
```

Keenam warna sama, `[150, 118, 78]`, yaitu cokelat: jalan tanah, bukan hijau. Luas dua bagian P-12 bila dijumlah sama dengan luas awal, 9350 meter persegi. Itu bukti pembelahan tidak menghilangkan bagian apa pun.

Bagian yang boleh diubah.

- **Baris 11, titik acuan.** Harus cocok dengan lokasi data Anda. Salah angka berarti gambar jatuh di tempat lain.
- **Baris 28 dan 45, daftar simpul.** Tambah, kurangi, atau ubah pasangan angkanya. Jalan butuh minimal dua simpul, poligon minimal tiga.
- **Baris 38-39 dan 50, isi kolom.** Ganti nama jalan atau nama persemaian.
- **Baris 56 dan 61, petak dan garis belah.** Ganti `'P-12'` dan angka `350`. Garis belah harus benar-benar memotong petak yang dipilih. Bila Anda mengganti kode petak, pesan di baris 58 masih menyebut P-12; ubah juga agar tidak membingungkan.

Jebakan.

- **Jalur data salah.** Bila `B2_DATA` menunjuk folder yang tidak ada, skrip berhenti di baris 15: `FileNotFoundError: [Errno 2] No such file or directory: 'C:/tidak_ada\\Jalan_Baru.gpkg'` Perhatikan jalur dalam pesan itu: garis miring biasa dan terbalik bercampur. Itu normal, jangan bingung.
- **Salinan ditimpa tiap kali dijalankan.** `shutil.copy` menimpa tanpa bertanya. Dari uji: dijalankan dua kali berturut-turut, hasilnya tetap 17 petak, karena salinan dibuat ulang dari data asli. Bahayanya muncul bila skrip lain memakai folder hasil yang sama.
- **Tidak ada petak terpilih.** Bila kode di baris 56 tidak ditemukan (misalnya salah ketik atau data lain), baris 57-58 menghentikan skrip dengan pesan `P-12 tidak ditemukan; pembelahan dibatalkan ...`. Ini disengaja. Tanpa pengaman itu, `luas_awal` menjadi `0`, dan garis belah bisa membelah setiap petak yang dilewatinya. Pada data bawaan, garis hanya memotong P-12, tetapi pada data lain hasilnya bisa lebih banyak petak yang terbelah. Itulah alasan komentar di baris 55.
- Lupa `commitChanges()`. Fitur baru hanya ada di memori dan hilang bila QGIS ditutup (uji: berkas dibaca ulang berisi 0 fitur).
- **Koordinat harus dalam meter.** Layer ini memakai UTM (EPSG:32749), jadi luas bernilai meter persegi. Pada koordinat derajat, angka luas tidak berarti.
- **Hasil commit tetap perlu dibaca.** Baris 42, 53, dan 65 mencetak hasil `commitChanges()`. Bila tertulis `False`, perubahan tidak tersimpan. Jangan lanjut sebelum penyebabnya jelas.

Latihan variasi.

1. Geser garis belah ke 330 meter. Berapa luas dua bagian P-12?
2. Ganti nama jalan menjadi `Jalan Inspeksi Barat`, lalu cetak isi kolom jalan dari layer salinan.

Contoh jawaban. Untuk nomor 1, ganti angka `350` di baris 61 menjadi `330` (dua tempat). Layar menampilkan bagian `4906.9` dan `4443.1` berubah menjadi `6817.6` dan `2532.4` meter persegi, dengan jumlah tetap 9350. Untuk nomor 2, ganti teks di baris 38, lalu tambahkan di akhir skrip:

```
PYTHON
for ft in jalan.getFeatures():
    print(ft["NAMA_JALAN"], ft["KELAS"], ft["PANJANG_M"])
```

Layar menampilkan `Jalan Inspeksi Barat Jalan tanah 428.5` pada baris terakhir.

> **Pola yang bisa dipakai ulang:** Menyunting layer lewat skrip selalu berbentuk empat langkah: **salin berkas**, `layer.startEditing()`, **buat atau ubah fitur** (`QgsFeature`, `addFeature`), lalu `layer.commitChanges()`. Ganti **nama berkas**, **daftar simpul**, dan **isi kolom**.

Skrip 15.4: Memeriksa dan memperbaiki geometri

Sebuah peta yang rapi punya aturan: bentuk harus sah, tidak ganda, tidak bertindihan, dan tidak berlubang. Memeriksanya satu per satu dengan mata sangat melelahkan. Skrip ini memeriksanya dengan alat. Berkasnya `b2_04_topologi.py`; di dalamnya tertulis "Skrip 4.1".

Tujuan. Menemukan dan memperbaiki geometri tidak sah, petak ganda, dan petak tumpang tindih pada layer latihan `Petak_Rusak`, lalu melaporkan celah.

Bahan. Paket data `paket-b2`, berkas `Petak_Rusak.gpkg`, layer `Petak_Rusak` (18 objek; rusaknya disengaja). Semua hasil disimpan di memori, jadi tidak ada berkas keluaran.

Istilah baru di skrip ini:

- **geometri sah**: Bentuk yang tidak menyilang dirinya sendiri.
- **duplikat**: Dua fitur yang sama persis.
- **tumpang tindih**: *overlap*.
- **celah**: *gap*, lubang di antara petak bertetangga.
- `**param`: Cara fungsi menerima isian dengan nama apa saja.

Ilustrasi 15.6: Alur skrip 15.4



Skrip utuh:

PYTHON

```
# Skrip 4.1: memeriksa dan memperbaiki galat topologi dasar pada Petak_Rusak (QGIS Python Console)
# Penulis: Badar Mubarok Yogaswara
import os
import processing
from qgis.core import QgsVectorLayer, QgsWkbTypes

DATA = os.environ.get("B2_DATA", r"C:/KPH_Contoh/paket-b2")
KELUAR = os.environ.get("B2_KELUAR", r"C:/KPH_Contoh/keluaran")
os.makedirs(KELUAR, exist_ok=True)
rusak = QgsVectorLayer(DATA + "/Petak_Rusak.gpkg|layername=Petak_Rusak", "Petak_Rusak", "ogr")
print("Jumlah objek Petak_Rusak:", rusak.featureCount())


def jalankan(alg, **param):
    return processing.run(alg, param)


# 1. Geometri tidak sah (mis. garis menyilang sendiri)
v = jalankan("native:checkvalidity", INPUT_LAYER=rusak, METHOD=2,
             VALID_OUTPUT="memory:", INVALID_OUTPUT="memory:", ERROR_OUTPUT="memory:")
print("Sah:", v["VALID_COUNT"], "| Tidak sah:", v["INVALID_COUNT"], "| Titik galat:", v["ERROR_COUNT"])
for f in v["INVALID_OUTPUT"].getFeatures():
    print("   tidak sah:", f["KODE_PETAK"], "|", f["_errors"])

# 2. Perbaiki geometri tidak sah lebih dulu (alat pemeriksa lain menolak geometri rusak)
perbaiki = jalankan("native:fixgeometries", INPUT=rusak, METHOD=1, OUTPUT="memory:")
fix = perbaiki["OUTPUT"]
print("Setelah perbaiki geometri:", fix.featureCount(), "objek")
v2 = jalankan("native:checkvalidity", INPUT_LAYER=fix, METHOD=2,
              VALID_OUTPUT="memory:", INVALID_OUTPUT="memory:", ERROR_OUTPUT="memory:")
print("Ulang cek: sah", v2["VALID_COUNT"], "| tidak sah", v2["INVALID_COUNT"])
for f in fix.getFeatures():
    if f["KODE_PETAK"] == "P-18":
        print("   P-18 setelah perbaikan:", QgsWkbTypes.displayString(f.geometry().wkbType()), "| luas m2:",
round(f.geometry().area(), 1))
```

```
# 3. Duplikat
d = jalankan("native:checkgeometryduplicate", INPUT=fix, UNIQUE_ID="fid", ERRORS="memory:")
print("Galat duplikat:", d["ERRORS"].featureCount())

# 4. Tumpang tindih (overlap) dan celah (gap)
o = jalankan("native:checkgeometryoverlap", INPUT=fix, UNIQUE_ID="fid", MIN_OVERLAP_AREA=0, ERRORS="memory:")
print("Galat tumpang tindih:", o["ERRORS"].featureCount())
print("   kolom:", [x.name() for x in o["ERRORS"].fields()])
for f in o["ERRORS"].getFeatures():
    print("   ", f.attributes()[1:])
g = jalankan("native:checkgeometrygap", INPUT=fix, UNIQUE_ID="fid", GAP_THRESHOLD=5000,
             NEIGHBORS="memory:", ERRORS="memory:", OUTPUT="memory:")
print("Galat celah:", g["ERRORS"].featureCount())
print("   kolom:", [x.name() for x in g["ERRORS"].fields()])
for f in g["ERRORS"].getFeatures():
    print("   ", f.attributes()[1:])

# 5. Perbaikan duplikat
bersih = jalankan("native:deleteduplicategeometries", INPUT=fix, OUTPUT="memory:", DUPLICATES="memory:")
print("Setelah hapus duplikat:", bersih["OUTPUT"].featureCount(), "objek | duplikat dibuang:",
bersih["DUPLICATE_COUNT"])

# 6. Perbaiki tumpang tindih (alat "Delete overlaps") pada lapisan yang sudah bebas duplikat
bersih_l = bersih["OUTPUT"]
o2 = jalankan("native:checkgeometryoverlap", INPUT=bersih_l, UNIQUE_ID="fid", MIN_OVERLAP_AREA=0,
ERRORS="memory:")
tanpa_tumpang = jalankan("native:fixgeometryoverlap", INPUT=bersih_l, ERRORS=o2["ERRORS"], UNIQUE_ID="fid",
                         OVERLAP_FEATURE_UNIQUE_IDX="gc_overlap_feature_fid", ERROR_VALUE_ID="gc_error",
                         OUTPUT="memory:", REPORT="memory:")["OUTPUT"]
o3 = jalankan("native:checkgeometryoverlap", INPUT=tanpa_tumpang, UNIQUE_ID="fid", MIN_OVERLAP_AREA=0,
ERRORS="memory:")
print("Setelah Delete overlaps | objek:", tanpa_tumpang.featureCount(), "| tumpang tindih tersisa:",
o3["ERRORS"].featureCount())
for f in tanpa_tumpang.getFeatures():
    if f["KODE_PETAK"] in ("P-06", "P-07"):
        print("   ", f["KODE_PETAK"], "luas m2:", round(f.geometry().area(), 1))
# Celah: alat "Fill gaps" dicoba pada data latihan dan melaporkan "Failed to merge with neighbor".
# Celah ini diperbaiki dengan tangan (Vertex Tool + snapping), seperti di Bab 4.
```

Penjelasan baris demi baris.

Baris	Tulisan	Artinya
1-5	komentar dan `import`	Judul, penulis, dan perkakas: `os`, `processing`, `QgsVectorLayer`, serta `QgsWkbTypes` untuk menamai jenis bentuk.
7-9	`DATA`, `KELUAR`, `os.makedirs(...)`	Lokasi data dan folder hasil. Folder hasil dibuat, tetapi skrip ini tidak memakainya.
10-11	`rusak = QgsVectorLayer(...)`	Buka layer latihan dan cetak jumlah objeknya.
14-15	`def jalankan(alg, param):`	Fungsi pembungkus. `alg` adalah nama alat. `param` menampung semua isian bernama, lalu `processing.run(alg, param)` meneruskannya sebagai kamus. Hasilnya, pemanggilan alat di bawah jadi lebih pendek.
19-20	`jalankan("native:checkvalidity", ...)`	Periksa kesahan geometri. Hasil sah, tidak sah, dan titik galat masing-masing dikirim ke layer sementara (`memory:`).
21	`print("Sah:", v["VALID_COUNT"], ...)`	Cetak tiga hitungan dari kamus hasil.

Baris	Tulisan	Artinya
22-23	`for f in v["INVALID_OUTPUT"].getFeatures():`	Cetak kode petak yang tidak sah dan jenis galatnya (kolom `_errors`).
26-28	`native:fixgeometries`	Perbaiki bentuk yang rusak lebih dulu, karena alat pemeriksa berikutnya menolak bentuk rusak (komentar baris 25). Hasilnya disimpan di kotak `fix`.
29-31	`checkvalidity` lagi	Periksa ulang `fix`: semua harus sah.
32-34	`if f["KODE_PETAK"] == "P-18":`	Cetak jenis bentuk dan luas P-18 setelah diperbaiki.
37-38	`native:checkgeometryduplicate`	Cari fitur ganda. `UNIQUE_ID="fid"` memberi tahu alat kolom mana yang unik untuk tiap fitur.
41-45	`native:checkgeometryoverlap`	Cari petak yang bertindihan. Baris 43 mencetak nama kolom hasil, dan baris 44-45 mencetak tiap galat tanpa kolom pertama (`[1:]`, potongan daftar).
46-51	`native:checkgeometrygap`	Cari celah. `GAP_THRESHOLD=5000` berarti: celah dengan luas sampai 5000 meter persegi dilaporkan.
54-55	`native:deleteduplicategeometries`	Hapus duplikat dan cetak jumlah objek serta duplikat yang dibuang.
58-62	`native:fixgeometryoverlap`	Perbaiki tumpang tindih pada layer bebas duplikat. Isian `gc_overlap_feature_fid` dan `gc_error` adalah nama kolom yang dihasilkan alat pemeriksa tadi.
63-64	`checkgeometryoverlap` lagi	Periksa ulang: sisa tumpang tindih harus 0.
65-67	`if f["KODE_PETAK"] in ("P-06", "P-07"):`	Cetak luas dua petak yang tadinya bertindihan.
68-69	komentar	Catatan penulis: alat "Fill gaps" gagal pada data latihan, jadi celah diperbaiki dengan tangan.

Hasil. Layar menampilkan (uji pada QGIS 4.0.2):

```
HASIL
Jumlah objek Petak_Rusak: 18
Sah: 17 | Tidak sah: 1 | Titik galat: 1
   tidak sah: P-18 | Self-intersection
Setelah perbaiki geometri: 18 objek
Ulang cek: sah 18 | tidak sah 0
   P-18 setelah perbaikan: MultiPolygon | luas m2: 450.0
Galat duplikat: 1
Galat tumpang tindih: 1
   kolom: ['gc_layerid', 'gc_layername', 'gc_errorx', 'gc_errory', 'gc_error', 'fid', 'gc_overlap_feature_fid']
    ['output', 312206.0, 9996165.0, 840.0, 7, 6]
Galat celah: 1
   kolom: ['gc_layerid', 'gc_layername', 'gc_partidx', 'gc_ringidx', 'gc_vertidx', 'gc_errorx', 'gc_errory',
'gc_error', 'gc_errorid']
    ['output', -1, -1, -1, 312101.875, 9996266.25, '265', 0]
Setelah hapus duplikat: 17 objek | duplikat dibuang: 1
Setelah Delete overlaps | objek: 17 | tumpang tindih tersisa: 0
    P-07 luas m2: 9350.0
    P-06 luas m2: 9900.0
```

Cara membacanya. P-18 menyilang dirinya sendiri (`Self-intersection`), dan setelah diperbaiki luasnya 450 meter persegi. Kolom `gc_error` pada galat tumpang tindih bernilai `840.0`: itu luas irisan dalam meter persegi, antara petak bernomor `fid` 7 dan 6. Buktinya, P-06 di data awal berluas 10740 dan setelah diperbaiki 9900. Selisihnya tepat 840. Nilai `265` pada galat celah kemungkinan luas celah. [kemungkinan]

Bagian yang boleh diubah.

- Baris 46, `GAP_THRESHOLD`. Ambang luas celah. Dari uji: bila diganti `100`, hasilnya `Galat celah: 0`, karena celah di data ini lebih luas daripada 100.
- Baris 41, 59, dan 63, `MIN_OVERLAP_AREA`. Tumpang tindih yang lebih kecil dari angka ini diabaikan. Nilai `0` melaporkan semuanya.
- **Baris 33 dan 66, kode petak yang dicetak.** Ganti sesuai petak yang ingin dilihat.

Jebakan.

- **Alat pemeriksa menolak bentuk rusak.** Dari uji: menjalankan `checkgeometryduplicate` langsung pada `Petak_Rusak` memberi pesan: `QgsProcessingException: Feature (18) from "r" has invalid geometry. Please fix the geometry or change the "Invalid features filtering" option for this input or globally in Processing settings.` Karena itu perbaikan geometri (baris 26) harus lebih dulu.
- **Hasil hilang saat QGIS ditutup.** Semua hasil bertempat di memori. Bila perlu disimpan, tambahkan alat penyimpan seperti di Skrip 15.5.
- **Kode petak tertulis tetap.** `P-18`, `P-06`, dan `P-07` hanya cocok dengan data latihan.
- **Celah tidak diperbaiki skrip.** Skrip hanya melaporkannya. Perbaikannya dilakukan dengan tangan, seperti di Buku B2.
- `fid` dan `gc_...`. `fid` adalah nomor urut bawaan GeoPackage. Awalan `gc_` kemungkinan singkatan *geometry checker*, nama kelompok alat ini. [kemungkinan]

Latihan variasi.

1. Cetak hanya kode petak yang tidak sah, tanpa jenis galatnya.
2. Naikkan `GAP_THRESHOLD` menjadi 10000. Apakah jumlah celah berubah?

Contoh jawaban. Nomor 1: ubah baris 23 menjadi `print(f["KODE_PETAK"])`. Layar menampilkan `P-18`. Nomor 2: ganti angka di baris 46. Hasilnya tetap `Galat celah: 1`, karena celah di data ini hanya satu dan luasnya jauh di bawah ambang itu.

> **Pola yang bisa dipakai ulang:** Pola "periksa, perbaiki, periksa ulang": `hasil = processing.run("ALAT_CEK", {...})`, baca hitungannya dari kamus hasil, perbaiki dengan alat perbaikan, lalu periksa lagi. Ganti **nama alat**, kolom `UNIQUE_ID`, dan **ambang**.

Skrip 15.5: Menambah kolom dengan rumus

Menghitung luas, umur, dan kelas untuk 16 petak dengan kalkulator itu membosankan. Skrip ini menyuruh QGIS mengisinya sekaligus. Berkas `b2_05_kalkulasi.py`; di dalamnya tertulis "Skrip 5.1".

Tujuan. Menambah lima kolom (luas, selisih luas, umur, kelas umur, kode lengkap), memeriksa hasilnya, lalu menyimpannya ke `Petak_Hitung.gpkg`.

Bahan. `Petak_KPH.gpkg`, layer `Petak`, dari paket `paket-b2`. Hasilnya, `Petak_Hitung.gpkg`, ditulis ke folder `B2_KELUAR` dan dipakai oleh Skrip 15.6 dan 15.7.

Istilah baru di skrip ini:

- **konstanta**: Kotak yang isinya tidak diubah, ditulis huruf besar.
- **kamus**: *dict*, pasangan nama dan nilai.
- **parameter bawaan**: Isian yang sudah punya nilai awal.
- **docstring**: Catatan di bawah `def`.
- **ekspresi QGIS**: Rumus di dalam teks.

Ilustrasi 15.7: Alur skrip 15.5



Skrip utuh:

```python
# Skrip 5.1: kalkulasi field pada lapisan Petak dengan ekspresi (QGIS Python Console)
# Penulis: Badar Mubarok Yogaswara
import os
import processing
from qgis.core import QgsVectorLayer

DATA = os.environ.get("B2_DATA", r"C:/KPH_Contoh/paket-b2")
KELUAR = os.environ.get("B2_KELUAR", r"C:/KPH_Contoh/keluaran")
os.makedirs(KELUAR, exist_ok=True)
petak = QgsVectorLayer(DATA + "/Petak_KPH.gpkg|layername=Petak", "Petak", "ogr")

TEKS, DESIMAL, BULAT = 2, 0, 1            # kode tipe field untuk alat Field calculator


def tambah(lyr, nama, tipe, rumus, panjang=0, presisi=0):
    """Menjalankan alat Field calculator dan mengembalikan lapisan hasil (sementara di memori)."""
    return processing.run("native:fieldcalculator", {
        "INPUT": lyr, "FIELD_NAME": nama, "FIELD_TYPE": tipe, "FIELD_LENGTH": panjang,
        "FIELD_PRECISION": presisi, "FORMULA": rumus, "OUTPUT": "memory:"})["OUTPUT"]


h = tambah(petak, "LUAS_HA", DESIMAL, "round(area($geometry) / 10000, 3)", 10, 3)
h = tambah(h, "SELISIH_HA", DESIMAL, "round(\"LUAS_HA\" - \"LUAS_LAMA_HA\", 3)", 10, 3)
h = tambah(h, "UMUR", BULAT, "2026 - \"TAHUN_TANAM\"")
```

```
h = tambah(h, "KELAS_UMUR", TEKS,
           "CASE WHEN \"UMUR\" < 10 THEN 'Muda' WHEN \"UMUR\" < 15 THEN 'Sedang' ELSE 'Tua' END", 10)
h = tambah(h, "KODE_LENGKAP", TEKS, "\"NAMA_KPH\" || ' / ' || \"KODE_PETAK\"", 30)

print("Kolom:", [f.name() for f in h.fields()])
for f in h.getFeatures():
    print(f["KODE_PETAK"], f["LUAS_HA"], f["LUAS_LAMA_HA"], f["SELISIH_HA"], f["UMUR"], f["KELAS_UMUR"],
f["KODE_LENGKAP"])

# Pemeriksaan: jumlah luas tiap KPH dan jumlah petak tiap kelas umur
luas = {}
kelas = {}
for f in h.getFeatures():
    luas[f["NAMA_KPH"]] = round(luas.get(f["NAMA_KPH"], 0) + f["LUAS_HA"], 3)
    kelas[f["KELAS_UMUR"]] = kelas.get(f["KELAS_UMUR"], 0) + 1
print("Luas per KPH (ha):", luas)
print("Petak per kelas umur:", kelas)
print("Total LUAS_HA:", round(sum(f["LUAS_HA"] for f in h.getFeatures()), 3))
beda = [f["KODE_PETAK"] for f in h.getFeatures() if abs(f["SELISIH_HA"]) > 0.06]
print("Petak dengan selisih lebih dari 0,06 ha:", beda)

# Simpan hasil agar dipakai di Bab 6 dan 7
processing.run("native:savefeatures", {"INPUT": h, "OUTPUT": KELUAR + "/Petak_Hitung.gpkg", "LAYER_NAME":
"Petak_Hitung"})
print("Tersimpan:", os.path.exists(KELUAR + "/Petak_Hitung.gpkg"))
```

Penjelasan baris demi baris.

Baris	Tulisan	Artinya
1-5	komentar dan `import`	Judul, penulis, dan tiga perkakas: `os`, `processing`, `QgsVectorLayer`.
7-10	`DATA`, `KELUAR`, `os.makedirs`, `petak`	Lokasi data, folder hasil (dibuat bila belum ada), dan layer `Petak`.
12	`TEKS, DESIMAL, BULAT = 2, 0, 1`	Tiga konstanta sekaligus. Alat kalkulator kolom meminta jenis kolom sebagai angka: 2 untuk teks, 0 untuk desimal, 1 untuk bilangan bulat. Nama yang jelas lebih mudah dibaca daripada angka.
15-19	`def tambah(lyr, nama, tipe, rumus, panjang=0, presisi=0):`	Fungsi pembungkus untuk alat `native:fieldcalculator`. `panjang=0` dan `presisi=0` adalah parameter bawaan: boleh tidak ditulis saat memanggil. Kalimat di dalam tiga kutip adalah docstring. Alat dipanggil dengan kamus isian, dan yang dikembalikan hanya lapisan hasil (`["OUTPUT"]`).
22	`h = tambah(petak, "LUAS_HA", DESIMAL, ...)`	Kolom pertama: luas dari bentuk, dibagi 10.000 (meter persegi menjadi hektare), dibulatkan tiga desimal.
23	`h = tambah(h, "SELISIH_HA", ...)`	Selisih antara luas hitungan dan kolom lama `LUAS_LAMA_HA`. Perhatikan `h` dipakai lagi sebagai masukan: tiap kolom ditambahkan ke hasil sebelumnya.
24	`"2026 - \"TAHUN_TANAM\""`	Umur = tahun acuan 2026 dikurangi tahun tanam.
25-26	`CASE WHEN ... END`	Kelas umur: di bawah 10 tahun Muda, di bawah 15 Sedang, selain itu Tua.
27	`"NAMA_KPH" ... ' / ' ... "KODE_PETAK"`	Rangkai dua kolom dengan tanda garis miring di antaranya. Dalam ekspresi QGIS, dua garis tegak berarti "sambung teks".

Baris	Tulisan	Artinya
29	`print("Kolom:", [f.name() for f in h.fields()])`	Cetak nama semua kolom.
30-31	`for f in h.getFeatures():`	Cetak tujuh isian tiap petak.
34-35	`luas = {}, kelas = {}`	Dua kamus kosong untuk menghitung.
36-38	`luas.get(f["NAMA_KPH"], 0)`	`kamus.get(kunci, 0)` mengambil isi bila kunci ada, dan `0` bila belum ada. Dengan itu, penjumlahan dan penghitungan berjalan tanpa menyiapkan kunci lebih dulu.
39-41	`print(...)`	Cetak luas tiap KPH, jumlah petak tiap kelas umur, dan total luas.
42-43	`beda = [...]`	Daftar petak yang selisih luasnya lebih dari 0,06 hektare. `abs` membuang tanda minus.
46-47	`native:savefeatures`, `os.path.exists`	Simpan `h` ke `Petak_Hitung.gpkg`, lalu periksa berkasnya ada.

Hasil. Layar menampilkan (uji pada QGIS 4.0.2):

```
HASIL
Kolom: ['fid', 'KODE_PETAK', 'NAMA_KPH', 'JENIS', 'TAHUN_TANAM', 'STATUS', 'LUAS_LAMA_HA', 'LUAS_HA',
'SELISIH_HA', 'UMUR', 'KELAS_UMUR', 'KODE_LENGKAP']
P-01 1.01 1.0 0.01 18 Tua KPH Alpha / P-01
P-02 0.93 1.0 -0.07 18 Tua KPH Alpha / P-02
P-03 0.995 0.95 0.045 14 Sedang KPH Beta / P-03
P-04 1.065 1.0 0.065 8 Muda KPH Beta / P-04
P-05 1.07 1.0 0.07 21 Tua KPH Alpha / P-05
P-06 0.99 1.05 -0.06 14 Sedang KPH Alpha / P-06
P-07 0.935 1.0 -0.065 8 Muda KPH Beta / P-07
P-08 1.005 1.0 0.005 11 Sedang KPH Beta / P-08
P-09 1.005 0.98 0.025 16 Tua KPH Gamma / P-09
P-10 1.055 1.0 0.055 14 Sedang KPH Gamma / P-10
P-11 0.995 1.0 -0.005 7 Muda KPH Gamma / P-11
P-12 0.935 1.0 -0.065 11 Sedang KPH Gamma / P-12
P-13 0.945 1.0 -0.055 21 Tua KPH Gamma / P-13
P-14 0.995 1.02 -0.025 12 Sedang KPH Gamma / P-14
P-15 1.065 1.0 0.065 6 Muda KPH Gamma / P-15
P-16 1.005 1.0 0.005 10 Sedang KPH Gamma / P-16
Luas per KPH (ha): {'KPH Alpha': 4.0, 'KPH Beta': 4.0, 'KPH Gamma': 8.0}
Petak per kelas umur: {'Tua': 5, 'Sedang': 7, 'Muda': 4}
Total LUAS_HA: 16.0
Petak dengan selisih lebih dari 0,06 ha: ['P-02', 'P-04', 'P-05', 'P-07', 'P-12', 'P-15']
Tersimpan: True
```

Pemeriksaan: 4 + 4 + 8 = 16 hektare, dan 5 + 7 + 4 = 16 petak. Angka-angka itu cocok dengan kolom lama.

Bagian yang boleh diubah.

- **Baris 24, tahun acuan 2026.** Ganti dengan tahun Anda. Dari uji: dengan `2030`, P-01 berumur 22 dan kelas umur menjadi `{'Tua': 11, 'Sedang': 5}`.
- **Baris 26, batas kelas umur.** Ganti angka 10 dan 15. Dari uji: dengan 12 dan 20, hasilnya `{'Sedang': 7, 'Muda': 7, 'Tua': 2}`.

- **Baris 22-27, rumus.** Tiap rumus adalah ekspresi QGIS yang sama dengan di kalkulator kolom. Ganti nama kolom dan hitungannya.
- **Baris 42, ambang 0.06.** Ganti sesuai toleransi selisih luas Anda.
- **Baris 46, nama berkas hasil.** Bila diganti, Skrip 15.6 harus ikut diubah.

Jebakan.

- **Tahun 2026 tertulis tetap.** Hasil umur berbeda bila skrip dipakai di tahun lain.
- **Kode jenis kolom bergantung pada alat.** Angka 2, 0, dan 1 di baris 12 adalah urutan pilihan pada alat. [CEK] bila versi QGIS Anda berbeda.
- **Satuan luas.** `area($geometry)` memakai satuan layer. Layer ini UTM (meter), jadi hasilnya meter persegi dan dibagi 10.000 menjadi hektare. Pada layer berkoordinat derajat, angkanya tidak berarti. Skrip tidak mengingatkan hal ini.
- **Selisih dihitung dari kolom lama.** Selisih bisa bernilai `None` bila kolom lama kosong, dan `abs(None)` menimbulkan galat. [kemungkinan]
- **Menjalankan ulang aman.** Dari uji: dua kali berturut-turut tetap menghasilkan 16 petak di `Petak_Hitung.gpkg`, karena berkas ditimpa.

Latihan variasi.

1. Ganti tahun acuan menjadi 2030 lalu lihat jumlah petak tiap kelas umur.
2. Tambahkan kolom `LUAS_M2` (luas dalam meter persegi, tanpa dibagi 10.000).

Contoh jawaban. Nomor 1 hanya mengganti angka di baris 24, hasilnya `{'Tua': 11, 'Sedang': 5}`. Nomor 2 menambah satu baris sesudah baris 22. Dari uji, dua petak pertama berluas `10100.0` dan `9300.0` meter persegi:

```python
h = tambah(h, "LUAS_M2", DESIMAL, "round(area($geometry), 1)", 12, 1)
```

> **Pola yang bisa dipakai ulang:** Bungkus alat yang sering dipakai menjadi fungsi, lalu rangkai: `h = tambah(h, "NAMA_KOLOM", JENIS, "RUMUS", panjang, presisi)`. Ganti **nama kolom**, **jenis**, dan **rumus**. Simpan dengan `native:savefeatures`.

Skrip 15.6: Menggabungkan tabel dengan join

Hasil pengukuran di lapangan sering datang sebagai tabel terpisah, misalnya berkas CSV. Join menempelkan kolomnya ke tabel petak, dengan kunci yang sama sebagai pencocok, seperti menyatukan dua daftar nama lewat nomor induk. Berkas `b2_06_join.py`; di dalamnya tertulis "Skrip 6.1".

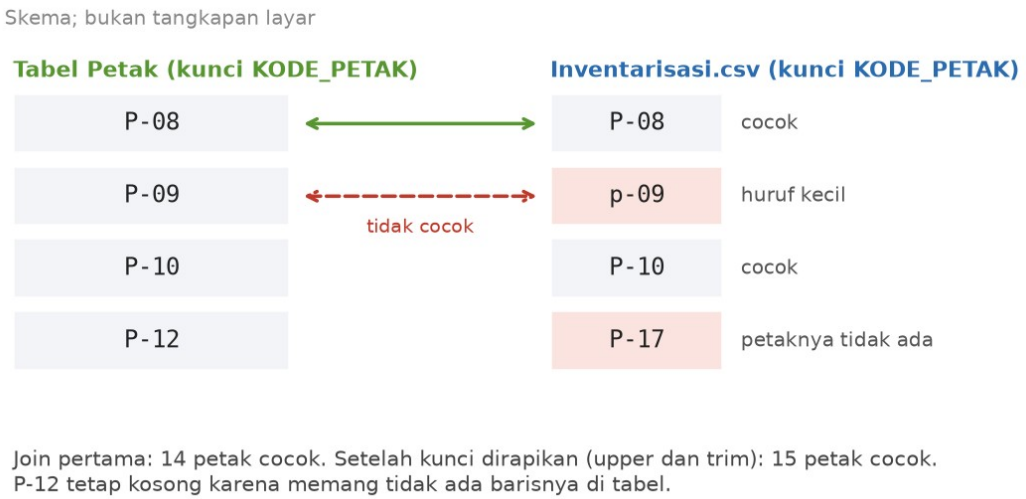
Tujuan. Menempelkan jumlah pohon dan volume dari `Inventarisasi.csv` ke petak dan memperbaiki kunci yang tidak cocok. Lalu menghitung kepadatan dan volume per hektare, mencoba join virtual, dan menyimpan hasilnya.

Bahan. Hasil Skrip 15.5 (`Petak_Hitung.gpkg` di `B2_KELUAR`) dan `Inventarisasi.csv` dari paket `paket-b2`. **Jalankan Skrip 15.5 lebih dulu.**

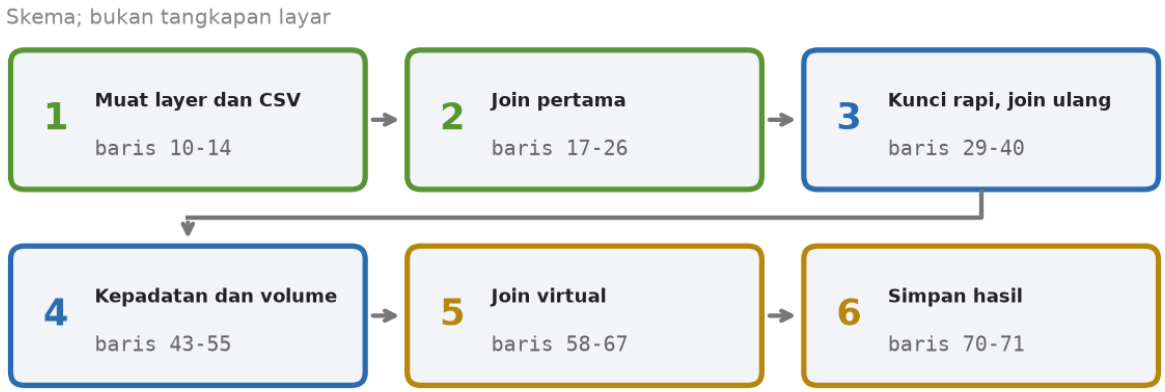
Istilah baru di skrip ini:

- **CSV**: Tabel teks dengan koma sebagai pemisah.
- **kunci join**: Kolom pencocok.
- **join permanen**: Hasilnya layer baru.
- **join virtual**: Hanya tampilan di layer, tanpa berkas baru.
- `lambda`: Fungsi sekali pakai, dibahas di Bab 7.

Ilustrasi 15.8: Contoh kunci tidak cocok pada join



Ilustrasi 15.9: Alur skrip 15.6



Skrip utuh:

```python
# Skrip 6.1: menggabungkan (join) tabel Inventarisasi.csv ke lapisan Petak (QGIS Python Console)
# Penulis: Badar Mubarok Yogaswara
# Jalankan Skrip 5.1 lebih dulu (menghasilkan Petak_Hitung.gpkg di folder keluaran).
import os
import processing
from qgis.core import QgsVectorLayer, QgsVectorLayerJoinInfo

DATA = os.environ.get("B2_DATA", r"C:/KPH_Contoh/paket-b2")
KELUAR = os.environ.get("B2_KELUAR", r"C:/KPH_Contoh/keluaran")
```

```
petak = QgsVectorLayer(KELUAR + "/Petak_Hitung.gpkg|layername=Petak_Hitung", "Petak_Hitung", "ogr")
uri = "file:///" + DATA.replace("\\", "/") + "/Inventarisasi.csv?
type=csv&delimiter=,&detectTypes=yes&geomType=none"
inv = QgsVectorLayer(uri, "Inventarisasi", "delimitedtext")
print("Tabel valid:", inv.isValid(), "| baris:", inv.featureCount())
print("Tipe kolom:", [(f.name(), f.typeName()) for f in inv.fields()])

# 1. Join permanen (hasilnya lapisan baru): kunci KODE_PETAK di kedua tabel
g = processing.run("native:joinattributestable", {
    "INPUT": petak, "FIELD": "KODE_PETAK", "INPUT_2": inv, "FIELD_2": "KODE_PETAK",
    "FIELDS_TO_COPY": ["JML_POHON", "VOLUME_M3", "TGL_UKUR"], "METHOD": 1, "DISCARD_NONMATCHING": False,
"PREFIX": "inv_",
    "OUTPUT": "memory:", "NON_MATCHING": "memory:"})
hasil = g["OUTPUT"]
cocok = [f["KODE_PETAK"] for f in hasil.getFeatures() if f["inv_JML_POHON"] is not None]
kosong = [f["KODE_PETAK"] for f in hasil.getFeatures() if f["inv_JML_POHON"] is None]
print("Join 1 | petak cocok:", len(cocok), "| petak tanpa pasangan:", kosong)
kode_petak = {f["KODE_PETAK"] for f in petak.getFeatures()}
print("          baris tabel yang tidak punya petak:", sorted(f["KODE_PETAK"] for f in inv.getFeatures() if
f["KODE_PETAK"] not in kode_petak))

# 2. Perbaiki kuncinya: huruf kecil menjadi huruf besar. Kolom bantu dibuat di tabel, lalu join diulang.
inv2 = processing.run("native:fieldcalculator", {
    "INPUT": inv, "FIELD_NAME": "KUNCI", "FIELD_TYPE": 2, "FIELD_LENGTH": 10, "FIELD_PRECISION": 0,
    "FORMULA": "upper(trim(\"KODE_PETAK\"))", "OUTPUT": "memory:"})["OUTPUT"]
g2 = processing.run("native:joinattributestable", {
    "INPUT": petak, "FIELD": "KODE_PETAK", "INPUT_2": inv2, "FIELD_2": "KUNCI",
    "FIELDS_TO_COPY": ["JML_POHON", "VOLUME_M3", "TGL_UKUR"], "METHOD": 1, "DISCARD_NONMATCHING": False,
"PREFIX": "inv_",
    "OUTPUT": "memory:", "NON_MATCHING": "memory:"})
h2 = g2["OUTPUT"]
cocok2 = [f for f in h2.getFeatures() if f["inv_JML_POHON"] is not None]
kosong2 = [f["KODE_PETAK"] for f in h2.getFeatures() if f["inv_JML_POHON"] is None]
print("Join 2 | petak cocok:", len(cocok2), "| petak tanpa pasangan:", kosong2,
      "| baris tabel tanpa petak:", sorted(f["KUNCI"] for f in inv2.getFeatures() if f["KUNCI"] not in
kode_petak))

# 3. Hitung kepadatan pohon dan volume per hektare dari hasil join
h3 = processing.run("native:fieldcalculator", {
    "INPUT": h2, "FIELD_NAME": "POHON_HA", "FIELD_TYPE": 0, "FIELD_LENGTH": 10, "FIELD_PRECISION": 1,
    "FORMULA": "round(\"inv_JML_POHON\" / \"LUAS_HA\", 1)", "OUTPUT": "memory:"})["OUTPUT"]
h3 = processing.run("native:fieldcalculator", {
    "INPUT": h3, "FIELD_NAME": "VOL_HA", "FIELD_TYPE": 0, "FIELD_LENGTH": 10, "FIELD_PRECISION": 1,
    "FORMULA": "round(\"inv_VOLUME_M3\" / \"LUAS_HA\", 1)", "OUTPUT": "memory:"})["OUTPUT"]
for f in h3.getFeatures():
    print(f["KODE_PETAK"], f["LUAS_HA"], f["inv_JML_POHON"], f["POHON_HA"], f["inv_VOLUME_M3"], f["VOL_HA"])
tot_pohon = sum(f["inv_JML_POHON"] for f in cocok2)
tot_vol = round(sum(f["inv_VOLUME_M3"] for f in cocok2), 1)
print("Total pohon (15 petak):", tot_pohon, "| total volume m3:", tot_vol)
tertinggi = max((f for f in h3.getFeatures() if f["VOL_HA"] is not None), key=lambda x: x["VOL_HA"])
print("Volume per ha tertinggi:", tertinggi["KODE_PETAK"], tertinggi["VOL_HA"])

# 4. Join virtual (tidak membuat berkas baru), setara Layer Properties > Joins
info = QgsVectorLayerJoinInfo()
info.setJoinLayer(inv)
info.setJoinFieldName("KODE_PETAK")
info.setTargetFieldName("KODE_PETAK")
info.setPrefix("inv_")
info.setUsingMemoryCache(True)
petak.addJoin(info)
print("Join virtual | kolom setelah join:", [f.name() for f in petak.fields()][-5:])
n = sum(1 for f in petak.getFeatures() if f["inv_JML_POHON"] is not None)
print("Join virtual | petak yang terisi:", n)

# 5. Simpan hasil akhir untuk Bab 7
processing.run("native:savefeatures", {"INPUT": h3, "OUTPUT": KELUAR + "/Petak_Inventarisasi.gpkg",
"LAYER_NAME": "Petak_Inventarisasi"})
print("Tersimpan:", os.path.exists(KELUAR + "/Petak_Inventarisasi.gpkg"))
```

Penjelasan baris demi baris.

Baris	Tulisan	Artinya
1-3	komentar	Judul, penulis, dan pengingat: jalankan Skrip 5.1 lebih dulu (di buku ini, Skrip 15.5).
4-6	`import ...`	`os`, `processing`, serta `QgsVectorLayer` dan `QgsVectorLayerJoinInfo` (pembuat join virtual).
8-9	`DATA`, `KELUAR`	Lokasi data dan lokasi hasil.
10	`petak = QgsVectorLayer(KELUAR + ...)`	Buka hasil Skrip 15.5. Bila berkas itu belum ada, layar menampilkan galat di bawah.
11	`uri = "file:///" + ...`	Susun alamat CSV. `.replace("\\", "/")` mengganti garis miring terbalik Windows menjadi garis miring biasa. Ujung alamat memberi tahu QGIS: ini CSV, pemisahnya koma, tipe kolom ditebak, dan tidak ada kolom bentuk.
12-14	`inv = QgsVectorLayer(uri, ..., "delimitedtext")`	Buka CSV sebagai layer tabel. Pembacanya `delimitedtext`, bukan `ogr`. Cetak status, jumlah baris, dan tipe kolom.
17-21	`native:joinattributestable`	Join pertama. `FIELD` dan `FIELD_2` adalah kolom kunci di tabel pertama dan kedua. `FIELDS_TO_COPY` memilih kolom yang ditempel. `PREFIX` menambah awalan `inv_`. `DISCARD_NONMATCHING: False` berarti petak tanpa pasangan tetap dipertahankan.
22-24	`cocok`, `kosong`	Dua daftar dengan list comprehension: petak yang kolom `inv_JML_POHON`-nya terisi, dan yang `None` (kosong).
25-26	`kode_petak = {...}`	Himpunan semua kode petak. Lalu cetak baris tabel yang tidak punya petak. Kurung kurawal `{}` dengan isi seperti ini disebut set comprehension.
29-31	`native:fieldcalculator`	Buat kolom bantu `KUNCI` di tabel CSV: `upper(trim(...))`. `trim` membuang spasi, `upper` membuat huruf besar. Maka `p-09` menjadi `P-09`.
32-40	join kedua	Ulangi join, kini dengan kunci `KUNCI`. Baris 39-40 mencetak jumlah yang cocok, yang kosong, dan baris tabel tanpa petak.
43-48	dua kali `fieldcalculator`	Hitung `POHON_HA` (jumlah pohon dibagi luas) dan `VOL_HA` (volume dibagi luas), dibulatkan satu desimal.
49-50	`for f in h3.getFeatures():`	Cetak enam isian tiap petak.
51-53	`tot_pohon`, `tot_vol`	Jumlahkan pohon dan volume dari petak yang cocok. Tulisan "15 petak" di baris 53 adalah angka tetap.
54-55	`max(..., key=lambda x: x["VOL_HA"])`	Cari petak dengan `VOL_HA` terbesar. `lambda` adalah fungsi mini: "diberi `x`, kembalikan `x["VOL_HA"]`". Petak yang `VOL_HA`-nya `None` disaring lebih dulu.

Baris	Tulisan	Artinya
58-64	`QgsVectorLayerJoinInfo()`	Join virtual: tentukan layer yang digabung, kolom kunci kedua sisi, dan awalan, lalu `petak.addJoin(info)`.
65-67	`[-5:]`, `sum(1 for ...)`	Cetak lima nama kolom terakhir, lalu hitung petak yang terisi. `[-5:]` berarti "lima terakhir".
70-71	`native:savefeatures`	Simpan `h3` ke `Petak_Inventarisasi.gpkg`, lalu periksa berkasnya ada.

Hasil. Layar menampilkan (uji pada QGIS 4.0.2):

```
HASIL
Tabel valid: True | baris: 16
Tipe kolom: [('KODE_PETAK', 'text'), ('JML_POHON', 'integer'), ('VOLUME_M3', 'double'), ('TGL_UKUR', 'date'),
('REGU', 'text')]
Join 1 | petak cocok: 14 | petak tanpa pasangan: ['P-09', 'P-12']
         baris tabel yang tidak punya petak: ['P-17', 'p-09']
Join 2 | petak cocok: 15 | petak tanpa pasangan: ['P-12'] | baris tabel tanpa petak: ['P-17']
P-01 1.01 970 960.4 246.8 244.4
P-02 0.93 842 905.4 235.0 252.7
P-03 0.995 989 994.0 202.7 203.7
P-04 1.065 796 747.4 125.0 117.4
P-05 1.07 920 859.8 275.7 257.7
P-06 0.99 863 871.7 181.7 183.5
P-07 0.935 846 904.8 119.8 128.1
P-08 1.005 844 839.8 148.5 147.8
P-09 1.005 920 915.4 218.1 217.0
P-10 1.055 813 770.6 180.5 171.1
P-11 0.995 892 896.5 113.0 113.6
P-12 0.935 None None None None
P-13 0.945 915 968.3 279.2 295.4
P-14 0.995 845 849.2 179.8 180.7
P-15 1.065 829 778.4 99.6 93.5
P-16 1.005 978 973.1 156.3 155.5
Total pohon (15 petak): 13262 | total volume m3: 2761.7
Volume per ha tertinggi: P-13 295.4
Join virtual | kolom setelah join: ['KODE_LENGKAP', 'inv_JML_POHON', 'inv_VOLUME_M3', 'inv_TGL_UKUR',
'inv_REGU']
Join virtual | petak yang terisi: 14
Tersimpan: True
```

Tiga pelajaran dari hasil ini. **Pertama**, join pertama hanya cocok 14: `p-09` huruf kecil gagal bertemu `P-09`. **Kedua**, setelah kunci dirapikan, 15 cocok. Satu petak (`P-12`) memang tidak punya data, dan satu baris tabel (`P-17`) tidak punya petak. **Ketiga**, join virtual hanya 14 petak terisi, karena ia memakai kunci asli yang belum dirapikan. Tabel tipe kolom juga memuat kolom `REGU` yang tidak ikut dicopy di join permanen, tetapi ikut tampil pada join virtual.

Bagian yang boleh diubah.

- Baris 19 dan 34, `FIELDS_TO_COPY`. Tambah atau kurangi kolom. Dari uji: menambah `"REGU"` membuat kolom `inv_REGU` ikut tersedia, misalnya `('P-01', 'Regu 1')`.
- Baris 19 dan 34, `PREFIX`. Ganti awalan `inv_`, lalu ganti juga di baris 22, 23, 37, 38, 45, 48, dan seterusnya.
- Baris 11, `delimiter=,`. Ganti bila CSV Anda memakai titik koma.
- **Baris 31, rumus kunci.** Ganti `upper(trim(...))` sesuai jenis ketidakcocokan.

- **Baris 44 dan 47, angka 1 pada pembulatan.** Ganti jumlah desimal.

Jebakan.

- **Lupa menjalankan Skrip 15.5.** Layer `petak` tidak valid, dan baris 17 berhenti dengan: `_core.QgsProcessingException: Invalid join field from layer 1: "KODE_PETAK" does not exist` Pesannya menyesatkan: kolomnya ada, hanya berkasnya belum ada. Cek Skrip 15.5 dulu.
- **Kunci tidak cocok tanpa galat.** Join tidak pernah protes bila ada yang tidak cocok. Baris 24 dan 26 ada untuk memeriksanya. Selalu baca keduanya.
- **Angka tetap.** "15 petak" di baris 53 hanya benar untuk data ini.
- **Join virtual tidak tersimpan.** Join di baris 64 hilang bila QGIS ditutup, dan ia tidak membuat berkas.
- **Tipe kolom CSV.** Jenis kolom ditebak dari isi (`detectTypes=yes`). Kolom dengan isi campur bisa terbaca sebagai teks.
- **Dua variabel serupa.** `hasil` dan `h2` dipakai bergantian untuk dua join. Jangan tertukar.

Latihan variasi.

1. Cetak kode petak yang kepadatannya (`POHON_HA`) di atas 900.
2. Ikutkan kolom `REGU` dalam join permanen kedua, lalu cetak regu empat petak pertama.

Contoh jawaban. Nomor 1, tambahkan di akhir skrip:

```python
padat = [f["KODE_PETAK"] for f in h3.getFeatures() if f["POHON_HA"] is not None and f["POHON_HA"] > 900]
print("Petak dengan POHON_HA di atas 900:", padat)
```

Layar menampilkan `['P-01', 'P-02', 'P-03', 'P-07', 'P-09', 'P-13', 'P-16']`. Nomor 2: tambahkan `"REGU"` di baris 34 dan cetak `[(f["KODE_PETAK"], f["inv_REGU"]) for f in h2.getFeatures()][:4]`. Layar menampilkan `[('P-01', 'Regu 1'), ('P-02', 'Regu 2'), ('P-03', 'Regu 3'), ('P-04', 'Regu 1')]`.

> **Pola yang bisa dipakai ulang:** Join yang aman: **cek dulu kuncinya** (berapa yang cocok, berapa tidak), **rapikan** dengan kolom bantu, **ulangi join**, lalu simpan. Ganti **nama kolom kunci**, **kolom yang disalin**, dan **awalan**.

Skrip 15.7: Membuat rekap untuk Kepala Seksi

Semua kerja tadi bermuara di sini: ringkasan yang bisa dibaca atasan dan diekspor. Berkas `b2_07_laporan.py`; di dalamnya tertulis "Skrip 7.1".

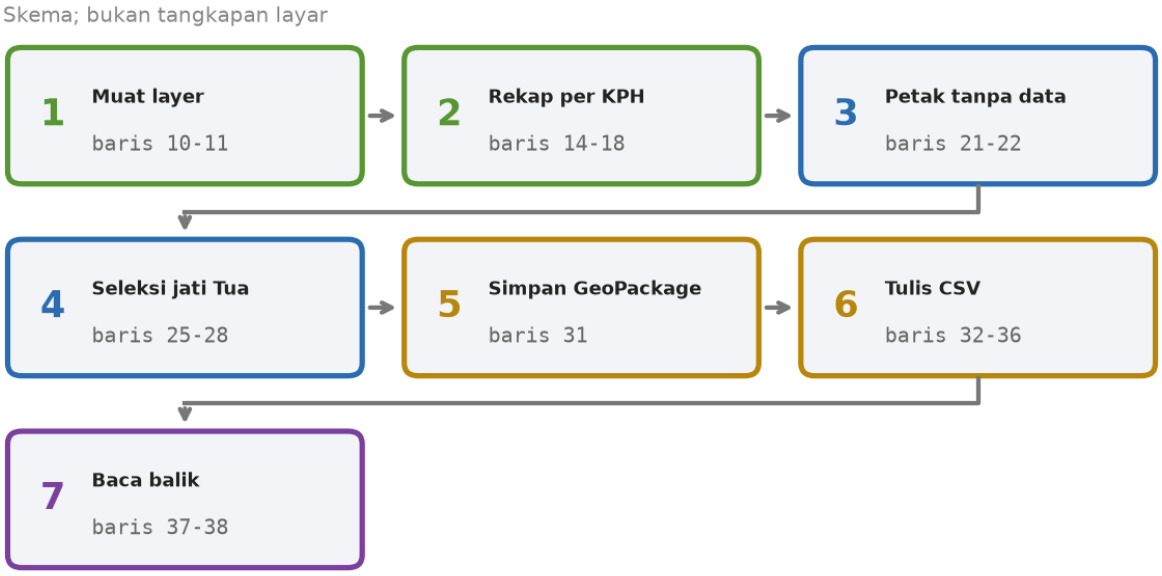
Tujuan. Merekap pohon per KPH, mendaftar petak tanpa data, memilih jati kelas umur Tua yang diurutkan menurut volume, lalu mengekspor ke GeoPackage dan CSV.

Bahan. `Petak_Inventarisasi.gpkg` di `B2_KELUAR`, hasil Skrip 15.5 dan 15.6. **Jalankan keduanya lebih dulu.** Skrip ini tidak memakai `B2_DATA`.

Istilah baru di skrip ini:

- **rekap per kategori**: Statistik tiap kelompok.
- **ekspor**: Menyimpan hasil ke berkas lain.
- `with open`: Membuka berkas dan otomatis menutupnya, Bab 9.
- `csv.writer`: Penulis baris CSV.

Ilustrasi 15.10: Alur skrip 15.7



Ilustrasi 15.11: Rantai Skrip 15.5 sampai 15.7



Jalankan berurutan: 15.5, lalu 15.6, lalu 15.7.

Skrip utuh:

```python
# Skrip 7.1: rekap akhir untuk Kepala Seksi dari lapisan Petak_Inventarisasi (QGIS Python Console)
# Penulis: Badar Mubarok Yogaswara
# Jalankan Skrip 5.1 dan 6.1 lebih dulu (menghasilkan Petak_Inventarisasi.gpkg di folder keluaran).
import csv
```

```
import os
import processing
from qgis.core import QgsVectorLayer

KELUAR = os.environ.get("B2_KELUAR", r"C:/KPH_Contoh/keluaran")
lap = QgsVectorLayer(KELUAR + "/Petak_Inventarisasi.gpkg|layername=Petak_Inventarisasi", "Petak_Inventarisasi",
"ogr")
print("Baris:", lap.featureCount())

# 1. Rekap per KPH dengan alat "Statistics by categories"
rek = processing.run("qgis:statisticsbycategories", {
    "INPUT": lap, "VALUES_FIELD_NAME": "inv_JML_POHON", "CATEGORIES_FIELD_NAME": ["NAMA_KPH"], "OUTPUT":
"memory:"})["OUTPUT"]
print("Kolom rekap:", [f.name() for f in rek.fields()])
for f in rek.getFeatures():
    print(f["NAMA_KPH"], "| jumlah petak berisi data:", f["count"], "| total pohon:", f["sum"], "| rata-rata:",
round(f["mean"], 1))

# 2. Petak yang datanya belum lengkap
kosong = [f["KODE_PETAK"] for f in lap.getFeatures() if f["inv_JML_POHON"] is None]
print("Petak tanpa data inventarisasi:", kosong)

# 3. Jati kelas umur Tua, diurutkan menurut volume per hektare
lap.selectByExpression("\"JENIS\" = 'Jati' AND \"KELAS_UMUR\" = 'Tua'")
print("Jati Tua terpilih:", lap.selectedFeatureCount())
for f in sorted(lap.selectedFeatures(), key=lambda x: -x["VOL_HA"]):
    print("  ", f["KODE_PETAK"], f["NAMA_KPH"], f["VOL_HA"], "m3/ha")

# 4. Ekspor yang terpilih ke GeoPackage dan ke CSV
processing.run("native:saveselectedfeatures", {"INPUT": lap, "OUTPUT": KELUAR + "/Jati_Tua.gpkg"})
with open(KELUAR + "/Jati_Tua.csv", "w", newline="", encoding="utf-8") as fh:
    w = csv.writer(fh)
    w.writerow(["KODE_PETAK", "NAMA_KPH", "LUAS_HA", "UMUR", "POHON_HA", "VOL_HA"])
    for f in lap.selectedFeatures():
        w.writerow([f["KODE_PETAK"], f["NAMA_KPH"], f["LUAS_HA"], f["UMUR"], f["POHON_HA"], f["VOL_HA"]])
hasil = QgsVectorLayer(KELUAR + "/Jati_Tua.gpkg", "Jati_Tua", "ogr")
print("Berkas Jati_Tua.gpkg:", hasil.featureCount(), "objek | Jati_Tua.csv ada:", os.path.exists(KELUAR +
"/Jati_Tua.csv"))
```

Penjelasan baris demi baris.

Baris	Tulisan	Artinya
1-3	komentar	Judul, penulis, dan pengingat urutan skrip.
4-7	`import csv`, `os`, `processing`, `QgsVectorLayer`	`csv` adalah perkakas untuk menulis berkas CSV.
9-11	`KELUAR`, `lap = ...`, `print("Baris:", ...)`	Buka layer hasil Skrip 15.6 dan cetak jumlah barisnya.
14-15	`qgis:statisticsbycategories`	Hitung statistik `inv_JML_POHON` per `NAMA_KPH`. Hasilnya layer sementara berisi kolom `count`, `sum`, `mean`, dan lain-lain.
16	`print("Kolom rekap:", ...)`	Cetak nama kolom hasil, agar Anda tahu kolom mana yang tersedia.
17-18	`for f in rek.getFeatures():`	Cetak tiap KPH: banyaknya petak berisi data, total pohon, dan rata-rata (satu desimal).
21-22	`kosong = [...]`	Daftar petak yang `inv_JML_POHON`-nya `None`.
25-26	`lap.selectByExpression(...)`	Pilih jati dengan kelas umur Tua, lalu cetak banyaknya.

Baris	Tulisan	Artinya
27-28	`sorted(..., key=lambda x: -x["VOL_HA"])`	Urutkan terpilih dari volume per hektare terbesar. Tanda minus membalik urutan, jadi yang besar di depan.
31	`native:saveselectedfeatures`	Simpan **hanya yang terpilih** ke `Jati_Tua.gpkg`.
32	`with open(..., "w", newline="", encoding="utf-8") as fh:`	Buka berkas CSV untuk ditulis. `newline=""` mencegah baris kosong ekstra di Windows. `encoding="utf-8"` menjaga huruf.
33-34	`w = csv.writer(fh), w.writerow([...])`	Buat penulis, lalu tulis baris judul kolom.
35-36	`for f in lap.selectedFeatures():`	Untuk tiap petak terpilih, tulis satu baris dengan enam isian. Urutannya harus sama dengan judul di baris 34.
37-38	`hasil = QgsVectorLayer(...), print(...)`	Baca balik `Jati_Tua.gpkg` dan cek `Jati_Tua.csv` ada.

Hasil. Layar menampilkan (uji pada QGIS 4.0.2):

```
HASIL
Baris: 16
Kolom rekap: ['NAMA_KPH', 'count', 'unique', 'min', 'max', 'range', 'sum', 'mean', 'median', 'stddev',
'minority', 'majority', 'q1', 'q3', 'iqr']
KPH Alpha | jumlah petak berisi data: 4 | total pohon: 3595.0 | rata-rata: 898.8
KPH Beta | jumlah petak berisi data: 4 | total pohon: 3475.0 | rata-rata: 868.8
KPH Gamma | jumlah petak berisi data: 7 | total pohon: 6192.0 | rata-rata: 884.6
Petak tanpa data inventarisasi: ['P-12']
Jati Tua terpilih: 5
   P-13 KPH Gamma 295.4 m3/ha
   P-05 KPH Alpha 257.7 m3/ha
   P-02 KPH Alpha 252.7 m3/ha
   P-01 KPH Alpha 244.4 m3/ha
   P-09 KPH Gamma 217.0 m3/ha
Berkas Jati_Tua.gpkg: 5 objek | Jati_Tua.csv ada: True
```

Pada sebagian komputer, dua baris peringatan `DeprecationWarning` tentang `parameterAsFields()` muncul sebelum hasil. Itu peringatan dari QGIS tentang alatnya, bukan galat, dan hasilnya tetap benar. Cek: 3595 + 3475 + 6192 = 13262, sama dengan total pohon di Skrip 15.6.

Bagian yang boleh diubah.

- **Baris 25, syarat seleksi.** Ganti jenis atau kelas umur.
- **Baris 27, tanda minus.** Hapus minus untuk mengurutkan dari terkecil.
- **Baris 34 dan 36, kolom CSV.** Daftar judul dan daftar isian harus sama panjang dan sama urutan.
- **Baris 31 dan 32, nama berkas hasil.** Ganti `Jati_Tua` sesuai isi.
- **Baris 15, kolom statistik dan kategori.** Ganti `inv_JML_POHON` menjadi `inv_VOLUME_M3` untuk rekap volume.

Jebakan.

- **Lupa Skrip 15.5 dan 15.6.** Layer tidak valid, dan baris 14 berhenti dengan: `_core.QgsProcessingException: Field "NAMA_KPH" does not exist.` Seperti pada Skrip 15.6, pesannya menyesatkan: yang kurang adalah berkasnya.
- **Nilai kosong mematahkan pengurutan.** Bila `VOL_HA` kosong (misalnya petak P-12 ikut terpilih), baris 27 berhenti dengan `TypeError: bad operand type for unary -: 'NoneType'`. Dari uji, ini terjadi saat hanya P-12 yang dipilih.
- Baris 18 memakai `round(f["mean"], 1)`. Bila `mean` kosong, ia gagal juga. [kemungkinan]
- Baris 37 membuka `Jati_Tua.gpkg` tanpa nama layer. Cukup bila hanya ada satu layer di berkas.
- **CSV memakai titik desimal.** Pembaca Indonesia yang membukanya di Excel berbahasa Indonesia mungkin melihat angka terbaca sebagai teks. [CEK]

Latihan variasi.

1. Pilih petak berjenis Sengon dan urutkan dari volume per hektare terkecil.
2. Ganti rekap per KPH menjadi rekap volume (`inv_VOLUME_M3`).

Contoh jawaban. Nomor 1: ganti baris 25 menjadi `lap.selectByExpression("\"JENIS\" = 'Sengon'")` dan hapus tanda minus di baris 27. Layar menampilkan empat petak: `P-15 93.5`, `P-11 113.6`, `P-04 117.4`, dan `P-07 128.1`. Tulisan "Jati Tua terpilih" tetap tercetak di baris 26 walau isinya Sengon, jadi ganti juga kata di dalamnya. Nomor 2: ganti `"inv_JML_POHON"` di baris 15 menjadi `"inv_VOLUME_M3"`. Dari uji, baris pertama rekap menjadi `KPH Alpha | jumlah petak berisi data: 4 | total pohon: 939.2 | rata-rata: 234.8`. Tulisan "total pohon" tetap tercetak walau isinya volume, jadi ganti juga katanya di baris 18.

> **Pola yang bisa dipakai ulang:** Rekap akhir hampir selalu: **statistik per kelompok**, **pilih yang penting**, **urutkan**, **ekspor ke GeoPackage dan CSV**. Ganti **kolom kelompok**, **syarat pilih**, dan **kolom yang diekspor**.

Cek paham

1. Apa bedanya seleksi dengan filter layer (`setSubsetString`)?
2. Mengapa Skrip 15.3 menyalin berkas dulu sebelum menggambar?
3. Mengapa join pertama di Skrip 15.6 hanya cocok 14 petak, padahal ada 15 petak yang punya data?
4. Mengapa Skrip 15.7 gagal bila Skrip 15.5 dan 15.6 belum dijalankan?

Jawaban.

1. Seleksi hanya menandai fitur yang terpilih, sedangkan fitur lain tetap ada. Filter menyembunyikan fitur lain sampai filternya dikosongkan.
2. Agar berkas asli di paket data tidak berubah. Semua edit terjadi di salinan.
3. Kunci `p-09` di tabel CSV ditulis huruf kecil, sehingga tidak sama dengan `P-09`. Setelah dirapikan dengan `upper` dan `trim`, 15 petak cocok.

4. Skrip 15.7 membaca `Petak_Inventarisasi.gpkg`, yang dibuat oleh Skrip 15.6 dari hasil Skrip 15.5. Tanpa berkas itu, layar tidak valid dan alat gagal.

Kesalahan umum

- **Membiarkan jalur contoh.** Jalur cadangan menunjuk folder di komputer penulis. Perbaikan: ganti jalur cadangan, atau isi pengaturan `B2_DATA` dan `B2_KELUAR`.
- **Salah ketik nama kolom di ekspresi.** Seleksi tidak galat, hanya memilih 0 fitur. Perbaikan: salin nama kolom dari hasil Skrip 15.1.
- **Menjalankan skrip tidak berurutan.** Skrip 15.6 dan 15.7 memakai berkas buatan skrip sebelumnya. Perbaikan: jalankan 15.5, 15.6, lalu 15.7.
- Lupa `commitChanges()` setelah edit. Perubahan tidak tersimpan. Perbaikan: akhiri tiap edit dengan `commitChanges()` dan baca hasilnya (`True` atau `False`).
- **Percaya join tanpa memeriksa kunci.** Perbaikan: selalu cetak yang tidak cocok, seperti baris 24 dan 26 di Skrip 15.6.

Ringkasan dan latihan

- Layer vektor punya tabel atribut. Skrip membacanya per **fitur** dan per **kolom**.
- `selectByExpression` menandai fitur dengan syarat, dan `selectbylocation` memilih menurut lokasi.
- Mengubah layer lewat skrip: salin, `startEditing`, tambah atau ubah, `commitChanges`.
- Alat pemeriksa topologi menolak bentuk rusak. Perbaiki bentuk lebih dulu, lalu periksa ulang.
- Tambah kolom lewat `native:fieldcalculator`, tempel tabel lewat `native:joinattributestable`.
- Rekap akhir: statistik per kelompok, seleksi, urut, lalu ekspor ke GeoPackage dan CSV.
- Tiga hal yang diganti untuk data baru: lokasi data, nama berkas dan layer, nama kolom.

Latihan. Buka `Titik_Plot.gpkg` dengan layer `Titik_Plot`. Cetak nama kolomnya saja, tanpa baris datanya. Lalu pilih plot sehat dan berdiameter 40 cm ke atas, dan cetak kodenya. Bagian mana dari Skrip 15.1 yang boleh Anda hapus?

Tabel perbandingan: pekerjaan di menu dan padanannya di skrip

Pekerjaan di QGIS	Padanan di skrip	Contoh di KPH Contoh
Membuka tabel atribut	`layer.getFeatures()`	Membaca 16 petak
Select by Expression	`layer.selectByExpression("...")`	Jati sebelum 2010: 4 petak
Select by Location	`native:selectbylocation`	22 plot di dalam petak
Toggle Editing, lalu simpan	`startEditing()`, `commitChanges()`	Jalan dan persemaian baru
Split Features	`layer.splitFeatures(garis, ...)`	P-12 menjadi dua bagian

Pekerjaan di QGIS	Padanan di skrip	Contoh di KPH Contoh
Geometry Checker	`native:checkvalidity` dan kawan-kawannya	P-18 menyilang diri
Field Calculator	`native:fieldcalculator`	Kolom `LUAS_HA` dan `UMUR`
Join Attributes by Field Value	`native:joinattributestable`	Pohon dan volume per petak
Statistics by Categories	`qgis:statisticsbycategories`	Total pohon per KPH
Export Selected Features	`native:saveselectedfeatures`	`Jati_Tua.gpkg`

BAB 16: Kisi Piksel yang Bisa Dihitung: Raster dan Citra dengan Skrip

> **Berdasarkan Seri B3:** Skrip bab ini berasal dari Buku B3, *Raster dan Citra Dasar* (Basic Class), berkas `b3_01_info.py` sampai `b3_07_proyek.py`. Di buku itu, pekerjaan raster dilakukan lewat menu dan klik. Bab ini menunjukkan sisi skripnya, dan ketujuh skrip dibahas apa adanya.

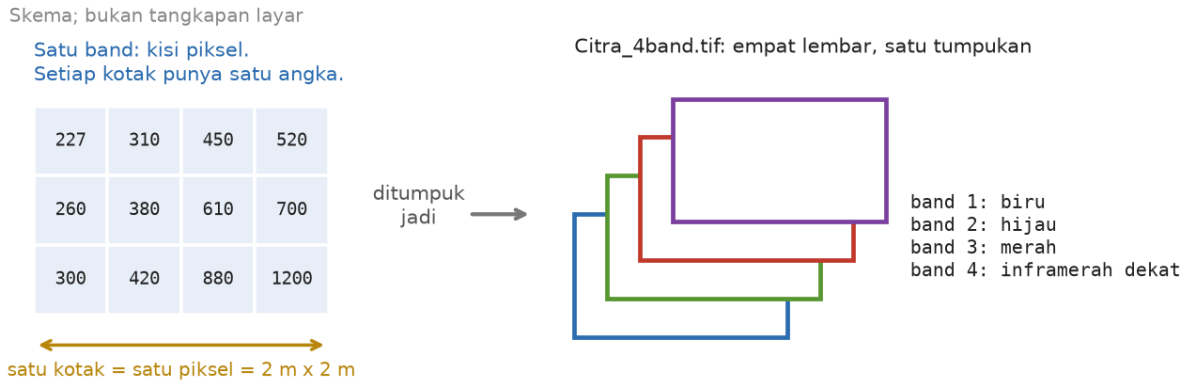
Studi kasus: "Seberapa hijau KPH kita, dan berapa luas lereng curamnya?"

Kepala Seksi membawa citra satelit dan peta tinggi KPH Contoh. Pertanyaannya dua: seberapa hijau tiap KPH, dan berapa hektare lahan yang lerengnya curam? Anda bisa menjawabnya dengan banyak klik di QGIS. Dengan skrip, satu kali jalan sudah cukup, dan Anda bisa mengulanginya untuk citra tahun depan. Tujuh skrip di bab ini membawa Anda dari memeriksa isi citra sampai tabel ringkasan per KPH.

Konsep: raster dalam tiga kalimat

Raster adalah gambar yang tersusun dari kotak-kotak kecil bernama **piksel**, seperti keramik di lantai. Setiap kotak menyimpan satu angka, misalnya kecerahan cahaya atau tinggi tanah. Satu lembar angka disebut **band**, dan citra satelit biasanya terdiri dari beberapa band yang ditumpuk.

Ilustrasi 16.1: Raster: kisi piksel dan band



Lima istilah yang perlu Anda kenal sekarang:

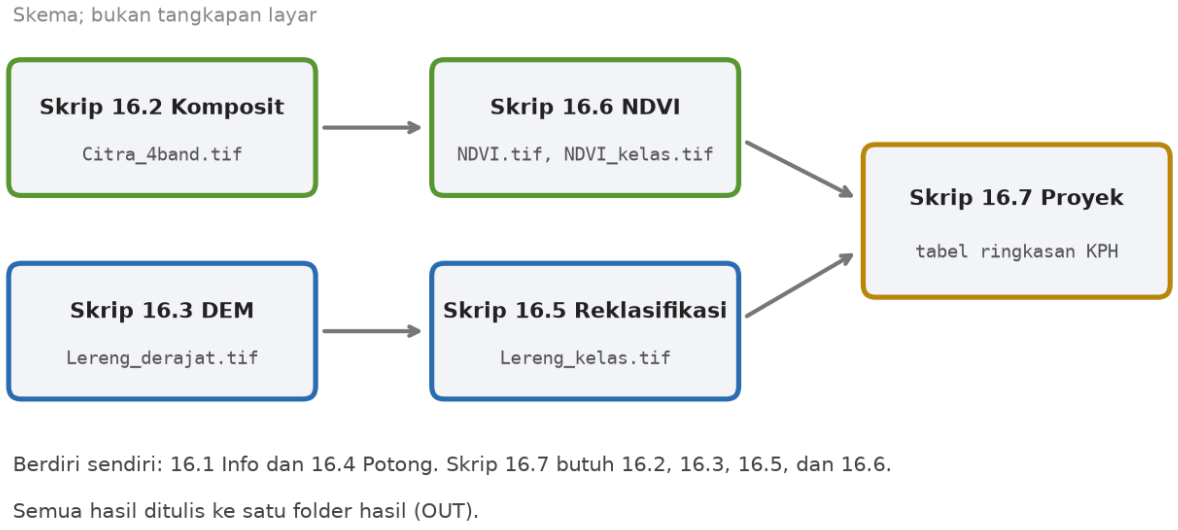
- **Piksel**: satu kotak raster. Pada data latihan ini, satu piksel mewakili 2 m x 2 m di lapangan.
- **Band**: satu lembar angka. Citra latihan memakai biru, hijau, merah, dan inframerah dekat.
- **DEM** (*Digital Elevation Model*): raster yang angkanya adalah tinggi tanah.
- **CRS**: sistem koordinat yang dipakai layer. Data latihan memakai UTM zona 49 Selatan (EPSG:32749), satuannya meter.

- **NoData**: nilai penanda "kotak ini kosong". Kotak seperti ini tidak ikut dihitung.

Peta jalan bab ini:

Skrip	Berkas	Pekerjaannya
16.1	`b3_01_info.py`	Memeriksa isi sebuah citra
16.2	`b3_02_komposit.py`	Menumpuk empat band menjadi satu citra
16.3	`b3_03_dem.py`	Membuat bayangan bukit, lereng, arah lereng, dan kontur
16.4	`b3_04_clip.py`	Memotong raster dengan poligon
16.5	`b3_05_reklas.py`	Mengelompokkan lereng menjadi tiga kelas
16.6	`b3_06_ndvi.py`	Menghitung indeks vegetasi NDVI
16.7	`b3_07_proyek.py`	Membuat tabel ringkasan per KPH

Ilustrasi 16.2: Rantai skrip bab 16



Skrip 16.1 dan 16.4 berdiri sendiri. Skrip 16.7 membutuhkan hasil 16.2, 16.3, 16.5, dan 16.6, jadi jalankan semuanya berurutan.

Dua baris lokasi di awal tiap skrip. Skrip bab ini membaca lokasi data dan folder hasil dari pengaturan sistem (variabel lingkungan, lihat Bab 8). Bila pengaturan itu tidak ada, skrip memakai alamat cadangan. Alamat cadangan hanya contoh, bukan alamat di komputer Anda:

```python
DATA = os.environ.get("B3_DATA", r"C:/KPH_Contoh/paket-b3")
OUT = os.environ.get("B3_OUT", r"C:/kerja_b3")
os.makedirs(OUT, exist_ok=True)
```

Ada dua cara menyesuaikannya. Cara pertama, atur variabel lingkungan `B3_DATA` dan `B3_OUT` ke lokasi di komputer Anda. Cara kedua, ganti alamat cadangan di ujung baris dengan lokasi folder `paket-b3` dan folder hasil Anda. Bila memilih cara kedua, ubah di setiap skrip dengan isi yang sama.

Baris `os.makedirs(OUT, exist_ok=True)` membuat folder hasil bila belum ada (lihat Bab 8). Anda tidak perlu membuatnya lebih dulu. Skrip 16.5 hanya memakai `OUT`, jadi tidak punya baris `DATA`. Skrip 16.7 hanya membaca hasil skrip lain, jadi tidak membuat folder.

Skrip 16.1: Memeriksa isi sebuah citra

Sebelum memakai citra, kenali dulu seperti Anda mengecek label kemasan. Berkasnya `b3_01_info.py`; di dalamnya tertulis "Skrip 1.1".

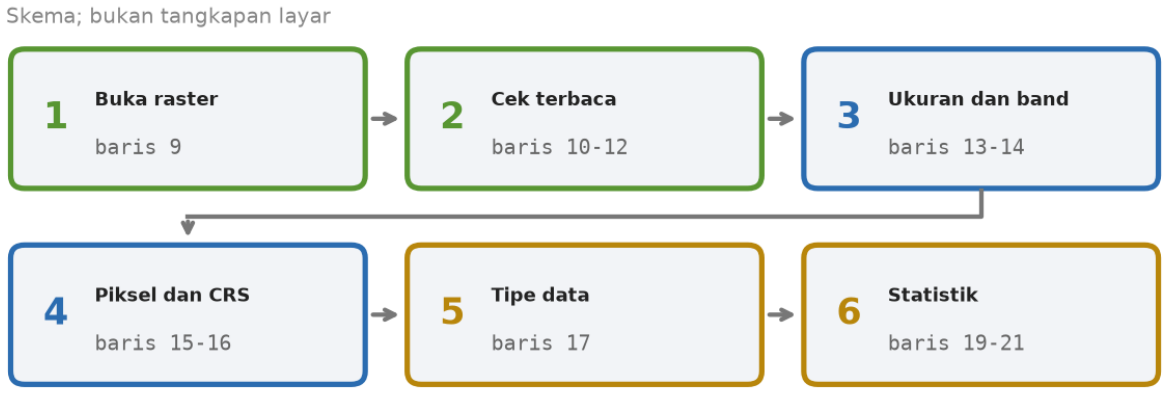
Tujuan. Memeriksa bahwa citra terbaca, lalu menampilkan ukuran, jumlah band, ukuran piksel, CRS, tipe data, dan statistik dasarnya.

Bahan. Paket data `paket-b3`, berkas `Citra_B04.tif` (band merah). Skrip memakai `processing`, alat bawaan Konsol Python QGIS. Cara menjalankannya ada di Bab 3.

Istilah baru di skrip ini:

- `QgsRasterLayer`: Perkakas pembuka raster.
- **tipe data**: Jenis angka yang disimpan tiap piksel.
- **statistik raster**: Angka terkecil, terbesar, dan rata-rata.
- **kamus hasil**: Alat mengembalikan pasangan nama dan nilai.
- `raise SystemExit("pesan")`: Menghentikan skrip seketika dan menampilkan pesan. Mirip `raise` di Bab 10.
- `os.environ.get(...)` dan `import os`: Baca pengaturan sistem, lihat Bab 8.

Ilustrasi 16.3: Alur skrip 16.1



Skrip utuh:

```python
# Skrip 1.1: memeriksa isi sebuah raster (jalankan di Python Console QGIS). Penulis: Badar Mubarok Yogaswara
# Ubah DATA ke folder paket-b3 di komputer Anda.
import os
import processing
from qgis.core import QgsRasterLayer

DATA = os.environ.get("B3_DATA", r"C:/KPH_Contoh/paket-b3")
```

```
lay = QgsRasterLayer(DATA + "/Citra_B04.tif", "Citra_B04")
if not lay.isValid():
    raise SystemExit("Raster tidak terbaca. Periksa DATA: " + DATA)
print("Terbaca:", lay.isValid())
print("Ukuran (kolom x baris):", lay.width(), "x", lay.height())
print("Jumlah band:", lay.bandCount())
print("Ukuran piksel (m):", lay.rasterUnitsPerPixelX(), "x", lay.rasterUnitsPerPixelY())
print("CRS:", lay.crs().authid())
print("Tipe data band 1:", lay.dataProvider().dataType(1).name)

hasil = processing.run("native:rasterlayerstatistics", {"INPUT": lay, "BAND": 1})
print("Jumlah piksel:", hasil["COUNT"])
print("Min:", hasil["MIN"], " Maks:", hasil["MAX"], " Rata-rata:", round(hasil["MEAN"], 1))
```

Penjelasan baris demi baris.

Baris	Tulisan	Artinya
1-2	`# Skrip 1.1 ...`	Komentar: judul, penulis, dan pengingat mengganti `DATA`.
3-5	`import os`, `import processing`, `from qgis.core import QgsRasterLayer`	Panggil perkakas sistem (`os`), perkakas alat (`processing`), dan pembuka raster.
7	`DATA = os.environ.get("B3_DATA", r"C:/KPH_Contoh/paket-b3")`	Lokasi folder data. Bila pengaturan `B3_DATA` ada, nilainya dipakai. Bila tidak, dipakai alamat cadangan. Huruf `r` membuat teks dibaca apa adanya. **Atur atau ganti dengan lokasi Anda.**
9	`lay = QgsRasterLayer(DATA + "/Citra_B04.tif", "Citra_B04")`	Buka citra dan simpan di kotak `lay`. Isian pertama adalah jalur lengkap (lokasi disambung dengan `+` ke nama berkas). Isian kedua nama layer di QGIS.
10-11	`if not lay.isValid(): raise SystemExit(...)`	Pengaman. Bila citra gagal dibuka, skrip berhenti dengan pesan yang menyebut isi `DATA`. Baris-baris di bawahnya tidak dijalankan.
12	`print("Terbaca:", lay.isValid())`	`True` bila citra berhasil dibuka. Setelah pengaman, hasilnya selalu `True`.
13	`lay.width()`, `lay.height()`	Lebar dan tinggi, dihitung dalam piksel.
14	`lay.bandCount()`	Jumlah band.
15	`rasterUnitsPerPixelX()` dan `...Y()`	Ukuran satu piksel menurut satuan CRS: lebar lalu tinggi.
16	`lay.crs().authid()`	Kode sistem koordinat, misalnya `EPSG:32749`.
17	`lay.dataProvider().dataType(1).name`	Tipe data band nomor 1. Penomoran band mulai dari 1, bukan 0.
19	`processing.run("native:rasterlayerstatistics", {...})`	Jalankan alat statistik raster. Isiannya berupa kamus: `INPUT` (layer) dan `BAND` (band yang dihitung). Hasilnya disimpan di kotak `hasil`.
20	`hasil["COUNT"]`	Ambil jumlah piksel dari kamus hasil.
21	`hasil["MIN"]`, `hasil["MAX"]`, `round(hasil["MEAN"], 1)`	Ambil nilai terkecil, terbesar, dan rata-rata (dibulatkan satu desimal).

Hasil. Layar menampilkan (uji pada QGIS 4.0.2, dengan `DATA` diganti ke folder salinan paket data):

```
HASIL
Terbaca: True
Ukuran (kolom x baris): 200 x 200
Jumlah band: 1
Ukuran piksel (m): 2.0 x 2.0
CRS: EPSG:32749
Tipe data band 1: UInt16
Jumlah piksel: 40000
Min: 227.0  Maks: 2061.0  Rata-rata: 449.0
```

Dari hasil ini Anda tahu: citra berukuran 200 x 200 piksel, jadi 40000 kotak. Tiap piksel 2 meter, sehingga citra mencakup 400 m x 400 m, yaitu 16 hektare. Tipe `UInt16` berarti bilangan bulat tanpa tanda minus, 16 bit.

Bagian yang boleh diubah.

- **Baris 7, lokasi data.** Atur `B3_DATA` atau ganti alamat cadangan ke komputer Anda.
- **Baris 9, nama berkas.** Ganti `Citra_B04.tif` untuk memeriksa citra lain. Dari uji: pada `Citra_B08.tif` (inframerah dekat) hasilnya min 160.0, maks 4878.0, rata-rata 3494.4.
- **Baris 17 dan 19, nomor band.** Pada citra empat band (Skrip 16.2), ganti angka 1 untuk memeriksa band lain.

Jebakan.

- **Jalur salah menghentikan skrip.** Bila `DATA` menunjuk folder yang salah atau kosong, baris 10-11 menghentikan skrip dengan pesan `Raster tidak terbaca. Periksa DATA:` diikuti isi `DATA`. Tanpa pengaman itu, skrip terus berjalan dan mencetak angka janggal: `Terbaca: False`, ukuran 0, dan nilai terkecil serta terbesar berupa angka raksasa. Bila Anda melihat pesan pengaman, periksa baris 7 dan nama berkas di baris 9.
- **Nomor band berlebih.** Dari uji, mengganti `BAND` menjadi 2 pada citra satu band berhenti dengan `QgsProcessingException: Invalid band number for BAND (2): Valid values for input raster are 1 to 1.`
- **Satuan tidak dicek.** Teks "(m)" di baris 15 hanya tulisan. Skrip tidak memeriksa apakah CRS memakai meter. Pada CRS berderajat, angkanya derajat.

Latihan variasi.

1. Periksa `Citra_B08.tif`, bukan `Citra_B04.tif`.
2. Tambahkan satu baris yang mencetak luas citra dalam hektare.

Contoh jawaban. Nomor 1: ganti nama berkas di baris 9 dan nama layer di sebelahnya. Layar menampilkan min `160.0`, maks `4878.0`, dan rata-rata `3494.4`. Nomor 2, tambahkan di akhir skrip:

```
PYTHON
print("Luas citra (ha):", hasil["COUNT"] * lay.rasterUnitsPerPixelX() * lay.rasterUnitsPerPixelY() / 10000)
```

Untuk `Citra_B04.tif`, layar menampilkan `Luas citra (ha): 16.0`.

> **Pola yang bisa dipakai ulang:** Pemeriksaan raster: `lay = QgsRasterLayer(JALUR, NAMA)`, lalu `lay.isValid()`, lalu ukuran dan band, lalu `processing.run("native:rasterlayerstatistics", {"INPUT": lay, "BAND": 1})`. Ganti **lokasi data**, **nama berkas**, dan **nomor band**.

Skrip 16.2: Menumpuk empat band menjadi satu citra

Empat berkas satu-band ibarat empat lembar transparan. Skrip ini menumpuknya menjadi satu berkas empat lembar, supaya mudah dirakit menjadi gambar berwarna atau dihitung bersama. Berkasnya `b3_02_komposit.py`; di dalamnya tertulis "Skrip 2.1".

Tujuan. Menggabungkan citra biru, hijau, merah, dan inframerah dekat menjadi satu citra empat band bernama `Citra_4band.tif`.

Bahan. Paket `paket-b3`: `Citra_B02.tif`, `Citra_B03.tif`, `Citra_B04.tif`, dan `Citra_B08.tif`. Folder hasil `OUT` dibuat otomatis bila belum ada. Hasilnya dipakai Skrip 16.6.

Istilah baru di skrip ini:

- `os.path.exists` dan `os.remove`: Memeriksa apakah berkas ada, lalu menghapusnya. `os.path.exists` dibahas di Bab 8.
- **komposit**: Citra multi-band.
- **urutan band**: Nomor band ditentukan urutan berkas.
- **GDAL**: Pustaka pengolah raster di dalam QGIS.
- **daftar**: List, Bab 5.

Ilustrasi 16.4: Alur skrip 16.2



Skrip utuh:

```python
# Skrip 2.1: menumpuk empat band menjadi satu raster multiband. Penulis: Badar Mubarok Yogaswara
import os
import processing
from qgis.core import QgsRasterLayer

DATA = os.environ.get("B3_DATA", r"C:/KPH_Contoh/paket-b3")
```

```
OUT = os.environ.get("B3_OUT", r"C:/kerja_b3")          # folder hasil (dibuat otomatis bila belum ada)
os.makedirs(OUT, exist_ok=True)

# Urutan penting: band 1 = biru (B02), 2 = hijau (B03), 3 = merah (B04), 4 = inframerah dekat (B08)
urut = [DATA + "/Citra_B02.tif", DATA + "/Citra_B03.tif", DATA + "/Citra_B04.tif", DATA + "/Citra_B08.tif"]
HASIL = OUT + "/Citra_4band.tif"
if os.path.exists(HASIL):
    os.remove(HASIL)          # gdal:merge tidak menimpa hasil lama
hasil = processing.run("gdal:merge", {
    "INPUT": urut,
    "SEPARATE": True,         # satu berkas = satu band
    "DATA_TYPE": 2,           # 2 = UInt16, sama dengan data asal
    "OUTPUT": OUT + "/Citra_4band.tif"})
stack = QgsRasterLayer(hasil["OUTPUT"], "Citra_4band")
if not stack.isValid():
    raise SystemExit("Citra_4band.tif tidak terbentuk. Periksa DATA dan OUT.")
print("Jumlah band:", stack.bandCount(), "| tipe:", stack.dataProvider().dataType(1).name)
print("Warna asli  : merah=band 3, hijau=band 2, biru=band 1")
print("Warna palsu : merah=band 4, hijau=band 3, biru=band 2")
```

Penjelasan baris demi baris.

Baris	Tulisan	Artinya
1-4	komentar dan `import`	Judul, penulis, `os`, `processing`, dan `QgsRasterLayer`.
6-7	`DATA = ..., OUT = ...`	Lokasi data dan folder hasil, dari pengaturan `B3_DATA` dan `B3_OUT` atau alamat cadangan. **Atur atau ganti keduanya.** Komentar di baris 7: folder hasil dibuat otomatis.
8	`os.makedirs(OUT, exist_ok=True)`	Buat folder hasil bila belum ada. Bila sudah ada, tidak terjadi apa-apa.
10	komentar	Pengingat: urutan penting. Band 1 biru (B02), 2 hijau (B03), 3 merah (B04), 4 inframerah dekat (B08).
11	`urut = [... , ... , ... , ...]`	Daftar empat jalur berkas, urutannya menentukan nomor band.
12-14	`HASIL = ..., if os.path.exists(HASIL): os.remove(HASIL)`	Pengaman. Bila `Citra_4band.tif` lama sudah ada, hapus dulu. Alat `gdal:merge` tidak menimpa berkas lama, sehingga tanpa baris ini hasil lama bisa tetap dipakai tanpa pesan.
15-19	`processing.run("gdal:merge", {...})`	Jalankan alat penggabung dari GDAL. `INPUT` adalah daftar berkas. `SEPARATE: True` berarti satu berkas menjadi satu band, bukan ditempel menyamping. `DATA_TYPE: 2` memilih tipe UInt16, sama dengan data asal. `OUTPUT` adalah jalur berkas hasil.
20	`stack = QgsRasterLayer(hasil["OUTPUT"], "Citra_4band")`	Buka hasil penggabungan sebagai layer.
21-22	`if not stack.isValid(): raise SystemExit(...)`	Pengaman. Bila berkas hasil tidak terbentuk atau tidak terbaca, skrip berhenti dengan pesan `Citra_4band.tif tidak terbentuk. Periksa DATA dan OUT.`
23	`print("Jumlah band:", ...)`	Cetak jumlah band dan tipe band 1.

Baris	Tulisan	Artinya
24-25	`print("Warna asli ...")`	Dua baris catatan biasa. Skrip tidak menghitung apa pun di sini: ia hanya mengingatkan nomor band untuk gambar warna asli dan warna palsu.

Hasil. Layar menampilkan (uji pada QGIS 4.0.2):

```
HASIL
Jumlah band: 4 | tipe: UInt16
Warna asli  : merah=band 3, hijau=band 2, biru=band 1
Warna palsu : merah=band 4, hijau=band 3, biru=band 2
```

Bukti urutan band benar: rata-rata band 1 sampai 4 pada hasil, yaitu 293.9, 607.0, 449.0, dan 3494.4, sama dengan rata-rata berkas B02, B03, B04, dan B08. Keempatnya diukur dengan alat statistik pada Skrip 16.1.

Bagian yang boleh diubah.

- **Baris 6-7, lokasi.** Atur `B3_DATA` dan `B3_OUT`, atau ganti alamat cadangannya.
- **Baris 11, urutan dan isi daftar.** Urutan menentukan nomor band. Dari uji: membuang `Citra_B08.tif` menghasilkan `Jumlah band: 3`.
- Baris 18, `DATA_TYPE`. Angka adalah nomor pilihan pada alat: 0 Byte, 1 Int16, 2 UInt16, 3 UInt32, 4 Int32, 5 Float32. Urutan ini berlaku untuk `gdal:merge`. [CEK] pada versi QGIS Anda.
- **Baris 12 dan 19, nama berkas hasil.** Bila diganti, ubah di kedua baris, dan Skrip 16.6 harus ikut diubah.

Jebakan.

- **Urutan band salah tanpa pesan.** Dari uji: menukar B08 dan B04 di baris 11 tetap menghasilkan 4 band tanpa galat. Akibatnya terlihat di Skrip 16.6, yaitu NDVI terbalik tanda.
- **Berkas hasil lama dihapus otomatis.** Baris 13-14 menghapus `Citra_4band.tif` yang lama setiap skrip dijalankan. Tanpa baris itu, `gdal:merge` tidak menimpa berkas lama. Contohnya hasil percobaan tiga band tetap dipakai walau Anda memberi empat berkas, dan Skrip 16.6 kemudian gagal. Akibat baris ini: bila Anda menyimpan berkas lain dengan nama yang sama, berkas itu ikut terhapus. Pakai nama lain di baris 12 dan 19 bila ingin menyimpannya.
- **Pesan pengaman.** Bila baris 22 menghentikan skrip, periksa `DATA` dan `OUT`, lalu pastikan keempat berkas `Citra_B02`, `B03`, `B04`, dan `B08` ada.

Latihan variasi.

1. Gabungkan hanya tiga band (biru, hijau, merah).
2. Pastikan hasilnya benar dengan mencetak rata-rata band 3.

Contoh jawaban. Nomor 1: hapus `, DATA + "/Citra_B08.tif"` dari baris 11. Berkas hasil lama dihapus otomatis oleh skrip. Layar menampilkan `Jumlah band: 3 | tipe: UInt16`. Nomor 2: tambahkan di akhir skrip:

```
PYTHON
st = processing.run("native:rasterlayerstatistics", {"INPUT": stack, "BAND": 3})
```

```
print("Rata-rata band 3:", round(st["MEAN"], 1))
```

Layar menampilkan `Rata-rata band 3: 449.0`, sama dengan `Citra_B04.tif`.

> **Pola yang bisa dipakai ulang:** Menumpuk band: `processing.run("gdal:merge", {"INPUT": [BERKAS1, BERKAS2, ...], "SEPARATE": True, "DATA_TYPE": KODE, "OUTPUT": HASIL})`. Ganti **daftar berkas** (urutan = nomor band), **kode tipe data**, dan **jalur hasil**.

Skrip 16.3: Dari satu peta tinggi menjadi empat turunan

Satu DEM bisa melahirkan banyak peta: bayangan bukit, kemiringan, arah hadap lereng, dan garis kontur. Skrip ini membuat keempatnya sekaligus. Berkasnya `b3_03_dem.py`; di dalamnya tertulis "Skrip 3.1".

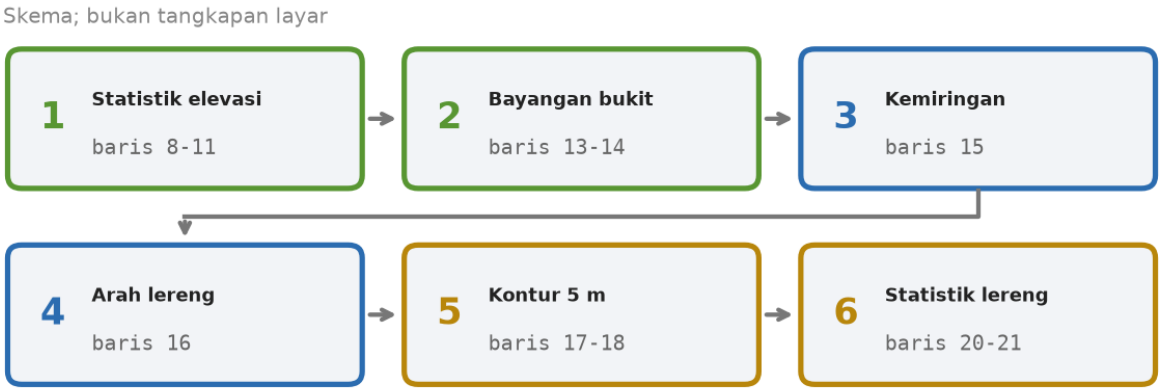
Tujuan. Dari `DEM_KPH.tif`, membuat `Hillshade.tif`, `Lereng_derajat.tif`, `Arah_lereng.tif`, dan `Kontur_5m.gpkg`, lalu mencetak statistik elevasi dan lereng.

Bahan. Paket `paket-b3`: `DEM_KPH.tif`. Folder hasil `OUT` dibuat otomatis bila belum ada. `Lereng_derajat.tif` dipakai Skrip 16.5.

Istilah baru di skrip ini:

- **hillshade**: Bayangan bukit, seolah matahari menyinari relief.
- **kemiringan**: *slope*, derajat.
- **arah lereng**: *aspect*, ke mana lereng menghadap.
- **kontur**: Garis tinggi sama.
- **azimut**: Arah datang cahaya, dalam derajat.

Ilustrasi 16.5: Alur skrip 16.3



Skrip utuh:

```
PYTHON
# Skrip 3.1: hillshade, kemiringan, arah lereng, dan kontur dari DEM. Penulis: Badar Mubarok Yogaswara
import os
import processing
```

```
DATA = os.environ.get("B3_DATA", r"C:/KPH_Contoh/paket-b3")
OUT = os.environ.get("B3_OUT", r"C:/kerja_b3")
os.makedirs(OUT, exist_ok=True)
dem = DATA + "/DEM_KPH.tif"

st = processing.run("native:rasterlayerstatistics", {"INPUT": dem, "BAND": 1})
print("Elevasi min/maks (m):", round(st["MIN"], 2), "/", round(st["MAX"], 2))

processing.run("native:hillshade", {"INPUT": dem, "Z_FACTOR": 1, "AZIMUTH": 300, "V_ANGLE": 40,
                                    "OUTPUT": OUT + "/Hillshade.tif"})
processing.run("native:slope", {"INPUT": dem, "Z_FACTOR": 1, "OUTPUT": OUT + "/Lereng_derajat.tif"})
processing.run("native:aspect", {"INPUT": dem, "Z_FACTOR": 1, "OUTPUT": OUT + "/Arah_lereng.tif"})
processing.run("gdal:contour", {"INPUT": dem, "BAND": 1, "INTERVAL": 5, "FIELD_NAME": "ELEV",
                                "OUTPUT": OUT + "/Kontur_5m.gpkg"})

sl = processing.run("native:rasterlayerstatistics", {"INPUT": OUT + "/Lereng_derajat.tif", "BAND": 1})
print("Lereng min/maks/rata-rata (derajat):", round(sl["MIN"], 2), "/", round(sl["MAX"], 2), "/",
round(sl["MEAN"], 2))
```

Penjelasan baris demi baris.

Baris	Tulisan	Artinya
1-3	komentar, `import os`, `import processing`	Judul, penulis, perkakas sistem `os`, dan perkakas `processing`.
5-6	`DATA`, `OUT`	Lokasi data dan folder hasil, dari pengaturan `B3_DATA` dan `B3_OUT` atau alamat cadangan. **Atur atau ganti keduanya.**
7	`os.makedirs(OUT, exist_ok=True)`	Buat folder hasil bila belum ada. Tanpa baris ini, alat di bawah gagal menulis hasil.
8	`dem = DATA + "/DEM_KPH.tif"`	Simpan jalur DEM di kotak `dem`, supaya dipakai berkali-kali.
10-11	`native:rasterlayerstatistics`	Hitung statistik DEM dan cetak elevasi terkecil dan terbesar, dibulatkan dua desimal. Masukannya cukup jalur berkas.
13-14	`native:hillshade`	Buat bayangan bukit. `Z_FACTOR` pengali tinggi. `AZIMUTH: 300` arah cahaya, dan `V_ANGLE: 40` tinggi cahaya di atas cakrawala. Hasil ditulis ke `OUT`. Kamus isian boleh ditulis di dua baris, asal kurungnya seimbang.
15	`native:slope`	Hitung kemiringan dalam derajat.
16	`native:aspect`	Hitung arah lereng.
17-18	`gdal:contour`	Buat garis kontur tiap 5 meter (`INTERVAL: 5`). Nilai tinggi disimpan di kolom `ELEV`. Hasilnya berkas vektor GeoPackage, bukan raster.
20-21	statistik lereng	Hitung statistik `Lereng_derajat.tif` yang baru dibuat, lalu cetak terkecil, terbesar, dan rata-rata.

Hasil. Layar menampilkan (uji pada QGIS 4.0.2):

```
HASIL
Elevasi min/maks (m): 40.03 / 74.25
Lereng min/maks/rata-rata (derajat): 0.01 / 21.89 / 6.39
```

Skrip ini hanya mencetak dua baris. Hasil utamanya berupa empat berkas di folder `OUT`. Dari uji, `Kontur_5m.gpkg` berisi 17 garis kontur, dengan kolom `fid`, `ID`, dan `ELEV`. Lereng terbesar 21,89 derajat, jadi lahan ini landai sampai agak curam.

Bagian yang boleh diubah.

- Baris 13, `AZIMUTH` dan `V_ANGLE`. Arah dan tinggi cahaya. Ubah untuk menggeser bayangan. Penulis memakai 300 dan 40. Nilai lazim di banyak peta adalah 315 dan 45. [CEK]
- Baris 13, 15, dan 16, `Z_FACTOR`. Pengali tinggi. Angka 1 benar bila satuan tinggi dan satuan horizontal sama (sama-sama meter).
- Baris 17, `INTERVAL`. Jarak antar kontur. Dari uji: `10` menghasilkan 6 garis kontur, bukan 17.
- Baris 17, `FIELD_NAME`. Nama kolom tinggi pada garis kontur.

Jebakan.

- **Folder hasil dibuat oleh baris 7.** Tanpa `os.makedirs`, alat pertama yang menulis hasil gagal dengan pesan GDAL `ERROR 4 ... No such file or directory`. Skrip lalu berhenti dengan `Could not load source layer ... not found`, yang hanya akibat berkas tidak pernah dibuat. Dengan baris 7, folder hasil terbentuk sendiri.
- **Satuan tinggi dan horizontal berbeda.** Bila DEM Anda memakai satuan tinggi yang berbeda dari satuan jarak, `Z_FACTOR` harus disesuaikan. Skrip selalu memakai 1 dan tidak membahasnya. [CEK]
- **Nilai kembalian tidak disimpan.** Baris 13-18 tidak menyimpan hasil `processing.run` di kotak. Tidak masalah karena hasilnya sudah ke berkas.
- **Hasil lama diganti.** Dari uji, menjalankan ulang memperbarui `Hillshade.tif` (waktu simpan berubah). Berbeda dengan `gdal:merge` di Skrip 16.2.

Latihan variasi.

1. Buat kontur tiap 10 meter, bukan 5.
2. Ganti arah cahaya ke 315 dengan tinggi 45 derajat.

Contoh jawaban. Nomor 1: ganti `5` menjadi `10` di baris 17 (dan nama `Kontur_5m` di baris 18 menjadi `Kontur_10m` agar tidak menimpa). Hasilnya berkas dengan 6 garis kontur. Nomor 2: ganti `300` menjadi `315` dan `40` menjadi `45` di baris 13. Layar tidak berubah, tetapi `Hillshade.tif` berbeda. Bandingkan keduanya dengan menampilkannya di QGIS.

> **Pola yang bisa dipakai ulang:** Turunan DEM selalu satu alat satu baris: `processing.run("native:ALAT", {"INPUT": DEM, "Z_FACTOR": 1, "OUTPUT": HASIL})`. Ganti **nama alat**, **parameter khusus** (azimut, interval), dan **jalur hasil**.

Skrip 16.4: Memotong raster dengan poligon

Memotong raster seperti menggunting kain mengikuti pola. Bagian di luar pola dibuang, atau ditandai kosong. Berkasnya `b3_04_clip.py`; di dalamnya tertulis "Skrip 4.1".

Tujuan. Memotong DEM dengan poligon `Area_Kajian`, menghitung piksel berisi dan luasnya, lalu memotong DEM dengan satu KPH saja.

Bahan. Paket `paket-b3`: `DEM_KPH.tif`, `Area_Kajian.gpkg`, dan `Batas_KPH.gpkg` (layer `Batas_KPH` dengan kolom `NAMA_KPH`). Folder hasil `OUT` dibuat otomatis bila belum ada.

Istilah baru di skrip ini:

- **mask**: Poligon pemotong.
- **crop to cutline**: Kotak raster dipangkas mengikuti poligon.
- **nilai unik**: Daftar nilai yang ada beserta jumlahnya.
- `setSubsetString`: Filter layer, sama dengan Skrip 15.2.

Ilustrasi 16.6: Alur skrip 16.4



Skrip utuh:

```python
# Skrip 4.1: memotong raster dengan poligon (clip by mask). Penulis: Badar Mubarok Yogaswara
import os
import processing
from qgis.core import QgsVectorLayer

DATA = os.environ.get("B3_DATA", r"C:/KPH_Contoh/paket-b3")
OUT = os.environ.get("B3_OUT", r"C:/kerja_b3")
os.makedirs(OUT, exist_ok=True)

# 1) Potong DEM dengan Area_Kajian (poligon tidak beraturan)
hasil = processing.run("gdal:cliprasterbymasklayer", {
    "INPUT": DATA + "/DEM_KPH.tif",
    "MASK": DATA + "/Area_Kajian.gpkg",
    "NODATA": -9999,
    "CROP_TO_CUTLINE": True,
    "KEEP_RESOLUTION": True,
    "OUTPUT": OUT + "/DEM_kajian.tif"})
u = processing.run("native:rasterlayeruniquevaluesreport", {"INPUT": hasil["OUTPUT"], "BAND": 1})
isi = u["TOTAL_PIXEL_COUNT"] - u["NODATA_PIXEL_COUNT"]
print("Piksel total:", u["TOTAL_PIXEL_COUNT"], "| NoData:", u["NODATA_PIXEL_COUNT"])
print("Piksel berisi nilai:", isi, "= luas (ha):", round(isi * 4 / 10000, 2))

# 2) Potong dengan satu KPH saja (filter lapisan batas lebih dulu)
batas = QgsVectorLayer(DATA + "/Batas_KPH.gpkg|layername=Batas_KPH", "Batas_KPH", "ogr")
batas.setSubsetString("\"NAMA_KPH\" = 'KPH Alpha'")
h2 = processing.run("gdal:cliprasterbymasklayer", {
    "INPUT": DATA + "/DEM_KPH.tif", "MASK": batas, "NODATA": -9999,
    "CROP_TO_CUTLINE": True, "KEEP_RESOLUTION": True, "OUTPUT": OUT + "/DEM_Alpha.tif"})
u2 = processing.run("native:rasterlayeruniquevaluesreport", {"INPUT": h2["OUTPUT"], "BAND": 1})
```

```
print("KPH Alpha: piksel total", u2["TOTAL_PIXEL_COUNT"], "| NoData", u2["NODATA_PIXEL_COUNT"])
```

Penjelasan baris demi baris.

Baris	Tulisan	Artinya
1-4	komentar dan `import`	Judul, penulis, `os`, `processing`, dan `QgsVectorLayer` (untuk membuka batas KPH).
6-8	`DATA`, `OUT`, `os.makedirs`	Lokasi data dan folder hasil (dari `B3_DATA` dan `B3_OUT` atau alamat cadangan), lalu pembuatan folder hasil bila belum ada. **Atur atau ganti lokasinya.**
11-17	`gdal:cliprasterbymasklayer`	Potong DEM dengan poligon `Area_Kajian`. `MASK` poligonnya. `NODATA: -9999` adalah penanda kotak kosong di luar poligon. `CROP_TO_CUTLINE: True` memangkas ukuran raster. `KEEP_RESOLUTION: True` menjaga ukuran piksel tetap 2 m.
18	`native:rasterlayeruniquevaluesreport`	Buat laporan nilai unik. Dari kamus hasil, skrip mengambil jumlah piksel total dan jumlah piksel NoData.
19	`isi = ... - ...`	Piksel berisi nilai = total dikurangi NoData.
20-21	`print(...)`	Cetak total dan NoData, lalu piksel berisi dan luasnya dalam hektare: piksel dikali 4 meter persegi, dibagi 10.000.
24	`batas = QgsVectorLayer(...)`	Buka layer batas KPH.
25	`batas.setSubsetString("\"NAMA_KPH\" = 'KPH Alpha'")`	Pasang filter: hanya KPH Alpha yang tampil.
26-28	`gdal:cliprasterbymasklayer`	Potong DEM lagi. Kali ini `MASK` adalah **layer** `batas`, bukan jalur berkas, sehingga filter ikut berlaku.
29-30	laporan nilai unik	Cetak piksel total dan NoData untuk KPH Alpha.

Hasil. Layar menampilkan (uji pada QGIS 4.0.2):

```
HASIL
Piksel total: 24800 | NoData: 6100
Piksel berisi nilai: 18700 = luas (ha): 7.48
KPH Alpha: piksel total 10000 | NoData 0
```

Pada sebagian komputer, baris `Warning 1: Field NAMA_KPH of width 255 truncated to 254.` ikut tampil. Itu peringatan GDAL yang tidak berbahaya. Bacalah hasilnya: poligon kajian meninggalkan 6100 kotak kosong. KPH Alpha berbentuk persegi 200 m x 200 m, jadi 100 x 100 piksel penuh, tanpa NoData.

Bagian yang boleh diubah.

- **Baris 6-7, lokasi.** Atur `B3_DATA` dan `B3_OUT`, atau ganti alamat cadangannya.
- **Baris 25, nama KPH.** Ganti `'KPH Alpha'` dengan `'KPH Beta'` atau `'KPH Gamma'`. Ejaannya harus sama persis dengan isi kolom.
- Baris 14 dan 27, `NODATA`. Pilih angka yang tidak mungkin muncul sebagai tinggi sungguhan.

- Baris 15-16, `CROP_TO_CUTLINE` dan `KEEP_RESOLUTION`. Ubah ke `False` untuk melihat akibatnya pada ukuran hasil.
- Baris 21, angka `4`. Luas satu piksel dalam meter persegi (2 m x 2 m).

Jebakan.

- **Angka 4 tertulis tetap.** Faktor luas mengandaikan piksel 2 m. Bila piksel Anda 10 m, luas salah tanpa pesan. Ambil ukuran piksel dari Skrip 16.1.
- **Filter ikut terbawa.** Baris 26-28 memakai layer yang difilter. Bila Anda menghapus baris 25, semua KPH ikut terpotong. Dari uji: tanpa filter, hasilnya 40000 piksel total (seluruh DEM), karena ketiga KPH menutup seluruh DEM.
- **Data tidak ditemukan.** Folder hasil dibuat otomatis, tetapi `DATA` tidak. Bila `DATA` salah, alat berhenti dengan galat `QgsProcessingException: Unable to execute algorithm`, disusul `Could not load source layer for INPUT: .../DEM_KPH.tif not found` (diuji). Folder hasil tetap terbentuk, tetapi kosong. Periksa baris 6.
- Nilai `-9999`. Bila DEM asli punya nilai itu, ia ikut dianggap kosong. [kemungkinan]
- **Filter hanya berlaku di sesi ini.** `setSubsetString` mengubah objek `batas`, bukan berkas.

Latihan variasi.

1. Potong DEM dengan KPH Beta. Berapa piksel total dan NoData-nya?
2. Cetak luas KPH Alpha dalam hektare dari hasil potongan.

Contoh jawaban. Nomor 1: ganti `'KPH Alpha'` di baris 25 dengan `'KPH Beta'`. Layar menampilkan `piksel total 10000 | NoData 0`. Nomor 2: tambahkan di akhir skrip:

```python
print("Luas KPH Alpha (ha):", (u2["TOTAL_PIXEL_COUNT"] - u2["NODATA_PIXEL_COUNT"]) * 4 / 10000)
```

Layar menampilkan `Luas KPH Alpha (ha): 4.0`.

> **Pola yang bisa dipakai ulang:** Potong lalu hitung: `processing.run("gdal:cliprasterbymasklayer", {"INPUT": RASTER, "MASK": POLIGON, "NODATA": ANGKA, "CROP_TO_CUTLINE": True, "KEEP_RESOLUTION": True, "OUTPUT": HASIL})`, lalu `native:rasterlayeruniquevaluesreport` untuk menghitung piksel. Ganti **raster**, **poligon (atau layer yang difilter)**, dan **angka NoData**.

Skrip 16.5: Mengelompokkan lereng menjadi tiga kelas

Peta kemiringan berisi puluhan ribu angka berbeda. Atasan lebih mudah membaca tiga kelas: datar, landai, curam. Berkasnya `b3_05_reklas.py`; di dalamnya tertulis "Skrip 5.1".

Tujuan. Mengubah kemiringan lereng menjadi tiga kelas, lalu menghitung jumlah piksel dan luas tiap kelas.

Bahan. `Lereng_derajat.tif` di folder `OUT`, hasil Skrip 16.3. **Jalankan Skrip 16.3 lebih dulu.** Skrip ini tidak memakai `DATA`.

Istilah baru di skrip ini:

- **reklasifikasi**: Mengganti rentang nilai menjadi nomor kelas.
- **tabel kelas**: Batas bawah, batas atas, nomor baru.
- **tabel hasil**: Layer berisi baris, bukan peta.
- **Byte**: Bilangan bulat 0 sampai 255.

Ilustrasi 16.7: Alur skrip 16.5



Skrip utuh:

```python
# Skrip 5.1: reklasifikasi kemiringan lereng menjadi tiga kelas dan hitung luasnya. Penulis: Badar Mubarok Yogaswara
# Jalankan Skrip 3.1 lebih dulu (butuh Lereng_derajat.tif).
import os
import processing
from qgis.core import QgsVectorLayer

OUT = os.environ.get("B3_OUT", r"C:/kerja_b3")
os.makedirs(OUT, exist_ok=True)

# Tabel: batas bawah, batas atas, kelas baru. Aturan batas: bawah <= nilai < atas (RANGE_BOUNDARIES = 1)
tabel = [0, 5, 1,       # kelas 1 = datar   (0 sampai kurang dari 5 derajat)
         5, 15, 2,      # kelas 2 = landai  (5 sampai kurang dari 15 derajat)
         15, 90, 3]     # kelas 3 = curam   (15 derajat ke atas)
r = processing.run("native:reclassifybytable", {
    "INPUT_RASTER": OUT + "/Lereng_derajat.tif", "RASTER_BAND": 1,
    "TABLE": tabel, "RANGE_BOUNDARIES": 1, "NODATA_FOR_MISSING": True,
    "NO_DATA": 255, "DATA_TYPE": 0,          # 0 = Byte
    "OUTPUT": OUT + "/Lereng_kelas.tif"})

u = processing.run("native:rasterlayeruniquevaluesreport", {"INPUT": r["OUTPUT"], "BAND": 1,
                                                            "OUTPUT_TABLE": "TEMPORARY_OUTPUT"})
nama = {1: "Datar", 2: "Landai", 3: "Curam"}
print("Kelas | jumlah piksel | luas (ha)")
for f in u["OUTPUT_TABLE"].getFeatures():
    k = int(f["value"])
    print(k, nama.get(k, "NoData"), f["count"], round(f["count"] * 4 / 10000, 2))
```

Penjelasan baris demi baris.

Baris	Tulisan	Artinya
1-5	komentar dan `import`	Judul, penulis, pengingat Skrip 3.1 (di buku ini 16.3), dan perkakas. `QgsVectorLayer` diimpor tetapi tidak dipakai.
7-8	`OUT = os.environ.get(...), os.makedirs(...)`	Folder hasil, dari pengaturan `B3_OUT` atau alamat cadangan, lalu dibuat bila belum ada. **Atur atau ganti dengan folder Anda**, sama dengan Skrip 16.3.
10-13	`tabel = [0, 5, 1, 5, 15, 2, 15, 90, 3]`	Satu daftar datar yang dibaca per tiga angka: batas bawah, batas atas, nomor kelas. Kelas 1 untuk 0 sampai kurang dari 5 derajat, kelas 2 untuk 5 sampai kurang dari 15, kelas 3 untuk 15 sampai 90.
14-18	`native:reclassifybytable`	Jalankan reklasifikasi. `RANGE_BOUNDARIES: 1` berarti batas bawah ikut dan batas atas tidak (bawah <= nilai < atas). `NODATA_FOR_MISSING: True` memberi NoData pada nilai di luar tabel. `NO_DATA: 255` adalah penanda NoData. `DATA_TYPE: 0` memilih tipe Byte.
20-21	`native:rasterlayeruniquevaluesreport`	Hitung nilai unik hasil. `OUTPUT_TABLE: "TEMPORARY_OUTPUT"` meminta tabel sementara berisi kolom `value` (nomor kelas) dan `count` (jumlah piksel).
22	`nama = {1: "Datar", 2: "Landai", 3: "Curam"}`	Kamus yang menerjemahkan nomor kelas menjadi nama.
23	`print("Kelas ...")`	Cetak judul tabel. Garis tegak di dalam tulisan hanyalah pemisah tampilan.
24-26	`for f in u["OUTPUT_TABLE"].getFeatures():`	Untuk tiap baris tabel: ubah nilai kelas menjadi bilangan bulat, cari namanya di kamus (`nama.get(k, "NoData")`: bila tidak ada, tulis NoData), lalu cetak jumlah piksel dan luas dalam hektare.

Hasil. Layar menampilkan (uji pada QGIS 4.0.2):

```
HASIL
Kelas | jumlah piksel | luas (ha)
1 Datar 19018 7.61
2 Landai 19902 7.96
3 Curam 1080 0.43
```

Cek: 19018 + 19902 + 1080 = 40000, seluruh piksel citra. Luasnya 7,61 + 7,96 + 0,43 = 16 hektare.

Bagian yang boleh diubah.

- **Baris 8, lokasi.** Atur `B3_OUT` atau ganti alamat cadangan di baris 7. Harus sama dengan `OUT` di Skrip 16.3.
- **Baris 11-13, batas kelas.** Ubah angkanya untuk mengubah luas tiap kelas. Dari uji: dengan batas 8 dan 20 derajat, hasilnya Datar 23930 piksel (9,57 ha), Landai 15981 (6,39 ha), dan Curam 89 (0,04 ha).

- Baris 16, `RANGE_BOUNDARIES`. Pilihannya: 0 untuk bawah < nilai <= atas, 1 untuk bawah <= nilai < atas, 2 untuk kedua batas ikut, 3 untuk kedua batas tidak ikut. Dari uji, pada data ini keempat pilihan menghasilkan hitungan yang sama, karena nyaris tidak ada piksel yang bernilai tepat di batas.
- Baris 17, `NO_DATA` dan `DATA_TYPE`. Angka 255 cocok dengan tipe Byte. Jangan memakai kelas bernomor 255.
- **Baris 22, nama kelas.** Ganti kata sesuai kebutuhan, dan tambah bila kelas bertambah.

Jebakan.

- **Lupa Skrip 16.3.** Folder `OUT` dibuat otomatis, tetapi masih kosong. Berkas lereng tidak ada, dan baris 14 berhenti dengan: `_core.QgsProcessingException: Unable to execute algorithm` lalu `Could not load source layer for INPUT_RASTER: .../Lereng_derajat.tif not found`
- **Batas atas 90.** Nilai lereng di atas 90 derajat jadi NoData. Lereng tidak mungkin melewati 90, jadi aman.
- **Angka 4 tertulis tetap.** Luas piksel 4 meter persegi, seperti di Skrip 16.4.
- **Tabel sementara hilang.** `TEMPORARY_OUTPUT` hanya ada selama sesi.

Latihan variasi.

1. Ubah batas kelas menjadi 8 dan 20 derajat.
2. Tambahkan satu kelas keempat, "Sangat curam", untuk lereng 20 derajat ke atas.

Contoh jawaban. Nomor 1: ganti isi `tabel` menjadi `[0, 8, 1, 8, 20, 2, 20, 90, 3]`. Layar menampilkan 9,57 ha, 6,39 ha, dan 0,04 ha. Nomor 2: ganti `tabel` menjadi `[0, 5, 1, 5, 15, 2, 15, 20, 3, 20, 90, 4]` dan `nama` menjadi `{1: "Datar", 2: "Landai", 3: "Curam", 4: "Sangat curam"}`. Layar menampilkan, untuk dua baris terakhir, `3 Curam 991 0.4` dan `4 Sangat curam 89 0.04`. Jumlahnya 1080 piksel, sama dengan kelas curam semula.

> **Pola yang bisa dipakai ulang:** Reklasifikasi: tulis tabel `[bawah, atas, kelas, ...]`, jalankan `native:reclassifybytable`, lalu `native:rasterlayeruniquevaluesreport` dan kamus nama untuk membaca luas. Ganti **batas kelas**, **raster masukan**, dan **nama kelas**.

Skrip 16.6: Menghitung NDVI dan merata-ratakan per KPH

Tanaman sehat memantulkan banyak cahaya inframerah dekat dan menyerap cahaya merah. NDVI membandingkan keduanya dalam satu angka. Berkasnya `b3_06_ndvi.py`; di dalamnya tertulis "Skrip 6.1".

Tujuan. Menghitung NDVI dari band inframerah dekat dan merah, mengelompokkannya menjadi empat kelas, menghitung luas tiap kelas, lalu mencari rata-rata NDVI tiap KPH.

Bahan. `Citra_4band.tif` di folder `OUT`, hasil Skrip 16.2, dan `Batas_KPH.gpkg` di `DATA`. **Jalankan Skrip 16.2 lebih dulu.**

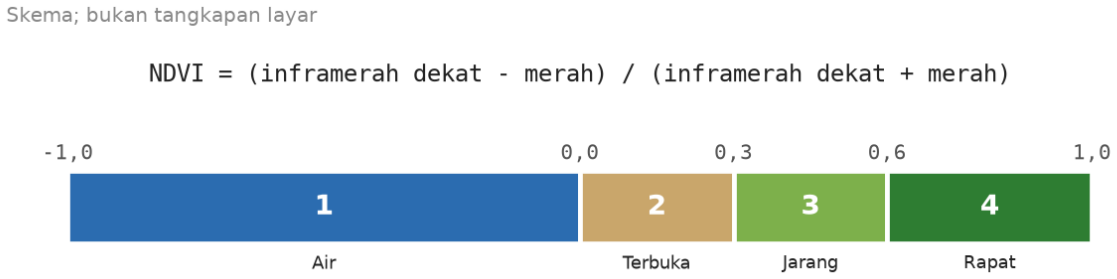
Istilah baru di skrip ini:

- **NDVI**: Indeks vegetasi, hasil membandingkan pantulan inframerah dekat dan merah.
- **kalkulator raster**: `QgsRasterCalculator`.

- **entri**: Satu band yang diberi nama pendek.
- **statistik zonal**: Statistik raster per poligon.
- **extent**: Batas luar raster.

Ilustrasi 16.8: NDVI dan empat kelasnya

Skema; bukan tangkapan layar



Kelas 1 air, 2 lahan terbuka, 3 vegetasi jarang, 4 vegetasi rapat. Dekat 1: tanaman rapat dan sehat.

Ilustrasi 16.9: Alur skrip 16.6

Skema; bukan tangkapan layar



Skrip utuh:

PYTHON

```
# Skrip 6.1: menghitung NDVI, mengelompokkannya, dan merata-ratakan per KPH. Penulis: Badar Mubarok Yogaswara
# Jalankan Skrip 2.1 lebih dulu (butuh Citra_4band.tif).
import os
import processing
from qgis.core import QgsRasterLayer, QgsProject
from qgis.analysis import QgsRasterCalculator, QgsRasterCalculatorEntry

DATA = os.environ.get("B3_DATA", r"C:/KPH_Contoh/paket-b3")
OUT = os.environ.get("B3_OUT", r"C:/kerja_b3")
os.makedirs(OUT, exist_ok=True)

stack = QgsRasterLayer(OUT + "/Citra_4band.tif", "Citra_4band")
nir = QgsRasterCalculatorEntry()
nir.ref, nir.raster, nir.bandNumber = "nir@1", stack, 4      # band 4 = inframerah dekat
red = QgsRasterCalculatorEntry()
red.ref, red.raster, red.bandNumber = "red@1", stack, 3      # band 3 = merah
```

```
rumus = '("nir@1" - "red@1") / ("nir@1" + "red@1")'
calc = QgsRasterCalculator(rumus, OUT + "/NDVI.tif", "GTiff", stack.extent(), stack.width(), stack.height(),
                           [nir, red], QgsProject.instance().transformContext())
print("Kode hasil (0 = sukses):", calc.processCalculation())

st = processing.run("native:rasterlayerstatistics", {"INPUT": OUT + "/NDVI.tif", "BAND": 1})
print("NDVI min/maks/rata-rata:", round(st["MIN"], 3), "/", round(st["MAX"], 3), "/", round(st["MEAN"], 3))

# Kelompokkan: sampai 0 = air; 0-0,3 = lahan terbuka; 0,3-0,6 = vegetasi jarang; di atas 0,6 = vegetasi rapat
processing.run("native:reclassifybytable", {
    "INPUT_RASTER": OUT + "/NDVI.tif", "RASTER_BAND": 1,
    "TABLE": [-1, 0, 1,  0, 0.3, 2,  0.3, 0.6, 3,  0.6, 1.01, 4],
    "RANGE_BOUNDARIES": 0, "NODATA_FOR_MISSING": True, "NO_DATA": 255, "DATA_TYPE": 0,
    "OUTPUT": OUT + "/NDVI_kelas.tif"})
u = processing.run("native:rasterlayeruniquevaluesreport", {"INPUT": OUT + "/NDVI_kelas.tif", "BAND": 1,
                                                            "OUTPUT_TABLE": "TEMPORARY_OUTPUT"})
nama = {1: "Air", 2: "Lahan terbuka", 3: "Vegetasi jarang", 4: "Vegetasi rapat"}
for f in u["OUTPUT_TABLE"].getFeatures():
    k = int(f["value"])
    print(k, nama.get(k, "NoData"), f["count"], "piksel =", round(f["count"] * 4 / 10000, 2), "ha")

# Rata-rata NDVI per KPH
z = processing.run("native:zonalstatisticsfb", {
    "INPUT": DATA + "/Batas_KPH.gpkg|layername=Batas_KPH", "INPUT_RASTER": OUT + "/NDVI.tif", "RASTER_BAND": 1,
    "COLUMN_PREFIX": "ndvi_", "STATISTICS": [2], "OUTPUT": "TEMPORARY_OUTPUT"})
for f in z["OUTPUT"].getFeatures():
    print(f["NAMA_KPH"], "NDVI rata-rata:", round(f["ndvi_mean"], 3))
```

Penjelasan baris demi baris.

Baris	Tulisan	Artinya
1-6	komentar dan `import`	Judul, penulis, pengingat, dan perkakas. `QgsProject` hanya dipakai untuk satu hal di baris 20. `QgsRasterCalculator` dan `QgsRasterCalculatorEntry` berasal dari `qgis.analysis`.
8-10	`DATA`, `OUT`, `os.makedirs`	Lokasi data dan folder hasil (dari `B3_DATA` dan `B3_OUT` atau alamat cadangan), lalu pembuatan folder hasil bila belum ada. **Atur atau ganti lokasinya.**
12	`stack = QgsRasterLayer(...)`	Buka `Citra_4band.tif`.
13-14	`nir = QgsRasterCalculatorEntry(), nir.ref, nir.raster, nir.bandNumber = ...`	Buat entri untuk inframerah dekat. Satu baris mengisi tiga sifat sekaligus: nama pendek `nir@1`, raster sumber, dan nomor band 4.
15-16	`red = ...`	Entri yang sama untuk merah: nama `red@1`, band 3.
18	`rumus = '("nir@1" - "red@1") / ("nir@1" + "red@1")'`	Rumus NDVI sebagai teks. Teks dibuka dengan kutip tunggal supaya kutip ganda di dalamnya aman.
19-20	`QgsRasterCalculator(...)`	Siapkan hitungan. Isiannya berurutan: rumus, berkas hasil, format `GTiff`, extent, lebar, tinggi, daftar entri, dan konteks transformasi. Urutannya tidak boleh tertukar.
21	`calc.processCalculation()`	Jalankan hitungan. Hasilnya angka: `0` berarti sukses.
23-24	statistik NDVI	Cetak NDVI terkecil, terbesar, dan rata-rata, dibulatkan tiga desimal.

Baris	Tulisan	Artinya
26	komentar	Pengingat batas kelas: sampai 0 air, 0 sampai 0,3 lahan terbuka, 0,3 sampai 0,6 vegetasi jarang, di atas 0,6 vegetasi rapat.
27-31	`native:reclassifybytable`	Kelompokkan NDVI menjadi empat kelas. Tabel `[-1, 0, 1, 0, 0.3, 2, ...]` dibaca per tiga angka. `RANGE_BOUNDARIES: 0` berarti bawah < nilai <= atas.
32-33	laporan nilai unik	Hitung jumlah piksel tiap kelas.
34	`nama = {1: "Air", ...}`	Kamus nama kelas.
35-37	`for f in u["OUTPUT_TABLE"].getFeatures():`	Cetak nama kelas, jumlah piksel, dan luas hektare.
40-42	`native:zonalstatisticsfb`	Hitung rata-rata NDVI di dalam tiap poligon KPH. `COLUMN_PREFIX: "ndvi_"` dan `STATISTICS: [2]` (rata-rata) membuat kolom `ndvi_mean`.
43-44	`for f in z["OUTPUT"].getFeatures():`	Cetak nama KPH dan rata-rata NDVI-nya.

Hasil. Layar menampilkan (uji pada QGIS 4.0.2):

```
HASIL
Kode hasil (0 = sukses): 0
NDVI min/maks/rata-rata: -0.328 / 0.903 / 0.735
1 Air 1203 piksel = 0.48 ha
2 Lahan terbuka 1767 piksel = 0.71 ha
3 Vegetasi jarang 4994 piksel = 2.0 ha
4 Vegetasi rapat 32036 piksel = 12.81 ha
KPH Alpha NDVI rata-rata: 0.814
KPH Beta NDVI rata-rata: 0.614
KPH Gamma NDVI rata-rata: 0.755
```

Cek: 1203 + 1767 + 4994 + 32036 = 40000 piksel, dan 0,48 + 0,71 + 2,0 + 12,81 = 16 hektare. Rata-rata tertinggi di KPH Alpha, terendah di KPH Beta. Nama kelas "Air" hanyalah label penulis untuk NDVI nol ke bawah.

Bagian yang boleh diubah.

- **Baris 14 dan 16, nomor band.** Band inframerah dekat dan merah. Harus cocok dengan urutan di Skrip 16.2.
- **Baris 18, rumus.** Ganti dengan indeks lain, selama memakai nama entri yang sama.
- **Baris 29, batas kelas.** Dari uji: mengganti 0.6 menjadi 0.7 menurunkan vegetasi rapat menjadi 30791 piksel (12,32 ha).
- **Baris 41-42, awalan dan statistik.** Ganti `ndvi_` bila perlu, dan ingat kolom hasilnya mengikuti awalan itu (`ndvi_mean`).

Jebakan.

- **Band tertukar membalik hasil tanpa pesan.** Dari uji: Skrip 16.2 dengan urutan B08 dan B04 tertukar tetap memberi kode hasil `0`. Namun hampir seluruh KPH menjadi "Air" dan rata-rata

NDVI bertanda minus (lihat blok di bawah daftar ini). NDVI negatif untuk hutan adalah tanda ada yang keliru.

- **Lupa Skrip 16.2.** Berkas empat band tidak ada, dan layar menampilkan galat bahwa sumber raster tidak dapat dimuat.
- **Pembagi nol.** Bila NIR dan merah sama-sama 0 pada satu piksel, rumus membagi nol. Dari uji pada citra buatan (blok 10 x 10 piksel dengan NIR dan merah 0): tidak ada galat, kode hasil tetap `0`, dan 100 piksel itu menjadi NoData, jadi tidak ikut terhitung di tabel kelas (jumlahnya 39900, bukan 40000).
- Kode `0 = sukses` bergantung versi QGIS. [CEK]
- **Pesan "Warning 1".** Peringatan GDAL tentang lebar kolom `NAMA_KPH` bukan galat.

Layar bila band tertukar:

HASIL

```
Kode hasil (0 = sukses): 0
NDVI min/maks/rata-rata: -0.903 / 0.328 / -0.735
1 Air 38797 piksel = 15.52 ha
2 Lahan terbuka 1191 piksel = 0.48 ha
3 Vegetasi jarang 12 piksel = 0.0 ha
KPH Alpha NDVI rata-rata: -0.814
KPH Beta NDVI rata-rata: -0.614
KPH Gamma NDVI rata-rata: -0.755
```

Latihan variasi.

1. Jadikan batas vegetasi rapat 0,7, bukan 0,6.
2. Cetak NDVI rata-rata tiap KPH dengan tiga desimal, tetapi hanya KPH yang di atas 0,7.

Contoh jawaban. Nomor 1: pada tabel di baris 29, ganti `0.3, 0.6, 3,  0.6, 1.01, 4` menjadi `0.3, 0.7, 3,  0.7, 1.01, 4`. Layar menampilkan `3 Vegetasi jarang 6239 piksel = 2.5 ha` dan `4 Vegetasi rapat 30791 piksel = 12.32 ha`. Nomor 2: ubah baris 44 menjadi `if f["ndvi_mean"] > 0.7:` lalu cetak di bawahnya. Layar menampilkan hanya KPH Alpha (0.814) dan KPH Gamma (0.755).

> **Pola yang bisa dipakai ulang:** Indeks dari band: buat entri per band (`ref`, `raster`, `bandNumber`), tulis rumus, jalankan `QgsRasterCalculator(...)`, lalu kelompokkan dengan `native:reclassifybytable` dan ratakan per poligon dengan `native:zonalstatisticsfb`. Ganti **nomor band**, **rumus**, dan **batas kelas**.

Skrip 16.7: Satu tabel ringkasan untuk Kepala Seksi

Semua hasil tadi disatukan dalam satu tabel: tiap KPH dengan luas, NDVI rata-rata, luas lereng curam, dan luas vegetasi rapat. Berkas `b3_07_proyek.py`; di dalamnya tertulis "Skrip 7.1".

Tujuan. Mencetak tabel ringkasan per KPH dengan memotong raster hasil sebelumnya untuk tiap KPH.

Bahan. Di folder `OUT`: `NDVI.tif` dan `NDVI_kelas.tif` (Skrip 16.6) serta `Lereng_kelas.tif` (Skrip 16.5). Di `DATA`: `Batas_KPH.gpkg`. **Jalankan Skrip 16.2, 16.3, 16.5, dan 16.6 lebih dulu.**

Istilah baru di skrip ini:

- `return`: Mengembalikan hasil sebuah fungsi (Bab 7).
- **tuple**: Daftar tetap, Bab 5.
- `%s`: Menyisipkan teks, Bab 9.
- **docstring**: Catatan di bawah `def`.

Ilustrasi 16.10: Alur skrip 16.7



Skrip utuh:

```
PYTHON
# Skrip 7.1: ringkasan per KPH (luas, NDVI rata-rata, luas lereng curam, luas vegetasi rapat). Penulis: Badar
Mubarok Yogaswara
# Jalankan Skrip 2.1, 3.1, 5.1, dan 6.1 lebih dulu (butuh NDVI.tif, NDVI_kelas.tif, Lereng_kelas.tif di OUT).
import os
import processing
from qgis.core import QgsVectorLayer

DATA = os.environ.get("B3_DATA", r"C:/KPH_Contoh/paket-b3")
OUT = os.environ.get("B3_OUT", r"C:/kerja_b3")
BATAS = DATA + "/Batas_KPH.gpkg|layername=Batas_KPH"


def luas_kelas(raster, mask, kelas):
    """Luas (ha) piksel bernilai 'kelas' di dalam mask."""
    pot = processing.run("gdal:cliprasterbymasklayer", {
        "INPUT": raster, "MASK": mask, "NODATA": 255, "CROP_TO_CUTLINE": True,
        "KEEP_RESOLUTION": True, "OUTPUT": "TEMPORARY_OUTPUT"})
    u = processing.run("native:rasterlayeruniquevaluesreport", {
        "INPUT": pot["OUTPUT"], "BAND": 1, "OUTPUT_TABLE": "TEMPORARY_OUTPUT"})
    for f in u["OUTPUT_TABLE"].getFeatures():
        if int(f["value"]) == kelas:
            return f["count"] * 4 / 10000
    return 0.0


print("KPH | luas (ha) | NDVI rata-rata | curam (ha) | vegetasi rapat (ha)")
for nama in ("KPH Alpha", "KPH Beta", "KPH Gamma"):
    lyr = QgsVectorLayer(BATAS, "batas", "ogr")
    lyr.setSubsetString("\"NAMA_KPH\" = '%s'" % nama)
    luas = sum(f.geometry().area() for f in lyr.getFeatures()) / 10000
    z = processing.run("native:zonalstatisticsfb", {
        "INPUT": lyr, "INPUT_RASTER": OUT + "/NDVI.tif", "RASTER_BAND": 1,
        "COLUMN_PREFIX": "n_", "STATISTICS": [2], "OUTPUT": "TEMPORARY_OUTPUT"})
```

```
    ndvi = [f["n_mean"] for f in z["OUTPUT"].getFeatures()][0]
    curam = luas_kelas(OUT + "/Lereng_kelas.tif", lyr, 3)
    rapat = luas_kelas(OUT + "/NDVI_kelas.tif", lyr, 4)
    print(nama, "|", round(luas, 2), "|", round(ndvi, 3), "|", round(curam, 2), "|", round(rapat, 2))
```

Penjelasan baris demi baris.

Baris	Tulisan	Artinya
1-5	komentar dan `import`	Judul, penulis, urutan skrip yang harus dijalankan lebih dulu, dan perkakas (`os` dipakai untuk membaca pengaturan).
7-9	`DATA`, `OUT`, `BATAS`	Lokasi data (dari `B3_DATA` atau alamat cadangan), folder hasil (dari `B3_OUT` atau alamat cadangan), dan alamat lengkap layer batas KPH. Dua baris pertama **diatur atau diganti**. Skrip ini tidak membuat folder: ia hanya membaca hasil skrip lain.
12-13	`def luas_kelas(raster, mask, kelas):`	Fungsi yang menghitung luas (hektare) piksel bernomor `kelas` di dalam poligon `mask`. Kalimat di dalam tiga kutip adalah docstring.
14-16	`gdal:cliprasterbymasklayer`	Potong raster dengan poligon. `NODATA: 255` cocok dengan Skrip 16.5 dan 16.6. `OUTPUT: "TEMPORARY_OUTPUT"` berarti hasil sementara, tidak ditulis ke berkas pilihan Anda.
17-18	`native:rasterlayeruniquevaluesreport`	Hitung jumlah piksel tiap nilai pada hasil potongan.
19-21	`for f in ...: if int(f["value"]) == kelas: return ...`	Cari baris untuk kelas yang diminta. Begitu ketemu, fungsi langsung mengembalikan luasnya: jumlah piksel dikali 4 meter persegi, dibagi 10.000.
22	`return 0.0`	Bila kelas itu tidak ada sama sekali di dalam poligon, luasnya 0.
25	`print("KPH ...")`	Cetak judul tabel.
26	`for nama in ("KPH Alpha", "KPH Beta", "KPH Gamma"):`	Ulangi untuk tiap KPH. Tanda kurung dengan koma adalah tuple: daftar tetap.
27-28	`lyr = QgsVectorLayer(...), setSubsetString("... '%s'" % nama)`	Buka layer batas yang baru, lalu saring ke satu KPH. Tanda `%s` diganti dengan isi `nama`. Layer dibuat baru tiap putaran supaya filter tidak menumpuk.
29	`luas = sum(f.geometry().area() for f in lyr.getFeatures()) / 10000`	Hitung luas KPH dari bentuk poligonnya, lalu ubah ke hektare.
30-32	`native:zonalstatisticsfb`	Hitung rata-rata NDVI di dalam KPH itu. Awalan kolom `n_`.
33	`ndvi = [f["n_mean"] for ...][0]`	Ambil nilai pertama dari daftar hasil. Tanda `[0]` mengambil anggota pertama.
34-35	`luas_kelas(...)`	Hitung luas lereng curam (kelas 3) dan vegetasi rapat (kelas 4) dengan fungsi tadi.
36	`print(nama, ...)`	Cetak satu baris tabel: nama, luas, NDVI, luas curam, dan luas vegetasi rapat, dipisah garis tegak.

Hasil. Layar menampilkan (uji pada QGIS 4.0.2):

```
HASIL
KPH | luas (ha) | NDVI rata-rata | curam (ha) | vegetasi rapat (ha)
KPH Alpha | 4.0 | 0.814 | 0.0 | 3.75
KPH Beta | 4.0 | 0.614 | 0.0 | 2.0
KPH Gamma | 8.0 | 0.755 | 0.43 | 7.06
```

Pada sebagian komputer, baris `Warning 1: Field NAMA_KPH ...` ikut tampil beberapa kali. Itu peringatan GDAL, bukan galat. Cek silang dengan skrip sebelumnya: lereng curam 0,43 hektare sama dengan hasil Skrip 16.5, dan vegetasi rapat 3,75 + 2,0 + 7,06 = 12,81 hektare sama dengan hasil Skrip 16.6. Seluruh lereng curam berada di KPH Gamma.

Bagian yang boleh diubah.

- **Baris 7-8, lokasi.** Atur `B3_DATA` dan `B3_OUT`, atau ganti alamat cadangannya. `OUT` harus sama dengan yang dipakai Skrip 16.3 sampai 16.6.
- **Baris 26, daftar nama KPH.** Harus sama persis dengan isi kolom `NAMA_KPH`. Tambah nama bila ada KPH lain.
- **Baris 34-35, nomor kelas.** Ganti 3 dan 4 sesuai kelas yang ingin dilihat. Dari uji: mengganti 3 menjadi 2 (landai) memberi 1,67, 0,29, dan 6,0 hektare, yang jumlahnya 7,96, sama dengan Skrip 16.5.
- **Baris 21, faktor 4.** Luas piksel dalam meter persegi.
- Baris 15, `NODATA: 255`. Harus sama dengan `NO_DATA` pada skrip yang membuat rasternya. Skrip 16.4 memakai -9999 karena itu DEM dengan nilai tinggi, sedangkan kelas bertipe Byte memakai 255.

Jebakan.

- **Nama KPH tertulis tetap.** Bila nama tidak ada di data, layer batas kosong. Dari uji dengan `"KPH Delta"`, skrip berhenti di baris 33 dengan `IndexError: list index out of range`, karena daftar hasil kosong.
- **Skrip pendahulu belum dijalankan, atau `OUT` berbeda.** Dari uji: galat `Could not load source layer for INPUT_RASTER: .../NDVI.tif not found`.
- **Faktor 4 tertulis tetap.** Sama seperti Skrip 16.4 dan 16.5.
- Hasil `luas_kelas` untuk kelas yang tidak ada adalah `0.0`. Itu benar, bukan galat. Seperti KPH Alpha dan Beta untuk kelas curam.
- **Peringatan lebar kolom.** Bukan masalah.

Latihan variasi.

1. Tampilkan luas lereng landai (kelas 2), bukan curam.
2. Ganti daftar KPH menjadi hanya `"KPH Gamma"`.

Contoh jawaban. Nomor 1: ganti `3` menjadi `2` di baris 34, dan kata "curam" di baris 25 menjadi "landai". Layar menampilkan `KPH Alpha | 4.0 | 0.814 | 1.67 | 3.75`, `KPH Beta | 4.0 | 0.614 | 0.29 | 2.0`, dan `KPH Gamma | 8.0 | 0.755 | 6.0 | 7.06`. Nomor 2: tulis `("KPH Gamma",)` di baris 26, dengan koma setelah nama. Layar menampilkan satu baris data: `KPH Gamma | 8.0 | 0.755 | 0.43`

`| 7.06`. Tanpa koma itu, Python membaca tanda kurung sebagai pengelompokan biasa dan memecah tulisan menjadi huruf. [kemungkinan]

> **Pola yang bisa dipakai ulang:** Tabel per wilayah: fungsi kecil yang memotong raster dan mengembalikan satu angka, lalu `for nama in (DAFTAR):` yang memanggilnya dan mencetak satu baris. Ganti **daftar nama**, **raster dan nomor kelas**, dan **faktor luas piksel**.

Cek paham

1. Mengapa urutan berkas di baris 11 Skrip 16.2 penting?
2. Apa yang terjadi pada Skrip 16.1 bila jalur `DATA` salah, dan baris mana yang menanganinya?
3. Dari mana angka 4 pada rumus luas berasal, dan kapan ia menjadi salah?
4. Skrip mana yang harus dijalankan lebih dulu sebelum Skrip 16.7?

Jawaban.

1. Urutan berkas menentukan nomor band: 1 biru, 2 hijau, 3 merah, 4 inframerah dekat. Skrip 16.6 memakai band 3 dan 4. Bila tertukar, NDVI terbalik tanda.
2. Skrip berhenti dengan pesan `Raster tidak terbaca. Periksa DATA: ...`. Itu hasil pengaman di baris 10-11 (`raise SystemExit`).
3. Dari ukuran piksel 2 m x 2 m, yaitu 4 meter persegi per piksel. Bila piksel data Anda berukuran lain, angka itu harus diganti.
4. Skrip 16.2, 16.3, 16.5, dan 16.6, karena Skrip 16.7 memakai `NDVI.tif`, `NDVI_kelas.tif`, dan `Lereng_kelas.tif`.

Kesalahan umum

- Membiarkan alamat cadangan `DATA` dan `OUT`. Keduanya hanya alamat contoh, bukan alamat di komputer Anda. Perbaikan: atur `B3_DATA` dan `B3_OUT`, atau ganti alamat cadangan di tiap skrip dengan isi yang sama.
- **Mengabaikan pesan pengaman.** Skrip 16.1 dan 16.2 berhenti dengan pesan bila citra tidak terbaca. Perbaikan: baca pesannya, lalu periksa `DATA` dan `OUT`.
- **Memakai `OUT` yang berbeda antarskrip.** Skrip 16.5, 16.6, dan 16.7 mencari berkas di `OUT`. Perbaikan: atur `B3_OUT` sekali, supaya semua skrip memakai folder yang sama.
- **Menukar urutan band.** Hasil salah tanpa pesan. Perbaikan: cocokkan rata-rata tiap band dengan berkas asalnya.
- **Memakai angka 4 untuk data lain.** Luas salah tanpa pesan. Perbaikan: ambil ukuran piksel dari Skrip 16.1 dan hitung ulang.
- **Menjalankan skrip tidak berurutan.** Skrip 16.5, 16.6, dan 16.7 membaca berkas buatan skrip lain. Perbaikan: ikuti peta ketergantungan di awal bab.

Ringkasan dan latihan

- Raster adalah kisi piksel. Satu lembar angka disebut **band**.

- `QgsRasterLayer` membuka raster, dan `processing.run` menjalankan alat. Hasilnya berupa kamus.
- `gdal:merge` menumpuk band, `native:slope` dan kawan-kawannya menurunkan DEM, `gdal:cliprasterbymasklayer` memotong.
- Reklasifikasi mengubah rentang nilai menjadi kelas, dan laporan nilai unik menghitung piksel tiap kelas.
- NDVI = (inframerah dekat - merah) / (inframerah dekat + merah), dihitung dengan `QgsRasterCalculator`.
- Pengaman di awal skrip: `os.makedirs` membuat folder hasil, dan `raise SystemExit` menghentikan skrip bila citra tidak terbentuk atau tidak terbaca.
- Tiga hal yang diganti untuk data baru: **lokasi data dan folder hasil**, **nama berkas**, dan **nomor band atau batas kelas**.

Latihan. Pada Skrip 16.1, periksa `Citra_B03.tif` (band hijau). Lalu pada Skrip 16.6, ubah batas vegetasi rapat menjadi 0,5 dan catat luasnya. Berapa hektare vegetasi rapat sekarang dibandingkan 12,81?

Tabel perbandingan: pekerjaan raster di menu dan di skrip

Pekerjaan di QGIS	Padanan di skrip	Contoh di KPH Contoh
Melihat Layer Properties	`lay.width()`, `lay.bandCount()`, `lay.crs()`	200 x 200 piksel, 1 band
Raster statistics	`native:rasterlayerstatistics`	Rata-rata `449.0`
Merge	`gdal:merge`	`Citra_4band.tif`
Hillshade, Slope, Aspect	`native:hillshade`, `native:slope`, `native:aspect`	Lereng maksimal 21,89 derajat
Contour	`gdal:contour`	17 garis kontur 5 m
Clip Raster by Mask Layer	`gdal:cliprasterbymasklayer`	DEM KPH Alpha 10000 piksel
Reclassify by table	`native:reclassifybytable`	Datar, landai, curam
Raster Calculator	`QgsRasterCalculator`	NDVI rata-rata 0,735
Zonal statistics	`native:zonalstatisticsfb`	NDVI per KPH

BAB 17: Dari Data ke Lembar Peta: Mewarnai, Memberi Label, dan Mencetak dengan Skrip

> **Berdasarkan Seri B4:** Skrip bab ini berasal dari Buku B4, *Kartografi dan Layout Peta* (Basic Class), berkas `skrip_4_1_kategori.py`, `skrip_4_2_klasifikasi.py`, `skrip_4_2b_raster.py`, `skrip_4_3_label.py`, `skrip_4_4_layout.py`, `skrip_4_5_ekspor.py`, dan `skrip_4_6_jembatan.py`. Di buku itu, simbologi, label, layout, dan ekspor dibahas lewat menu dan klik. Bab ini menunjukkan sisi skripnya.

> **Satu skrip di bab ini mengosongkan proyek Anda:** Skrip 17.7 (berkas `skrip_4_6_jembatan.py`) memanggil `proyek.clear()` pada baris 15. Perintah itu membuang semua layer dan layout dari proyek QGIS yang sedang terbuka, tanpa bertanya. Jalankan skrip itu hanya di proyek kosong atau proyek cadangan. Simpan pekerjaan Anda lebih dulu.

Studi kasus: "Peta volume kayu untuk rapat besok"

Kepala Seksi minta peta volume kayu per petak untuk rapat besok pagi. Anda bisa mengklik menu satu per satu: warna, label, layout, ekspor. Hasilnya bagus. Tetapi bulan depan data volume berubah, dan semua klik harus diulang.

Dengan skrip, Anda menyusun peta itu sekali. Bulan depan Anda cukup menjalankannya lagi, dan peta baru jadi dalam hitungan detik.

Konsep: peta jadi dalam tiga kalimat

Peta yang enak dibaca dirakit bertahap, seperti menyusun poster. Layer adalah bahan, simbologi adalah cat, label adalah tulisan, layout adalah lembar posternya, dan ekspor adalah mencetaknya. Skrip PyQGIS mengerjakan tiap tahap itu dengan beberapa baris.

Ilustrasi 17.1: Lima tahap merakit peta



Lima istilah yang perlu Anda kenal sekarang:

- **Simbologi** (*symbology*): aturan warna dan bentuk gambar sebuah layer di peta.

- **Penyaji** (*renderer*): objek QGIS yang memegang aturan simbologi satu layer.
- **Simbol**: satu tampilan, misalnya satu isian poligon cokelat dengan garis tepi tipis.
- **Label**: tulisan di atas fitur peta, misalnya kode petak.
- **Layout**: lembar peta untuk dicetak, berisi peta, judul, legenda, skala, dan panah utara.

Sebelum mulai: urutan menjalankan dan bahan

Skrip di bab ini tidak berdiri sendiri semua. Beberapa skrip mengambil layer yang dibuat skrip sebelumnya, lewat nama layer di proyek. Karena itu urutannya penting. Tabel berikut merangkumnya.

Skrip	Berkas	Syarat sebelum dijalankan
17.1	`skrip_4_1_kategori.py`	Tidak ada. Memuat layer sendiri.
17.2	`skrip_4_2_klasifikasi.py`	Jalankan 17.1 lebih dulu, di proyek yang sama.
17.3	`skrip_4_2b_raster.py`	Tidak ada, tetapi sebaiknya di proyek terpisah (lihat Jebakan 17.3).
17.4	`skrip_4_3_label.py`	Jalankan 17.1 lebih dulu.
17.5	`skrip_4_4_layout.py`	Jalankan 17.1, 17.2, dan 17.4 di proyek yang sama. Jangan jalankan 17.3.
17.6	`skrip_4_5_ekspor.py`	Jalankan 17.5 lebih dulu.
17.7	`skrip_4_6_jembatan.py`	Tidak ada, tetapi mengosongkan proyek. Jalankan di proyek kosong.

Menjalankan. Buka Konsol Python QGIS atau Editor Skrip, seperti yang Anda siapkan di Bab 3. Tempel isi skrip, lalu jalankan. Semua skrip memakai satu folder kerja, yaitu isi variabel `FOLDER`.

Bahan. Buat satu folder kerja, misalnya `D:/KPH_Contoh`. Salin lima berkas ke dalamnya.

- Dari `paket-b4`: `Petak_KPH.gpkg` (layer `Petak_KPH`, 16 petak) dan `Sungai_Jalan.gpkg` (layer `Sungai` dan `Jalan`).
- Dari `paket-data`: `Hotspot_Nasional.gpkg` (12 titik panas) dan `DTM_Drone.tif` (peta tinggi).
- Dari `paket-b1`: `Hotspot_derajat.gpkg`. Hanya Skrip 17.7 yang memakainya.

Di setiap skrip, `FOLDER` tertulis `r"D:/KPH_Contoh"`, yaitu jalur komputer penulis. Ganti dengan jalur folder Anda. Gunakan garis miring biasa `/`, seperti dijelaskan di Bab 8.

Pengujian. Semua hasil di bab ini dijalankan di QGIS 4.0.2 tanpa jendela utama (mode tanpa tampilan). Urutan uji: 17.1, 17.2, 17.4, 17.5, 17.6 dalam satu proyek. Skrip 17.3 dan 17.7 diuji sendiri-sendiri. Hasil tampilan visual dibaca dari berkas PDF dan PNG yang dihasilkan.

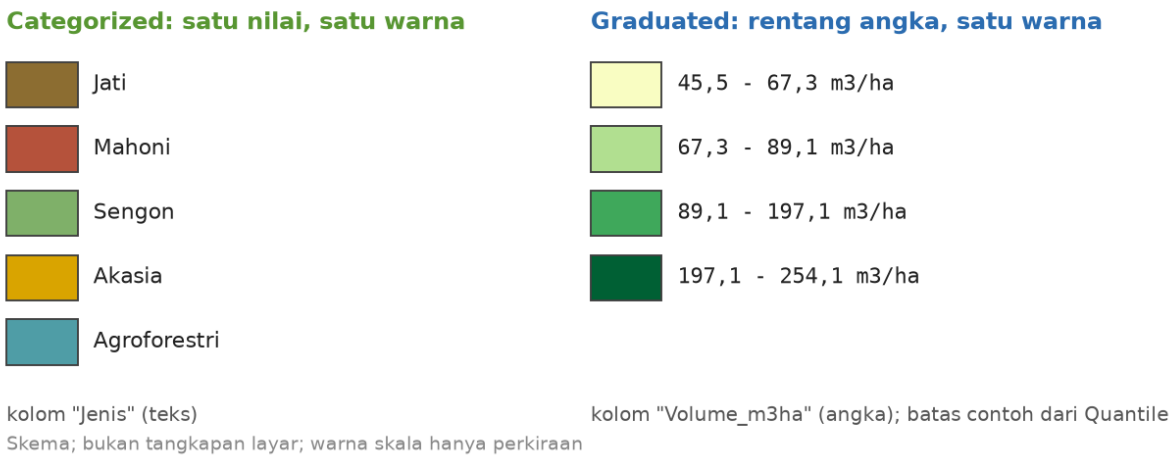
Skrip 17.1: Muat layer dan beri warna per jenis

Berkas `skrip_4_1_kategori.py` sama dengan yang dipakai di Buku B4. Di dalam berkasnya tertulis "Skrip 4.1" mengikuti penomoran buku itu.

Tujuan. Memuat empat layer KPH Contoh, mewarnai petak menurut jenis tanaman, lalu memberi gaya pada sungai, jalan, dan titik panas.

Bahan. Folder kerja berisi `Petak_KPH.gpkg`, `Sungai_Jalan.gpkg`, dan `Hotspot_Nasional.gpkg`. Skrip ini memuat layer sendiri, jadi tidak butuh skrip lain lebih dulu. Proyek yang sedang terbuka langsung berubah.

Ilustrasi 17.2: Perbedaan Categorized dan Graduated



Gambar di atas menunjukkan dua cara mewarnai. Skrip ini memakai cara kiri: **Categorized**, satu nilai teks satu warna. Cara kanan, **Graduated**, dipakai di Skrip 17.2.

Skrip ini bekerja dalam delapan tahap. Gambar berikut menunjukkan urutannya, lengkap dengan nomor baris.

Ilustrasi 17.3: Alur skrip 17.1



Skrip utuh:

```python
# Skrip 4.1: memuat empat layer KPH Contoh, lalu menata simbologinya.
# Petak diwarnai per jenis tanaman (Categorized); garis dan titik memakai Single Symbol.
from qgis.core import (QgsProject, QgsVectorLayer, QgsFillSymbol, QgsLineSymbol, QgsMarkerSymbol,
                       QgsRendererCategory, QgsCategorizedSymbolRenderer, QgsSingleSymbolRenderer,
                       QgsCoordinateReferenceSystem)

FOLDER = r"D:/KPH_Contoh"   # ganti dengan folder kerja Anda
proyek = QgsProject.instance()
proyek.setCrs(QgsCoordinateReferenceSystem("EPSG:32749"))   # CRS proyek: UTM 49S (meter)

def muat(berkas, lapisan, nama):
    layer = QgsVectorLayer("%s/%s|layername=%s" % (FOLDER, berkas, lapisan), nama, "ogr")
    assert layer.isValid(), "Layer tidak terbaca: " + berkas
    proyek.addMapLayer(layer)          # layer yang ditambah belakangan berada di atas
    return layer

petak = muat("Petak_KPH.gpkg", "Petak_KPH", "Petak_KPH")
sungai = muat("Sungai_Jalan.gpkg", "Sungai", "Sungai")
jalan = muat("Sungai_Jalan.gpkg", "Jalan", "Jalan")
titik = muat("Hotspot_Nasional.gpkg", "Hotspot_Nasional", "Hotspot_Nasional")

# Petak: satu warna untuk tiap jenis
warna = {"Jati": "#8C6D31", "Mahoni": "#B5523B", "Sengon": "#7FB069",
         "Akasia": "#D9A400", "Agroforestri": "#4F9DA6"}
kategori = []
for nama, hexa in warna.items():
    simbol = QgsFillSymbol.createSimple({"color": hexa, "outline_color": "#404040", "outline_width": "0.3"})
    kategori.append(QgsRendererCategory(nama, simbol, nama))
petak.setRenderer(QgsCategorizedSymbolRenderer("Jenis", kategori))

# Sungai: garis biru; Jalan: garis cokelat putus-putus; Titik panas: lingkaran merah
sungai.setRenderer(QgsSingleSymbolRenderer(QgsLineSymbol.createSimple(
    {"line_color": "#2B6CB0", "line_width": "0.8"})))
jalan.setRenderer(QgsSingleSymbolRenderer(QgsLineSymbol.createSimple(
    {"line_color": "#5A4632", "line_width": "0.5", "line_style": "dash"})))
titik.setRenderer(QgsSingleSymbolRenderer(QgsMarkerSymbol.createSimple(
    {"name": "circle", "color": "#E03C31", "outline_color": "#FFFFFF", "size": "2.6"})))

for layer in (petak, sungai, jalan, titik):
    layer.triggerRepaint()
print("Kategori dipasang:", [k.label() for k in petak.renderer().categories()])
print("Layer di proyek:", [l.name() for l in proyek.mapLayers().values()])
```

Penjelasan baris demi baris. Nomor baris mengikuti berkas di atas.

Baris	Tulisan	Artinya
1-2	`# Skrip 4.1 ...`	Komentar. Python melewatinya. Isinya menyebut bahwa skrip memuat empat layer: petak, sungai, jalan, dan titik panas.
3-5	`from qgis.core import (...)`	Panggil sembilan perkakas QGIS. Tanda kurung membolehkan daftar nama disambung ke beberapa baris.
7	`FOLDER = r"D:/KPH_Contoh"`	Isi kotak `FOLDER` dengan lokasi folder kerja. Huruf `r` membuat teks dibaca apa adanya. Bagian ini yang Anda ganti.
8	`proyek = QgsProject.instance()`	Ambil proyek QGIS yang sedang terbuka, lalu simpan di kotak `proyek`.

Baris	Tulisan	Artinya
9	`proyek.setCrs(...)`	Atur sistem koordinat proyek ke EPSG:32749, yaitu UTM zona 49 Selatan dengan satuan meter.
11-15	`def muat(berkas, lapisan, nama):`	Buat fungsi bernama `muat`, yaitu resep yang bisa dipakai berkali-kali dengan tiga isian (dibahas di Bab 7). Isinya membuat layer, memeriksanya, menambahkannya ke proyek, lalu mengembalikannya.
12	`layer = QgsVectorLayer("%s/%s...`	Susun alamat data dari tiga isian: folder, berkas, lalu garis tegak dan nama layer di dalam berkas. Tanda `%s` diganti isian berurutan oleh tanda `%` di ujung (format teks, Bab 9). Isian terakhir, `"ogr"`, adalah pembaca data vektor.
13	`assert layer.isValid(), "Layer tidak terbaca: " + berkas`	Pemeriksa (`assert`, Bab 10). Fungsi `isValid()` bernilai `True` bila data berhasil dibaca. Bila salah, skrip berhenti dengan `AssertionError` dan pesan `Layer tidak terbaca:` diikuti nama berkasnya. Tanpa baris ini, jalur yang salah menghasilkan layer kosong tanpa peringatan.
14	`proyek.addMapLayer(layer)`	Tambahkan layer ke proyek. Layer yang ditambah belakangan berada di atas.
17-20	`petak = muat(...)` dan tiga baris di bawahnya	Panggil `muat` empat kali. Hasilnya masuk ke kotak `petak`, `sungai`, `jalan`, dan `titik`.
23-24	`warna = {...}`	Kamus (Bab 5): pasangan nama jenis dan kode warna heksadesimal. Contoh `"#8C6D31"` adalah cokelat. Nama jenis harus persis sama dengan isi kolom `Jenis`.
25	`kategori = []`	Siapkan daftar kosong untuk menampung kategori.
26	`for nama, hexa in warna.items():`	Ulangi untuk tiap pasangan di kamus. Di tiap putaran, `nama` berisi nama jenis dan `hexa` berisi kode warnanya.
27	`simbol = QgsFillSymbol.createSimple({...})`	Buat simbol isian poligon. Kamus kecil di dalamnya mengatur warna isi, warna garis tepi, dan tebal garis tepi (dalam milimeter).
28	`kategori.append(QgsRendererCategory(...))`	Buat satu kategori dari tiga hal: nilai yang dicari, simbolnya, dan tulisan di legenda. Lalu tambahkan ke daftar.
29	`petak.setRenderer(QgsCategorizedSymbolRenderer("Jenis", kategori))`	Pasang penyaji berkategori pada petak. Penyaji melihat kolom `Jenis`, lalu memilih simbol menurut isinya.
32-33	`sungai.setRenderer(...)`	Beri sungai satu simbol garis biru setebal 0,8 mm. `QgsSingleSymbolRenderer` berarti satu simbol untuk semua fitur.
34-35	`jalan.setRenderer(...)`	Beri jalan garis cokelat putus-putus. Pengaturan `"line_style": "dash"` membuatnya putus-putus.
36-37	`titik.setRenderer(...)`	Beri titik panas simbol lingkaran merah dengan garis tepi putih, ukuran 2,6.
39-40	`for layer in (...): layer.triggerRepaint()`	Ulangi untuk keempat layer. Minta QGIS menggambar ulang agar warna baru terlihat.

Baris	Tulisan	Artinya
41	`print("Kategori dipasang:", [...])`	Tampilkan daftar tulisan kategori. Tanda kurung siku dengan `for` di dalamnya adalah *list comprehension* (Bab 6).
42	`print("Layer di proyek:", [...])`	Tampilkan nama semua layer di proyek.

Hasil. Layar menampilkan (berdasarkan uji pada QGIS 4.0.2):

```
HASIL
Kategori dipasang: ['Jati', 'Mahoni', 'Sengon', 'Akasia', 'Agroforestri']
Layer di proyek: ['Hotspot_Nasional', 'Jalan', 'Petak_KPH', 'Sungai']
```

Urutan nama pada baris kedua mengikuti catatan internal QGIS, bukan urutan di panel Layers. Di peta (saya merendernya sendiri untuk memastikan), petak tampil dalam lima warna menurut jenis, sungai berupa garis biru, jalan putus-putus cokelat, dan 12 titik merah.

Bagian yang boleh diubah.

- **Baris 7, lokasi data.** Ganti jalur dengan folder kerja Anda.
- **Baris 23-24, warna.** Ganti kode heksadesimal untuk mengubah warna jenis tertentu. Nama jenisnya tidak boleh diubah.
- Baris 27, `outline_width`. Angka `"0.3"` adalah tebal garis tepi petak. Ganti menjadi `"0.1"` untuk garis lebih tipis.
- Baris 33 dan 35, `line_width`. Ganti angkanya untuk mengatur tebal sungai dan jalan.
- Baris 35, `line_style`. Ganti `"dash"` menjadi `"solid"` untuk garis penuh.
- Baris 37, `size`. Angka `"2.6"` adalah ukuran titik panas.
- Baris 29, `"Jenis"`. Ganti dengan kolom teks lain untuk mewarnai menurut kolom itu.

Jebakan.

- **Jalur salah menghentikan skrip.** Bila `FOLDER` menunjuk folder yang tidak ada, baris 13 menghentikan skrip dengan `AssertionError: Layer tidak terbaca:` diikuti nama berkas pertama yang gagal. Pesan ini menolong, karena Anda tahu penyebabnya sebelum melihat peta kosong. Tanpa baris 13, skrip jalan terus, keempat layer kosong, dan peta tidak menampilkan apa pun. Bila pesan itu muncul, periksa baris 7 dan nama berkas.
- **Jenis yang tidak ada di kamus.** Misalnya Anda menghapus `"Akasia"` dari kamus `warna`. Petak berjenis Akasia lalu tidak punya kategori. Uji menunjukkan `categoryIndexForValue("Akasia")` bernilai `-1`, artinya tidak ada kategori yang cocok. Petak itu tidak tergambar di peta: pada hasil render saya, petak B03 (Akasia) tampil putih tanpa isian, dan legenda tidak memuat Akasia.
- **Komentar bisa tertinggal dari isi.** Python tidak memeriksa komentar. Bila Anda menambah layer, ubah juga komentar di baris 1. Saat membaca skrip orang lain, percayai kodenya, bukan komentarnya.
- **Skrip berikutnya menimpa warna.** Skrip 17.2 memasang penyaji baru pada layer `Petak_KPH`. Warna per jenis dari skrip ini lalu hilang, diganti warna menurut volume.

Latihan variasi.

1. Ganti warna Jati menjadi hijau tua `#2E7D32`.
2. Buat sungai lebih tebal, yaitu 1,2.

Contoh jawaban, hanya bagian yang berubah:

```python
warna = {"Jati": "#2E7D32", "Mahoni": "#B5523B", "Sengon": "#7FB069",
         "Akasia": "#D9A400", "Agroforestri": "#4F9DA6"}
```

```python
sungai.setRenderer(QgsSingleSymbolRenderer(QgsLineSymbol.createSimple(
    {"line_color": "#2B6CB0", "line_width": "1.2"})))
```

Saya menambah satu baris `print` untuk memeriksa. Hasilnya `Warna Jati: #2e7d32 | tebal sungai: 1.2`.

> **Pola yang bisa dipakai ulang:** Skrip ini adalah kerangka "muat, siapkan warna, pasang penyaji". Untuk layer lain, ganti tiga hal: **lokasi dan nama layer** (baris 7 dan 17-20), **kolom yang dipakai** (baris 29), dan **kamus warna** (baris 23-24). Kamus warna harus memuat semua nilai yang ada di kolom itu.

Skrip 17.2: Klasifikasi angka menjadi kelas warna

Berkas `skrip_4_2_klasifikasi.py`. Di dalam berkasnya tertulis "Skrip 4.2".

Tujuan. Mewarnai petak menurut `Volume_m3ha` dalam empat kelas, membandingkan tiga metode pembagian kelas, dan menampilkan petak yang belum diukur sebagai layer abu-abu tersendiri.

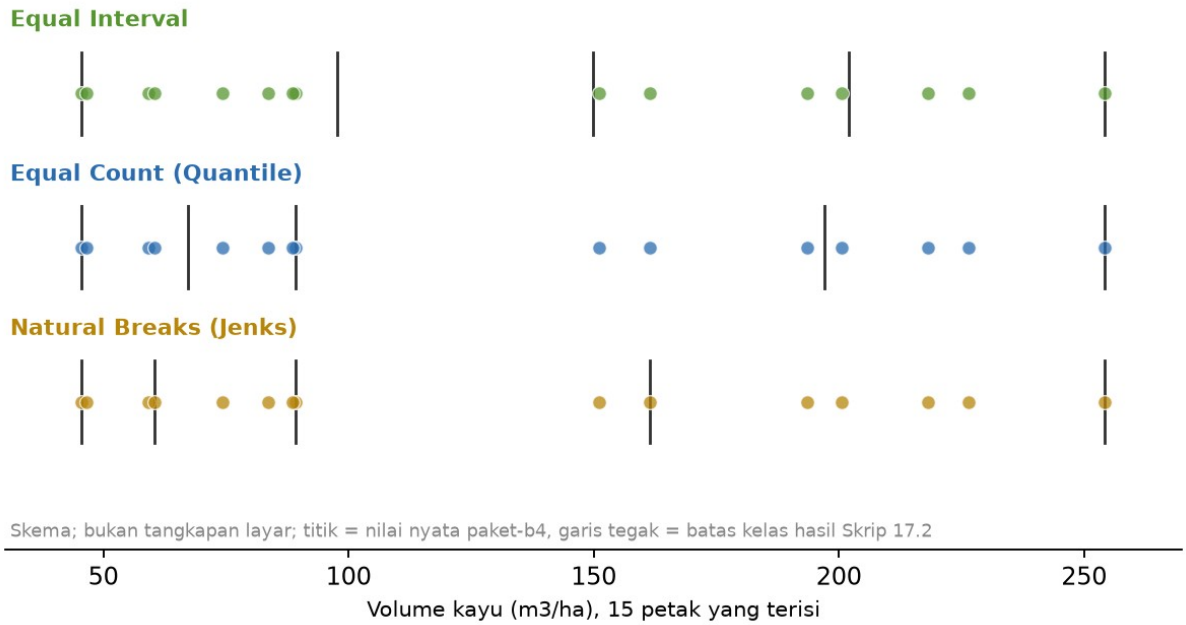
Bahan. Jalankan Skrip 17.1 lebih dulu, di proyek yang sama, agar layer `Petak_KPH` sudah ada. Folder kerja juga memuat `Petak_KPH.gpkg`, karena baris 43 membukanya sekali lagi. Dari 16 petak, 15 punya volume dan satu petak, `G08`, kosong.

Angka seperti volume tidak cocok diwarnai satu warna per nilai, karena hampir semua nilainya berbeda. Solusinya membagi angka menjadi **kelas** (rentang), lalu memberi satu warna untuk tiap kelas. Penyaji untuk ini bernama **Graduated**.

Bayangkan Anda membagi murid ke dalam kelas. Ada tiga cara membaginya, dan hasilnya berbeda.

- **Equal Interval**: tiap kelas selebar sama. Rentang nilai dipotong rata, tanpa melihat banyaknya petak.
- **Quantile** (di QGIS bernama *Equal Count*): tiap kelas berisi petak yang jumlahnya hampir sama.
- **Natural Breaks** (juga disebut *Jenks*): batas kelas dipasang di celah besar antar kelompok nilai.

Ilustrasi 17.4: Tiga metode membagi 15 nilai volume



Gambar di atas memakai 15 nilai volume yang nyata. Perhatikan Equal Interval: kelas keduanya kosong, karena tidak ada petak bervolume antara 98 dan 150. Skrip ini menghitung semua itu.

Ilustrasi 17.5: Alur skrip 17.2



Skrip utuh:

```python
# Skrip 4.2: simbologi Graduated untuk Volume_m3ha, membandingkan metode klasifikasi,
# dan menangani petak yang nilainya kosong (NULL)
from qgis.core import (QgsProject, QgsStyle, QgsGraduatedSymbolRenderer, QgsClassificationEqualInterval,
```

```
                      QgsClassificationQuantile, QgsClassificationJenks, QgsFillSymbol, QgsVectorLayer,
                      QgsSingleSymbolRenderer, QgsSymbolLayerUtils)

FOLDER = r"D:/KPH_Contoh"   # ganti dengan folder kerja Anda

petak = QgsProject.instance().mapLayersByName("Petak_KPH")[0]

# 1. Periksa nilai kosong. Metode Quantile dan Natural Breaks membacanya sebagai nol.
kosong = [f["ID_Petak"] for f in petak.getFeatures() if f["Volume_m3ha"] is None]
print("Petak tanpa nilai:", kosong)

# 2. Saring sementara: hanya petak yang terisi ikut dihitung
petak.setSubsetString('"Volume_m3ha" IS NOT NULL')

def pasang(kelas_metode, jumlah=4):
    metode = kelas_metode()     # objek metode baru tiap kali dipakai
    rend = QgsGraduatedSymbolRenderer("Volume_m3ha")
    rend.setSourceSymbol(QgsFillSymbol.createSimple({"outline_color": "#404040", "outline_width": "0.3"}))
    rend.setClassificationMethod(metode)
    rend.updateClasses(petak, jumlah)
    rend.updateColorRamp(QgsStyle.defaultStyle().colorRamp("YlGn"))   # skala kuning ke hijau, objek baru tiap
kali
    return rend

for kelas_metode in (QgsClassificationEqualInterval, QgsClassificationQuantile, QgsClassificationJenks):
    rend = pasang(kelas_metode)
    print(rend.classificationMethod().name())
    for i, r in enumerate(rend.ranges()):
        # nilai tepat di batas masuk kelas bawahnya; kelas pertama memuat nilai terkecil
        n = sum(1 for f in petak.getFeatures()
                if (r.lowerValue() < f["Volume_m3ha"] <= r.upperValue()) or (i == 0 and f["Volume_m3ha"] ==
r.lowerValue()))
        print("   %7.1f - %7.1f   %d petak" % (r.lowerValue(), r.upperValue(), n))

# 3. Pakai Quantile (4 kelas) pada layer, lalu lepas saringan
rend = pasang(QgsClassificationQuantile)
petak.setRenderer(rend)
petak.setSubsetString("")
petak.triggerRepaint()

# 4. Petak yang belum diukur ditampilkan sebagai layer tersendiri berwarna abu-abu
belum = QgsVectorLayer(FOLDER + "/Petak_KPH.gpkg|layername=Petak_KPH", "Belum diukur", "ogr")
belum.setSubsetString('"Volume_m3ha" IS NULL')
belum.setRenderer(QgsSingleSymbolRenderer(QgsFillSymbol.createSimple(
    {"color": "#D0D0D0", "outline_color": "#404040", "outline_width": "0.3", "style": "b_diagonal"})))
proyek = QgsProject.instance()
proyek.addMapLayer(belum, False)              # False: belum dimasukkan ke panel Layers
akar = proyek.layerTreeRoot()
posisi = akar.children().index(akar.findLayer(petak.id()))
akar.insertLayer(posisi, belum)               # tepat di atas Petak_KPH
print("Layer Belum diukur:", belum.featureCount(), "petak")
```

Penjelasan baris demi baris. Bagian yang sama dengan Skrip 17.1 tidak dijelaskan ulang. Baris 3-5 dan baris 7 mengikuti pola yang sama, hanya nama perkakasnya berbeda.

Baris	Tulisan	Artinya
3-5	`from qgis.core import (...)`	Panggil sepuluh perkakas. Satu di antaranya, `QgsSymbolLayerUtils`, tidak dipakai. Impor yang tersisa tidak berpengaruh.

Baris	Tulisan	Artinya
9	`petak = QgsProject.instance().mapLayersByName("Petak_KPH")[0]`	Cari layer di proyek menurut namanya. Hasilnya berupa daftar, jadi `[0]` mengambil yang pertama.
12	`kosong = [f["ID_Petak"] for f in ... if f["Volume_m3ha"] is None]`	Kumpulkan kode petak yang volumenya kosong. Nilai kosong di Python bernama `None`.
13	`print("Petak tanpa nilai:", kosong)`	Tampilkan petak yang kosong.
16	`petak.setSubsetString('"Volume_m3ha" IS NOT NULL')`	Pasang saringan sementara: layer hanya "melihat" petak yang volumenya terisi. Teks luar memakai kutip tunggal supaya nama kolom di dalamnya boleh memakai kutip ganda.
18	`def pasang(kelas_metode, jumlah=4):`	Fungsi dengan dua isian. Isian `jumlah` punya nilai bawaan 4, jadi boleh tidak diisi (Bab 7).
19	`metode = kelas_metode()`	`kelas_metode` berisi satu "cetakan" metode. Tanda kurung membuat satu objek baru dari cetakan itu. Komentar mengingatkan: objek baru tiap kali.
20	`rend = QgsGraduatedSymbolRenderer("Volume_m3ha")`	Buat penyaji bertingkat yang membaca kolom `Volume_m3ha`.
21	`rend.setSourceSymbol(...)`	Tetapkan simbol dasar: hanya garis tepi abu tua. Warna isi akan datang dari skala warna.
22	`rend.setClassificationMethod(metode)`	Pilih metode pembagian kelas.
23	`rend.updateClasses(petak, jumlah)`	Hitung batas kelas dari data layer, sebanyak `jumlah` kelas.
24	`rend.updateColorRamp(...colorRamp("YlGn"))`	Pasang skala warna `YlGn`, dari kuning ke hijau. Objek skala warna dibuat baru tiap kali.
25	`return rend`	Kembalikan penyaji yang sudah jadi.
27	`for kelas_metode in (A, B, C):`	Ulangi tiga kali, sekali untuk tiap metode. Nama metode disimpan di dalam tanda kurung seperti nilai biasa.
28-29	`rend = pasang(kelas_metode), print(...)`	Buat penyaji dengan metode itu, lalu tampilkan nama metodenya.
30	`for i, r in enumerate(rend.ranges()):`	Ulangi untuk tiap kelas. `enumerate` menambahkan nomor urut `i` dari 0 (Bab 6).
32-33	`n = sum(1 for f in ... if (...) or (...))`	Hitung petak di kelas ini. Tiap petak yang memenuhi syarat menyumbang angka 1. Syarat pertama, `batas_bawah < nilai <= batas_atas`, adalah perbandingan berantai. Syarat kedua membuat nilai terkecil ikut masuk kelas pertama.
34	`print("  %7.1f – %7.1f %d petak" % (...))`	Cetak batas bawah, batas atas, dan jumlah petak. `%7.1f` berarti angka desimal, satu angka di belakang koma, lebar 7 karakter.

Baris	Tulisan	Artinya
37-38	`rend = pasang(QgsClassificationQuantile), petak.setRenderer(rend)`	Pilih Quantile, empat kelas, lalu pasang di layer.
39-40	`petak.setSubsetString(""), petak.triggerRepaint()`	Lepas saringan, supaya petak kosong muncul kembali di peta. Lalu gambar ulang.
43	`belum = QgsVectorLayer(FOLDER + "/Petak_KPH.gpkg...`	Buka data yang sama sekali lagi sebagai layer baru bernama `Belum diukur`.
44	`belum.setSubsetString('"Volume_m3ha" IS NULL')`	Saringan kebalikannya: hanya petak yang volumenya kosong.
45-46	`belum.setRenderer(...)`	Beri simbol isian abu-abu dengan pola garis miring (`b_diagonal`).
47	`proyek = QgsProject.instance()`	Ambil proyek dan simpan di kotak `proyek`. Baris 9 dan 38 tadi memanggilnya langsung.
48	`proyek.addMapLayer(belum, False)`	Daftarkan layer ke proyek. Isian `False` berarti belum dimasukkan ke panel Layers.
49	`akar = proyek.layerTreeRoot()`	Ambil "akar pohon" panel Layers.
50	`posisi = akar.children().index(akar.findLayer(petak.id()))`	Cari nomor urut `Petak_KPH` di panel Layers.
51	`akar.insertLayer(posisi, belum)`	Sisipkan `Belum diukur` pada nomor urut itu, sehingga tepat di atas `Petak_KPH`.
52	`print("Layer Belum diukur:", ...)`	Tampilkan jumlah petak di layer itu.

Hasil. Layar menampilkan (berdasarkan uji pada QGIS 4.0.2):

```
HASIL
Petak tanpa nilai: ['G08']
Equal Interval
      45.5 -    97.7   8 petak
      97.7 -   149.8   0 petak
     149.8 -   202.0   4 petak
     202.0 -   254.1   3 petak
Equal Count (Quantile)
      45.5 -    67.3   4 petak
      67.3 -    89.1   4 petak
      89.1 -   197.1   3 petak
     197.1 -   254.1   4 petak
Natural Breaks (Jenks)
      45.5 -    60.4   4 petak
      60.4 -    89.1   4 petak
      89.1 -   161.4   2 petak
     161.4 -   254.1   5 petak
Layer Belum diukur: 1 petak
```

Dari hasil ini Anda tahu tiga hal. Pertama, `G08` kosong. Kedua, Equal Interval menyisakan satu kelas tanpa petak. Ketiga, setiap metode menghasilkan 15 petak bila dijumlahkan.

Satu aturan batas kelas perlu Anda tahu. Nilai yang tepat berada di batas masuk ke kelas di bawahnya. Contohnya `A02` bervolume 89,1. Pada metode Quantile, nilai itu masuk kelas 67,3 sampai 89,1, bukan kelas berikutnya.

Bagian yang boleh diubah.

- Baris 18, `jumlah=4`. Ganti angka bawaan untuk mengubah jumlah kelas. Anda juga boleh menulis `pasang(kelas_metode, 5)`.
- Baris 24, `"YlGn"`. Ganti dengan nama skala warna lain, misalnya `"Blues"`. Uji: `"Blues"` berjalan baik dan menghasilkan warna kelas dari `#f7fbff` sampai `#08306b`.
- **Baris 37, metode.** Ganti `QgsClassificationQuantile` dengan `QgsClassificationJenks` atau `QgsClassificationEqualInterval`.
- **Baris 12, 16, 20, 33, dan 44, nama kolom.** Ganti `Volume_m3ha` dengan kolom angka lain. Ganti di semua baris itu.
- Baris 45, `"color": "#D0D0D0"`. Ganti untuk mengubah warna abu-abu petak kosong.

Jebakan.

- **Skrip 17.1 belum dijalankan.** Layar menampilkan `IndexError: list index out of range` pada baris 9. Artinya daftar hasil pencarian kosong, karena layer `Petak_KPH` tidak ada di proyek. Jalankan Skrip 17.1 dulu.
- **Nilai kosong dibaca sebagai nol.** Baris 11 menyatakan bahwa Quantile dan Natural Breaks membaca nilai kosong sebagai nol. Saya mengujinya dengan membuang saringan di baris 16. Hasilnya, batas bawah kelas pertama Quantile menjadi 0,0 dan Natural Breaks menghasilkan kelas `0.0 – 0.0`. Karena itu skrip menyaring dulu.
- **Salah ketik nama kolom di baris 12.** Menulis `f["Volumee"]` menghasilkan `KeyError: 'Volumee'`. Salin nama kolom dari tabel atribut.
- **Salah ketik nama kolom di baris 16.** Menulis `"Volumee" IS NOT NULL` tidak menghasilkan galat di baris itu. Saringan tidak bekerja, dan galat muncul kemudian di baris 33: `TypeError: '<' not supported between instances of 'float' and 'NoneType'`. Petak `G08` ikut terhitung, dan angka kosongnya tidak bisa dibandingkan. Pelajarannya: galat bisa muncul jauh dari sumber salahnya.
- **Petak kosong tidak digambar.** Saya merender peta tanpa layer `Belum diukur`. Hasilnya, kotak petak `G08` kosong (putih), karena penyaji bertingkat tidak punya kelas untuk nilai kosong. Itulah alasan skrip menambah layer abu-abu tersendiri di baris 43-52. Jangan hapus layer itu bila petak kosong harus tetap terlihat.
- **Dijalankan ulang.** Setiap kali skrip ini dijalankan, satu layer `Belum diukur` baru ditambahkan lagi (uji: dua kali jalan memberi dua layer bernama `Belum diukur`). Hapus layer lama lebih dulu, atau mulai dari proyek baru.
- **Menimpa warna Skrip 17.1.** Setelah skrip ini jalan, petak berwarna menurut volume, bukan menurut jenis. Itu wajar. Untuk kembali, jalankan ulang bagian penyaji di Skrip 17.1.
- **Jangan membagi satu objek ke banyak penyaji.** Catatan Buku B4 menyebut bahwa memakai ulang satu objek skala warna pada beberapa penyaji membuat QGIS berhenti

mendadak. Karena itu baris 19 dan 24 membuat objek baru tiap kali. Hal ini tidak saya ulangi di sini. [CEK]

Latihan variasi.

1. Pasang metode Natural Breaks dengan **lima** kelas pada layer, bukan Quantile empat kelas.
2. Cetak batas kelas hasilnya.

Contoh jawaban. Ganti baris 37, lalu tambahkan satu baris `print`:

```
PYTHON
rend = pasang(QgsClassificationJenks, 5)
print("Pilihan latihan:", [(round(r.lowerValue(), 1), round(r.upperValue(), 1)) for r in rend.ranges()])
```

Layar menampilkan `Pilihan latihan: [(45.5, 60.4), (60.4, 89.1), (89.1, 161.4), (161.4, 226.4), (226.4, 254.1)]`.

> **Pola yang bisa dipakai ulang:** Kerangka klasifikasi: **saring nilai kosong, bungkus dalam fungsi, ulangi untuk tiap metode**. Untuk data lain, ganti tiga hal: **nama layer** (baris 9 dan 43), **nama kolom angka** (baris 12, 16, 20, 33, dan 44), dan **jumlah kelas** (baris 18). Fungsi `pasang` tidak perlu disentuh.

Skrip 17.3: Warna untuk peta tinggi dan bayangan bukit

Berkas `skrip_4_2b_raster.py`. Di dalam berkasnya tertulis "Skrip 4.2b". Skrip ini sedikit keluar dari alur, karena menangani raster, bukan vektor.

Tujuan. Memberi skala warna Viridis lima kelas pada peta tinggi (DTM), lalu membuat layer salinan yang menampilkan bayangan bukit.

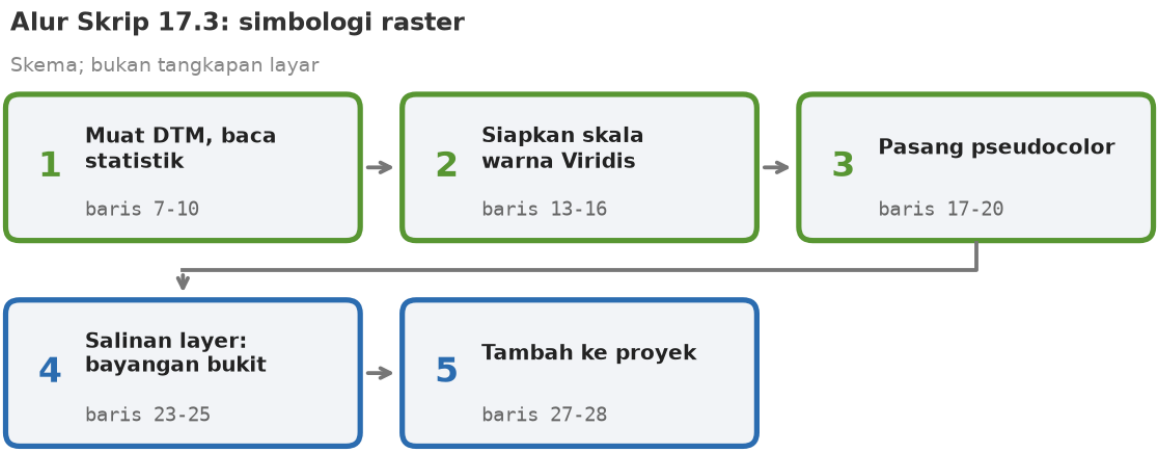
Bahan. Berkas `DTM_Drone.tif` di folder kerja. Skrip ini berdiri sendiri, karena memuat raster sendiri.

Raster adalah gambar dari kotak-kotak kecil (piksel), dan tiap kotak menyimpan satu angka. Di sini angkanya tinggi tempat dalam meter, dari 15 sampai 25 meter. Raster dibahas lebih lengkap di Bab 16.

Tiga istilah baru:

- **Pseudocolor**: tiap rentang angka diberi satu warna.
- **Hillshade**: bayangan bukit, yang memberi kesan relief dari arah cahaya.
- **Band**: satu lapis angka dalam raster. DTM hanya punya satu.

Ilustrasi 17.6: Alur skrip 17.3



Skrip utuh:

```python
# Skrip 4.2b: simbologi raster. Peta tinggi (DTM) diberi skala warna, lalu dibuat versi bayangan bukit
from qgis.core import (QgsProject, QgsRasterLayer, QgsStyle, QgsColorRampShader,
                       QgsRasterShader, QgsSingleBandPseudoColorRenderer, QgsHillshadeRenderer, Qgis)

FOLDER = r"D:/KPH_Contoh"   # ganti dengan folder kerja Anda

dtm = QgsRasterLayer(FOLDER + "/DTM_Drone.tif", "DTM_Drone")
assert dtm.isValid(), "Raster tidak terbaca: " + FOLDER + "/DTM_Drone.tif"
statistik = dtm.dataProvider().bandStatistics(1)
print("Tinggi terendah %.1f m, tertinggi %.1f m" % (statistik.minimumValue, statistik.maximumValue))

# 1. Pseudocolor: tiap rentang tinggi diberi warna dari skala Viridis
fungsi = QgsColorRampShader(statistik.minimumValue, statistik.maximumValue)
fungsi.setSourceColorRamp(QgsStyle.defaultStyle().colorRamp("Viridis"))
fungsi.setClassificationMode(Qgis.ShaderClassificationMethod.EqualInterval)
fungsi.classifyColorRamp(5)                        # 5 kelas warna sama lebar
bayangan_warna = QgsRasterShader()
bayangan_warna.setRasterShaderFunction(fungsi)
dtm.setRenderer(QgsSingleBandPseudoColorRenderer(dtm.dataProvider(), 1, bayangan_warna))
print("Kelas warna:", [(round(i.value, 1), i.color.name()) for i in fungsi.colorRampItemList()])

# 2. Hillshade (bayangan bukit) pada salinan layer: arah cahaya 315 derajat, tinggi matahari 45 derajat
bukit = QgsRasterLayer(FOLDER + "/DTM_Drone.tif", "DTM_Bayangan")
bukit.setRenderer(QgsHillshadeRenderer(bukit.dataProvider(), 1, 315, 45))
print("Renderer bukit:", bukit.renderer().type())

QgsProject.instance().addMapLayer(dtm)
QgsProject.instance().addMapLayer(bukit)
```

Penjelasan baris demi baris.

Baris	Tulisan	Artinya
2-3	`from qgis.core import (...)`	Panggil delapan perkakas, termasuk `Qgis` yang memuat daftar pilihan bernama.
5	`FOLDER = ...`	Lokasi folder kerja. Ganti dengan jalur Anda.

Baris	Tulisan	Artinya
7	`dtm = QgsRasterLayer(FOLDER + "/DTM_Drone.tif", "DTM_Drone")`	Buka raster sebagai layer. Hanya dua isian: alamat dan nama. Raster tidak butuh isian `"ogr"`.
8	`assert dtm.isValid(), "Raster tidak terbaca: " + ...`	Pemeriksa (Bab 10). Bila raster tidak terbaca, skrip berhenti dengan `AssertionError` dan pesan `Raster tidak terbaca:` diikuti jalur berkasnya.
9	`statistik = dtm.dataProvider().bandStatistics(1)`	Minta penyedia data layer menghitung statistik band nomor 1: terendah, tertinggi, rata-rata.
10	`print("Tinggi terendah %.1f m, ..." % (...))`	Tampilkan tinggi terendah dan tertinggi, satu desimal.
13	`fungsi = QgsColorRampShader(min, max)`	Buat pengatur warna untuk rentang dari tinggi terendah sampai tertinggi.
14	`fungsi.setSourceColorRamp(...colorRamp("Viridis"))`	Pakai skala warna Viridis. Skala ini bergerak dari ungu tua ke kuning.
15	`fungsi.setClassificationMode(...EqualInterval)`	Pembagian kelas dengan selebar sama. Pilihan namanya diambil dari perkakas `Qgis`.
16	`fungsi.classifyColorRamp(5)`	Bagi skala warna menjadi lima kelas.
17-18	`bayangan_warna = QgsRasterShader()` lalu `setRasterShaderFunction(fungsi)`	Bungkus pengatur warna ke dalam "pewarna raster". Nama kotak `bayangan_warna` membingungkan, karena isinya pewarna biasa, bukan bayangan bukit.
19	`dtm.setRenderer(QgsSingleBandPseudoColorRenderer(...))`	Pasang penyaji pseudocolor pada layer: band 1, diwarnai oleh pewarna tadi.
20	`print("Kelas warna:", [(round(i.value, 1), i.color.name()) ...])`	Tampilkan tiap kelas: nilai batasnya dan kode warnanya.
23	`bukit = QgsRasterLayer(...)`	Buka berkas yang sama sekali lagi sebagai layer baru, `DTM_Bayangan`.
24	`bukit.setRenderer(QgsHillshadeRenderer(provider, 1, 315, 45))`	Pasang penyaji bayangan bukit. Angka 315 adalah arah cahaya dalam derajat (dari barat laut). Angka 45 adalah tinggi matahari dalam derajat.
25	`print("Renderer bukit:", bukit.renderer().type())`	Tampilkan jenis penyaji, untuk memastikan terpasang.
27-28	`QgsProject.instance().addMapLayer(...)`	Tambahkan kedua layer ke proyek. `bukit` ditambah belakangan, jadi berada di atas `dtm`.

Hasil. Layar menampilkan (berdasarkan uji pada QGIS 4.0.2):

```
HASIL
Tinggi terendah 15.0 m, tertinggi 25.0 m
Kelas warna: [(15.0, '#440154'), (17.5, '#3b528b'), (20.0, '#21908d'), (22.5, '#5dc963'), (25.0, '#fde725')]
Renderer bukit: hillshade
```

Lima nilai batas tampil: 15,0, 17,5, 20,0, 22,5, dan 25,0 meter. Warna pertama `#440154` adalah ungu tua, dan warna terakhir `#fde725` adalah kuning. Data DTM latihan memang buatan, jadi rentang tingginya sempit.

Bagian yang boleh diubah.

- **Baris 7 dan 23, nama berkas.** Ganti `DTM_Drone.tif` dengan raster lain. Ganti di kedua baris.
- **Baris 14, nama skala warna.** Contoh: `"Magma"`.
- **Baris 16, jumlah kelas.** Angka 5 boleh diganti.
- **Baris 24, 315 dan 45.** Ubah arah dan tinggi cahaya. Contoh `225, 30` memberi bayangan lebih panjang dari arah barat daya.

Jebakan.

- **Berkas tidak ditemukan menghentikan skrip.** Bila jalur salah, baris 8 menghentikan skrip dengan `AssertionError: Raster tidak terbaca:` diikuti jalur lengkapnya. Tanpa baris itu, skrip tetap berjalan dan mencetak tinggi terendah sekitar 1,8 kali 10 pangkat 308, tinggi tertinggi minus angka yang sama, lalu `Kelas warna: []`. Angka raksasa itu tanda raster tidak terbaca. Bila pesan muncul, periksa jalur di baris 5.
- **Nama skala warna salah.** Menulis `"Viridiss"` tidak menimbulkan galat. QGIS diam-diam memakai skala lain, dari biru ke merah. Uji: warna pertama `#0000ff`, warna terakhir `#cc0033`. Periksa `Kelas warna` pada hasil.
- **Bayangan bukit menutup peta.** Layer `DTM_Bayangan` berada di atas semua layer lain. Bila skrip ini dijalankan di proyek yang sama sebelum Skrip 17.5, layout hasil ekspor tertutup bidang abu-abu, dan petak tidak tampak. Hal ini saya lihat langsung dari hasil ekspor. Jalankan skrip ini di proyek terpisah, atau setelah layout diekspor.
- **Dua layer dari satu berkas.** Ini sengaja. Satu layer tidak bisa memakai dua penyaji sekaligus, jadi skrip membuat salinan.

Latihan variasi. Ubah skrip menjadi tiga kelas, skala warna Magma, dan cahaya dari arah 225 derajat dengan tinggi matahari 30 derajat.

Contoh jawaban, hanya bagian yang berubah:

```
PYTHON
fungsi.setSourceColorRamp(QgsStyle.defaultStyle().colorRamp("Magma"))
fungsi.classifyColorRamp(3)
bukit.setRenderer(QgsHillshadeRenderer(bukit.dataProvider(), 1, 225, 30))
```

Layar menampilkan `Kelas warna: [(15.0, '#000004'), (20.0, '#b6377a'), (25.0, '#fcfdbf')]` dan `Renderer bukit: hillshade`.

> **Pola yang bisa dipakai ulang:** Kerangka raster: **buka, baca statistik, siapkan pewarna, pasang penyaji**. Untuk raster lain, ganti tiga hal: **nama berkas** (baris 7 dan 23), **nama skala warna** (baris 14), dan **jumlah kelas** (baris 16).

Skrip 17.4: Label berisi kode petak dan volume

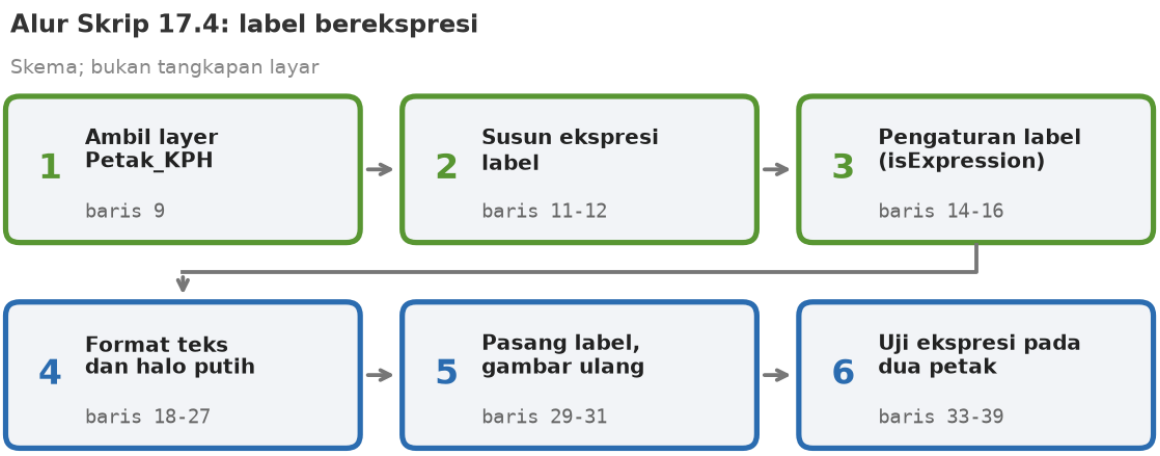
Berkas `skrip_4_3_label.py`. Di dalam berkasnya tertulis "Skrip 4.3".

Tujuan. Memberi label pada tiap petak berisi kode dan volumenya, dengan garis tepi putih agar terbaca, lalu menguji hasil ekspresi pada dua petak.

Bahan. Jalankan Skrip 17.1 lebih dulu, agar layer `Petak_KPH` sudah ada di proyek. Skrip ini tidak membaca berkas lain.

Label adalah tulisan di atas peta. Isinya bisa kolom biasa, bisa juga **ekspresi** (rumus kecil yang menyusun teks dari beberapa kolom). Garis tepi putih di sekeliling huruf disebut **halo**. Halo membuat huruf tetap terbaca di atas warna gelap.

Ilustrasi 17.7: Alur skrip 17.4



Skrip utuh:

```python
# Skrip 4.3: label petak berisi ID dan volume, dengan garis tepi putih agar terbaca
from qgis.core import (QgsProject, QgsPalLayerSettings, QgsTextFormat, QgsTextBufferSettings,
                       QgsVectorLayerSimpleLabeling, QgsExpression, QgsExpressionContext,
                       QgsExpressionContextUtils)
from qgis.PyQt.QtGui import QColor, QFont

FOLDER = r"D:/KPH_Contoh"   # ganti dengan folder kerja Anda

petak = QgsProject.instance().mapLayersByName("Petak_KPH")[0]

ekspresi = ('"ID_Petak" || \'\n\' || '
            'coalesce(to_string(round("Volume_m3ha")) || \' m3/ha\', \'belum diukur\')')

pengaturan = QgsPalLayerSettings()
pengaturan.fieldName = ekspresi
pengaturan.isExpression = True
```

```
teks = QgsTextFormat()
teks.setFont(QFont("Arial"))
teks.setSize(8)
teks.setColor(QColor("#202020"))
garis = QgsTextBufferSettings()
garis.setEnabled(True)
garis.setSize(0.8)
garis.setColor(QColor("white"))
teks.setBuffer(garis)
pengaturan.setFormat(teks)

petak.setLabeling(QgsVectorLayerSimpleLabeling(pengaturan))
petak.setLabelsEnabled(True)
petak.triggerRepaint()

# periksa hasil ekspresi pada dua petak
konteks = QgsExpressionContext()
konteks.appendScopes(QgsExpressionContextUtils.globalProjectLayerScopes(petak))
for f in petak.getFeatures():
    if f["ID_Petak"] in ("A01", "G08"):
        konteks.setFeature(f)
        print(f["ID_Petak"], "->", repr(QgsExpression(ekspresi).evaluate(konteks)))
```

Penjelasan baris demi baris. Garis tegak ganda `||` di dalam ekspresi berfungsi menyambung teks. Karena tanda garis tegak mengganggu tabel, sel di bawah menyebutnya "garis tegak ganda".

Baris	Tulisan	Artinya
2-5	`from qgis.core import (...)` dan `from qgis.PyQt.QtGui import QColor, QFont`	Panggil perkakas label dari QGIS, lalu perkakas warna dan huruf dari PyQt (perangkat jendela yang dipakai QGIS).
7	`FOLDER = ...`	Didefinisikan, tetapi tidak dipakai di skrip ini. Tidak berbahaya.
9	`petak = ...mapLayersByName("Petak_KPH")[0]`	Cari layer `Petak_KPH` di proyek.
11-12	`ekspresi = (... '...' ...)`	Rumus label, ditulis sebagai teks dalam dua potongan yang disambung otomatis oleh tanda kurung. Hasilnya, dalam bahasa QGIS: kode petak, lalu baris baru, lalu volume yang dibulatkan diikuti `m3/ha`. Bila volumenya kosong, tampil `belum diukur`. Penjelasan bagian-bagiannya ada di bawah tabel.
14	`pengaturan = QgsPalLayerSettings()`	Buat kotak pengaturan label yang masih kosong.
15-16	`pengaturan.fieldName = ekspresi, pengaturan.isExpression = True`	Isi pengaturan: teks yang dilabel adalah `ekspresi`, dan teks itu harus dibaca sebagai rumus, bukan nama kolom. Titik di tengah artinya "milik pengaturan ini" (Bab 11).
18	`teks = QgsTextFormat()`	Buat kotak format teks yang kosong.
19-21	`teks.setFont(QFont("Arial")), setSize(8), setColor(QColor("#202020"))`	Atur jenis huruf Arial, ukuran 8, dan warna abu sangat tua.

Baris	Tulisan	Artinya
22-26	`garis = QgsTextBufferSettings()` dan baris di bawahnya	Buat pengaturan halo. Nyalakan, atur tebal 0,8, warna putih, lalu pasang ke format teks.
27	`pengaturan.setFormat(teks)`	Pasang format teks ke pengaturan label.
29	`petak.setLabeling(QgsVectorLayerSimpleLabeling(pengaturan))`	Pasang pengaturan sebagai label layer. "Simple" berarti satu pengaturan untuk semua fitur.
30-31	`petak.setLabelsEnabled(True)`, `petak.triggerRepaint()`	Nyalakan label, lalu gambar ulang.
34-35	`konteks = QgsExpressionContext()` dan `appendScopes(...)`	Siapkan "ruang pandang" untuk menghitung ekspresi: variabel global, proyek, dan layer ini.
36-37	`for f in petak.getFeatures(): if f["ID_Petak"] in ("A01", "G08"):`	Ulangi untuk tiap petak, tetapi hanya proses petak `A01` dan `G08`.
38	`konteks.setFeature(f)`	Beritahu ruang pandang: petak ini yang sedang dihitung.
39	`print(..., repr(QgsExpression(ekspresi).evaluate(konteks)))`	Hitung ekspresi pada petak itu, lalu cetak dengan `repr`. Fungsi `repr` menampilkan tanda khusus apa adanya, jadi baris baru tampil sebagai `\n`.

Membaca ekspresi di baris 11-12. Rumus label itu terdiri dari empat bagian. Garis tegak ganda menyambung dua teks. `round` membulatkan angka ke bilangan bulat. `to_string` mengubah angka menjadi teks, supaya bisa disambung.

`coalesce` mengambil isian pertama yang tidak kosong, dan bila semuanya kosong dipakai isian terakhir. Karena angka kosong disambung dengan teks hasilnya kosong, petak `G08` jatuh ke teks `belum diukur`.

Dua tanda di baris 11-12 sering membingungkan. Pertama, `\'` adalah kutip tunggal yang ditulis di dalam teks berkutip tunggal. Garis miring terbalik di depannya berarti "ini bagian teks, bukan penutup".

Kedua, `\n` adalah baris baru. Karena itu label tampil dalam dua baris.

Hasil. Layar menampilkan (berdasarkan uji pada QGIS 4.0.2):

```
HASIL
A01 -> 'A01\n46 m3/ha'
G08 -> 'G08\nbelum diukur'
```

Petak `A01` bervolume 45,5, dan hasilnya dibulatkan menjadi 46. Di peta, tiap petak yang digambar punya label dua baris. Label `G08` berbunyi "belum diukur" bila petak itu digambar (lihat Jebakan).

Bagian yang boleh diubah.

- **Baris 11-12, ekspresi.** Ganti untuk menyusun label lain. Nama kolom ditulis dengan kutip ganda, dan teks biasa dengan kutip tunggal.
- **Baris 19, jenis huruf.** Ganti `"Arial"` dengan huruf yang terpasang di komputer Anda.
- **Baris 20, ukuran.** Ganti angka 8.
- **Baris 21, warna.** Ganti kode warna huruf.
- **Baris 24, tebal halo.** Ganti 0,8. Nilai 0 mematikan efek.
- **Baris 37, ID petak.** Ganti `"A01", "G08"` dengan petak yang ingin Anda periksa.

Jebakan.

- **Lupa garis miring terbalik.** Menulis ekspresi baru dengan kutip tunggal di dalam kutip tunggal, misalnya `'"ID_Petak" || ' - ' || "Jenis"'`, menghasilkan `TypeError: unsupported operand type(s) for -: 'str' and 'str'`. Python mengira tanda kutip di tengah menutup teks. Perbaikan: tulis `\' - \'`.
- **Skrip 17.1 belum dijalankan.** Layar menampilkan `IndexError: list index out of range` pada baris 9.
- **Nama layer diganti di panel Layers.** Baris 9 mencari layer bernama `Petak_KPH`. Bila Anda mengganti nama layer itu, pencarian menghasilkan daftar kosong dan layar menampilkan `IndexError` yang sama. Samakan nama di baris 9 dengan nama layer yang sekarang. Skrip 17.5 tidak mengubah nama layer, jadi skrip ini tetap bisa dijalankan sesudahnya.
- Label `G08` hilang setelah Skrip 17.2. Hasil render saya: bila skrip ini dijalankan setelah Skrip 17.1 saja, petak `G08` berlabel `belum diukur`. Tetapi bila Skrip 17.2 sudah jalan, penyaji bertingkat tidak punya kelas untuk nilai kosong, sehingga `G08` tidak digambar dan labelnya ikut hilang. Hanya 15 petak yang berlabel. Ekspresi `coalesce` tetap benar, seperti terlihat pada hasil uji skrip (baris 39). Yang berbeda adalah petak itu tidak dirender. Menurut saya, label hanya dibuat untuk fitur yang digambar. [kemungkinan]
- `FOLDER` tidak dipakai. Jalur yang salah tidak berpengaruh di skrip ini.

Latihan variasi. Ubah label menjadi kode petak diikuti jenisnya, dalam satu baris, dipisah ` - `. Contoh: `A01 - Jati`.

Contoh jawaban. Ganti baris 11-12 dengan satu baris:

```python
ekspresi = '"ID_Petak" || \' - \' || "Jenis"'
```

Layar menampilkan `A01 -> 'A01 - Jati'` dan `G08 -> 'G08 - Agroforestri'`.

> **Pola yang bisa dipakai ulang:** Kerangka label: **ekspresi, pengaturan, format teks, pasang**. Untuk layer lain, ganti tiga hal: **nama layer** (baris 9), **ekspresi** (baris 11-12), dan **gaya huruf dan halo** (baris 19-26). Sisanya tidak perlu disentuh.

Skrip 17.5: Menyusun lembar peta A4

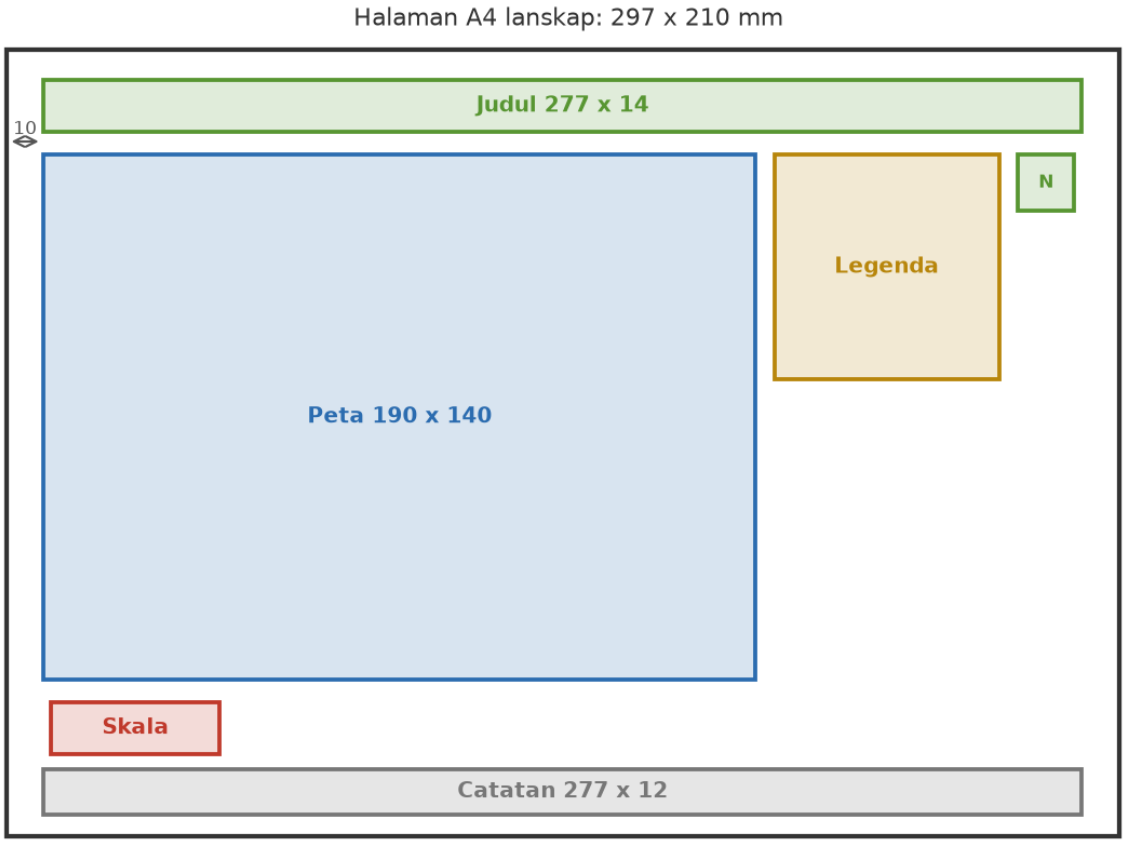
Berkas `skrip_4_4_layout.py`. Di dalam berkasnya tertulis "Skrip 4.4". Skrip ini yang terpanjang kedua di bab ini, tetapi isinya berulang. Ia memasang enam item satu per satu, dengan pola yang hampir sama.

Tujuan. Menyusun layout peta A4 lanskap yang lengkap (peta, judul, legenda, batang skala, panah utara, catatan), lalu mengekspornya ke PDF dan PNG.

Bahan. Layer hasil Skrip 17.1, 17.2, dan 17.4 sudah ada di proyek: `Petak_KPH`, `Hotspot_Nasional`, `Sungai`, `Jalan`, dan `Belum diukur` (dari Skrip 17.2). Skrip ini mengekspor dua berkas ke `FOLDER`: `Peta_KPH.pdf` dan `Peta_KPH.png`.

Lembar peta dibangun dari **item**: kotak-kotak yang ditaruh di halaman, masing-masing berisi satu hal. Letak dan ukuran item diukur dalam **milimeter**. Titik acuannya pojok kiri atas halaman. Bayangkan menempel stiker di kertas A4 dengan penggaris.

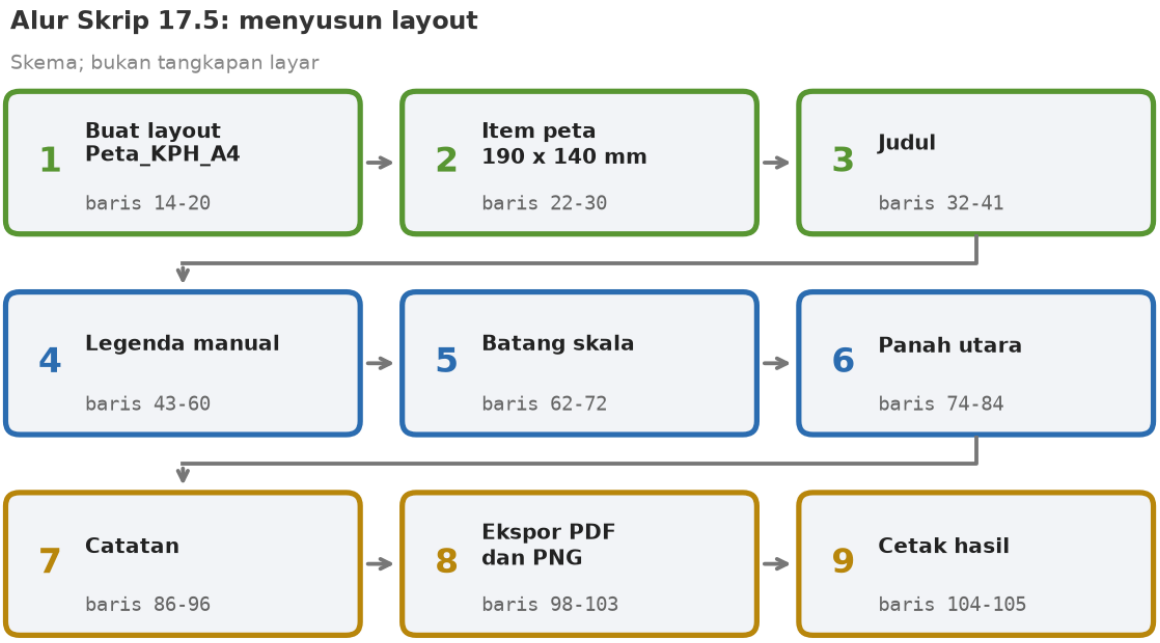
Ilustrasi 17.8: Letak item di halaman A4



Skema; bukan tangkapan layar; ukuran dan titik pojok kiri atas sesuai Skrip 17.5, legenda dan skala hanya perkiraan ukuran

Pada skala 1:3000, satu milimeter di kertas sama dengan 3000 milimeter, atau 3 meter, di lapangan. Maka peta 190 kali 140 milimeter menampilkan area 570 kali 420 meter. Data latihan berukuran 400 kali 400 meter, jadi pas.

Ilustrasi 17.9: Alur skrip 17.5



Skrip utuh:

```python
# Skrip 4.4: menyusun layout peta A4 lanskap (peta, judul, legenda, skala, panah utara) dan mengekspornya
from qgis.core import (QgsProject, QgsPrintLayout, QgsLayoutItemMap, QgsLayoutItemLabel, QgsLayoutItemLegend,
                       QgsLayoutItemScaleBar, QgsLayoutItemPicture, QgsLayoutItemShape, QgsLayoutPoint,
QgsLayoutSize,
                       QgsLayoutExporter, QgsRectangle, QgsApplication, QgsLayerTree, QgsLegendStyle,
                       QgsTextFormat, Qgis, QgsMapLayerLegendUtils)
from qgis.PyQt.QtCore import Qt
from qgis.PyQt.QtGui import QColor, QFont
import os

FOLDER = r"D:/KPH_Contoh"   # ganti dengan folder kerja Anda
proyek = QgsProject.instance()
mm = Qgis.LayoutUnit.Millimeters

lama = proyek.layoutManager().layoutByName("Peta_KPH_A4")
if lama is not None:                            # dijalankan ulang: buang layout lama agar nama tidak bentrok
    proyek.layoutManager().removeLayout(lama)
tata = QgsPrintLayout(proyek)
tata.initializeDefaults()                       # halaman A4 lanskap (297 x 210 mm)
tata.setName("Peta_KPH_A4")
proyek.layoutManager().addLayout(tata)

# 1. Peta: ukuran 190 x 140 mm pada skala 1:3000
peta = QgsLayoutItemMap(tata)
peta.setCrs(proyek.crs())
peta.attemptMove(QgsLayoutPoint(10, 28, mm))
peta.attemptResize(QgsLayoutSize(190, 140, mm))   # ukuran dan letak diatur lebih dulu
peta.zoomToExtent(QgsRectangle(312000, 9996000, 312400, 9996400))   # pas ke area KPH
peta.setScale(3000)                               # skala diatur terakhir
```

```
peta.setFrameEnabled(True)
tata.addLayoutItem(peta)

# 2. Judul
judul = QgsLayoutItemLabel(tata)
judul.setText("PETA KPH CONTOH: VOLUME KAYU PER PETAK")
fmt = QgsTextFormat()
fmt.setFont(QFont("Arial", 18, QFont.Weight.Bold))
fmt.setSize(18)
judul.setTextFormat(fmt)
judul.attemptMove(QgsLayoutPoint(10, 8, mm))
judul.attemptResize(QgsLayoutSize(277, 14, mm))
tata.addLayoutItem(judul)

# 3. Legenda: dibuat manual agar nama entrinya ramah dibaca
legenda = QgsLayoutItemLegend(tata)
legenda.setLinkedMap(peta)
legenda.setSyncMode(Qgis.LegendSyncMode.Manual)   # isi legenda tidak ikut berubah otomatis
akar = legenda.model().rootGroup()
nama_baru = {"Petak_KPH": "Volume kayu (m3/ha)", "Hotspot_Nasional": "Titik panas",
             "Sungai": "Sungai", "Jalan": "Jalan", "Belum diukur": "Petak belum diukur"}
for simpul in list(akar.children()):
    if simpul.name() in nama_baru:
        baru = nama_baru[simpul.name()]
        if len(legenda.model().layerLegendNodes(simpul)) == 1:   # entri tunggal: ganti label entri (nama layer
di proyek tetap)
            QgsMapLayerLegendUtils.setLegendNodeUserLabel(simpul, 0, baru)
            legenda.model().refreshLayerLegend(simpul)
        else:                                                    # beberapa entri: ganti judul kelompoknya
            simpul.setCustomProperty("legend/title-label", baru)
legenda.setTitle("Legenda")
legenda.attemptMove(QgsLayoutPoint(205, 28, mm))
tata.addLayoutItem(legenda)

# 4. Batang skala: 2 segmen, tiap segmen 100 m
skala = QgsLayoutItemScaleBar(tata)
skala.setLinkedMap(peta)
skala.setStyle("Single Box")
skala.setUnits(Qgis.DistanceUnit.Meters)
skala.setUnitLabel("m")
skala.setNumberOfSegments(2)
skala.setNumberOfSegmentsLeft(0)
skala.setUnitsPerSegment(100)
skala.attemptMove(QgsLayoutPoint(12, 174, mm))
tata.addLayoutItem(skala)

# 5. Panah utara (gambar SVG bawaan QGIS)
panah = QgsLayoutItemPicture(tata)
for folder in QgsApplication.svgPaths():
    berkas = os.path.join(folder, "arrows", "NorthArrow_02.svg")
    if os.path.exists(berkas):
        panah.setPicturePath(berkas)
        break
panah.attemptMove(QgsLayoutPoint(270, 28, mm))
panah.attemptResize(QgsLayoutSize(15, 15, mm))
panah.setLinkedMap(peta)                          # panah ikut arah utara peta
tata.addLayoutItem(panah)

# 6. Teks skala angka, CRS, sumber, dan penulis
catatan = QgsLayoutItemLabel(tata)
catatan.setText("Skala 1:3.000 (A4). Sistem koordinat: WGS 84 / UTM zona 49S (EPSG:32749).\n"
                "Sumber: data sintetis latihan, bukan kondisi lapangan. Penyusun: Analis Muda, KPH Contoh.")
fmt2 = QgsTextFormat()
fmt2.setFont(QFont("Arial"))
fmt2.setSize(8)
catatan.setTextFormat(fmt2)
catatan.attemptMove(QgsLayoutPoint(10, 192, mm))
catatan.attemptResize(QgsLayoutSize(277, 12, mm))
```

```
tata.addLayoutItem(catatan)

# 7. Ekspor
pengekspor = QgsLayoutExporter(tata)
hasil_pdf = pengekspor.exportToPdf(os.path.join(FOLDER, "Peta_KPH.pdf"), QgsLayoutExporter.PdfExportSettings())
pengaturan = QgsLayoutExporter.ImageExportSettings()
pengaturan.dpi = 150
hasil_png = pengekspor.exportToImage(os.path.join(FOLDER, "Peta_KPH.png"), pengaturan)
print("PDF:", hasil_pdf, " PNG:", hasil_png)
print("Skala peta: 1:%d" % round(peta.scale()), " Tinggi peta (mm):", round(peta.rect().height()))
```

Penjelasan baris demi baris. Enam item memakai pola yang sama: buat item, atur letak dan ukuran, isi, lalu `addLayoutItem`. Karena itu tabel ini dikelompokkan per item.

Baris	Tulisan	Artinya
2-8	`from ... import (...), import os`	Panggil banyak perkakas layout. Lima di antaranya tidak dipakai di skrip ini: `QgsLayoutItemShape`, `QgsLayerTree`, `QgsLegendStyle`, `Qt`, dan `QColor`. Impor yang tersisa tidak berpengaruh. Perkakas `QgsMapLayerLegendUtils` dipakai untuk mengganti tulisan entri legenda.
10-12	`FOLDER`, `proyek`, `mm`	Folder kerja, proyek yang terbuka, dan satuan milimeter yang disimpan di kotak `mm` agar tidak ditulis panjang.
14-16	`lama = ...layoutByName("Peta_KPH_A4"), if lama is not None:, removeLayout(lama)`	Cari layout bernama `Peta_KPH_A4` di proyek. Bila ada (`is not None`), buang dulu. Dengan begitu skrip aman dijalankan ulang, karena dua layout tidak boleh sama namanya.
17	`tata = QgsPrintLayout(proyek)`	Buat layout baru di proyek ini.
18	`tata.initializeDefaults()`	Isi pengaturan awal: satu halaman A4 lanskap, 297 kali 210 mm.
19-20	`tata.setName("Peta_KPH_A4"), ...addLayout(tata)`	Beri nama layout, lalu daftarkan ke manajer layout proyek. Nama ini dipakai Skrip 17.6.
23-24	`peta = QgsLayoutItemMap(tata), peta.setCrs(proyek.crs())`	Buat item peta. Pakai sistem koordinat yang sama dengan proyek.
25-26	`attemptMove(QgsLayoutPoint(10, 28, mm)), attemptResize(QgsLayoutSize(190, 140, mm))`	Letakkan pojok kiri atas peta pada 10 mm dari kiri dan 28 mm dari atas. Ukurannya 190 kali 140 mm. Kata `attempt` berarti "coba": QGIS bisa menolak bila ada penyebab lain.
27	`peta.zoomToExtent(QgsRectangle(...))`	Arahkan isi peta ke satu persegi panjang: kiri, bawah, kanan, atas, dalam koordinat UTM.
28	`peta.setScale(3000)`	Atur skala menjadi 1:3000. Komentar mengingatkan: skala diatur terakhir.
29-30	`setFrameEnabled(True), tata.addLayoutItem(peta)`	Beri bingkai di sekeliling peta, lalu tambahkan item ke halaman.
33-38	`judul = QgsLayoutItemLabel(tata) ... setTextFormat(fmt)`	Buat item label. Isi tulisannya, atur huruf Arial 18 tebal, lalu pasang format. `QFont.Weight.Bold` berarti tebal.

Baris	Tulisan	Artinya
39-41	`attemptMove`, `attemptResize`, `addLayoutItem`	Letakkan judul di pojok 10 mm dari kiri dan 8 mm dari atas, selebar 277 mm. Lalu tambahkan.
44-46	`legenda = QgsLayoutItemLegend(tata)`, `setLinkedMap`, `setSyncMode(...Manual)`	Buat legenda, hubungkan ke item peta, lalu pilih mode manual. Mode manual berarti isi legenda tidak ikut berubah otomatis, jadi nama entrinya boleh diganti.
47	`akar = legenda.model().rootGroup()`	Ambil isi legenda, yaitu daftar entri layer.
48-49	`nama_baru = {...}`	Kamus: nama layer lama dan nama ramah yang diinginkan, misalnya `Petak_KPH` menjadi `Volume kayu (m3/ha)`.
50-57	`for simpul in list(akar.children()):` ...	Ulangi untuk tiap entri legenda. Bila nama layer ada di kamus, ambil nama barunya (baris 52). Layer yang hanya punya satu entri (baris 53) diganti tulisan entrinya dengan `setLegendNodeUserLabel`, dan nama layer di proyek tetap. Layer berentri banyak, yaitu `Petak_KPH`, diganti judul kelompoknya lewat properti `legend/title-label` (baris 56-57). Fungsi `list(...)` membuat salinan daftar agar aman dibaca sambil diubah.
58-60	`setTitle`, `attemptMove`, `addLayoutItem`	Beri judul legenda, letakkan pada 205 mm dari kiri dan 28 mm dari atas, lalu tambahkan.
63-70	`skala = QgsLayoutItemScaleBar(tata)` ... `setUnitsPerSegment(100)`	Buat batang skala: gaya `Single Box`, satuan meter, 2 segmen, tiap segmen 100 meter, dan tanpa segmen di kiri nol. Total panjang 200 meter.
71-72	`attemptMove`, `addLayoutItem`	Letakkan di bawah peta, lalu tambahkan.
75	`panah = QgsLayoutItemPicture(tata)`	Buat item gambar untuk panah utara.
76-80	`for folder in QgsApplication.svgPaths():` ... `break`	Cari berkas `NorthArrow_02.svg` di semua folder gambar bawaan QGIS. Begitu ditemukan, pasang sebagai gambar, lalu `break` menghentikan pencarian.
81-84	`attemptMove`, `attemptResize`, `setLinkedMap(peta)`, `addLayoutItem`	Letakkan panah di kanan atas, ukuran 15 kali 15 mm. Hubungkan ke peta supaya panah ikut arah utara peta.
87-93	`catatan = QgsLayoutItemLabel(tata)` ... `setTextFormat(fmt2)`	Buat label kedua berisi skala, sistem koordinat, sumber, dan penyusun. Dua potongan teks disambung otomatis oleh tanda kurung, dan `\n` memecah baris. Huruf Arial ukuran 8.
94-96	`attemptMove`, `attemptResize`, `addLayoutItem`	Letakkan di dasar halaman, selebar 277 mm.
99	`pengekspor = QgsLayoutExporter(tata)`	Buat pengekspor untuk layout ini.

Baris	Tulisan	Artinya
100	`hasil_pdf = pengekspor.exportToPdf(...)`	Ekspor ke PDF. Hasilnya angka: 0 berarti berhasil.
101-103	`pengaturan = ...ImageExportSettings(),pengaturan.dpi = 150, exportToImage(...)`	Ekspor ke PNG pada 150 dpi. Dibahas lebih lanjut di Skrip 17.6.
104-105	`print(...)`	Cetak hasil ekspor, lalu skala dan tinggi peta. `round(peta.scale())` membulatkan skala; `%d` mencetaknya sebagai bilangan bulat.

Hasil. Layar menampilkan (berdasarkan uji pada QGIS 4.0.2, urutan 17.1, 17.2, 17.4, 17.5):

```
HASIL
ERROR 6: The PNG driver does not support update access to existing datasets.
PDF: 0  PNG: 0
Skala peta: 1:3000  Tinggi peta (mm): 140
```

Baris `ERROR 6` berasal dari pustaka gambar GDAL dan tidak berarti ekspor gagal. Buktinya ada di baris di bawahnya: `PDF: 0  PNG: 0` berarti kedua ekspor berhasil, dan kedua berkas ada di folder kerja. Pesan itu muncul pada setiap ekspor PNG di uji ini.

Lembar yang dihasilkan memuat judul "PETA KPH CONTOH: VOLUME KAYU PER PETAK", peta petak berwarna dengan label, legenda, batang skala 0 sampai 200 meter, panah utara, dan catatan sumber. Entri legenda tertulis `Volume kayu (m3/ha)` dan `Petak belum diukur`. Nama layer di proyek tidak berubah: tetap `Belum diukur`, `Hotspot_Nasional`, `Jalan`, `Petak_KPH`, dan `Sungai`.

Bagian yang boleh diubah.

- **Baris 25-26 dan seterusnya, angka letak.** Semua angka di `QgsLayoutPoint(x, y, mm)` dan `QgsLayoutSize(lebar, tinggi, mm)` dalam milimeter. Ubah untuk menggeser atau memperbesar item.
- **Baris 27, ekstensi.** Ganti keempat koordinat dengan area yang ingin ditampilkan, dalam satuan peta.
- **Baris 28, skala.** Ganti 3000, misalnya 4000 untuk area lebih luas.
- **Baris 34, judul.** Ganti teks judul.
- **Baris 48-49, nama entri legenda.** Ganti isi kamus. Kunci kiri harus persis sama dengan nama layer.
- **Baris 67-70, batang skala.** Ganti jumlah segmen dan panjang tiap segmen.
- **Baris 88-89, catatan.** Ganti teks catatan.
- Baris 102, `dpi`. Ganti 150 untuk resolusi PNG lain.

Jebakan.

- **Legenda manual hanya mengganti tulisan, bukan nama layer.** Baris 50-57 mengubah tulisan di legenda, sedangkan nama layer di panel Layers tetap `Petak_KPH`,

`Hotspot_Nasional`, dan seterusnya. Karena itu skrip lain yang mencari layer menurut nama tetap bisa jalan. Bila Anda mengganti baris 54 atau 57 dengan `simpul.setName(...)`, nama layer di proyek ikut berubah. Pada uji, menjalankan Skrip 17.2 sesudah itu menghasilkan `IndexError: list index out of range`. Kunci kiri di kamus baris 48-49 harus memakai nama layer yang sebenarnya.

- **Dijalankan ulang.** Layout bernama sama tidak boleh ada dua. Baris 14-16 mencari layout `Peta_KPH_A4` yang lama dan membuangnya sebelum layout baru dibuat, jadi skrip aman diulang. Tanpa tiga baris itu, QGIS menolak layout kedua dan skrip gagal dengan `RuntimeError: wrapped C/C++ object of type QgsPrintLayout has been deleted`. Ingat bahwa layout lama ikut terbuang. Bila Anda sudah mengubah layout itu dengan tangan, ubahan itu hilang. Untuk menyimpan keduanya, ganti nama di baris 19.
- **Urutan mengatur peta.** Bila Anda menukar baris 27 dan 28 (skala diatur dulu, lalu `zoomToExtent`), skala berubah mengikuti area. Uji saya: skala menjadi `1:2857`, bukan `1:3000`. Atur ukuran dulu, lalu `zoomToExtent`, lalu `setScale`.
- **Angka tetap.** Ekstensi di baris 27 hanya cocok dengan data latihan. Tulisan `Skala 1:3.000` di baris 88 diketik manual. Bila Anda mengganti skala di baris 28, teks catatan tidak ikut berubah. Samakan dua-duanya.
- **Panah utara tidak ketemu.** Bila nama berkas SVG di baris 77 salah, skrip tidak menimbulkan galat. Uji: `panah.picturePath()` kosong (`''`), dan panah tidak tampil. Cek bahwa panah muncul di hasil.
- **Layer raster dari Skrip 17.3 ikut.** Bila Skrip 17.3 sudah dijalankan di proyek yang sama, layer bayangan bukit menutup petak. Hasil uji: peta abu-abu. Legenda juga memuat entri `DTM_Bayangan` dan `DTM_Drone` dengan nama aslinya.
- Pesan `ERROR 6` tidak berbahaya. Lihat bagian Hasil.

Latihan variasi. Ubah peta menjadi skala 1:4000, dan perbarui tulisan di catatan agar tetap benar.

Contoh jawaban. Ganti dua baris:

```python
peta.setScale(4000)                                # skala diatur terakhir
```

```python
catatan.setText("Skala 1:4.000 (A4). Sistem koordinat: WGS 84 / UTM zona 49S (EPSG:32749).\n"
```

Layar menampilkan `Skala peta: 1:4000  Tinggi peta (mm): 140`.

> **Pola yang bisa dipakai ulang:** Kerangka layout: **buat layout, lalu tiap item: buat, letakkan, isi, tambahkan**. Untuk peta lain, ganti empat hal: **nama layout** (baris 19), **ekstensi dan skala** (baris 27-28), **teks judul dan catatan** (baris 34 dan 88-89), dan **kamus legenda** (baris 48-49).

Skrip 17.6: Ekspor ke PDF dan PNG tiga resolusi

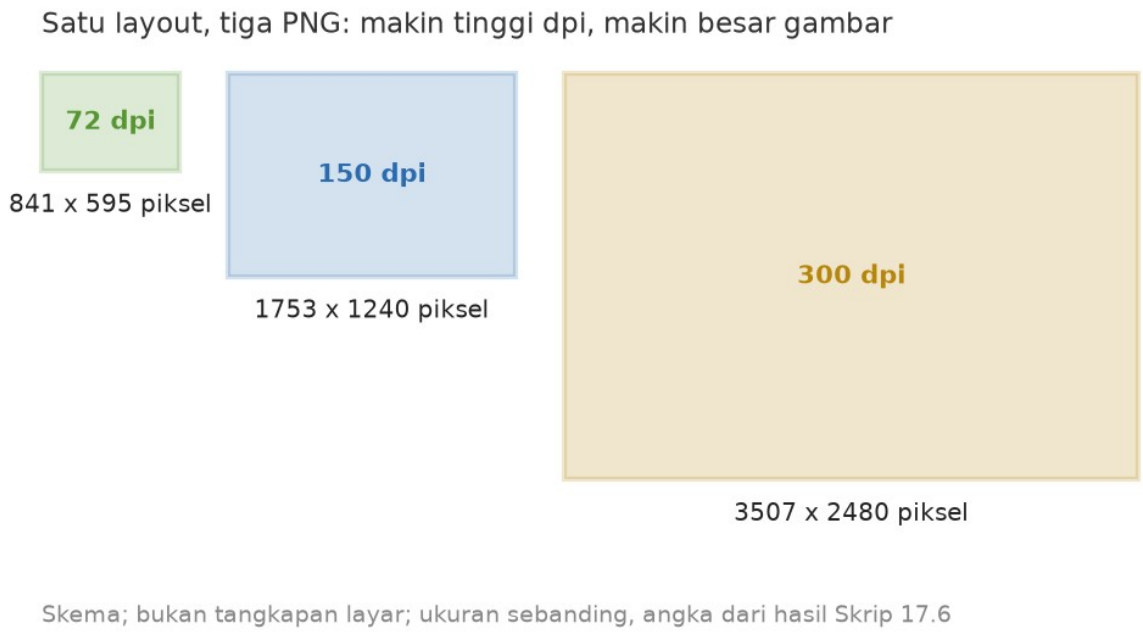
Berkas `skrip_4_5_ekspor.py`. Di dalam berkasnya tertulis "Skrip 4.5". Skrip ini pendek, dan sebagian besar logikanya sudah Anda kenal dari baris 99-105 Skrip 17.5.

Tujuan. Mengekspor layout `Peta_KPH_A4` ke satu PDF dan tiga PNG dengan resolusi berbeda, lalu mencatat ukuran tiap berkas.

Bahan. Layout `Peta_KPH_A4` dari Skrip 17.5 sudah ada di proyek. Hasilnya empat berkas di `FOLDER`: `Peta_KPH_A4.pdf`, `Peta_KPH_72.png`, `Peta_KPH_150.png`, dan `Peta_KPH_300.png`.

dpi (*dots per inch*) adalah banyaknya titik gambar tiap inci kertas. Bayangkan sulaman: makin rapat jahitannya, makin halus gambarnya, tetapi makin besar bahan yang dipakai. **PDF** menyimpan huruf dan garis sebagai rumus (vektor), jadi tetap tajam di ukuran berapa pun. **PNG** menyimpan kotak-kotak piksel, jadi ketajamannya ditentukan dpi.

Ilustrasi 17.10: Ukuran PNG pada tiga resolusi



Ilustrasi 17.11: Alur skrip 17.6



Skrip utuh:

```python
# Skrip 4.5: mengekspor layout Peta_KPH_A4 ke PDF dan PNG pada beberapa resolusi, lalu mencatat ukuran berkas
import os
from qgis.core import QgsProject, QgsLayoutExporter

FOLDER = r"D:/KPH_Contoh"   # ganti dengan folder kerja Anda

tata = QgsProject.instance().layoutManager().layoutByName("Peta_KPH_A4")
assert tata is not None, "Layout Peta_KPH_A4 belum ada; jalankan skrip 4.4 lebih dulu"
pengekspor = QgsLayoutExporter(tata)

# PDF: huruf dan garis tetap berupa vektor
pdf = os.path.join(FOLDER, "Peta_KPH_A4.pdf")
hasil = pengekspor.exportToPdf(pdf, QgsLayoutExporter.PdfExportSettings())
print("PDF  hasil=%s  ukuran=%d byte" % (hasil, os.path.getsize(pdf)))

# PNG: tiga resolusi
for dpi in (72, 150, 300):
    pengaturan = QgsLayoutExporter.ImageExportSettings()
    pengaturan.dpi = dpi
    png = os.path.join(FOLDER, "Peta_KPH_%d.png" % dpi)
    hasil = pengekspor.exportToImage(png, pengaturan)
    from qgis.PyQt.QtGui import QImage
    gambar = QImage(png)
    print("PNG %3d dpi  hasil=%s  %d x %d piksel  %d byte" % (dpi, hasil, gambar.width(), gambar.height(),
os.path.getsize(png)))
```

Penjelasan baris demi baris. Hanya hal yang baru dijelaskan.

Baris	Tulisan	Artinya
2-3	`import os`, `from qgis.core import ...`	Panggil `os` untuk urusan jalur dan ukuran berkas, serta perkakas ekspor QGIS.
7	`tata = QgsProject.instance().layoutManager().layoutByName("Peta_KPH_A4")`	Ambil layout menurut namanya dari manajer layout. Bila nama tidak ada, hasilnya `None`.
8	`assert tata is not None, "Layout ... belum ada; ..."`	Pemeriksa (Bab 10). Bila layout tidak ditemukan, skrip berhenti dengan `AssertionError` dan pesan yang menyuruh Anda menjalankan Skrip 17.5 lebih dulu. Pesan itu menyebut "skrip 4.4", yaitu nomor di berkasnya.
9	`pengekspor = QgsLayoutExporter(tata)`	Buat pengekspor.
12	`pdf = os.path.join(FOLDER, "Peta_KPH_A4.pdf")`	Gabungkan folder dan nama berkas menjadi satu jalur, dengan pemisah yang benar untuk komputer Anda.
13	`hasil = pengekspor.exportToPdf(pdf, ...PdfExportSettings())`	Ekspor ke PDF dengan pengaturan bawaan. Angka hasil disimpan di `hasil`.

Baris	Tulisan	Artinya
14	`print("PDF  hasil=%s ukuran=%d byte" % (hasil, os.path.getsize(pdf)))`	Cetak angka hasil dan ukuran berkas dalam byte. `%s` untuk teks atau angka apa pun, `%d` untuk bilangan bulat.
17	`for dpi in (72, 150, 300):`	Ulangi tiga kali. Di tiap putaran, `dpi` berisi satu nilai dari daftar.
18-19	`pengaturan = ...ImageExportSettings(),pengaturan.dpi = dpi`	Buat pengaturan gambar baru, lalu isi resolusinya.
20	`png = os.path.join(FOLDER, "Peta_KPH_%d.png" % dpi)`	Susun nama berkas yang memuat angka dpi, misalnya `Peta_KPH_150.png`.
21	`hasil = pengekspor.exportToImage(png, pengaturan)`	Ekspor ke PNG.
22	`from qgis.PyQt.QtGui import QImage`	Panggil perkakas pembaca gambar. Menaruhnya di dalam perulangan kurang lazim. Hasilnya tetap benar, karena Python hanya mengimpor sekali. Anda boleh memindahkannya ke atas.
23	`gambar = QImage(png)`	Buka PNG yang baru jadi, untuk membaca ukurannya.
24	`print("PNG %3d dpi hasil=%s  %d x %d piksel %d byte" % (...))`	Cetak dpi, angka hasil, lebar dan tinggi piksel, dan ukuran berkas. `%3d` memberi lebar tiga karakter agar sejajar.

Hasil. Layar menampilkan (berdasarkan uji pada QGIS 4.0.2, urutan 17.1, 17.2, 17.4, 17.5, 17.6):

```
HASIL
PDF  hasil=0  ukuran=207837 byte
ERROR 6: The PNG driver does not support update access to existing datasets.
PNG  72 dpi  hasil=0  841 x 595 piksel  68981 byte
ERROR 6: The PNG driver does not support update access to existing datasets.
PNG 150 dpi  hasil=0  1753 x 1240 piksel  181906 byte
ERROR 6: The PNG driver does not support update access to existing datasets.
PNG 300 dpi  hasil=0  3507 x 2480 piksel  404416 byte
```

Ukuran piksel selalu sama untuk dpi yang sama: 841 kali 595, 1753 kali 1240, dan 3507 kali 2480. Ukuran byte bergantung isi peta dan bisa sedikit berbeda di komputer Anda. Pesan `ERROR 6` sama dengan di Skrip 17.5 dan tidak berbahaya.

Bagian yang boleh diubah.

- **Baris 7, nama layout.** Harus persis sama dengan nama di Skrip 17.5. Ubah juga nama di pesan baris 8.
- **Baris 17, daftar dpi.** Ganti isi tanda kurung, misalnya `(96, 200)`.
- **Baris 12 dan 20, nama berkas.** Ganti nama sesuka Anda. Nama di baris 12 beda dengan `Peta_KPH.pdf` dari Skrip 17.5, jadi hasilnya dua PDF.

Jebakan.

- **Layout belum dibuat.** Bila Skrip 17.5 belum jalan, `layoutByName` mengembalikan `None`, dan baris 8 menghentikan skrip dengan `AssertionError: Layout Peta_KPH_A4 belum ada; ...`. Tanpa baris itu, skrip terus berjalan dan gagal belakangan dengan `FileNotFoundError` pada nama berkas PDF. Pesan itu menyesatkan, karena masalah sebenarnya ada di layout, bukan di berkas.
- **Berkas lama tidak membuktikan ekspor berhasil.** Folder bisa masih berisi PDF dan PNG dari proses sebelumnya, sehingga ukuran berkas yang tercetak bisa berasal dari berkas lama. Yang menentukan adalah angka `hasil`. Nilai 0 berarti berhasil, nilai selain 0 berarti gagal. Selalu baca angka itu.
- **Berkas lama ditimpa.** Berkas bernama sama diganti tanpa tanya. Pindahkan hasil penting sebelum menjalankan ulang.
- **Resolusi sangat tinggi.** Ukuran PNG naik cepat bersama dpi. Pada 300 dpi A4 sudah 3507 kali 2480 piksel. Tambah dpi hanya bila perlu.

Latihan variasi. Ekspor PNG hanya pada 96 dan 200 dpi, tanpa PDF.

Contoh jawaban. Hapus baris 11-14 dan ganti baris 17:

```python
for dpi in (96, 200):
```

Layar menampilkan `PNG  96 dpi  hasil=0  1122 x 793 piksel` dan `PNG 200 dpi  hasil=0  2338 x 1653 piksel`, masing-masing diikuti ukuran byte.

> **Pola yang bisa dipakai ulang:** Kerangka ekspor: **ambil layout menurut nama, buat pengekspor, ulangi untuk tiap pengaturan**. Untuk layout lain, ganti tiga hal: **nama layout** (baris 7), **daftar dpi** (baris 17), dan **nama berkas** (baris 12 dan 20).

Skrip 17.7: Proyek mini dari data mentah sampai PDF

Berkas `skrip_4_6_jembatan.py`. Di dalam berkasnya tertulis "Skrip 4.6". Skrip ini disebut "jembatan", karena menyambungkan kelas dasar dengan kelas menengah. Ia memadukan seleksi, alat Processing, simbologi, layout, dan ekspor dalam satu berkas.

> **Skrip ini mengosongkan proyek yang sedang terbuka:** Baris 15, `proyek.clear()`, membuang semua layer dan layout dari proyek QGIS yang terbuka saat skrip dijalankan. Tidak ada pertanyaan sebelumnya. Uji saya: proyek berisi tujuh layer dan layout `Peta_KPH_A4`. Setelah skrip ini, proyek hanya berisi lima layer baru dan layout `Peta_Sempadan`. Semua milik Anda hilang dari proyek (berkas data di folder aman). Selalu simpan proyek Anda lebih dulu, atau jalankan skrip ini di proyek kosong.

Tujuan. Memeriksa sistem koordinat titik panas, lalu mengubahnya ke UTM. Setelah itu, menyeleksi petak jati bervolume besar dan membuat sempadan sungai 25 meter. Petak dan titik panas dipotong dengan sempadan itu. Terakhir, skrip membuat peta dan menyimpan proyek.

Bahan. Folder kerja berisi `Hotspot_derajat.gpkg`, `Petak_KPH.gpkg`, dan `Sungai_Jalan.gpkg`. Skrip berdiri sendiri, tetapi mengosongkan proyek. Hasilnya `Peta_Sempadan.pdf` dan `KPH_Contoh_B4.qgz` di `FOLDER`.

Lima istilah baru, yang dijelaskan lebih lengkap di Bab 18:

- **Reproject**: mengubah sistem koordinat sebuah layer.
- **Buffer**: zona selebar jarak tertentu di sekeliling objek.
- **Clip**: memotong sebuah layer dengan bentuk layer lain.
- `processing.run`: perintah memanggil alat Processing dari skrip.
- `assert`: pemeriksa yang menghentikan skrip bila syaratnya tidak terpenuhi.

Sempadan sungai di skrip ini hanya angka latihan, 25 meter. Itu bukan ketentuan resmi.

Ilustrasi 17.12: Alur skrip 17.7



Skrip utuh:

```python
# Skrip 4.6 (Bab Jembatan): proyek mini terpadu B1-B4 dan pembuka geoprocessing.
# Alur: periksa CRS, reproject titik panas, seleksi atribut, buffer sungai, clip, simbologi, layout, ekspor.
import os
import processing
from qgis.core import (QgsProject, QgsVectorLayer, QgsCoordinateReferenceSystem, QgsFillSymbol,
                       QgsSingleSymbolRenderer, QgsMarkerSymbol, QgsLineSymbol, QgsPrintLayout,
                       QgsLayoutItemMap, QgsLayoutItemLabel, QgsLayoutItemLegend, QgsLayoutItemScaleBar,
                       QgsLayoutPoint, QgsLayoutSize, QgsLayoutExporter, QgsRectangle, QgsTextFormat, Qgis)
```

```
from qgis.PyQt.QtGui import QFont

FOLDER = r"D:/KPH_Contoh"   # ganti dengan folder kerja Anda
JARAK_SEMPADAN = 25         # meter; angka latihan, bukan ketentuan resmi

proyek = QgsProject.instance()
proyek.clear()
proyek.setCrs(QgsCoordinateReferenceSystem("EPSG:32749"))

def muat(berkas, lapisan, nama):
    layer = QgsVectorLayer("%s/%s|layername=%s" % (FOLDER, berkas, lapisan), nama, "ogr")
    assert layer.isValid(), "Layer tidak terbaca: " + nama
    return layer

# 1. Periksa CRS: titik panas kiriman kantor lain masih berderajat (B1)
mentah = muat("Hotspot_derajat.gpkg", "Hotspot_derajat", "Hotspot_mentah")
print("CRS titik panas mentah:", mentah.crs().authid(), "| satuan:", mentah.crs().mapUnits())

# 2. Reproject ke UTM 49S (alat Processing pertama)
hasil = processing.run("native:reprojectlayer", {
    "INPUT": mentah, "TARGET_CRS": QgsCoordinateReferenceSystem("EPSG:32749"), "OUTPUT": "memory:Titik_Panas"})
titik = hasil["OUTPUT"]
print("Titik panas setelah reproject:", titik.featureCount(), "titik,", titik.crs().authid())

# 3. Data utama
petak = muat("Petak_KPH.gpkg", "Petak_KPH", "Petak KPH")
sungai = muat("Sungai_Jalan.gpkg", "Sungai", "Sungai")

# 4. Seleksi atribut (B2): petak jati bervolume tinggi
petak.selectByExpression('"Jenis" = \'Jati\' AND "Volume_m3ha" > 150')
print("Jati dengan volume > 150 m3/ha:", sorted(f["ID_Petak"] for f in petak.selectedFeatures()))
petak.removeSelection()

# 5. Buffer sungai (alat Processing kedua): sempadan contoh 25 m, digabung jadi satu bidang
sempadan = processing.run("native:buffer", {
    "INPUT": sungai, "DISTANCE": JARAK_SEMPADAN, "SEGMENTS": 16, "DISSOLVE": True,
    "OUTPUT": "memory:Sempadan"})["OUTPUT"]
luas_sempadan = sum(f.geometry().area() for f in sempadan.getFeatures())
print("Luas buffer sungai: %.0f m2 (%.2f ha)" % (luas_sempadan, luas_sempadan / 10000))

# 6. Clip petak dengan buffer: bagian petak yang masuk sempadan
petak_sempadan = processing.run("native:clip", {
    "INPUT": petak, "OVERLAY": sempadan, "OUTPUT": "memory:Petak_di_sempadan"})["OUTPUT"]
total = 0
for f in sorted(petak_sempadan.getFeatures(), key=lambda x: x["ID_Petak"]):
    luas = f.geometry().area()
    total += luas
    print("   %s  %-6s %7.0f m2" % (f["ID_Petak"], f["Jenis"], luas))
print("Total petak dalam sempadan: %.0f m2 (%.2f ha), %d petak" % (total, total / 10000,
petak_sempadan.featureCount()))

# 7. Clip titik panas dengan buffer: titik panas yang berada di dalam sempadan
titik_dalam = processing.run("native:clip", {
    "INPUT": titik, "OVERLAY": sempadan, "OUTPUT": "memory:Titik_di_sempadan"})["OUTPUT"]
print("Titik panas di dalam sempadan:", titik_dalam.featureCount(), "dari", titik.featureCount())

# 8. Simbologi (B4) dan penyusunan layer; urutan: yang ditambah belakangan berada di atas
petak.setRenderer(QgsSingleSymbolRenderer(QgsFillSymbol.createSimple(
    {"color": "#E8F0D8", "outline_color": "#404040", "outline_width": "0.3"})))
sempadan.setRenderer(QgsSingleSymbolRenderer(QgsFillSymbol.createSimple(
    {"color": "#8FB8E0", "outline_color": "#2B6CB0", "outline_width": "0.2", "style": "solid"})))
petak_sempadan.setRenderer(QgsSingleSymbolRenderer(QgsFillSymbol.createSimple(
    {"color": "#F2A65A", "outline_color": "#8A4B00", "outline_width": "0.3"})))
sungai.setRenderer(QgsSingleSymbolRenderer(QgsLineSymbol.createSimple({"line_color": "#2B6CB0", "line_width":
"0.8"})))
titik.setRenderer(QgsSingleSymbolRenderer(QgsMarkerSymbol.createSimple(
    {"name": "circle", "color": "#E03C31", "outline_color": "#FFFFFF", "size": "2.6"})))
for layer in (petak, sempadan, petak_sempadan, sungai, titik):
```

```
    proyek.addMapLayer(layer)
petak_sempadan.setName("Petak di dalam sempadan")
sempadan.setName("Sempadan sungai %g m" % JARAK_SEMPADAN)
titik.setName("Titik panas")

# 9. Layout ringkas dan ekspor
mm = Qgis.LayoutUnit.Millimeters
tata = QgsPrintLayout(proyek)
tata.initializeDefaults()
tata.setName("Peta_Sempadan")
proyek.layoutManager().addLayout(tata)
peta = QgsLayoutItemMap(tata)
peta.setCrs(proyek.crs())
peta.attemptMove(QgsLayoutPoint(10, 28, mm))
peta.attemptResize(QgsLayoutSize(190, 140, mm))
peta.zoomToExtent(QgsRectangle(312000, 9996000, 312400, 9996400))
peta.setScale(3000)
peta.setFrameEnabled(True)
tata.addLayoutItem(peta)
judul = QgsLayoutItemLabel(tata)
judul.setText("SEMPADAN SUNGAI DI KPH CONTOH (CONTOH %g M)" % JARAK_SEMPADAN)
fmt = QgsTextFormat()
fmt.setFont(QFont("Arial", 16, QFont.Weight.Bold))
fmt.setSize(16)
judul.setTextFormat(fmt)
judul.attemptMove(QgsLayoutPoint(10, 10, mm))
judul.attemptResize(QgsLayoutSize(277, 12, mm))
tata.addLayoutItem(judul)
legenda = QgsLayoutItemLegend(tata)
legenda.setLinkedMap(peta)
legenda.setTitle("Legenda")
legenda.attemptMove(QgsLayoutPoint(205, 28, mm))
tata.addLayoutItem(legenda)
skala = QgsLayoutItemScaleBar(tata)
skala.setLinkedMap(peta)
skala.setStyle("Single Box")
skala.setUnits(Qgis.DistanceUnit.Meters)
skala.setUnitLabel("m")
skala.setNumberOfSegments(2)
skala.setNumberOfSegmentsLeft(0)
skala.setUnitsPerSegment(100)
skala.attemptMove(QgsLayoutPoint(12, 174, mm))
tata.addLayoutItem(skala)
hasil_pdf = QgsLayoutExporter(tata).exportToPdf(os.path.join(FOLDER, "Peta_Sempadan.pdf"),
QgsLayoutExporter.PdfExportSettings())
print("Ekspor PDF:", hasil_pdf, "(0 = berhasil)")
proyek.write(os.path.join(FOLDER, "KPH_Contoh_B4.qgz"))
print("Proyek disimpan:", os.path.exists(os.path.join(FOLDER, "KPH_Contoh_B4.qgz")))
```

Penjelasan baris demi baris. Bagian layout (baris 80-117) mengulang pola Skrip 17.5 dan dirangkum saja.

Baris	Tulisan	Artinya
1-9	komentar dan `import`	Dua komentar pembuka, lalu perkakas: `os`, `processing` (pemanggil alat Processing), dan banyak perkakas QGIS.
11-12	`FOLDER`, `JARAK_SEMPADAN = 25`	Folder kerja dan jarak sempadan dalam meter. Nama kotak dengan huruf besar menandakan nilai tetap, yang diatur di awal.
14-16	`proyek.clear()` dan `setCrs`	**Peringatan:** `clear()` mengosongkan seluruh proyek. Lalu atur CRS proyek ke EPSG:32749.

Baris	Tulisan	Artinya
18-21	`def muat(...)` dengan `assert`	Fungsi pemuat. Beda dengan Skrip 17.1, ia hanya membuat layer dan tidak menambahkannya ke proyek. Baris 20: bila `layer.isValid()` salah, skrip berhenti dengan pesan `Layer tidak terbaca:` diikuti nama layer.
24-25	`mentah = muat(...), print(... mapUnits())`	Muat titik panas berderajat, lalu cetak kode CRS dan satuan petanya.
28-31	`processing.run("native:reprojectlayer", {...})`	Panggil alat reproject. Isian pertama nama alat, isian kedua kamus pengaturan: layer masukan, CRS tujuan, dan `OUTPUT` berupa `"memory:Titik_Panas"`, yaitu layer sementara di memori. Hasilnya kamus, dan layer baru diambil dengan `["OUTPUT"]`.
34-35	`petak = muat(...), sungai = muat(...)`	Muat petak dan sungai.
38	`petak.selectByExpression('"Jenis" = \'Jati\' AND "Volume_m3ha" > 150')`	Pilih petak yang jenisnya Jati dan volumenya lebih dari 150. Seleksi ini hanya menandai petak, tanpa mengubah data.
39-40	`print(... sorted(... for f in petak.selectedFeatures())), removeSelection()`	Cetak kode petak terpilih, urut abjad. Lalu hapus seleksi.
43-45	`sempadan = processing.run("native:buffer", {...})["OUTPUT"]`	Buat buffer sungai sejauh `JARAK_SEMPADAN`. `SEGMENTS: 16` mengatur kehalusan sudut, dan `DISSOLVE: True` menggabungkan semua buffer menjadi satu bidang.
46-47	`luas_sempadan = sum(... for f in ...), print(...)`	Jumlahkan luas bidang buffer, lalu cetak dalam meter persegi dan hektare. Satu hektare adalah 10.000 meter persegi.
50-51	`petak_sempadan = processing.run("native:clip", {...})["OUTPUT"]`	Potong petak dengan bidang buffer. Yang tersisa hanya bagian petak di dalam sempadan.
52-57	`total = 0` dan perulangan `for`	Siapkan penjumlah `total` bernilai 0. Ulangi untuk tiap potongan, urut kode petak. Tambahkan luasnya ke `total` (`total += luas`), lalu cetak satu baris. Terakhir cetak jumlahnya.
53	`key=lambda x: x["ID_Petak"]`	Aturan pengurutan: urutkan menurut kolom `ID_Petak`. `lambda` adalah fungsi mini satu baris (Bab 7).
60-62	`titik_dalam = processing.run("native:clip", ...)`	Potong titik panas dengan buffer yang sama, lalu cetak jumlahnya.
65-73	`petak.setRenderer(...)` dan empat baris serupa	Beri simbol tunggal pada lima layer. Pola sama dengan Skrip 17.1.
74-75	`for layer in (...): proyek.addMapLayer(layer)`	Tambahkan lima layer ke proyek. Urutan penambahan menentukan tumpukan: yang terakhir berada di atas.
76-78	`petak_sempadan.setName(...), sempadan.setName(...), titik.setName(...)`	Beri nama ramah pada tiga layer sementara. Nama sempadan memakai `%g`, yaitu format angka yang mencetak 25 sebagai `25`. Nilainya diambil dari `JARAK_SEMPADAN`, jadi nama ikut berubah bila jarak diganti.

Baris	Tulisan	Artinya
80-117	`mm = ...` sampai `tata.addLayoutItem(skala)`	Layout ringkas: peta, judul, legenda, dan batang skala. Pola sama dengan Skrip 17.5. Perbedaannya, legenda tidak dibuat manual (baris 103-107), dan tidak ada panah utara atau catatan. Judul peta (baris 95) juga memakai `%g` dan `JARAK_SEMPADAN`.
118-119	`hasil_pdf = QgsLayoutExporter(tata).exportToPdf(...)`	Ekspor PDF, lalu cetak angka hasil.
120	`proyek.write(os.path.join(FOLDER, "KPH_Contoh_B4.qgz"))`	Simpan proyek ke berkas `.qgz`.
121	`print("Proyek disimpan:", os.path.exists(...))`	Periksa bahwa berkas ada.

Hasil. Layar menampilkan (berdasarkan uji pada QGIS 4.0.2):

```
HASIL
CRS titik panas mentah: EPSG:4326 | satuan: 6
Titik panas setelah reproject: 12 titik, EPSG:32749
Jati dengan volume > 150 m3/ha: ['G05', 'G06', 'G07']
Luas buffer sungai: 29435 m2 (2.94 ha)
   A04  Jati       739 m2
   B01  Mahoni    1300 m2
   B02  Agroforestri    5736 m2
   B03  Akasia    5663 m2
   B04  Sengon     203 m2
   G01  Mahoni     134 m2
   G02  Sengon    5690 m2
   G03  Jati       558 m2
   G05  Jati      5612 m2
   G06  Jati      1438 m2
Total petak dalam sempadan: 27072 m2 (2.71 ha), 10 petak
Titik panas di dalam sempadan: 6 dari 12
Ekspor PDF: 0 (0 = berhasil)
Proyek disimpan: True
```

Cara membaca hasil. Angka `4326` adalah kode lintang-bujur (derajat).

Angka `satuan: 6` adalah kode satuan peta untuk derajat. Uji saya: `int(Qgis.DistanceUnit.Degrees)` bernilai 6. Setelah reproject, 12 titik panas tetap 12, tetapi sistem koordinatnya UTM. Sepuluh dari 16 petak tersentuh sempadan, dengan luas total 27.072 meter persegi.

Bagian yang boleh diubah.

- **Baris 11, folder.** Ganti jalur komputer Anda.
- **Baris 12, jarak sempadan.** Ganti 25 menjadi angka lain. Nama layer sempadan dan judul peta ikut berubah. Komentar dan teks cetak lain tidak ikut (lihat Jebakan).
- **Baris 38, ambang volume.** Ganti `150`, atau ganti `'Jati'` dengan jenis lain.
- **Baris 44, jumlah segmen.** Ganti `16`. Angka kecil membuat ujung buffer bersudut kasar.
- **Baris 65-73, warna simbol.** Ganti kode warna.

- **Baris 90-91, ekstensi dan skala peta.** Sama seperti Skrip 17.5.

Jebakan.

- `proyek.clear()` mengosongkan proyek. Lihat kotak peringatan di awal skrip ini. Hasil uji: tujuh layer dan layout `Peta_KPH_A4` hilang, diganti lima layer baru dan layout `Peta_Sempadan`.
- **Layer sementara tidak ikut tersimpan.** Baris 120 menyimpan proyek, tetapi tiga layer hasil Processing hanya ada di memori. Saya menyimpan proyek lalu membukanya lagi di sesi baru. Hasilnya: `Petak di dalam sempadan`, `Sempadan sungai 25 m`, dan `Titik panas` ikut muncul, tetapi ketiganya **kosong** (0 fitur). Hanya `Petak KPH` dan `Sungai` yang berisi. Untuk menyimpan hasil, tulis keluaran ke berkas, misalnya `"OUTPUT": FOLDER + "/Sempadan.gpkg"` pada baris 45.
- **Jalur salah menghentikan skrip dengan pesan jelas.** Baris 20 menghasilkan `AssertionError: Layer tidak terbaca: Hotspot_mentah`. Pesan di Skrip 17.1 menyebut nama berkas, sedangkan di sini nama layer. Periksa `FOLDER` dan nama berkas.
- **Nama layer dan judul mengikuti angka, teks lain tidak.** Ganti `JARAK_SEMPADAN` menjadi 50. Layer lalu bernama `Sempadan sungai 50 m` (baris 77) dan judul peta berbunyi `(CONTOH 50 M)` (baris 95), karena keduanya memakai `%g`. Komentar di baris 42 yang menyebut "25 m" tetap, jadi ubah sendiri bila perlu.
- **Teks cetak tetap.** Baris 39 mencetak `volume > 150` sebagai teks tetap. Bila Anda mengubah ambang di baris 38, teksnya tidak ikut.

Latihan variasi. Lebarkan sempadan menjadi 50 meter, lalu bandingkan luas hasilnya.

Contoh jawaban. Ubah baris 12:

```python
JARAK_SEMPADAN = 50         # meter; angka latihan, bukan ketentuan resmi
```

Layar menampilkan `Luas buffer sungai: 62790 m2 (6.28 ha)` dan `Total petak dalam sempadan: 51831 m2 (5.18 ha), 10 petak`. Jumlah titik panas dalam sempadan tetap `6 dari 12`. Layer sempadan di panel bernama `Sempadan sungai 50 m`, dan judul peta berbunyi `(CONTOH 50 M)`.

> **Pola yang bisa dipakai ulang:** Kerangka proyek mini: bersihkan, periksa data, ubah CRS, analisis dengan `processing.run`, beri gaya, susun layout, ekspor, simpan. Untuk wilayah lain, ganti empat hal: **folder dan nama berkas** (baris 11, 24, 34-35), **jarak sempadan** (baris 12), **ambang seleksi** (baris 38), dan **ekstensi peta** (baris 90). Bila tidak ingin proyek terbuka terhapus, buang baris 15 setelah memahami akibatnya.

Cek paham

1. Apa beda penyaji Categorized dan Graduated?
2. Mengapa Skrip 17.2 menyaring petak kosong sebelum menghitung kelas?
3. Mengapa Skrip 17.5 aman dijalankan dua kali, padahal dua layout tidak boleh sama namanya?
4. Apa yang terjadi bila `FOLDER` di Skrip 17.1 salah, dan baris mana yang menyebabkannya?

Jawaban.

1. Categorized memberi satu warna untuk tiap nilai teks, misalnya tiap jenis tanaman. Graduated memberi satu warna untuk tiap rentang angka, misalnya tiap kelas volume.
2. Quantile dan Natural Breaks membaca nilai kosong sebagai nol. Tanpa saringan, kelas pertama ikut tertarik ke nol, dan batas kelas menjadi tidak mencerminkan data.
3. Baris 14-16 Skrip 17.5 mencari layout lama bernama `Peta_KPH_A4` dan membuangnya sebelum layout baru dibuat. Jadi tidak pernah ada dua layout sama nama.
4. Skrip berhenti dengan `AssertionError: Layer tidak terbaca:` diikuti nama berkas. Penyebabnya baris 13 (`assert layer.isValid()`), yang memeriksa bahwa data berhasil dibaca.

Kesalahan umum

- **Menjalankan skrip tidak berurutan.** Skrip 17.2, 17.4, dan 17.5 mengandalkan layer dari skrip sebelumnya. Perbaikan: ikuti tabel urutan di awal bab.
- **Membiarkan jalur contoh.** `FOLDER` menunjuk folder di komputer penulis. Perbaikan: ganti jalur di tiap skrip. Bila muncul `AssertionError` berpesan "tidak terbaca", periksa jalur dan nama berkas.
- **Menjalankan Skrip 17.7 di proyek yang berisi pekerjaan.** Perbaikan: simpan proyek dulu, atau pakai proyek kosong.
- **Mengira hasil ekspor berhasil karena berkas ada.** Perbaikan: baca angka `hasil`. Nilai 0 berarti berhasil.

Ringkasan dan latihan

- Peta jadi dirakit bertahap: layer, simbologi, label, layout, ekspor.
- **Categorized** mewarnai menurut nilai teks, **Graduated** menurut rentang angka, dan tiga metode kelas memberi batas berbeda.
- Layout tersusun dari item yang diletakkan dengan milimeter. Atur ukuran dulu, lalu cakupan, lalu skala.
- Ekspor PDF tetap tajam. PNG bergantung dpi. Selalu baca angka `hasil`.
- Baris `assert` menghentikan skrip dengan pesan jelas bila data atau layout tidak ada.
- Dua hal yang perlu diingat: legenda manual hanya mengganti tulisan, bukan nama layer, dan `proyek.clear()` mengosongkan proyek.

Latihan. Buat salinan Skrip 17.5 dengan layout bernama `Peta_Volume_A4`, skala 1:3500, dan judul "PETA VOLUME KAYU". Ekspor PDF-nya saja. Bagian mana yang Anda ubah, dan bagian mana yang Anda hapus?

Tabel perbandingan: empat cara menata tampilan peta

Pekerjaan	Perkakas di skrip	Isian utama	Skrip di bab ini
Warna per jenis	`QgsCategorizedSymbolRenderer`	kolom teks dan kategori	17.1
Warna per kelas angka	`QgsGraduatedSymbolRenderer`	kolom angka dan metode	17.2
Warna raster	`QgsSingleBandPseudoColorRenderer`	skala warna dan kelas	17.3
Tulisan di peta	`QgsPalLayerSettings`	ekspresi dan format teks	17.4
Lembar peta	`QgsPrintLayout`	item, letak, ukuran (mm)	17.5 dan 17.7
Berkas siap edar	`QgsLayoutExporter`	nama layout dan dpi	17.6

BAB 18: Dari Peta ke Jawaban: Buffer, Overlay, Join, dan Model dengan Skrip

> **Berdasarkan Seri I1:** Skrip bab ini berasal dari Buku I1, *Analisis Vektor dan Geoprocessing* (Intermediate Class), berkas `1_1_cek_data.py`, `2_1_buffer_sempadan.py`, `3_1_overlay.py`, `4_1_dissolve_rekap.py`, `5_1_spatial_join.py`, `6_1_buat_model.py`, dan `7_1_laporan_terpadu.py`. Di buku itu, analisis vektor dibahas lewat menu dan klik. Bab ini menunjukkan sisi skripnya.

Studi kasus: "Berapa hektare yang benar-benar bisa dipanen?"

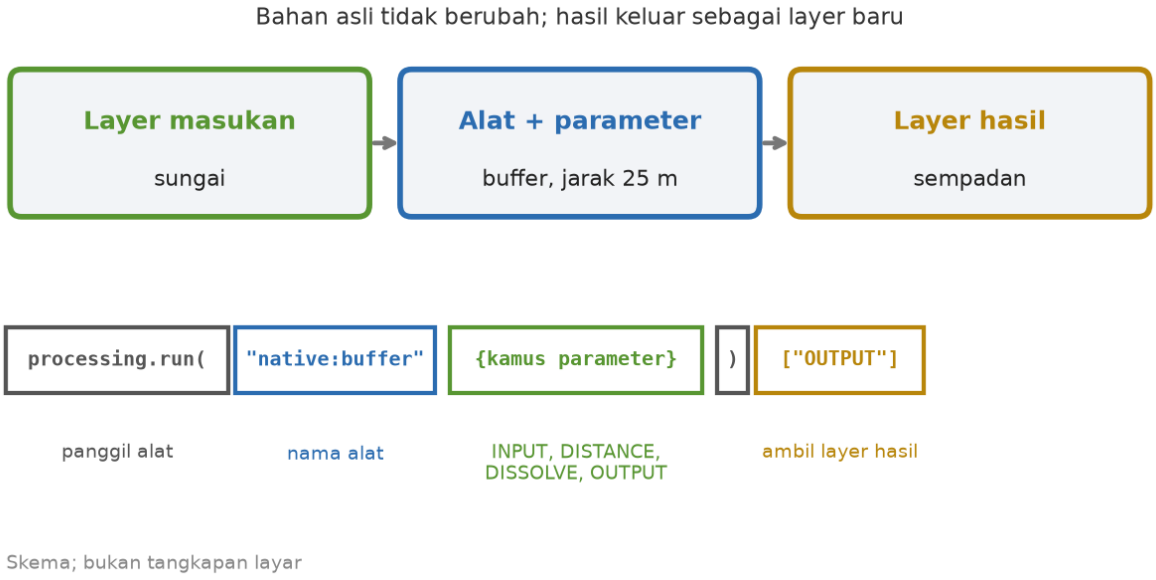
Kepala Seksi bertanya: "Dari 16 hektare petak kita, berapa hektare yang berada di luar jarak aman sungai dan permukiman? Rinci per KPH dan per jenis tanaman. Berapa titik panas yang jatuh di areal itu?" Jawabannya tidak terbaca di layar. Anda harus menghitungnya.

Alat penghitungnya sudah ada di QGIS. Skrip menyatukan alat-alat itu menjadi satu resep yang bisa diulang, misalnya bila Kepala Seksi minta jarak aman diganti 50 meter.

Konsep: geoprocessing dalam tiga kalimat

Geoprocessing adalah mengolah layer lewat sebuah alat untuk menghasilkan layer baru. Bayangkan mesin penggiling: bahan masuk, mesin bekerja menurut pengaturan, hasil giling keluar, dan bahan aslinya tidak berubah. Di skrip, satu perintah, `processing.run`, memanggil mesin itu.

Ilustrasi 18.1: Anatomi satu panggilan alat



Skema; bukan tangkapan layar

Lima istilah yang perlu Anda kenal sekarang:

- **Alat** (*algorithm*): satu mesin dengan tugas tertentu, misalnya `native:buffer`. Nama alat selalu ditulis dalam tanda kutip.

- **Parameter**: pengaturan sebuah alat, ditulis dalam kamus (Bab 5), misalnya jarak buffer.
- **Layer sementara**: hasil yang hanya ada di memori. Di skrip ditulis `"memory:"`. Layer ini hilang saat QGIS ditutup.
- **Geometri valid**: bentuk objek yang tidak rusak, misalnya poligon yang tidak menyilang dirinya sendiri.
- `processing.run`: perintah memanggil alat. Hasilnya berupa kamus. Layer hasil diambil dengan `["OUTPUT"]`.

Sebelum mulai: bahan, jalur, dan urutan

Bahan. Siapkan dua folder data.

- `paket-i1`: `Petak.gpkg` (16 petak, kolom `KODE_PETAK`, `JENIS`, `TAHUN_TANAM`), `Sungai.gpkg` (2 garis: Sungai Tengah berorde 1 dan Anak Sungai Barat berorde 2), `Permukiman.gpkg` (2 poligon), `Jalan.gpkg` (2 garis), dan `Perhutanan_Sosial.gpkg` (1 poligon).
- `paket-data`: `Batas_KPH.gpkg` (3 KPH: Alpha, Beta, Gamma) dan `Hotspot_Nasional.gpkg` (12 titik panas).

Semua layer memakai sistem koordinat EPSG:32749 (UTM zona 49 Selatan, satuan meter). Kode petak di seri ini berbentuk `P07`, tanpa tanda hubung.

Jalur. Skrip memakai tiga lokasi di komputer penulis: `PAKET = r"C:\latihan\paket-i1"`, `DATA = r"C:\latihan\paket-data"`, dan (di Skrip 18.6 dan 18.7) `HASIL = r"C:\latihan\hasil"`. Ganti dengan lokasi folder Anda.

Nama berkas disambung dengan garis miring terbalik, misalnya `PAKET + r"\Petak.gpkg"`. Itu benar di Windows. Di macOS dan Ubuntu pemisah folder adalah garis miring biasa, jadi Anda perlu mengganti `r"\Petak.gpkg"` menjadi `"/Petak.gpkg"` di semua baris pemuatan. Bagian macOS dan Ubuntu ini belum saya uji. [CEK]

Urutan. Skrip 18.1 sampai 18.6 berdiri sendiri, karena masing-masing memuat datanya sendiri. Skrip 18.7 membaca berkas model yang ditulis Skrip 18.6, jadi jalankan 18.6 lebih dulu.

Menjalankan. Buka Konsol Python QGIS atau Editor Skrip (Bab 3), tempel isi skrip, lalu jalankan.

Pengujian. Semua hasil di bab ini dijalankan di QGIS 4.0.2 tanpa jendela utama. Saya menjalankan salinan skrip dengan tiga lokasi di atas diganti ke lokasi salinan data. Kode yang ditampilkan di bab ini tetap isi berkas asli.

Skrip 18.1: Memeriksa data sebelum dianalisis

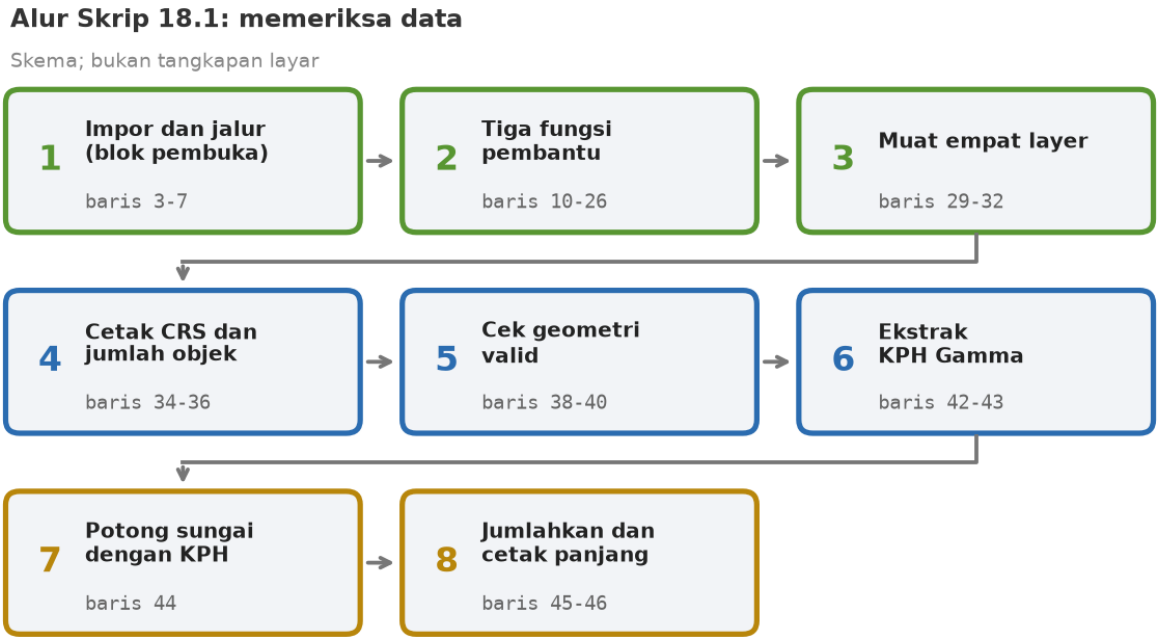
Berkas `1_1_cek_data.py`. Di dalam berkasnya tidak tertulis nomor skrip, hanya nama berkas. Skrip ini punya dua tugas: memeriksa data, lalu memperkenalkan "blok pembuka" yang muncul di enam skrip bab ini.

Tujuan. Memuat empat layer, memeriksa sistem koordinat dan jumlah objeknya, memeriksa kevalidan geometri petak, lalu mengukur panjang sungai yang masuk KPH Gamma.

Bahan. `paket-i1` (`Petak`, `Sungai`, `Permukiman`) dan `paket-data` (`Batas_KPH`). Skrip berdiri sendiri.

Sebelum menghitung apa pun, periksa dulu bahannya. Itu seperti memeriksa timbangan sebelum menimbang. Bila data rusak, semua angka sesudahnya menyesatkan.

Ilustrasi 18.2: Alur skrip 18.1



Skrip utuh:

```python
# -*- coding: utf-8 -*-
# Penulis: Badar Mubarok Yogaswara. Dijalankan di Python Console QGIS. Data sintetis, bukan kondisi lapangan nyata.
from qgis.core import QgsVectorLayer, QgsProject, QgsProperty
import processing

PAKET = r"C:\latihan\paket-i1"      # ubah sesuai letak folder di komputer Anda
DATA = r"C:\latihan\paket-data"


def muat(berkas, lapisan, nama):
    """Memuat satu layer GeoPackage dan menambahkannya ke proyek."""
    lyr = QgsVectorLayer("%s|layername=%s" % (berkas, lapisan), nama, "ogr")
    if not lyr.isValid():
        raise RuntimeError("Layer tidak terbaca: " + berkas)
    QgsProject.instance().addMapLayer(lyr)
    return lyr


def luas_ha(lyr):
    """Jumlah luas semua poligon dalam hektare (CRS harus UTM, satuan meter)."""
    return sum(f.geometry().area() for f in lyr.getFeatures()) / 10000.0


def buf(lyr, jarak):
    """Buffer dengan dissolve, hasil sementara."""
    return processing.run("native:buffer", {"INPUT": lyr, "DISTANCE": jarak, "DISSOLVE": True, "OUTPUT": "memory:"})["OUTPUT"]
```

```
petak = muat(PAKET + r"\Petak.gpkg", "Petak", "Petak")
sungai = muat(PAKET + r"\Sungai.gpkg", "Sungai", "Sungai")
kampung = muat(PAKET + r"\Permukiman.gpkg", "Permukiman", "Permukiman")
kph = muat(DATA + r"\Batas_KPH.gpkg", "Batas_KPH", "Batas_KPH")

# 1. Periksa CRS dan jumlah objek tiap layer
for lyr in (petak, sungai, kampung, kph):
    print(lyr.name(), lyr.crs().authid(), lyr.featureCount(), "objek")

# 2. Periksa kevalidan geometri (laporan, bukan perbaikan)
hasil = processing.run("native:checkvalidity", {"INPUT_LAYER": petak, "METHOD": 2, "VALID_OUTPUT": "memory:",
"INVALID_OUTPUT": "memory:", "ERROR_OUTPUT": "memory:"})
print("Petak valid:", hasil["VALID_COUNT"], "tidak valid:", hasil["INVALID_COUNT"])

# 3. Potong sungai dengan KPH Gamma saja (Clip), hasil sementara
gamma = processing.run("native:extractbyexpression", {"INPUT": kph, "EXPRESSION": "\"NAMA_KPH\" = 'KPH Gamma'",
"OUTPUT": "memory:"})["OUTPUT"]
potong = processing.run("native:clip", {"INPUT": sungai, "OVERLAY": gamma, "OUTPUT": "memory:"})["OUTPUT"]
panjang = sum(f.geometry().length() for f in potong.getFeatures())
print("Sungai di KPH Gamma: %d ruas, %.1f m" % (potong.featureCount(), panjang))
```

Blok pembuka: baris 1-28

Baris 1-28 sama persis di enam skrip: `1_1`, `2_1`, `3_1`, `4_1`, `5_1`, dan `7_1`. Penulis menyalinnya ke tiap berkas supaya setiap skrip bisa dijalankan sendiri. Penjelasannya hanya di sini. Di skrip lain, Anda cukup membaca "baris 1-28: blok pembuka".

Ilustrasi 18.3: Blok pembuka dan pemakainya



Enam skrip memakai blok yang sama persis (baris 1-28). Hanya muat dipakai oleh semuanya.
Skema; bukan tangkapan layar; ya atau tidak dihitung dari isi skrip (setelah baris 28)

Gambar di atas hasil membaca isi skrip. Fungsi `muat` dipakai semua. Fungsi `luas_ha` dan `buf` hanya dipakai sebagian, dan `QgsProperty` hanya di Skrip 18.2. Kelebihan itu tidak berbahaya, hanya mengisi baris.

Baris	Tulisan	Artinya
1	`# -*- coding: utf-8 -*-`	Komentar khusus. Ia memberi tahu Python bahwa berkas memakai huruf UTF-8, supaya huruf dan simbol aman.
2	`# Penulis: ...`	Komentar: nama penulis, tempat menjalankan, dan pengingat bahwa datanya sintetis.

Baris	Tulisan	Artinya
3	`from qgis.core import QgsVectorLayer, QgsProject, QgsProperty`	Panggil tiga perkakas QGIS: pembuka layer vektor, pengakses proyek, dan pengatur nilai berubah per objek.
4	`import processing`	Panggil perkakas yang menjalankan alat Processing.
6-7	`PAKET = r"C:\latihan\paket-i1", DATA = ...`	Dua kotak berisi lokasi folder data. Komentar di ujung baris 6 menyuruh Anda mengubahnya. Huruf `r` membuat garis miring terbalik dibaca apa adanya.
10	`def muat(berkas, lapisan, nama):`	Buat fungsi `muat` dengan tiga isian (Bab 7).
11	`"""Memuat satu layer ..."""`	*Docstring*: kalimat penjelasan fungsi, ditulis dalam tiga tanda kutip. Python tidak menjalankannya.
12	`lyr = QgsVectorLayer("%s...`	Susun alamat data dari berkas, tanda garis tegak, dan nama layer. Lalu buka sebagai layer bernama `nama` dengan pembaca `"ogr"`. Tanda `%s` diisi oleh dua nilai di dalam kurung setelah tanda `%`.
13-14	`if not lyr.isValid(): raise RuntimeError(...)`	Bila layer tidak berhasil dibuka, hentikan skrip dengan pesan `Layer tidak terbaca:` diikuti nama berkas. `raise` berarti mengangkat galat dengan sengaja (Bab 10).
15	`QgsProject.instance().addMapLayer(lyr)`	Tambahkan layer ke proyek yang terbuka.
16	`return lyr`	Kembalikan layer ke pemanggil fungsi.
19-21	`def luas_ha(lyr):`	Fungsi penjumlah luas. Baris 21 menjumlahkan `f.geometry().area()` dari semua fitur, lalu membaginya 10.000,0 supaya menjadi hektare. Luas dihitung dalam satuan peta, jadi layer harus berkoordinat meter.
24-26	`def buf(lyr, jarak):`	Fungsi buffer. Ia memanggil alat `native:buffer` dengan penggabungan hasil (`DISSOLVE: True`), hasilnya layer sementara, lalu mengembalikannya.

Isi khusus Skrip 18.1: baris 29-46

Baris	Tulisan	Artinya
29-32	`petak = muat(...)` dan tiga baris di bawahnya	Panggil `muat` empat kali. Tiap panggilan membuka satu layer, menambahkannya ke proyek, lalu menyimpannya di kotak. Dua isian pertama memberi berkas dan nama layer di dalamnya. Isian ketiga adalah nama di panel Layers.
35-36	`for lyr in (petak, sungai, kampung, kph): print(...)`	Ulangi untuk keempat layer. Cetak nama, kode sistem koordinat (`crs().authid()`), dan jumlah objek (`featureCount()`).
39	`hasil = processing.run("native:checkvalidity", {...})`	Panggil alat pemeriksa geometri pada `petak`. `METHOD: 2` memilih cara pemeriksaan GEOS. Tiga isian berakhiran `_OUTPUT` meminta layer sementara untuk objek valid, objek tidak valid, dan daftar galat.

Baris	Tulisan	Artinya
40	`print("Petak valid:", hasil["VALID_COUNT"], ...)`	Cetak jumlah petak valid dan tidak valid, yang dilaporkan alat di kamus hasil.
43	`gamma = processing.run("native:extractbyexpression", {...})["OUTPUT"]`	Ambil hanya KPH Gamma dari layer batas. Ekspresinya ditulis di dalam teks berkutip ganda, jadi kutip ganda untuk nama kolom ditulis `\"`.
44	`potong = processing.run("native:clip", {...})["OUTPUT"]`	Potong sungai dengan bentuk KPH Gamma. Hanya bagian sungai di dalam KPH itu yang tersisa.
45	`panjang = sum(f.geometry().length() for f in ...)`	Jumlahkan panjang semua ruas sungai hasil potongan, dalam meter.
46	`print("Sungai di KPH Gamma: %d ruas, %.1f m" % (...))`	Cetak jumlah ruas (`%d`, bilangan bulat) dan panjang total (`%.1f`, satu desimal).

Hasil. Layar menampilkan (berdasarkan uji pada QGIS 4.0.2):

```
HASIL
Petak EPSG:32749 16 objek
Sungai EPSG:32749 2 objek
Permukiman EPSG:32749 2 objek
Batas_KPH EPSG:32749 3 objek
Petak valid: 16 tidak valid: 0
Sungai di KPH Gamma: 2 ruas, 570.3 m
```

Dari hasil ini Anda tahu: keempat layer memakai EPSG:32749 (meter), 16 petak semuanya valid, dan sungai yang masuk KPH Gamma sepanjang sekitar 570 meter. Empat layer baru juga muncul di panel Layers.

Bagian yang boleh diubah.

- **Baris 6-7, lokasi data.** Ganti dengan folder Anda.
- **Baris 29-32, nama berkas dan layer.** Ganti untuk memeriksa layer lain, misalnya `Jalan.gpkg` dengan layer `Jalan`.
- **Baris 43, nama KPH.** Ganti `'KPH Gamma'` dengan `'KPH Beta'` atau `'KPH Alpha'`.
- Baris 39, `METHOD`. Angka ini memilih cara pemeriksaan: 0 untuk pengaturan digitasi, 1 untuk QGIS, 2 untuk GEOS.

Jebakan.

- **Jalur salah.** Bila `PAKET` menunjuk folder yang tidak ada, skrip berhenti di baris 29 dengan `RuntimeError: Layer tidak terbaca: C:\latihan\paket-i1\Petak.gpkg`. Pesan ini jelas dan menyebut berkas yang bermasalah. Fungsi `muat` memang dibuat untuk itu.

- **Layer ganda.** Setiap skrip dijalankan, empat layer ditambahkan lagi ke proyek. Menjalankan dua kali menghasilkan layer `Batas_KPH` ganda, `Permukiman` ganda, dan seterusnya (uji saya: delapan layer setelah dua kali jalan). Hapus layer lama atau mulai dari proyek baru.
- **Fungsi yang tidak dipakai.** `luas_ha` dan `buf` ada di blok pembuka, tetapi skrip ini tidak memanggilnya. Itu bukan galat. Baca skrip dengan kritis: yang ada belum tentu dipakai.
- **Ukuran dalam meter.** Skrip menganggap satuan peta meter. Bila layer Anda berkoordinat derajat, panjang dan luas menjadi tidak berarti. Periksa baris `EPSG:` pada hasil.

Latihan variasi. Ubah skrip supaya mengukur sungai di KPH Beta.

Contoh jawaban. Ganti baris 43, lalu ganti juga kata `KPH Gamma` di dalam tanda kutip pada baris 46 menjadi `KPH Beta`, karena tulisan itu ditulis tetap:

```python
gamma = processing.run("native:extractbyexpression", {"INPUT": kph, "EXPRESSION": "\"NAMA_KPH\" = 'KPH Beta'",
"OUTPUT": "memory:"})["OUTPUT"]
```

Layar menampilkan `Sungai di KPH Beta: 1 ruas, 26.9 m`. Bila baris 46 tidak diganti, layar tetap menampilkan `Sungai di KPH Gamma: 1 ruas, 26.9 m`, padahal angkanya milik KPH Beta. Nama kotak `gamma` tidak perlu diganti, karena Python tidak peduli artinya. Untuk keterbacaan, sebaiknya Anda menggantinya juga.

> **Pola yang bisa dipakai ulang:** Kerangka pemeriksaan data: **muat dengan pemeriksa, cetak CRS dan jumlah, cek validitas, lalu hitung satu angka**. Untuk data lain, ganti tiga hal: **lokasi** (baris 6-7), **nama berkas dan layer** (baris 29-32), dan **ekspresi pemilih** (baris 43). Blok pembuka tidak perlu disentuh.

Skrip 18.2: Zona aman di sekeliling sungai

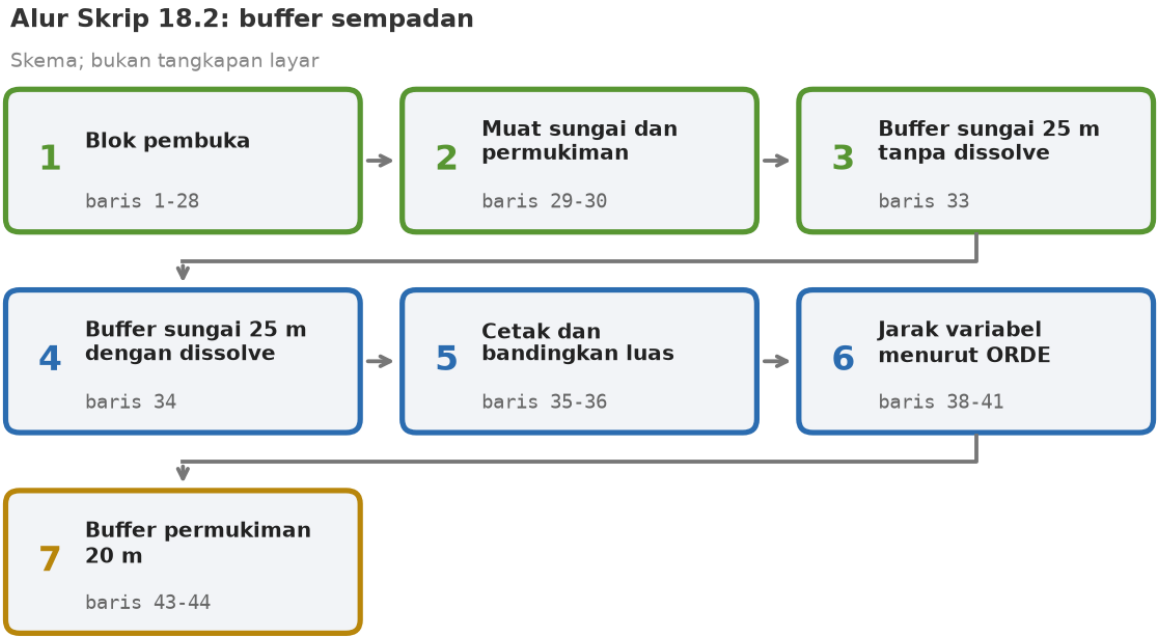
Berkas `2_1_buffer_sempadan.py`. Skrip ini hanya berisi hal baru dari baris 29 ke bawah.

Tujuan. Membuat buffer sungai 25 meter dengan dan tanpa penggabungan (dissolve), membuat buffer dengan jarak yang berbeda menurut orde sungai, lalu membuat buffer permukiman 20 meter dan membandingkan luasnya.

Bahan. `Sungai.gpkg` (kolom `ORDE`) dan `Permukiman.gpkg` dari `paket-i1`.

Buffer adalah zona selebar jarak tertentu di sekeliling sebuah objek. Bayangkan Anda berjalan menyusuri sungai sambil menggambar garis 25 meter di kiri dan kanannya. **Dissolve** artinya menggabungkan zona-zona yang bertumpuk menjadi satu bidang. Tanpa dissolve, daerah yang tumpang tindih terhitung dua kali.

Ilustrasi 18.4: Alur skrip 18.2



Skrip utuh:

```python
# -*- coding: utf-8 -*-
# Penulis: Badar Mubarok Yogaswara. Dijalankan di Python Console QGIS. Data sintetis, bukan kondisi lapangan nyata.
from qgis.core import QgsVectorLayer, QgsProject, QgsProperty
import processing

PAKET = r"C:\latihan\paket-i1"      # ubah sesuai letak folder di komputer Anda
DATA = r"C:\latihan\paket-data"


def muat(berkas, lapisan, nama):
    """Memuat satu layer GeoPackage dan menambahkannya ke proyek."""
    lyr = QgsVectorLayer("%s|layername=%s" % (berkas, lapisan), nama, "ogr")
    if not lyr.isValid():
        raise RuntimeError("Layer tidak terbaca: " + berkas)
    QgsProject.instance().addMapLayer(lyr)
    return lyr


def luas_ha(lyr):
    """Jumlah luas semua poligon dalam hektare (CRS harus UTM, satuan meter)."""
    return sum(f.geometry().area() for f in lyr.getFeatures()) / 10000.0


def buf(lyr, jarak):
    """Buffer dengan dissolve, hasil sementara."""
```

```
    return processing.run("native:buffer", {"INPUT": lyr, "DISTANCE": jarak, "DISSOLVE": True, "OUTPUT":
"memory:"})["OUTPUT"]


sungai = muat(PAKET + r"\Sungai.gpkg", "Sungai", "Sungai")
kampung = muat(PAKET + r"\Permukiman.gpkg", "Permukiman", "Permukiman")

# Buffer sungai 25 m, tanpa dissolve dan dengan dissolve
tanpa = processing.run("native:buffer", {"INPUT": sungai, "DISTANCE": 25, "DISSOLVE": False, "OUTPUT":
"memory:"})["OUTPUT"]
gabung = buf(sungai, 25)
print("Tanpa dissolve: %d objek, %.2f ha (tumpang tindih terhitung dua kali)" % (tanpa.featureCount(),
luas_ha(tanpa)))
print("Dengan dissolve: %d objek, %.2f ha" % (gabung.featureCount(), luas_ha(gabung)))

# Jarak berbeda menurut orde sungai: 30 m untuk orde 1, 15 m untuk orde 2
rumus = 'CASE WHEN "ORDE" = 1 THEN 30 ELSE 15 END'
variabel = processing.run("native:buffer", {"INPUT": sungai, "DISTANCE": QgsProperty.fromExpression(rumus),
"DISSOLVE": True, "OUTPUT": "memory:"})["OUTPUT"]
print("Jarak variabel: %.2f ha" % luas_ha(variabel))

# Buffer permukiman 20 m
print("Buffer permukiman 20 m: %.2f ha" % luas_ha(buf(kampung, 20)))
```

Penjelasan baris demi baris. Baris 1-28 adalah blok pembuka, sama dengan Skrip 18.1. Hanya `muat`, `luas_ha`, `buf`, dan `QgsProperty` yang dipakai di sini.

Baris	Tulisan	Artinya
1-28	blok pembuka	Sama persis dengan Skrip 18.1.
29-30	`sungai = muat(...)`, `kampung = muat(...)`	Muat dua layer.
33	`tanpa = processing.run("native:buffer", {... "DISSOLVE": False ...})["OUTPUT"]`	Panggil buffer langsung, dengan jarak 25 dan tanpa penggabungan. Hasilnya satu bidang untuk tiap ruas sungai.
34	`gabung = buf(sungai, 25)`	Panggil fungsi pembantu `buf` dari blok pembuka. Fungsi itu melakukan buffer dengan penggabungan. Hasilnya sama dengan memanggil langsung, hanya lebih singkat.
35-36	`print("... %d objek, %.2f ha ..." % (..., luas_ha(...)))`	Cetak jumlah objek dan luas dalam hektare, dua desimal. Fungsi `luas_ha` dipanggil di dalam `print`.
39	`rumus = 'CASE WHEN "ORDE" = 1 THEN 30 ELSE 15 END'`	Rumus dalam bahasa ekspresi QGIS, ditulis sebagai teks. Artinya: bila kolom `ORDE` bernilai 1, jaraknya 30 meter. Selain itu, 15 meter.
40	`"DISTANCE": QgsProperty.fromExpression(rumus)`	Alih-alih satu angka, jarak diisi dengan rumus. QGIS menghitung rumus itu untuk tiap objek sungai. Jadi tiap sungai punya jarak sendiri.
41	`print("Jarak variabel: %.2f ha" % luas_ha(variabel))`	Cetak luas hasilnya.

Baris	Tulisan	Artinya
44	`print("Buffer permukiman 20 m: %.2f ha" % luas_ha(buf(kampung, 20)))`	Satu baris, tiga langkah: buffer permukiman 20 meter, hitung luasnya, lalu cetak.

Hasil. Layar menampilkan (berdasarkan uji pada QGIS 4.0.2):

```
HASIL
Tanpa dissolve: 2 objek, 3.37 ha (tumpang tindih terhitung dua kali)
Dengan dissolve: 1 objek, 3.12 ha
Jarak variabel: 3.28 ha
Buffer permukiman 20 m: 1.18 ha
```

Selisih 3,37 dan 3,12, yaitu 0,25 hektare, adalah daerah tumpang tindih yang terhitung dua kali bila tanpa dissolve. Jarak variabel menghasilkan 3,28 hektare, lebih lebar dari 25 meter karena sungai berorde 1 mendapat 30 meter.

Bagian yang boleh diubah.

- **Baris 33-34, jarak 25.** Ganti di kedua baris sekaligus. Bila hanya satu, perbandingan dissolve menjadi tidak adil.
- **Baris 39, rumus.** Ganti angka 30 dan 15 untuk mengubah lebar tiap orde. Angka itu hanya contoh latihan, bukan ketentuan resmi sempadan sungai.
- **Baris 44, jarak 20.** Ganti jarak buffer permukiman.
- Baris 33, `"DISSOLVE": False`. Ganti `True` untuk menggabungkan hasil.

Jebakan.

- **Kolom salah ketik di dalam rumus.** Bila Anda menulis `"ORDEE"`, tidak ada galat. QGIS menganggap kolom itu kosong, sehingga semua sungai jatuh ke jarak 15 meter. Uji saya: `Jarak variabel: 1.22 ha`, jauh lebih kecil dari 3,28. Periksa angka hasil, bukan hanya ada atau tidaknya galat.
- **Satuan harus meter.** Fungsi `luas_ha` membaca luas dalam satuan peta. Pada layer berkoordinat derajat, angkanya tidak berarti.
- **Dua tempat untuk satu angka.** Jarak 25 ada di baris 33 dan 34. Mengubah satu baris saja membuat tampilan "tanpa" dan "dengan" dissolve tidak setara.
- **Tumpang tindih.** Tanpa dissolve, luas terlalu besar (3,37 lawan 3,12). Selalu gabungkan sebelum menghitung luas total sempadan.

Latihan variasi. Lebarkan sempadan sungai menjadi 50 meter, baik tanpa maupun dengan dissolve.

Contoh jawaban. Ganti angka 25 pada baris 33 dan 34:

```
PYTHON
tanpa = processing.run("native:buffer", {"INPUT": sungai, "DISTANCE": 50, "DISSOLVE": False, "OUTPUT":
"memory:"})["OUTPUT"]
gabung = buf(sungai, 50)
```

Layar menampilkan `Tanpa dissolve: 2 objek, 7.50 ha ...` dan `Dengan dissolve: 1 objek, 6.52 ha`. Selisihnya 0,98 hektare, jauh lebih besar daripada selisih pada 25 meter (0,25 hektare). Itu wajar: sempadan yang lebih lebar saling bertumpuk lebih banyak.

> **Pola yang bisa dipakai ulang:** Kerangka buffer: muat layer, panggil `native:buffer` dengan jarak dan DISSOLVE, hitung luas. Untuk objek lain, ganti tiga hal: **layer masukan** (baris 29-30), **jarak** (baris 33-34 dan 44), dan **pilihan DISSOLVE** (baris 33). Untuk jarak yang berbeda tiap objek, ganti angka dengan `QgsProperty.fromExpression(...)`.

Skrip 18.3: Dua peta bertumpuk, tiga cara membaca tumpukannya

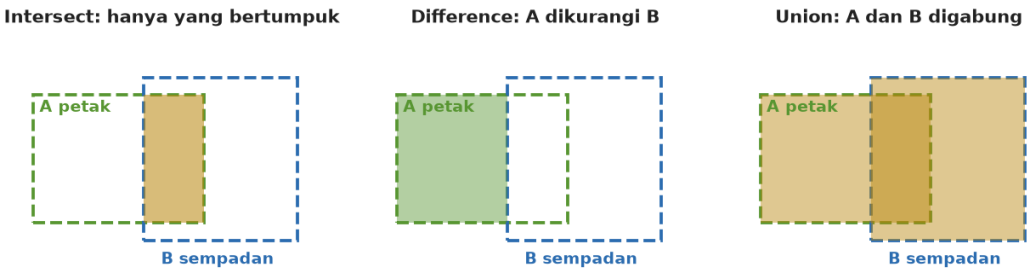
Berkas `3_1_overlay.py`. Seperti Skrip 18.2, baris 1-28 adalah blok pembuka.

Tujuan. Memperlihatkan tiga operasi tumpang susun: Intersect, Difference, dan Union. Skrip menghitung luas bagian petak yang masuk sempadan, luas areal efektif, serta luas blok perhutanan sosial di dalam dan di luar petak.

Bahan. `Petak.gpkg`, `Sungai.gpkg`, `Permukiman.gpkg`, dan `Perhutanan_Sosial.gpkg` dari `paket-i1`.

Bayangkan dua lembar plastik transparan yang ditumpuk: lembar petak dan lembar sempadan. Ada tiga cara menggunting hasilnya, seperti pada gambar.

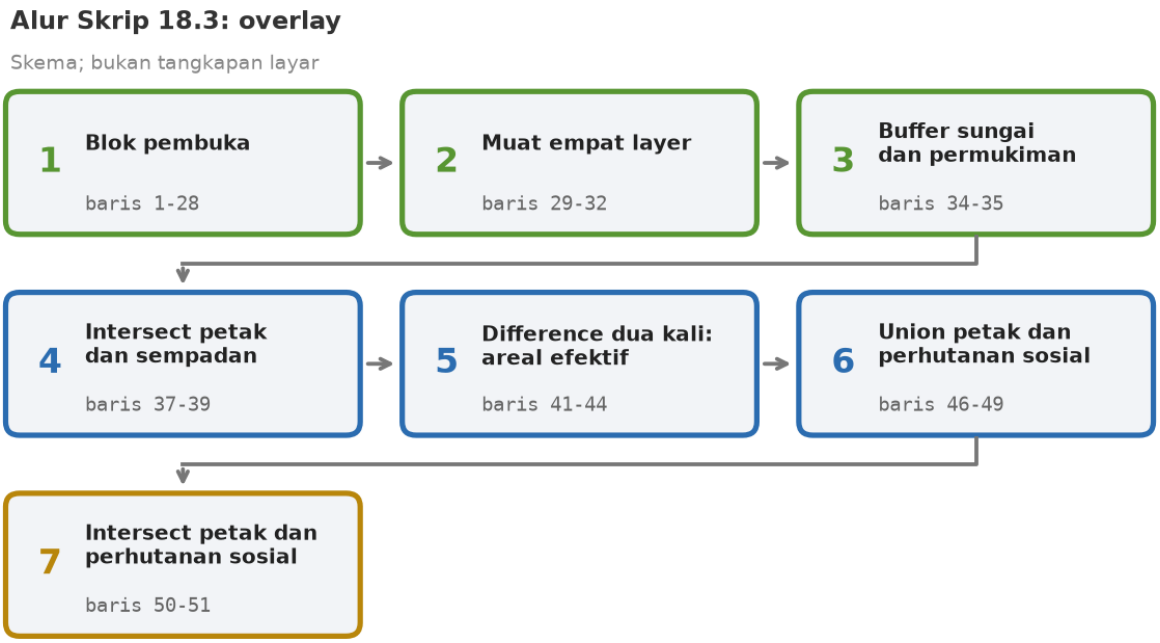
Ilustrasi 18.5: Intersect, Difference, dan Union



Skema; bukan tangkapan layar; warna menandai hasil

- **Intersect**: ambil hanya bagian yang bertumpuk.
- **Difference**: ambil bagian lembar pertama yang tidak tertutup lembar kedua. Urutan masukan menentukan hasil.
- **Union**: ambil seluruh bagian kedua lembar, dengan garis potong di tempat mereka bertumpuk.

Ilustrasi 18.6: Alur skrip 18.3



Skrip utuh:

```python
# -*- coding: utf-8 -*-
# Penulis: Badar Mubarok Yogaswara. Dijalankan di Python Console QGIS. Data sintetis, bukan kondisi lapangan nyata.
from qgis.core import QgsVectorLayer, QgsProject, QgsProperty
import processing

PAKET = r"C:\latihan\paket-i1"      # ubah sesuai letak folder di komputer Anda
DATA = r"C:\latihan\paket-data"


def muat(berkas, lapisan, nama):
    """Memuat satu layer GeoPackage dan menambahkannya ke proyek."""
    lyr = QgsVectorLayer("%s|layername=%s" % (berkas, lapisan), nama, "ogr")
    if not lyr.isValid():
        raise RuntimeError("Layer tidak terbaca: " + berkas)
    QgsProject.instance().addMapLayer(lyr)
    return lyr


def luas_ha(lyr):
    """Jumlah luas semua poligon dalam hektare (CRS harus UTM, satuan meter)."""
    return sum(f.geometry().area() for f in lyr.getFeatures()) / 10000.0


def buf(lyr, jarak):
    """Buffer dengan dissolve, hasil sementara."""
    return processing.run("native:buffer", {"INPUT": lyr, "DISTANCE": jarak, "DISSOLVE": True, "OUTPUT": "memory:"})["OUTPUT"]


petak = muat(PAKET + r"\Petak.gpkg", "Petak", "Petak")
sungai = muat(PAKET + r"\Sungai.gpkg", "Sungai", "Sungai")
kampung = muat(PAKET + r"\Permukiman.gpkg", "Permukiman", "Permukiman")
ps = muat(PAKET + r"\Perhutanan_Sosial.gpkg", "Perhutanan_Sosial", "Perhutanan_Sosial")
```

```
sempadan = buf(sungai, 25)
zona_kampung = buf(kampung, 20)

# Intersect: bagian petak yang masuk sempadan sungai
irisan = processing.run("native:intersection", {"INPUT": petak, "OVERLAY": sempadan, "OUTPUT": "memory:"})
["OUTPUT"]
print("Intersect: %d potongan, %.2f ha" % (irisan.featureCount(), luas_ha(irisan)))

# Difference: petak dikurangi sempadan, lalu dikurangi zona permukiman
sisa = processing.run("native:difference", {"INPUT": petak, "OVERLAY": sempadan, "OUTPUT": "memory:"})
["OUTPUT"]
efektif = processing.run("native:difference", {"INPUT": sisa, "OVERLAY": zona_kampung, "OUTPUT": "memory:"})
["OUTPUT"]
print("Difference sungai: %.2f ha; ditambah permukiman: %.2f ha" % (luas_ha(sisa), luas_ha(efektif)))

# Union: petak digabung dengan blok perhutanan sosial
gab = processing.run("native:union", {"INPUT": petak, "OVERLAY": ps, "OUTPUT": "memory:"})["OUTPUT"]
luar = [f for f in gab.getFeatures() if f["KODE_PETAK"] is None]
print("Union: %d objek, %d di luar petak (%.2f ha)" % (gab.featureCount(), len(luar), sum(f.geometry().area()
for f in luar) / 10000.0))
irisan_ps = processing.run("native:intersection", {"INPUT": petak, "OVERLAY": ps, "OUTPUT": "memory:"})
["OUTPUT"]
print("Petak dalam perhutanan sosial: %d potongan, %.2f ha dari %.2f ha blok" % (irisan_ps.featureCount(),
luas_ha(irisan_ps), luas_ha(ps)))
```

Penjelasan baris demi baris. Baris 1-28 adalah blok pembuka. Fungsi `buf` dan `luas_ha` dipakai di sini.

Baris	Tulisan	Artinya
29-32	`petak = muat(...)` dan tiga baris di bawahnya	Muat empat layer. Kotak `ps` berisi perhutanan sosial.
34-35	`sempadan = buf(sungai, 25), zona_kampung = buf(kampung, 20)`	Buat dua bidang zona aman memakai fungsi pembantu `buf`.
38	`irisan = processing.run("native:intersection", {...})["OUTPUT"]`	Intersect: ambil bagian petak yang bertumpuk dengan sempadan. `INPUT` adalah lembar pertama, dan `OVERLAY` adalah lembar tumpukannya.
39	`print("Intersect: %d potongan, %.2f ha" % (...))`	Cetak jumlah potongan dan luasnya.
42	`sisa = processing.run("native:difference", {...})["OUTPUT"]`	Difference pertama: petak dikurangi sempadan.
43	`efektif = processing.run("native:difference", {"INPUT": sisa, ...})["OUTPUT"]`	Difference kedua: hasil pertama dikurangi zona permukiman. Inilah areal efektif. Hasil satu alat menjadi masukan alat berikutnya, seperti rantai.
44	`print("Difference sungai: ...; ditambah permukiman: ...")`	Cetak luas sesudah pengurangan pertama, lalu sesudah pengurangan kedua.

Baris	Tulisan	Artinya
47	`gab = processing.run("native:union", {...})["OUTPUT"]`	Union: gabungkan petak dengan blok perhutanan sosial.
48	`luar = [f for f in gab.getFeatures() if f["KODE_PETAK"] is None]`	Kumpulkan objek hasil union yang `KODE_PETAK`-nya kosong. Objek yang hanya berasal dari blok perhutanan sosial tidak membawa kode petak. Jadi bagian itu berada di luar petak.
49	`print("Union: %d objek, %d di luar petak (%.2f ha)" % (...))`	Cetak jumlah semua objek, jumlah bagian di luar petak (`len(luar)`), dan luas bagian itu.
50-51	`irisan_ps = ..., print(...)`	Intersect petak dengan blok perhutanan sosial, lalu cetak jumlah potongan, luas, dan luas seluruh blok.

Hasil. Layar menampilkan (berdasarkan uji pada QGIS 4.0.2):

```
HASIL
Intersect: 8 potongan, 2.83 ha
Difference sungai: 13.17 ha; ditambah permukiman: 12.39 ha
Union: 23 objek, 1 di luar petak (0.29 ha)
Petak dalam perhutanan sosial: 6 potongan, 3.47 ha dari 3.77 ha blok
```

Cara membaca. Dari 16 hektare petak, sempadan sungai mengambil 2,83 hektare, dan sisanya 13,17 hektare. Zona permukiman mengambil 0,78 hektare lagi, sehingga areal efektif 12,39 hektare. Cek angka: 16,00 dikurangi 2,83 sama dengan 13,17, dan 13,17 dikurangi 12,39 sama dengan 0,78.

Pada blok perhutanan sosial, 3,47 hektare ada di dalam petak dan 0,29 hektare di luar. Jumlahnya, 3,76, sama dengan 3,77 setelah pembulatan.

Bagian yang boleh diubah.

- **Baris 34-35, jarak buffer.** Ganti 25 dan 20.
- **Baris 29-32, layer.** Ganti dengan layer lain, tetapi kolom `KODE_PETAK` di baris 48 harus ada di layer petak Anda.
- **Baris 38, 42, 43, 47, 50, urutan masukan.** Tukar `INPUT` dan `OVERLAY` untuk melihat akibatnya pada Difference.

Jebakan.

- **Urutan masukan Difference.** Difference tidak simetris. Saya menukar masukan baris 42 (sempadan sebagai `INPUT`, petak sebagai `OVERLAY`). Hasilnya `Difference sungai: 0.29 ha`, yaitu bagian sempadan yang berada di luar petak. Untuk areal efektif, petaklah yang harus jadi `INPUT`.
- **Penanda "di luar petak" memakai kolom kosong.** Baris 48 menganggap `KODE_PETAK` kosong berarti "bukan dari petak". Bila data petak Anda sendiri punya kode yang kosong, hitungannya salah.
- **Jumlah objek berbeda antar alat.** Intersect menghasilkan 8 potongan. Union menghasilkan 23 objek. Jumlah objek tidak sama dengan jumlah petak. Jangan menghitung petak dari jumlah potongan.

- **Satuan meter.** Sama dengan Skrip 18.2: `luas_ha` hanya benar untuk layer berkoordinat meter.

Latihan variasi. Hitung luas bagian petak yang masuk zona permukiman (`zona_kampung`) dengan Intersect.

Contoh jawaban. Tambahkan dua baris di akhir skrip:

```python
irisan_kampung = processing.run("native:intersection", {"INPUT": petak, "OVERLAY": zona_kampung, "OUTPUT": "memory:"})["OUTPUT"]
print("Petak dalam zona permukiman: %d potongan, %.2f ha" % (irisan_kampung.featureCount(), luas_ha(irisan_kampung)))
```

Layar menampilkan `Petak dalam zona permukiman: 5 potongan, 1.18 ha`. Angka 1,18 sama dengan luas buffer permukiman di Skrip 18.2. Artinya seluruh zona itu berada di dalam petak.

> **Pola yang bisa dipakai ulang:** Kerangka tumpang susun: siapkan dua layer, panggil alat dengan `INPUT` dan `OVERLAY`, hitung luas hasil. Untuk soal lain, ganti tiga hal: **alat** (`intersection`, `difference`, atau `union`), **urutan masukan**, dan **layer tumpukan**. Hasil satu alat boleh langsung dipakai alat berikutnya.

Skrip 18.4: Melebur batas dan merekap luas

Berkas `4_1_dissolve_rekap.py`. Baris 1-28 adalah blok pembuka.

Tujuan. Dari areal efektif (petak dikurangi sempadan sungai dan zona permukiman), membuat rekap luas per jenis tanaman dengan dissolve, memecah bidang gabungan menjadi bagian-bagian, lalu membandingkannya dengan statistik berkelompok.

Bahan. `Petak.gpkg` (kolom `JENIS`, `TAHUN_TANAM`), `Sungai.gpkg`, dan `Permukiman.gpkg` dari `paket-i1`.

Dissolve melebur batas antar objek yang punya isi kolom sama, seperti menghapus tembok antar kamar yang sama fungsinya. Petak berjenis Jati yang berjauhan lalu menjadi satu objek. Objek hasil seperti itu disebut **multipart** (satu objek terdiri dari banyak bagian terpisah). Lawannya **singlepart**: satu objek, satu bagian.

Ilustrasi 18.7: Alur skrip 18.4



Skrip utuh:

```python
# -*- coding: utf-8 -*-
# Penulis: Badar Mubarok Yogaswara. Dijalankan di Python Console QGIS. Data sintetis, bukan kondisi lapangan nyata.
from qgis.core import QgsVectorLayer, QgsProject, QgsProperty
import processing

PAKET = r"C:\latihan\paket-i1"      # ubah sesuai letak folder di komputer Anda
DATA = r"C:\latihan\paket-data"


def muat(berkas, lapisan, nama):
    """Memuat satu layer GeoPackage dan menambahkannya ke proyek."""
    lyr = QgsVectorLayer("%s|layername=%s" % (berkas, lapisan), nama, "ogr")
    if not lyr.isValid():
        raise RuntimeError("Layer tidak terbaca: " + berkas)
    QgsProject.instance().addMapLayer(lyr)
    return lyr


def luas_ha(lyr):
    """Jumlah luas semua poligon dalam hektare (CRS harus UTM, satuan meter)."""
    return sum(f.geometry().area() for f in lyr.getFeatures()) / 10000.0


def buf(lyr, jarak):
    """Buffer dengan dissolve, hasil sementara."""
```

```
    return processing.run("native:buffer", {"INPUT": lyr, "DISTANCE": jarak, "DISSOLVE": True, "OUTPUT":
"memory:"})["OUTPUT"]


petak = muat(PAKET + r"\Petak.gpkg", "Petak", "Petak")
sungai = muat(PAKET + r"\Sungai.gpkg", "Sungai", "Sungai")
kampung = muat(PAKET + r"\Permukiman.gpkg", "Permukiman", "Permukiman")

sisa = processing.run("native:difference", {"INPUT": petak, "OVERLAY": buf(sungai, 25), "OUTPUT": "memory:"})
["OUTPUT"]
efektif = processing.run("native:difference", {"INPUT": sisa, "OVERLAY": buf(kampung, 20), "OUTPUT":
"memory:"})["OUTPUT"]

# Dissolve per jenis tanaman. Nama kolom diperiksa dulu: salah ketik tidak boleh lolos diam-diam
KOLOM = "JENIS"
assert petak.fields().indexOf(KOLOM) >= 0, "Kolom tidak ada di layer Petak: " + KOLOM
per_jenis = processing.run("native:dissolve", {"INPUT": efektif, "FIELD": [KOLOM], "OUTPUT": "memory:"})
["OUTPUT"]
awal = processing.run("native:dissolve", {"INPUT": petak, "FIELD": [KOLOM], "OUTPUT": "memory:"})["OUTPUT"]
luas_awal = {f["JENIS"]: f.geometry().area() / 10000.0 for f in awal.getFeatures()}
for f in sorted(per_jenis.getFeatures(), key=lambda f: f["JENIS"]):
    print("%-7s awal %.2f ha, efektif %.2f ha (multipart: %s)" % (f["JENIS"], luas_awal[f["JENIS"]],
f.geometry().area() / 10000.0, f.geometry().isMultipart()))

# Dissolve tanpa field: satu poligon untuk seluruh areal efektif
semua = processing.run("native:dissolve", {"INPUT": efektif, "OUTPUT": "memory:"})["OUTPUT"]
print("Dissolve semua: %d objek, %.2f ha" % (semua.featureCount(), luas_ha(semua)))

# Pecah multipart menjadi satu bagian per objek
tunggal = processing.run("native:multiparttosingleparts", {"INPUT": per_jenis, "OUTPUT": "memory:"})["OUTPUT"]
print("Setelah multipart ke singlepart: %d objek" % tunggal.featureCount())

# Atribut lain setelah dissolve: nilai diambil dari satu objek saja (hati-hati)
for f in sorted(per_jenis.getFeatures(), key=lambda f: f["JENIS"]):
    print(f["JENIS"], "TAHUN_TANAM tersisa =", f["TAHUN_TANAM"])

# Rekap luas tanpa dissolve: tambah field luas, lalu Statistics by categories
hitung = processing.run("native:fieldcalculator", {"INPUT": efektif, "FIELD_NAME": "LUAS_HA", "FIELD_TYPE": 0,
"FIELD_LENGTH": 10, "FIELD_PRECISION": 4, "FORMULA": "area($geometry) / 10000", "OUTPUT": "memory:"})["OUTPUT"]
statistik = processing.run("qgis:statisticsbycategories", {"INPUT": hitung, "VALUES_FIELD_NAME": "LUAS_HA",
"CATEGORIES_FIELD_NAME": ["JENIS"], "OUTPUT": "memory:"})["OUTPUT"]
for f in sorted(statistik.getFeatures(), key=lambda f: f["JENIS"]):
    print("Statistik %-7s %d petak, jumlah luas %.2f ha" % (f["JENIS"], f["count"], f["sum"]))
```

Penjelasan baris demi baris. Baris 1-28 adalah blok pembuka.

Baris	Tulisan	Artinya
29-31	`petak = muat(...)`, …	Muat tiga layer.
33-34	`sisa = ...difference...`, `efektif = ...difference...`	Dua pengurangan berantai, seperti di Skrip 18.3. Bedanya, buffer dibuat di tempat dengan `buf(sungai, 25)` sebagai isian. Hasil akhirnya, `efektif`, adalah areal efektif.
36-37	komentar dan `KOLOM = "JENIS"`	Simpan nama kolom pengelompok di satu kotak bernama `KOLOM`. Nama itu hanya ditulis sekali, lalu dipakai di baris 38, 39, dan 40.
38	`assert petak.fields().indexOf(KOLOM) >= 0, "Kolom tidak ada di layer Petak: " + KOLOM`	Pengaman nama kolom. `petak.fields().indexOf(KOLOM)` mencari urutan kolom itu di tabel petak. Bila kolom tidak ada, hasilnya -1. `assert` memastikan hasilnya 0 atau lebih. Bila tidak, skrip berhenti dengan pesan di sesudah koma (Bab 10).

Baris	Tulisan	Artinya
39	`per_jenis = processing.run("native:dissolve", {... "FIELD": [KOLOM] ...})["OUTPUT"]`	Lebur areal efektif per jenis. Isian `FIELD` berupa daftar kolom: objek dengan isi `JENIS` sama digabung.
40	`awal = processing.run("native:dissolve", {"INPUT": petak, ...})["OUTPUT"]`	Lakukan hal sama pada petak awal, untuk perbandingan.
41	`luas_awal = {f["JENIS"]: f.geometry().area() / 10000.0 for f in awal.getFeatures()}`	Susun kamus: kunci berupa nama jenis, nilai berupa luas awal dalam hektare. Ini *dictionary comprehension*, mirip list comprehension tetapi menghasilkan kamus.
42	`for f in sorted(per_jenis.getFeatures(), key=lambda f: f["JENIS"]):`	Ulangi tiap objek hasil, urut abjad menurut jenis. `lambda f: f["JENIS"]` adalah aturan urutan: "ambil kolom `JENIS` dari tiap objek" (Bab 7).
43	`print("%-7s awal %.2f ha, efektif %.2f ha (multipart: %s)" % (...))`	Cetak jenis, luas awal, luas efektif, dan apakah objek itu multipart. `%-7s` berarti teks rata kiri selebar 7 karakter.
46-47	`semua = ...dissolve... (tanpa FIELD), print(...)`	Lebur tanpa menyebut kolom: semua objek menjadi satu bidang. Cetak jumlah objek dan luasnya.
50-51	`tunggal = ..."native:multiparttosingleparts"...`	Pecah objek multipart menjadi satu objek per bagian. Cetak jumlah objek sesudahnya.
54-55	`for f in sorted(...): print(f["JENIS"], "TAHUN_TANAM tersisa =", ...)`	Cetak nilai `TAHUN_TANAM` yang tersisa tiap jenis setelah dissolve. Komentar baris 53 mengingatkan agar berhati-hati.
58	`hitung = processing.run("native:fieldcalculator", {...})["OUTPUT"]`	Tambahkan kolom `LUAS_HA` ke tiap petak efektif. `FIELD_TYPE: 0` berarti angka desimal, `FIELD_PRECISION: 4` empat angka di belakang koma, dan `FORMULA` adalah rumus luas dibagi 10.000.
59	`statistik = processing.run("qgis:statisticsbycategories", {...})["OUTPUT"]`	Hitung statistik kolom `LUAS_HA` untuk tiap kategori `JENIS`. Alat ini hanya menghitung, tidak melebur.
60-61	`for f in sorted(statistik.getFeatures(), ...): print(... f["count"], f["sum"])`	Cetak jumlah potongan dan jumlah luas tiap jenis. Kolom `count` dan `sum` dibuat otomatis oleh alat.

Hasil. Layar menampilkan (berdasarkan uji pada QGIS 4.0.2):

```
HASIL
Jati    awal 6.00 ha, efektif 4.65 ha (multipart: True)
```

```
Mahoni  awal 4.00 ha, efektif 3.63 ha (multipart: True)
Pinus   awal 3.00 ha, efektif 2.08 ha (multipart: True)
Sengon  awal 3.00 ha, efektif 2.03 ha (multipart: True)
Dissolve semua: 1 objek, 12.39 ha
Setelah multipart ke singlepart: 21 objek
Jati TAHUN_TANAM tersisa = 2008
Mahoni TAHUN_TANAM tersisa = 2015
Pinus TAHUN_TANAM tersisa = 2008
Sengon TAHUN_TANAM tersisa = 2015
Statistik Jati    6 petak, jumlah luas 4.65 ha
Statistik Mahoni  4 petak, jumlah luas 3.63 ha
Statistik Pinus   3 petak, jumlah luas 2.08 ha
Statistik Sengon  3 petak, jumlah luas 2.03 ha
```

Pada uji tanpa jendela utama, QGIS menambah satu baris peringatan `DeprecationWarning` sebelum empat baris statistik. Peringatan itu berasal dari alat bawaan QGIS, bukan dari skrip, dan tidak memengaruhi hasil.

Dua cara menghasilkan angka yang sama. Rekap dissolve (4,65, 3,63, 2,08, 2,03) sama dengan jumlah luas dari statistik. Bedanya, dissolve menghasilkan objek geometri baru, sedangkan statistik hanya menghasilkan tabel angka.

Bagian yang boleh diubah.

- Baris 37, `KOLOM = "JENIS"`. Ganti dengan kolom lain, misalnya `"TAHUN_TANAM"`. Nama kolom harus persis sama dengan di tabel atribut. Pengaman baris 38 ikut memeriksanya.
- **Baris 33-34, jarak buffer.** Ganti 25 dan 20.
- Baris 58, `FIELD_PRECISION`. Ganti 4 untuk mengubah jumlah angka di belakang koma.
- Baris 59, `CATEGORIES_FIELD_NAME`. Ganti kolom pengelompok.

Jebakan.

- **Pengaman nama kolom (baris 37-38).** Alat dissolve tidak protes bila nama kolom salah. Semua objek melebur menjadi satu, dan skrip tetap mencetak angka yang keliru. Baris 38 mencegahnya. Saya mengganti `KOLOM` menjadi `"JENSI"`. Skrip berhenti dengan `AssertionError: Kolom tidak ada di layer Petak: JENSI`, sebelum ada angka tercetak. Bila Anda mengganti kolom, ubah `KOLOM` saja, jangan menulis nama kolom langsung di baris 39-40.
- **Pengaman hanya menjaga satu kotak.** Baris 38 memeriksa `KOLOM` pada layer petak. Nama `JENIS` di baris 41-43, 54, dan 59-61 ditulis langsung dan tidak diperiksa. Setelah mengganti kolom, periksa angka hasilnya.
- **Atribut lain setelah dissolve tidak representatif.** Setelah dissolve, kolom selain kolom pengelompok mengambil nilai dari satu objek saja. `TAHUN_TANAM` Jati tertulis 2008, padahal petak Jati ditanam di tahun berbeda (2008, 2012, 2015, 2019). Jangan memakai kolom itu sebagai fakta.
- **Multipart sulit dipakai.** Hasil dissolve per jenis berupa 4 objek multipart. Bila perlu menghitung bagian, pecah dulu (baris 50).
- **Peringatan tanpa dampak.** Pesan `DeprecationWarning` pada alat statistik bukan galat Anda.

Latihan variasi. Rekap luas areal efektif per tahun tanam, bukan per jenis.

Contoh jawaban. Biarkan baris 39-43 seperti semula, karena `per_jenis` yang dibuat di sana masih dipakai baris 50 dan 54 (bila baris itu dihapus, skrip berhenti dengan `NameError: name 'per_jenis' is not defined`). Sisipkan tiga baris ini tepat sesudah baris 43:

```
PYTHON
per_tahun = processing.run("native:dissolve", {"INPUT": efektif, "FIELD": ["TAHUN_TANAM"], "OUTPUT":
"memory:"})["OUTPUT"]
for f in sorted(per_tahun.getFeatures(), key=lambda f: f["TAHUN_TANAM"]):
    print(f["TAHUN_TANAM"], "efektif %.2f ha" % (f.geometry().area() / 10000.0))
```

Empat baris jenis tetap tampil lebih dulu, lalu layar menampilkan `2008 efektif 2.89 ha`, `2012 efektif 3.59 ha`, `2015 efektif 2.81 ha`, dan `2019 efektif 3.09 ha`. Jumlahnya 12,38, sama dengan 12,39 setelah pembulatan.

> **Pola yang bisa dipakai ulang:** Kerangka rekap: **siapkan areal, dissolve menurut kolom, cetak luas per kelompok**. Untuk rekap lain, ganti tiga hal: **layer masukan** (baris 29-34), **kolom pengelompok** (baris 37 dan 59), dan **kolom yang dicetak** (baris 41-43). Bila hanya perlu angka, statistik berkelompok (baris 58-61) lebih ringan.

Skrip 18.5: Menghubungkan data dari letaknya

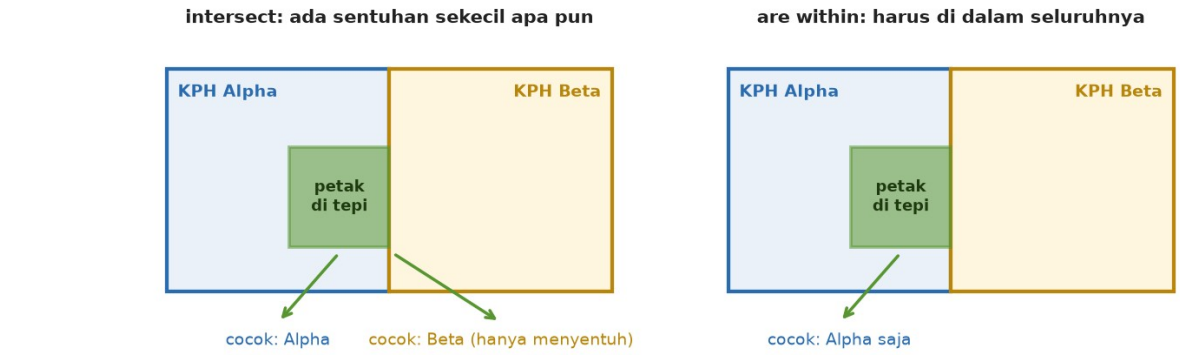
Berkas `5_1_spatial_join.py`. Skrip ini yang terpanjang di bab ini, dan paling padat. Tabel dan hasilnya dikelompokkan per tugas. Baris 1-28 adalah blok pembuka.

Tujuan. Memberi nama KPH pada tiap petak berdasarkan letaknya, menghitung titik panas per petak dan per KPH, memilih petak yang kena titik panas, mencari jalan terdekat dari tiap titik, lalu menghitung titik per jenis tanaman.

Bahan. `Petak.gpkg` dan `Jalan.gpkg` dari `paket-i1`, serta `Batas_KPH.gpkg` dan `Hotspot_Nasional.gpkg` dari `paket-data`.

Spatial join menyalin atribut dari satu layer ke layer lain menurut hubungan letak, bukan menurut kolom yang sama. Bayangkan menempelkan label nama KPH pada tiap petak dengan aturan: "tempel label dari KPH tempat petak ini berada". Aturan hubungan letak itu disebut **predikat** (misalnya *intersect*, *are within*). Ada dua cara menggabung: **satu-ke-banyak** (satu petak bisa muncul di beberapa baris bila cocok dengan beberapa objek) dan **satu-ke-satu** (hanya satu baris tiap petak).

Ilustrasi 18.8: Predikat intersect dan are within



Skema; bukan tangkapan layar; ilustrasi umum, bukan data tertentu

Ilustrasi 18.9: Alur skrip 18.5



Skrip utuh:

```python
# -*- coding: utf-8 -*-
# Penulis: Badar Mubarok Yogaswara. Dijalankan di Python Console QGIS. Data sintetis, bukan kondisi lapangan nyata.
from qgis.core import QgsVectorLayer, QgsProject, QgsProperty
import processing
```

```
PAKET = r"C:\latihan\paket-i1"      # ubah sesuai letak folder di komputer Anda
DATA = r"C:\latihan\paket-data"


def muat(berkas, lapisan, nama):
    """Memuat satu layer GeoPackage dan menambahkannya ke proyek."""
    lyr = QgsVectorLayer("%s|layername=%s" % (berkas, lapisan), nama, "ogr")
    if not lyr.isValid():
        raise RuntimeError("Layer tidak terbaca: " + berkas)
    QgsProject.instance().addMapLayer(lyr)
    return lyr


def luas_ha(lyr):
    """Jumlah luas semua poligon dalam hektare (CRS harus UTM, satuan meter)."""
    return sum(f.geometry().area() for f in lyr.getFeatures()) / 10000.0


def buf(lyr, jarak):
    """Buffer dengan dissolve, hasil sementara."""
    return processing.run("native:buffer", {"INPUT": lyr, "DISTANCE": jarak, "DISSOLVE": True, "OUTPUT":
"memory:"})["OUTPUT"]


from collections import Counter

petak = muat(PAKET + r"\Petak.gpkg", "Petak", "Petak")
jalan = muat(PAKET + r"\Jalan.gpkg", "Jalan", "Jalan")
kph = muat(DATA + r"\Batas_KPH.gpkg", "Batas_KPH", "Batas_KPH")
titik = muat(DATA + r"\Hotspot_Nasional.gpkg", "Hotspot_Nasional", "Hotspot_Nasional")

# 1. Spatial join petak ke KPH. Predikat 0 = intersect, 5 = are within. Metode 0 = satu-ke-banyak
for nama, pred in (("intersect", 0), ("are within", 5)):
    j = processing.run("native:joinattributesbylocation", {"INPUT": petak, "PREDICATE": [pred], "JOIN": kph,
"JOIN_FIELDS": ["NAMA_KPH"], "METHOD": 0, "OUTPUT": "memory:"})["OUTPUT"]
    print("Join dengan %-10s: %d baris (petak ada 16)" % (nama, j.featureCount()))

# 2. Join yang benar: 'are within', ambil satu (metode 1)
petak_kph = processing.run("native:joinattributesbylocation", {"INPUT": petak, "PREDICATE": [5], "JOIN": kph,
"JOIN_FIELDS": ["NAMA_KPH"], "METHOD": 1, "OUTPUT": "memory:"})["OUTPUT"]
print("Petak per KPH:", dict(Counter(f["NAMA_KPH"] for f in petak_kph.getFeatures())))

# 3. Hitung titik panas per petak
hitung = processing.run("native:countpointsinpolygon", {"POLYGONS": petak_kph, "POINTS": titik, "FIELD":
"JML_TITIK", "OUTPUT": "memory:"})["OUTPUT"]
per_petak = {f["KODE_PETAK"]: int(f["JML_TITIK"]) for f in hitung.getFeatures() if f["JML_TITIK"] > 0}
print("Titik per petak:", per_petak, "jumlah =", sum(per_petak.values()))
per_kph = Counter()
for f in hitung.getFeatures():
    per_kph[f["NAMA_KPH"]] += int(f["JML_TITIK"])
print("Titik per KPH:", dict(per_kph))

# 4. Ringkasan berdasarkan lokasi (join summary): hitung titik per petak
ringkas = processing.run("native:joinbylocationsummary", {"INPUT": petak, "PREDICATE": [0], "JOIN": titik,
"JOIN_FIELDS": ["ID_Titik"], "SUMMARIES": [0], "DISCARD_NONMATCHING": False, "OUTPUT": "memory:"})["OUTPUT"]
kolom = ringkas.fields()[ringkas.fields().count() - 1].name()
print("Kolom hasil ringkasan:", kolom)
print("Total count:", int(sum((f[kolom] or 0) for f in ringkas.getFeatures())))

# 5. Select by location: petak yang beririsan dengan titik panas
processing.run("native:selectbylocation", {"INPUT": petak, "PREDICATE": [0], "INTERSECT": titik, "METHOD": 0})
print("Petak terpilih:", sorted(f["KODE_PETAK"] for f in petak.selectedFeatures()))
petak.removeSelection()

# 6. Jalan terdekat dari tiap titik panas
dekat = processing.run("native:joinbynearest", {"INPUT": titik, "INPUT_2": jalan, "FIELDS_TO_COPY": ["NAMA"],
"NEIGHBORS": 1, "OUTPUT": "memory:"})["OUTPUT"]
baris = sorted((f["distance"], f["ID_Titik"], f["NAMA"]) for f in dekat.getFeatures())
```

```
print("Terdekat ke jalan: %s %.1f m (%s); terjauh: %s %.1f m" % (baris[0][1], baris[0][0], baris[0][2],
baris[-1][1], baris[-1][0]))

# 7. Titik panas ke petak (are within, satu-ke-satu), lalu hitung titik per jenis
titik_petak = processing.run("native:joinattributesbylocation", {"INPUT": titik, "PREDICATE": [5], "JOIN":
petak, "JOIN_FIELDS": ["KODE_PETAK", "JENIS"], "METHOD": 1, "OUTPUT": "memory:"})["OUTPUT"]
print("Titik di petak P07:", sorted(f["ID_Titik"] for f in titik_petak.getFeatures() if f["KODE_PETAK"] ==
"P07"))
per_jenis = {}
for f in titik_petak.getFeatures():
    per_jenis.setdefault(f["JENIS"], []).append(f["ID_Titik"])
for jenis, daftar in sorted(per_jenis.items()):
    print("Titik pada %-7s: %d %s" % (jenis, len(daftar), sorted(daftar)))
```

Penjelasan baris demi baris. Baris 1-28 adalah blok pembuka. Dua perkakas Python baru muncul di sini. `Counter` adalah penghitung otomatis: ia menghitung berapa kali tiap nilai muncul. `setdefault` adalah perintah kamus: "ambil isi kunci ini, dan bila belum ada, buat dulu dengan isi awal ini."

Baris	Tulisan	Artinya
29	`from collections import Counter`	Panggil `Counter` dari modul `collections` (Bab 8).
31-34	`petak = muat(...)` dan tiga baris di bawahnya	Muat empat layer.
37	`for nama, pred in (("intersect", 0), ("are within", 5)):`	Ulangi dua kali, sekali untuk tiap pasangan nama dan nomor predikat. Pasangan ditulis sebagai tuple di dalam tuple (Bab 5).
38	`j = processing.run("native:joinattributesbylocation", {...})["OUTPUT"]`	Gabungkan atribut petak dengan KPH menurut letak. `PREDICATE` berupa daftar nomor predikat. `JOIN_FIELDS` memilih kolom yang disalin. `METHOD: 0` berarti satu-ke-banyak.
39	`print("Join dengan %-10s: %d baris (petak ada 16)" % (...))`	Cetak jumlah baris hasil. Angka 16 pada teksnya diketik manual, bukan dihitung.
42	`petak_kph = ...joinattributesbylocation... "PREDICATE": [5] ... "METHOD": 1 ...`	Ulangi dengan predikat 5 (*are within*) dan `METHOD: 1`, yaitu satu-ke-satu. Hasilnya petak yang sudah bernama KPH.
43	`print("Petak per KPH:", dict(Counter(f["NAMA_KPH"] for f in ...)))`	Hitung berapa petak tiap KPH. `Counter` menerima daftar nama KPH, `dict` mengubah hasilnya menjadi kamus.
46	`hitung = processing.run("native:countpointsinpolygon", {...})["OUTPUT"]`	Hitung berapa titik panas jatuh di tiap petak. Hasilnya masuk kolom baru `JML_TITIK`.
47-48	`per_petak = {...}, print(...)`	Kamus kode petak ke jumlah titik, hanya untuk petak yang jumlahnya lebih dari 0. Cetak kamus dan jumlah totalnya.
49-52	`per_kph = Counter()` dan perulangan	Siapkan penghitung kosong. Untuk tiap petak, tambahkan jumlah titiknya ke KPH-nya (+=). Cetak hasilnya.

Baris	Tulisan	Artinya
55	`ringkas = processing.run("native:joinbylocationsummary", {...})["OUTPUT"]`	Gabung berdasarkan letak, tetapi hasilnya ringkasan, bukan salinan baris. `SUMMARIES: [0]` meminta ringkasan nomor 0, yaitu `count` (jumlah).
56	`kolom = ringkas.fields()[ringkas.fields().count() - 1].name()`	Ambil nama kolom paling akhir. Skrip tidak tahu nama kolom ringkasan, jadi memakai kolom terakhir. Dari hasil: `ID_Titik_count`.
58	`print("Total count:", int(sum((f[kolom] or 0) for f in ...)))`	Jumlahkan kolom ringkasan. `f[kolom] or 0` mengganti nilai kosong dengan 0.
61	`processing.run("native:selectbylocation", {... "PREDICATE": [0], "INTERSECT": titik, "METHOD": 0})`	Pilih petak yang beririsan dengan titik panas. Alat ini tidak membuat layer: ia menandai (menyeleksi) petak di layer `petak`. `METHOD: 0` berarti seleksi baru.
62-63	`print(... sorted(... petak.selectedFeatures()), petak.removeSelection()`	Cetak kode petak terpilih, urut abjad. Lalu hapus seleksi.
66	`dekat = processing.run("native:joinbynearest", {...})["OUTPUT"]`	Untuk tiap titik panas, cari jalan terdekat. `NEIGHBORS: 1` berarti hanya satu tetangga. Alat menambah kolom `distance` otomatis.
67	`baris = sorted((f["distance"], f["ID_Titik"], f["NAMA"]) for f in ...)`	Kumpulkan tiga hal tiap titik dalam satu tuple, lalu urutkan. Tuple diurutkan menurut isian pertama, yaitu jarak.
68	`print("Terdekat ke jalan: ...%s %.1f m (%s); terjauh: ..." % (...))`	`baris[0]` adalah yang terdekat, dan `baris[-1]` yang terjauh. Indeks `-1` berarti "yang terakhir".
71	`titik_petak = ...joinattributesbylocation... "INPUT": titik, "JOIN": petak ...`	Kali ini titik yang diberi atribut petak: kode petak dan jenis tanaman.
72	`print("Titik di petak P07:", sorted(... if f["KODE_PETAK"] == "P07"))`	Cetak titik yang jatuh di petak P07.
73-75	`per_jenis = {}` dan perulangan dengan `setdefault`	Siapkan kamus kosong. Tiap titik dimasukkan ke daftar milik jenisnya. Bila jenis itu belum punya daftar, `setdefault` membuatnya dulu.
76-77	`for jenis, daftar in sorted(per_jenis.items()) : print(...)`	Cetak tiap jenis, jumlah titik, dan daftar titiknya.

Hasil. Layar menampilkan (berdasarkan uji pada QGIS 4.0.2):

```
HASIL
Join dengan intersect : 30 baris (petak ada 16)
Join dengan are within: 16 baris (petak ada 16)
Petak per KPH: {'KPH Alpha': 4, 'KPH Beta': 4, 'KPH Gamma': 8}
Titik per petak: {'P05': 1, 'P06': 1, 'P03': 1, 'P07': 3, 'P08': 1, 'P10': 1, 'P11': 1, 'P12': 1, 'P13': 1,
'P14': 1} jumlah = 12
Titik per KPH: {'KPH Alpha': 2, 'KPH Beta': 5, 'KPH Gamma': 5}
Kolom hasil ringkasan: ID_Titik_count
Total count: 12
Petak terpilih: ['P03', 'P05', 'P06', 'P07', 'P08', 'P10', 'P11', 'P12', 'P13', 'P14']
Terdekat ke jalan: HS-002 12.1 m (Jalan Inspeksi); terjauh: HS-001 225.2 m
Titik di petak P07: ['HS-004', 'HS-006', 'HS-011']
Titik pada Jati   : 4 ['HS-001', 'HS-003', 'HS-005', 'HS-010']
Titik pada Mahoni : 2 ['HS-002', 'HS-008']
Titik pada Pinus  : 2 ['HS-009', 'HS-012']
Titik pada Sengon : 4 ['HS-004', 'HS-006', 'HS-007', 'HS-011']
```

Hal terpenting pada hasil ini: join dengan `intersect` menghasilkan **30** baris, padahal petak hanya 16. Selisihnya 14 baris. Itu berasal dari petak di tepi batas KPH yang menyentuh KPH tetangganya, sehingga muncul baris tambahan. Dengan `are within` jumlahnya tepat 16.

Semua jumlah titik juga konsisten: 12 titik, baik dihitung per petak, per KPH, maupun lewat ringkasan.

Jebakan nomor predikat: dua alat, dua daftar

Angka predikat tidak sama di semua alat. Skrip ini memakai `5` untuk *are within* pada alat join (baris 38 dan 71), tetapi `0` untuk *intersect* pada alat seleksi (baris 61). Itu benar.

Namun bila Anda memakai angka 5 di alat seleksi, artinya lain. Saya memeriksa daftarnya dengan membaca definisi alat di QGIS 4.0.2:

Predikat	`joinattributesbylocation` dan `joinbylocationsummary`	`selectbylocation`
intersect	0	0
contain	1	1
disjoint	tidak ada	2
equal	2	3
touch	3	4
overlap	4	5
are within	5	6
cross	6	7

Saya juga mengujinya langsung pada data ini. Alat seleksi dengan angka 0 memilih 16 petak, angka 5 memilih 0 petak (karena 5 di alat itu berarti *overlap*), dan angka 6 memilih 16 petak. Alat join dengan angka 5 mencocokkan 16 petak, sedangkan angka 6 mencocokkan 0 (karena 6 di alat itu berarti *cross*). Kesimpulannya: **selalu periksa daftar predikat tiap alat**, jangan menyalin angka dari alat lain.

Bagian yang boleh diubah.

- **Baris 38 dan 42, predikat dan metode.** Ganti angka predikat sesuai tabel. Ganti `METHOD` 0 atau 1 sesuai kebutuhan.
- **Baris 46, 55, 61, 66, 71, layer pasangan.** Ganti layer yang dihubungkan.
- Baris 66, `NEIGHBORS`. Ganti 1 menjadi 2 untuk mencari dua jalan terdekat. Uji saya: hasilnya 24 baris untuk 12 titik, karena data jalan hanya berisi dua ruas.
- **Baris 72, kode petak.** Ganti `"P07"` dengan petak lain.
- Baris 55, `SUMMARIES`. Ganti 0 dengan nomor ringkasan lain. Nomor 5 adalah `sum`, dan nomor 6 `mean`.

Jebakan.

- **Satu-ke-banyak menggandakan baris.** Join `intersect` berbaris 30, bukan 16. Baris 37-39 memperlihatkannya sebagai pelajaran. Untuk "satu petak, satu KPH", pakai `are within` dan `METHOD: 1`.
- **Nama kolom ringkasan ditebak.** Baris 56 mengambil kolom paling akhir, bukan menyebut nama pasti. Hasil uji: `ID_Titik_count`. Bila Anda mengubah `JOIN_FIELDS` atau `SUMMARIES`, kolom terakhir bisa berbeda.
- **Angka 16 di teks adalah angka mati.** Baris 39 menulis "petak ada 16" sebagai teks tetap. Pada data lain, tulisan ini salah.
- **Seleksi menandai layer asli.** Baris 61 menyeleksi di layer `petak`. Baris 63 menghapus seleksinya. Bila skrip berhenti sebelum baris 63, petak tetap terpilih di proyek.
- **Urutan kamus.** Urutan isi kamus `Titik per petak` tidak urut abjad. Pakai `sorted` bila butuh urutan.

Latihan variasi. Cetak titik panas yang jatuh di petak P05, bukan P07.

Contoh jawaban. Ganti `P07` menjadi `P05` pada baris 72:

```
PYTHON
print("Titik di petak P05:", sorted(f["ID_Titik"] for f in titik_petak.getFeatures() if f["KODE_PETAK"] ==
"P05"))
```

Layar menampilkan `Titik di petak P05: ['HS-008']`.

> **Pola yang bisa dipakai ulang:** Kerangka join letak: **pilih predikat dan metode, panggil alat, ringkas dengan Counter atau kamus**. Untuk soal lain, ganti empat hal: **layer masukan dan layer pasangan**, **nomor predikat (sesuai alatnya)**, **kolom yang disalin** (`JOIN_FIELDS`), dan **metode** (0 satu-ke-banyak, 1 satu-ke-satu).

Skrip 18.6: Merangkai alat menjadi model

Berkas `6_1_buat_model.py`. Skrip ini tidak memuat data. Ia membuat satu berkas model. Skrip ini tidak memakai blok pembuka.

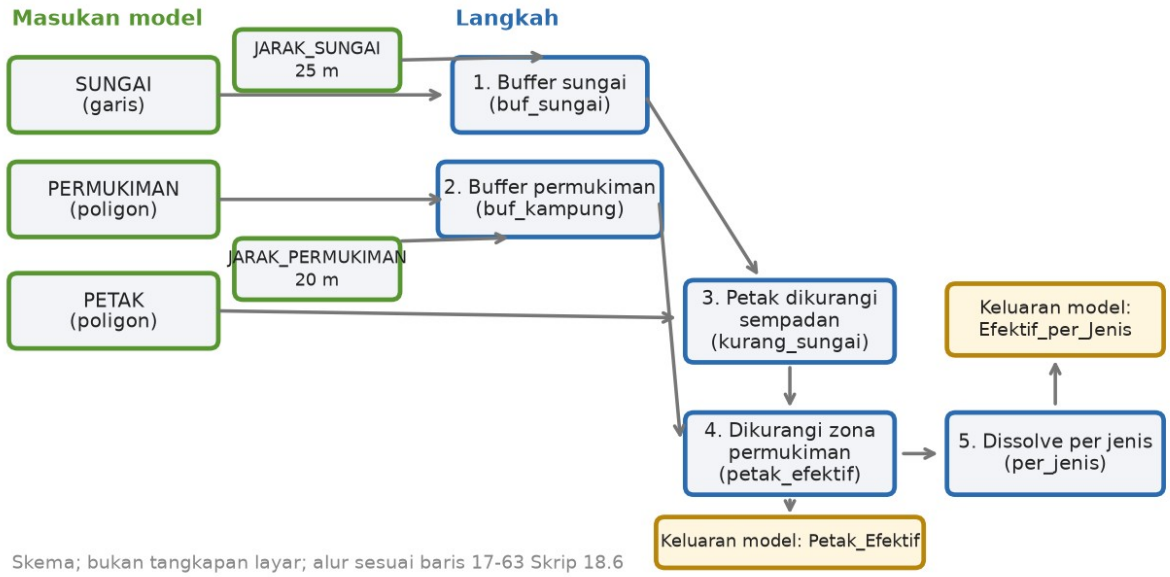
Tujuan. Membuat model Processing bernama `areal_efektif` lewat kode: petak dikurangi sempadan sungai, lalu dikurangi zona permukiman, lalu dilebur per jenis. Model disimpan ke berkas `areal_efektif.model3`.

Bahan. Tidak ada data masukan. Skrip menulis satu berkas ke folder `HASIL`.

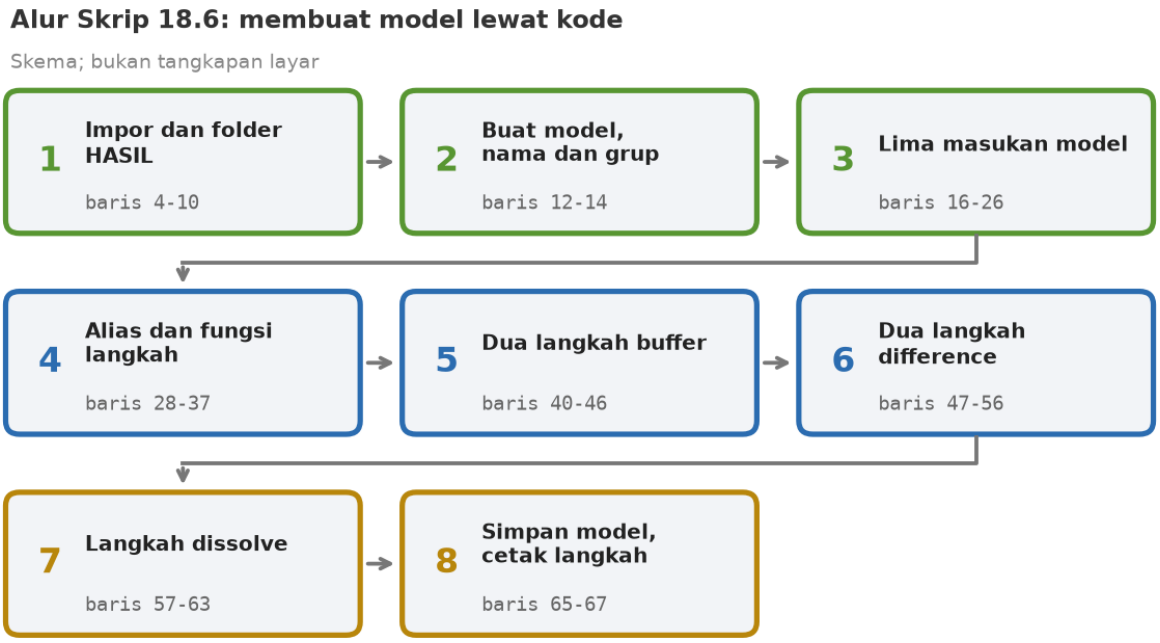
Model adalah resep tertulis dari rangkaian alat. Bayangkan resep masakan: bahan (masukan), langkah-langkah (alat), dan hidangan jadi (keluaran).

Sekali ditulis, resep bisa dimasak berkali-kali dengan jumlah bahan berbeda. Di QGIS, resep ini bisa disusun di Model Designer lewat klik. Skrip ini menyusunnya lewat kode.

Ilustrasi 18.10: Rantai langkah model areal efektif



Ilustrasi 18.11: Alur skrip 18.6



Skrip utuh:

```
PYTHON
# -*- coding: utf-8 -*-
# Penulis: Badar Mubarok Yogaswara. Membuat model "Areal Efektif" lewat Python (setara dengan menyusunnya di
Model Designer).
# Jalankan di Python Console QGIS. Hasil: berkas areal_efektif.model3 di folder HASIL.
from qgis.core import (QgsProcessingModelAlgorithm, QgsProcessingModelParameter,
QgsProcessingModelChildAlgorithm,
                       QgsProcessingModelChildParameterSource, QgsProcessingModelOutput,
                       QgsProcessingParameterFeatureSource, QgsProcessingParameterNumber, QgsProcessing)

HASIL = r"C:\latihan\hasil"      # ubah sesuai komputer Anda; folder dibuat bila belum ada
import os
os.makedirs(HASIL, exist_ok=True)

m = QgsProcessingModelAlgorithm()
m.setName("areal_efektif")
m.setGroup("Latihan I1")

# --- Masukan model (yang diisi pengguna saat menjalankan)
p = QgsProcessingParameterFeatureSource("PETAK", "Petak", [QgsProcessing.TypeVectorPolygon])
m.addModelParameter(p, QgsProcessingModelParameter("PETAK"))
p = QgsProcessingParameterFeatureSource("SUNGAI", "Sungai", [QgsProcessing.TypeVectorLine])
m.addModelParameter(p, QgsProcessingModelParameter("SUNGAI"))
p = QgsProcessingParameterFeatureSource("PERMUKIMAN", "Permukiman", [QgsProcessing.TypeVectorPolygon])
m.addModelParameter(p, QgsProcessingModelParameter("PERMUKIMAN"))
p = QgsProcessingParameterNumber("JARAK_SUNGAI", "Jarak aman sungai (m)", QgsProcessingParameterNumber.Double,
25)
m.addModelParameter(p, QgsProcessingModelParameter("JARAK_SUNGAI"))
p = QgsProcessingParameterNumber("JARAK_PERMUKIMAN", "Jarak aman permukiman (m)",
QgsProcessingParameterNumber.Double, 20)
m.addModelParameter(p, QgsProcessingModelParameter("JARAK_PERMUKIMAN"))

MP = QgsProcessingModelChildParameterSource


def langkah(id_, alg, deskripsi, params):
    c = QgsProcessingModelChildAlgorithm(alg)
    c.setChildId(id_)
    c.setDescription(deskripsi)
    for k, v in params.items():
        c.addParameterSources(k, [v])
    return c


# --- Langkah 1 dan 2: buffer sungai dan permukiman (dengan dissolve)
m.addChildAlgorithm(langkah("buf_sungai", "native:buffer", "Buffer sungai", {
    "INPUT": MP.fromModelParameter("SUNGAI"), "DISTANCE": MP.fromModelParameter("JARAK_SUNGAI"),
    "DISSOLVE": MP.fromStaticValue(True)}))
m.addChildAlgorithm(langkah("buf_kampung", "native:buffer", "Buffer permukiman", {
    "INPUT": MP.fromModelParameter("PERMUKIMAN"), "DISTANCE": MP.fromModelParameter("JARAK_PERMUKIMAN"),
    "DISSOLVE": MP.fromStaticValue(True)}))
# --- Langkah 3: petak dikurangi sempadan sungai
m.addChildAlgorithm(langkah("kurang_sungai", "native:difference", "Petak dikurangi sempadan", {
    "INPUT": MP.fromModelParameter("PETAK"), "OVERLAY": MP.fromChildOutput("buf_sungai", "OUTPUT")}))
# --- Langkah 4: hasilnya dikurangi zona permukiman (keluaran model: Petak_Efektif)
c4 = langkah("petak_efektif", "native:difference", "Dikurangi zona permukiman", {
    "INPUT": MP.fromChildOutput("kurang_sungai", "OUTPUT"), "OVERLAY": MP.fromChildOutput("buf_kampung",
"OUTPUT")})
c4.setModelOutputs({"Petak_Efektif": QgsProcessingModelOutput("Petak_Efektif", "Petak_Efektif")})
c4.modelOutput("Petak_Efektif").setChildId("petak_efektif")
c4.modelOutput("Petak_Efektif").setChildOutputName("OUTPUT")
m.addChildAlgorithm(c4)
# --- Langkah 5: dissolve per jenis (keluaran model: Efektif_per_Jenis)
c5 = langkah("per_jenis", "native:dissolve", "Dissolve per jenis", {
```

```
    "INPUT": MP.fromChildOutput("petak_efektif", "OUTPUT"), "FIELD": MP.fromStaticValue(["JENIS"])})
c5.setModelOutputs({"Efektif_per_Jenis": QgsProcessingModelOutput("Efektif_per_Jenis", "Efektif_per_Jenis")})
c5.modelOutput("Efektif_per_Jenis").setChildId("per_jenis")
c5.modelOutput("Efektif_per_Jenis").setChildOutputName("OUTPUT")
m.addChildAlgorithm(c5)

berkas = os.path.join(HASIL, "areal_efektif.model3")
print("Simpan model:", m.toFile(berkas), berkas)
print("Langkah:", [c.description() for c in m.childAlgorithms().values()])
```

Penjelasan baris demi baris. Skrip ini punya banyak perkakas baru yang namanya panjang. Anda tidak perlu menghafalnya. Cukup pahami pola: **masukan dulu, lalu langkah, lalu simpan**.

Baris	Tulisan	Artinya
1-3	komentar	Judul, penulis, dan cara menjalankan. Komentar baris 3 menyebut hasilnya berkas `areal_efektif.model3` di folder `HASIL`.
4-6	`from qgis.core import (...)`	Panggil delapan perkakas khusus model.
8	`HASIL = r"C:\latihan\hasil"`	Folder tujuan. Komentar di ujung baris menyebut folder dibuat bila belum ada, sesuai baris 10.
9-10	`import os, os.makedirs(HASIL, exist_ok=True)`	Pengaman folder: buat folder `HASIL` bila belum ada (Bab 8). `exist_ok=True` berarti tidak masalah bila folder sudah ada. Tanpa baris ini, folder yang belum ada tidak dibuat sendiri.
12-14	`m = QgsProcessingModelAlgorithm(), setName, setGroup`	Buat model kosong di kotak `m`. Beri nama `areal_efektif` dan grup `Latihan I1`.
17-18	`p = QgsProcessingParameterFeatureSource("PETAK", "Petak", [...])` lalu `m.addModelParameter(p, ...)`	Siapkan satu masukan model: kode `PETAK`, label `Petak`, dan jenis geometri yang diizinkan (poligon). Lalu daftarkan ke model.
19-22	dua pasang baris serupa	Masukan `SUNGAI` (garis) dan `PERMUKIMAN` (poligon). Kotak `p` dipakai ulang tiap kali.
23-26	`QgsProcessingParameterNumber("JARAK_SUNGAI", ..., 25)`	Dua masukan angka: jarak sungai (bawaan 25) dan jarak permukiman (bawaan 20). Nilai bawaan muncul sebagai isian awal saat model dijalankan.
28	`MP = QgsProcessingModelChildParameterSource`	Alias: nama pendek `MP` untuk perkakas yang namanya panjang.
31-37	`def langkah(id_, alg, deskripsi, params):`	Fungsi pembuat satu langkah. Tiga isian pertama: kode langkah, nama alat, dan deskripsi. Isian `params` adalah kamus pengaturan. Nama `id_` memakai garis bawah agar tidak bentrok dengan fungsi bawaan `id`.

Baris	Tulisan	Artinya
35-36	`for k, v in params.items(): c.addParameterSources(k, [v])`	Untuk tiap pasangan pengaturan, pasang sumbernya ke langkah.
41-46	`m.addChildAlgorithm(langkah("buf_sungai", "native:buffer", ...))`	Tambahkan dua langkah buffer. Sumber `MP.fromModelParameter("SUNGAI")` artinya "ambil dari masukan model". Sumber `MP.fromStaticValue(True)` artinya "pakai nilai tetap ini".
48-49	`langkah("kurang_sungai", "native:difference", {...})`	Langkah difference: petak dikurangi hasil buffer sungai. `MP.fromChildOutput("buf_sungai", "OUTPUT")` artinya "ambil keluaran langkah `buf_sungai`".
51-56	`c4 = langkah("petak_efektif", ...)` dan baris `c4.`	Langkah difference kedua. Lalu tandai keluarannya sebagai keluaran model bernama `Petak_Efektif` (tiga baris `c4.setModelOutputs`, `modelOutput(...)`), dan tambahkan ke model.
58-63	`c5 = langkah("per_jenis", "native:dissolve", {...})` dan baris `c5.`	Langkah dissolve per `JENIS`. Keluarannya ditandai sebagai keluaran model `Efektif_per_Jenis`.
65	`berkas = os.path.join(HASIL, "areal_efektif.model3")`	Susun nama berkas tujuan.
66	`print("Simpan model:", m.toFile(berkas), berkas)`	Simpan model ke berkas. `toFile` mengembalikan `True` bila berhasil.
67	`print("Langkah:", [c.description() for c in m.childAlgorithms().values()])`	Cetak deskripsi semua langkah.

Hasil. Layar menampilkan (berdasarkan uji pada QGIS 4.0.2):

```
HASIL
Simpan model: True ...\hasil\areal_efektif.model3
Langkah: ['Buffer permukiman', 'Buffer sungai', 'Petak dikurangi sempadan', 'Dissolve per jenis', 'Dikurangi zona permukiman']
```

Jalur di baris pertama mengikuti isi `HASIL` di komputer Anda. Di sini saya menyingkatnya. Berkas model yang dihasilkan berukuran sekitar 26 kilobyte.

Perhatikan urutan di baris kedua: bukan urutan penulisan di skrip. Pada hasil uji, daftar itu tampil menurut urutan abjad kode langkah (`buf_kampung`, `buf_sungai`, `kurang_sungai`, `per_jenis`, `petak_efektif`). Urutan itu tidak memengaruhi cara model bekerja, karena tiap langkah menyebut sumbernya sendiri.

Bagian yang boleh diubah.

- Baris 8, `HASIL`. Ganti dengan folder Anda.

- **Baris 13-14, nama dan grup model.** Ganti sesuka Anda.
- **Baris 23 dan 25, angka bawaan.** Ganti 25 dan 20 untuk mengubah isian awal model.
- Baris 59, `["JENIS"]`. Ganti dengan kolom pengelompok lain.

Jebakan.

- **Folder dibuat otomatis.** Baris 10 membuat folder `HASIL` bila belum ada. Saya mengujinya dengan folder yang belum ada: folder dibuat, dan berkas model tersimpan. Akibatnya, salah ketik pada `HASIL` tidak menimbulkan galat. Skrip membuat folder baru di tempat yang salah. Periksa jalur yang dicetak di hasil.
- **Nama keluaran model.** Nama yang ditulis di skrip ini `Petak_Efektif` dan `Efektif_per_Jenis`. Skrip 18.7 memakai nama huruf kecil (`petak_efektif`). Penjelasannya ada di Skrip 18.7.
- **Belum teruji di antarmuka.** Berkas model ini berhasil dibaca dan dijalankan oleh Skrip 18.7. Saya belum membukanya di Model Designer, jadi tampilan kotak-kotaknya belum saya lihat. [CEK]
- `QgsProcessingParameterNumber.Double`. Penulisan ini berjalan baik di QGIS 4.0.2. Pada versi lain nama pilihannya bisa berbeda. [CEK]
- **Menimpa model lama.** Menjalankan ulang skrip menulis berkas yang sama. Salin dulu bila Anda sudah mengubah model di Model Designer.

Latihan variasi. Ubah jarak bawaan model menjadi 50 meter untuk sungai dan 30 meter untuk permukiman.

Contoh jawaban. Ganti baris 23 dan 25:

```python
p = QgsProcessingParameterNumber("JARAK_SUNGAI", "Jarak aman sungai (m)", QgsProcessingParameterNumber.Double, 50)
```

```python
p = QgsProcessingParameterNumber("JARAK_PERMUKIMAN", "Jarak aman permukiman (m)", QgsProcessingParameterNumber.Double, 30)
```

Saya membuka kembali model hasilnya, lalu mencetak nilai bawaannya. Layar menampilkan `bawaan: 50 30`.

> **Pola yang bisa dipakai ulang:** Kerangka model: **buat model, tambahkan masukan, tambahkan langkah satu per satu dengan sumbernya, tandai keluaran, simpan**. Untuk resep lain, ganti tiga hal: **daftar masukan** (baris 17-26), **langkah dan alatnya** (baris 41-63), dan **nama berkas** (baris 65). Pola `langkah(...)` tidak perlu disentuh.

Skrip 18.7: Laporan areal efektif dalam satu skrip

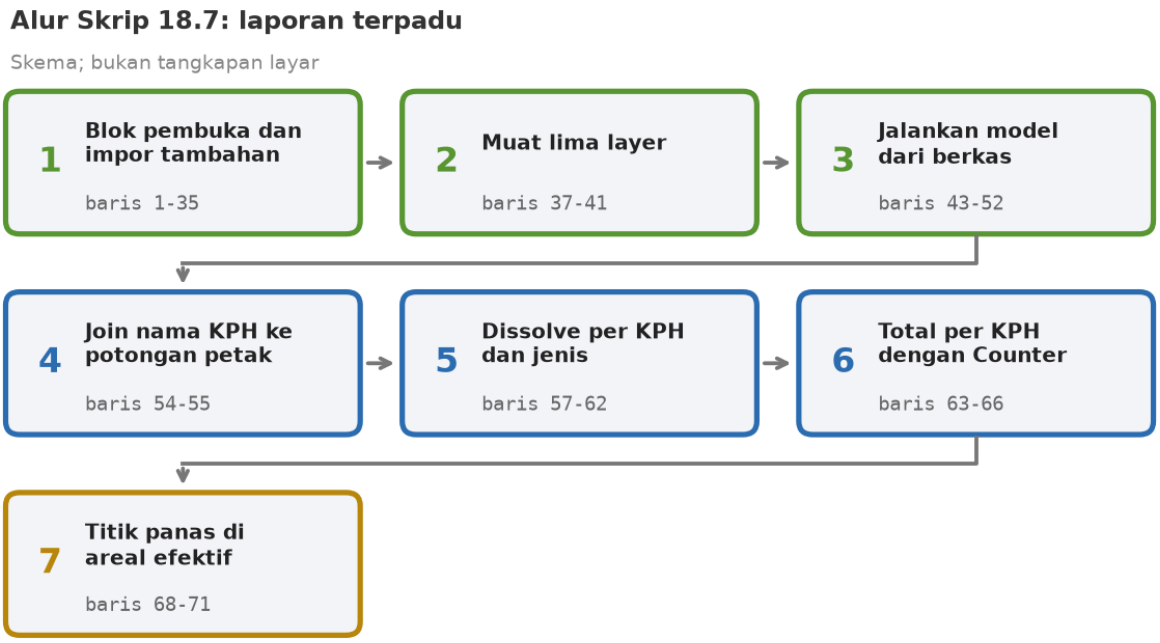
Berkas `7_1_laporan_terpadu.py`. Skrip ini memuat blok pembuka (baris 1-28), lalu menjalankan model dari Skrip 18.6.

Tujuan. Menjalankan model areal efektif dari berkas, memberi nama KPH pada hasilnya, merekap per KPH dan jenis, menyimpan rekap ke GeoPackage, dan menghitung titik panas yang jatuh di areal efektif.

Bahan. Lima layer: `Petak`, `Sungai`, `Permukiman` (`paket-i1`), `Batas_KPH`, dan `Hotspot_Nasional` (`paket-data`). Jalankan Skrip 18.6 lebih dulu supaya berkas `areal_efektif.model3` ada di folder `HASIL`. Skrip ini menghasilkan tiga berkas GeoPackage di folder itu.

Skrip ini menjawab pertanyaan Kepala Seksi di awal bab. Ia menyatukan semua hal sebelumnya: memuat, menjalankan resep, menghubungkan lokasi, merekap, dan menyeleksi.

Ilustrasi 18.12: Alur skrip 18.7



Skrip utuh:

```python
# -*- coding: utf-8 -*-
# Penulis: Badar Mubarok Yogaswara. Dijalankan di Python Console QGIS. Data sintetis, bukan kondisi lapangan nyata.
from qgis.core import QgsVectorLayer, QgsProject, QgsProperty
import processing

PAKET = r"C:\latihan\paket-i1"      # ubah sesuai letak folder di komputer Anda
DATA = r"C:\latihan\paket-data"


def muat(berkas, lapisan, nama):
    """Memuat satu layer GeoPackage dan menambahkannya ke proyek."""
    lyr = QgsVectorLayer("%s|layername=%s" % (berkas, lapisan), nama, "ogr")
    if not lyr.isValid():
```

```
        raise RuntimeError("Layer tidak terbaca: " + berkas)
    QgsProject.instance().addMapLayer(lyr)
    return lyr


def luas_ha(lyr):
    """Jumlah luas semua poligon dalam hektare (CRS harus UTM, satuan meter)."""
    return sum(f.geometry().area() for f in lyr.getFeatures()) / 10000.0


def buf(lyr, jarak):
    """Buffer dengan dissolve, hasil sementara."""
    return processing.run("native:buffer", {"INPUT": lyr, "DISTANCE": jarak, "DISSOLVE": True, "OUTPUT":
"memory:"})["OUTPUT"]


import os
from collections import Counter
from qgis.core import QgsProcessingModelAlgorithm
from qgis.core import QgsProcessingModelAlgorithm

HASIL = r"C:\latihan\hasil"      # folder keluaran; dibuat bila belum ada
os.makedirs(HASIL, exist_ok=True)

petak = muat(PAKET + r"\Petak.gpkg", "Petak", "Petak")
sungai = muat(PAKET + r"\Sungai.gpkg", "Sungai", "Sungai")
kampung = muat(PAKET + r"\Permukiman.gpkg", "Permukiman", "Permukiman")
kph = muat(DATA + r"\Batas_KPH.gpkg", "Batas_KPH", "Batas_KPH")
titik = muat(DATA + r"\Hotspot_Nasional.gpkg", "Hotspot_Nasional", "Hotspot_Nasional")

# 1. Jalankan model dari Bab 6 (berkas .model3 hasil skrip 6_1 atau hasil susunan Anda sendiri)
model = QgsProcessingModelAlgorithm()
assert model.fromFile(HASIL + r"\areal_efektif.model3"), "Model tidak terbaca; jalankan skrip 6_1 lebih dulu"
hasil = processing.run(model, {
    "PETAK": petak, "SUNGAI": sungai, "PERMUKIMAN": kampung,
    "JARAK_SUNGAI": 25, "JARAK_PERMUKIMAN": 20,
    "petak_efektif": HASIL + r"\Petak_Efektif.gpkg",
    "efektif_per_jenis": HASIL + r"\Efektif_per_Jenis.gpkg"})
efektif = QgsVectorLayer(hasil["petak_efektif"], "Petak_Efektif", "ogr")
print("Areal efektif: %.2f ha dari %.2f ha" % (luas_ha(efektif), luas_ha(petak)))

# 2. Spatial join: setiap potongan petak diberi nama KPH
efektif_kph = processing.run("native:joinattributesbylocation", {"INPUT": efektif, "PREDICATE": [5], "JOIN":
kph, "JOIN_FIELDS": ["NAMA_KPH"], "METHOD": 1, "OUTPUT": "memory:"})["OUTPUT"]

# 3. Dissolve per KPH dan jenis, simpan ke GeoPackage
rekap = processing.run("native:dissolve", {"INPUT": efektif_kph, "FIELD": ["NAMA_KPH", "JENIS"], "OUTPUT":
HASIL + r"\Rekap_KPH_Jenis.gpkg"})["OUTPUT"]
rekap = QgsVectorLayer(rekap, "Rekap_KPH_Jenis", "ogr")
print("%-10s %-7s %s" % ("KPH", "Jenis", "Efektif (ha)"))
for f in sorted(rekap.getFeatures(), key=lambda f: (f["NAMA_KPH"], f["JENIS"])):
    print("%-10s %-7s %.2f" % (f["NAMA_KPH"], f["JENIS"], f.geometry().area() / 10000.0))
per_kph = Counter()
for f in efektif_kph.getFeatures():
    per_kph[f["NAMA_KPH"]] += f.geometry().area() / 10000.0
print("Total per KPH (ha):", {k: round(v, 2) for k, v in sorted(per_kph.items())})

# 4. Select by location: titik panas yang jatuh di areal efektif
processing.run("native:selectbylocation", {"INPUT": titik, "PREDICATE": [0], "INTERSECT": efektif, "METHOD":
0})
print("Titik panas di areal efektif: %d dari %d" % (titik.selectedFeatureCount(), titik.featureCount()))
titik.removeSelection()
```

Penjelasan baris demi baris. Baris 1-28 adalah blok pembuka.

Baris	Tulisan	Artinya
29-32	`import os`, `from collections import Counter`, dua baris `from qgis.core import QgsProcessingModelAlgorithm`	Panggil perkakas tambahan. Baris 31 dan 32 sama persis. Pemanggilan ganda itu tidak merusak apa pun.
34-35	`HASIL = ...`, `os.makedirs(HASIL, exist_ok=True)`	Folder keluaran, dibuat bila belum ada. Komentarnya sesuai dengan kodenya.
37-41	`petak = muat(...)` dan empat baris serupa	Muat lima layer.
44-45	`model = QgsProcessingModelAlgorithm()`, `assert model.fromFile(...), "Model tidak terbaca; ..."`	Buat model kosong, lalu isi dari berkas `areal_efektif.model3`. `fromFile` menjawab benar atau salah. `assert` memastikan jawabannya benar (Bab 10). Bila salah, skrip berhenti dengan pesan di sesudah koma, yang menyuruh Anda menjalankan Skrip 18.6 lebih dulu.
46-50	`hasil = processing.run(model, {...})`	Jalankan model. Isian pertama adalah model itu sendiri, bukan nama alat. Kamusnya memuat masukan model (`PETAK`, `SUNGAI`, `PERMUKIMAN`, dua jarak) dan dua keluaran: `petak_efektif` dan `efektif_per_jenis`, yang diisi alamat berkas GeoPackage. Keluaran langsung ditulis ke berkas, bukan ke memori.
51	`efektif = QgsVectorLayer(hasil["petak_efektif"], "Petak_Efektif", "ogr")`	Buka berkas keluaran sebagai layer. Kunci `hasil` berupa alamat berkas.
52	`print("Areal efektif: %.2f ha dari %.2f ha" % (...))`	Cetak luas areal efektif dan luas seluruh petak.
55	`efektif_kph = ...joinattributesbylocation... "PREDICATE": [5] ... "METHOD": 1 ...`	Beri tiap potongan petak nama KPH-nya, dengan *are within* dan satu-ke-satu, seperti di Skrip 18.5.
58-59	`rekap = processing.run("native:dissolve", {... "FIELD": ["NAMA_KPH", "JENIS"], "OUTPUT": HASIL + ...}) ["OUTPUT"]`	Lebur per pasangan KPH dan jenis. Keluarannya ditulis ke `Rekap_KPH_Jenis.gpkg`. Alat mengembalikan alamat berkas, lalu baris 59 membukanya sebagai layer.
60-62	`print(...)` dan perulangan	Cetak judul kolom, lalu satu baris per pasangan, urut menurut KPH lalu jenis (`key=lambda f: (f["NAMA_KPH"], f["JENIS"])`, kunci berupa tuple).

Baris	Tulisan	Artinya
63-65	`per_kph = Counter()` dan perulangan	Jumlahkan luas (dalam hektare) tiap KPH dengan `+=`.
66	`print("Total per KPH (ha):", {k: round(v, 2) for k, v in sorted(per_kph.items())})`	Cetak total per KPH, dibulatkan dua desimal, memakai kamus comprehension.
69	`processing.run("native:selectbylocation", {... "PREDICATE": [0], "INTERSECT": efektif ...})`	Pilih titik panas yang beririsan dengan areal efektif. Perhatikan: di alat seleksi, nomor 0 berarti *intersect*.
70-71	`print(... titik.selectedFeatureCount() ...), titik.removeSelection()`	Cetak jumlah titik terpilih dari seluruh titik. Lalu hapus seleksinya.

Hasil. Layar menampilkan (berdasarkan uji pada QGIS 4.0.2):

```
HASIL
Areal efektif: 12.39 ha dari 16.00 ha
KPH        Jenis   Efektif (ha)
KPH Alpha  Jati    1.00
KPH Alpha  Mahoni  2.00
KPH Alpha  Pinus   1.00
KPH Beta   Jati    0.80
KPH Beta   Pinus   0.80
KPH Beta   Sengon  1.57
KPH Gamma  Jati    2.85
KPH Gamma  Mahoni  1.63
KPH Gamma  Pinus   0.28
KPH Gamma  Sengon  0.46
Total per KPH (ha): {'KPH Alpha': 4.0, 'KPH Beta': 3.17, 'KPH Gamma': 5.22}
Titik panas di areal efektif: 7 dari 12
```

Inilah jawaban untuk Kepala Seksi. Dari 16,00 hektare, 12,39 hektare berada di luar jarak aman. Jumlah ketiga KPH, 4,00 ditambah 3,17 ditambah 5,22, sama dengan 12,39. Sebanyak 7 dari 12 titik panas jatuh di areal efektif.

Bagian yang boleh diubah.

- Baris 34, `HASIL`. Ganti dengan folder Anda. Folder harus sama dengan yang dipakai Skrip 18.6.
- **Baris 48, jarak.** Angka `25` dan `20` adalah jarak aman. Ganti untuk skenario lain.
- **Baris 58, kolom pengelompok.** Ganti `["NAMA_KPH", "JENIS"]` dengan kolom lain.
- **Baris 49-50, nama berkas keluaran.** Ganti nama berkas GeoPackage.

Jebakan.

- **Pengaman berkas model (baris 45).** Bila `areal_efektif.model3` tidak ada di `HASIL`, skrip berhenti di baris 45 dengan `AssertionError: Model tidak terbaca; jalankan skrip 6_1 lebih dulu`. Pesan menyebut skrip 6_1, yaitu Skrip 18.6 di bab ini. Pada uji saya, folder `HASIL`

kosong. Tanpa `assert`, kegagalan tidak terlihat di baris itu. Skrip baru berhenti di baris 51 dengan `KeyError: 'petak_efektif'`, yang tidak menyebut berkas model. Bila pesan pengaman muncul, jalankan Skrip 18.6 lebih dulu.

- **Nama keluaran model berubah huruf.** Skrip 18.6 menamai keluaran `Petak_Efektif`. Hasil `processing.run` memakai kunci `petak_efektif`. Hasil uji saya: kunci yang ada adalah `CHILD_INPUTS`, `CHILD_RESULTS`, `efektif_per_jenis`, dan `petak_efektif`. Dugaan saya, QGIS membuat nama aman dari nama keluaran, yaitu huruf kecil dan garis bawah. [kemungkinan] Bila Anda mengganti nama keluaran di model, kunci di baris 49-51 juga berubah.
- **Potongan yang melintasi batas KPH.** Baris 55 memakai *are within*. Potongan petak yang melintasi batas dua KPH tidak berada seluruhnya di satu KPH. Potongan seperti itu kemungkinan tidak mendapat nama KPH. Pada data ini hal itu tidak terjadi: dari 16 potongan, 0 yang tanpa nama KPH, dan jumlah per KPH sama dengan total areal efektif. Pada data lain, periksa dulu. [kemungkinan]
- **Berkas lama ditimpa.** Tiga berkas GeoPackage di `HASIL` ditulis ulang tiap kali.
- **Pemanggilan ganda.** Baris 31-32 mengimpor hal yang sama dua kali. Bukan galat, hanya kelebihan.
- **Komentar menyebut "Bab 6".** Baris 43 merujuk ke bab di Buku I1. Di buku ini, modelnya dibuat oleh Skrip 18.6.

Latihan variasi. Jalankan skenario "bagaimana bila" dengan jarak aman sungai 50 meter dan permukiman 30 meter.

Contoh jawaban. Ganti baris 48:

```python
    "JARAK_SUNGAI": 50, "JARAK_PERMUKIMAN": 30,
```

Layar menampilkan `Areal efektif: 9.69 ha dari 16.00 ha`, total per KPH `{'KPH Alpha': 3.71, 'KPH Beta': 2.34, 'KPH Gamma': 3.64}`, dan `Titik panas di areal efektif: 2 dari 12`. Baris `KPH Gamma  Pinus  0.00` juga muncul: areal Pinus di KPH Gamma menyusut sampai 0,00 hektare setelah dibulatkan.

> **Pola yang bisa dipakai ulang:** Kerangka laporan: **jalankan model, hubungkan ke wilayah, dissolve per kelompok, cetak tabel, seleksi titik**. Untuk laporan lain, ganti empat hal: **model dan masukannya** (baris 44-50), **layer wilayah** (baris 55), **kolom pengelompok** (baris 58), dan **layer titik** (baris 69). Satu model bisa dijalankan dengan jarak berbeda hanya dengan mengubah dua angka.

Cek paham

1. Mengapa join `intersect` pada Skrip 18.5 menghasilkan 30 baris, padahal petak hanya 16?
2. Angka 5 berarti apa pada alat `joinattributesbylocation`, dan apa pada `selectbylocation`?
3. Apa fungsi `["OUTPUT"]` di ujung `processing.run(...)`?

Jawaban.

1. Petak di tepi batas KPH menyentuh KPH tetangganya. Dengan `intersect` dan satu-ke-banyak, petak seperti itu cocok dengan dua KPH, sehingga muncul dua baris.
2. Di `joinattributesbylocation`, 5 berarti *are within*. Di `selectbylocation`, 5 berarti *overlap*, sedangkan *are within* bernomor 6.
3. `processing.run` mengembalikan kamus hasil. `["OUTPUT"]` mengambil layer hasil dari kamus itu.

Kesalahan umum

- **Membiarkan jalur contoh.** `PAKET`, `DATA`, dan `HASIL` menunjuk folder di komputer penulis. Perbaikan: ganti ketiganya, dan ganti garis miring terbalik bila Anda memakai macOS atau Ubuntu.
- **Menyalin nomor predikat dari alat lain.** Perbaikan: buka daftar predikat tiap alat, atau gunakan tabel di Skrip 18.5.
- **Tidak membaca hasil, hanya mencari galat.** Beberapa salah ketik tidak menimbulkan galat, tetapi mengubah hasil (kolom `ORDEE` pada rumus buffer, atau nama kolom yang tidak dijaga `assert`). Perbaikan: bandingkan angka hasil dengan perkiraan Anda, dan periksa nama kolom dengan `assert` seperti di baris 38 Skrip 18.4.
- **Menghitung luas pada layer berkoordinat derajat.** Perbaikan: pastikan CRS dalam meter sebelum memakai `luas_ha`.
- **Menjalankan Skrip 18.7 sebelum 18.6.** Perbaikan: jalankan 18.6 dulu, atau pastikan berkas model ada.

Ringkasan dan latihan

- Geoprocessing mengolah layer lewat alat. Di skrip, `processing.run("nama_alat", {parameter})["OUTPUT"]` memanggil satu alat.
- Hasil satu alat boleh langsung menjadi masukan alat berikutnya, sehingga rantai analisis bisa ditulis dalam beberapa baris.
- Buffer membuat zona aman, overlay (Intersect, Difference, Union) menggunting tumpukan, dissolve melebur batas, dan spatial join menghubungkan data menurut letak.
- Nomor predikat berbeda antar alat. Periksa tabelnya.
- Model menyimpan rangkaian alat sebagai resep. Satu model bisa dijalankan berulang dengan jarak berbeda.

Latihan. Buat salinan Skrip 18.7 untuk skenario sempadan sungai 15 meter dan zona permukiman 10 meter. Catat luas areal efektif dan jumlah titik panas yang jatuh di dalamnya. Bagian mana yang Anda ubah?

Tabel perbandingan: alat geoprocessing di bab ini

Pertanyaan	Alat	Isian utama	Skrip
Berapa luas zona aman?	`native:buffer`	`DISTANCE`, `DISSOLVE`	18.2

BAB 19: Piksel Bicara Angka: Analisis Raster dan Klasifikasi Citra dengan Skrip

> **Berdasarkan Seri I2:** Skrip bab ini berasal dari Buku I2, *Analisis Raster dan Pengindraan Jauh Dasar* (Intermediate Class), berkas `i2_01_aljabar_raster.py` sampai `i2_07_terpadu.py`. Di buku itu, topik raster dibahas lewat menu dan klik QGIS. Bab ini menunjukkan sisi skripnya. Ketujuh skrip saling bersambung, sebab hasil satu skrip dipakai skrip berikutnya.

Studi kasus: "Seberapa tinggi, seberapa curam, lahan apa?"

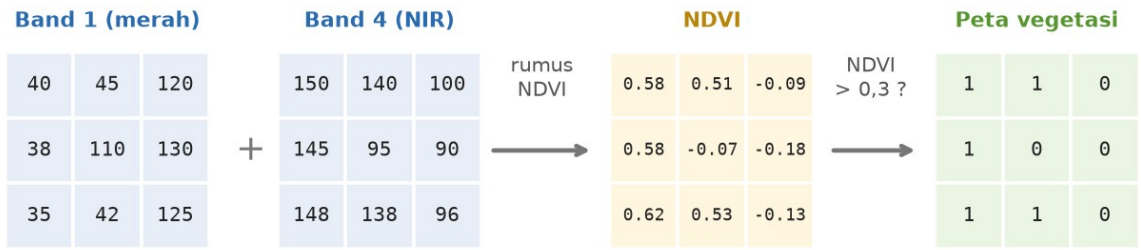
Kepala Seksi mengirim tiga pertanyaan sekaligus. Seberapa luas tajuk pohon yang tingginya lima meter atau lebih? Petak mana yang terlalu curam? Lalu lahan apa saja yang ada di citra baru, dan seberapa bisa peta lahannya dipercaya? Anda bisa menjawabnya lewat menu QGIS satu per satu. Tetapi bulan depan datang citra baru dan pertanyaannya sama. Dengan skrip, Anda cukup mengganti jalur data.

Konsep: raster dalam skrip, dalam tiga kalimat

Raster itu kertas berpetak, dan tiap petak menyimpan satu angka, misalnya tinggi tanah atau pantulan cahaya. Skrip mengolah angka-angka itu dengan dua cara. Cara pertama, Anda memanggil alat QGIS lewat `processing.run`, seperti menekan tombol di Processing Toolbox. Cara kedua, Anda memuat angka ke NumPy lalu menghitungnya sendiri.

Ilustrasi 19.1: Rumus dijalankan di tiap petak

Skema; bukan tangkapan layar



Rumus yang sama dijalankan di tiap petak yang letaknya sama. Hasilnya raster baru.

Angka di gambar hanya contoh. Rata-rata peta 0/1 = bagian petak bernilai 1.

Lima istilah yang perlu Anda kenal sekarang:

- **Piksel**: satu petak raster, yang menyimpan satu angka.
- **Band**: satu lapis angka di dalam raster. Citra berwarna punya beberapa band, misalnya merah, hijau, biru, dan inframerah.
- `processing.run`: perintah untuk menjalankan alat Processing QGIS dari skrip.

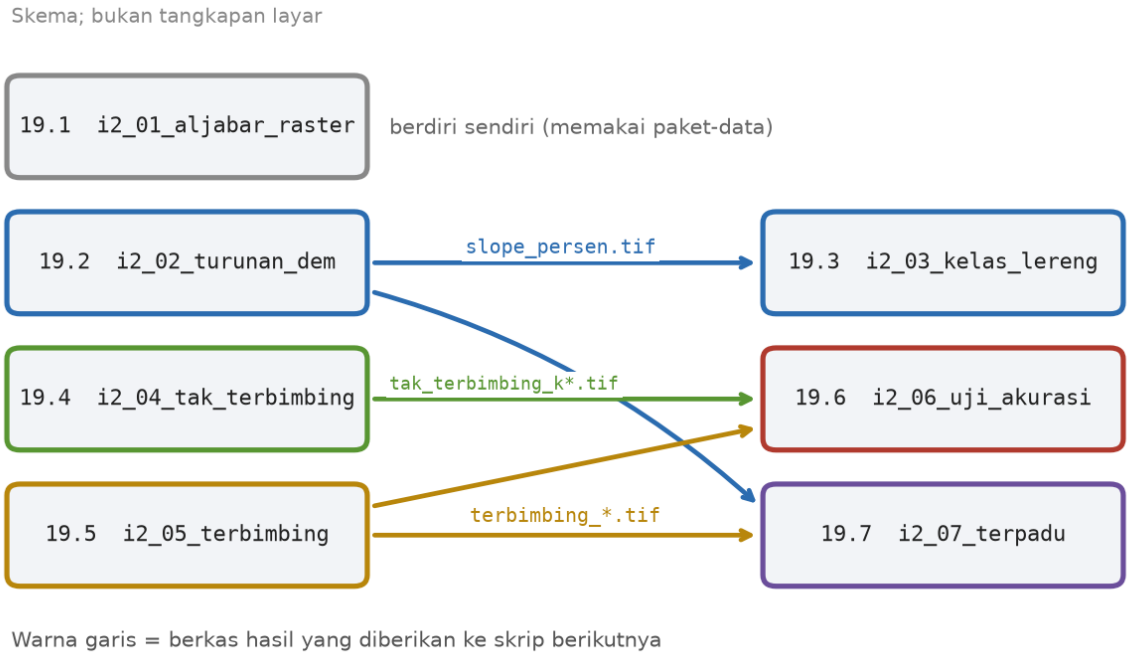
- **Array** NumPy: tabel angka di memori yang bisa dihitung sekaligus. Pustaka NumPy dibahas di Bab 12.
- **GDAL** (di Python bernama `osgeo`): pustaka pembaca dan penulis berkas raster.

Dua cara bekerja, dan urutan ketujuh skrip

Cara kerja	Alat yang dipakai	Dipakai di skrip
Panggil alat QGIS	`processing.run("native:...")`	19.1, 19.2, 19.3, 19.7
Hitung sendiri	NumPy dan GDAL	19.4, 19.5
Campuran	Alat QGIS dan NumPy bersama	19.6

Skrip 19.1 berdiri sendiri. Skrip lain saling memberi berkas hasil, seperti gambar berikut.

Ilustrasi 19.2: Siapa memberi berkas ke siapa



Jalankan berurutan: 19.1, 19.2, 19.3, 19.4, 19.5, 19.6, 19.7. Bila Anda melompat, skrip berikutnya tidak menemukan berkas yang ia butuhkan.

Siapkan dulu: data, folder, dan tempat menjalankan

1. Siapkan dua paket data. Paket `paket-i2` memuat `DEM_Lereng.tif`, `Citra_Lahan.tif`, `Area_Latih.gpkg`, dan `Titik_Uji.gpkg`. Paket `paket-data` memuat `Citra_Komposit.tif`, `DSM_Drone.tif`, `DTM_Drone.tif`, dan `Batas_KPH.gpkg`.
2. Putuskan satu folder hasil di komputer Anda. Skrip menulis semua peta hasil ke sana.
3. Buka **Plugins ► Python Console**, klik **Show Editor**, lalu buka skrip. Cara ini dijelaskan di Bab 3.
4. Ganti jalur di bagian atas skrip, lalu klik **Run**.

> **Jalur di skrip hanyalah contoh:** Di tiap skrip, jalur data dibaca dari variabel lingkungan `I2_DATA` (folder `paket-data`) atau `I2_PAKET` (folder `paket-i2`). Bila variabel itu belum Anda atur, skrip memakai alamat bawaan `C:/KPH_Contoh/paket-data` atau `C:/KPH_Contoh/paket-i2`. Jalur hasil `C:/temp/hasil_i2` ditulis langsung di skrip. Anda wajib menyesuaikannya: atur variabelnya, atau ganti alamat bawaan dengan jalur milik Anda (cara kerja `os.environ.get` dijelaskan di Bab 16).

Satu hal lagi: ketujuh skrip memakai modul `processing`, yang hanya ada di dalam QGIS. Bila Anda menjalankannya dari terminal `python-qgis` biasa, muncul `ModuleNotFoundError: No module named 'processing'`. Pakai Editor Skrip QGIS. Nomor skrip di bab ini (19.1 dan seterusnya) mengikuti bab ini. Di dalam berkas, nomor yang tertulis mengikuti Buku I2 (Bab 1 sampai Bab 6).

Skrip 19.1: Berhitung dengan piksel

Berkas `i2_01_aljabar_raster.py`. Di dalam berkasnya tertulis "I2 Bab 1".

Tujuan. Menghitung NDVI (indeks kehijauan), menandai piksel bervegetasi, menghitung tinggi tajuk dari dua raster drone, lalu menandai tajuk setinggi lima meter atau lebih.

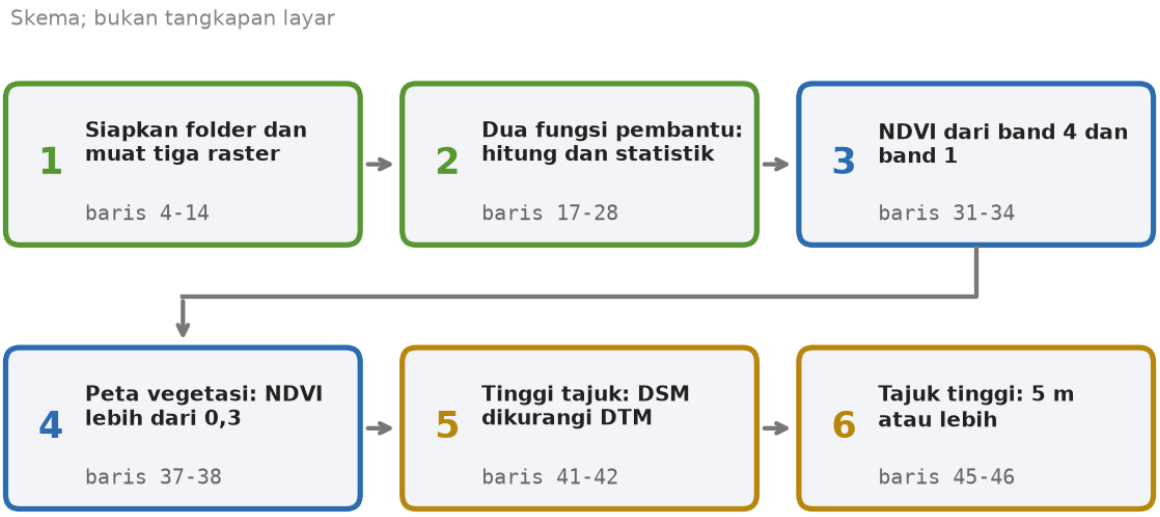
Bahan. Paket `paket-data`: `Citra_Komposit.tif` (300 x 300 piksel, 1 m, empat band), `DSM_Drone.tif` dan `DTM_Drone.tif` (masing-masing 800 x 800 piksel, 0,5 m). Skrip dijalankan dari Editor Skrip QGIS.

Lima istilah baru untuk skrip ini:

- **NDVI**: angka kehijauan tiap piksel, dihitung dari band inframerah dekat (NIR) dan band merah. Makin hijau, makin besar. Rumusnya (NIR dikurangi R) dibagi (NIR ditambah R).
- **DSM**: model tinggi permukaan, yaitu puncak pohon dan atap. **DTM**: model tinggi tanah saja.
- **CHM**: tinggi tajuk, yaitu DSM dikurangi DTM.
- **Ekspresi**: rumus yang Anda tulis untuk Raster Calculator, misalnya `"A@1" - "B@1"`. Bentuk `"nama@1"` berarti layer bernama `nama`, band 1.
- **Alat native:rastercalc**: nama alat Raster Calculator bila dipanggil dari skrip.

Skrip ini bekerja dalam enam tahap, dengan nomor baris seperti di gambar.

Ilustrasi 19.3: Alur skrip 19.1



Skrip utuh:

PYTHON

```
# -*- coding: utf-8 -*-
# I2 Bab 1: aljabar raster (NDVI, tinggi tajuk, mask). Penulis: Badar Mubarok Yogaswara
# Jalankan di Python Console QGIS. Ubah dua jalur di bawah sesuai komputer Anda.
import os
import processing
from qgis.core import QgsRasterLayer, QgsProcessingFeedback

PAKET = os.environ.get("I2_DATA", r"C:/KPH_Contoh/paket-data")     # folder paket-data
HASIL = r"C:/temp/hasil_i2"                                                    # folder hasil (dibuat bila
belum ada)
os.makedirs(HASIL, exist_ok=True)

citra = QgsRasterLayer(os.path.join(PAKET, "Citra_Komposit.tif"), "Citra_Komposit")
dsm = QgsRasterLayer(os.path.join(PAKET, "DSM_Drone.tif"), "DSM_Drone")
dtm = QgsRasterLayer(os.path.join(PAKET, "DTM_Drone.tif"), "DTM_Drone")


def hitung(ekspresi, layers, keluar, ref):
    """Menjalankan Raster calculator dengan ukuran sel, luas, dan CRS mengikuti layer acuan."""
    return processing.run("native:rastercalc", {
        "EXPRESSION": ekspresi, "LAYERS": layers,
        "CELL_SIZE": ref.rasterUnitsPerPixelX(), "EXTENT": ref.extent(), "CRS": ref.crs(),
        "OUTPUT": os.path.join(HASIL, keluar)})["OUTPUT"]


def statistik(jalur, band=1):
    lyr = QgsRasterLayer(jalur, "x")
    s = lyr.dataProvider().bandStatistics(band)
    return s.minimumValue, s.maximumValue, s.mean, s.stdDev


# 1) NDVI = (NIR - R) / (NIR + R); band 1 = R, band 4 = NIR
ndvi = hitung('("Citra_Komposit@4" - "Citra_Komposit@1") / ("Citra_Komposit@4" + "Citra_Komposit@1")',
              [citra], "ndvi.tif", citra)
print("NDVI min, maks, rata-rata, simpangan baku:", [round(v, 3) for v in statistik(ndvi)])

# 2) NDVI > 0,3 menjadi 1 (vegetasi) dan selain itu 0
```

```
veg = hitung('"ndvi@1" > 0.3', [QgsRasterLayer(ndvi, "ndvi")], "veg.tif", citra)
print("Rata-rata veg (=bagian piksel bervegetasi):", round(statistik(veg)[2], 4))

# 3) tinggi tajuk (CHM) = DSM - DTM
chm = hitung('"DSM_Drone@1" - "DTM_Drone@1"', [dsm, dtm], "chm.tif", dsm)
print("CHM min, maks, rata-rata:", [round(v, 2) for v in statistik(chm)[:3]])

# 4) tajuk tinggi: tinggi >= 5 m
tinggi = hitung('"chm@1" >= 5', [QgsRasterLayer(chm, "chm")], "tajuk_tinggi.tif", dsm)
print("Bagian piksel dengan tinggi >= 5 m:", round(statistik(tinggi)[2], 4))
```

Penjelasan baris demi baris. Skrip ini panjang, jadi penjelasannya dibagi tiga blok. Nomor baris mengikuti berkas di atas.

Blok A: baris 1 sampai 14, persiapan.

Baris	Tulisan	Artinya
1-3	`# -*- coding: utf-8 -*-` dan dua komentar	Komentar. Baris 1 memberi tahu Python bahwa berkas memakai huruf UTF-8, agar huruf Indonesia terbaca benar. Baris 2-3 memuat judul, nama penulis, dan pesan: ubah dua jalur.
4	`import os`	Panggil perkakas `os` untuk urusan folder dan jalur (Bab 8).
5	`import processing`	Panggil modul `processing`, tempat perintah `processing.run`. Hanya ada di dalam QGIS.
6	`from qgis.core import QgsRasterLayer, QgsProcessingFeedback`	Panggil dua perkakas QGIS. `QgsRasterLayer` membuka raster. `QgsProcessingFeedback` tidak dipakai di skrip ini. Ia sisa dan tidak berpengaruh.
8	`PAKET = os.environ.get("I2_DATA", r"C:/KPH_Contoh/paket-data")`	Isi kotak `PAKET` dengan lokasi folder `paket-data`. `os.environ.get` membaca variabel lingkungan `I2_DATA`; bila belum diatur, dipakai alamat bawaan di belakangnya. Huruf `r` di depan kutip membuat teks dibaca apa adanya. Atur variabelnya atau ganti alamat bawaan.
9	`HASIL = r"C:/temp/hasil_i2"`	Lokasi folder hasil. Ganti sesuai komputer Anda.
10	`os.makedirs(HASIL, exist_ok=True)`	Buat folder hasil bila belum ada. Bagian `exist_ok=True` berarti: bila sudah ada, tidak apa-apa.
12-14	`citra = QgsRasterLayer(os.path.join(PAKET, "Citra_Komposit.tif"), "Citra_Komposit")`	Buka tiga raster ke kotak `citra`, `dsm`, dan `dtm`. `os.path.join` menyambung folder dengan nama berkas. Isian kedua adalah nama layer di QGIS. Nama ini dipakai lagi di dalam rumus.

Blok B: baris 17 sampai 28, dua fungsi pembantu. Fungsi adalah resep bernama yang bisa dipakai berulang (Bab 7). Skrip ini menulis dua resep, lalu memakainya empat kali.

Baris	Tulisan	Artinya
17	`def hitung(ekspresi, layers, keluar, ref):`	Mulai resep bernama `hitung`. Ia menerima empat bahan: rumus, daftar layer, nama berkas hasil, dan layer acuan.
18	`"""Menjalankan Raster calculator ..."""`	Docstring, yaitu catatan satu kalimat tentang apa yang dikerjakan fungsi.
19-22	`return processing.run("native:rastercalc", {...})["OUTPUT"]`	Jalankan alat Raster Calculator. Isi kurung kurawal adalah pengaturan alat. `EXPRESSION` rumus. `LAYERS` layer yang dipakai rumus. `CELL_SIZE`, `EXTENT`, dan `CRS` (ukuran piksel, luas area, sistem koordinat) meniru layer acuan. `OUTPUT` jalur berkas hasil. `["OUTPUT"]` di ujung mengambil jalur itu dari jawaban alat, dan `return` mengirimnya kembali.
25	`def statistik(jalur, band=1):`	Resep kedua. Bagian `band=1` adalah nilai bawaan: bila Anda tidak menyebut band, ia memakai band 1.
26-27	`lyr = QgsRasterLayer(jalur, "x")` dan `s = ...bandStatistics(band)`	Buka raster hasil, lalu minta QGIS menghitung statistik band. Nama layer `x` tidak penting di sini.
28	`return s.minimumValue, s.maximumValue, s.mean, s.stdDev`	Kembalikan empat angka sekaligus: terkecil, terbesar, rata-rata, dan simpangan baku (ukuran sebaran). Empat angka yang dipisah koma menjadi satu paket, yang disebut tuple (Bab 5).

Blok C: baris 31 sampai 46, empat perhitungan.

Baris	Tulisan	Artinya
32-33	`ndvi = hitung('(...)', [citra], "ndvi.tif", citra)`	Hitung NDVI. Rumus diapit kutip tunggal supaya kutip ganda boleh dipakai di dalamnya. `"Citra_Komposit@4"` berarti layer `Citra_Komposit`, band 4 (NIR). Band 1 adalah merah. Hasilnya disimpan sebagai `ndvi.tif`, dan jalurnya masuk kotak `ndvi`.
34	`print(..., [round(v, 3) for v in statistik(ndvi)])`	Ambil statistik NDVI, bulatkan tiap angka ke tiga desimal dengan list comprehension (Bab 6), lalu cetak daftarnya.
37	`veg = hitung('"ndvi@1" > 0.3', [QgsRasterLayer(ndvi, "ndvi")], "veg.tif", citra)`	Rumus perbandingan: hasilnya 1 bila NDVI lebih dari 0,3, selain itu 0. Layer `ndvi` dibuka dari berkas hasil tadi. Nama `"ndvi"` harus sama dengan nama di rumus.
38	`print(..., round(statistik(veg)[2], 4))`	Angka `[2]` mengambil isi ketiga dari tuple, yaitu rata-rata (hitungan mulai dari 0). Rata-rata peta berisi 0 dan 1 sama dengan bagian piksel yang bernilai 1.
41	`chm = hitung('"DSM_Drone@1" - "DTM_Drone@1"', [dsm, dtm], "chm.tif", dsm)`	Kurangkan DTM dari DSM untuk mendapat tinggi tajuk. Ukuran piksel mengikuti `dsm` (0,5 m).
42	`print(..., [round(v, 2) for v in statistik(chm)[:3]])`	Potongan `[:3]` mengambil tiga isi pertama: terkecil, terbesar, rata-rata.

Baris	Tulisan	Artinya
45-46	`tinggi = hitung('"chm@1" >= 5', ...)`	Tandai piksel bertajuk lima meter atau lebih (1 atau 0), lalu cetak rata-ratanya.

Hasil. Layar menampilkan (berdasarkan uji pada QGIS 4.0.2):

```
HASIL
NDVI min, maks, rata-rata, simpangan baku: [-0.26, 0.899, 0.338, 0.281]
Rata-rata veg (=bagian piksel bervegetasi): 0.6666
CHM min, maks, rata-rata: [0.0, 13.96, 2.6]
Bagian piksel dengan tinggi >= 5 m: 0.1887
```

Dari hasil ini Anda tahu: sekitar dua pertiga piksel citra bervegetasi (0,6666). Tajuk tertinggi 13,96 m. Dan 18,87% piksel bertajuk lima meter atau lebih. Area drone berukuran 800 x 800 piksel, tiap piksel 0,5 m x 0,5 m, jadi luasnya 16 hektare. Luas tajuk tinggi kira-kira 0,1887 x 16 = 3,02 hektare. Itu jawaban untuk Kepala Seksi. [dihitung dari hasil skrip]

Bagian yang boleh diubah.

- **Baris 8 dan 9, jalur.** Ganti dengan lokasi `paket-data` dan folder hasil di komputer Anda.
- **Baris 37, angka 0.3.** Ambang kehijauan. Pada uji, ambang `0.5` menurunkan rata-rata `veg` dari 0,6666 menjadi 0,4309.
- **Baris 45, angka 5.** Ambang tinggi tajuk. Pada uji, ambang `10` menurunkan rata-rata dari 0,1887 menjadi 0,0294. Kata "5 m" di teks baris 46 tidak ikut berubah, jadi ganti juga teksnya.
- **Baris 32, nomor band.** Bila citra Anda menyimpan NIR di band lain, ubah angka setelah tanda `@`.
- **Angka di round(...).** Mengatur jumlah desimal yang tampil. Tidak memengaruhi berkas hasil.

Jebakan.

- **Jalur data salah.** Bila `PAKET` menunjuk folder yang tidak ada, skrip tidak berhenti di baris 12. Ia baru berhenti di baris 32 dengan `QgsProcessingException: Unable to execute algorithm` dan `Incorrect parameter value for EXTENT`. Penyebabnya: layer tidak terbaca, sehingga luasnya kosong. Pelajarannya: bila galat menyebut `EXTENT`, curigai jalur data.
- **Nama layer dalam rumus berbeda.** Bila rumus menulis `"Citra@4"` padahal layernya bernama `Citra_Komposit`, hasilnya `QgsProcessingException: Error occurred while performing calculation`. Pesannya tidak menyebut nama yang salah. Samakan nama di baris 12-14 dengan nama di rumus.
- **Band 4 bertanda "Alpha".** GDAL menandai band keempat berkas ini sebagai transparansi. Untuk hitungan, ia tetap angka biasa, jadi rumus NDVI benar. Tetapi saat menampilkan komposit warna, pakai band 1 sampai 3 saja. [CEK di tampilan QGIS Anda]
- **Layer yang dibuat di dalam daftar.** Pada baris 37 dan 45, `QgsRasterLayer(...)` ditaruh langsung di dalam daftar tanpa kotak. Pada uji ini berhasil. Skrip 19.2 baris 28 memberi saran lebih aman: simpan layer di kotak sendiri agar tidak dibuang Python lebih awal. [kemungkinan]

Latihan variasi.

1. Ganti ambang NDVI menjadi 0,5 dan ambang tajuk menjadi 10 meter. Lihat bagian piksel yang tersisa.
2. Cetak luas tajuk tinggi dalam hektare, bukan bagian piksel.

Contoh jawaban nomor 1, hanya bagian yang berubah:

```python
veg = hitung('"ndvi@1" > 0.5', [QgsRasterLayer(ndvi, "ndvi")], "veg.tif", citra)
tinggi = hitung('"chm@1" >= 10', [QgsRasterLayer(chm, "chm")], "tajuk_tinggi.tif", dsm)
```

Layar menampilkan `0.4309` untuk vegetasi dan `0.0294` untuk tajuk tinggi (tulisan "5 m" di baris 46 masih lama).

Contoh jawaban nomor 2, satu baris tambahan di akhir skrip asli:

```python
print("Luas tajuk >= 5 m (ha):", round(statistik(tinggi)[2] * 640000 * 0.25 / 10000, 2))
```

Layar menampilkan `Luas tajuk >= 5 m (ha): 3.02`. Angka 640000 adalah jumlah piksel, dan 0,25 adalah luas satu piksel dalam meter persegi.

> **Pola yang bisa dipakai ulang:** Skrip ini memberi dua resep yang berguna di mana saja. Untuk raster apa pun: `hasil = hitung('RUMUS', [layer1, layer2], "nama.tif", layer_acuan)`, lalu `statistik(hasil)`. Yang Anda ganti: **rumus**, **daftar layer**, **nama berkas hasil**, dan **layer acuan** yang menentukan ukuran piksel.

Skrip 19.2: Membaca bentuk bumi dari DEM

Berkas `i2_02_turunan_dem.py`. Di dalam berkasnya tertulis "I2 Bab 2".

Tujuan. Membuat peta kemiringan, arah lereng, dan bayangan bukit dari satu DEM, lalu mengubah kemiringan dari derajat menjadi persen dan mencetak statistik keempat peta.

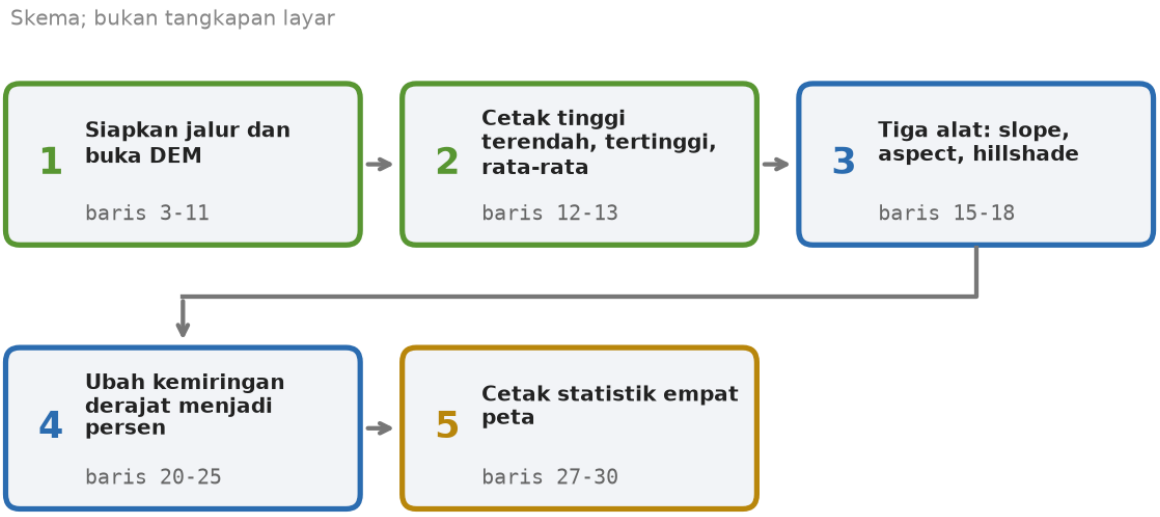
Bahan. Paket `paket-i2`, berkas `DEM_Lereng.tif` (200 x 200 piksel, 2 m). Skrip dijalankan dari Editor Skrip QGIS.

Lima istilah baru untuk skrip ini:

- **DEM**: raster yang tiap pikselnya menyimpan tinggi tanah.
- **Kemiringan** (*slope*): seberapa miring permukaan, dalam derajat atau persen.
- **Arah lereng** (*aspect*): ke mana lereng menghadap, diukur dari utara searah jarum jam (90 derajat = timur, 180 = selatan).
- **Bayangan bukit** (*hillshade*): gambar bayangan seolah bukit disinari matahari.
- **Faktor Z**: pengali tinggi, dipakai bila satuan tinggi dan satuan peta berbeda. Di data ini sama-sama meter, jadi nilainya 1.

Skrip ini bekerja dalam lima tahap.

Ilustrasi 19.4: Alur skrip 19.2



Skrip utuh:

```python
# -*- coding: utf-8 -*-
# I2 Bab 2: turunan DEM (kemiringan, arah lereng, bayangan bukit). Penulis: Badar Mubarok Yogaswara
import os
import processing
from qgis.core import QgsRasterLayer

PAKET_I2 = os.environ.get("I2_PAKET", r"C:/KPH_Contoh/paket-i2")     # folder paket-i2
HASIL = r"C:/temp/hasil_i2"
os.makedirs(HASIL, exist_ok=True)

dem = QgsRasterLayer(os.path.join(PAKET_I2, "DEM_Lereng.tif"), "DEM_Lereng")
s = dem.dataProvider().bandStatistics(1)
print("DEM min, maks, rata-rata:", round(s.minimumValue, 1), round(s.maximumValue, 1), round(s.mean, 1))

kemiringan = processing.run("native:slope", {"INPUT": dem, "Z_FACTOR": 1.0, "OUTPUT": os.path.join(HASIL,
"slope_derajat.tif")})["OUTPUT"]
arah = processing.run("native:aspect", {"INPUT": dem, "Z_FACTOR": 1.0, "OUTPUT": os.path.join(HASIL,
"aspect.tif")})["OUTPUT"]
bayangan = processing.run("native:hillshade", {"INPUT": dem, "Z_FACTOR": 1.0, "AZIMUTH": 315.0, "V_ANGLE":
45.0,
                                               "OUTPUT": os.path.join(HASIL, "hillshade.tif")})["OUTPUT"]

# kemiringan dalam persen = tan(derajat) x 100
lyr = QgsRasterLayer(kemiringan, "slope_derajat")
persen = processing.run("native:rastercalc", {
    "EXPRESSION": 'tan("slope_derajat@1" * 3.14159265358979 / 180) * 100', "LAYERS": [lyr],
    "CELL_SIZE": dem.rasterUnitsPerPixelX(), "EXTENT": dem.extent(), "CRS": dem.crs(),
    "OUTPUT": os.path.join(HASIL, "slope_persen.tif")})["OUTPUT"]

for nama, jalur in (("Slope (derajat)", kemiringan), ("Aspect (derajat dari utara)", arah), ("Hillshade (0-
255)", bayangan), ("Slope (persen)", persen)):
    lapisan = QgsRasterLayer(jalur, "x")      # simpan di variabel agar tidak dibuang
    st = lapisan.dataProvider().bandStatistics(1)
    print("%-28s min %.2f  maks %.2f  rata-rata %.2f" % (nama, st.minimumValue, st.maximumValue, st.mean))
```

Penjelasan baris demi baris. Bagian yang sama dengan Skrip 19.1 (impor, `os.makedirs`, `QgsRasterLayer`) tidak diulang. Hanya yang baru dijelaskan.

Baris	Tulisan	Artinya
7-9	`PAKET_I2 = os.environ.get("I2_PAKET", ...), HASIL = ..., os.makedirs(...)`	Dua jalur seperti 19.1. `PAKET_I2` dibaca dari variabel `I2_PAKET`, dengan alamat bawaan `C:/KPH_Contoh/paket-i2`. Folder hasil dibuat bila belum ada.
11	`dem = QgsRasterLayer(..., "DEM_Lereng")`	Buka DEM.
12-13	`s = dem.dataProvider().bandStatistics(1)` lalu `print(...)`	Hitung statistik DEM, lalu cetak tinggi terendah, tertinggi, dan rata-rata, dibulatkan satu desimal. Ini cara cepat memeriksa bahwa DEM terbaca.
15	`kemiringan = processing.run("native:slope", {...})["OUTPUT"]`	Jalankan alat kemiringan. `Z_FACTOR` 1.0 berarti satuan sama. Hasilnya berupa derajat, dan jalurnya masuk kotak `kemiringan`.
16	`arah = processing.run("native:aspect", {...})["OUTPUT"]`	Alat arah lereng, dengan pola sama.
17-18	`bayangan = processing.run("native:hillshade", {...})["OUTPUT"]`	Alat bayangan bukit. `AZIMUTH` 315 artinya sinar datang dari barat laut. `V_ANGLE` 45 adalah tinggi matahari di atas cakrawala. Perintah ini terbagi dua baris karena panjang, dan itu boleh di dalam kurung.
21	`lyr = QgsRasterLayer(kemiringan, "slope_derajat")`	Buka peta kemiringan yang baru dibuat. Namanya `slope_derajat`, dan nama ini muncul di rumus baris 23.
22-25	`persen = processing.run("native:rastercalc", {...})["OUTPUT"]`	Ubah derajat menjadi persen dengan rumus `tan(derajat x pi / 180) x 100`. Kalikan dengan pi per 180 mengubah derajat menjadi radian, satuan yang dipakai fungsi `tan`. Angka pi diketik langsung. Ukuran piksel dan luas mengikuti DEM.
27	`for nama, jalur in ((...), (...), (...), (...)):`	Ulangi empat kali. Tiap isian adalah pasangan: judul dan jalur peta. Kotak `nama` dan `jalur` langsung terisi dari pasangan itu.
28-29	`lapisan = QgsRasterLayer(jalur, "x")` lalu `st = ...bandStatistics(1)`	Buka peta, hitung statistiknya. Komentar di baris 28 mengingatkan: simpan layer di kotak agar tidak dibuang.
30	`print("%-28s min %.2f  maks %.2f  rata-rata %.2f" % (...))`	Cetak satu baris. `%-28s` memberi judul lebar 28 huruf, rata kiri. `%.2f` mencetak angka dengan dua desimal. Tanda `%` di tengah mengisi lubang-lubang itu dengan empat nilai (Bab 9).

Hasil. Layar menampilkan:

```
HASIL
DEM min, maks, rata-rata: 89.6 150.7 114.7
Slope (derajat)              min 0.13  maks 37.34  rata-rata 12.57
Aspect (derajat dari utara)  min 0.01  maks 359.99  rata-rata 200.88
Hillshade (0-255)            min 44.98  maks 248.16  rata-rata 174.62
```

```
Slope (persen)              min 0.22  maks 76.30  rata-rata 23.09
```

Dari hasil ini Anda tahu: tanah di area ini naik dari 89,6 sampai 150,7 m. Lereng paling curam 37,34 derajat, sama dengan 76,30 persen (tan 37,34 derajat sama dengan 0,763). Rata-rata arah lereng (200,88) tidak bermakna sebagai arah, sebab 359 dan 1 derajat sebenarnya berdekatan. Pakai peta aspect untuk melihat sebaran, bukan untuk dirata-ratakan.

Bagian yang boleh diubah.

- **Baris 7 dan 8, jalur.** Ganti sesuai komputer Anda.
- **Baris 15-17, Z_FACTOR.** Ubah hanya bila satuan tinggi DEM berbeda dari satuan petanya. Di data ini biarkan 1.0.
- **Baris 17, AZIMUTH dan V_ANGLE.** Arah sinar dan tinggi matahari. Pada uji, azimut `135.0` mengubah hillshade menjadi min 58,68, maks 246,79, rata-rata 172,47.
- **Baris 27, daftar pasangan.** Tambah atau hapus pasangan untuk mengatur peta yang dicetak statistiknya.

Jebakan.

- **Jalur DEM salah.** Layar menampilkan `ERROR 4: ... No such file or directory`. Lalu baris statistik berisi angka aneh: `DEM min, maks, rata-rata: 1.7976931348623157e+308 -1.7976931348623157e+308 0.0`. Angka raksasa itu tanda layer kosong. Setelahnya skrip berhenti di baris 22 dengan `Incorrect parameter value for EXTENT`. Pelajarannya: baca baris statistik DEM dulu.
- **Persen bukan derajat.** Lereng 45 derajat sama dengan 100 persen, bukan 50 persen. Itu sebabnya skrip memakai `tan`, bukan sekadar dikali 100 lalu dibagi 90.
- **Skrip berikutnya bergantung pada slope_persen.tif.** Skrip 19.3 dan 19.7 memakai berkas ini. Jalankan 19.2 lebih dulu.
- **Aspect adalah arah, bukan ukuran.** Jangan merata-ratakannya.

Latihan variasi.

1. Ganti azimut matahari menjadi 135 dan lihat statistik hillshade.
2. Cetak statistik hanya untuk dua peta: kemiringan dalam derajat dan dalam persen.

Contoh jawaban nomor 1, hanya bagian yang berubah (baris 17):

```
PYTHON
bayangan = processing.run("native:hillshade", {"INPUT": dem, "Z_FACTOR": 1.0, "AZIMUTH": 135.0, "V_ANGLE":
45.0,
                                               "OUTPUT": os.path.join(HASIL, "hillshade.tif")})["OUTPUT"]
```

Layar menampilkan `Hillshade (0-255)              min 58.68  maks 246.79  rata-rata 172.47`.

Contoh jawaban nomor 2, baris 27 menjadi:

```
PYTHON
for nama, jalur in (("Slope (derajat)", kemiringan), ("Slope (persen)", persen)):
```

Layar hanya menampilkan dua baris Slope di bawah baris DEM.

> **Pola yang bisa dipakai ulang:** Memanggil alat QGIS selalu berbentuk: `hasil = processing.run("NAMA_ALAT", {"INPUT": layer, ...pengaturan..., "OUTPUT": jalur})["OUTPUT"]`. Untuk alat lain, ganti **nama alat** dan **pengaturannya**. Nama alat dan nama pengaturan dibaca dari jendela alat di QGIS, lewat tombol **Advanced ► Copy as Python Command**. [CEK di QGIS Anda]

Skrip 19.3: Dari angka ke kelas lereng

Berkas `i2_03_kelas_lereng.py`. Di dalam berkasnya tertulis "I2 Bab 3". Skrip ini memakai `slope_persen.tif` dari Skrip 19.2, jadi jalankan 19.2 lebih dulu.

Tujuan. Mengelompokkan kemiringan (persen) menjadi lima kelas, lalu menghitung luas tiap kelas di tiap petak KPH.

Bahan. Hasil Skrip 19.2 (`slope_persen.tif` di folder hasil) dan `Batas_KPH.gpkg` (layer `Batas_KPH`, tiga petak) dari `paket-data`. Skrip dijalankan dari Editor Skrip QGIS.

Lima istilah baru untuk skrip ini:

- **Reklasifikasi**: mengganti rentang angka menjadi kelas, seperti mengubah nilai ujian 0 sampai 100 menjadi huruf A sampai E.
- **Tabel reklasifikasi**: daftar tiga isian per kelas: awal rentang, akhir rentang, dan nomor kelas baru.
- **Aturan batas**: nilai yang tepat di batas masuk kelas bawah atau atas. Di skrip ini masuk kelas bawah.
- **Zona dan histogram zonal**: zona adalah poligon tempat raster dirangkum (di sini petak KPH). Histogram zonal menghitung jumlah piksel tiap kelas di dalam tiap zona.
- **NoData**: piksel tanpa nilai.

Ilustrasi 19.5: Alur skrip 19.3



Skrip utuh:

```python
# -*- coding: utf-8 -*-
# I2 Bab 3: reklasifikasi kemiringan dan luas kelas lereng per KPH. Penulis: Badar Mubarok Yogaswara
# Jalankan setelah i2_02_turunan_dem.py (memakai slope_persen.tif di folder hasil).
import os
import processing
from qgis.core import QgsRasterLayer, QgsVectorLayer

PAKET = os.environ.get("I2_DATA", r"C:/KPH_Contoh/paket-data")
HASIL = r"C:/temp/hasil_i2"

persen = QgsRasterLayer(os.path.join(HASIL, "slope_persen.tif"), "slope_persen")
batas = QgsVectorLayer(os.path.join(PAKET, "Batas_KPH.gpkg") + "|layername=Batas_KPH", "Batas_KPH", "ogr")

# tabel: awal, akhir, kelas baru. Aturan batas: awal < nilai <= akhir (RANGE_BOUNDARIES = 0)
tabel = [-1, 8, 1,
         8, 15, 2,
         15, 25, 3,
         25, 45, 4,
         45, 100000, 5]
kelas = processing.run("native:reclassifybytable", {
    "INPUT_RASTER": persen, "RASTER_BAND": 1, "TABLE": tabel, "NO_DATA": -9999,
    "RANGE_BOUNDARIES": 0, "NODATA_FOR_MISSING": False, "DATA_TYPE": 5,
    "OUTPUT": os.path.join(HASIL, "kelas_lereng.tif")})["OUTPUT"]

lyr_kelas = QgsRasterLayer(kelas, "kelas_lereng")
hist = processing.run("native:zonalhistogram", {
    "INPUT_RASTER": lyr_kelas, "RASTER_BAND": 1, "INPUT_VECTOR": batas, "COLUMN_PREFIX": "K",
    "OUTPUT": os.path.join(HASIL, "luas_kelas_per_kph.gpkg")})["OUTPUT"]

luas_piksel_ha = lyr_kelas.rasterUnitsPerPixelX() * lyr_kelas.rasterUnitsPerPixelY() / 10000.0
hasil = QgsVectorLayer(hist, "hist", "ogr")
print("Luas kelas lereng (ha) per KPH; piksel = %.4f ha" % luas_piksel_ha)
for f in hasil.getFeatures():
    baris = ["%s=%.2f" % (nm, f[nm] * luas_piksel_ha) for nm in hasil.fields().names() if nm.startswith("K")]
    print(f["NAMA_KPH"], "  ", "  ".join(baris))
```

Penjelasan baris demi baris. Nomor baris mengikuti berkas di atas.

Baris	Tulisan	Artinya
1-3	komentar	Baris 3 mengingatkan: jalankan setelah Skrip 19.2, karena memakai `slope_persen.tif`.
4-6	`import os`, `import processing`, `from qgis.core import QgsRasterLayer, QgsVectorLayer`	Panggil perkakas. `QgsVectorLayer` kali ini dipakai untuk membuka batas KPH (layer titik, garis, atau poligon).
8-9	`PAKET = os.environ.get("I2_DATA", ...)`, `HASIL = ...`	Lokasi `paket-data` (untuk batas KPH; dibaca dari `I2_DATA` seperti 19.1) dan folder hasil. Sesuaikan keduanya. Folder hasil harus sudah ada dari Skrip 19.2.
11	`persen = QgsRasterLayer(os.path.join(HASIL, "slope_persen.tif"), "slope_persen")`	Buka peta kemiringan persen hasil Skrip 19.2.

Baris	Tulisan	Artinya
12	`batas = QgsVectorLayer(..., "Batas_KPH", "ogr")`	Buka layer batas. Sumber datanya adalah nama berkas, lalu tanda garis tegak, lalu `layername=Batas_KPH`. Bagian setelah garis tegak memilih satu layer di dalam GeoPackage. Isian ketiga, `"ogr"`, adalah pembaca data vektor (dibahas di Bab 15).
14	komentar	Menyebut aturan batas: awal lebih kecil dari nilai, dan nilai kurang dari atau sama dengan akhir.
15-19	`tabel = [-1, 8, 1, 8, 15, 2, ...]`	Tabel reklasifikasi. Tiap tiga angka adalah satu kelas: awal, akhir, kelas baru. Misalnya `-1, 8, 1` berarti nilai di atas -1 sampai 8 menjadi kelas 1, sehingga nilai 0 ikut kelas 1. Daftar ditulis lima baris agar mudah dibaca, tetapi bagi Python ini satu daftar berisi 15 angka.
20-23	`kelas = processing.run("native:reclassifybytable", {...})["OUTPUT"]`	Jalankan alat reklasifikasi. `RASTER_BAND` 1 memilih band satu. `NO_DATA` -9999 adalah angka penanda piksel kosong. `RANGE_BOUNDARIES` 0 memilih aturan batas "awal lebih kecil dari nilai, nilai kurang dari atau sama dengan akhir". `NODATA_FOR_MISSING` False berarti nilai yang tidak tercakup tabel dibiarkan apa adanya, tidak dijadikan NoData. `DATA_TYPE` 5 memilih tipe angka hasil. Pada berkas hasil uji, tipenya Float32.
25	`lyr_kelas = QgsRasterLayer(kelas, "kelas_lereng")`	Buka peta kelas yang baru dibuat.
26-28	`hist = processing.run("native:zonalhistogram", {...})["OUTPUT"]`	Jalankan histogram zonal. `INPUT_VECTOR` adalah batas KPH sebagai zona. `COLUMN_PREFIX "K"` membuat kolom bernama `K1`, `K2`, dan seterusnya. Isinya jumlah piksel kelas itu di tiap petak. Hasilnya berkas GeoPackage.
30	`luas_piksel_ha = lyr_kelas.rasterUnitsPerPixelX() * lyr_kelas.rasterUnitsPerPixelY() / 10000.0`	Luas satu piksel dalam hektare: lebar kali tinggi dalam meter, dibagi 10.000. Piksel 2 m x 2 m = 4 m persegi = 0,0004 ha. Angka ini dihitung dari raster, jadi menyesuaikan sendiri.
31	`hasil = QgsVectorLayer(hist, "hist", "ogr")`	Buka tabel hasil histogram.
32	`print("Luas kelas lereng (ha) per KPH; piksel = %.4f ha" % luas_piksel_ha)`	Cetak judul. `%.4f` mengisi empat desimal.
33	`for f in hasil.getFeatures():`	Ulangi satu kali untuk tiap petak (tiap baris tabel).
34	`baris = ["%s=%.2f" % (nm, f[nm] * luas_piksel_ha) for nm in hasil.fields().names() if nm.startswith("K")]`	Satu baris, empat langkah. Ambil semua nama kolom (`fields().names()`). Simpan hanya yang berawalan `K`. Kalikan jumlah piksel dengan luas satu piksel. Rakit jadi teks seperti `K1=2.71`. Hasilnya daftar teks.

Baris	Tulisan	Artinya
35	`print(f["NAMA_KPH"], "  ", "  ".join(baris))`	Cetak nama petak, lalu semua teks tadi disambung dengan dua spasi (`"  ".join(...)`).

Hasil. Layar menampilkan:

```
HASIL
Luas kelas lereng (ha) per KPH; piksel = 0.0004 ha
KPH Alpha    K1=2.71  K2=0.14  K3=0.18  K4=0.48  K5=0.49
KPH Beta     K1=2.05  K2=0.12  K3=0.20  K4=1.12  K5=0.51
KPH Gamma    K1=0.23  K2=0.91  K3=2.37  K4=3.12  K5=1.37
```

Dari hasil ini Anda tahu: jumlah tiap baris cocok dengan luas petak, yaitu 4, 4, dan 8 hektare (misalnya KPH Alpha: 2,71 + 0,14 + 0,18 + 0,48 + 0,49 = 4,00). Itu tanda hitungan benar. Kelas 4 dan 5 (di atas 25 persen) memuat 0,97 ha di KPH Alpha, 1,63 ha di KPH Beta, dan 4,49 ha di KPH Gamma. Petak Gamma paling berisiko. [dihitung dari hasil skrip]

Bagian yang boleh diubah.

- **Baris 8 dan 9, jalur.** Ganti sesuai komputer Anda.
- **Baris 15-19, tabel.** Ubah batas kelas sesuai aturan instansi Anda. [CEK] Pada uji, tabel tiga kelas (0 sampai 15, 15 sampai 30, 30 sampai 1000) menghasilkan kolom `K1`, `K2`, `K3`.
- **Baris 27 dan 34, huruf K.** Keduanya harus sepasang. Bila Anda mengganti `COLUMN_PREFIX` menjadi `"Kls"`, ganti juga `startswith("K")` menjadi `startswith("Kls")`.
- **Baris 35, NAMA_KPH.** Nama kolom pengenal petak. Untuk batas lain, ganti dengan nama kolom di layer itu.
- **Baris 12, layer zona.** Ganti dengan poligon lain, misalnya batas desa, untuk luas kelas per desa.

Jebakan.

- **Nilai di luar tabel tidak jadi kelas.** Tabel di skrip mencakup nilai di atas -1 sampai 100000, jadi nilai 0 ikut kelas 1 dan lereng yang sangat curam (di atas 1000 persen) ikut kelas 5. Tetapi bila batas atas diturunkan menjadi 1000 (seperti pada versi skrip yang lama dan pada Latihan variasi nomor 1), nilai di atas 1000 tidak masuk kelas mana pun. Karena `NODATA_FOR_MISSING` False, nilainya dibiarkan. Saya mengujinya dengan batas atas 1000 pada salinan `slope_persen.tif` yang saya ubah: 100 piksel di baris ke-11 (kolom ke-11 sampai ke-110) saya isi 0, dan 25 piksel di baris ke-51 (kolom ke-51 sampai ke-75) saya isi 2000. Hasilnya, nilai 0 masuk `K1`, tetapi nilai 2000 membuat kolom tambahan `K2000` muncul di tabel:

```
HASIL
Luas kelas lereng (ha) per KPH; piksel = 0.0004 ha
KPH Alpha    K1=2.71  K2=0.14  K3=0.18  K4=0.48  K5=0.49  K2000=0.00
KPH Beta     K1=2.05  K2=0.12  K3=0.20  K4=1.12  K5=0.51  K2000=0.00
KPH Gamma    K1=0.27  K2=0.90  K3=2.36  K4=3.09  K5=1.37  K2000=0.01
```

- **Pada data latihan, jebakan ini tidak terjadi.** Kemiringan pada data latihan berkisar 0,22 sampai 76,30 persen (40.000 piksel), jauh di bawah 1000, jadi kolom `K2000` tidak muncul, baik

dengan batas atas 1000 maupun 100000. Pada DEM lain dengan tebing yang sangat curam, nilai di atas 1000 bisa ada. Karena itu skrip kini memakai batas atas `100000` di baris terakhir tabel (baris 19), bukan `1000`. Pada raster buatan yang sama, dengan batas atas `100000` kolom `K2000` hilang dan luasnya pindah ke `K5` (KPH Gamma: K5 naik dari 1,37 menjadi 1,38 ha). Bila Anda memakai tabel lain dengan batas atas rendah, jebakan ini bisa muncul lagi. Batas bawah `-1` pada baris pertama tabel memang sengaja: tanpa itu, nilai tepat 0 (dataran yang benar-benar datar) ikut terlewat dari kelas 1. Bila `NODATA_FOR_MISSING` diganti `True`, nilai di luar tabel menjadi `-9999` (NoData), bukan dibiarkan. Itu juga sah, asal Anda sadar luasnya hilang dari hitungan.

- **Skrip 19.2 belum dijalankan.** Berkas `slope_persen.tif` tidak ada. Galatnya: `Could not create raster output ...: Attempt to create 0x0 dataset is illegal,sizes must be larger than zero.` Pesan ini tidak menyebut berkas yang hilang. Cek dulu isi folder hasil.
- **Filter huruf K.** Baris 34 mengambil semua kolom berawalan `K`. Bila tabel batas Anda punya kolom lain berawalan `K`, ia ikut tercetak sebagai luas. Beri awalan yang unik. [kemungkinan]
- **Label satuan.** Judul di baris 32 menulis "(ha)". Bila Anda mengubah perkalian di baris 34, ubah juga judulnya.

Latihan variasi.

1. Ganti tabel menjadi tiga kelas: 0 sampai 15, 15 sampai 30, dan 30 sampai 1000.
2. Cetak luas dalam meter persegi, bukan hektare.

Contoh jawaban nomor 1, baris 15-19 menjadi:

```
PYTHON
tabel = [-1, 15, 1,
         15, 30, 2,
         30, 1000, 3]
```

Layar menampilkan tiga kolom per petak: `KPH Alpha  K1=2.84  K2=0.30  K3=0.86`, `KPH Beta  K1=2.17  K2=0.53  K3=1.30`, dan `KPH Gamma  K1=1.14  K2=4.26  K3=2.60`.

Contoh jawaban nomor 2, baris 34 menjadi:

```
PYTHON
    baris = ["%s=%.2f" % (nm, f[nm] * luas_piksel_ha * 10000) for nm in hasil.fields().names() if
nm.startswith("K")]
```

Layar menampilkan misalnya `KPH Alpha  K1=27056.00  K2=1388.00  K3=1828.00  K4=4808.00  K5=4920.00`. Jumlahnya 40.000 meter persegi, sama dengan 4 hektare. Judul "(ha)" di baris 32 sudah tidak benar, jadi ganti juga.

Pola yang bisa dipakai ulang: Reklasifikasi lalu luas per zona selalu dua langkah: `native:reclassifybytable` (ganti **tabel**) dan `native:zonalhistogram` (ganti **layer zona** dan **awalan kolom**). Luas = jumlah piksel kali luas satu piksel. Cocokkan awalan kolom di alat dengan filter `startswith`.

Skrip 19.4: Biarkan citra yang bicara

Berkas `i2_04_tak_terbimbing.py`. Di dalam berkasnya tertulis "I2 Bab 4". Ini skrip terberat di bab ini (69 baris), jadi penjelasannya dibagi lima blok.

Tujuan. Mengelompokkan seluruh piksel citra menjadi 6 kluster, lalu 10 kluster, dengan k-means yang ditulis sendiri memakai NumPy. Setelah itu tiap kluster diberi nama kelas lahan dengan bantuan area latih.

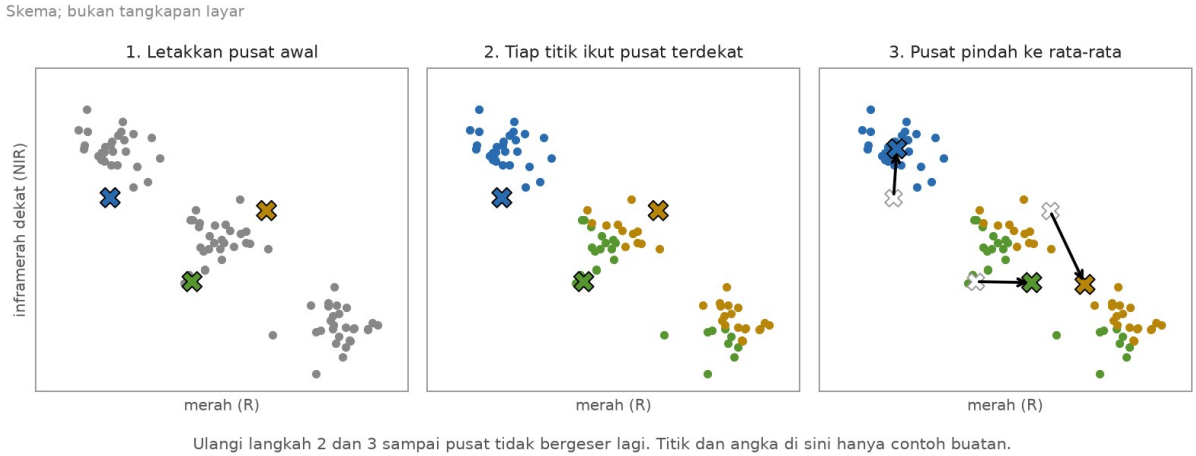
Bahan. Paket `paket-i2`: `Citra_Lahan.tif` (300 x 300 piksel, 1 m, band berurutan merah, hijau, biru, NIR) dan `Area_Latih.gpkg` (24 poligon latih, kolom `Kode` berisi 1 sampai 4). Skrip dijalankan dari Editor Skrip QGIS. Pada komputer uji, skrip selesai dalam sekitar 10 detik.

Gagasan pokoknya: **klasifikasi tak terbimbing** mengelompokkan piksel yang angka bandnya mirip, tanpa diberi contoh. Bayangkan Anda menyortir kelereng menurut warna tanpa tahu namanya. Anda hanya tahu mana yang mirip. Baru sesudah itu Anda memberi nama tiap tumpukan.

Lima istilah baru untuk skrip ini:

- **Klasifikasi**: mengubah citra berisi angka pantulan menjadi peta kelas, misalnya hutan atau sawah.
- **Kluster**: satu kelompok piksel yang angka bandnya mirip.
- **K-means**: cara mengelompokkan dengan memindah pusat kelompok berulang kali. Huruf k adalah jumlah kelompok yang Anda minta.
- **Area latih**: lokasi yang kelas sebenarnya sudah Anda ketahui dari lapangan.
- **Rasterisasi**: mengubah poligon menjadi raster.

Ilustrasi 19.6: Tiga langkah k-means



Gambar itu inti k-means. Pusat itu seperti pos jaga. Tiap piksel ikut pos terdekat. Lalu pos pindah ke tengah-tengah para pengikutnya. Ulangi sampai pos tidak bergeser lagi.

Ilustrasi 19.7: Alur skrip 19.4



Skrip utuh:

```python
# -*- coding: utf-8 -*-
# I2 Bab 4: klasifikasi tak terbimbing dengan k-means (ditulis dengan NumPy). Penulis: Badar Mubarok Yogaswara
# Jalankan di Python Console QGIS. Ubah dua jalur di bawah sesuai komputer Anda.
import os
import numpy as np
import processing
from osgeo import gdal

PAKET_I2 = os.environ.get("I2_PAKET", r"C:/KPH_Contoh/paket-i2")
HASIL = r"C:/temp/hasil_i2"
os.makedirs(HASIL, exist_ok=True)
JUMLAH_KLUSTER = [6, 10]    # dua percobaan: 6 kluster, lalu 10 kluster
NAMA = {1: "Hutan", 2: "Kebun", 3: "Sawah", 4: "Lahan Terbuka"}

ds = gdal.Open(os.path.join(PAKET_I2, "Citra_Lahan.tif"))
gt, proj = ds.GetGeoTransform(), ds.GetProjection()
citra = ds.ReadAsArray().astype("float64")                  # bentuk (4, tinggi, lebar)
_, tinggi, lebar = citra.shape
piksel = citra.reshape(4, -1).T                             # satu baris = satu piksel, empat kolom = empat
band

# lokasi yang kelasnya sudah diketahui (area latih) dipakai hanya untuk MEMBERI NAMA kluster
latih_tif = os.path.join(HASIL, "area_latih.tif")
processing.run("gdal:rasterize", {
    "INPUT": os.path.join(PAKET_I2, "Area_Latih.gpkg"), "FIELD": "Kode", "UNITS": 1, "WIDTH": 1.0, "HEIGHT":
1.0,
```

```
    "EXTENT": "312000,312300,9996000,9996300 [EPSG:32749]", "NODATA": 0, "DATA_TYPE": 0, "INIT": 0, "OUTPUT":
latih_tif})
latih = gdal.Open(latih_tif).ReadAsArray().ravel()


def kmeans(x, k, ulang=30, seed=1):
    """K-means sederhana: pusat awal dipilih acak, lalu dihitung ulang sampai stabil."""
    rng = np.random.default_rng(seed)
    pusat = x[rng.choice(len(x), k, replace=False)]
    for _ in range(ulang):
        jarak = ((x[:, None, :] - pusat[None, :, :]) ** 2).sum(axis=2)     # jarak kuadrat tiap piksel ke tiap
pusat
        label = jarak.argmin(axis=1)
        baru = np.array([x[label == i].mean(axis=0) if np.any(label == i) else pusat[i] for i in range(k)])
        if np.allclose(baru, pusat, atol=1e-3):
            break
        pusat = baru
    return label, pusat


def simpan(arr, nama):
    jalur = os.path.join(HASIL, nama)
    out = gdal.GetDriverByName("GTiff").Create(jalur, lebar, tinggi, 1, gdal.GDT_Byte)
    out.SetGeoTransform(gt)
    out.SetProjection(proj)
    out.GetRasterBand(1).WriteArray(arr.astype("uint8"))
    out.FlushCache()
    out = None
    return jalur


for k in JUMLAH_KLUSTER:
    label, pusat = kmeans(piksel, k)
    peta = np.zeros(len(piksel), dtype=int)
    print("\n=== %d kluster ===" % k)
    print("Kluster | piksel  persen |   R     G     B    NIR | piksel latih per kelas (H/Kb/Sw/LT) -> diberi
nama")
    for i in np.argsort(-pusat[:, 3]):                      # urut menurut NIR menurun
        n = int((label == i).sum())
        hitung = np.bincount(latih[(label == i) & (latih > 0)], minlength=5)[1:]
        kelas = int(hitung.argmax()) + 1 if hitung.sum() else 0     # suara terbanyak; 0 = tidak ada piksel
latih
        if kelas == 0:
            print("  PERINGATAN: kluster %d tidak punya piksel latih; kelasnya 0 (belum bernama)." % (i + 1))
        peta[label == i] = kelas
        r, g, b, nir = pusat[i]
        print("  %2d    | %6d  %5.1f%% | %5.1f %5.1f %5.1f %5.1f | %s -> %s" % (
            i + 1, n, 100.0 * n / label.size, r, g, b, nir, "/".join(str(int(v)) for v in hitung),
NAMA.get(kelas, "(belum bernama)")))
    print("Tersimpan:", simpan(peta.reshape(tinggi, lebar), "tak_terbimbing_k%d.tif" % k))
```

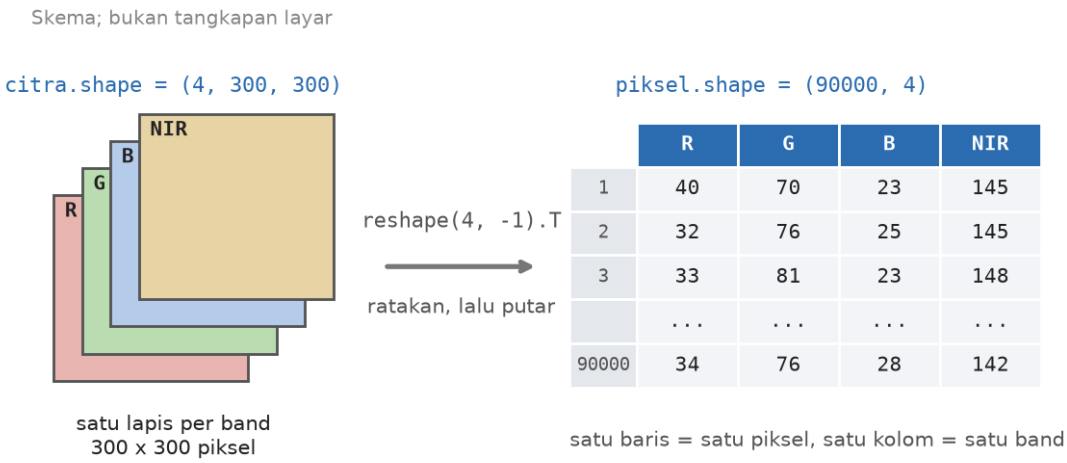
Penjelasan baris demi baris. Nomor baris mengikuti berkas di atas.

Blok A: baris 1 sampai 19, membaca citra menjadi tabel piksel.

Baris	Tulisan	Artinya
1-3	komentar	Judul, penulis, dan pesan: ubah dua jalur di bawah.
4-7	`import os`, `import numpy as np`, `import processing`, `from osgeo import gdal`	Panggil empat perkakas. Kata `as np` memberi NumPy nama pendek `np`.

Baris	Tulisan	Artinya
9-11	`PAKET_I2`, `HASIL`, `os.makedirs(...)`	`PAKET_I2` dibaca dari `I2_PAKET` (seperti 19.2), `HASIL` jalur hasil, dan pembuatan folder hasil. Sesuaikan jalurnya.
12	`JUMLAH_KLUSTER = [6, 10]`	Daftar jumlah kluster yang dicoba: dua percobaan.
13	`NAMA = {1: "Hutan", 2: "Kebun", 3: "Sawah", 4: "Lahan Terbuka"}`	Kamus (Bab 5): nomor kelas di sebelah kiri, nama kelas di sebelah kanan.
15	`ds = gdal.Open(...)`	Buka citra dengan GDAL.
16	`gt, proj = ds.GetGeoTransform(), ds.GetProjection()`	Dua nilai dalam satu baris. `gt` memuat letak dan ukuran piksel. `proj` adalah sistem koordinat. Keduanya disimpan supaya peta hasil punya lokasi yang sama dengan citra.
17	`citra = ds.ReadAsArray().astype("float64")`	Baca semua band menjadi array. Lalu ubah ke angka desimal agar hitungan jarak tidak meluap.
18	`_, tinggi, lebar = citra.shape`	Bentuk array adalah (4, 300, 300): empat band, tinggi, lebar. Tanda _ membuang jumlah band karena tidak dipakai.
19	`piksel = citra.reshape(4, -1).T`	Ratakan tiap band menjadi satu deret, lalu putar tabel. Hasilnya 90.000 baris (satu baris satu piksel) dan 4 kolom (satu kolom satu band).

Ilustrasi 19.8: Dari empat band menjadi tabel piksel



Blok B: baris 21 sampai 26, area latih menjadi raster.

Baris	Tulisan	Artinya
21	komentar	Area latih di sini hanya dipakai untuk MEMBERI NAMA kluster. Kelompoknya sendiri dibuat tanpa contoh.
22	`latih_tif = os.path.join(HASIL, "area_latih.tif")`	Tentukan nama berkas hasil rasterisasi.
23-25	`processing.run("gdal:rasterize", {...})`	Ubah poligon menjadi raster. `FIELD "Kode"` berarti piksel di dalam poligon bernilai isi kolom `Kode`. `UNITS` 1 dan `WIDTH`, `HEIGHT` 1.0 memilih ukuran piksel 1 meter. `EXTENT` adalah luas area, yang harus sama dengan citra. `NODATA`, `INIT`, dan `DATA_TYPE` membuat piksel di luar poligon bernilai 0, dengan tipe Byte.
26	`latih = gdal.Open(latih_tif).ReadAsArray().ravel()`	Baca raster itu, lalu `.ravel()` meratakannya menjadi satu deret 90.000 angka. Urutannya sama dengan urutan baris di `piksel`.

Blok C: baris 29 sampai 40, fungsi k-means.

Baris	Tulisan	Artinya
29	`def kmeans(x, k, ulang=30, seed=1):`	Resep k-means. `x` tabel piksel, `k` jumlah kluster, `ulang` batas putaran, `seed` kunci pengacak agar hasil sama tiap kali dijalankan.
30	docstring	Catatan satu kalimat: pusat awal dipilih acak, lalu dihitung ulang sampai stabil.
31	`rng = np.random.default_rng(seed)`	Siapkan pembuat angka acak dengan kunci `seed`.
32	`pusat = x[rng.choice(len(x), k, replace=False)]`	Pilih k piksel acak (tanpa kembar) sebagai pusat awal.
33	`for _ in range(ulang):`	Ulangi paling banyak 30 kali. Tanda `_` berarti nomor putaran tidak dipakai.
34	`jarak = ((x[:, None, :] - pusat[None, :, :]) ** 2).sum(axis=2)`	Hitung jarak kuadrat tiap piksel ke tiap pusat sekaligus. Bagian `None` hanya menata sumbu agar tiap piksel dibandingkan dengan tiap pusat. Hasilnya tabel 90.000 baris kali k kolom.
35	`label = jarak.argmin(axis=1)`	Untuk tiap piksel, ambil nomor pusat yang jaraknya terkecil. Itu "pos" yang ia ikuti.
36	`baru = np.array([x[label == i].mean(axis=0) if np.any(label == i) else pusat[i] for i in range(k)])`	Hitung pusat baru: rata-rata para anggota tiap kluster. Bila sebuah kluster kosong, pusat lamanya dipertahankan.
37-38	`if np.allclose(baru, pusat, atol=1e-3): break`	Bila pusat nyaris tidak bergeser (kurang dari 0,001), hentikan perulangan dengan `break`.

Baris	Tulisan	Artinya
39	`pusat = baru`	Pakai pusat baru untuk putaran berikutnya.
40	`return label, pusat`	Kembalikan dua hal: kluster tiap piksel dan pusat akhir.

Blok D: baris 43 sampai 51, fungsi simpan.

Baris	Tulisan	Artinya
43-44	`def simpan(arr, nama):` lalu `jalur = ...`	Resep menyimpan array menjadi berkas. Jalur diambil dari folder hasil dan `nama`.
45	`out = gdal.GetDriverByName("GTiff").Create(jalur, lebar, tinggi, 1, gdal.GDT_Byte)`	Buat berkas GeoTIFF kosong berukuran sama dengan citra, satu band, tipe Byte (angka 0 sampai 255).
46-47	`out.SetGeoTransform(gt)` dan `out.SetProjection(proj)`	Beri lokasi dan sistem koordinat yang sama dengan citra.
48	`out.GetRasterBand(1).WriteArray(arr.astype("uint8"))`	Tulis angka ke band pertama, sebagai bilangan bulat kecil.
49-50	`out.FlushCache()` dan `out = None`	Paksa data tertulis ke disk, lalu tutup berkas. Cara menutup berkas di GDAL adalah mengosongkan kotaknya.
51	`return jalur`	Kembalikan jalur berkas.

Blok E: baris 54 sampai 69, perulangan utama.

Baris	Tulisan	Artinya
54-56	`for k in JUMLAH_KLUSTER:` lalu `label, pusat = kmeans(piksel, k)`	Ulangi untuk k = 6, lalu k = 10. Jalankan k-means. `peta` dibuat sebagai deret 90.000 angka 0.
57-58	`print(...)`	Cetak judul dan kepala tabel.
59	`for i in np.argsort(-pusat[:, 3]):`	Telusuri kluster dari yang NIR-nya terbesar. `pusat[:, 3]` adalah kolom keempat (NIR). Tanda minus membalik urutan.
60	`n = int((label == i).sum())`	Hitung jumlah piksel di kluster `i`.
61	`hitung = np.bincount(latih[(label == i) & (latih > 0)], minlength=5)[1:]`	Ambil piksel milik kluster `i` yang juga ada di area latih, lalu hitung berapa dari tiap kelas 1 sampai 4. Hasilnya empat angka. Nama `hitung` di sini berarti "hasil hitung", bukan fungsi seperti di Skrip 19.1.
62	`kelas = int(hitung.argmax()) + 1 if hitung.sum() else 0`	Bila ada piksel latih: ambil kelas dengan jumlah terbanyak (suara terbanyak). Tanda `+ 1` karena nomor urut mulai dari 0. Bila tidak ada, kelasnya 0.
63-64	`if kelas == 0:` lalu `print("PERINGATAN: ...")`	Pengaman. Bila kluster tidak punya piksel latih (kelas 0), cetak peringatan berisi nomor kluster. Tanpa baris ini, kluster tanpa nama lolos tanpa tanda.
65	`peta[label == i] = kelas`	Isi peta untuk seluruh piksel kluster itu dengan kelasnya.
66	`r, g, b, nir = pusat[i]`	Pecah empat rata-rata band pusat kluster menjadi empat kotak.

Baris	Tulisan	Artinya
67-68	`print("  %2d ..." % (...))`	Cetak satu baris tabel. `"/".join(str(int(v)) for v in hitung)` menyambung empat angka jadi teks seperti `335/6/0/0`. `NAMA.get(kelas, "(belum bernama)")` mencari nama kelas, dan bila tidak ada (kelas 0), memakai teks cadangan.
69	`print("Tersimpan:", simpan(peta.reshape(tinggi, lebar), "tak_terbimbing_k%d.tif" % k))`	Bentuk peta kembali menjadi 300 x 300, simpan dengan nama yang memuat k, lalu cetak jalurnya.

Hasil. Layar menampilkan (nomor kluster boleh berbeda di komputer Anda):

```
HASIL
=== 6 kluster ===
Kluster | piksel  persen |   R     G     B    NIR | piksel latih per kelas (H/Kb/Sw/LT) -> diberi nama
   2    |  14051   15.6% |  44.8  86.0  41.0 153.4 | 335/6/0/0 -> Hutan
   3    |  21582   24.0% |  33.5  74.2  28.3 141.4 | 529/3/0/0 -> Hutan
   4    |  43309   48.1% |  60.7 104.1  48.1 119.3 | 0/855/864/0 -> Sawah
   5    |   3131    3.5% | 146.3 128.3 113.6 109.4 | 0/0/0/308 -> Lahan Terbuka
   6    |   5920    6.6% | 126.6 116.5  95.6  92.0 | 0/0/0/486 -> Lahan Terbuka
   1    |   2007    2.2% | 110.1 108.7  82.7  79.3 | 0/0/0/70 -> Lahan Terbuka
Tersimpan: C:/temp/hasil_i2\tak_terbimbing_k6.tif

=== 10 kluster ===
Kluster | piksel  persen |   R     G     B    NIR | piksel latih per kelas (H/Kb/Sw/LT) -> diberi nama
   3    |   7864    8.7% |  45.6  87.6  44.1 156.6 | 240/0/0/0 -> Hutan
   6    |  14670   16.3% |  37.4  77.8  31.9 149.0 | 372/0/0/0 -> Hutan
   9    |   9895   11.0% |  29.6  69.9  25.6 138.6 | 243/0/0/0 -> Hutan
   2    |  10658   11.8% |  60.9 103.5  52.1 134.4 | 1/351/14/0 -> Kebun
   5    |  12387   13.8% |  49.6  93.0  37.9 122.2 | 8/413/9/0 -> Kebun
  10    |  10580   11.8% |  71.8 114.8  56.7 119.6 | 0/57/366/0 -> Sawah
   7    |   3037    3.4% | 146.6 128.5 113.8 109.6 | 0/0/0/297 -> Lahan Terbuka
   1    |  13013   14.5% |  59.1 102.9  45.2 106.5 | 0/43/475/0 -> Sawah
   8    |   5685    6.3% | 127.1 116.9  96.0  92.6 | 0/0/0/453 -> Lahan Terbuka
   4    |   2211    2.5% | 112.4 109.0  84.6  78.3 | 0/0/0/114 -> Lahan Terbuka
Tersimpan: C:/temp/hasil_i2\tak_terbimbing_k10.tif
```

Cara membaca: tiap baris adalah satu kluster. Kolom `H/Kb/Sw/LT` adalah jumlah piksel latih Hutan, Kebun, Sawah, dan Lahan Terbuka di kluster itu. Pada 6 kluster, kluster 4 memuat 855 piksel latih Kebun dan 864 piksel latih Sawah, hampir sama banyak. Satu kluster tidak bisa menjadi dua kelas. Skrip memberinya nama Sawah dengan selisih tipis, jadi **Kebun hilang dari peta**. Pada 10 kluster, Kebun dan Sawah mulai terpisah. Harganya: pekerjaan memberi nama bertambah.

Bagian yang boleh diubah.

- **Baris 9 dan 10, jalur.** Ganti sesuai komputer Anda.
- **Baris 12, jumlah kluster.** Pada uji, `[4]` menghasilkan empat kluster di mana Kebun dan Sawah menyatu: `1 | 43361 48.2% ... | 0/853/864/0 -> Sawah`. Kebun tidak punya kluster sendiri.

- **Baris 29, seed=1.** Pada uji, `seed=2` dengan 6 kluster menghasilkan kluster yang berbeda, dan sekarang Kebun punya kluster sendiri (`10/581/11/0 -> Kebun`). Jadi hasil k-means bergantung pada titik awal acak.
- **Baris 29, ulang=30.** Batas putaran. Bila terlalu kecil, pusat belum stabil. Bila besar, tidak merugikan, sebab perulangan berhenti sendiri (`break`) saat pusat stabil.
- **Baris 13 dan 61, nama kelas.** Bila jumlah kelas berubah, ubah `NAMA`, lalu sesuaikan `minlength=5` (jumlah kelas ditambah 1) pada baris 61.
- **Baris 25, EXTENT.** Harus sama dengan luas citra Anda.

Jebakan.

- **Kluster tanpa piksel latih diberi kelas 0.** Pada uji dengan `JUMLAH_KLUSTER = [40]`, kluster 38 tidak punya satu pun piksel latih. Baris 63-64 mencetak `PERINGATAN: kluster 38 tidak punya piksel latih; kelasnya 0 (belum bernama).` tepat di atas barisnya, yang berbunyi `38 | 784 0.9% | 106.3 106.1 79.3 69.0 | 0/0/0/0 -> (belum bernama).` Peta hasil bernilai 0 di tempat itu. Jangan abaikan peringatan itu. Peta berisi 0 tidak bisa dinilai di Skrip 19.6, yang berhenti dengan pesan jelas. Perbaikannya: kurangi `JUMLAH_KLUSTER`, atau tambah area latih.
- **Luas EXTENT tidak cocok dengan citra.** Pada uji dengan luas 200 m x 200 m, skrip berhenti di baris 61 dengan `ValueError: operands could not be broadcast together with shapes (90000,) (40000,).` Artinya: citra punya 90.000 piksel, tetapi area latih hanya 40.000.
- **Nomor kluster tidak bermakna.** Kluster 2 di komputer Anda belum tentu kluster 2 di buku ini. Yang penting pola isinya.
- **Menamai kluster memakai area latih membuat cara ini setengah terbimbing.** Itu wajar, tetapi jangan pakai titik uji (Skrip 19.6) untuk memberi nama. Titik uji harus tetap "buta" sampai penilaian.

Latihan variasi.

1. Jalankan hanya dengan 4 kluster.
2. Ubah `seed` menjadi 2 dan jalankan hanya dengan 6 kluster. Apakah Kebun masih hilang?

Contoh jawaban nomor 1, baris 12 menjadi `JUMLAH_KLUSTER = [4]`. Layar menampilkan:

```
HASIL
=== 4 kluster ===
Kluster | piksel  persen |   R     G     B    NIR | piksel latih per kelas (H/Kb/Sw/LT) -> diberi nama
   2    |  14044   15.6% |  44.8  86.0  41.1 153.4 | 335/7/0/0 -> Hutan
   3    |  21637   24.0% |  33.5  74.2  28.3 141.4 | 529/4/0/0 -> Hutan
   1    |  43361   48.2% |  60.8 104.1  48.2 119.2 | 0/853/864/0 -> Sawah
   4    |  10958   12.2% | 129.5 118.5  98.6  94.6 | 0/0/0/864 -> Lahan Terbuka
```

Contoh jawaban nomor 2, baris 12 menjadi `JUMLAH_KLUSTER = [6]` dan baris 29 menjadi `def kmeans(x, k, ulang=30, seed=2):`. Layar menampilkan:

```
HASIL
=== 6 kluster ===
Kluster | piksel  persen |   R     G     B    NIR | piksel latih per kelas (H/Kb/Sw/LT) -> diberi nama
   5    |  13312   14.8% |  43.5  84.7  40.4 154.7 | 342/0/0/0 -> Hutan
```

```
  4    |  19397   21.6% |  32.6  73.1  27.8 142.6 | 512/0/0/0 -> Hutan
  6    |  15072   16.7% |  68.9 111.7  57.0 127.2 | 0/250/302/0 -> Sawah
  1    |  16833   18.7% |  51.8  95.1  40.6 124.8 | 10/581/11/0 -> Kebun
  2    |  14538   16.2% |  60.3 103.7  45.9 106.7 | 0/33/551/0 -> Sawah
  3    |  10848   12.1% | 129.8 118.6  98.9  94.4 | 0/0/0/864 -> Lahan Terbuka
```

Kebun muncul lagi, walau hanya karena titik awal acak berbeda. Pelajarannya: jangan percaya satu kali jalan. Ulangi dengan seed berbeda.

> **Pola yang bisa dipakai ulang:** Kerangka klasifikasi tak terbimbing: **baca citra menjadi tabel piksel** (baris 15-19), **kelompokkan** (`kmeans(piksel, k)`), **beri nama kluster** dengan area latih (baris 59-65), **simpan** peta (`simpan(...)`). Untuk citra lain, ganti **jalur citra**, **luas EXTENT**, **NAMA**, dan **JUMLAH_KLUSTER**.

Skrip 19.5: Ajari citra mengenal lahan

Berkas `i2_05_terbimbing.py`. Di dalam berkasnya tertulis "I2 Bab 5". Skrip ini padat, jadi yang sama dengan Skrip 19.4 dijelaskan singkat dan hanya yang baru yang dibedah.

Tujuan. Melatih komputer dengan area latih, lalu mengklasifikasi seluruh citra dengan dua aturan: jarak minimum dan kemiripan maksimum.

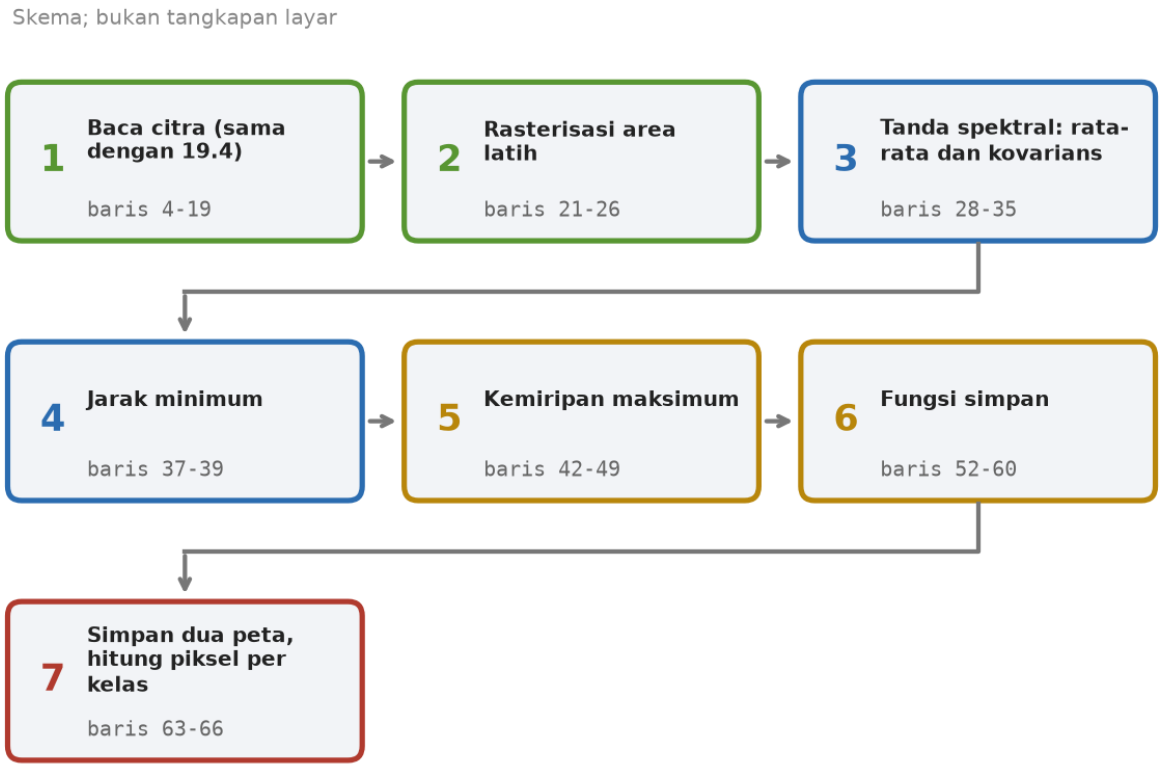
Bahan. Paket `paket-i2`: `Citra_Lahan.tif` dan `Area_Latih.gpkg`, sama seperti Skrip 19.4. Skrip dijalankan dari Editor Skrip QGIS.

Gagasan pokoknya: pada klasifikasi **terbimbing**, Anda memberi contoh lebih dulu. Komputer mempelajari ciri angka tiap kelas, lalu menerapkannya ke seluruh citra. Seperti melatih petugas baru: tunjukkan beberapa contoh tiap jenis lahan, lalu minta ia menilai sisanya.

Lima istilah baru untuk skrip ini:

- **Klasifikasi terbimbing**: klasifikasi dengan contoh dari Anda.
- **Tanda spektral**: ciri angka band satu kelas, yaitu rata-rata dan sebarannya.
- **Jarak minimum**: piksel masuk kelas yang rata-ratanya paling dekat.
- **Kemiripan maksimum**: piksel masuk kelas yang paling cocok, dengan memperhitungkan seberapa lebar sebaran kelas itu.
- **Kovarians**: ukuran sebaran band satu kelas, dan apakah band-band naik turun bersama. Kelas dengan sebaran sempit seperti sekelompok murid yang tingginya seragam. Kelas dengan sebaran lebar seperti murid dengan tinggi beragam.

Ilustrasi 19.9: Alur skrip 19.5



Skrip utuh:

```
PYTHON
# -*- coding: utf-8 -*-
# I2 Bab 5: klasifikasi terbimbing (jarak minimum dan kemiripan maksimum, ditulis dengan NumPy). Penulis: Badar Mubarok Yogaswara
# Jalankan di Python Console QGIS. Ubah dua jalur di bawah sesuai komputer Anda.
import os
import numpy as np
import processing
from osgeo import gdal

PAKET_I2 = os.environ.get("I2_PAKET", r"C:/KPH_Contoh/paket-i2")
HASIL = r"C:/temp/hasil_i2"
os.makedirs(HASIL, exist_ok=True)
NAMA = {1: "Hutan", 2: "Kebun", 3: "Sawah", 4: "Lahan Terbuka"}

citra_tif = os.path.join(PAKET_I2, "Citra_Lahan.tif")
ds = gdal.Open(citra_tif)
gt, proj = ds.GetGeoTransform(), ds.GetProjection()
citra = ds.ReadAsArray().astype("float64")
_, tinggi, lebar = citra.shape
piksel = citra.reshape(4, -1).T

# 1) ubah poligon area latih menjadi raster kelas (piksel di dalam poligon bernilai Kode, lainnya 0)
latih_tif = os.path.join(HASIL, "area_latih.tif")
processing.run("gdal:rasterize", {
    "INPUT": os.path.join(PAKET_I2, "Area_Latih.gpkg"), "FIELD": "Kode", "UNITS": 1, "WIDTH": 1.0, "HEIGHT": 1.0,
    "EXTENT": "312000,312300,9996000,9996300 [EPSG:32749]", "NODATA": 0, "DATA_TYPE": 0, "INIT": 0, "OUTPUT": latih_tif})
latih = gdal.Open(latih_tif).ReadAsArray().ravel()
```

```
# 2) tanda spektral: rata-rata dan sebaran (kovarians) tiap kelas dari piksel latihnya
rata, kov = {}, {}
print("Kelas          | piksel latih |   R     G     B    NIR (rata-rata)")
for k in NAMA:
    x = piksel[latih == k]
    assert len(x) > 4, "Area latih kelas %d terlalu sedikit" % k
    rata[k], kov[k] = x.mean(axis=0), np.cov(x.T)
    print("%-14s | %6d       | %5.1f %5.1f %5.1f %5.1f" % (NAMA[k], len(x), *rata[k]))

# 3a) jarak minimum: tiap piksel masuk kelas yang rata-ratanya paling dekat
jarak = np.stack([((piksel - rata[k]) ** 2).sum(axis=1) for k in NAMA], axis=1)
jarak_min = jarak.argmin(axis=1) + 1


# 3b) kemiripan maksimum: seperti jarak minimum, tetapi memperhitungkan sebaran tiap kelas (kovarians)
def skor(x, mu, c):
    d = x - mu
    return -0.5 * np.einsum("ij,jk,ik->i", d, np.linalg.inv(c), d) - 0.5 * np.log(np.linalg.det(c))


kemiripan = np.stack([skor(piksel, rata[k], kov[k]) for k in NAMA], axis=1)
maks_lik = kemiripan.argmax(axis=1) + 1


def simpan(arr, nama):
    jalur = os.path.join(HASIL, nama)
    out = gdal.GetDriverByName("GTiff").Create(jalur, lebar, tinggi, 1, gdal.GDT_Byte)
    out.SetGeoTransform(gt)
    out.SetProjection(proj)
    out.GetRasterBand(1).WriteArray(arr.reshape(tinggi, lebar).astype("uint8"))
    out.FlushCache()
    out = None
    print("Tersimpan:", jalur)


simpan(jarak_min, "terbimbing_jarak_min.tif")
simpan(maks_lik, "terbimbing_maks_lik.tif")
for nama, arr in (("jarak minimum", jarak_min), ("kemiripan maksimum", maks_lik)):
    print(nama, "-> piksel per kelas:", {NAMA[k]: int((arr == k).sum()) for k in NAMA})
```

Penjelasan baris demi baris. Baris 1-26 hampir sama dengan Skrip 19.4 (impor, jalur, membaca citra menjadi tabel `piksel`, merasterisasi area latih menjadi `latih`). Bedanya hanya tidak ada `JUMLAH_KLUSTER`. Yang baru mulai dari baris 28.

Blok A: baris 28 sampai 35, tanda spektral tiap kelas.

Baris	Tulisan	Artinya
29	`rata, kov = {}, {}`	Siapkan dua kamus kosong dalam satu baris: satu untuk rata-rata, satu untuk kovarians.
30	`print("Kelas ...")`	Cetak kepala tabel.
31	`for k in NAMA:`	Ulangi untuk tiap kelas. Menelusuri kamus berarti menelusuri kuncinya: 1, 2, 3, 4.
32	`x = piksel[latih == k]`	Pilih baris piksel yang lokasinya ada di area latih kelas `k`.
33	`assert len(x) > 4, "Area latih kelas %d terlalu sedikit" % k`	Pengaman. `assert` menghentikan skrip bila syaratnya salah (lihat Bab 10). Di sini syaratnya: kelas punya lebih dari 4 piksel latih. Tanpa itu, kovarians tidak bisa dihitung dan peta rusak tanpa kabar.

Baris	Tulisan	Artinya
34	`rata[k], kov[k] = x.mean(axis=0), np.cov(x.T)`	Hitung rata-rata tiap band, dan matriks kovarians 4 x 4. Simpan keduanya di kamus dengan kunci kelas.
35	`print("%-14s ..." % (NAMA[k], len(x), *rata[k]))`	Cetak satu baris: nama kelas, jumlah piksel latih, dan empat rata-rata. Tanda bintang `*` membuka daftar empat angka menjadi empat isian terpisah.

Blok B: baris 37 sampai 39, jarak minimum.

Baris	Tulisan	Artinya
38	`jarak = np.stack([((piksel - rata[k]) ** 2).sum(axis=1) for k in NAMA], axis=1)`	Untuk tiap kelas, hitung jarak kuadrat semua piksel ke rata-rata kelas itu. `np.stack` menyusun empat hasil menjadi satu tabel 90.000 baris kali 4 kolom.
39	`jarak_min = jarak.argmin(axis=1) + 1`	Pilih kelas dengan jarak terkecil. Tanda `+ 1` karena nomor kelas mulai dari 1, bukan 0.

Blok C: baris 42 sampai 49, kemiripan maksimum.

Baris	Tulisan	Artinya
43-45	`def skor(x, mu, c):` lalu rumus	Resep yang menghitung "skor kecocokan" tiap piksel dengan satu kelas. Anda tidak perlu menghafal rumusnya. Yang perlu diingat: makin besar skor, makin cocok. Rumus memakai kebalikan matriks kovarians (`np.linalg.inv`) dan determinannya (`np.linalg.det`), sehingga kelas yang sebarannya lebar tidak dirugikan.
48	`kemiripan = np.stack([skor(piksel, rata[k], kov[k]) for k in NAMA], axis=1)`	Hitung skor tiap piksel untuk tiap kelas, lalu susun menjadi tabel 90.000 kali 4.
49	`maks_lik = kemiripan.argmax(axis=1) + 1`	Pilih kelas dengan skor terbesar (`argmax`, kebalikan `argmin`).

Blok D: baris 52 sampai 66, menyimpan dan menghitung.

Baris	Tulisan	Artinya
52-60	`def simpan(arr, nama):`	Mirip fungsi `simpan` di Skrip 19.4, dengan dua beda. Pembentukan ulang menjadi 300 x 300 (`arr.reshape(tinggi, lebar)`) dilakukan di dalam fungsi. Dan fungsi mencetak sendiri jalur yang tersimpan, tanpa `return`.
63-64	`simpan(jarak_min, ...)` dan `simpan(maks_lik, ...)`	Simpan dua peta hasil.
65	`for nama, arr in (("jarak minimum", jarak_min), ("kemiripan maksimum", maks_lik)):`	Ulangi untuk dua pasangan: judul dan array hasil.
66	`print(nama, "-> piksel per kelas:", {NAMA[k]: int((arr == k).sum()) for k in NAMA})`	Cetak kamus berisi nama kelas dan jumlah pikselnya. Bentuk `{kunci: nilai for ...}` adalah kamus yang dibuat dengan satu baris perulangan.

Hasil. Layar menampilkan:

```
HASIL
Kelas          | piksel latih |   R     G     B    NIR (rata-rata)
Hutan          |    864       |  35.4  81.8  35.5 145.8
Kebun          |    864       |  55.7 101.8  46.9 126.5
Sawah          |    864       |  61.3 108.2  50.4 111.3
Lahan Terbuka  |    864       | 133.9 119.3 102.4  96.0
Tersimpan: C:/temp/hasil_i2\terbimbing_jarak_min.tif
Tersimpan: C:/temp/hasil_i2\terbimbing_maks_lik.tif
jarak minimum -> piksel per kelas: {'Hutan': 33484, 'Kebun': 23204, 'Sawah': 22470, 'Lahan Terbuka': 10842}
kemiripan maksimum -> piksel per kelas: {'Hutan': 33415, 'Kebun': 23194, 'Sawah': 22447, 'Lahan Terbuka':
10944}
```

Dari hasil ini Anda tahu: Kebun dan Sawah memang mirip. Rata-rata keduanya dekat (NIR 126,5 lawan 111,3), yang membuat keduanya menyatu di Skrip 19.4. Kali ini keempat kelas ada di peta. Selisih dua aturan kecil: 4.300 dari 90.000 piksel diberi kelas berbeda. Mana yang benar-benar lebih akurat? Jangan menilainya dari mata. Skrip 19.6 mengukurnya.

Bagian yang boleh diubah.

- **Baris 9 dan 10, jalur.** Ganti sesuai komputer Anda.
- **Baris 12, NAMA.** Ubah nama kelas atau jumlahnya, sesuai kolom `Kode` pada area latih Anda.
- **Baris 25, EXTENT.** Harus sama dengan luas citra.
- **Baris 33, batas minimum piksel latih.** Angka 4 adalah batas bawah agar kovarians bisa dihitung. Naikkan bila Anda ingin menuntut area latih yang lebih banyak.
- **Baris 63-64, nama berkas.** Ganti nama berkas hasil.
- **Baris 65, daftar pasangan.** Tambahkan metode lain di sini bila Anda menulisnya.

Jebakan.

- **Area latih kosong atau terlalu sedikit.** Baris 33 menghentikan skrip dengan `AssertionError: Area latih kelas 4 terlalu sedikit`. Saya mengosongkan area latih kelas Lahan Terbuka pada salinan skrip untuk mengujinya. Tanpa baris pengaman itu, skrip jalan terus: layar menampilkan beberapa `RuntimeWarning` (yang pertama `Mean of empty slice`) dan baris `Lahan Terbuka | 0 | nan nan nan nan`, dan hasilnya `{'Hutan': 0, 'Kebun': 0, 'Sawah': 0, 'Lahan Terbuka': 90000}`, yaitu seluruh peta menjadi satu kelas. Bila galat ini muncul, periksa kolom `Kode` pada area latih dan `EXTENT`-nya. Jangan menghapus baris 33 supaya galat semacam ini tidak diam-diam lolos.
- **Luas EXTENT tidak cocok.** Sama seperti Skrip 19.4: bentuk array tidak sama dan skrip berhenti dengan `ValueError`.
- **Area latih bukan titik uji.** Jangan menilai peta dengan piksel yang sama dengan yang dipakai melatih. Hasilnya terlalu bagus. Skrip 19.6 memakai titik uji yang terpisah.
- **Impor tak terpakai.** Skrip mengimpor `processing` dan memakainya hanya untuk rasterisasi. Tidak ada yang rusak. Ini hanya contoh membaca kode dengan kritis.

Latihan variasi.

1. Cetak jumlah piksel per kelas sebagai persen, bukan jumlah.
2. Cetak berapa piksel yang kelasnya berbeda antara dua metode.

Contoh jawaban nomor 1, baris 66 menjadi:

```
PYTHON
    print(nama, "-> persen per kelas:", {NAMA[k]: round(100.0 * int((arr == k).sum()) / arr.size, 1) for k in
NAMA})
```

Layar menampilkan `jarak minimum -> persen per kelas: {'Hutan': 37.2, 'Kebun': 25.8, 'Sawah': 25.0, 'Lahan Terbuka': 12.0}` dan `kemiripan maksimum -> persen per kelas: {'Hutan': 37.1, 'Kebun': 25.8, 'Sawah': 24.9, 'Lahan Terbuka': 12.2}`. Kata `int(...)` di tengah penting. Tanpa itu, layar menampilkan `np.float64(37.2)` di tiap nilai.

Contoh jawaban nomor 2, satu baris di akhir skrip:

```
PYTHON
print("Piksel yang kelasnya beda antara dua metode:", int((jarak_min != maks_lik).sum()))
```

Layar menampilkan `Piksel yang kelasnya beda antara dua metode: 4300`.

> **Pola yang bisa dipakai ulang:** Klasifikasi terbimbing selalu tiga langkah: **tanda spektral** tiap kelas dari area latih, **skor** tiap piksel terhadap tiap kelas, lalu **argmin atau argmax** untuk memilih kelas. Untuk data baru, ganti **jalur citra dan area latih**, **EXTENT**, dan **NAMA**. Aturan penentu (jarak minimum atau kemiripan maksimum) boleh dibiarkan.

Skrip 19.6: Seberapa benar peta ini?

Berkas `i2_06_uji_akurasi.py`. Di dalam berkasnya tertulis "I2 Bab 6". Skrip ini memakai empat peta dari Skrip 19.4 dan 19.5, jadi jalankan keduanya lebih dulu.

Tujuan. Menilai empat peta hasil klasifikasi dengan 100 titik uji: membuat matriks kesalahan, menghitung akurasi keseluruhan dan kappa, merinci akurasi tiap kelas, lalu membandingkannya dengan alat GRASS `r.kappa`.

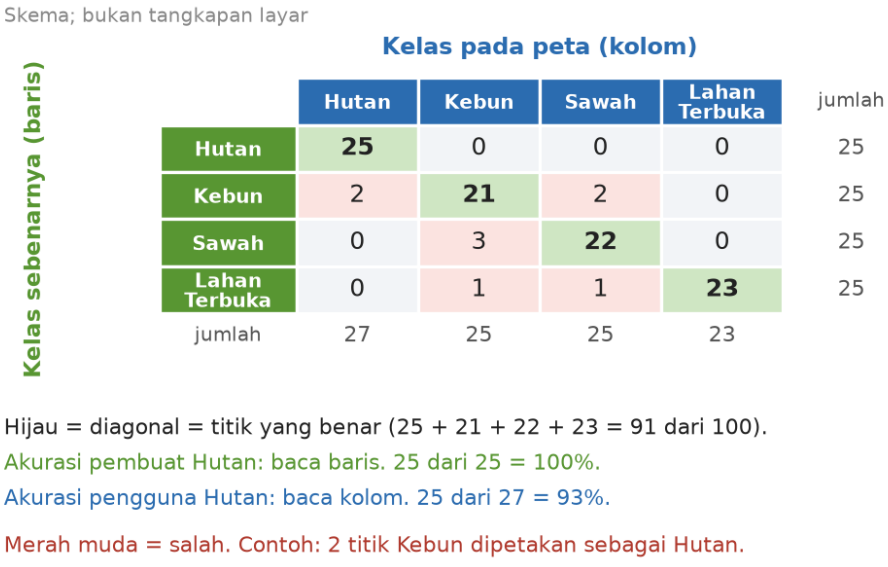
Bahan. `Titik_Uji.gpkg` dari `paket-i2` (100 titik, 25 per kelas, kolom `Kode_Acuan` berisi kelas sebenarnya dari 1 sampai 4), serta empat peta di folder hasil: `tak_terbimbing_k6.tif`, `tak_terbimbing_k10.tif`, `terbimbing_jarak_min.tif`, dan `terbimbing_maks_lik.tif`. Bagian akhir skrip memanggil alat GRASS. Alat itu harus aktif di Processing QGIS Anda. [CEK]

Gagasan pokoknya seperti ujian. Soal ujian (titik uji) harus berbeda dari soal latihan (area latih). Kunci jawaban (kelas sebenarnya di lapangan) harus bisa dipercaya. Dari perbandingan jawaban peta dan kunci, muncul satu tabel yang disebut matriks kesalahan.

Lima istilah baru untuk skrip ini:

- **Titik uji**: titik yang kelas sebenarnya Anda ketahui dan tidak ikut melatih.
- **Matriks kesalahan**: tabel kelas sebenarnya (baris) lawan kelas pada peta (kolom).
- **Akurasi keseluruhan**: titik yang benar dibagi semua titik.
- **Akurasi pembuat dan pengguna**: ketepatan dilihat dari sisi lapangan (per baris) dan dari sisi peta (per kolom).
- **Kappa**: kecocokan setelah dikurangi kecocokan karena kebetulan.

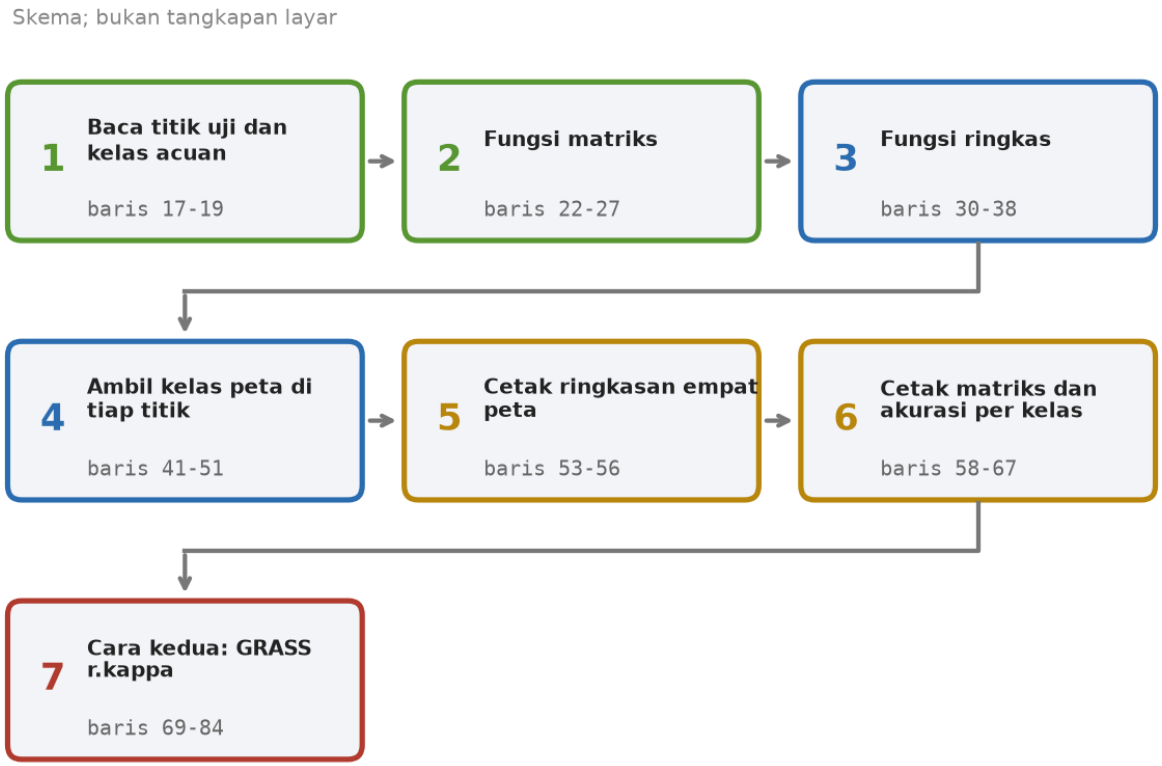
Ilustrasi 19.10: Cara membaca matriks kesalahan



Rumus yang dipakai skrip, dengan angka dari gambar:

- **Akurasi keseluruhan** = jumlah diagonal dibagi jumlah semua titik = (25 + 21 + 22 + 23) dibagi 100 = 91%.
- **Akurasi pembuat** kelas X = diagonal dibagi jumlah baris X. Untuk Hutan: 25 dibagi 25 = 100%.
- **Akurasi pengguna** kelas X = diagonal dibagi jumlah kolom X. Untuk Hutan: 25 dibagi 27 = 93%.
- **Kappa** = (akurasi keseluruhan dikurangi p) dibagi (1 dikurangi p), dengan p adalah kecocokan yang diharapkan secara kebetulan. Di sini p = 0,25, jadi kappa = (0,91 - 0,25) dibagi 0,75 = 0,88.

Ilustrasi 19.11: Alur skrip 19.6



Skrip utuh:

PYTHON

```
# -*- coding: utf-8 -*-
# I2 Bab 6: uji akurasi (matriks kesalahan, akurasi keseluruhan, kappa). Penulis: Badar Mubarok Yogaswara
# Jalankan setelah skrip bab 4 dan bab 5 (memakai peta hasilnya di folder hasil).
import os
import numpy as np
import processing
from qgis.core import QgsRasterLayer, QgsVectorLayer

PAKET_I2 = os.environ.get("I2_PAKET", r"C:/KPH_Contoh/paket-i2")
HASIL = r"C:/temp/hasil_i2"
NAMA = {1: "Hutan", 2: "Kebun", 3: "Sawah", 4: "Lahan Terbuka"}
PETA = [("Tak terbimbing, 6 kluster", "tak_terbimbing_k6.tif"),
        ("Tak terbimbing, 10 kluster", "tak_terbimbing_k10.tif"),
        ("Terbimbing, jarak minimum", "terbimbing_jarak_min.tif"),
        ("Terbimbing, kemiripan maksimum", "terbimbing_maks_lik.tif")]
```

```
titik = QgsVectorLayer(os.path.join(PAKET_I2, "Titik_Uji.gpkg") + "|layername=Titik_Uji", "Titik_Uji", "ogr")
acuan = np.array([f["Kode_Acuan"] for f in titik.getFeatures()])
print("Jumlah titik uji:", len(acuan), "| per kelas:", {NAMA[k]: int((acuan == k).sum()) for k in NAMA})


def matriks(acuan, hasil):
    """Baris = kelas acuan (lapangan), kolom = kelas hasil peta."""
    m = np.zeros((4, 4), int)
    for a, h in zip(acuan, hasil):
        m[a - 1, h - 1] += 1
    return m


def ringkas(m):
    n = m.sum()
    oa = np.trace(m) / n
    pe = (m.sum(axis=1) * m.sum(axis=0)).sum() / n ** 2        # kecocokan yang diharapkan karena kebetulan
    kappa = (oa - pe) / (1 - pe)
    with np.errstate(divide="ignore", invalid="ignore"):       # kolom kosong (kelas tidak pernah muncul di
peta) menghasilkan nan
        pa = np.diag(m) / m.sum(axis=1)                        # akurasi pembuat: dari yang benar di lapangan,
berapa yang terpetakan benar
        ua = np.diag(m) / m.sum(axis=0)                        # akurasi pengguna: dari yang dipetakan kelas X,
berapa yang benar
    return oa, kappa, pa, ua


hasil_semua = {}
for judul, berkas in PETA:
    lyr = QgsRasterLayer(os.path.join(HASIL, berkas), "peta")
    sampel = processing.run("native:rastersampling", {"INPUT": titik, "RASTERCOPY": lyr, "COLUMN_PREFIX":
"Peta_", "OUTPUT": "TEMPORARY_OUTPUT"})["OUTPUT"]
    pasang = [(a, f["Peta_1"]) for a, f in zip(acuan, sampel.getFeatures())]
    hasil = np.array([int(h) for a, h in pasang if h is not None])
    acuan_ok = np.array([a for a, h in pasang if h is not None])
    if len(hasil) < len(acuan):
        print("PERINGATAN: %d titik uji di luar raster dilewati" % (len(acuan) - len(hasil)))
    assert hasil.min() >= 1, "Peta berisi kelas 0 (kluster tanpa nama)"
    hasil_semua[judul] = (matriks(acuan_ok, hasil), ringkas(matriks(acuan_ok, hasil)))

print("\nRingkasan:")
print("%-34s | akurasi keseluruhan | kappa" % "Peta")
for judul, (m, (oa, kappa, pa, ua)) in hasil_semua.items():
    print("%-34s | %5.0f%% (%d dari %d)    | %.2f" % (judul, 100 * oa, np.trace(m), m.sum(), kappa))

judul = "Terbimbing, kemiripan maksimum"
m, (oa, kappa, pa, ua) = hasil_semua[judul]
print("\nMatriks kesalahan: %s (baris = acuan, kolom = hasil peta)" % judul)
print("%-14s" % "" + "".join("%14s" % NAMA[k] for k in NAMA) + "   jumlah")
for i, k in enumerate(NAMA):
    print("%-14s" % NAMA[k] + "".join("%14d" % v for v in m[i]) + "%8d" % m[i].sum())
print("%-14s" % "jumlah" + "".join("%14d" % v for v in m.sum(axis=0)))
print("\nKelas          | akurasi pembuat | akurasi pengguna")
for i, k in enumerate(NAMA):
    print("%-14s | %6.0f%%          | %6.0f%%" % (NAMA[k], 100 * pa[i], 100 * ua[i]))

# Cara kedua (alat GRASS di Processing QGIS): r.kappa membandingkan dua raster, yaitu peta hasil dan raster
acuan.
# Raster acuan dibuat dari titik uji: piksel yang tidak ada titiknya bernilai 0 (kosong) dan diabaikan.
acuan_tif = os.path.join(HASIL, "acuan_titik.tif")
processing.run("gdal:rasterize", {
    "INPUT": os.path.join(PAKET_I2, "Titik_Uji.gpkg"), "FIELD": "Kode_Acuan", "UNITS": 1, "WIDTH": 1.0,
"HEIGHT": 1.0,
    "EXTENT": "312000,312300,9996000,9996300 [EPSG:32749]", "NODATA": 0, "DATA_TYPE": 0, "INIT": 0, "OUTPUT":
acuan_tif})
laporan = os.path.join(HASIL, "kappa_maks_lik.txt")
```

```
processing.run("grass:r.kappa", {"classification": os.path.join(HASIL, "terbimbing_maks_lik.tif"), "reference":
acuan_tif,
                                 "title": "Uji akurasi", "-h": False, "-w": False, "output": laporan})
with open(laporan) as f:
    baris = f.read().splitlines()
print("\nLaporan r.kappa (bagian ringkasan):")
for i, b in enumerate(baris):
    if b.startswith(("Kappa\t", "Obs Correct")):
        print(b)
        print(baris[i + 1])
```

Penjelasan baris demi baris. Skrip ini 84 baris, jadi dibagi empat blok. Nomor baris mengikuti berkas di atas.

Blok A: baris 1 sampai 19, persiapan.

Baris	Tulisan	Artinya
1-3	komentar	Baris 3 mengingatkan: jalankan setelah Skrip 19.4 dan 19.5.
4-7	`import os`, `import numpy as np`, `import processing`, `from qgis.core import ...`	Panggil perkakas yang dipakai.
9-11	`PAKET_I2`, `HASIL`, `NAMA`	`PAKET_I2` dibaca dari `I2_PAKET`, `HASIL` jalur hasil, dan kamus nama kelas seperti di Skrip 19.4. Sesuaikan jalurnya.
12-15	`PETA = [("Tak terbimbing, 6 kluster", "tak_terbimbing_k6.tif"), ...]`	Daftar berisi empat pasangan: judul peta dan nama berkasnya. Daftar ini yang akan dinilai satu per satu.
17	`titik = QgsVectorLayer(..., "Titik_Uji", "ogr")`	Buka layer titik uji. Pola sumber datanya sama dengan Skrip 19.3.
18	`acuan = np.array([f["Kode_Acuan"] for f in titik.getFeatures()])`	Ambil kolom `Kode_Acuan` dari semua titik menjadi satu array. Urutannya sama dengan urutan titik.
19	`print("Jumlah titik uji:", len(acuan), ..., {...})`	Cetak jumlah titik, lalu jumlah titik tiap kelas lewat kamus yang dibuat dalam satu baris. Teks di dalam kutip hanyalah penanda.

Blok B: baris 22 sampai 38, dua fungsi hitung.

Baris	Tulisan	Artinya
22-23	`def matriks(acuan, hasil):`	Resep yang membuat matriks. Docstring mengingatkan: baris adalah kelas acuan (lapangan), kolom adalah kelas hasil peta.
24	`m = np.zeros((4, 4), int)`	Siapkan tabel 4 kali 4 berisi bilangan bulat 0.
25-26	`for a, h in zip(acuan, hasil):` lalu `m[a - 1, h - 1] += 1`	`zip` menjodohkan titik pertama dengan titik pertama, dan seterusnya. Untuk tiap pasangan, tambah satu di sel (baris `a`, kolom `h`). Dikurangi 1 karena nomor sel mulai dari 0, sedangkan nomor kelas mulai dari 1.
27	`return m`	Kembalikan matriksnya.

Baris	Tulisan	Artinya
30-31	`def ringkas(m):` lalu `n = m.sum()`	Resep yang merangkum matriks. `n` adalah jumlah semua titik.
32	`oa = np.trace(m) / n`	`np.trace` menjumlahkan diagonal. Dibagi `n` memberi akurasi keseluruhan.
33-34	`pe = ...` lalu `kappa = (oa - pe) / (1 - pe)`	`pe` adalah kecocokan yang diharapkan karena kebetulan (p di rumus). Lalu hitung kappa.
35-37	`with np.errstate(divide="ignore", invalid="ignore"):`	Blok dengan aturan khusus: jangan memberi peringatan bila ada pembagian dengan nol. Kolom kosong (kelas yang tidak pernah muncul di peta) akan membagi dengan nol, dan hasilnya `nan`. `pa` adalah akurasi pembuat (diagonal dibagi jumlah baris). `ua` adalah akurasi pengguna (diagonal dibagi jumlah kolom).
38	`return oa, kappa, pa, ua`	Kembalikan empat hasil sekaligus.

Blok C: baris 41 sampai 67, menilai tiap peta dan mencetak.

Baris	Tulisan	Artinya
41-43	`hasil_semua = {}` lalu `for judul, berkas in PETA:` lalu `lyr = QgsRasterLayer(...)`	Siapkan kamus kosong untuk menampung hasil. Untuk tiap peta di daftar, buka rasternya.
44	`sampel = processing.run("native:rastersampling", {...})["OUTPUT"]`	Ambil nilai peta di tiap titik uji. `COLUMN_PREFIX "Peta_"` membuat kolom baru bernama `Peta_1` (awalan ditambah nomor band). `TEMPORARY_OUTPUT` berarti hasil sementara di memori, bukan berkas.
45	`pasang = [(a, f["Peta_1"]) for a, f in zip(acuan, sampel.getFeatures())]`	Jodohkan kelas acuan tiap titik dengan kelas peta di titik itu. Hasilnya daftar pasangan. Bila sebuah titik jatuh di luar raster, kelas petanya `None` (tidak ada nilai, lihat Bab 7).
46	`hasil = np.array([int(h) for a, h in pasang if h is not None])`	Ambil kelas peta dari pasangan, tetapi hanya yang bukan `None`. Kata `if` di ujung daftar menyaring: pasangan yang kosong dilewati.
47	`acuan_ok = np.array([a for a, h in pasang if h is not None])`	Saringan yang sama untuk kelas acuan. Hasilnya `acuan_ok` selalu sepanjang `hasil`, sehingga keduanya tetap berpasangan.
48-49	`if len(hasil) < len(acuan):` lalu `print("PERINGATAN: ...")`	Bila ada titik yang terbuang karena di luar raster, cetak berapa banyak. Titik yang dilewati tidak hilang diam-diam.
50	`assert hasil.min() >= 1, "Peta berisi kelas 0 (kluster tanpa nama)"`	Pengaman. `assert` menghentikan skrip bila syaratnya salah (lihat Bab 10). Syaratnya: kelas terkecil di peta minimal 1. Nilai 0 berarti ada kluster yang belum bernama (lihat Skrip 19.4).
51	`hasil_semua[judul] = (matriks(acuan_ok, hasil), ringkas(matriks(acuan_ok, hasil)))`	Simpan satu pasangan di kamus: matriksnya, dan ringkasannya. Fungsi `matriks` dipanggil dua kali untuk hasil yang sama. Boleh disederhanakan, tetapi tidak salah.
53-56	`for judul, (m, (oa, kappa, pa, ua)) in hasil_semua.items():`	Cetak ringkasan tiap peta. Bagian dalam kurung membongkar pasangan bersarang menjadi lima kotak. Format `%5.0f%%` mencetak persen tanpa desimal, dan `%%` mencetak tanda persen.

Baris	Tulisan	Artinya
58-59	`judul = "Terbimbing, kemiripan maksimum"` lalu `m, (...) = hasil_semua[judul]`	Pilih satu peta untuk dirinci, lalu ambil matriks dan ringkasannya dari kamus.
60-64	`print(...)` dan `for i, k in enumerate(NAMA):`	Cetak matriks sebagai tabel. `enumerate(NAMA)` memberi nomor baris `i` (mulai 0) dan nomor kelas `k` (mulai 1). `"%14d" % v` memberi lebar 14 huruf pada tiap angka. Baris terakhir mencetak jumlah per kolom.
65-67	`print(...)` dan `for i, k in enumerate(NAMA):`	Cetak akurasi pembuat dan pengguna tiap kelas.

Blok D: baris 69 sampai 84, cara kedua dengan GRASS.

Baris	Tulisan	Artinya
69-70	komentar	Cara kedua memakai alat GRASS `r.kappa`, yang membandingkan dua raster: peta hasil dan raster acuan.
71-74	`processing.run("gdal:rasterize", {...})`	Ubah titik uji menjadi raster acuan. Piksel tanpa titik bernilai 0, dan diabaikan oleh `r.kappa`. Pengaturannya sama dengan di Skrip 19.4.
75	`laporan = os.path.join(HASIL, "kappa_maks_lik.txt")`	Nama berkas teks untuk laporan.
76-77	`processing.run("grass:r.kappa", {...})`	Jalankan alat. `classification` adalah peta yang dinilai, `reference` adalah raster acuan, dan `output` adalah berkas laporan.
78-79	`with open(laporan) as f:` lalu `baris = f.read().splitlines()`	Buka berkas teks, baca seluruhnya, lalu pecah menjadi daftar baris. Blok `with` menutup berkas sendiri (Bab 9).
80-84	`print(...), for i, b in enumerate(baris):` lalu `if b.startswith((..., ...)):`	Telusuri baris laporan. Bila sebuah baris diawali kata "Kappa" yang diikuti karakter tab, atau diawali "Obs Correct", cetak baris itu dan baris sesudahnya (`baris[i + 1]`), tempat angkanya berada. Dua awalan ditulis dalam kurung sebagai satu pasangan. Karakter tab ditulis `\t` (garis miring terbalik lalu huruf t), jadi tidak bisa berubah menjadi spasi saat disalin.

Hasil. Layar menampilkan:

```
HASIL
Jumlah titik uji: 100 | per kelas: {'Hutan': 25, 'Kebun': 25, 'Sawah': 25, 'Lahan Terbuka': 25}

Ringkasan:
Peta                              | akurasi keseluruhan | kappa
Tak terbimbing, 6 kluster         |    75% (75 dari 100)    | 0.67
Tak terbimbing, 10 kluster        |    98% (98 dari 100)    | 0.97
Terbimbing, jarak minimum         |    88% (88 dari 100)    | 0.84
Terbimbing, kemiripan maksimum    |    91% (91 dari 100)    | 0.88

Matriks kesalahan: Terbimbing, kemiripan maksimum (baris = acuan, kolom = hasil peta)
                       Hutan         Kebun         Sawah Lahan Terbuka   jumlah
Hutan                     25             0             0             0       25
```

```
Kebun                     2            21             2             0      25
Sawah                     0             3            22             0      25
Lahan Terbuka             0             1             1            23      25
jumlah                   27            25            25            23

Kelas          | akurasi pembuat | akurasi pengguna
Hutan          |    100%         |     93%
Kebun          |     84%         |     84%
Sawah          |     88%         |     88%
Lahan Terbuka  |     92%         |    100%

Laporan r.kappa (bagian ringkasan):
Kappa         Kappa Variance   MCC
0.880000      0.001455 0.880470
Obs Correct   Total Obs        % Observed Correct
91            100              91.000000
```

Dari hasil ini Anda tahu empat hal. Pertama, peta kemiripan maksimum benar di 91 dari 100 titik, dengan kappa 0,88. Kedua, hitungan NumPy dan `r.kappa` memberi angka sama (kappa 0,88; 91 dari 100), jadi dua cara saling membenarkan. Ketiga, Kebun dan Sawah paling sering tertukar (baris Kebun: 2 jadi Hutan, 2 jadi Sawah). Keempat, peta 10 kluster tampak paling baik (98%), tetapi ada tiga catatan di bawah.

Hati-hati membaca angka.

- **Satu angka keseluruhan bisa menipu.** Peta 6 kluster mendapat 75%, tetapi Kebun hilang dari peta. Akurasi pembuat Kebun 0%. Dari 25 titik Kebun, 18 menjadi Sawah dan 7 menjadi Hutan. [terbukti: lihat Latihan variasi]
- **Selisih kecil belum tentu nyata.** Hanya ada 100 titik, jadi satu titik berarti satu poin persen. Selang kepercayaan 95% yang kasar (rumus p ± 1,96 x akar dari p(1-p)/n) untuk 91% kira-kira 85% sampai 97%. Untuk 98%, kira-kira 95% sampai 100%. Kedua selang itu bertumpang tindih. [dihitung dari rumus]
- **Peta 10 kluster diberi nama dengan area latih.** Caranya setengah terbimbing. Data ini juga sintetis dan bersih. Pada data sungguhan, selisih tak terbimbing dan terbimbing jarang sekecil ini.

Bagian yang boleh diubah.

- **Baris 9 dan 10, jalur.** Ganti sesuai komputer Anda.
- **Baris 12-15, daftar PETA.** Tambah atau hapus pasangan untuk menilai peta lain, asal berkasnya ada di folder hasil dan kelasnya bernomor 1 sampai 4.
- **Baris 58, judul peta yang dirinci.** Harus persis sama dengan salah satu judul di `PETA`. Pada uji, `"Terbimbing, jarak minimum"` menghasilkan matriks dengan Kebun: 5 titik jadi Hutan dan 20 benar.
- **Baris 11 dan 24, jumlah kelas.** Bila jumlah kelas bukan empat, ubah `NAMA` dan ukuran `(4, 4)` di baris 24.
- **Baris 71-74, luas EXTENT.** Harus sama dengan luas citra.

Jebakan.

- **Nilai 0 pada peta menghentikan skrip.** Peta tak terbimbing bisa bernilai 0 bila sebuah kluster tidak punya piksel latih (lihat Jebakan Skrip 19.4). Baris 50 menangkapnya: skrip berhenti dengan `AssertionError: Peta berisi kelas 0 (kluster tanpa nama).` Pengaman ini perlu karena tanpanya hasilnya diam-diam salah. Pada baris 26, `m[a - 1, h - 1]` untuk `h` = 0 berarti kolom nomor -1. Python menghitung -1 dari belakang, yaitu kolom terakhir, Lahan Terbuka. Titik itu terhitung salah ke Lahan Terbuka, tanpa pesan galat. Saya mengujinya pada peta buatan yang menjadikan kelas Hutan bernilai 0, dengan baris 50 dilepas. Seluruh 25 titik Hutan masuk kolom Lahan Terbuka:

HASIL

```
Matriks kesalahan: Uji nol (baris = acuan, kolom = hasil peta)
                      Hutan         Kebun         Sawah Lahan Terbuka   jumlah
Hutan                     0             0             0            25       25
Kebun                     0            21             2             2       25
Sawah                     0             3            22             0       25
Lahan Terbuka             0             1             1            23       25
jumlah                    0            25            25            50
```

Kolom Hutan kosong, dan akurasi pengguna Hutan tercetak `nan%`. Petunjuknya: ada kolom yang kosong, atau jumlah kolom Lahan Terbuka membengkak. Dengan baris 50 terpasang, Anda tidak sampai ke tampilan ini.

- **Titik di luar raster dilewati, bukan dihitung.** Bila ada titik uji yang jatuh di luar peta, `Peta_1` kosong (`None`). Baris 45-47 membuang titik itu, dan baris 48-49 mencetak `PERINGATAN: 1 titik uji di luar raster dilewati`. Saya mengujinya dengan menambah satu titik di luar area. Skrip selesai, dan hasilnya sama dengan 100 titik asli. Peringatan muncul satu kali per peta. Bacalah peringatan itu: titik yang terbuang berarti penilaian memakai titik lebih sedikit.
- **Alat GRASS tidak ada.** Bagian baris 69-84 butuh penyedia GRASS aktif di Processing. Bila tidak ada, bagian itu gagal, sedangkan cetakan sebelum baris 69 tetap tampil. [CEK di QGIS Anda]
- **Karakter tab ditulis `\t`.** Pada baris 82, awalan `"Kappa\t"` memakai `\t` untuk karakter tab, sebab laporan GRASS memisahkan kolom dengan tab. Bila awalan ini berisi tab asli, lalu editor mengubah tab menjadi spasi saat Anda menyalin, baris Kappa tidak ditemukan dan hanya baris `Obs Correct` yang tercetak, tanpa galat. Penulisan `\t` aman dari masalah itu. Jangan mengganti `\t` dengan spasi.
- **Berkas laporan dibuka tanpa encoding.** Di baris 78 tidak ada pengaturan huruf. Untuk laporan berisi angka dan huruf Latin, ini aman.

Latihan variasi.

1. Cetak matriks kesalahan peta 6 kluster, bukan peta kemiripan maksimum.
2. Cetak akurasi pembuat Kebun untuk keempat peta.

Contoh jawaban nomor 1, baris 58 menjadi:

PYTHON

```
judul = "Tak terbimbing, 6 kluster"
```

Layar menampilkan matriks berikut. Baris Kebun menunjukkan masalahnya: 7 titik jadi Hutan, 0 benar, dan 18 jadi Sawah.

HASIL

```
Matriks kesalahan: Tak terbimbing, 6 kluster (baris = acuan, kolom = hasil peta)
                        Hutan         Kebun         Sawah Lahan Terbuka   jumlah
Hutan                      25             0             0             0       25
Kebun                       7             0            18             0       25
Sawah                       0             0            25             0       25
Lahan Terbuka               0             0             0            25       25
jumlah                     32             0            43            25
```

Contoh jawaban nomor 2, tambahkan di akhir bagian cetak (sebelum baris 69):

PYTHON

```
for judul, (m, (oa, kappa, pa, ua)) in hasil_semua.items():
    print("%-34s akurasi pembuat Kebun: %3.0f%%" % (judul, 100 * pa[1]))
```

Layar menampilkan `0%` untuk peta 6 kluster, `100%` untuk 10 kluster, `80%` untuk jarak minimum, dan `84%` untuk kemiripan maksimum. Angka `pa[1]` adalah isi kedua, yaitu kelas Kebun (hitungan mulai dari 0).

> **Pola yang bisa dipakai ulang:** Uji akurasi adalah tiga langkah: **ambil kelas peta di titik uji** (`native:rastersampling`), **isi matriks** (`matriks(acuan_ok, hasil)`), **rangkum** (`ringkas(m)`). Untuk peta lain, ganti **daftar PETA**, **berkas titik uji dan nama kolom acuannya**, dan **NAMA**. Baca selalu peringatan titik yang dilewati, dan cek apakah ada kolom kosong di matriks.

Skrip 19.7: Lahan terbuka di lereng curam

Berkas `i2_07_terpadu.py`. Di dalam berkasnya tertulis "I2 Bab 6 (latihan terpadu)". Skrip ini memakai `terbimbing_maks_lik.tif` (Skrip 19.5) dan `slope_persen.tif` (Skrip 19.2).

Tujuan. Menghitung luas lahan terbuka yang berada di lereng curam (di atas 25 persen), dan berapa persen dari seluruh lahan terbuka pada peta.

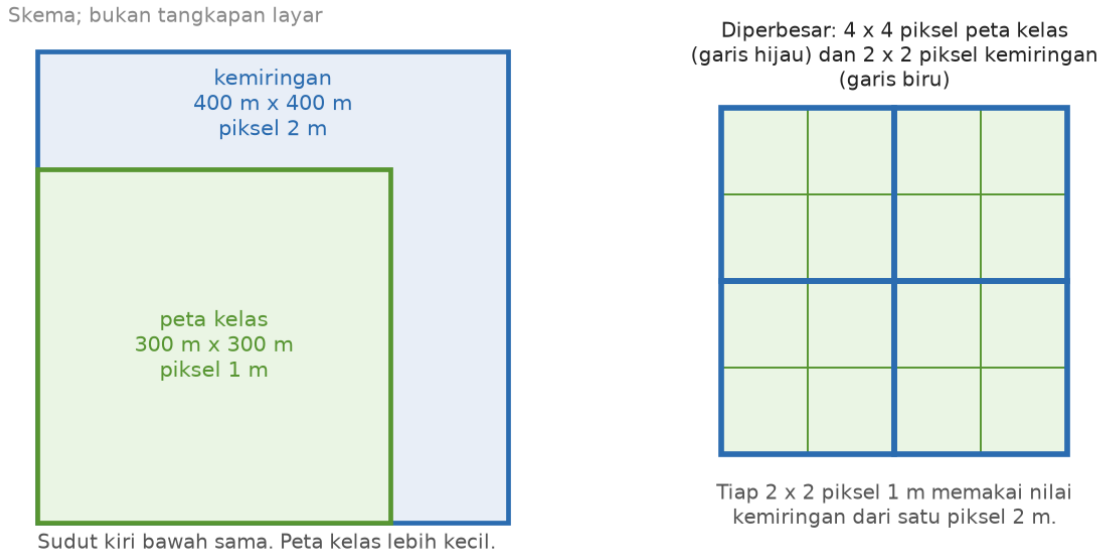
Bahan. Dua hasil sebelumnya di folder hasil. Peta kelas berpiksel 1 m dan berukuran 300 m x 300 m. Peta kemiringan berpiksel 2 m dan berukuran 400 m x 400 m. Keduanya berbagi sudut kiri bawah yang sama.

Lima istilah baru untuk skrip ini:

- **Perbandingan logika**: rumus yang hasilnya benar (1) atau salah (0), misalnya `"x@1" > 25`.
- **AND**: operator "dan". Hasilnya 1 hanya bila kedua syarat benar di piksel yang sama.
- **Piksel beda ukuran**: dua raster yang pikselnya tidak sama besar. Raster Calculator tetap bisa menggabungkannya bila Anda menentukan ukuran hasil.
- **Array benar-salah**: array yang isinya `True` atau `False`, hasil perbandingan seperti `arr == 4`.

- **Menjumlahkan benar-salah**: `arr.sum()` pada array 0 dan 1 (atau benar dan salah) menghitung berapa piksel yang bernilai 1.

Ilustrasi 19.12: Dua raster beda ukuran



Ilustrasi 19.13: Alur skrip 19.7



Skrip utuh:

```
PYTHON
# -*- coding: utf-8 -*-
# I2 Bab 6 (latihan terpadu): lahan terbuka di lereng curam, dari peta kelas (1 m) dan kemiringan (2 m).
```

```
# Jalankan setelah skrip bab 2 dan bab 5. Penulis: Badar Mubarok Yogaswara
import os
import numpy as np
import processing
from osgeo import gdal
from qgis.core import QgsRasterLayer

HASIL = r"C:/temp/hasil_i2"
kelas = QgsRasterLayer(os.path.join(HASIL, "terbimbing_maks_lik.tif"), "peta_kelas")
persen = QgsRasterLayer(os.path.join(HASIL, "slope_persen.tif"), "slope_persen")

# Dua raster berbeda ukuran piksel (1 m dan 2 m). Raster calculator memakai ukuran dan luas yang Anda tentukan,
# di sini mengikuti peta kelas: piksel 1 m, area 300 m x 300 m. Nilai kemiringan diambil dari piksel 2 m yang
menutupinya.
AMBANG = 25                       # batas kemiringan (persen)
KELAS_UJI = 4                     # kelas yang dicari (4 = Lahan Terbuka)
NAMA_KELAS = {1: "hutan", 2: "kebun", 3: "sawah", 4: "lahan terbuka"}[KELAS_UJI]
luas_ha = kelas.rasterUnitsPerPixelX() * kelas.rasterUnitsPerPixelY() / 10000.0     # luas satu piksel (ha)
keluar = os.path.join(HASIL, "terbuka_curam.tif")
processing.run("native:rastercalc", {
    "EXPRESSION": '("peta_kelas@1" = %d) AND ("slope_persen@1" > %d)' % (KELAS_UJI, AMBANG), "LAYERS": [kelas,
persen],
    "CELL_SIZE": kelas.rasterUnitsPerPixelX(), "EXTENT": kelas.extent(), "CRS": kelas.crs(), "OUTPUT": keluar})
arr = gdal.Open(keluar).ReadAsArray()
print("Piksel %s di lereng > %d persen:" % (NAMA_KELAS, AMBANG), int(arr.sum()), "= %.3f ha" % (arr.sum() *
luas_ha))
terbuka = gdal.Open(os.path.join(HASIL, "terbimbing_maks_lik.tif")).ReadAsArray() == KELAS_UJI
print("Seluruh %s pada peta: %d piksel = %.3f ha; bagian di lereng curam: %.1f%%" % (
    NAMA_KELAS, terbuka.sum(), terbuka.sum() * luas_ha, 100.0 * arr.sum() / terbuka.sum()))
```

Penjelasan baris demi baris. Nomor baris mengikuti berkas di atas.

Baris	Tulisan	Artinya
1-3	komentar	Judul, penulis, dan pesan: jalankan setelah skrip kemiringan (19.2) dan klasifikasi terbimbing (19.5).
4-8	`import os`, `import numpy as np`, `import processing`, `from osgeo import gdal`, `from qgis.core import QgsRasterLayer`	Panggil perkakas. Kotak `np` sebenarnya tidak dipakai langsung di skrip ini (sisa impor). Tidak berpengaruh.
10	`HASIL = r"C:/temp/hasil_i2"`	Satu-satunya jalur di skrip ini. Ganti sesuai komputer Anda.
11-12	`kelas = QgsRasterLayer(...)` dan `persen = QgsRasterLayer(...)`	Buka dua peta hasil. Nama layer `peta_kelas` dan `slope_persen` harus sama dengan nama di rumus baris 22.
14-15	komentar	Menjelaskan bahwa dua raster berbeda ukuran piksel. Hasil mengikuti peta kelas (piksel 1 m, area 300 m x 300 m). Nilai kemiringan diambil dari piksel 2 m yang menutupinya.
16	`AMBANG = 25`	Batas kemiringan dalam persen. Semua pengaturan analisis kini ada di sini, bukan terselip di dalam rumus.
17	`KELAS_UJI = 4`	Nomor kelas yang dicari. Angka 4 berarti Lahan Terbuka.
18	`NAMA_KELAS = {1: "hutan", 2: "kebun", 3: "sawah", 4: "lahan terbuka"}[KELAS_UJI]`	Kamus nama kelas (Bab 5), lalu langsung diambil isinya dengan kunci `KELAS_UJI`. Hasilnya teks `"lahan terbuka"`. Teks ini dipakai pada hasil cetakan supaya selalu cocok dengan kelas yang dicari.

Baris	Tulisan	Artinya
19	`luas_ha = kelas.rasterUnitsPerPixelX() * kelas.rasterUnitsPerPixelY() / 10000.0`	Luas satu piksel dalam hektare: lebar piksel kali tinggi piksel, dibagi 10.000 (1 hektare = 10.000 meter persegi). Ukuran piksel dibaca dari peta kelas, jadi hitungan tetap benar bila pikselnya bukan 1 m.
20	`keluar = os.path.join(HASIL, "terbuka_curam.tif")`	Nama berkas hasil.
21-23	`processing.run("native:rastercalc", {...})`	Jalankan Raster Calculator. Rumus dirakit dengan `%d`, yang diisi `KELAS_UJI` dan `AMBANG`. Dengan nilai bawaan, rumusnya `("peta_kelas@1" = 4) AND ("slope_persen@1" > 25)`: bernilai 1 di piksel yang kelasnya 4 (Lahan Terbuka) dan kemiringannya lebih dari 25 persen. Di dalam rumus ini, satu tanda = berarti "sama dengan". Ukuran piksel, luas, dan CRS mengikuti peta kelas.
24	`arr = gdal.Open(keluar).ReadAsArray()`	Baca hasil menjadi array berisi 0 dan 1.
25	`print("Piksel %s di lereng > %d persen:" % (NAMA_KELAS, AMBANG), int(arr.sum()), "= %.3f ha" % (arr.sum() * luas_ha))`	`arr.sum()` menjumlahkan semua nilai, yang berarti menghitung piksel bernilai 1. Dikali `luas_ha` memberi luas dalam hektare. Nama kelas dan ambang di teks diisi dari `NAMA_KELAS` dan `AMBANG`.
26	`terbuka = gdal.Open(...).ReadAsArray() == KELAS_UJI`	Baca peta kelas, lalu bandingkan dengan `KELAS_UJI`. Hasilnya array benar-salah: `True` di piksel kelas yang dicari.
27-28	`print("Seluruh %s pada peta: ..." % (...))`	Cetak jumlah piksel kelas itu, luasnya (`terbuka.sum() * luas_ha`), dan persen yang berada di lereng curam: `100.0 * arr.sum() / terbuka.sum()`.

Hasil. Layar menampilkan:

```
HASIL
Piksel lahan terbuka di lereng > 25 persen: 7709 = 0.771 ha
Seluruh lahan terbuka pada peta: 10944 piksel = 1.094 ha; bagian di lereng curam: 70.4%
```

Dari hasil ini Anda tahu: sekitar 0,77 hektare lahan terbuka berada di lereng curam. Itu 70,4% dari seluruh lahan terbuka pada peta (1,094 ha). Jawaban untuk Kepala Seksi: sekitar tujuh dari sepuluh bagian lahan terbuka ada di lereng curam, sehingga ia kandidat pertama untuk kegiatan rehabilitasi. Saya memeriksa angka ini dengan hitungan mandiri NumPy yang mencocokkan koordinat tiap piksel 1 m ke piksel 2 m yang menutupinya. Hasilnya sama: 7.709 dari 10.944 piksel. Ingat, ini data sintetis, jadi angkanya hanya untuk belajar.

Bagian yang boleh diubah.

- **Baris 17, nomor kelas.** Cukup ganti di satu tempat. Rumus, pembanding di baris 26, dan teks cetakan ikut berubah. Pada uji, kelas 1 (Hutan) dengan kemiringan lebih dari 15 menghasilkan 14.612 piksel (1,461 ha) dari 33.415 piksel hutan (43,7%).
- **Baris 16, angka 25.** Ambang kemiringan. Ganti sesuai aturan instansi Anda. [CEK]
- **Baris 18, kamus nama.** Bila kelas Anda bukan empat, ubah kamus ini.
- **Baris 19, luas satu piksel.** Dihitung otomatis dari ukuran piksel peta kelas. Untuk piksel 0,5 m, luas satu piksel 0,25 meter persegi, dan skrip menghitungnya sendiri. Tidak ada angka yang perlu Anda ganti.

Jebakan.

- **Nomor kelas di luar kamus.** Bila `KELAS_UJI` bukan 1 sampai 4, baris 18 berhenti dengan `KeyError`, sebab kamus tidak punya kunci itu. Itu pengaman yang disengaja: skrip tidak mencetak laporan dengan nama kelas yang salah. Tambahkan kelas baru ke kamus lebih dulu.
- **Kelas ada di kamus tetapi tidak ada di peta.** Bila `terbuka.sum()` bernilai 0, skrip tidak berhenti dengan `ZeroDivisionError`. Ia mencetak `RuntimeWarning: invalid value encountered in divide` dan persennya tertulis `nan%`.
- **Skrip sebelumnya belum dijalankan.** Bila `terbimbing_maks_lik.tif` tidak ada, skrip berhenti di baris 21 dengan `QgsProcessingException: Unable to execute algorithm`, lalu `Incorrect parameter value for EXTENT`. Galat yang sama muncul karena layer kosong tidak punya luas (lihat Jebakan Skrip 19.1). Bila hanya `slope_persen.tif` yang tidak ada, skrip TIDAK berhenti dan tidak memberi pesan apa pun, tetapi angkanya ngawur dan bisa berbeda tiap kali dijalankan (pada uji tercetak `10863 = 1.086 ha ... 99.3%`, lalu `10728 = 1.073 ha ... 98.0%`, padahal yang benar `7709 = 0.771 ha ... 70.4%`). Jalankan Skrip 19.2 dan 19.5 lebih dulu, dan periksa bahwa kedua berkas ada.
- **Dua raster tidak saling menutupi.** Peta kelas hanya sebagian dari area kemiringan. Hasil hanya berlaku di area yang dimiliki kedua raster. [kemungkinan] Di data ini, area peta kelas berada seluruhnya di dalam area kemiringan (cocok dengan ukuran masing-masing raster).
- **Impor tak terpakai.** `np` tidak dipakai. Impor yang tidak terpakai bisa dihapus, dan skrip tetap berjalan sama.

Latihan variasi.

1. Hitung hutan (kelas 1) yang berada di lereng lebih dari 15 persen.
2. Hitung sawah (kelas 3) yang berada di lereng 8 persen atau kurang.

Contoh jawaban nomor 1, baris 16 dan 17 menjadi:

```python
AMBANG = 15                     # batas kemiringan (persen)
KELAS_UJI = 1                   # kelas yang dicari (1 = Hutan)
```

Layar menampilkan `Piksel hutan di lereng > 15 persen: 14612 = 1.461 ha`, lalu `Seluruh hutan pada peta: 33415 piksel = 3.342 ha; bagian di lereng curam: 43.7%`. Kata "hutan" muncul sendiri dari kamus di baris 18.

Contoh jawaban nomor 2: ubah baris 16 menjadi `AMBANG = 8` dan baris 17 menjadi `KELAS_UJI = 3`. Ganti juga tanda `>` pada rumus (baris 22) menjadi `<=`, dan pada teks baris 25 menjadi `<=`. Layar menampilkan `6511 = 0.651 ha` dan sawah seluruhnya `22447 piksel = 2.245 ha`, dengan bagian di lereng datar `29.0%`. Kata "lereng curam" di baris 28 tidak lagi tepat, jadi ubah sesuai kebutuhan.

> **Pola yang bisa dipakai ulang:** Menggabungkan dua analisis adalah satu rumus dan satu hitungan: **rumus logika** di Raster Calculator (ganti **kelas** dan **ambang**), lalu **arr.sum()** kali **luas satu piksel**. Selalu tentukan ukuran piksel dan luas hasil dari raster yang paling Anda percaya.

Cek paham

1. Mengapa rata-rata peta berisi 0 dan 1 sama dengan bagian piksel yang bernilai 1?
2. Pada hasil 6 kluster di Skrip 19.4, mengapa kelas Kebun tidak ada di peta?
3. Mengapa uji akurasi memakai titik uji, bukan area latih?

Jawaban.

1. Karena menjumlahkan 0 dan 1 sama dengan menghitung jumlah angka 1, dan membaginya dengan jumlah semua piksel memberi bagiannya. Contoh: rata-rata `veg` sebesar 0,6666 berarti sekitar dua pertiga piksel bervegetasi.
2. Satu kluster (kluster 4) memuat hampir sama banyak piksel latih Kebun dan Sawah. Satu kluster hanya bisa diberi satu nama, jadi Sawah menang dan Kebun hilang.
3. Menilai peta dengan piksel yang dipakai melatihnya membuat hasil terlalu bagus, seperti murid diuji dengan soal latihannya sendiri. Titik uji berada di luar area latih.

Kesalahan umum

- **Melompati urutan skrip.** Perbaikan: jalankan 19.2 sebelum 19.3 dan 19.7, serta 19.4 dan 19.5 sebelum 19.6.
- **Membiarkan jalur contoh.** Jalur bawaan hanyalah alamat contoh. Perbaikan: ganti jalur di bagian atas tiap skrip.
- **Menjalankan dari terminal python-qgis biasa.** Muncul `ModuleNotFoundError: No module named 'processing'`. Perbaikan: pakai Editor Skrip di dalam QGIS.
- **Percaya satu angka.** Akurasi keseluruhan bagus tidak berarti semua kelas ada. Perbaikan: baca juga matriks, akurasi per kelas, dan jumlah piksel per kelas.

Ringkasan dan latihan

- Skrip raster bekerja dengan dua cara: **memanggil alat QGIS** (`processing.run`) atau **menghitung sendiri** dengan NumPy dan GDAL.
- **Aljabar raster** menjalankan rumus di tiap piksel. Rata-rata peta 0/1 adalah bagian piksel bernilai 1.
- Turunan DEM (kemiringan, arah lereng, bayangan bukit) lahir dari satu DEM. Kemiringan persen sama dengan tan derajat dikali 100.
- **K-means** mengelompokkan piksel tanpa contoh, dan hasilnya bergantung pada titik awal acak. Klasifikasi **terbimbing** memakai area latih.

- **Uji akurasi** memakai titik uji yang terpisah. Waspadai kolom kosong dan nilai 0 pada peta.

Latihan. Ubah Skrip 19.1 agar menghitung NDVI pada `Citra_Lahan.tif` dari `paket-i2`. Band berurutan sama (R, G, B, NIR). Bagian mana yang Anda ganti, dan bagian mana yang boleh dihapus karena tidak dipakai lagi?

Tabel perbandingan: tujuh skrip sekilas

Skrip	Masukan	Keluaran utama	Butuh dulu
19.1 aljabar raster	`Citra_Komposit.tif`, `DSM_Drone.tif`, `DTM_Drone.tif`	`ndvi.tif`, `veg.tif`, `chm.tif`, `tajuk_tinggi.tif`	tidak ada
19.2 turunan DEM	`DEM_Lereng.tif`	`slope_derajat.tif`, `aspect.tif`, `hillshade.tif`, `slope_persen.tif`	tidak ada
19.3 kelas lereng	`slope_persen.tif`, `Batas_KPH.gpkg`	`kelas_lereng.tif`, `luas_kelas_per_kph.gpkg`	19.2
19.4 tak terbimbing	`Citra_Lahan.tif`, `Area_Latih.gpkg`	`area_latih.tif`, `tak_terbimbing_k6.tif`, `tak_terbimbing_k10.tif`	tidak ada
19.5 terbimbing	`Citra_Lahan.tif`, `Area_Latih.gpkg`	`terbimbing_jarak_min.tif`, `terbimbing_maks_lik.tif`	tidak ada
19.6 uji akurasi	`Titik_Uji.gpkg`, empat peta hasil	`acuan_titik.tif`, `kappa_maks_lik.txt`	19.4 dan 19.5
19.7 terpadu	`terbimbing_maks_lik.tif`, `slope_persen.tif`	`terbuka_curam.tif`	19.2 dan 19.5

BAB 20: Titik yang Bisa Dipercaya: Survei Lapangan dan Basis Data dengan Skrip

> **Berdasarkan Seri I3:** Skrip bab ini berasal dari Buku I3, *Survei Lapangan dan Basis Data* (Intermediate Class), berkas `i3_01_statistik_gnss.py` sampai `i3_09_gabung_dan_bentrok_id.py`. Di buku itu, topik survei dibahas lewat menu dan klik QGIS. Bab ini menunjukkan sisi skripnya. Sembilan skrip ini terbagi tiga kelompok: mengukur alat, menyiapkan wadah data, dan memeriksa data.

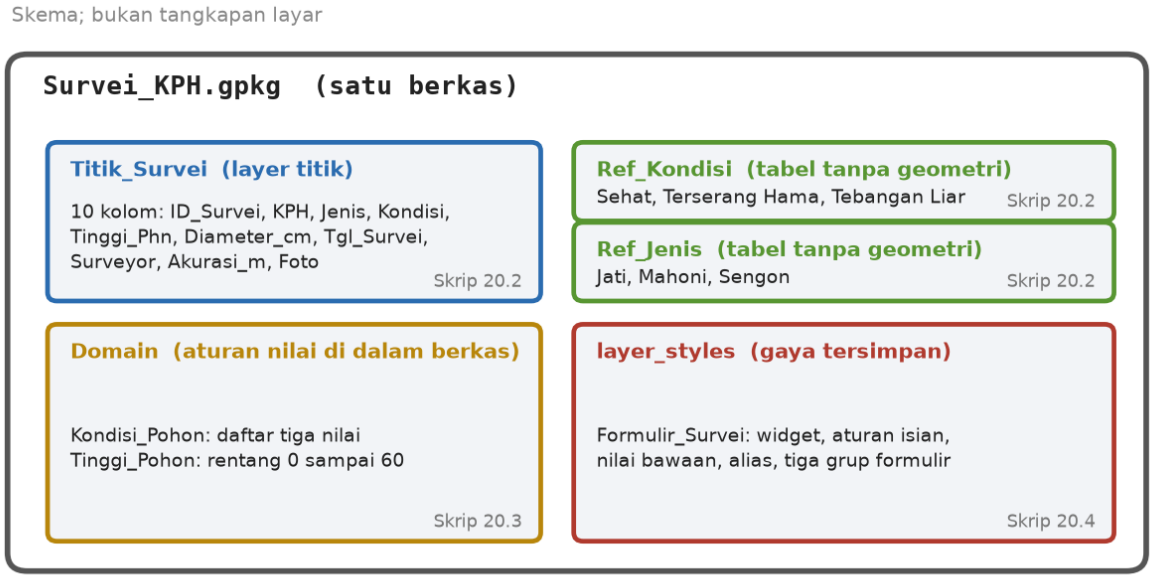
Studi kasus: "Titik ini benar-benar di sini?"

Tim surveyor KPH Contoh mencatat titik pohon memakai ponsel. Kepala Seksi mengajukan tiga pertanyaan. Seberapa tepat titik dari ponsel itu? Bagaimana menyiapkan wadah data supaya surveyor tidak salah ketik? Dan setelah data masuk, bagaimana memastikan mutunya? Anda bisa menjawab semuanya lewat menu QGIS. Dengan skrip, jawabannya bisa diulang untuk survei berikutnya, dan pemeriksaannya tidak bergantung pada ingatan.

Konsep: survei yang bisa dipercaya, dalam tiga kalimat

Data survei itu seperti buku catatan lapangan: berguna hanya bila isinya benar dan bisa diperiksa. Pekerjaan Anda ada tiga. Pertama, mengukur seberapa tepat alat Anda (Skrip 20.1). Kedua, menyiapkan wadah dan penjaga nilai agar salah ketik tertahan sejak awal (Skrip 20.2 sampai 20.5). Ketiga, memeriksa dan menanyai data yang sudah masuk (Skrip 20.6 sampai 20.9).

Ilustrasi 20.1: Isi satu berkas GeoPackage survei



Lima istilah yang perlu Anda kenal sekarang:

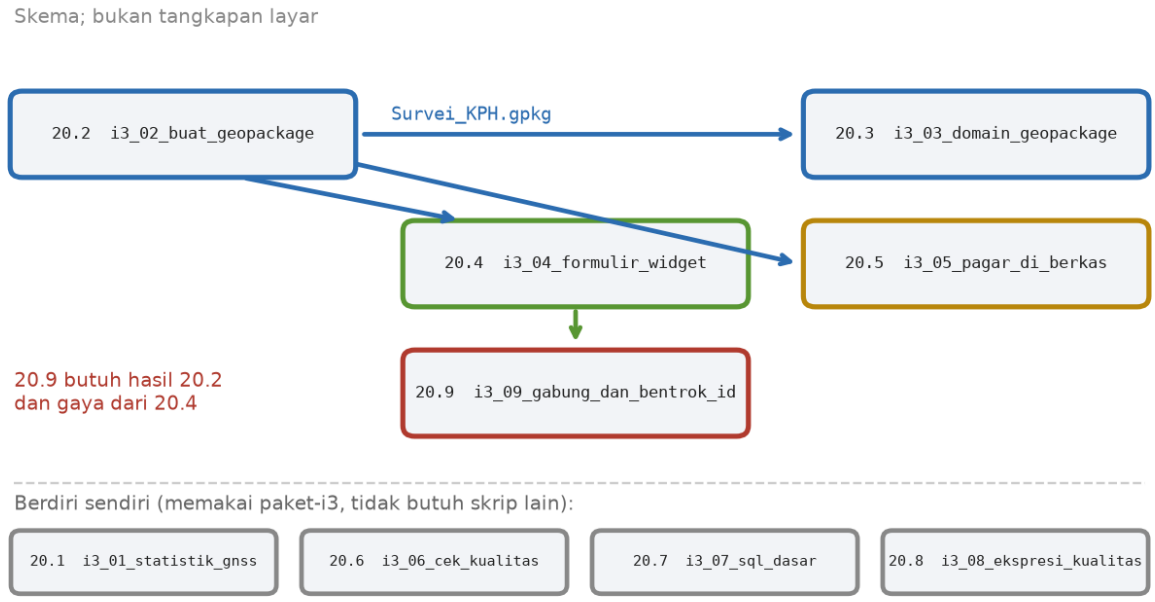
- **GNSS**: sistem satelit penentu posisi. Ponsel Anda memakainya untuk menunjukkan letak Anda.

- **GeoPackage**: satu berkas `.gpkg` yang bisa memuat banyak layer, tabel, dan pengaturan tampilan. Isinya basis data SQLite.
- **Tabel tanpa geometri**: tabel biasa tanpa titik, garis, atau poligon, misalnya daftar pilihan.
- **Domain**: daftar atau rentang nilai yang sah untuk satu kolom.
- **SQL**: bahasa untuk bertanya kepada basis data.

Siapkan dulu: folder, data, dan urutan skrip

1. Buat folder `KPH_Contoh`. Salin folder `paket-i3` ke dalamnya. Paket itu memuat `Gnss_Uji.gpkg` dan `Survei_Lapangan.gpkg`.
2. Skrip 20.2 membuat berkas `Survei_KPH.gpkg` di folder `KPH_Contoh`. Skrip 20.3, 20.4, 20.5, dan 20.9 memakainya.
3. Jalankan skrip dari **Plugins ► Python Console ► Show Editor** (Bab 3). Ganti jalur di bagian atas skrip dengan jalur Anda.

Ilustrasi 20.2: Siapa memakai hasil siapa



> **Jalur contoh milik komputer penulis:** Di tiap skrip, jalur data menunjuk folder `D:/KPH_Contoh/...`. Itu jalur contoh milik penulis. Ganti dengan lokasi folder di komputer Anda. Buku ini menampilkan skrip apa adanya, jadi Anda akan melihat jalur itu di blok kode.

Satu catatan tentang tempat menjalankan. Skrip 20.1 sampai 20.7 juga berjalan dari terminal `python-qgis` biasa. Skrip 20.8 dan 20.9 paling aman dari Python Console QGIS. Pada uji, Skrip 20.9 dari terminal biasa gagal dengan `ModuleNotFoundError: No module named 'processing'`. Skrip 20.8 dari terminal biasa, tanpa aplikasi QGIS yang disiapkan lebih dulu, berhenti dengan pesan `Segmentation fault` (kode keluar 139 di Git Bash; di PowerShell kodenya -1073741819). Memanggil `Processing.initialize()` saja tidak cukup. Bila aplikasinya disiapkan lebih dulu dengan `QgsApplication([], False)` lalu `initQgis()` (lihat Bab 21), Skrip 20.8 berjalan tanpa masalah itu.

Nomor skrip di bab ini (20.1 dan seterusnya) mengikuti bab ini. Di dalam berkas, nomor yang tertulis mengikuti Buku I3 (misalnya "SKRIP I3.1").

Skrip 20.1: Seberapa tepat titik GNSS?

Berkas `i3_01_statistik_gnss.py`. Di dalam berkasnya tertulis "SKRIP I3.1".

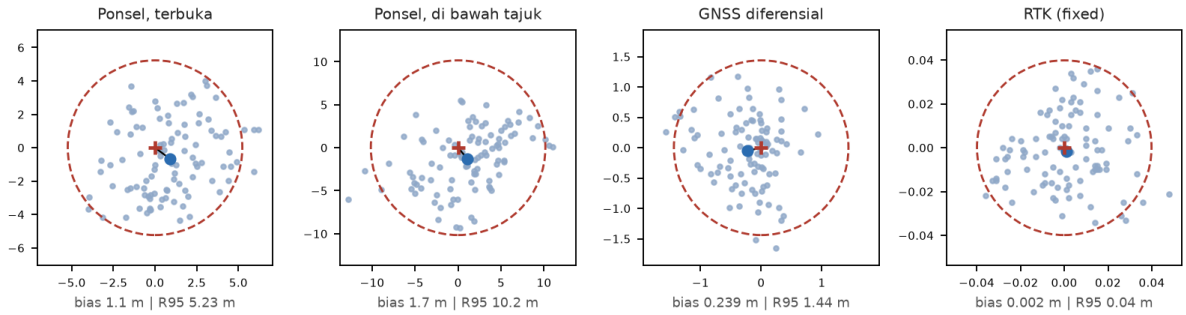
Tujuan. Mengukur seberapa tepat dan seberapa rapat pembacaan GNSS dari empat cara ukur, dibandingkan dengan satu titik yang posisinya diketahui benar.

Bahan. Paket `paket-i3`, berkas `Gnss_Uji.gpkg` dengan dua layer: `Gnss_Uji` (400 pembacaan, 100 untuk tiap `Mode`) dan `Titik_Referensi` (satu titik). Koordinatnya UTM zona 49 Selatan, satuan meter. [terbukti: diperiksa dengan `ogrinfo`]

Gagasan pokoknya: bayangkan menembak sasaran. Bila semua peluru menumpuk di satu sisi, tembakan Anda **presisi** tetapi tidak **akurat**. Bila peluru tersebar tetapi rata-ratanya di tengah, Anda akurat tetapi tidak presisi. Yang kita cari: keduanya baik.

Ilustrasi 20.3: Empat sebaran titik GNSS

Skema; bukan tangkapan layar



Tanda + merah = titik referensi. Titik biru besar = rata-rata pembacaan. Lingkaran putus-putus = R95 (jarak yang memuat 95 dari 100 pembacaan). Skala tiap kotak berbeda.

Istilah baru untuk skrip ini:

- **Titik referensi**: titik yang koordinat benarnya sudah diketahui, dipakai sebagai kunci jawaban.
- **Bias**: jarak antara titik rata-rata semua pembacaan dan titik referensi. Ukuran akurasi.
- **Presisi**: sebaran pembacaan di sekitar titik rata-ratanya sendiri.
- **Galat RMS** (RMSE): satu angka yang memuat bias dan sebaran sekaligus.
- **CEP50 dan R95**: jarak yang mencakup separuh pembacaan, dan jarak yang mencakup 95 dari 100 pembacaan. R95 menjawab pertanyaan "seburuk apa kemungkinan terburuknya".

Ilustrasi 20.4: Alur skrip 20.1



Skrip utuh:

```python
# -*- coding: utf-8 -*-
# [SKRIP I3.1] Menghitung akurasi dan presisi GNSS dari 100 pembacaan per mode terhadap titik referensi.
# Penulis: Badar Mubarok Yogaswara
# Dijalankan di Python Console QGIS (atau python-qgis.bat). Ubah JALUR bila folder data Anda berbeda.
import math
import os
from qgis.core import QgsVectorLayer

JALUR = r"D:/KPH_Contoh/paket-i3/Gnss_Uji.gpkg"

pembacaan = QgsVectorLayer(JALUR + "|layername=Gnss_Uji", "Gnss_Uji", "ogr")
referensi = QgsVectorLayer(JALUR + "|layername=Titik_Referensi", "Titik_Referensi", "ogr")
assert pembacaan.isValid() and referensi.isValid(), "Layer tidak terbaca: periksa JALUR"

titik_ref = next(referensi.getFeatures()).geometry().asPoint()

# kelompokkan koordinat per mode
per_mode = {}
for fitur in pembacaan.getFeatures():
    p = fitur.geometry().asPoint()
    per_mode.setdefault(fitur["Mode"], []).append((p.x(), p.y()))


def persentil(nilai, q):
    """Persentil dengan interpolasi linear (sama dengan numpy.percentile)."""
    u = sorted(nilai)
    k = (len(u) - 1) * q / 100.0
    a = int(math.floor(k))
    b = min(a + 1, len(u) - 1)
    return u[a] + (u[b] - u[a]) * (k - a)


print("%-15s %8s %8s %8s %8s %8s" % ("Mode", "Bias", "Presisi", "RMSE", "CEP50", "R95"))
for mode, pts in per_mode.items():
    jarak = [math.hypot(x - titik_ref.x(), y - titik_ref.y()) for x, y in pts]       # galat ke titik benar
    cx = sum(x for x, _ in pts) / len(pts)
    cy = sum(y for _, y in pts) / len(pts)
    bias = math.hypot(cx - titik_ref.x(), cy - titik_ref.y())                        # akurasi (geser rata-
rata)
```

```
    presisi = math.sqrt(sum((x - cx) ** 2 + (y - cy) ** 2 for x, y in pts) / len(pts))  # sebaran di sekitar rata-rata
    rmse = math.sqrt(sum(j * j for j in jarak) / len(jarak))
    print("%-15s %8.3f %8.3f %8.3f %8.3f %8.3f" % (mode, bias, presisi, rmse, persentil(jarak, 50), persentil(jarak, 95)))
```

Penjelasan baris demi baris. Dua blok. Nomor baris mengikuti berkas di atas.

Blok A: baris 1 sampai 21, membaca dan mengelompokkan.

Baris	Tulisan	Artinya
1-4	komentar	Baris 1 memberi tahu Python bahwa berkas memakai huruf UTF-8. Baris 2-4 memuat judul, penulis, dan cara menjalankan.
5-7	`import math`, `import os`, `from qgis.core import QgsVectorLayer`	Panggil perkakas hitung (`math`), perkakas jalur (`os`, tidak dipakai di skrip ini), dan pembuka layer vektor.
9	`JALUR = r"D:/KPH_Contoh/paket-i3/Gnss_Uji.gpkg"`	Lokasi berkas data. Ganti dengan jalur Anda. Huruf `r` membuat teks dibaca apa adanya.
11-12	`pembacaan = QgsVectorLayer(JALUR + ..., "Gnss_Uji", "ogr")`	Buka dua layer dari satu berkas. Sumber datanya adalah `JALUR`, disambung tanda garis tegak dan `layername=` plus nama layer. Isian terakhir `"ogr"` adalah pembaca data vektor.
13	`assert pembacaan.isValid() and referensi.isValid(), "..."`	Penjaga. Bila salah satu layer gagal terbuka, skrip berhenti dan menampilkan pesan itu. `assert` berarti "pastikan".
15	`titik_ref = next(referensi.getFeatures()).geometry().asPoint()`	Ambil titik referensi. `getFeatures()` memberi daftar fitur, `next(...)` mengambil isi pertamanya, lalu `.geometry().asPoint()` mengubahnya menjadi koordinat x dan y.
18	`per_mode = {}`	Kamus kosong (Bab 5). Nanti: kunci adalah nama mode, isinya daftar koordinat.
19-21	`for fitur in pembacaan.getFeatures():` lalu `per_mode.setdefault(fitur["Mode"], []).append((p.x(), p.y()))`	Telusuri 400 pembacaan. `setdefault(kunci, [])` mengambil daftar milik mode itu, atau membuat daftar kosong bila belum ada. `.append((x, y))` menambahkan sepasang koordinat ke daftar itu. Hasilnya empat kunci, masing-masing berisi 100 pasang.

Blok B: baris 24 sampai 41, menghitung.

Baris	Tulisan	Artinya
24-30	`def persentil(nilai, q):`	Resep yang menghitung persentil: nilai di mana `q` persen data berada di bawahnya. Docstring menyebut hasilnya sama dengan `numpy.percentile`. Saya mencocokkannya pada data ini sampai enam desimal, dan hasilnya sama.
26	`u = sorted(nilai)`	Urutkan dari terkecil.
27-29	`k = (len(u) - 1) * q / 100.0` lalu `a`, `b`	Hitung posisi (boleh pecahan). `a` adalah posisi bulat di bawahnya, `b` posisi bulat di atasnya (tidak melewati ujung).

Baris	Tulisan	Artinya
30	`return u[a] + (u[b] - u[a]) * (k - a)`	Campur dua isi terdekat sesuai sisa pecahan. Itu yang disebut interpolasi linear.
33	`print("%-15s %8s ..." % ("Mode", "Bias", ...))`	Cetak kepala tabel. `%-15s` memberi lebar 15 huruf, rata kiri. `%8s` memberi lebar 8, rata kanan.
34	`for mode, pts in per_mode.items():`	Ulangi tiap mode. `.items()` memberi pasangan kunci dan isinya.
35	`jarak = [math.hypot(x - titik_ref.x(), y - titik_ref.y()) for x, y in pts]`	Hitung jarak lurus tiap pembacaan ke titik benar. `math.hypot` adalah rumus Pythagoras. Tulisan `for x, y in pts` membuka tiap pasangan menjadi dua kotak.
36-37	`cx = sum(x for x, _ in pts) / len(pts)` dan `cy = ...`	Titik rata-rata: jumlah semua x dibagi banyaknya. Tanda `_` membuang bagian yang tidak dipakai.
38	`bias = math.hypot(cx - titik_ref.x(), cy - titik_ref.y())`	Bias: jarak titik rata-rata ke titik benar.
39	`presisi = math.sqrt(sum((x - cx) ** 2 + (y - cy) ** 2 for x, y in pts) / len(pts))`	Presisi: akar dari rata-rata kuadrat jarak tiap pembacaan ke titik rata-ratanya sendiri.
40	`rmse = math.sqrt(sum(j * j for j in jarak) / len(jarak))`	Galat RMS: akar dari rata-rata kuadrat jarak ke titik benar.
41	`print("%-15s %8.3f ..." % (mode, bias, presisi, rmse, persentil(jarak, 50), persentil(jarak, 95)))`	Cetak satu baris. CEP50 adalah persentil ke-50, dan R95 persentil ke-95. `%8.3f` mencetak angka dengan tiga desimal.

Hasil. Layar menampilkan (satuannya meter):

```
HASIL
Mode                 Bias  Presisi     RMSE    CEP50      R95
Ponsel_Terbuka      1.100    3.137    3.324    2.943    5.233
Ponsel_Tajuk        1.696    5.618    5.868    5.227   10.157
Diferensial         0.239    0.743    0.780    0.630    1.439
RTK_Fix             0.002    0.024    0.024    0.020    0.040
```

Dari hasil ini Anda tahu: ponsel di area terbuka punya R95 sekitar 5,2 m, dan ponsel di bawah tajuk sekitar 10,2 m, dua kali lebih buruk. Alat diferensial R95-nya 1,4 m, dan RTK 4 cm. Bias ponsel (1,1 dan 1,7 m) jauh lebih kecil daripada sebarannya. Artinya rata-rata banyak pembacaan menolong, tetapi satu pembacaan tunggal bisa meleset beberapa meter. Data ini sintetis. Angkanya bukan spesifikasi alat apa pun, jadi ukur sendiri alat Anda di titik referensi.

Bagian yang boleh diubah.

- **Baris 9, jalur.** Ganti dengan lokasi `Gnss_Uji.gpkg` di komputer Anda.
- **Baris 11-12, nama layer.** Ganti untuk data GNSS Anda sendiri.

- **Baris 21, nama kolom Mode.** Kolom yang membedakan cara ukur. Ganti dengan kolom pengelompok lain.
- **Baris 41, angka 50 dan 95.** Ganti dengan persentil lain. Pada uji, angka `90` (dan judul `R90`) menghasilkan R90 4,829 untuk ponsel terbuka, 8,742 untuk ponsel di bawah tajuk, 1,241 untuk diferensial, dan 0,036 untuk RTK.

Jebakan.

- **Jalur salah.** Layar menampilkan `AssertionError: Layer tidak terbaca: periksa JALUR`. Pesan itu datang dari baris 13. Periksa jalur dan nama layer.
- **Nama kolom salah ketik.** Pada uji, menulis `"Moda"` alih-alih `"Mode"` menghasilkan `KeyError: 'Moda'` di baris 21. Salin nama kolom dari tabel atribut.
- **Layer referensi kosong.** Perintah `next(...)` di baris 15 berhenti dengan `StopIteration` bila tidak ada titik. [kemungkinan: perilaku standar Python, belum diuji pada layer ini]
- **Satuan.** Angka di tabel adalah meter karena data berkoordinat UTM. Pada data berkoordinat derajat, angka jarak tidak bermakna. Cek sistem koordinat layer Anda.
- **Impor tak terpakai.** `import os` tidak dipakai. Tidak ada yang rusak.

Latihan variasi.

1. Ganti R95 menjadi R90.
2. Cetak hanya cara ukur yang R95-nya kurang dari 2 m.

Contoh jawaban nomor 1: ubah `"R95"` di baris 33 menjadi `"R90"`, dan `persentil(jarak, 95)` di baris 41 menjadi `persentil(jarak, 90)`. Layar menampilkan angka R90 seperti di atas.

Contoh jawaban nomor 2, baris 41 menjadi dua baris:

```
PYTHON
    if persentil(jarak, 95) < 2:
        print("%-15s %8.3f %8.3f %8.3f %8.3f %8.3f" % (mode, bias, presisi, rmse, persentil(jarak, 50),
persentil(jarak, 95)))
```

Layar hanya menampilkan dua baris data: `Diferensial` dan `RTK_Fix`.

> **Pola yang bisa dipakai ulang:** Statistik per kelompok selalu tiga langkah: **kelompokkan** (`per_mode.setdefault(kunci, []).append(nilai)`), **hitung** per kelompok, **cetak**. Untuk data lain, ganti **jalur**, **nama layer**, dan **nama kolom pengelompok**. Fungsi `persentil` boleh dipakai untuk daftar angka apa pun.

Skrip 20.2: Menyiapkan lemari survei

Berkas `i3_02_buat_geopackage.py`. Di dalam berkasnya tertulis "SKRIP I3.2".

Tujuan. Membuat satu berkas GeoPackage kosong untuk survei: satu layer titik `Titik_Survei` (sepuluh kolom) dan dua tabel daftar pilihan, `Ref_Kondisi` dan `Ref_Jenis`.

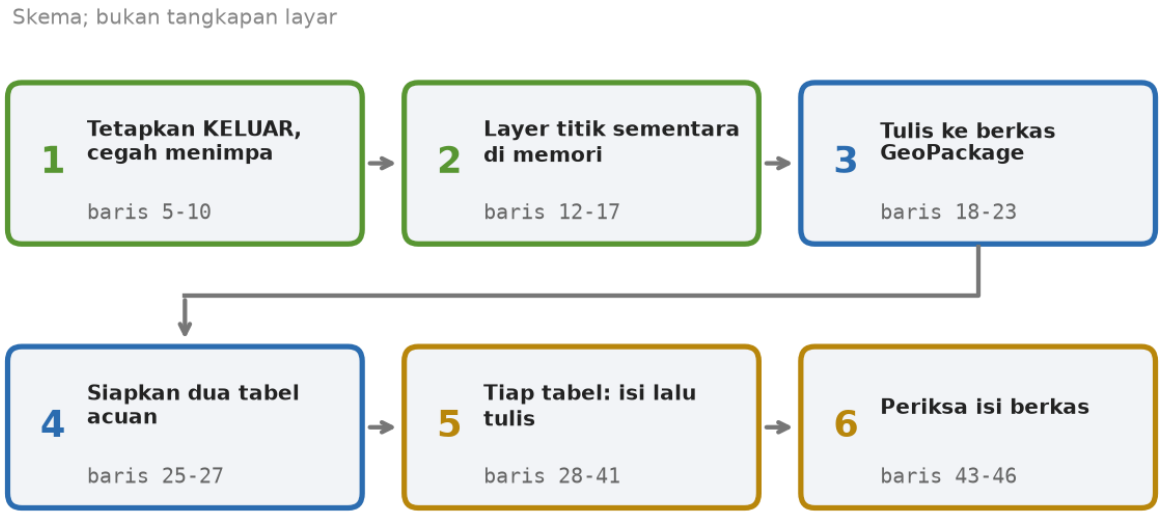
Bahan. Tidak ada data masukan. Skrip menulis `Survei_KPH.gpkg` ke folder `KPH_Contoh`. Skrip menolak berjalan bila berkas itu sudah ada.

Istilah baru untuk skrip ini:

- **Layer memori**: layer sementara yang hidup di memori komputer, belum menjadi berkas.
- **Skema**: daftar kolom beserta jenisnya.
- **Penulis berkas** (`QgsVectorFileWriter`): perkakas QGIS yang menulis layer ke berkas.

Skrip ini bekerja dalam enam tahap.

Ilustrasi 20.5: Alur skrip 20.2



Skrip utuh:

```python
# -*- coding: utf-8 -*-
# [SKRIP I3.2] Membuat GeoPackage survei kosong (layer Titik_Survei + tabel acuan Ref_Kondisi dan Ref_Jenis)
# Penulis: Badar Mubarok Yogaswara
# Dijalankan di Python Console QGIS atau python-qgis.bat. Hasil: Survei_KPH.gpkg di folder KELUAR.
import os
from qgis.core import (QgsVectorLayer, QgsVectorFileWriter, QgsCoordinateReferenceSystem, QgsProject,
QgsFeature)

KELUAR = r"D:/KPH_Contoh/Survei_KPH.gpkg"     # ganti dengan folder proyek Anda
os.makedirs(os.path.dirname(KELUAR), exist_ok=True)
assert not os.path.exists(KELUAR), "Berkas sudah ada: ganti nama atau pindahkan dulu agar tidak tertimpa"

# 1. Titik_Survei: titik berkoordinat UTM 49S dengan sepuluh field
skema = ("Point?crs=EPSG:32749"
         "&field=ID_Survei:string(10)&field=KPH:string(20)&field=Jenis:string(20)&field=Kondisi:string(20)"
         "&field=Tinggi_Phn:double&field=Diameter_cm:double&field=Tgl_Survei:date"
         "&field=Surveyor:string(30)&field=Akurasi_m:double&field=Foto:string(100)")
sementara = QgsVectorLayer(skema, "Titik_Survei", "memory")
opsi = QgsVectorFileWriter.SaveVectorOptions()
opsi.driverName = "GPKG"
opsi.layerName = "Titik_Survei"
opsi.fileEncoding = "UTF-8"
hasil = QgsVectorFileWriter.writeAsVectorFormatV3(sementara, KELUAR, QgsProject.instance().transformContext(),
opsi)
assert hasil[0] == QgsVectorFileWriter.NoError, hasil

# 2. Dua tabel acuan (tanpa geometri) ditambahkan ke berkas yang sama
acuan = {"Ref_Kondisi": ("Kondisi", ["Sehat", "Terserang Hama", "Tebangan Liar"]),
         "Ref_Jenis": ("Jenis", ["Jati", "Mahoni", "Sengon"])}
```

```
for nama, (kolom, nilai) in acuan.items():
    tabel = QgsVectorLayer("None?field=%s:string(20)" % kolom, nama, "memory")
    tabel.startEditing()
    for v in nilai:
        f = QgsFeature(tabel.fields())
        f[kolom] = v
        tabel.addFeature(f)
    tabel.commitChanges()
    o = QgsVectorFileWriter.SaveVectorOptions()
    o.driverName = "GPKG"
    o.layerName = nama
    o.actionOnExistingFile = QgsVectorFileWriter.CreateOrOverwriteLayer      # tambah layer, jangan timpa
berkas
    h = QgsVectorFileWriter.writeAsVectorFormatV3(tabel, KELUAR, QgsProject.instance().transformContext(), o)
    assert h[0] == QgsVectorFileWriter.NoError, h

# 3. Periksa isi berkas
for nama in ["Titik_Survei", "Ref_Kondisi", "Ref_Jenis"]:
    lyr = QgsVectorLayer(KELUAR + "|layername=" + nama, nama, "ogr")
    print(nama, "valid:", lyr.isValid(), "| baris:", lyr.featureCount(), "| field:", [f.name() for f in
lyr.fields()])
```

Penjelasan baris demi baris. Tiga blok. Nomor baris mengikuti berkas di atas.

Blok A: baris 1 sampai 23, layer titik.

Baris	Tulisan	Artinya
1-4	komentar	Judul, penulis, dan hasil yang diharapkan: `Survei_KPH.gpkg`.
5-6	`import os` dan `from qgis.core import (...)`	Panggil perkakas. Tanda kurung memungkinkan daftar panjang dipecah ke beberapa baris. `QgsCoordinateReferenceSystem` diimpor tetapi tidak dipakai.
8	`KELUAR = r"D:/KPH_Contoh/Survei_KPH.gpkg"`	Lokasi berkas yang akan dibuat. Ganti dengan folder proyek Anda.
9	`os.makedirs(os.path.dirname(KELUAR), exist_ok=True)`	`os.path.dirname` mengambil bagian folder dari jalur. Folder itu dibuat bila belum ada, tanpa bertanya.
10	`assert not os.path.exists(KELUAR), "..."`	Penjaga: bila berkas sudah ada, skrip berhenti dengan pesan. Tujuannya agar kerja Anda tidak tertimpa.
13-16	`skema = ("Point?crs=EPSG:32749" "&field=ID_Survei:string(10)" ...)`	Satu teks panjang disambung dari beberapa potongan di dalam kurung. Isinya: jenis geometri (titik), sistem koordinat (UTM 49 Selatan), lalu sepuluh kolom dengan pola `field=nama:jenis`. Jenisnya `string(10)` (teks sampai 10 huruf), `double` (angka desimal), dan `date` (tanggal).
17	`sementara = QgsVectorLayer(skema, "Titik_Survei", "memory")`	Buat layer kosong di memori dari skema itu. Kata `"memory"` menunjuk penyedia data layer memori.
18-21	`opsi = QgsVectorFileWriter.SaveVectorOptions()` lalu empat penugasan	Siapkan pengaturan penulisan. Format `GPKG`, nama layer di dalam berkas, dan huruf UTF-8. Tiap baris mengisi satu sifat objek `opsi`.

Baris	Tulisan	Artinya
22	`hasil = QgsVectorFileWriter.writeAsVectorFormatV3(sementara, KELUAR, ..., opsi)`	Tulis layer sementara ke berkas. Isian ketiga, `transformContext()`, adalah aturan transformasi koordinat proyek, yang wajib diberikan. Hasilnya sepasang: status dan pesan.
23	`assert hasil[0] == QgsVectorFileWriter.NoError, hasil`	Pastikan status pertama (`hasil[0]`) berarti "tanpa galat". Bila tidak, tampilkan isi `hasil`.

Blok B: baris 26 sampai 41, dua tabel acuan.

Baris	Tulisan	Artinya
26-27	`acuan = {"Ref_Kondisi": ("Kondisi", [...]), "Ref_Jenis": ("Jenis", [...])}`	Kamus: kunci adalah nama tabel, isinya sepasang (nama kolom, daftar nilai).
28	`for nama, (kolom, nilai) in acuan.items():`	Ulangi dua kali. Tanda kurung di tengah membuka pasangan bersarang, sehingga tiga kotak terisi sekaligus: `nama`, `kolom`, `nilai`.
29	`tabel = QgsVectorLayer("None?field=%s:string(20)" % kolom, nama, "memory")`	Layer memori tanpa geometri (kata `None`), dengan satu kolom teks. Tanda `%s` diisi nama kolom.
30	`tabel.startEditing()`	Mulai mode sunting, seperti menekan tombol pensil di QGIS.
31-34	`for v in nilai:` lalu `f = QgsFeature(tabel.fields()), f[kolom] = v, tabel.addFeature(f)`	Untuk tiap nilai, buat satu baris, isi kolomnya, dan tambahkan ke tabel.
35	`tabel.commitChanges()`	Simpan sunting ke layer memori.
36-39	`o = QgsVectorFileWriter.SaveVectorOptions()` lalu pengaturan	Pengaturan penulisan seperti blok A, dengan satu tambahan: `CreateOrOverwriteLayer`. Artinya: tambahkan layer ke berkas yang sudah ada, jangan timpa berkasnya.
40-41	`h = ...writeAsVectorFormatV3(tabel, KELUAR, ..., o)` lalu `assert`	Tulis tabel ke berkas yang sama, lalu pastikan tanpa galat.

Blok C: baris 44 sampai 46, memeriksa isi.

Baris	Tulisan	Artinya
44-45	`for nama in ["Titik_Survei", "Ref_Kondisi", "Ref_Jenis"]:` lalu `lyr = QgsVectorLayer(...)`	Buka kembali tiap layer dari berkas, sebagai bukti bahwa tertulis.
46	`print(nama, "valid:", lyr.isValid(), ..., [f.name() for f in lyr.fields()])`	Cetak nama, apakah terbuka, jumlah baris, dan daftar nama kolom. Bagian terakhir adalah list comprehension (Bab 6).

Hasil. Layar menampilkan:

```
HASIL
Titik_Survei valid: True | baris: 0 | field: ['fid', 'ID_Survei', 'KPH', 'Jenis', 'Kondisi', 'Tinggi_Phn',
'Diameter_cm', 'Tgl_Survei', 'Surveyor', 'Akurasi_m', 'Foto']
Ref_Kondisi valid: True | baris: 3 | field: ['fid', 'Kondisi']
Ref_Jenis valid: True | baris: 3 | field: ['fid', 'Jenis']
```

Dari hasil ini Anda tahu: ketiganya terbuka. `Titik_Survei` kosong (0 baris) tetapi sudah punya kolom. Dua tabel acuan masing-masing tiga baris. Kolom `fid` ikut dibuat otomatis oleh GeoPackage sebagai nomor baris.

Bagian yang boleh diubah.

- **Baris 8, KELUAR.** Lokasi dan nama berkas.
- **Baris 13-16, skema.** Tambah, hapus, atau ganti kolom. Ganti `EPSG:32749` bila data Anda memakai sistem koordinat lain. Pada uji, menambah `&field=Catatan:string(100)` memunculkan kolom `Catatan` sebagai kolom terakhir.
- **Baris 26-27, isi acuan.** Ganti nilai pilihan. Pada uji, menambah `"Akasia"` di daftar jenis menghasilkan `Ref_Jenis` dengan 4 baris.

Jebakan.

- **Berkas sudah ada.** Menjalankan skrip dua kali berhenti di baris 10 dengan `AssertionError: Berkas sudah ada: ganti nama atau pindahkan dulu agar tidak tertimpa.` Itu memang disengaja. Ganti nama `KELUAR` atau pindahkan berkas lama.
- **Folder dibuat tanpa tanya.** Baris 9 membuat folder baru bila belum ada. Bila Anda salah ketik jalur, folder yang salah ikut dibuat.
- **Impor tak terpakai dan nama kotak.** `QgsCoordinateReferenceSystem` tidak dipakai. Kotak `hasil` (baris 22) dan `h` (baris 40) dipakai untuk hal serupa dengan nama berbeda. Itu tidak salah, hanya kurang konsisten.
- **Skrip ini prasyarat.** Skrip 20.3, 20.4, 20.5, dan 20.9 butuh `Survei_KPH.gpkg`. Jalankan ini dulu.

Latihan variasi.

1. Tambahkan kolom `Catatan` (teks 100 huruf) ke `Titik_Survei`.
2. Tambahkan "Akasia" ke daftar jenis pohon.

Contoh jawaban nomor 1, baris 16 menjadi (jangan lupa hapus berkas lama dulu):

```
PYTHON
        "&field=Surveyor:string(30)&field=Akurasi_m:double&field=Foto:string(100)&field=Catatan:string(100)")
```

Contoh jawaban nomor 2, baris 27 menjadi:

```
PYTHON
        "Ref_Jenis": ("Jenis", ["Jati", "Mahoni", "Sengon", "Akasia"])}
```

Layar menampilkan `Titik_Survei valid: True | baris: 0 | field: [..., 'Foto', 'Catatan']` dan `Ref_Jenis valid: True | baris: 4 | field: ['fid', 'Jenis']`.

> **Pola yang bisa dipakai ulang:** Membuat berkas GeoPackage lewat skrip selalu tiga langkah: **buat layer memori dari skema** (`QgsVectorLayer("Point?crs=...&field=...", nama, "memory")`), **tulis dengan writeAsVectorFormatV3** (ganti **nama layer**), dan **tambahkan layer lain** dengan `CreateOrOverwriteLayer`. Yang Anda ganti: **jalur KELUAR**, **skema kolom**, dan **isi tabel acuan**.

Tiga lapis penjagaan: peta untuk Skrip 20.3 sampai 20.5

Tiga skrip berikut menjaga isian `Kondisi` dengan tiga cara yang kekuatannya berbeda. Anggap saja tiga lapis pagar di kebun. Pagar pertama dipasang di dalam berkas, pagar kedua di formulir QGIS, dan pagar ketiga adalah penjaga di pintu basis data. Hasil uji menunjukkan hanya pagar ketiga yang menolak penulis mana pun, termasuk skrip.

Ilustrasi 20.6: Tiga lapis penjagaan



Hasil uji skrip 20.3, 20.4, dan 20.5 pada QGIS 4.0.2. Hanya lapis ke-3 yang menolak penulis mana pun.

Skrip 20.3: Menanam domain nilai di dalam berkas

Berkas `i3_03_domain_geopackage.py`. Di dalam berkasnya tertulis "SKRIP I3.3". Jalankan setelah Skrip 20.2.

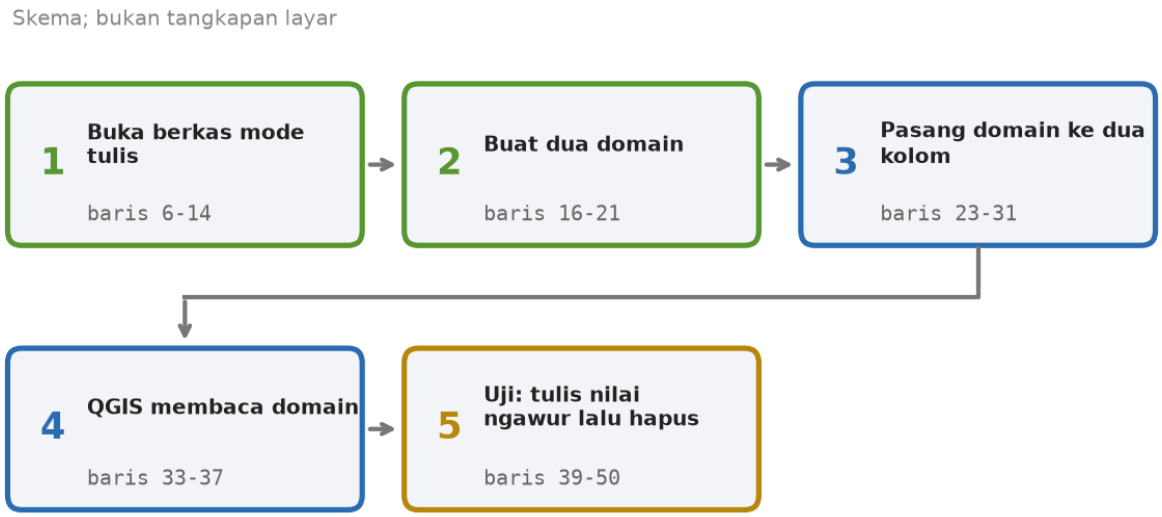
Tujuan. Menanam dua domain di dalam GeoPackage: daftar kondisi pohon, dan rentang tinggi pohon 0 sampai 60 m. Lalu membuktikan dua hal: QGIS membaca domain itu, tetapi penulisan lewat program tidak otomatis ditolak.

Bahan. `Survei_KPH.gpkg` hasil Skrip 20.2. Skrip ini **mengubah** berkas itu (membuka dengan mode tulis). Jalankan dari terminal `python-qgis` atau dari Python Console QGIS. Komentar di berkas menyebut `python-qgis.bat` karena butuh modul `osgeo`. [CEK: modul `osgeo` tersedia di Python Console QGIS Anda]

Istilah baru untuk skrip ini:

- **Domain daftar**: kolom hanya boleh berisi nilai dari daftar. **Domain rentang**: kolom hanya boleh berisi angka dalam rentang.
- **Widget**: kotak isian di formulir, misalnya kotak pilihan atau kotak angka.
- **Mode tulis**: membuka berkas dengan izin mengubah isinya.
- **WKT**: cara menulis bentuk geometri sebagai teks, misalnya `POINT(312100 9996100)`.

Ilustrasi 20.7: Alur skrip 20.3



Skrip utuh:

```python
# -*- coding: utf-8 -*-
# [SKRIP I3.3] Domain nilai yang tersimpan DI DALAM GeoPackage (daftar Kondisi dan rentang Tinggi_Phn),
# lalu membuktikan dua hal: QGIS ikut membacanya, dan penulisan lewat program tidak otomatis ditolak.
# Penulis: Badar Mubarok Yogaswara
# Jalankan setelah skrip I3.2, dengan python-qgis.bat (butuh modul osgeo.ogr). Bekerja pada berkas
Survei_KPH.gpkg.
from osgeo import ogr, gdal
from qgis.core import QgsVectorLayer, QgsVectorLayerUtils

gdal.UseExceptions()
ogr.UseExceptions()
GPKG = r"D:/KPH_Contoh/Survei_KPH.gpkg"

ds = ogr.Open(GPKG, 1)
lyr = ds.GetLayerByName("Titik_Survei")

# 1. Buat dua domain pada tingkat berkas
ds.AddFieldDomain(ogr.CreateCodedFieldDomain(
    "Kondisi_Pohon", "Kondisi pohon hasil survei", ogr.OFTString, ogr.OFSTNone,
    {"Sehat": "Sehat", "Terserang Hama": "Terserang Hama", "Tebangan Liar": "Tebangan Liar"}))
ds.AddFieldDomain(ogr.CreateRangeFieldDomain(
    "Tinggi_Pohon", "Tinggi pohon 0 sampai 60 m", ogr.OFTReal, ogr.OFSTNone, 0.0, True, 60.0, True))

# 2. Pasang domain ke kolomnya
for kolom, domain in [("Kondisi", "Kondisi_Pohon"), ("Tinggi_Phn", "Tinggi_Pohon")]:
    i = lyr.GetLayerDefn().GetFieldIndex(kolom)
    lama = lyr.GetLayerDefn().GetFieldDefn(i)
    fd = ogr.FieldDefn(lama.GetName(), lama.GetType())          # salinan definisi kolom; yang lama tidak boleh
diubah langsung
    fd.SetDomainName(domain)
    lyr.AlterFieldDefn(i, fd, ogr.ALTER_DOMAIN_FLAG)
print("Domain di berkas:", ds.GetFieldDomainNames())
ds = None

# 3. QGIS membaca domain itu sebagai widget formulir
uji = QgsVectorLayer(GPKG + "|layername=Titik_Survei", "uji", "ogr")
for kolom in ["Kondisi", "Tinggi_Phn"]:
    w = uji.editorWidgetSetup(uji.fields().indexOf(kolom))
    print("Widget %-11s: %s" % (kolom, w.type()))
```

```
# 4. Domain bukan gembok: nilai di luar daftar tetap bisa tertulis lewat program
ds = ogr.Open(GPKG, 1)
lyr = ds.GetLayerByName("Titik_Survei")
f = ogr.Feature(lyr.GetLayerDefn())
f.SetGeometry(ogr.CreateGeometryFromWkt("POINT(312100 9996100)"))
f.SetField("ID_Survei", "UJI-DOMAIN")
f.SetField("Kondisi", "ngawur")
f.SetField("Tinggi_Phn", 250.0)
lyr.CreateFeature(f)
print("Nilai 'ngawur' dan tinggi 250 m: tertulis tanpa ditolak")
ds.ExecuteSQL("DELETE FROM Titik_Survei WHERE ID_Survei = 'UJI-DOMAIN'")       # bersihkan baris uji
ds = None
```

Penjelasan baris demi baris. Nomor baris mengikuti berkas di atas.

Baris	Tulisan	Artinya
1-5	komentar	Baris 5 mengingatkan: jalankan setelah Skrip 20.2.
6-7	`from osgeo import ogr, gdal` dan `from qgis.core import QgsVectorLayer, QgsVectorLayerUtils`	Panggil perkakas GDAL untuk berkas vektor (`ogr`) dan raster (`gdal`), serta perkakas QGIS. `QgsVectorLayerUtils` tidak dipakai.
9-10	`gdal.UseExceptions()` dan `ogr.UseExceptions()`	Nyalakan "alarm". Tanpa baris ini, GDAL diam saja saat gagal. Dengan baris ini, kegagalan menghentikan skrip dengan pesan galat.
11	`GPKG = r"D:/KPH_Contoh/Survei_KPH.gpkg"`	Lokasi berkas hasil Skrip 20.2. Ganti dengan jalur Anda.
13	`ds = ogr.Open(GPKG, 1)`	Buka berkas. Angka `1` berarti mode tulis. Kotak `ds` adalah "berkas yang terbuka".
14	`lyr = ds.GetLayerByName("Titik_Survei")`	Ambil layer `Titik_Survei` dari berkas itu.
17-19	`ds.AddFieldDomain(ogr.CreateCodedFieldDomain("Kondisi_Pohon", ..., {...}))`	Buat domain daftar bernama `Kondisi_Pohon` lalu tambahkan ke berkas. Isiannya: nama, keterangan, jenis teks, dan kamus tiga nilai. Di kamus ini, kiri adalah nilai yang disimpan dan kanan adalah teks yang tampil. Keduanya sama.
20-21	`ds.AddFieldDomain(ogr.CreateRangeFieldDomain("Tinggi_Pohon", ..., 0.0, True, 60.0, True))`	Buat domain rentang. Isiannya: nama, keterangan, jenis angka desimal, batas bawah 0, `True` bila batas bawah ikut sah, batas atas 60, dan `True` bila batas atas ikut sah.
24	`for kolom, domain in [("Kondisi", "Kondisi_Pohon"), ("Tinggi_Phn", "Tinggi_Pohon")]:`	Ulangi dua kali. Tiap pasangan: nama kolom dan nama domain yang akan dipasang.

Baris	Tulisan	Artinya
25-26	`i = lyr.GetLayerDefn().GetFieldIndex(kolom)` dan `lama = ...GetFieldDefn(i)`	Cari nomor urut kolom, lalu ambil definisinya yang lama.
27	`fd = ogr.FieldDefn(lama.GetName(), lama.GetType())`	Buat salinan definisi kolom. Komentar di baris itu menjelaskan: definisi yang lama tidak boleh diubah langsung.
28-29	`fd.SetDomainName(domain)` lalu `lyr.AlterFieldDefn(i, fd, ogr.ALTER_DOMAIN_FLAG)`	Pasang nama domain ke salinan, lalu terapkan ke kolom. Bendera `ALTER_DOMAIN_FLAG` memberi tahu GDAL bahwa yang diubah hanya domainnya.
30-31	`print("Domain di berkas:", ds.GetFieldDomainNames())` lalu `ds = None`	Cetak nama domain yang kini ada. Lalu `ds = None` menutup berkas. Di GDAL, perubahan baru benar-benar tertulis saat kotaknya dikosongkan.
34-37	`uji = QgsVectorLayer(GPKG + ..., "uji", "ogr")` lalu perulangan	Buka layer yang sama lewat QGIS. Untuk tiap kolom, tanyakan widget formulirnya dengan `editorWidgetSetup(...)`, lalu cetak jenisnya (`w.type()`). Pola `%-11s` memberi lebar 11 huruf.
40-41	`ds = ogr.Open(GPKG, 1)` lalu `lyr = ...`	Buka lagi dalam mode tulis, untuk uji berikutnya.
42-47	`f = ogr.Feature(lyr.GetLayerDefn())` lalu `SetGeometry`, `SetField`, `CreateFeature`	Susun satu titik uji dengan isian ngawur: `ID_Survei` `UJI-DOMAIN`, kondisi `ngawur`, dan tinggi 250 m. Lalu tulis ke berkas. Bila domain adalah gembok, penulisan ini ditolak.
48	`print("Nilai 'ngawur' dan tinggi 250 m: tertulis tanpa ditolak")`	Cetak kesimpulan. Teks ini tertulis tetap. Bila baris 47 menolak, skrip sudah berhenti sebelum sampai sini.
49	`ds.ExecuteSQL("DELETE FROM Titik_Survei WHERE ID_Survei = 'UJI-DOMAIN'")`	Bersihkan baris uji dengan perintah SQL (dijelaskan di Skrip 20.7).
50	`ds = None`	Tutup berkas.

Hasil. Layar menampilkan:

```
HASIL
Domain di berkas: ['Kondisi_Pohon', 'Tinggi_Pohon']
Widget Kondisi    : ValueMap
Widget Tinggi_Phn : Range
Nilai 'ngawur' dan tinggi 250 m: tertulis tanpa ditolak
```

Dari hasil ini Anda tahu: dua domain tersimpan di berkas, dan QGIS membacanya sebagai kotak pilihan (`ValueMap`) dan kotak angka (`Range`). Tetapi nilai `ngawur` dan tinggi 250 m tertulis tanpa ditolak. **Domain adalah petunjuk bagi formulir, bukan gembok.**

Bagian yang boleh diubah.

- **Baris 11, jalur.** Ganti dengan lokasi `Survei_KPH.gpkg`.
- **Baris 17-21, isi domain.** Ganti daftar kondisi atau batas rentang. Pada uji, batas `50.0` dan keterangan `"Tinggi pohon 0 sampai 50 m"` menghasilkan domain dengan batas atas 50,0 (dicek lewat `GetMaxAsDouble()`).
- **Baris 24, pasangan kolom dan domain.** Tambah pasangan untuk kolom lain. Pada uji, menambah domain daftar `Jenis_Pohon` dan pasangan `("Jenis", "Jenis_Pohon")` membuat `Jenis` ikut bertipe `ValueMap`.
- **Baris 35, daftar kolom yang dicek.** Tambah nama kolom untuk melihat widget-nya.

Jebakan.

- **Menjalankan dua kali.** Layar menampilkan `RuntimeError: A domain of identical name already exists`. Domain dengan nama sama tidak bisa ditambah dua kali. Hanya jalankan sekali per berkas.
- **Skrip 20.2 belum dijalankan.** Layar menampilkan `RuntimeError: ...Survei_KPH.gpkg: No such file or directory`. Cek isi folder Anda.
- **Skrip berhenti di tengah jalan.** Baris uji `UJI-DOMAIN` ditulis di baris 47 dan dihapus di baris 49. Bila skrip berhenti di antaranya, baris uji itu tertinggal di data Anda. [kemungkinan: dari bacaan kode]
- **Mengubah berkas asli.** Skrip ini membuka dengan mode tulis. Beri salinan cadangan bila berkas Anda berharga.
- **Domain bukan gembok.** Ini inti hasil uji. Jangan mengira data otomatis bersih karena ada domain. Pemeriksaan di Skrip 20.6 tetap perlu.

Latihan variasi.

1. Ubah batas atas tinggi menjadi 50 m.
2. Tambahkan domain daftar `Jenis_Pohon` (Jati, Mahoni, Sengon) untuk kolom `Jenis`.

Contoh jawaban nomor 1, baris 20-21 menjadi:

```python
ds.AddFieldDomain(ogr.CreateRangeFieldDomain(
    "Tinggi_Pohon", "Tinggi pohon 0 sampai 50 m", ogr.OFTReal, ogr.OFSTNone, 0.0, True, 50.0, True))
```

Contoh jawaban nomor 2, sisipkan sebelum baris 20, dan tambahkan pasangan `("Jenis", "Jenis_Pohon")` di baris 24:

```python
ds.AddFieldDomain(ogr.CreateCodedFieldDomain(
    "Jenis_Pohon", "Jenis pohon", ogr.OFTString, ogr.OFSTNone,
    {"Jati": "Jati", "Mahoni": "Mahoni", "Sengon": "Sengon"}))
```

Layar menampilkan `Domain di berkas: ['Jenis_Pohon', 'Kondisi_Pohon', 'Tinggi_Pohon']`. Bila Anda juga menambah `"Jenis"` ke daftar di baris 35, layar menambah baris `Widget Jenis : ValueMap`.

> **Pola yang bisa dipakai ulang:** Menanam domain di berkas itu tiga langkah: **buat domain** (`CreateCodedFieldDomain` atau `CreateRangeFieldDomain`), **tambahkan ke berkas** (`AddFieldDomain`), **pasang ke kolom** (`SetDomainName` lalu `AlterFieldDefn`). Yang diganti: **jalur berkas**, **nama domain dan isinya**, **nama kolom**.

Skrip 20.4: Merancang formulir isian

Berkas `i3_04_formulir_widget.py`. Di dalam berkasnya tertulis "SKRIP I3.4". Ini skrip terpanjang di bab ini (138 baris, sembilan tahap). Penjelasannya dibagi lima blok. Jalankan setelah Skrip 20.2.

Tujuan. Mengatur formulir isian untuk layer `Titik_Survei`: kotak pilihan, batas angka, aturan wajib isi, nilai otomatis, label ramah, dan tata letak berkelompok. Semuanya disimpan sebagai gaya bawaan di dalam GeoPackage.

Bahan. `Survei_KPH.gpkg` hasil Skrip 20.2 (Skrip 20.3 boleh sudah atau belum dijalankan). Skrip ini menulis tabel gaya di dalam berkas. Skrip ini juga memakai proyek QGIS yang sedang terbuka. Jalankan dari Python Console QGIS atau terminal `python-qgis`.

Gagasannya: **gaya** (*style*) QGIS menyimpan cara sebuah layer tampil, termasuk bentuk formulirnya. Bila gaya disimpan di dalam GeoPackage, siapa pun yang membuka berkas itu mendapat formulir yang sama, tanpa berkas tambahan.

Ilustrasi 20.8: Alur skrip 20.4



Skrip utuh:

```python
# -*- coding: utf-8 -*-
# [SKRIP I3.4] Memasang domain nilai (Value Map, Value Relation, Range), aturan isian, nilai bawaan, widget foto,
# dan tata letak formulir pada layer Titik_Survei, lalu menyimpannya sebagai gaya bawaan di dalam GeoPackage.
# Penulis: Badar Mubarok Yogaswara
# Jalankan setelah skrip I3.2. Dijalankan di Python Console QGIS atau python-qgis.bat.
from qgis.core import (QgsVectorLayer, QgsEditorWidgetSetup, QgsFieldConstraints, QgsDefaultValue, QgsEditFormConfig,
                       QgsAttributeEditorContainer, QgsAttributeEditorField, Qgis, QgsProject, QgsFeature, QgsGeometry,
                       QgsPointXY, QgsVectorLayerUtils)

GPKG = r"D:/KPH_Contoh/Survei_KPH.gpkg"          # sama dengan KELUAR pada skrip I3.2

lyr = QgsVectorLayer(GPKG + "|layername=Titik_Survei", "Titik_Survei", "ogr")
assert lyr.isValid()
idx = lyr.fields().indexOf


def widget(nama_field, jenis, konfig):
    lyr.setEditorWidgetSetup(idx(nama_field), QgsEditorWidgetSetup(jenis, konfig))


# 1. Domain nilai ------------------------------------------------------------------------
# a) Value Map: daftar pendek yang tetap. Isi peta: {teks yang tampil: nilai yang disimpan}
pilihan = ["Sehat", "Terserang Hama", "Tebangan Liar"]
widget("Kondisi", "ValueMap", {"map": [{p: p} for p in pilihan]})
widget("KPH", "ValueMap", {"map": [{"KPH Alpha": "KPH Alpha"}, {"KPH Beta": "KPH Beta"}, {"KPH Gamma": "KPH Gamma"}]})

# b) Value Relation: pilihan diambil dari tabel acuan, jadi mengubah daftar cukup mengubah tabel
ref_jenis = QgsVectorLayer(GPKG + "|layername=Ref_Jenis", "Ref_Jenis", "ogr")
QgsProject.instance().addMapLayer(ref_jenis)
# LayerName dan LayerSource ikut disimpan agar pilihan tetap ketemu saat gaya dibaca di proyek lain
widget("Jenis", "ValueRelation", {"Layer": ref_jenis.id(), "LayerName": "Ref_Jenis", "LayerSource": ref_jenis.source(),
                                  "LayerProviderName": "ogr", "Key": "Jenis", "Value": "Jenis", "AllowNull": False,
                                  "OrderByValue": True, "AllowMulti": False, "NofColumns": 1, "FilterExpression": ""})

# c) Range: angka dibatasi rentang
widget("Tinggi_Phn", "Range", {"Min": 0.0, "Max": 60.0, "Step": 0.1, "Style": "SpinBox", "Precision": 1, "AllowNull": False})
widget("Diameter_cm", "Range", {"Min": 0.0, "Max": 200.0, "Step": 0.5, "Style": "SpinBox", "Precision": 1, "AllowNull": False})

# d) Tanggal dan foto
widget("Tgl_Survei", "DateTime", {"display_format": "yyyy-MM-dd", "field_format": "yyyy-MM-dd", "calendar_popup": True,
                                  "allow_null": False})
widget("Foto", "ExternalResource", {"DocumentViewer": 1, "DocumentViewerHeight": 0, "DocumentViewerWidth": 0,
                                    "FileWidget": True, "FileWidgetButton": True, "RelativeStorage": 1,
                                    "StorageMode": 0})

# 2. Aturan isian (constraint) -----------------------------------------------------------
K = QgsFieldConstraints
lyr.setFieldConstraint(idx("ID_Survei"), K.ConstraintNotNull, K.ConstraintStrengthHard)
lyr.setFieldConstraint(idx("ID_Survei"), K.ConstraintUnique, K.ConstraintStrengthHard)
lyr.setFieldConstraint(idx("Kondisi"), K.ConstraintNotNull, K.ConstraintStrengthHard)
lyr.setConstraintExpression(idx("Tinggi_Phn"), '"Tinggi_Phn" BETWEEN 0 AND 60', "Tinggi pohon harus 0 sampai 60 m")
lyr.setFieldConstraint(idx("Tinggi_Phn"), K.ConstraintExpression, K.ConstraintStrengthHard)
```

```
lyr.setConstraintExpression(idx("Akurasi_m"), '"Akurasi_m" <= 5', "Akurasi lebih dari 5 m: ulangi pengukuran
bila bisa")
lyr.setFieldConstraint(idx("Akurasi_m"), K.ConstraintExpression, K.ConstraintStrengthSoft)      # soft = hanya
peringatan

# 3. Nilai bawaan ------------------------------------------------------------------------------
lyr.setDefaultValueDefinition(idx("Tgl_Survei"), QgsDefaultValue("to_date(now())"))
lyr.setDefaultValueDefinition(idx("Surveyor"), QgsDefaultValue("@user_account_name"))
lyr.setDefaultValueDefinition(idx("ID_Survei"), QgsDefaultValue(
    "'SV-' || lpad(to_string(coalesce(maximum(to_int(substr(\"ID_Survei\", 4))), 0) + 1), 3, '0')"))

# Label yang ramah dibaca (alias) lengkap dengan satuan
ALIAS = {"ID_Survei": "ID survei", "KPH": "Petak KPH", "Jenis": "Jenis pohon", "Kondisi": "Kondisi pohon",
         "Tinggi_Phn": "Tinggi pohon (m)", "Diameter_cm": "Diameter batang (cm)", "Tgl_Survei": "Tanggal
survei",
         "Surveyor": "Nama surveyor", "Akurasi_m": "Akurasi GNSS (m)", "Foto": "Foto"}
for nm, al in ALIAS.items():
    lyr.setFieldAlias(idx(nm), al)

# 4. Tata letak formulir (Drag and drop designer) ----------------------------------------------------
cfg = lyr.editFormConfig()
cfg.setLayout(Qgis.AttributeFormLayout.DragAndDrop)
cfg.clearTabs()
akar = cfg.invisibleRootContainer()
akar.clear()


def kelompok(judul, field_field, kolom=1):
    c = QgsAttributeEditorContainer(judul, akar)
    c.setType(Qgis.AttributeEditorContainerType.GroupBox)
    c.setColumnCount(kolom)
    for nm in field_field:
        c.addChildElement(QgsAttributeEditorField(nm, idx(nm), c))
    akar.addChildElement(c)


kelompok("Identitas", ["ID_Survei", "KPH", "Surveyor", "Tgl_Survei"])
kelompok("Pohon", ["Jenis", "Kondisi", "Tinggi_Phn", "Diameter_cm"])
kelompok("Bukti", ["Foto", "Akurasi_m"])
lyr.setEditFormConfig(cfg)

# 5. Simpan sebagai gaya bawaan di dalam GeoPackage (tabel layer_styles) -----------------------------------
hasil_simpan, galat = lyr.saveStyleToDatabaseV2("Formulir_Survei", "Formulir survei pohon", True, "")
print("Simpan gaya ke GeoPackage:", hasil_simpan, galat or "(tanpa pesan galat)")

# 6. Uji: layer baru yang membaca ulang berkas langsung memakai formulir ini
baru = QgsVectorLayer(GPKG + "|layername=Titik_Survei", "uji_baca", "ogr")
pesan, ok = baru.loadDefaultStyle()
print("Gaya bawaan terbaca:", ok)
print("Widget Kondisi:", baru.editorWidgetSetup(baru.fields().indexOf("Kondisi")).type())
print("Widget Tinggi_Phn:", baru.editorWidgetSetup(baru.fields().indexOf("Tinggi_Phn")).type())
print("Tata letak:", baru.editFormConfig().layout(), "| alias Tinggi_Phn:",
baru.attributeAlias(baru.fields().indexOf("Tinggi_Phn")))

# 7. Uji aturan: satu fitur salah (tinggi 250 m, kondisi kosong) dan satu fitur benar
def periksa(atribut):
    f = QgsFeature(baru.fields())
    f.setGeometry(QgsGeometry.fromPointXY(QgsPointXY(312100, 9996100)))
    for k, v in atribut.items():
        f[k] = v
    out = []
    for nm in ["ID_Survei", "Kondisi", "Tinggi_Phn", "Akurasi_m"]:
        ok, galat = QgsVectorLayerUtils.validateAttribute(baru, f, baru.fields().indexOf(nm))
        if not ok:
            out.append("%s: %s" % (nm, "; ".join(galat)))
    return out
```

```
print("Fitur salah :", periksa({"ID_Survei": "SV-001", "Kondisi": None, "Tinggi_Phn": 250.0, "Akurasi_m":
8.4}))
print("Fitur benar :", periksa({"ID_Survei": "SV-001", "Kondisi": "Sehat", "Tinggi_Phn": 12.0, "Akurasi_m":
2.0}))

# 8. Uji: di proyek baru, tabel acuan dicari lewat nama layer, dan nilai bawaan terisi
from qgis.core import QgsValueRelationFieldFormatter
proyek = QgsProject()                                  # proyek baru; proyek milik pengguna tidak dikosongkan
acuan = QgsVectorLayer(GPKG + "|layername=Ref_Jenis", "Ref_Jenis", "ogr")
proyek.addMapLayer(acuan)
uji = QgsVectorLayer(GPKG + "|layername=Titik_Survei", "uji_proyek_baru", "ogr")
uji.loadDefaultStyle()
konfig = uji.editorWidgetSetup(uji.fields().indexOf("Jenis")).config()
print("Tabel acuan ditemukan:", QgsValueRelationFieldFormatter.resolveLayer(konfig, proyek))
baris = QgsVectorLayerUtils.createFeature(uji)
print("Nilai bawaan -> ID_Survei:", baris["ID_Survei"], "| Tgl_Survei:", baris["Tgl_Survei"].toString("yyyy-MM-
dd"), "| Surveyor:", baris["Surveyor"])

# 9. Uji: tata letak formulir terbaca ulang dari gaya yang tersimpan (kelompok dan isinya)
print("Susunan formulir:")
cfg_uji = uji.editFormConfig()                         # simpan dulu di variabel agar tidak hilang saat dibaca
for anak in cfg_uji.invisibleRootContainer().children():
    print("- [%s]" % anak.name(), ", ".join(c.name() for c in anak.children()))
print("Widget Foto:", uji.editorWidgetSetup(uji.fields().indexOf("Foto")).type(),
      "| widget Tgl_Survei:", uji.editorWidgetSetup(uji.fields().indexOf("Tgl_Survei")).type())
```

Penjelasan baris demi baris. Nomor baris mengikuti berkas di atas.

Blok A: baris 1 sampai 18, persiapan.

Baris	Tulisan	Artinya
1-5	komentar	Judul, penulis, dan pesan: jalankan setelah Skrip 20.2.
6-8	`from qgis.core import (...)`	Satu daftar panjang perkakas QGIS dalam tiga baris. Sebagian, misalnya `QgsEditFormConfig`, tidak dipakai langsung.
10	`GPKG = r"D:/KPH_Contoh/Survei_KPH.gpkg"`	Lokasi berkas. Sama dengan `KELUAR` di Skrip 20.2. Ganti dengan jalur Anda.
12-13	`lyr = QgsVectorLayer(...)` lalu `assert lyr.isValid()`	Buka layer `Titik_Survei` dan pastikan terbuka. Bila gagal, skrip berhenti tanpa pesan (tidak ada teks setelah `assert`).
14	`idx = lyr.fields().indexOf`	Simpan sebuah fungsi di kotak. Tanpa tanda kurung, yang disimpan adalah fungsinya, bukan hasilnya. Sesudahnya, `idx("Kondisi")` memberi nomor urut kolom `Kondisi`.
17-18	`def widget(nama_field, jenis, konfig):` lalu `lyr.setEditorWidgetSetup(...)`	Resep pembantu: pasang satu widget ke satu kolom. `jenis` adalah nama widget (teks), dan `konfig` adalah pengaturannya (kamus). Resep ini dipakai tujuh kali di blok B.

Blok B: baris 21 sampai 44, tujuh widget. Istilah baru: **Value Map** (kotak pilihan dari daftar tetap), **Value Relation** (kotak pilihan dari tabel acuan), dan **Range** (kotak angka dengan batas).

Baris	Tulisan	Artinya
23-24	`pilihan = [...]` lalu `widget("Kondisi", "ValueMap", {"map": [{p: p} for p in pilihan]})`	Value Map untuk `Kondisi`. Isi `map` adalah daftar kamus kecil `{teks yang tampil: nilai yang disimpan}`. Karena keduanya sama, tiap kamus berbentuk `{p: p}`. Daftarnya dibuat dengan list comprehension dari `pilihan`.
25	`widget("KPH", "ValueMap", {"map": [{"KPH Alpha": "KPH Alpha"}, ...]})`	Value Map untuk `KPH`, tiga isinya ditulis langsung.
28-29	`ref_jenis = QgsVectorLayer(...)` lalu `QgsProject.instance().addMapLayer(ref_jenis)`	Buka tabel acuan `Ref_Jenis`, lalu masukkan ke proyek yang sedang terbuka. Widget Value Relation butuh layer itu ada di proyek.
31-33	`widget("Jenis", "ValueRelation", {...})`	Value Relation untuk `Jenis`. `Layer`, `LayerName`, dan `LayerSource` menunjuk tabel acuan. Komentar di baris 30 menjelaskan: semuanya disimpan agar pilihan tetap ketemu saat gaya dibaca di proyek lain. `Key` dan `Value` adalah kolom nilai dan kolom tampil. `AllowNull` False melarang kosong. `OrderByValue` True mengurutkan pilihan.
36-37	`widget("Tinggi_Phn", "Range", {"Min": 0.0, "Max": 60.0, "Step": 0.1, ...})`	Range untuk `Tinggi_Phn` dan `Diameter_cm`. `Min` dan `Max` batas angka, `Step` kelipatan tiap klik, `Precision` jumlah desimal tampil.
40-41	`widget("Tgl_Survei", "DateTime", {...})`	Widget tanggal dengan kalender pop-up. Format tampil dan simpan `yyyy-MM-dd`.
42-44	`widget("Foto", "ExternalResource", {...})`	Widget foto. Di antarmuka QGIS, ia bernama Attachment. Pengaturannya memuat tampilan gambar terpadu, tombol pilih berkas, dan penyimpanan jalur relatif. Arti tiap angka mengikuti pilihan di jendela Attachment. [CEK]

Blok C: baris 46 sampai 67, aturan isian, nilai bawaan, dan alias. Istilah baru: **aturan isian** (*constraint*: syarat sebuah isian), **tegas** (menolak simpan) lawan **lunak** (hanya memberi peringatan), dan **nilai bawaan** (isian yang terisi sendiri).

Baris	Tulisan	Artinya
47	`K = QgsFieldConstraints`	Beri nama pendek `K` untuk perkakas panjang ini, agar baris-baris berikutnya muat.
48-50	`lyr.setFieldConstraint(idx("ID_Survei"), K.ConstraintNotNull, K.ConstraintStrengthHard)`	Pasang aturan tegas: `ID_Survei` tidak boleh kosong, dan harus unik (baris 49). Aturan serupa untuk `Kondisi` di baris 50.
51-52	`lyr.setConstraintExpression(idx("Tinggi_Phn"), '...', "pesan")` lalu `setFieldConstraint(...)`	Aturan berupa ekspresi: tinggi harus antara 0 dan 60, dengan pesan galat. Baris 52 mengaktifkannya sebagai aturan tegas.

Baris	Tulisan	Artinya
53-54	`lyr.setConstraintExpression(idx("Akurasi_m"), ...)` lalu `K.ConstraintStrengthSoft`	Aturan lunak: akurasi sebaiknya 5 m atau kurang. Bila lebih, hanya muncul peringatan.
57-58	`lyr.setDefaultValueDefinition(idx("Tgl_Survei"), QgsDefaultValue("to_date(now())"))` dan baris untuk `Surveyor`	Nilai bawaan lewat ekspresi. Tanggal diisi tanggal hari ini. `Surveyor` diisi nama akun yang sedang masuk (`@user_account_name`).
59-60	`lyr.setDefaultValueDefinition(idx("ID_Survei"), QgsDefaultValue("..."))`	Nilai bawaan `ID_Survei`: ekspresi panjang yang dijelaskan di bawah tabel.
63-65	`ALIAS = {"ID_Survei": "ID survei", ...}`	Kamus: nama kolom asli di kiri, label ramah di kanan.
66-67	`for nm, al in ALIAS.items():` lalu `lyr.setFieldAlias(idx(nm), al)`	Pasang label ramah ke tiap kolom. Nama kolom di basis data tidak berubah.

Ekspresi nilai bawaan `ID_Survei` di baris 59-60 dibaca dari dalam ke luar. `substr("ID_Survei", 4)` mengambil teks mulai huruf keempat, yaitu `001` dari `SV-001`. `to_int(...)` mengubahnya menjadi angka 1. `maximum(...)` mencari angka terbesar di seluruh kolom. `coalesce(..., 0)` memakai 0 bila kolom masih kosong. Lalu ditambah 1, diubah jadi teks (`to_string`), diisi nol di depan sampai tiga huruf (`lpad`), dan diawali `SV-`. Hasil pada tabel kosong: `SV-001`.

Blok D: baris 69 sampai 93, tata letak dan menyimpan gaya.

Baris	Tulisan	Artinya
70-71	`cfg = lyr.editFormConfig()` lalu `cfg.setLayout(Qgis.AttributeFormLayout.DragAndDrop)`	Ambil pengaturan formulir layer, lalu pilih perancang "seret dan lepas" (*drag and drop*).
72-74	`cfg.clearTabs()`, `akar = cfg.invisibleRootContainer()`, `akar.clear()`	Kosongkan susunan lama. `akar` adalah wadah induk yang tak tampak, tempat semua grup akan dimasukkan.
77-83	`def kelompok(judul, field_field, kolom=1):`	Resep membuat satu grup: buat wadah bernama `judul`, bertipe kotak grup (`GroupBox`), jumlah lajur `kolom` (bawaan 1), isi dengan kolom-kolom dari daftar, lalu tambahkan ke `akar`.
86-88	`kelompok("Identitas", [...])`, `kelompok("Pohon", [...])`, `kelompok("Bukti", [...])`	Tiga grup: Identitas (empat kolom), Pohon (empat kolom), dan Bukti (foto dan akurasi).
89	`lyr.setEditFormConfig(cfg)`	Terapkan susunan baru ke layer.

Baris	Tulisan	Artinya
92	`hasil_simpan, galat = lyr.saveStyleToDatabaseV2("Formulir_Survei", "Formulir survei pohon", True, "")`	Simpan gaya ke dalam GeoPackage (di tabel bernama `layer_styles`). Isiannya: nama gaya, keterangan, `True` agar dipakai sebagai gaya bawaan, dan teks kosong (isi berkas formulir buatan sendiri, yang tidak dipakai di sini). Arti tiap isian dibaca dari dokumentasi API QGIS. [CEK] Hasilnya sepasang: status dan pesan.
93	`print("Simpan gaya ke GeoPackage:", hasil_simpan, galat or "(tanpa pesan galat)")`	Cetak status. Bentuk `a or b` berarti: pakai `a` bila ada isinya, kalau kosong pakai `b`.

Blok E: baris 95 sampai 138, empat uji.

Baris	Tulisan	Artinya
96-98	`baru = QgsVectorLayer(...), pesan, ok = baru.loadDefaultStyle(), print(...)`	Buka layer baru dari berkas, lalu minta ia memuat gaya bawaan dari dalam berkas. Cetak apakah berhasil.
99-101	`print("Widget Kondisi:", ...)` dan dua baris lain	Cetak jenis widget, tata letak, dan alias yang terbaca ulang. Ini bukti gaya benar-benar tersimpan.
104-114	`def periksa(atribut):`	Resep uji aturan. Buat satu fitur sementara di memori (tidak disimpan), isi dari kamus `atribut`, lalu tanya QGIS tiap kolom: lolos aturan atau tidak (`QgsVectorLayerUtils.validateAttribute`). Pesan galat dikumpulkan di daftar `out`. Kotak `ok` dan `galat` dipakai lagi dengan arti lain dibanding baris 92.
117-118	`print("Fitur salah :", periksa({...}))` dan `print("Fitur benar :", ...)`	Uji dua fitur: satu bermasalah (tinggi 250, kondisi kosong, akurasi 8,4), satu baik.
121-124	`from qgis.core import QgsValueRelationFieldFormatter, proyek = QgsProject(), acuan = ..., proyek.addMapLayer(acuan)`	Impor boleh di tengah skrip. Lalu **buat proyek baru yang terpisah** (`QgsProject()`, dengan tanda kurung), dan masukkan hanya tabel acuan ke proyek itu. Tujuannya meniru proyek bersih. Proyek yang sedang Anda buka tidak disentuh.
125-128	`uji = QgsVectorLayer(...), uji.loadDefaultStyle(), print("Tabel acuan ditemukan:", ...)`	Layer baru memuat gaya, lalu cek bahwa widget `Jenis` menemukan tabel acuan lewat nama layernya.
129-130	`baris = QgsVectorLayerUtils.createFeature(uji)` lalu `print(...)`	Buat fitur baru dengan semua nilai bawaan terisi, lalu cetak `ID_Survei`, `Tgl_Survei`, dan `Surveyor`.
133-138	`cfg_uji = uji.editFormConfig()` lalu perulangan	Cetak susunan grup formulir dan isinya, plus jenis widget `Foto` dan `Tgl_Survei`. Komentar baris 134 menjelaskan: simpan di kotak dulu agar tidak hilang saat dibaca.

Hasil. Layar menampilkan (tanggal dan nama akun mengikuti komputer Anda):

```
HASIL
Simpan gaya ke GeoPackage: SaveStyleResult.Success (tanpa pesan galat)
Gaya bawaan terbaca: True
Widget Kondisi: ValueMap
Widget Tinggi_Phn: Range
Tata letak: 1 | alias Tinggi_Phn: Tinggi pohon (m)
Fitur salah : ['Kondisi: value is NULL', 'Tinggi_Phn: Tinggi pohon harus 0 sampai 60 m check failed',
'Akurasi_m: Akurasi lebih dari 5 m: ulangi pengukuran bila bisa check failed']
Fitur benar : []
Tabel acuan ditemukan: <QgsVectorLayer: 'Ref_Jenis' (ogr)>
Nilai bawaan -> ID_Survei: SV-001 | Tgl_Survei: 2026-10-07 | Surveyor: [nama akun Windows Anda]
Susunan formulir:
- [Identitas] ID_Survei, KPH, Surveyor, Tgl_Survei
- [Pohon] Jenis, Kondisi, Tinggi_Phn, Diameter_cm
- [Bukti] Foto, Akurasi_m
Widget Foto: ExternalResource | widget Tgl_Survei: DateTime
```

Dari hasil ini Anda tahu: gaya tersimpan dan terbaca ulang. Fitur salah menghasilkan tiga peringatan (kondisi kosong, tinggi di luar rentang, akurasi lebih dari 5 m), sedangkan fitur benar lolos. ID bawaan `SV-001`. Pesan galat dari QGIS berbahasa Inggris (`value is NULL`, `check failed`). `ExternalResource` adalah nama internal widget yang di antarmuka bernama Attachment.

Bagian yang boleh diubah.

- **Baris 10, jalur.** Ganti dengan lokasi `Survei_KPH.gpkg`.
- **Baris 23, pilihan.** Ganti daftar kondisi. Samakan dengan domain di Skrip 20.3.
- **Baris 36-37, batas angka.** Ubah `Min`, `Max`, dan `Step`.
- **Baris 53, aturan akurasi.** Pada uji, `"Akurasi_m" <= 3` dengan pesan "Akurasi lebih dari 3 m" membuat fitur berakurasi 4 m mendapat peringatan.
- **Baris 60, awalan 'SV-'.** Ganti awalan ID.
- **Baris 63-65, ALIAS.** Ganti label. Pada uji, `"Tinggi (meter)"` tercetak sebagai alias `Tinggi_Phn` di baris 101.
- **Baris 86-88, grup.** Tambah, hapus, atau pindahkan kolom antargrup.

Jebakan.

- **Baris 122 memakai proyek terpisah, supaya proyek Anda aman.** Tahap 8 butuh proyek bersih untuk menguji bahwa tabel acuan ditemukan lewat nama layer. Baris `proyek = QgsProject()` membuat proyek baru di memori. Pada uji, saya menambah layer `layer_milik_saya` ke proyek yang terbuka sebelum menjalankan skrip. Sesudahnya daftar layer berisi `['Ref_Jenis', 'layer_milik_saya']`, jadi layer saya selamat. Bila baris itu diganti `QgsProject.instance()` lalu `removeAllMapLayers()`, seluruh layer proyek Anda terhapus. Jangan mengubahnya begitu.
- **Aturan formulir tidak menahan skrip.** Pada uji, saya memuat gaya itu ke layer, lalu menulis satu fitur berkondisi kosong dan tinggi 250 m lewat kode (`addFeature` lalu `commitChanges`). Hasilnya `commit: True` dan satu baris tersimpan. Aturan isian bekerja di formulir, bukan saat skrip menulis. Untuk menolak penulisan skrip, lihat pagar di Skrip 20.5.

- **Skrip 20.2 belum dijalankan.** Skrip berhenti di baris 13 dengan `AssertionError` tanpa pesan, karena `assert` di baris itu tidak diberi teks. Periksa jalur dan isi folder.
- **Baris 29 menambah `Ref_Jenis` ke proyek yang terbuka.** Widget Value Relation membutuhkannya di tahap 6. Menjalankan skrip berulang-ulang bisa menumpuk layer `Ref_Jenis` kembar di proyek itu. Hapus yang lama bila perlu.
- **ID otomatis bisa bentrok.** Rumus "terbesar ditambah satu" aman hanya bila satu orang yang menulis. Dua surveyor luring mendapat nomor yang sama. Skrip 20.9 membuktikannya.
- **Sembilan tahap dalam satu berkas.** Bila ada yang salah, ubah satu blok dulu, jalankan, lalu lanjut ke blok berikutnya.

Latihan variasi.

1. Ubah batas akurasi lunak menjadi 3 m, lalu uji fitur berakurasi 4 m.
2. Ganti label (alias) `Tinggi_Phn` menjadi "Tinggi (meter)".

Contoh jawaban nomor 1, baris 53 menjadi:

```python
lyr.setConstraintExpression(idx("Akurasi_m"), '"Akurasi_m" <= 3', "Akurasi lebih dari 3 m: ulangi pengukuran bila bisa")
```

Lalu tambahkan satu baris setelah baris 118:

```python
print("Akurasi 4 m  :", periksa({"ID_Survei": "SV-001", "Kondisi": "Sehat", "Tinggi_Phn": 12.0, "Akurasi_m": 4.0}))
```

Layar menampilkan `Akurasi 4 m  : ['Akurasi_m: Akurasi lebih dari 3 m: ulangi pengukuran bila bisa check failed']`.

Contoh jawaban nomor 2, di kamus `ALIAS` baris 64 ganti `"Tinggi_Phn": "Tinggi pohon (m)"` menjadi `"Tinggi_Phn": "Tinggi (meter)"`. Layar menampilkan `Tata letak: 1 | alias Tinggi_Phn: Tinggi (meter)`.

> **Pola yang bisa dipakai ulang:** Formulir lewat skrip selalu: **pasang widget** (`widget("KOLOM", "JENIS", {...})`), **pasang aturan** (`setFieldConstraint` atau `setConstraintExpression`), **pasang nilai bawaan** (`QgsDefaultValue`), **susun grup** (`kelompok("JUDUL", [...])`), lalu **simpan gaya** (`saveStyleToDatabaseV2`). Untuk layer lain, ganti **jalur**, **nama layer**, **nama kolom**, dan **isi pilihan**.

Skrip 20.5: Pagar di dalam berkas

Berkas `i3_05_pagar_di_berkas.py`. Di dalam berkasnya tertulis "SKRIP I3.5". Jalankan setelah Skrip 20.2. Skrip ini bekerja pada **salinan** berkas.

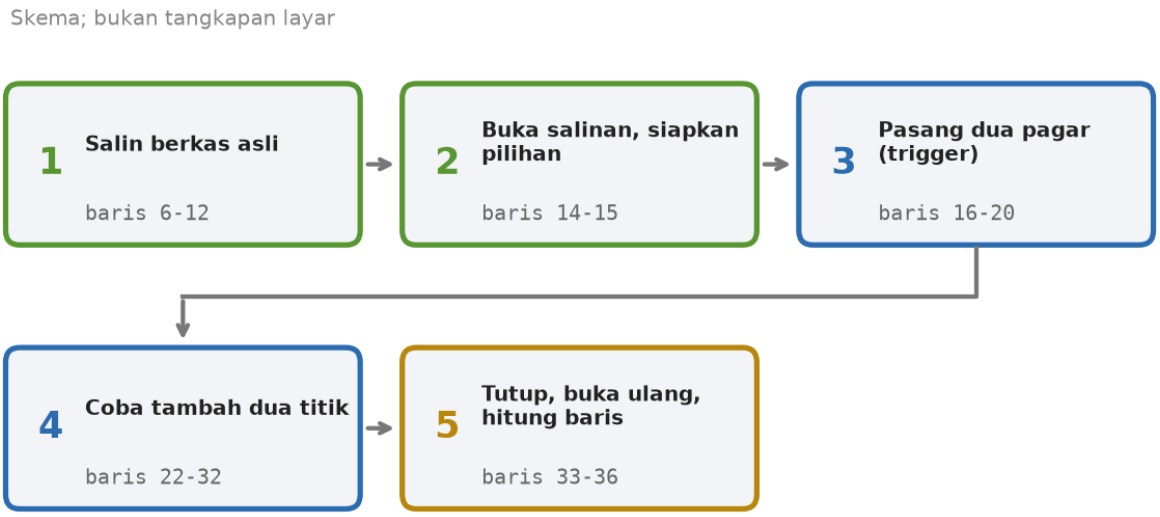
Tujuan. Memasang "pagar" di dalam GeoPackage yang menolak nilai `Kondisi` di luar tiga pilihan baku, lalu mencoba dengan nilai yang salah (`sehat` huruf kecil) dan yang benar (`Sehat`).

Bahan. `Survei_KPH.gpkg` hasil Skrip 20.2. Skrip menyalinnya ke `Survei_KPH_pagar.gpkg`, lalu bekerja di salinan itu. Skrip memakai `osgeo`, bukan `qgis.core`, jadi bisa dijalankan dari terminal `python-qgis`.

Istilah baru untuk skrip ini:

- **Trigger** (di bab ini disebut **pagar**): aturan di dalam basis data yang dijalankan otomatis setiap ada data masuk atau berubah.
- **try dan except**: cara mencoba sesuatu dan menangani kegagalannya tanpa membuat skrip berhenti (Bab 10).
- **Salinan kerja**: berkas tiruan untuk bereksperimen supaya yang asli aman.

Ilustrasi 20.9: Alur skrip 20.5



Skrip utuh:

```python
# -*- coding: utf-8 -*-
# [SKRIP I3.5] Pagar di dalam berkas: trigger SQLite yang menolak nilai Kondisi di luar tiga pilihan baku.
# Penulis: Badar Mubarok Yogaswara
# Bekerja pada SALINAN GeoPackage dari skrip I3.2 (jangan dipasang dulu pada berkas asli sebelum Anda yakin).
# Dijalankan memakai python-qgis.bat (butuh modul osgeo.ogr).
import shutil
from osgeo import ogr

ogr.UseExceptions()
ASLI = r"D:/KPH_Contoh/Survei_KPH.gpkg"
SALINAN = r"D:/KPH_Contoh/Survei_KPH_pagar.gpkg"
shutil.copy(ASLI, SALINAN)

ds = ogr.Open(SALINAN, 1)
PILIHAN = "'Sehat','Terserang Hama','Tebangan Liar'"
for nama, kejadian in [("cek_kondisi_tambah", "INSERT"), ("cek_kondisi_ubah", "UPDATE OF Kondisi")]:
    ds.ExecuteSQL(
        "CREATE TRIGGER %s BEFORE %s ON Titik_Survei "
        "WHEN NEW.Kondisi IS NULL OR NEW.Kondisi NOT IN (%s) "
        "BEGIN SELECT RAISE(ABORT, 'Kondisi harus salah satu dari tiga pilihan baku'); END;" % (nama, kejadian,
PILIHAN))

lyr = ds.GetLayerByName("Titik_Survei")
```

```
for coba in ["sehat", "Sehat"]:
    f = ogr.Feature(lyr.GetLayerDefn())
    f.SetGeometry(ogr.CreateGeometryFromWkt("POINT(312100 9996100)"))
    f.SetField("ID_Survei", "UJI-" + coba)
    f.SetField("Kondisi", coba)
    try:
        lyr.CreateFeature(f)
        print("Kondisi '%s' -> DITERIMA" % coba)
    except Exception as e:
        print("Kondisi '%s' -> DITOLAK: %s" % (coba, str(e).split(":")[-1].strip()))
ds = None

ds = ogr.Open(SALINAN)
print("Jumlah baris tersimpan:", ds.GetLayerByName("Titik_Survei").GetFeatureCount())
```

Penjelasan baris demi baris. Nomor baris mengikuti berkas di atas.

Baris	Tulisan	Artinya
1-5	komentar	Komentar menekankan: bekerja pada SALINAN, dan butuh `python-qgis` (modul `osgeo`).
6-7	`import shutil` dan `from osgeo import ogr`	`shutil` adalah perkakas menyalin berkas. `ogr` membuka dan mengubah berkas vektor.
9	`ogr.UseExceptions()`	Nyalakan alarm: kegagalan GDAL menghentikan skrip, atau bisa ditangkap dengan `try`.
10-11	`ASLI = ...` dan `SALINAN = ...`	Dua jalur: berkas asli dari Skrip 20.2, dan nama salinannya. Ganti sesuai komputer Anda.
12	`shutil.copy(ASLI, SALINAN)`	Salin berkas. Bila `SALINAN` sudah ada, ia ditimpa tanpa peringatan.
14	`ds = ogr.Open(SALINAN, 1)`	Buka salinan dengan mode tulis.
15	`PILIHAN = "'Sehat','Terserang Hama','Tebangan Liar'"`	Teks daftar tiga pilihan, dengan kutip tunggal agar siap dimasukkan ke SQL.
16	`for nama, kejadian in [("cek_kondisi_tambah", "INSERT"), ("cek_kondisi_ubah", "UPDATE OF Kondisi")]:`	Ulangi dua kali. Satu pagar untuk penambahan baris (`INSERT`), satu untuk perubahan kolom `Kondisi` (`UPDATE OF Kondisi`).
17-20	`ds.ExecuteSQL("CREATE TRIGGER %s BEFORE %s ON Titik_Survei WHEN ... BEGIN ... END;" % (...))`	Satu kalimat SQL panjang yang dirakit dari potongan. Bacalah: "buat pagar bernama X, sebelum kejadian Y pada tabel `Titik_Survei`, bila kondisi baru kosong atau tidak ada di daftar, batalkan dengan pesan: Kondisi harus salah satu dari tiga pilihan baku". Perintah `RAISE(ABORT, ...)` yang membatalkan. Tiga tanda `%s` diisi nama, kejadian, dan daftar pilihan.
22	`lyr = ds.GetLayerByName("Titik_Survei")`	Ambil layer yang akan dicoba.
23	`for coba in ["sehat", "Sehat"]:`	Dua percobaan: huruf kecil (salah) lalu huruf besar di awal (benar).

Baris	Tulisan	Artinya
24-27	`f = ogr.Feature(...)`, `f.SetGeometry(...)`, `f.SetField("ID_Survei", "UJI-" + coba)`, `f.SetField("Kondisi", coba)`	Susun satu titik uji. Tanda `+` menyambung teks `UJI-` dengan nilai `coba`.
28-30	`try:` lalu `lyr.CreateFeature(f)` dan `print(... DITERIMA)`	Coba tulis titik ke berkas. Bila berhasil, cetak DITERIMA.
31-32	`except Exception as e:` lalu `print(... str(e).split(":")[-1].strip())`	Bila gagal, blok ini yang jalan. Pesan galat `e` diubah jadi teks, dipotong di tanda titik dua terakhir (`[-1]` berarti yang paling belakang), dan spasi di tepi dibuang. Hasilnya hanya pesan pagar.
33	`ds = None`	Tutup berkas.
35-36	`ds = ogr.Open(SALINAN)` lalu `print(... GetFeatureCount())`	Buka lagi tanpa mode tulis, dan hitung baris yang benar-benar tersimpan.

Hasil. Layar menampilkan:

HASIL

```
Kondisi 'sehat' -> DITOLAK: Kondisi harus salah satu dari tiga pilihan baku
Kondisi 'Sehat' -> DITERIMA
Jumlah baris tersimpan: 1
```

Dari hasil ini Anda tahu: `sehat` (huruf kecil) **ditolak** oleh pagar, dan `Sehat` diterima. Hanya satu baris tersimpan. Beda dengan Skrip 20.3: pagar ini menolak siapa pun yang menulis, termasuk skrip.

Bagian yang boleh diubah.

- **Baris 10-11, jalur.** Ganti `ASLI` dan `SALINAN`.
- **Baris 15, PILIHAN.** Ganti daftar pilihan sah. Juga ubah pesan di baris 20 bila jumlah pilihannya berubah. Pada uji, menambah `'Mati'` ke `PILIHAN`, dengan uji `["Mati", "Sehat"]`, menghasilkan `Kondisi 'Mati' -> DITERIMA`, `Kondisi 'Sehat' -> DITERIMA`, dan 2 baris tersimpan.
- **Baris 16, nama dan kejadian pagar.** Ganti nama pagar. Untuk memagari kolom lain, ubah `Kondisi` di kejadian dan di syarat `WHEN`.
- **Baris 23, nilai uji.** Ganti dengan nilai lain yang ingin dicoba.

Jebakan.

- **NOT IN peka huruf besar dan kecil.** Itu persis yang diperagakan: `sehat` ditolak karena tidak sama dengan `Sehat`.
- **Salinan lama ditimpa.** Pada uji, menjalankan skrip dua kali tidak menimbulkan galat, dan jumlah barisnya tetap 1. Salinan lama (dan pekerjaan di dalamnya) hilang tiap kali.
- **Berkas asli tidak ada.** Layar menampilkan `FileNotFoundError: [Errno 2] No such file or directory: ...`. Jalankan Skrip 20.2 dulu.

- **Pesan dipotong.** Tanpa pemotongan di baris 32, pesan mentahnya `failed to execute insert : Kondisi harus salah satu dari tiga pilihan baku`. Pemotongan memakai tanda titik dua terakhir. Bila format pesan GDAL berubah di versi lain, hasil potongan bisa berbeda. [kemungkinan]
- **Pagar membuat penulisan lebih ketat.** Surveyor yang mengetik di aplikasi lapangan akan mendapat penolakan juga. Pasang pagar hanya bila aturannya sudah pasti.

Latihan variasi.

1. Tambahkan "Mati" sebagai pilihan sah, lalu uji nilai `Mati` dan `Sehat`.
2. Tampilkan pesan galat lengkap tanpa dipotong.

Contoh jawaban nomor 1, baris 15 dan 23 menjadi:

```python
PILIHAN = "'Sehat','Terserang Hama','Tebangan Liar','Mati'"
```

```python
for coba in ["Mati", "Sehat"]:
```

Layar menampilkan `Kondisi 'Mati' -> DITERIMA`, `Kondisi 'Sehat' -> DITERIMA`, dan `Jumlah baris tersimpan: 2`.

Contoh jawaban nomor 2, baris 32 menjadi:

```python
        print("Kondisi '%s' -> DITOLAK: %s" % (coba, str(e)))
```

Layar menampilkan `Kondisi 'sehat' -> DITOLAK: failed to execute insert : Kondisi harus salah satu dari tiga pilihan baku`.

> **Pola yang bisa dipakai ulang:** Pagar di berkas itu satu kalimat SQL: `CREATE TRIGGER nama BEFORE kejadian ON tabel WHEN syarat BEGIN SELECT RAISE(ABORT, 'pesan'); END;`. Yang diganti: **nama pagar**, **kejadian** (`INSERT` atau `UPDATE OF kolom`), **tabel**, **syarat**, dan **pesan**. Selalu coba pada salinan dulu.

Skrip 20.6: Memeriksa mutu data survei

Berkas `i3_06_cek_kualitas.py`. Di dalam berkasnya tertulis "SKRIP I3.6".

Tujuan. Menjalankan delapan pemeriksaan pada data survei (kosong, ejaan tidak baku, rentang tinggi, tanggal, akurasi, ID ganda, koordinat kembar, letak titik) dan mencetak daftar ID yang bermasalah. Skrip ini hanya membaca, tidak mengubah apa pun.

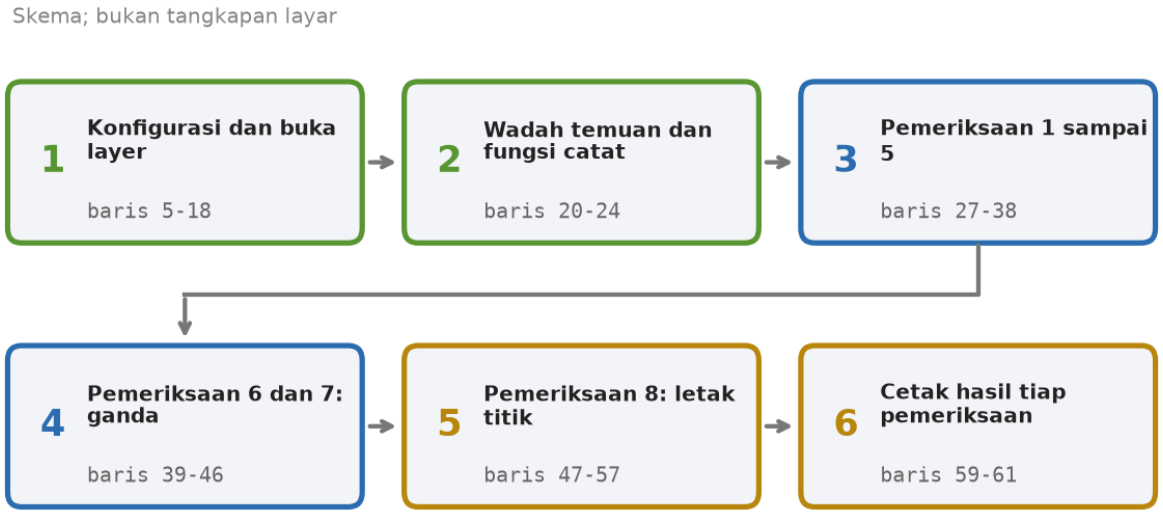
Bahan. `Survei_Lapangan.gpkg` dari `paket-i3`: layer `Titik_Survei` (61 titik, kolom `ID_Survei`, `KPH`, `Kondisi`, `Tinggi_Phn`, `Tgl_Survei`, `Akurasi_m`) dan layer `Batas_KPH` (tiga petak, kolom `NAMA_KPH`). Skrip tidak butuh skrip lain.

Gagasannya seperti daftar periksa barang masuk: ada yang kosong, ada yang di luar rentang, ada yang ganda, ada yang letaknya aneh. Aturan yang ditulis sekali bisa dijalankan setiap kali data baru datang.

Istilah baru untuk skrip ini:

- **Kualitas data**: seberapa layak data dipakai untuk tujuannya.
- **Kelengkapan**: apakah isian wajib sudah terisi.
- **Keabsahan**: apakah nilainya ada di daftar atau rentang yang sah.
- **Konsistensi**: apakah bagian-bagian data saling cocok, misalnya label petak dan letak titik.
- **Keunikan**: apakah ada baris ganda.

Ilustrasi 20.10: Alur skrip 20.6



Skrip utuh:

```python
# -*- coding: utf-8 -*-
# [SKRIP I3.6] Pemeriksa kualitas data survei: kelengkapan, nilai baku, rentang, duplikat, letak, dan akurasi GNSS.
# Penulis: Badar Mubarok Yogaswara
# Dijalankan di Python Console QGIS atau python-qgis.bat. Hanya MEMBACA data; tidak mengubah apa pun.
from collections import Counter
from qgis.core import QgsVectorLayer, QgsGeometry

JALUR = r"D:/KPH_Contoh/paket-i3/Survei_Lapangan.gpkg"
BAKU = {"Sehat", "Terserang Hama", "Tebangan Liar"}
PERIODE = ("2026-09-01", "2026-09-12")          # tanggal survei yang sah (teks ISO, bisa dibandingkan urutannya)
BATAS_TINGGI = (0, 60)
BATAS_AKURASI = 5.0

titik = QgsVectorLayer(JALUR + "|layername=Titik_Survei", "Titik_Survei", "ogr")
batas = QgsVectorLayer(JALUR + "|layername=Batas_KPH", "Batas_KPH", "ogr")
assert titik.isValid() and batas.isValid(), "Layer tidak terbaca: periksa JALUR"
```

```
fitur = list(titik.getFeatures())
petak = {f["NAMA_KPH"]: f.geometry() for f in batas.getFeatures()}

temuan = {}                                        # nama pemeriksaan -> daftar ID_Survei bermasalah


def catat(nama, ids):
    temuan[nama] = sorted(ids)


# 1. Kelengkapan: Kondisi tidak boleh kosong
catat("Kondisi kosong", [f["ID_Survei"] for f in fitur if f["Kondisi"] is None or str(f["Kondisi"]).strip() ==
""])
# 2. Nilai baku: Kondisi harus persis salah satu pilihan (huruf besar-kecil dan spasi dihitung)
catat("Kondisi tidak baku", [f["ID_Survei"] for f in fitur if f["Kondisi"] is not None and f["Kondisi"] not in
BAKU])
# 3. Rentang: tinggi pohon masuk akal
catat("Tinggi di luar rentang", [f["ID_Survei"] for f in fitur
                                 if f["Tinggi_Phn"] is not None and not (BATAS_TINGGI[0] <= f["Tinggi_Phn"] <=
BATAS_TINGGI[1])])
# 4. Tanggal di luar periode survei
catat("Tanggal di luar periode", [f["ID_Survei"] for f in fitur
                                  if f["Tgl_Survei"] is not None and not (PERIODE[0] <=
f["Tgl_Survei"].toString("yyyy-MM-dd") <= PERIODE[1])])
# 5. Akurasi GNSS buruk
catat("Akurasi lebih dari %.0f m" % BATAS_AKURASI, [f["ID_Survei"] for f in fitur if f["Akurasi_m"] is not None
and f["Akurasi_m"] > BATAS_AKURASI])
# 6. ID ganda
hitung_id = Counter(f["ID_Survei"] for f in fitur)
catat("ID ganda", [i for i, n in hitung_id.items() if n > 1])
# 7. Koordinat kembar (dibulatkan ke 1 mm)
hitung_xy = Counter((round(f.geometry().asPoint().x(), 3), round(f.geometry().asPoint().y(), 3)) for f in
fitur)
kembar = {xy for xy, n in hitung_xy.items() if n > 1}
catat("Koordinat kembar", [f["ID_Survei"] for f in fitur
                           if (round(f.geometry().asPoint().x(), 3), round(f.geometry().asPoint().y(), 3)) in
kembar])
# 8. Letak: titik harus berada di dalam salah satu petak, dan petaknya harus sama dengan label KPH
di_luar, salah_label = [], []
for f in fitur:
    g = f.geometry()
    dalam = [nama for nama, pg in petak.items() if g.within(pg)]
    if not dalam:
        di_luar.append(f["ID_Survei"])
    elif f["KPH"] not in dalam:
        salah_label.append(f["ID_Survei"])
catat("Di luar semua petak", di_luar)
catat("Label KPH tidak sesuai letak", salah_label)

print("Jumlah baris diperiksa:", len(fitur))
for nama, ids in temuan.items():
    print("%-32s %2d  %s" % (nama, len(ids), ", ".join(ids)))
```

Penjelasan baris demi baris. Dua blok besar. Nomor baris mengikuti berkas di atas.

Blok A: baris 1 sampai 24, persiapan.

Baris	Tulisan	Artinya
1-4	komentar	Judul, penulis, dan penegasan: hanya MEMBACA, tidak mengubah data.
5	`from collections import Counter`	Panggil perkakas penghitung. `Counter` menghitung berapa kali tiap isi muncul.

Baris	Tulisan	Artinya
6	`from qgis.core import QgsVectorLayer, QgsGeometry`	Panggil pembuka layer. `QgsGeometry` tidak dipakai langsung.
8	`JALUR = r"D:/KPH_Contoh/paket-i3/Survei_Lapangan.gpkg"`	Lokasi data. Ganti dengan jalur Anda.
9	`BAKU = {"Sehat", "Terserang Hama", "Tebangan Liar"}`	Himpunan (*set*): kumpulan isi tanpa kembar, dengan kurung kurawal tanpa titik dua. Cepat untuk menjawab "ada di daftar atau tidak".
10	`PERIODE = ("2026-09-01", "2026-09-12")`	Dua tanggal sah, ditulis sebagai teks tahun-bulan-hari. Teks bentuk ini bisa dibandingkan urutannya seperti tanggal.
11-12	`BATAS_TINGGI = (0, 60)` dan `BATAS_AKURASI = 5.0`	Batas tinggi pohon dan batas akurasi GNSS. Tiga angka ini adalah pengaturan kriteria Anda.
14-16	`titik = ...`, `batas = ...`, `assert ...`	Buka dua layer dan pastikan terbuka.
17	`fitur = list(titik.getFeatures())`	Baca semua titik sekali jadi daftar. Dengan begitu bisa dipakai berulang-ulang di pemeriksaan berikutnya.
18	`petak = {f["NAMA_KPH"]: f.geometry() for f in batas.getFeatures()}`	Kamus yang dibuat dengan satu baris perulangan: nama petak di kiri, bentuk poligonnya di kanan.
20	`temuan = {}`	Kamus kosong untuk menampung hasil: nama pemeriksaan menjadi kunci, daftar ID bermasalah menjadi isinya.
23-24	`def catat(nama, ids):` lalu `temuan[nama] = sorted(ids)`	Resep kecil: simpan daftar ID yang sudah diurutkan di bawah nama pemeriksaan.

Blok B: baris 27 sampai 61, delapan pemeriksaan dan cetakan.

Baris	Tulisan	Artinya
28	`catat("Kondisi kosong", [f["ID_Survei"] for f in fitur if f["Kondisi"] is None or str(f["Kondisi"]).strip() == ""])`	Pemeriksaan 1, kelengkapan. List comprehension dengan syarat `if`: ambil ID titik yang `Kondisi`-nya kosong (`is None`) atau hanya berisi spasi (`str(...).strip() == ""`).
30	`catat("Kondisi tidak baku", [... if f["Kondisi"] is not None and f["Kondisi"] not in BAKU])`	Pemeriksaan 2, keabsahan nilai. Yang terisi tetapi tidak ada di `BAKU`. Huruf besar-kecil dan spasi dihitung.
32-33	`... if f["Tinggi_Phn"] is not None and not (BATAS_TINGGI[0] <= f["Tinggi_Phn"] <= BATAS_TINGGI[1])`	Pemeriksaan 3, rentang. Bentuk `a <= x <= b` membaca "x antara a dan b". Tanda `not` membalik: ambil yang di luar rentang. Bagian `is not None and` adalah pengaman: isian kosong (NULL) dilewati dulu, sehingga skrip tidak berhenti.

Baris	Tulisan	Artinya
35-36	`... if f["Tgl_Survei"] is not None and not (PERIODE[0] <= f["Tgl_Survei"].toString("yyyy-MM-dd") <= PERIODE[1])`	Pemeriksaan 4, tanggal. `toString("yyyy-MM-dd")` mengubah tanggal menjadi teks bentuk tahun-bulan-hari, agar bisa dibandingkan dengan `PERIODE`. Pengamannya sama: tanggal kosong dilewati.
38	`catat("Akurasi lebih dari %.0f m" % BATAS_AKURASI, [... if f["Akurasi_m"] is not None and f["Akurasi_m"] > BATAS_AKURASI])`	Pemeriksaan 5, akurasi GNSS. Nama pemeriksaan ikut berubah mengikuti `BATAS_AKURASI`. Akurasi kosong dilewati.
40-41	`hitung_id = Counter(f["ID_Survei"] for f in fitur)` lalu `catat("ID ganda", [i for i, n in hitung_id.items() if n > 1])`	Pemeriksaan 6, keunikan. Hitung kemunculan tiap ID, lalu ambil ID yang muncul lebih dari satu kali.
43-44	`hitung_xy = Counter((round(x, 3), round(y, 3)) ...)` lalu `kembar = {xy for xy, n in hitung_xy.items() if n > 1}`	Pemeriksaan 7, koordinat kembar. Koordinat dibulatkan ke tiga desimal (satu milimeter), lalu dihitung. Pasangan `(x, y)` dipakai sebagai kunci. Baris 44 membuat himpunan pasangan yang muncul lebih dari sekali.
45-46	`catat("Koordinat kembar", [... if (round(...), round(...)) in kembar])`	Ambil semua titik yang koordinatnya termasuk `kembar`. Karena itu SV-049 muncul dua kali di hasil.
48	`di_luar, salah_label = [], []`	Siapkan dua daftar kosong dalam satu baris.
49-55	`for f in fitur:` lalu `dalam = [...], if not dalam:, elif f["KPH"] not in dalam:`	Pemeriksaan 8, letak. Untuk tiap titik, cari nama petak yang memuatnya (`g.within(pg)`). Bila tidak ada petak, titik di luar. Bila ada tetapi label `KPH` titik tidak termasuk petak itu, label salah. Pola `if ... elif` memilih salah satu cabang.
56-57	`catat("Di luar semua petak", di_luar)` dan `catat("Label KPH tidak sesuai letak", salah_label)`	Simpan dua hasil.
59-61	`print("Jumlah baris diperiksa:", len(fitur))` dan perulangan `for nama, ids in temuan.items():`	Cetak jumlah baris, lalu satu baris per pemeriksaan: nama (lebar 32), jumlah, dan daftar ID disambung koma.

Hasil. Layar menampilkan:

```
HASIL
Jumlah baris diperiksa: 61
Kondisi kosong                   2  SV-031, SV-040
Kondisi tidak baku               3  SV-005, SV-012, SV-020
Tinggi di luar rentang           2  SV-007, SV-033
Tanggal di luar periode          1  SV-050
```

```
Akurasi lebih dari 5 m            2  SV-015, SV-044
ID ganda                          1  SV-049
Koordinat kembar                  4  SV-049, SV-049, SV-051, SV-058
Di luar semua petak               3  SV-059, SV-060, SV-061
Label KPH tidak sesuai letak      2  SV-010, SV-025
```

Dari hasil ini Anda tahu: dari 61 titik, ada sembilan jenis temuan. Dua baris tanpa kondisi, tiga ejaan tidak baku, dua tinggi di luar 0 sampai 60 m, satu tanggal di luar periode, dua akurasi lebih dari 5 m, satu ID ganda, empat titik berkoordinat kembar, tiga titik di luar semua petak, dan dua titik yang labelnya tidak cocok dengan letak. Jumlah baris, kondisi kosong, tidak baku, tinggi, tanggal, akurasi, dan ID ganda sama dengan yang tercatat di `hasil_diharapkan.json` pada paket data. Satu titik bisa kena beberapa pemeriksaan. Karena itu jumlah temuan tidak sama dengan jumlah titik yang bermasalah.

Bagian yang boleh diubah.

- **Baris 8, jalur.** Ganti dengan lokasi data Anda.
- **Baris 9, BAKU.** Daftar nilai sah untuk `Kondisi`. Samakan dengan domain di Skrip 20.3.
- **Baris 10, PERIODE.** Rentang tanggal survei Anda.
- **Baris 11-12, batas tinggi dan akurasi.** Pada uji, `BATAS_AKURASI = 3.0` memberi `Akurasi lebih dari 3 m  18` temuan (dari 2). Judul pemeriksaan ikut berubah otomatis berkat `%.0f`.
- **Baris 27-57, pemeriksaan baru.** Tambah satu panggilan `catat(...)` dengan syarat Anda sendiri.

Jebakan.

- **Isian kosong pada kolom angka dan tanggal tidak ditandai.** Pengaman `is not None and` di baris 33, 36, dan 38 membuat skrip tidak berhenti bila `Tinggi_Phn`, `Tgl_Survei`, atau `Akurasi_m` kosong. Tanpa pengaman, Python berhenti dengan galat seperti `TypeError: '<=' not supported between instances of 'int' and 'NoneType'`. Tetapi isian kosong itu juga tidak masuk daftar "di luar rentang", "di luar periode", atau "akurasi lebih dari 5 m". Hanya kolom `Kondisi` yang punya pemeriksaan kosong (pemeriksaan 1). Untuk kolom lain, tambah pemeriksaan kosong sendiri. [terbukti: skrip jalan sampai selesai pada data uji berisi NULL]
- **Pemeriksaan baru perlu pengaman sendiri.** Pengaman hanya dipasang pada tiga pemeriksaan di atas. Untuk kolom lain, tulis `is not None and` di depan syarat, seperti pada contoh jawaban latihan 2.
- **Satu titik, banyak temuan.** Jumlah temuan dihitung per pemeriksaan. Titik dengan dua masalah muncul dua kali.
- **Koordinat kembar dibulatkan ke milimeter.** Bila satuan peta Anda bukan meter, tiga desimal bermakna lain.
- **Titik tepat di tepi petak.** Fungsi `within` tidak menghitung titik yang persis di garis batas. [kemungkinan: dari dokumentasi fungsi, belum diuji]
- **Impor tak terpakai.** `QgsGeometry` tidak dipakai langsung.

Latihan variasi.

1. Ganti batas akurasi menjadi 3 m.
2. Tambahkan pemeriksaan: diameter lebih dari 45 cm.

Contoh jawaban nomor 1, baris 12 menjadi `BATAS_AKURASI = 3.0`. Layar menampilkan baris `Akurasi lebih dari 3 m` dengan 18 ID.

Contoh jawaban nomor 2, sisipkan sebelum baris 39 (komentar "6. ID ganda"):

```
PYTHON
catat("Diameter lebih dari 45 cm", [f["ID_Survei"] for f in fitur if f["Diameter_cm"] is not None and
f["Diameter_cm"] > 45])
```

Layar menampilkan `Diameter lebih dari 45 cm  13  SV-005, SV-006, SV-010, SV-019, SV-033, SV-038, SV-045, SV-046, SV-048, SV-050, SV-051, SV-052, SV-058.`

> **Pola yang bisa dipakai ulang:** Pemeriksaan data selalu satu bentuk: **catat("NAMA", [ID for f in fitur if SYARAT])**. Untuk aturan baru, ganti **nama pemeriksaan** dan **syarat**. Hasilnya otomatis masuk ke cetakan akhir.

Skrip 20.7: Bertanya ke basis data dengan SQL

Berkas `i3_07_sql_dasar.py`. Di dalam berkasnya tertulis "SKRIP I3.7". Skrip ini bekerja pada **salinan** data, karena satu kalimatnya (`UPDATE`) mengubah isi.

Tujuan. Menjalankan sepuluh kalimat SQL dasar pada GeoPackage survei dan melihat hasilnya: menghitung, menyaring, mengelompokkan, menggabungkan tabel, lalu memperbaiki ejaan.

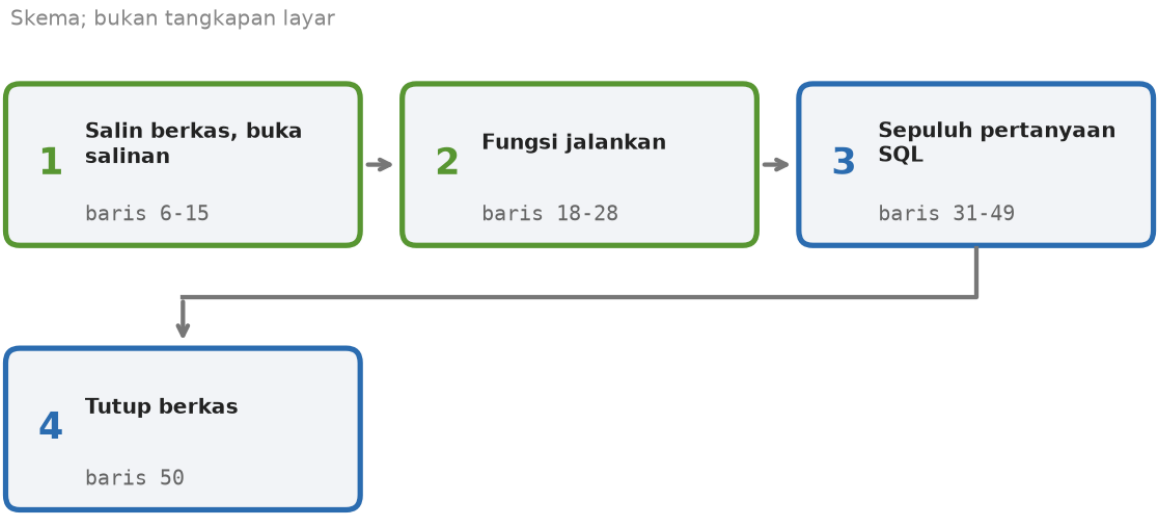
Bahan. `Survei_Lapangan.gpkg` dari `paket-i3` (layer `Titik_Survei`, `Ref_Kondisi`, `Batas_KPH`). Skrip menyalinnya ke `Survei_Lapangan_salinan.gpkg`. Bahasa SQL di GeoPackage adalah SQL milik SQLite.

Gagasannya: **SQL** itu seperti bertanya kepada petugas arsip dengan kalimat yang baku. Anda tidak mencari satu per satu. Anda cukup menyebut tabel, syarat, dan bentuk jawaban yang diinginkan.

Lima istilah baru untuk skrip ini:

- **SELECT ... FROM**: pilih kolom dari sebuah tabel.
- **WHERE**: saring baris yang memenuhi syarat.
- **GROUP BY dan HAVING**: kelompokkan baris, lalu (dengan `HAVING`) saring kelompoknya.
- **JOIN**: gabungkan dua tabel lewat kolom yang sama. `LEFT JOIN` mempertahankan semua baris tabel kiri.
- **UPDATE**: ubah isi baris.

Ilustrasi 20.11: Alur skrip 20.7



Skrip utuh:

PYTHON

```
# -*- coding: utf-8 -*-
# [SKRIP I3.7] SQL dasar pada GeoPackage survei: SELECT, WHERE, ORDER BY, GROUP BY, HAVING, JOIN, UPDATE.
# Penulis: Badar Mubarok Yogaswara
# Bekerja pada SALINAN data (UPDATE mengubah isi). Dijalankan memakai python-qgis.bat (modul osgeo.ogr).
# Bahasa SQL yang dipakai GeoPackage adalah SQL milik SQLite.
import os
import shutil
from osgeo import ogr

ogr.UseExceptions()
ASLI = r"D:/KPH_Contoh/paket-i3/Survei_Lapangan.gpkg"
SALINAN = r"D:/KPH_Contoh/Survei_Lapangan_salinan.gpkg"
os.makedirs(os.path.dirname(SALINAN), exist_ok=True)
shutil.copy(ASLI, SALINAN)
ds = ogr.Open(SALINAN, 1)


def jalankan(judul, sql):
    print("\n-- " + judul)
    print(sql)
    hasil = ds.ExecuteSQL(sql)
    if hasil is None:                            # perintah tanpa tabel hasil (UPDATE, DELETE)
        return
    nama = [hasil.GetLayerDefn().GetFieldDefn(i).GetName() for i in
range(hasil.GetLayerDefn().GetFieldCount())]
    print(" | ".join(nama))
    for baris in hasil:
        print(" | ".join(str(baris.GetField(i)) for i in range(len(nama))))
    ds.ReleaseResultSet(hasil)


jalankan("1. Berapa baris? ", "SELECT COUNT(*) AS jumlah FROM Titik_Survei")
jalankan("2. Pilih kolom dan batasi baris", "SELECT ID_Survei, Kondisi, Tinggi_Phn FROM Titik_Survei LIMIT 3")
jalankan("3. Saring dengan WHERE", "SELECT ID_Survei, KPH, Tinggi_Phn FROM Titik_Survei WHERE Kondisi =
'Tebangan Liar' ORDER BY ID_Survei")
jalankan("4. Hitung per kondisi (GROUP BY)", "SELECT Kondisi, COUNT(*) AS jumlah FROM Titik_Survei GROUP BY
Kondisi ORDER BY jumlah DESC")
jalankan("5. Rata-rata tinggi per KPH (hanya nilai wajar)",
         "SELECT KPH, COUNT(*) AS n, ROUND(AVG(Tinggi_Phn), 1) AS rata_tinggi FROM Titik_Survei "
```

```
        "WHERE Tinggi_Phn BETWEEN 0 AND 60 GROUP BY KPH ORDER BY KPH")
jalankan("6. Cari ID ganda (HAVING)", "SELECT ID_Survei, COUNT(*) AS n FROM Titik_Survei GROUP BY ID_Survei
HAVING COUNT(*) > 1")
jalankan("7. Gabung dengan tabel acuan: Kondisi yang tidak ada di daftar baku (LEFT JOIN)",
        "SELECT t.ID_Survei, t.Kondisi FROM Titik_Survei t LEFT JOIN Ref_Kondisi r ON t.Kondisi = r.Kondisi "
        "WHERE r.Kondisi IS NULL ORDER BY t.ID_Survei")
jalankan("8. Gabung dengan batas KPH lewat nama (JOIN)",
        "SELECT b.NAMA_KPH, COUNT(*) AS jumlah_titik FROM Titik_Survei t JOIN Batas_KPH b ON t.KPH =
b.NAMA_KPH "
        "GROUP BY b.NAMA_KPH ORDER BY b.NAMA_KPH")
jalankan("9. Perbaiki ejaan (UPDATE) pada salinan",
        "UPDATE Titik_Survei SET Kondisi = 'Sehat' WHERE LOWER(TRIM(Kondisi)) = 'sehat'")
jalankan("10. Periksa lagi setelah UPDATE",
        "SELECT t.ID_Survei, t.Kondisi FROM Titik_Survei t LEFT JOIN Ref_Kondisi r ON t.Kondisi = r.Kondisi "
        "WHERE r.Kondisi IS NULL ORDER BY t.ID_Survei")
ds = None
```

Penjelasan baris demi baris. Tiga blok. Nomor baris mengikuti berkas di atas.

Blok A: baris 1 sampai 15, menyiapkan salinan.

Baris	Tulisan	Artinya
1-5	komentar	Penegasan: bekerja pada SALINAN, dan bahasa SQL-nya milik SQLite.
6-8	`import os`, `import shutil`, `from osgeo import ogr`	Panggil perkakas jalur, salin berkas, dan GDAL.
10	`ogr.UseExceptions()`	Nyalakan alarm: kesalahan SQL menghentikan skrip dengan pesan.
11-12	`ASLI = ...` dan `SALINAN = ...`	Dua jalur: berkas asli (dari `paket-i3`), dan salinan kerja. Ganti keduanya.
13	`os.makedirs(os.path.dirname(SALINAN), exist_ok=True)`	Pastikan folder salinan ada.
14-15	`shutil.copy(ASLI, SALINAN)` lalu `ds = ogr.Open(SALINAN, 1)`	Salin berkas (menimpa salinan lama tanpa tanya), lalu buka salinan dengan mode tulis.

Blok B: baris 18 sampai 28, fungsi `jalankan`.

Baris	Tulisan	Artinya
18	`def jalankan(judul, sql):`	Resep yang menerima judul dan satu kalimat SQL. Dipakai sepuluh kali di blok C.
19-20	`print("\n-- " + judul)` lalu `print(sql)`	Cetak judul dan kalimat SQL-nya. Kode `\n` berarti baris baru, jadi ada baris kosong sebelum judul.
21	`hasil = ds.ExecuteSQL(sql)`	Kirim kalimat ke basis data. Jawabannya berupa tabel hasil.
22-23	`if hasil is None:` lalu `return`	Kalimat seperti `UPDATE` tidak menghasilkan tabel. Bila hasilnya kosong, `return` langsung keluar dari resep.
24	`nama = [hasil.GetLayerDefn().GetFieldDefn(i).GetName() for i in range(...)]`	Ambil nama semua kolom hasil. List comprehension memakai `range(...)`, deret angka 0, 1, 2, dan seterusnya sebanyak kolom.

Baris	Tulisan	Artinya
25	`print(... .join(nama))`	Cetak nama kolom, disambung dengan teks pemisah (spasi, garis tegak, spasi) lewat `join`.
26-27	`for baris in hasil:` lalu `print(...join(str(baris.GetField(i)) for i in range(len(nama))))`	Untuk tiap baris hasil, ambil isi tiap kolom (`GetField(i)`), ubah ke teks (`str`), dan cetak disambung dengan pemisah yang sama. Isian kosong tercetak `None`.
28	`ds.ReleaseResultSet(hasil)`	Lepas tabel hasil setelah selesai. Kebiasaan merapikan.

Blok C: baris 31 sampai 50, sepuluh pertanyaan. Tiap pertanyaan memanggil `jalankan(judul, sql)`. Berikut kalimat SQL-nya dan artinya.

No	Kalimat SQL (ringkas)	Artinya
1	`SELECT COUNT(*) AS jumlah FROM Titik_Survei`	Hitung semua baris. `AS jumlah` memberi nama kolom hasil.
2	`SELECT ID_Survei, Kondisi, Tinggi_Phn FROM Titik_Survei LIMIT 3`	Pilih tiga kolom saja, dan hanya tiga baris pertama (`LIMIT 3`).
3	`... WHERE Kondisi = 'Tebangan Liar' ORDER BY ID_Survei`	Saring baris yang kondisinya `Tebangan Liar`, urutkan menurut ID.
4	`SELECT Kondisi, COUNT(*) AS jumlah ... GROUP BY Kondisi ORDER BY jumlah DESC`	Kelompokkan per kondisi, hitung tiap kelompok, urutkan dari terbanyak (`DESC` berarti menurun).
5	`SELECT KPH, COUNT(*) AS n, ROUND(AVG(Tinggi_Phn), 1) AS rata_tinggi ... WHERE Tinggi_Phn BETWEEN 0 AND 60 GROUP BY KPH`	Rata-rata tinggi per petak, hanya dari tinggi yang wajar (0 sampai 60). `AVG` rata-rata, `ROUND(..., 1)` satu desimal.
6	`SELECT ID_Survei, COUNT(*) AS n ... GROUP BY ID_Survei HAVING COUNT(*) > 1`	Cari ID ganda. `HAVING` menyaring kelompok sesudah dihitung. `WHERE` menyaring baris sebelum dikelompokkan.
7	`SELECT t.ID_Survei, t.Kondisi FROM Titik_Survei t LEFT JOIN Ref_Kondisi r ON t.Kondisi = r.Kondisi WHERE r.Kondisi IS NULL`	Gabungkan titik dengan tabel acuan lewat kolom `Kondisi`. Huruf `t` dan `r` adalah nama pendek tabel. Baris yang tidak punya pasangan di acuan memberi `r.Kondisi` kosong. Saring yang kosong: itulah kondisi tidak baku.

No	Kalimat SQL (ringkas)	Artinya
8	`SELECT b.NAMA_KPH, COUNT(*) AS jumlah_titik FROM Titik_Survei t JOIN Batas_KPH b ON t.KPH = b.NAMA_KPH GROUP BY b.NAMA_KPH`	Hitung titik per petak lewat nama. `JOIN` biasa hanya menyimpan baris yang cocok.
9	`UPDATE Titik_Survei SET Kondisi = 'Sehat' WHERE LOWER(TRIM(Kondisi)) = 'sehat'`	Perbaiki ejaan. `TRIM` membuang spasi di tepi, `LOWER` mengubah ke huruf kecil. Semua yang berbunyi "sehat" diubah menjadi `Sehat`.
10	(sama dengan nomor 7)	Periksa lagi setelah perbaikan.

Hasil. Layar menampilkan:

HASIL

```
-- 1. Berapa baris?
SELECT COUNT(*) AS jumlah FROM Titik_Survei
jumlah
61

-- 2. Pilih kolom dan batasi baris
SELECT ID_Survei, Kondisi, Tinggi_Phn FROM Titik_Survei LIMIT 3
ID_Survei | Kondisi | Tinggi_Phn
SV-001 | Terserang Hama | 9.8
SV-002 | Terserang Hama | 16.7
SV-003 | Sehat | 24.5

-- 3. Saring dengan WHERE
SELECT ID_Survei, KPH, Tinggi_Phn FROM Titik_Survei WHERE Kondisi = 'Tebangan Liar' ORDER BY ID_Survei
ID_Survei | KPH | Tinggi_Phn
SV-011 | KPH Beta | 28.4
SV-014 | KPH Beta | 29.3
SV-038 | KPH Beta | 25.0
SV-044 | KPH Beta | 4.3
SV-054 | KPH Gamma | 28.7

-- 4. Hitung per kondisi (GROUP BY)
SELECT Kondisi, COUNT(*) AS jumlah FROM Titik_Survei GROUP BY Kondisi ORDER BY jumlah DESC
Kondisi | jumlah
Sehat | 32
Terserang Hama | 19
Tebangan Liar | 5
None | 2
sehat | 1
Terserang hama | 1
Sehat  | 1

-- 5. Rata-rata tinggi per KPH (hanya nilai wajar)
SELECT KPH, COUNT(*) AS n, ROUND(AVG(Tinggi_Phn), 1) AS rata_tinggi FROM Titik_Survei WHERE Tinggi_Phn BETWEEN
0 AND 60 GROUP BY KPH ORDER BY KPH
KPH | n | rata_tinggi
KPH Alpha | 18 | 18.4
KPH Beta | 21 | 18.8
KPH Gamma | 20 | 17.7

-- 6. Cari ID ganda (HAVING)
SELECT ID_Survei, COUNT(*) AS n FROM Titik_Survei GROUP BY ID_Survei HAVING COUNT(*) > 1
ID_Survei | n
SV-049 | 2
```

```
-- 7. Gabung dengan tabel acuan: Kondisi yang tidak ada di daftar baku (LEFT JOIN)
SELECT t.ID_Survei, t.Kondisi FROM Titik_Survei t LEFT JOIN Ref_Kondisi r ON t.Kondisi = r.Kondisi WHERE
r.Kondisi IS NULL ORDER BY t.ID_Survei
ID_Survei | Kondisi
SV-005 | sehat
SV-012 | Sehat
SV-020 | Terserang hama
SV-031 | None
SV-040 | None

-- 8. Gabung dengan batas KPH lewat nama (JOIN)
SELECT b.NAMA_KPH, COUNT(*) AS jumlah_titik FROM Titik_Survei t JOIN Batas_KPH b ON t.KPH = b.NAMA_KPH GROUP BY
b.NAMA_KPH ORDER BY b.NAMA_KPH
NAMA_KPH | jumlah_titik
KPH Alpha | 19
KPH Beta | 21
KPH Gamma | 21

-- 9. Perbaiki ejaan (UPDATE) pada salinan
UPDATE Titik_Survei SET Kondisi = 'Sehat' WHERE LOWER(TRIM(Kondisi)) = 'sehat'

-- 10. Periksa lagi setelah UPDATE
SELECT t.ID_Survei, t.Kondisi FROM Titik_Survei t LEFT JOIN Ref_Kondisi r ON t.Kondisi = r.Kondisi WHERE
r.Kondisi IS NULL ORDER BY t.ID_Survei
ID_Survei | Kondisi
SV-020 | Terserang hama
SV-031 | None
SV-040 | None
```

Dari hasil ini Anda tahu empat hal. Pertama, ada 61 baris. Kedua, nomor 4 membongkar masalah ejaan: selain tiga nilai baku, ada `sehat`, `Terserang hama`, `Sehat ` (dengan spasi di ujung), dan dua nilai kosong (`None`). Ketiga, nomor 7 menangkap lima baris tidak baku, termasuk dua yang kosong (SV-031 dan SV-040). Keempat, nomor 10 menunjukkan `UPDATE` memperbaiki dua baris (SV-005 dan SV-012). Sisanya, `Terserang hama` dan dua yang kosong, tidak tertangkap aturan `UPDATE` itu, jadi masih harus diperbaiki dengan cara lain.

Bagian yang boleh diubah.

- **Baris 11-12, jalur.** Ganti `ASLI` dan `SALINAN`.
- **Baris 31-49, kalimat SQL.** Ganti syarat, kolom, dan tabel. Ini inti latihan Anda.
- **Baris 37, BETWEEN 0 AND 60.** Ubah rentang tinggi wajar.
- **Baris 46, perbaikan.** Pakai pola `UPDATE tabel SET kolom = nilai WHERE syarat`, tetapi hanya di salinan.

Jebakan.

- **UPDATE tersimpan di salinan, bukan di asli.** Saya memeriksanya dengan `ogrinfo` sesudah skrip selesai: di salinan, `SV-005` dan `SV-012` bernilai `Sehat`. Di berkas asli tetap `sehat` dan `Sehat ` (dengan spasi). Jadi `ds = None` di baris 50 sudah cukup untuk menyimpan perubahan.
- **Kalimat SQL salah ketik.** Pada uji, `SELEC` menghasilkan `RuntimeError: In ExecuteSQL(): sqlite3_prepare_v2(SELEC COUNT(*) AS jumlah FROM Titik_Survei): near "SELEC": syntax error`. Nama kolom yang salah, misalnya `Kondisii`, menghasilkan `no such column: Kondisii`. Pesan menyebut apa yang salah.
- **Salinan lama ditimpa.** Baris 14 menimpa tanpa tanya. Perbaikan di salinan yang belum Anda simpan ke tempat lain hilang.

- **Jangan arahkan UPDATE ke berkas asli.** Ganti `SALINAN` di baris 15 menjadi `ASLI` hanya bila Anda sungguh ingin mengubah data asli, dan setelah membuat cadangan.
- **Isian kosong tampil None.** Itu cara Python menulis "tidak ada nilai". Dalam SQL, bandingkan kosong dengan `IS NULL`, bukan `= NULL`.

Latihan variasi.

1. Tampilkan tiga pohon tertinggi.
2. Hitung rata-rata diameter per jenis pohon.

Contoh jawaban, ganti isi baris 31-49 dengan dua pemanggilan ini (dan biarkan `ds = None` di akhir):

```python
jalankan("L1. Tiga pohon tertinggi", "SELECT ID_Survei, KPH, Tinggi_Phn FROM Titik_Survei ORDER BY Tinggi_Phn DESC LIMIT 3")
jalankan("L2. Rata-rata diameter per jenis", "SELECT Jenis, COUNT(*) AS n, ROUND(AVG(Diameter_cm), 1) AS rata_diameter FROM Titik_Survei GROUP BY Jenis ORDER BY Jenis")
```

Layar menampilkan, untuk nomor 1: `SV-033 | KPH Gamma | 250.0`, `SV-014 | KPH Beta | 29.3`, dan `SV-054 | KPH Gamma | 28.7`. Untuk nomor 2: `Jati | 21 | 34.6`, `Mahoni | 21 | 31.3`, dan `Sengon | 19 | 27.5`. Perhatikan nilai 250: itu pohon setinggi 250 m, jelas salah ketik. SQL membantu menemukannya.

> **Pola yang bisa dipakai ulang:** Menjalankan SQL dari skrip: **salin data**, **buka salinan**, lalu **jalankan("JUDUL", "KALIMAT SQL")**. Untuk pertanyaan baru, ganti **judul** dan **kalimat SQL** saja. Nama tabel dan kolom dibaca dari hasil Skrip 20.2.

Skrip 20.8: Aturan kualitas sebagai ekspresi QGIS

Berkas `i3_08_ekspresi_kualitas.py`. Di dalam berkasnya tertulis "SKRIP I3.8". Jalankan dari Python Console QGIS. Dari terminal `python-qgis` polos, aplikasi QGIS harus disiapkan dulu (lihat Jebakan).

Tujuan. Menulis aturan kualitas sebagai ekspresi QGIS, menghitung berapa baris yang melanggar tiap aturan, lalu menggabung semua aturan menjadi satu kolom virtual `Status_Mutu`.

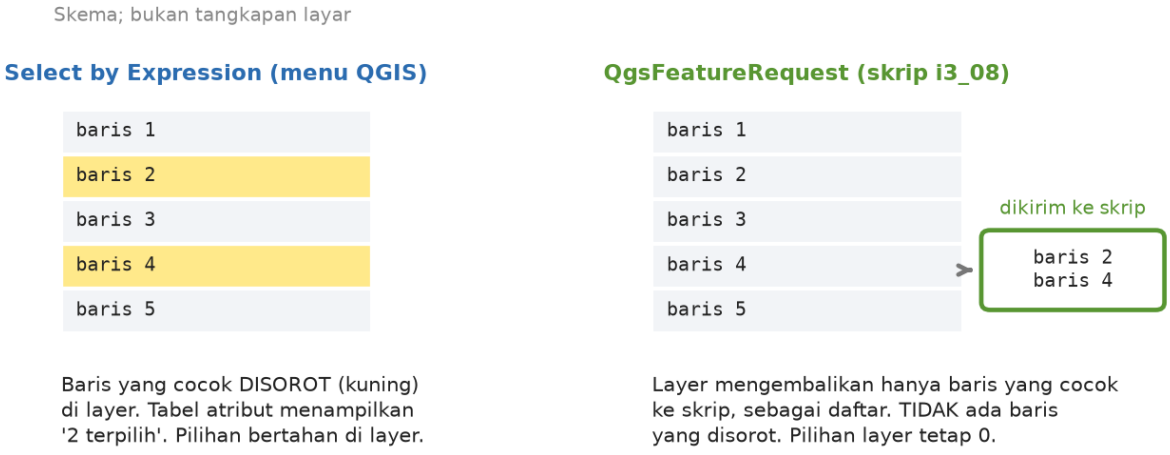
Bahan. `Survei_Lapangan.gpkg` dari `paket-i3` (layer `Titik_Survei` dan `Batas_KPH`). Skrip hanya membaca data. Kolom virtual yang dibuatnya hidup di memori dan tidak ditulis ke berkas. [terbukti: daftar kolom di berkas tidak berubah]

Istilah baru untuk skrip ini:

- **Ekspresi QGIS**: rumus yang QGIS pahami, misalnya `"Tinggi_Phn" NOT BETWEEN 0 AND 60`. Anda pernah memakainya di Select by Expression dan Field Calculator.
- **QgsFeatureRequest**: permintaan data dengan saringan: "berikan hanya baris yang memenuhi ekspresi ini".
- **Konteks ekspresi**: lingkungan tempat ekspresi dihitung. Isinya variabel dan layer di proyek.
- **Kolom virtual**: kolom yang dihitung dari ekspresi tiap kali dibaca, dan tidak disimpan di berkas.
- **CASE WHEN**: urutan pemeriksaan "bila A maka X, bila B maka Y, selain itu Z".

> **Bukan Select by Expression:** Komentar di berkas menyebut tiap aturan dihitung dengan `QgsFeatureRequest` (filter ekspresi). Jangan samakan dengan **Select by Expression**, menu QGIS yang menyorot (memilih) baris di layer. `QgsFeatureRequest` meminta layer mengembalikan baris yang cocok ke skrip, **tanpa menyorot apa pun**. Pada uji, jumlah baris terpilih di layer tetap 0 setelah skrip selesai. Bedanya terlihat di gambar berikut.

Ilustrasi 20.12: Select by Expression lawan QgsFeatureRequest



Ilustrasi 20.13: Alur skrip 20.8



Skrip utuh:

```python
# -*- coding: utf-8 -*-
# [SKRIP I3.8] Aturan kualitas data sebagai ekspresi QGIS: tiap aturan dihitung dengan QgsFeatureRequest
(filter ekspresi)
# dan semuanya digabung menjadi satu kolom virtual Status_Mutu.
# Penulis: Badar Mubarok Yogaswara
# Dijalankan di Python Console QGIS (proyek boleh kosong). Hanya MEMBACA data.
```

```
from qgis.core import (QgsVectorLayer, QgsProject, QgsExpression, QgsFeatureRequest, QgsField,
QgsExpressionContext,
                       QgsExpressionContextUtils)
from qgis.PyQt.QtCore import QMetaType

JALUR = r"D:/KPH_Contoh/paket-i3/Survei_Lapangan.gpkg"
titik = QgsVectorLayer(JALUR + "|layername=Titik_Survei", "Titik_Survei", "ogr")
batas = QgsVectorLayer(JALUR + "|layername=Batas_KPH", "Batas_KPH", "ogr")
QgsProject.instance().addMapLayers([titik, batas])      # ekspresi overlay_within memerlukan layer batas ada di
proyek

ATURAN = {
    "Kondisi kosong": '"Kondisi" IS NULL',
    "Kondisi tidak baku": '"Kondisi" IS NOT NULL AND "Kondisi" NOT IN (\'Sehat\',\'Terserang Hama\',\'Tebangan
Liar\')',
    "Tinggi di luar 0-60": '"Tinggi_Phn" NOT BETWEEN 0 AND 60',
    "Akurasi lebih dari 5 m": '"Akurasi_m" > 5',
    "Tanggal di luar periode": '"Tgl_Survei" NOT BETWEEN to_date(\'2026-09-01\') AND to_date(\'2026-09-12\')',
    "ID ganda": 'count("ID_Survei", group_by:="ID_Survei") > 1',
    "Di luar semua petak": 'NOT overlay_within(\'Batas_KPH\')',
    "Label KPH tidak sesuai": 'overlay_within(\'Batas_KPH\', "NAMA_KPH")[0] <> "KPH"',
}

print("Jumlah baris:", titik.featureCount())
konteks = QgsExpressionContext()
konteks.appendScopes(QgsExpressionContextUtils.globalProjectLayerScopes(titik))    # agar count(), overlay_*()
bekerja
for nama, ekspresi in ATURAN.items():
    permintaan = QgsFeatureRequest().setFilterExpression(ekspresi).setExpressionContext(konteks)
    n = len(list(titik.getFeatures(permintaan)))
    print("%-26s %d" % (nama, n))

# Satu kolom virtual: kata kunci masalah pertama yang ditemukan, atau OK
cabang = " ".join("WHEN %s THEN '%s'" % (e, n.replace("'", "''")) for n, e in ATURAN.items())
titik.addExpressionField("CASE %s ELSE 'OK' END" % cabang, QgsField("Status_Mutu", QMetaType.Type.QString))
hitung = {}
for f in titik.getFeatures():
    hitung[f["Status_Mutu"]] = hitung.get(f["Status_Mutu"], 0) + 1
print("Status_Mutu:", hitung)
```

Penjelasan baris demi baris. Empat blok. Nomor baris mengikuti berkas di atas.

Blok A: baris 1 sampai 13, membuka layer.

Baris	Tulisan	Artinya
1-5	komentar	Penegasan: hanya MEMBACA data, proyek boleh kosong.
6-8	`from qgis.core import (...)` dan `from qgis.PyQt.QtCore import QMetaType`	Panggil perkakas QGIS. `QgsExpression` tidak dipakai langsung. `QMetaType` memberi nama jenis kolom, misalnya teks.
10-12	`JALUR = ...`, `titik = ...`, `batas = ...`	Lokasi data dan dua layer. Ganti jalur dengan milik Anda.
13	`QgsProject.instance().addMapLayers([titik, batas])`	Masukkan kedua layer ke proyek yang sedang terbuka. Komentar di baris itu menjelaskan: fungsi `overlay_within` mencari layer batas di proyek. Setiap kali skrip dijalankan, layer ditambah lagi.

Blok B: baris 15 sampai 24, kamus aturan.

Baris	Tulisan	Artinya
15-24	`ATURAN = {"Kondisi kosong": '"Kondisi" IS NULL', ...}`	Kamus delapan aturan. Kunci adalah nama aturan, isi adalah ekspresi QGIS sebagai teks. Kutip tunggal membungkus ekspresi, jadi tanda kutip tunggal di dalamnya ditulis dengan garis miring terbalik.
16	`'"Kondisi" IS NULL'`	Kondisi kosong.
17	`'"Kondisi" IS NOT NULL AND "Kondisi" NOT IN (...)'`	Terisi tetapi tidak ada di daftar baku. Bagian `IS NOT NULL` perlu karena `NOT IN` tidak menangkap isian kosong.
18-20	`NOT BETWEEN 0 AND 60, > 5, NOT BETWEEN to_date(...) AND to_date(...)`	Tinggi di luar rentang, akurasi lebih dari 5 m, dan tanggal di luar periode.
21	`'count("ID_Survei", group_by:="ID_Survei") > 1'`	ID ganda: hitung baris ber-ID sama, lebih dari satu berarti ganda.
22	`'NOT overlay_within(\'Batas_KPH\')'`	Titik tidak berada di dalam poligon layer `Batas_KPH` mana pun.
23	`'overlay_within(\'Batas_KPH\', "NAMA_KPH")[0] <> "KPH"'`	Nama petak tempat titik berada berbeda dari label `KPH` di titik itu. Tanda `[0]` mengambil isi pertama.

Blok C: baris 26 sampai 32, menghitung tiap aturan.

Baris	Tulisan	Artinya
26	`print("Jumlah baris:", titik.featureCount())`	Cetak jumlah baris, sebagai pembanding.
27-28	`konteks = QgsExpressionContext()` lalu `konteks.appendScopes(...globalProjectLayerScopes(titik))`	Siapkan lingkungan hitung: variabel global, proyek, dan layer `titik`. Tanpa ini, fungsi seperti `count()` dan `overlay_within()` tidak bekerja.
29	`for nama, ekspresi in ATURAN.items():`	Ulangi untuk tiap aturan.
30	`permintaan = QgsFeatureRequest().setFilterExpression(ekspresi).setExpressionContext(konteks)`	Buat permintaan data dengan saringan ekspresi, lalu pasang konteksnya. Dua pemanggilan disambung dengan titik.
31	`n = len(list(titik.getFeatures(permintaan)))`	Minta layer baris yang cocok, kumpulkan jadi daftar, hitung.
32	`print("%-26s %d" % (nama, n))`	Cetak nama aturan dan jumlah pelanggar.

Blok D: baris 34 sampai 40, kolom virtual dan sebarannya.

Baris	Tulisan	Artinya
35	`cabang = " ".join("WHEN %s THEN '%s'" % (e, n.replace("'", "''")) for n, e in ATURAN.items())`	Rakit bagian `WHEN aturan THEN 'nama'` untuk tiap aturan, lalu sambung dengan spasi. Bagian `n.replace("'", "''")` mengganti tiap kutip tunggal di nama aturan dengan dua kutip tunggal. Dalam teks ekspresi, dua kutip berarti satu kutip biasa. Tanpa ini, kutip di nama aturan menutup teks lebih awal dan merusak ekspresi.
36	`titik.addExpressionField("CASE %s ELSE 'OK' END" % cabang, QgsField("Status_Mutu", QMetaType.Type.QString))`	Tambah kolom virtual `Status_Mutu`. Isinya: aturan pertama yang cocok, atau `OK` bila tidak ada. Urutan di kamus menentukan siapa yang menang. Penulisan `QMetaType.Type.QString` adalah gaya QGIS 4. [CEK: belum dicoba di QGIS 3.x]
37-39	`hitung = {}` lalu `for f in titik.getFeatures():` dan `hitung[...] = hitung.get(f["Status_Mutu"], 0) + 1`	Penghitung manual. `hitung.get(kunci, 0)` mengambil isi kunci itu, atau 0 bila belum ada. Tambah satu untuk tiap baris.
40	`print("Status_Mutu:", hitung)`	Cetak sebaran status.

Hasil. Layar menampilkan:

```
HASIL
Jumlah baris: 61
Kondisi kosong             2
Kondisi tidak baku         3
Tinggi di luar 0-60        2
Akurasi lebih dari 5 m     2
Tanggal di luar periode    1
ID ganda                   2
Di luar semua petak        3
Label KPH tidak sesuai     2
Status_Mutu: {'OK': 44, 'Kondisi tidak baku': 3, 'Tinggi di luar 0-60': 2, 'Label KPH tidak sesuai': 2,
'Akurasi lebih dari 5 m': 2, 'Kondisi kosong': 2, 'ID ganda': 2, 'Tanggal di luar periode': 1, 'Di luar semua
petak': 3}
```

Dari hasil ini Anda tahu: jumlah pelanggar tiap aturan cocok dengan Skrip 20.6, kecuali satu hal. `ID ganda` di sini 2, sedangkan di Skrip 20.6 hanya 1. Sebabnya: Skrip 20.6 menghitung ID yang ganda (SV-049, satu ID). Skrip ini menghitung baris yang ber-ID ganda (dua baris SV-049). Lalu `Status_Mutu` memperlihatkan 44 baris `OK` dan 17 baris bermasalah, dan 44 + 17 = 61. Tidak ada baris yang kena dua aturan sekaligus pada data ini, jadi jumlah 17 sama dengan jumlah semua pelanggaran.

Bagian yang boleh diubah.

- **Baris 10, jalur.** Ganti dengan lokasi data Anda.
- **Baris 15-24, aturan.** Tambah, hapus, atau ubah ekspresi dan batasnya. Pada uji, menambah `"Diameter lebih dari 45 cm": '"Diameter_cm" > 45'` menghasilkan 13 pelanggar, tetapi di `Status_Mutu` hanya 10 (tiga baris sudah lebih dulu kena aturan lain). Satu baris `Label KPH tidak sesuai` pun tergeser. Ingat: aturan pertama yang cocok menang.

- **Baris 19, batas akurasi.** Pada uji, `> 3` (dan nama aturan "Akurasi lebih dari 3 m") menghasilkan 18 pelanggar.
- **Baris 36, ELSE 'OK'.** Ganti kata `OK` dengan label lain.

Jebakan.

- **Kolom virtual tidak disimpan.** Pada uji, setelah skrip jalan, `Status_Mutu` ada di layer di memori, tetapi tidak ada di daftar kolom bila layer dibaca ulang dari berkas. Bila Anda butuh kolom tetap, buat dengan cara lain.
- **Menjalankan ulang menambah layer kembar.** Pada uji, dua kali menjalankan menghasilkan daftar layer proyek `['Batas_KPH', 'Batas_KPH', 'Titik_Survei', 'Titik_Survei']`. Hapus layer lama dulu, atau mulai dari proyek kosong.
- **Kutip tunggal di nama aturan sudah diamankan di baris 35.** Pada uji, nama aturan `Akurasi lebih dari 5 m'x` tetap tampil utuh di `Status_Mutu` karena kutipnya digandakan. Bila Anda mengubah baris 35 dan membuang `replace`, seluruh `Status_Mutu` bisa bernilai kosong (`{None: 61}`) tanpa pesan galat. Pengaman ini hanya untuk nama aturan. Kutip di dalam ekspresi aturan tetap harus Anda tulis dengan benar.
- **Paling aman dari Python Console QGIS.** Dari terminal `python-qgis` polos, tanpa aplikasi QGIS yang disiapkan lebih dulu, skrip ini berhenti dengan `Segmentation fault` pada uji (kode keluar 139 di Git Bash; di PowerShell -1073741819). Memanggil `Processing.initialize()` saja tidak menolong: skrip tetap berhenti. Yang menolong adalah menyiapkan aplikasinya di awal skrip, yaitu `from qgis.core import QgsApplication`, `app = QgsApplication([], False)`, lalu `app.initQgis()`. Pada uji, dengan tiga baris itu skrip berjalan benar, dengan atau tanpa `Processing.initialize()` (lihat Bab 21).
- **Batas akurasi tertulis di dua tempat.** Aturan di baris 19 dan batas di Skrip 20.4. Bila Anda mengubah satu, ubah yang lain.

Latihan variasi.

1. Tambahkan aturan "Diameter lebih dari 45 cm".
2. Ubah batas akurasi menjadi 3 m.

Contoh jawaban nomor 1, tambahkan satu baris di kamus `ATURAN` (misalnya sebelum aturan `Label KPH tidak sesuai`):

```python
    "Diameter lebih dari 45 cm": '"Diameter_cm" > 45',
```

Layar menampilkan `Diameter lebih dari 45 cm  13` dan, di `Status_Mutu`, `'OK': 35` dengan `'Diameter lebih dari 45 cm': 10`.

Contoh jawaban nomor 2, baris 19 menjadi:

```python
    "Akurasi lebih dari 3 m": '"Akurasi_m" > 3',
```

Layar menampilkan `Akurasi lebih dari 3 m  18` dan `'OK': 33` di `Status_Mutu`.

> **Pola yang bisa dipakai ulang:** Aturan kualitas dengan ekspresi: **kamus ATURAN** (nama dan ekspresi), **QgsFeatureRequest().setFilterExpression(...)** untuk menghitung, dan **CASE WHEN ... END** untuk menggabungkan. Untuk aturan baru, tambah satu baris di kamus. Ganti **jalur**, **nama layer**, dan **ekspresi**.

Skrip 20.9: Dua surveyor, satu nomor yang sama

Berkas `i3_09_gabung_dan_bentrok_id.py`. Di dalam berkasnya tertulis "SKRIP I3.9". Jalankan setelah Skrip 20.2 dan 20.4, dari Python Console QGIS.

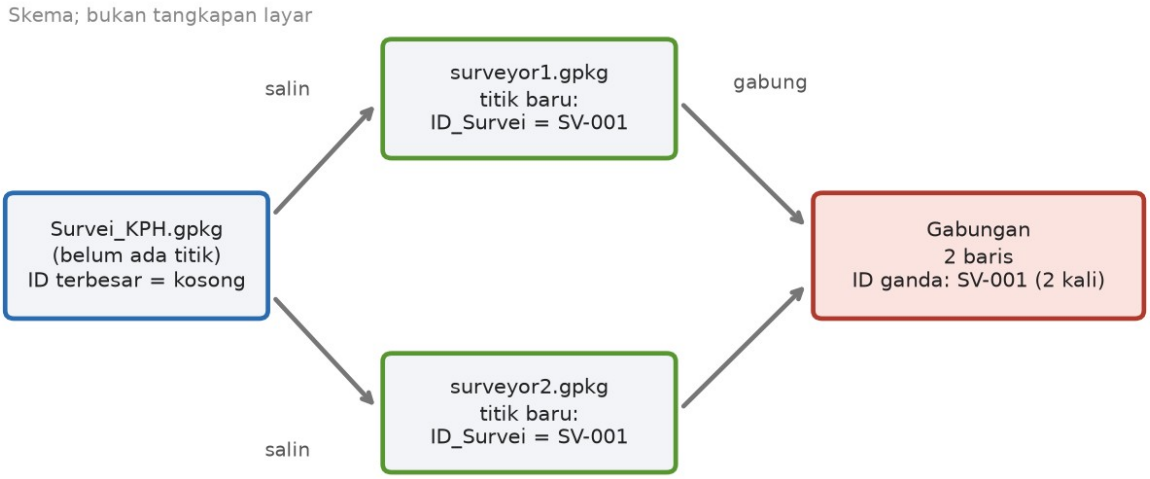
Tujuan. Menunjukkan mengapa nomor ID otomatis "terbesar ditambah satu" bisa bentrok bila dua surveyor bekerja luring dari salinan yang sama. Lalu menggabungkan hasil keduanya dan mencari ID ganda.

Bahan. `Survei_KPH.gpkg` hasil Skrip 20.2, lengkap dengan gaya dari Skrip 20.4. Skrip membuat dua salinan (`surveyor1.gpkg`, `surveyor2.gpkg`) di folder `KPH_Contoh`, menambah satu titik di tiap salinan, lalu menggabungkannya di memori.

Istilah baru untuk skrip ini:

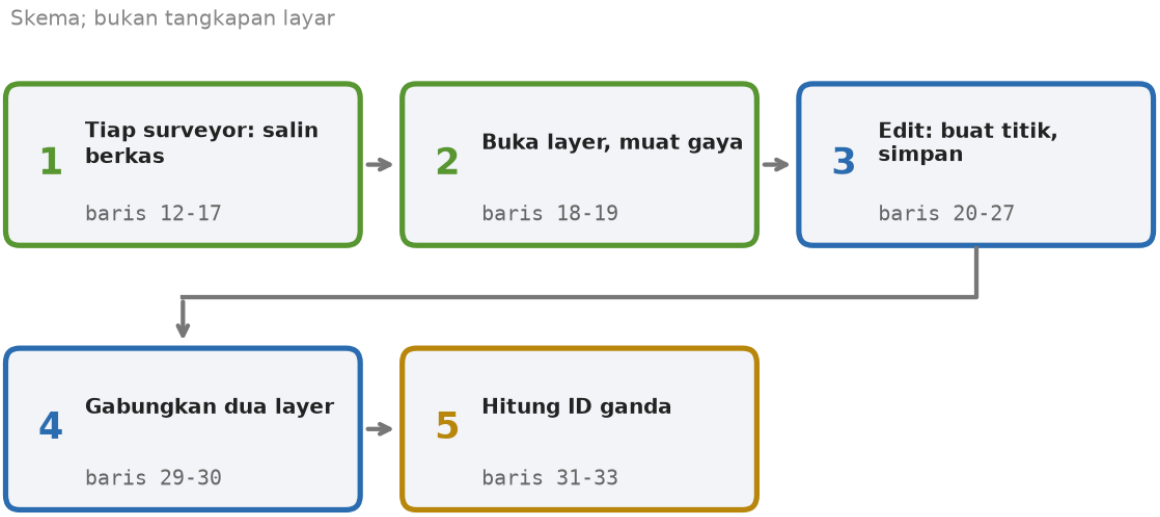
- **Luring** (*offline*): bekerja tanpa jaringan.
- **Bentrok ID**: dua baris berbeda mendapat nomor pengenal yang sama.
- **Menggabung** (*merge*): menyatukan isi dua layer menjadi satu layer.

Ilustrasi 20.14: Dua surveyor mendapat nomor yang sama



Keduanya bekerja luring dan tidak tahu satu sama lain. Rumus maksimum + 1 memberi nomor yang sama.

Ilustrasi 20.15: Alur skrip 20.9



Skrip utuh:

```python
# -*- coding: utf-8 -*-
# [SKRIP I3.9] Dua surveyor bekerja luring dari salinan yang sama. Skrip ini menunjukkan mengapa nomor ID otomatis
# "maksimum + 1" bisa bentrok, lalu menggabungkan hasil keduanya dengan Processing dan mencari ID ganda.
# Penulis: Badar Mubarok Yogaswara
# Jalankan setelah skrip I3.2 dan I3.4 (butuh Survei_KPH.gpkg beserta gayanya). Dijalankan di Python Console QGIS.
import os
import shutil
import processing
from collections import Counter
from qgis.core import QgsVectorLayer, QgsVectorLayerUtils, QgsGeometry, QgsPointXY, QgsProject

KERJA = r"D:/KPH_Contoh"
ASLI = KERJA + "/Survei_KPH.gpkg"
punya = {}
for nama, (x, y) in {"surveyor1": (312050, 9996050), "surveyor2": (312150, 9996060)}.items():
    salinan = "%s/%s.gpkg" % (KERJA, nama)
    shutil.copy(ASLI, salinan)                                  # tiap surveyor membawa salinan sendiri
    lyr = QgsVectorLayer(salinan + "|layername=Titik_Survei", nama, "ogr")
    lyr.loadDefaultStyle()                                      # ID otomatis datang dari gaya yang tersimpan
    lyr.startEditing()
    f = QgsVectorLayerUtils.createFeature(lyr, QgsGeometry.fromPointXY(QgsPointXY(x, y)))
    f["Kondisi"] = "Sehat"
    f["Tinggi_Phn"] = 12.0
    lyr.addFeature(f)
    assert lyr.commitChanges(), lyr.commitErrors()
    punya[nama] = lyr
    print("%s membuat ID: %s" % (nama, f["ID_Survei"]))

gabungan = processing.run("native:mergevectorlayers", {
    "LAYERS": [punya["surveyor1"], punya["surveyor2"]], "CRS": "EPSG:32749", "OUTPUT": "memory:Gabungan"})
["OUTPUT"]
hitung = Counter(f["ID_Survei"] for f in gabungan.getFeatures())
print("Jumlah baris setelah digabung:", gabungan.featureCount())
print("ID ganda:", {k: v for k, v in hitung.items() if v > 1})
```

Penjelasan baris demi baris. Dua blok. Nomor baris mengikuti berkas di atas.

Blok A: baris 1 sampai 27, tiap surveyor membuat satu titik.

Baris	Tulisan	Artinya
1-5	komentar	Judul dan prasyarat: jalankan setelah Skrip 20.2 dan 20.4.
6-10	`import os`, `import shutil`, `import processing`, `from collections import Counter`, `from qgis.core import (...)`	Panggil perkakas. `os` dan `QgsProject` diimpor tetapi tidak dipakai.
12-13	`KERJA = r"D:/KPH_Contoh"` dan `ASLI = KERJA + "/Survei_KPH.gpkg"`	Folder kerja, dan jalur berkas asli yang dibuat Skrip 20.2. Tanda `+` menyambung teks.
14	`punya = {}`	Kamus kosong untuk menyimpan layer tiap surveyor.
15	`for nama, (x, y) in {"surveyor1": (312050, 9996050), "surveyor2": (312150, 9996060)}.items():`	Ulangi dua kali. Kamus ditulis langsung di baris `for`: nama surveyor dan koordinat titik yang akan ia buat. Tanda kurung `(x, y)` membuka pasangan koordinat.
16	`salinan = "%s/%s.gpkg" % (KERJA, nama)`	Rakit jalur berkas salinan, misalnya `D:/KPH_Contoh/surveyor1.gpkg`.
17	`shutil.copy(ASLI, salinan)`	Tiap surveyor membawa salinan sendiri. Salinan lama ditimpa tanpa tanya.
18-19	`lyr = QgsVectorLayer(...)` lalu `lyr.loadDefaultStyle()`	Buka layer `Titik_Survei` di salinan itu, lalu muat gaya bawaannya. Nilai bawaan ID otomatis datang dari gaya yang disimpan Skrip 20.4.
20	`lyr.startEditing()`	Mulai mode sunting.
21	`f = QgsVectorLayerUtils.createFeature(lyr, QgsGeometry.fromPointXY(QgsPointXY(x, y)))`	Buat satu fitur baru di titik `(x, y)`, dengan semua nilai bawaan terisi, termasuk ID.
22-23	`f["Kondisi"] = "Sehat"` dan `f["Tinggi_Phn"] = 12.0`	Isi dua kolom.
24	`lyr.addFeature(f)`	Tambahkan ke layer.
25	`assert lyr.commitChanges(), lyr.commitErrors()`	Simpan sunting. Bila gagal, skrip berhenti dan menampilkan daftar galatnya.
26-27	`punya[nama] = lyr` dan `print("%s membuat ID: %s" % (nama, f["ID_Survei"]))`	Simpan layer di kamus, lalu cetak ID yang dibuat.

Blok B: baris 29 sampai 33, menggabung dan mencari ID ganda.

Baris	Tulisan	Artinya
29-30	`gabungan = processing.run("native:mergevectorlayers", {...})["OUTPUT"]`	Gabungkan dua layer jadi satu. `LAYERS` adalah daftar layer yang digabung. `CRS` memilih sistem koordinat hasil. Nilai `"memory:Gabungan"` berarti hasil disimpan di layer memori bernama `Gabungan`.
31	`hitung = Counter(f["ID_Survei"] for f in gabungan.getFeatures())`	Hitung berapa kali tiap ID muncul.
32	`print("Jumlah baris setelah digabung:", gabungan.featureCount())`	Cetak jumlah baris hasil gabungan.
33	`print("ID ganda:", {k: v for k, v in hitung.items() if v > 1})`	Cetak kamus berisi hanya ID yang muncul lebih dari satu kali. Kamus dibuat dengan satu baris perulangan dan syarat `if`.

Hasil. Layar menampilkan:

```
HASIL
surveyor1 membuat ID: SV-001
surveyor2 membuat ID: SV-001
Jumlah baris setelah digabung: 2
ID ganda: {'SV-001': 2}
```

Dari hasil ini Anda tahu: kedua surveyor mendapat `SV-001`. Setelah digabung ada dua baris, dan ID `SV-001` muncul dua kali. Rumus "terbesar ditambah satu" memang cuma bekerja bila hanya satu orang yang mengedit. Kedua surveyor mulai dari salinan yang sama, jadi sama-sama melihat "terbesar = kosong" dan mengambil nomor 1.

Bagian yang boleh diubah.

- **Baris 12, KERJA.** Ganti dengan folder kerja Anda.
- **Baris 15, nama dan koordinat surveyor.** Tambah surveyor atau ganti titiknya. Pada uji, menambah `"surveyor3": (312250, 9996070)` dan `LAYERS` berisi ketiga layer (`list(punya.values())`) menghasilkan tiga ID `SV-001` dan `ID ganda: {'SV-001': 3}`.
- **Baris 22-23, isian.** Ubah nilai kondisi dan tinggi titik.
- **Baris 30, CRS.** Ganti bila data Anda memakai sistem koordinat lain.

Jebakan.

- **Gaya dari Skrip 20.4 belum tersimpan.** ID otomatis tidak terisi, nilainya `None`. Pada uji, layar menampilkan `surveyor1 membuat ID: None`, `surveyor2 membuat ID: None`, dan `ID ganda: {None: 2}`. Itu bukan bukti keberhasilan. Itu tanda gaya tidak termuat.
- **Butuh Python Console QGIS.** Modul `processing` hanya ada di dalam QGIS. Dari terminal `python-qgis` polos, layar menampilkan `ModuleNotFoundError: No module named 'processing'`.

- **Salinan lama ditimpa.** Baris 17 menimpa `surveyor1.gpkg` dan `surveyor2.gpkg` bila sudah ada.
- **Hasil gabungan hanya di memori.** Layer `Gabungan` hilang saat QGIS ditutup. Ekspor dulu bila ingin menyimpannya.
- **Impor tak terpakai.** `os` dan `QgsProject` tidak dipakai.

Latihan variasi.

1. Tambahkan surveyor ketiga.
2. Beri tiap surveyor awalan ID sendiri, supaya tidak bentrok.

Contoh jawaban nomor 1, baris 15 dan 30 menjadi:

```
PYTHON
for nama, (x, y) in {"surveyor1": (312050, 9996050), "surveyor2": (312150, 9996060), "surveyor3": (312250, 9996070)}.items():
```

```
PYTHON
    "LAYERS": list(punya.values()), "CRS": "EPSG:32749", "OUTPUT": "memory:Gabungan"})["OUTPUT"]
```

Layar menampilkan `surveyor3 membuat ID: SV-001`, `Jumlah baris setelah digabung: 3`, dan `ID ganda: {'SV-001': 3}`.

Contoh jawaban nomor 2, sisipkan satu baris sebelum baris 22:

```
PYTHON
    f["ID_Survei"] = "S%s-001" % nama[-1]
```

Layar menampilkan `surveyor1 membuat ID: S1-001`, `surveyor2 membuat ID: S2-001`, `Jumlah baris setelah digabung: 2`, dan `ID ganda: {}`. Awalan per surveyor membuat nomor tidak bentrok. Tanda `nama[-1]` mengambil huruf terakhir nama, yaitu angka 1 atau 2.

> **Pola yang bisa dipakai ulang:** Menggabungkan hasil banyak surveyor: **salin berkas per orang**, **buat dan simpan fitur**, **native:mergevectorlayers** untuk menggabung, lalu **Counter(...)** untuk mencari ID ganda. Ganti **nama dan koordinat surveyor**, **nama kolom ID**, dan **CRS**. Hindari bentrok dengan **awalan per surveyor** atau kode unik.

Cek paham

1. Apa beda akurasi dan presisi pada pembacaan GNSS? Angka mana di Skrip 20.1 yang mengukur masing-masing?
2. Mengapa domain di Skrip 20.3 tidak menolak nilai `ngawur`, padahal pagar di Skrip 20.5 menolak `sehat`?
3. Mengapa `ID ganda` bernilai 2 di Skrip 20.8, tetapi 1 di Skrip 20.6?

Jawaban.

1. Akurasi adalah seberapa dekat rata-rata pembacaan ke titik benar: angka **Bias**. Presisi adalah seberapa rapat pembacaan satu sama lain: angka **Presisi**. Angka RMSE, CEP50, dan R95 memuat keduanya.
2. Domain hanya petunjuk bagi formulir QGIS. Penulisan lewat skrip tidak melewati formulir. Pagar (trigger) dijalankan oleh basis data setiap ada tulisan masuk, siapa pun penulisnya.
3. Skrip 20.6 menghitung ID yang ganda (satu ID: SV-049). Skrip 20.8 menghitung baris yang ber-ID ganda (dua baris SV-049).

Kesalahan umum

- **Membiarkan jalur contoh.** Jalur `D:/KPH_Contoh` adalah milik penulis. Perbaikan: ganti dengan jalur Anda di bagian atas tiap skrip.
- **Menjalankan skrip tidak berurutan.** Skrip 20.3, 20.4, 20.5, dan 20.9 butuh `Survei_KPH.gpkg` dari Skrip 20.2. Perbaikan: jalankan 20.2 lebih dulu, dan jangan jalankan 20.2 dua kali.
- **Menjalankan Skrip 20.4 berkali-kali di proyek yang sama.** Baris 29 menambah layer `Ref_Jenis` tiap kali jalan, sehingga layer kembar menumpuk. Perbaikan: hapus layer lama, atau mulai dari proyek kosong.
- **Mengira domain atau aturan formulir sudah menjaga data.** Uji menunjukkan penulisan lewat skrip tetap lolos. Perbaikan: pasang pagar (Skrip 20.5) bila data penting, dan tetap periksa mutu (Skrip 20.6 sampai 20.8).

Ringkasan dan latihan

- **Akurasi** (bias) dan **presisi** (sebaran) berbeda. **R95** memberi gambaran terburuk yang jujur.
- **GeoPackage** memuat banyak layer, tabel acuan, domain, dan gaya dalam satu berkas.
- Tiga lapis penjagaan: **domain** di berkas, **formulir** QGIS, dan **pagar** (trigger). Hanya pagar yang menolak skrip.
- **Kualitas data** diperiksa dengan aturan yang ditulis sekali: dengan Python (Skrip 20.6), SQL (20.7), atau ekspresi QGIS (20.8).
- **ID otomatis** "terbesar ditambah satu" bentrok bila banyak orang bekerja luring. Beri awalan per surveyor.

Latihan. Skrip 20.5 memagari kolom `Kondisi`. Aturan mana di Skrip 20.8 (misalnya tinggi di luar 0 sampai 60) tidak bisa ditangkap oleh pagar `Kondisi` itu? Pagar baru seperti apa yang Anda butuhkan, dan di bagian mana dari kalimat `CREATE TRIGGER` yang Anda ganti?

Tabel perbandingan: sembilan skrip sekilas

Skrip	Mengubah berkas?	Butuh dulu	Dijalankan dari
20.1 statistik GNSS	Tidak, hanya membaca	tidak ada	Konsol QGIS atau `python-qgis`
20.2 buat GeoPackage	Ya, membuat berkas baru	tidak ada	Konsol QGIS atau `python-qgis`

Skrip	Mengubah berkas?	Butuh dulu	Dijalankan dari
20.3 domain	Ya, menanam domain	20.2	Konsol QGIS atau `python-qgis`
20.4 formulir	Ya, menulis gaya	20.2	Konsol QGIS atau `python-qgis`
20.5 pagar	Ya, pada salinan	20.2	`python-qgis`
20.6 cek kualitas	Tidak, hanya membaca	tidak ada	Konsol QGIS atau `python-qgis`
20.7 SQL dasar	Ya, pada salinan	tidak ada	`python-qgis`
20.8 ekspresi kualitas	Tidak, kolom virtual di memori	tidak ada	Konsol QGIS (terminal bila Processing disiapkan)
20.9 gabung dan bentrok ID	Ya, membuat dua salinan	20.2 dan 20.4	Konsol QGIS saja

BAB 21: Memanggil Alat dari Skrip: Satu Perintah, Satu Analisis, Satu Resep untuk Banyak Berkas

> **Berdasarkan Seri I4:** Skrip bab ini berasal dari Buku I4, *Otomasi Dasar dengan Python* (Intermediate Class), berkas `b2_01_jalankan_alat.py`, `b2_02_rantai_vektor.py`, `b2_03_raster_alat.py`, `b3_01_batch_ndvi.py`, `b3_02_batch_per_kph.py`, dan `b3_03_batch_latar_belakang.py`. Di dalam berkasnya tertulis nomor Skrip 2.1 sampai 3.3, mengikuti penomoran buku itu. Di bab ini keenamnya diberi nomor Skrip 21.1 sampai 21.6.

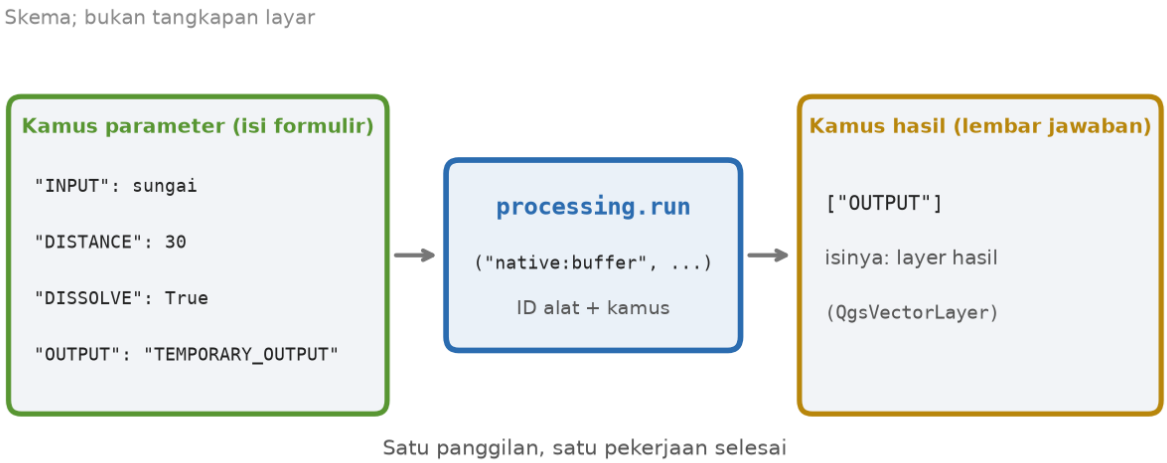
Studi kasus: "Sempadan sungai, tiga puluh ubin, dan sore yang pendek"

Kepala Seksi minta dua hal sebelum akhir pekan. Pertama, luas jalur sempadan sungai 30 meter di tiap KPH, lengkap dengan jumlah titik panasnya. Kedua, NDVI untuk semua ubin foto drone yang baru masuk. Mengklik alat satu per satu akan menghabiskan sore Anda. Dengan skrip, Anda memanggil alat Processing lewat tulisan, merangkainya, mengulanginya untuk banyak berkas, dan membiarkan QGIS tetap hidup selagi bekerja.

Konsep: memanggil alat dengan formulir pesanan

Di QGIS, Anda memakai alat lewat jendela dengan kolom isian. Di skrip, kolom isian itu ditulis sebagai **kamus parameter**. Bayangkan formulir pesanan di loket: Anda menyebut nama layanan, mengisi kolomnya, lalu petugas mengembalikan lembar jawaban. Lembar jawaban itu adalah **kamus hasil**.

Ilustrasi 21.1: Satu panggilan, kamus masuk dan kamus keluar



Nama alat di loket itu disebut **ID alat**, misalnya `native:buffer`. Bagian sebelum titik dua adalah nama kelompok alat (`native` berarti alat bawaan QGIS), dan bagian sesudahnya nama alatnya. Kamus sudah dibahas di Bab 5, jadi yang baru di sini hanyalah isinya.

Lima istilah baru Anda:

- **Alat Processing**: satu fungsi analisis di QGIS, seperti Buffer, yang juga ada di Processing Toolbox.
- **ID alat**: nama unik alat untuk skrip, misalnya `native:buffer`.
- **Kamus parameter**: isian alat, berupa pasangan nama dan nilai.
- **Kamus hasil**: jawaban alat, diambil dengan nama keluarannya, misalnya `["OUTPUT"]`.
- **Keluaran sementara** (`TEMPORARY_OUTPUT`): hasil perantara yang tidak ditulis ke berkas pilihan Anda.

Dua cara menjalankan, dan satu tambahan untuk Processing

Bab 14 menjelaskan dua cara menjalankan skrip PyQGIS: di dalam QGIS, atau mandiri dengan kerangka `QgsApplication`. Semua skrip bab ini berjalan dengan cara pertama tanpa tambahan, sebab Konsol Python QGIS sudah memuat `qgis.core` dan `processing`.

Untuk cara mandiri, ada tambahan khusus. Python biasa tidak menemukan perkakas `processing` begitu saja. Pada uji, tulisan `import processing` di skrip mandiri menghasilkan `ModuleNotFoundError: No module named 'processing'`. Kerangka berikut sudah diuji pada QGIS 4.0.2.

```python
import sys
from qgis.core import QgsApplication

QgsApplication.setPrefixPath(r"C:/Program Files/QGIS 4.0.2/apps/qgis", True)
app = QgsApplication([], False)
app.initQgis()

sys.path.append(r"C:/Program Files/QGIS 4.0.2/apps/qgis/python/plugins")
from processing.core.Processing import Processing
Processing.initialize()
from qgis.analysis import QgsNativeAlgorithms
QgsApplication.processingRegistry().addProvider(QgsNativeAlgorithms())

# ... tempel isi skrip Anda di sini ...

app.exitQgis()
```

Baris	Artinya
`sys.path.append(...)`	Tambahkan folder plugin QGIS ke daftar tempat Python mencari perkakas. Jalur ini milik komputer penulis dan akan berbeda di komputer Anda [CEK].
`Processing.initialize()`	Nyalakan sistem Processing.
`addProvider(QgsNativeAlgorithms())`	Daftarkan alat bawaan (`native:`) ke buku telepon alat.

Kerangka ini hanya untuk cara mandiri. Bila bagian `Processing.initialize()` dan `addProvider` dibuang, skrip tetap bisa mengimpor `processing`, tetapi alatnya tidak dikenal. Pada uji, hasilnya `QgsProcessingException: Error: Algorithm native:buffer not found`.

Skrip 21.1: Memanggil satu alat, Buffer

Tujuan. Mencari ID alat Buffer, mengintip nama-nama isiannya, menjalankannya pada layer sungai dengan jarak 30 meter, lalu membaca hasilnya.

Bahan. Paket data `paket-i4`, berkas `Sungai.gpkg` (dua garis sungai). Berkas `b2_01_jalankan_alat.py` dijalankan dari Editor Skrip QGIS, atau dengan kerangka mandiri di atas. Ganti `DATA` di baris 6 dengan lokasi `paket-i4` Anda.

Skrip ini bekerja dalam enam tahap. Gambar berikut menunjukkan urutannya.

Ilustrasi 21.2: Alur skrip 21.1



Skrip utuh:

```python
# [SKRIP 2.1: Menjalankan satu alat Processing dari Python (Buffer)]
# Penulis: Badar Mubarok Yogaswara
import processing
from qgis.core import QgsProject, QgsVectorLayer, QgsApplication

DATA = "D:/Latihan/paket-i4"
HASIL = "D:/Latihan/hasil_i4"

sungai = QgsVectorLayer(DATA + "/Sungai.gpkg", "Sungai", "ogr")

# 0. Mencari ID alat dengan kata kunci
for a in sorted(QgsApplication.processingRegistry().algorithms(), key=lambda x: x.id()):
    if a.id().startswith("native:") and "buffer" in a.id():
        print("ID:", a.id(), "-", a.displayName())

# 1. Mengintip "formulir" alat: nama parameter dan artinya
alat = QgsApplication.processingRegistry().algorithmById("native:buffer")
print("Alat:", alat.displayName())
for p in alat.parameterDefinitions():
    print("  ", p.name(), "=", p.description())

# 2. Menjalankan alat. Parameter berupa kamus (dictionary): nama parameter -> nilai.
hasil = processing.run("native:buffer", {
```

```
    "INPUT": sungai,
    "DISTANCE": 30,                 # satuan meter karena CRS-nya UTM
    "SEGMENTS": 8,
    "END_CAP_STYLE": 0,             # 0 = bulat
    "JOIN_STYLE": 0,                # 0 = bulat
    "MITER_LIMIT": 2,
    "DISSOLVE": True,               # gabungkan semua penyangga jadi satu bentuk
    "OUTPUT": "TEMPORARY_OUTPUT",   # keluaran sementara, tidak menulis berkas
})

# 3. Hasilnya juga kamus. Kuncinya nama keluaran. Untuk vektor sementara, isinya sudah berupa layer.
sempadan = hasil["OUTPUT"]
print("Jenis keluaran:", type(sempadan).__name__)
print("Fitur:", sempadan.featureCount())
luas = sum(f.geometry().area() for f in sempadan.getFeatures()) / 10000
print("Luas sempadan sungai:", round(luas, 2), "ha")

# 4. Beri nama yang jelas lalu tampilkan di proyek
sempadan.setName("Sempadan_30m")
QgsProject.instance().addMapLayer(sempadan)
print("Layer di proyek:", [l.name() for l in QgsProject.instance().mapLayers().values()])
```

Penjelasan baris demi baris. Dua blok. Blok pertama mengenali alatnya, blok kedua menjalankannya.

Blok A: mengenali alat (baris 3-20).

Baris	Tulisan	Artinya
3-4	`import processing` dan `from qgis.core import ...`	Panggil perkakas Processing dan tiga perkakas QGIS. `QgsProject` dan `QgsVectorLayer` dibahas di Bab 14.
6-7	`DATA` dan `HASIL`	Dua kotak jalur folder. Di skrip ini hanya `DATA` yang dipakai.
9	`sungai = QgsVectorLayer(...)`	Buka layer sungai, seperti di Bab 14.
12-14	`for a in sorted(...algorithms(), key=lambda x: x.id()):`	Ambil daftar semua alat, urutkan menurut ID. `lambda x: x.id()` adalah resep mini yang berbunyi "ambil ID-nya" (dibahas di Bab 7). Untuk tiap alat yang ID-nya diawali `native:` dan memuat kata `buffer`, cetak ID dan nama tampilannya.
17-18	`alat = ...algorithmById("native:buffer")`	Ambil satu alat dengan ID-nya, lalu cetak nama tampilannya.
19-20	`for p in alat.parameterDefinitions():`	Ulangi untuk tiap isian alat. Cetak nama isian (`p.name()`) dan artinya (`p.description()`). Inilah "formulir" alat itu.

Blok B: menjalankan dan membaca hasil (baris 22-44).

Baris	Tulisan	Artinya
23-32	`hasil = processing.run("native:buffer", {...})`	Jalankan alat. Isian pertama adalah ID alat, isian kedua kamus parameter. Hasilnya disimpan di kotak `hasil`.
24	`"INPUT": sungai`	Layer masukan.

Baris	Tulisan	Artinya
25	`"DISTANCE": 30`	Jarak penyangga. Satuannya mengikuti CRS layer. Di data ini CRS-nya UTM, jadi 30 meter.
26-29	`"SEGMENTS"`, `"END_CAP_STYLE"`, `"JOIN_STYLE"`, `"MITER_LIMIT"`	Pengaturan bentuk ujung dan sudut. Angka 0 berarti "bulat" (lihat tabel pilihan di bawah).
30	`"DISSOLVE": True`	Gabungkan semua penyangga menjadi satu bentuk.
31	`"OUTPUT": "TEMPORARY_OUTPUT"`	Simpan hasil sebagai keluaran sementara.
35-37	`sempadan = hasil["OUTPUT"]`	Ambil jawaban dari kamus hasil dengan kunci `OUTPUT`. Untuk vektor sementara, isinya sudah berupa layer. Cetak jenisnya dengan `type(...).__name__` dan jumlah fitur dengan `featureCount()`.
38	`luas = sum(f.geometry().area() for f in ...) / 10000`	Jumlahkan luas semua fitur, lalu ubah meter persegi menjadi hektare.
42-43	`sempadan.setName(...)` lalu `addMapLayer(...)`	Beri nama yang jelas, lalu tampilkan di proyek. Tanpa `addMapLayer`, layer sementara tidak muncul di panel Layers.
44	`[l.name() for l in ...mapLayers().values()]`	Cetak nama semua layer di proyek.

Angka pilihan untuk `END_CAP_STYLE` dan `JOIN_STYLE` bisa dilihat dengan `processing.algorithmHelp("native:buffer")`. Pada uji, bantuan itu mencatat `END_CAP_STYLE`: 0 Round, 1 Flat, 2 Square. Untuk `JOIN_STYLE`: 0 Round, 1 Miter, 2 Bevel.

Hasil. Layar menampilkan (berdasarkan uji pada QGIS 4.0.2):

```
HASIL
ID: native:buffer - Buffer
ID: native:bufferbym - Variable width buffer (by M value)
ID: native:multiringconstantbuffer - Multi-ring buffer (constant distance)
ID: native:singlesidedbuffer - Single sided buffer
ID: native:taperedbuffer - Tapered buffers
ID: native:wedgebuffers - Create wedge buffers
Alat: Buffer
   INPUT = Input layer
   DISTANCE = Distance
   SEGMENTS = Segments
   END_CAP_STYLE = End cap style
   JOIN_STYLE = Join style
   MITER_LIMIT = Miter limit
   DISSOLVE = Dissolve result
   SEPARATE_DISJOINT = Keep disjoint results separate
   OUTPUT = Buffered
Jenis keluaran: QgsVectorLayer
Fitur: 1
Luas sempadan sungai: 4.48 ha
Layer di proyek: ['Sempadan_30m']
```

Ada enam alat yang ID-nya memuat kata "buffer". Alat Buffer punya sembilan isian, dan skrip hanya mengisi delapan. Isian `SEPARATE_DISJOINT` tidak diisi, jadi alat memakai nilai bawaannya. Hasil: satu fitur seluas 4,48 hektare.

Bagian yang boleh diubah.

- **Baris 6, lokasi data.** Ganti sesuai komputer Anda.
- Baris 13, kata `buffer`. Ganti dengan kata lain, misalnya `clip` atau `raster`, untuk mencari alat lain.
- **Baris 17, ID alat.** Ganti dengan ID lain, lalu baca isiannya di blok yang mengintip formulir.
- **Baris 25, jarak.** Ganti angka 30. Satuan mengikuti CRS.
- Baris 30, `DISSOLVE`. `False` menghasilkan satu fitur per sungai.
- **Baris 42, nama layer.** Ganti sesuai kebutuhan.

Jebakan. Semua kesalahan di bawah sudah dicoba pada QGIS 4.0.2.

- **Nama isian salah ketik tidak menimbulkan galat.** `"DISTANC": 30` menggantikan `"DISTANCE": 30`. Skrip jalan mulus, tetapi alat memakai nilai bawaannya sendiri. Luas menjadi `1.44 ha`, bukan `4.48 ha`. Tidak ada pesan apa pun. Salin nama isian dari hasil blok yang mengintip formulir.
- **Layer masukan gagal dibuka tidak menimbulkan galat.** Bila `Sungai.gpkg` diganti `Sungay.gpkg`, hasilnya `Fitur: 0` dan `Luas sempadan sungai: 0.0 ha`. Alat bekerja pada layer kosong. Periksa `sungai.isValid()` lebih dulu (Bab 14).
- **ID alat salah ketik menimbulkan galat.** `"native:buffr"` menghasilkan `QgsProcessingException: Error: Algorithm native:buffr not found`.
- **Keluaran ke berkas bukan layer.** Bila `"OUTPUT"` diisi jalur berkas, `hasil["OUTPUT"]` berisi **teks** jalur itu. Baris 36 mencetak `Jenis keluaran: str`, lalu baris 37 berhenti dengan `AttributeError: 'str' object has no attribute 'featureCount'`.
- **Satuan jarak mengikuti CRS.** `Sungai.gpkg` diubah ke EPSG:4326 (derajat), lalu skrip dijalankan. Hasilnya `Luas sempadan sungai: 0.28 ha`, angka yang tidak bermakna, karena 30 dibaca sebagai 30 derajat. Pastikan CRS berupa meter sebelum memakai `DISTANCE`.

Latihan variasi.

1. Jalankan lagi dengan `DISSOLVE` bernilai `False`. Berapa fitur dan berapa luasnya?
2. Mengapa luasnya berbeda dari 4,48 hektare?

Contoh jawaban (sudah diuji). Untuk nomor 1, ubah baris 30 menjadi `"DISSOLVE": False,`. Layar menampilkan `Fitur: 2` dan `Luas sempadan sungai: 4.82 ha`. Untuk nomor 2, kemungkinan besar bagian yang tumpang tindih di pertemuan dua sungai terhitung dua kali bila tidak digabung. Selisihnya 0,34 hektare [kemungkinan, belum diukur terpisah].

> **Pola yang bisa dipakai ulang:** Kerangka memanggil alat: `hasil = processing.run("ID_ALAT", {"INPUT": LAYER, ..., "OUTPUT": "TEMPORARY_OUTPUT"})`, lalu `layer_hasil = hasil["OUTPUT"]`. Ganti ID alat dan isi kamus. Daftar nama isian ada di `alat.parameterDefinitions()`.

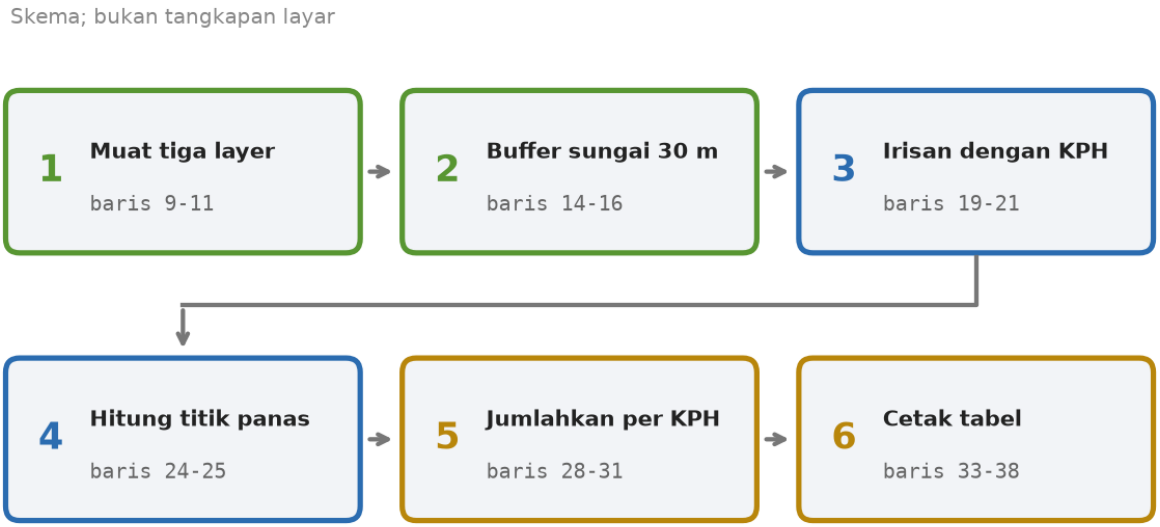
Skrip 21.2: Merangkai alat vektor

Keluaran satu alat bisa langsung menjadi masukan alat berikutnya, seperti ban berjalan di pabrik. Skrip ini merangkai tiga alat, lalu menggabungkan hasilnya dalam satu tabel kecil.

Tujuan. Menghitung luas sempadan sungai 30 meter di tiap KPH, persennya terhadap luas KPH, dan jumlah titik panas per KPH.

Bahan. Paket data `paket-i4`: `Batas_KPH.gpkg`, `Sungai.gpkg`, dan `Hotspot_Nasional.gpkg`. Berkas `b2_02_rantai_vektor.py`. Ganti `DATA` di baris 6.

Ilustrasi 21.3: Alur skrip 21.2



Skrip utuh:

```python
# [SKRIP 2.2: Rantai alat vektor: sempadan sungai dan titik panas per KPH]
# Penulis: Badar Mubarok Yogaswara
import processing
from qgis.core import QgsProject, QgsVectorLayer

DATA = "D:/Latihan/paket-i4"
HASIL = "D:/Latihan/hasil_i4"

batas = QgsVectorLayer(DATA + "/Batas_KPH.gpkg", "Batas_KPH", "ogr")
sungai = QgsVectorLayer(DATA + "/Sungai.gpkg", "Sungai", "ogr")
hotspot = QgsVectorLayer(DATA + "/Hotspot_Nasional.gpkg", "Hotspot_Nasional", "ogr")

# Langkah 1: penyangga (buffer) 30 m di kiri-kanan sungai
sempadan = processing.run("native:buffer", {
    "INPUT": sungai, "DISTANCE": 30, "SEGMENTS": 8, "END_CAP_STYLE": 0, "JOIN_STYLE": 0,
    "MITER_LIMIT": 2, "DISSOLVE": True, "OUTPUT": "TEMPORARY_OUTPUT"})["OUTPUT"]

# Langkah 2: irisan (intersection). Hasil langkah 1 langsung menjadi masukan langkah 2.
irisan = processing.run("native:intersection", {
    "INPUT": batas, "OVERLAY": sempadan, "INPUT_FIELDS": ["NAMA_KPH"],
    "OVERLAY_FIELDS": [], "OUTPUT": "TEMPORARY_OUTPUT"})["OUTPUT"]

# Langkah 3: hitung titik panas di tiap KPH
```

```
dengan_titik = processing.run("native:countpointsinpolygon", {
    "POLYGONS": batas, "POINTS": hotspot, "FIELD": "JML_HOTSPOT", "OUTPUT": "TEMPORARY_OUTPUT"})["OUTPUT"]

# Langkah 4: gabungkan angkanya dalam satu tabel kecil
luas_sempadan = {}
for f in irisan.getFeatures():
    nama = f["NAMA_KPH"]
    luas_sempadan[nama] = luas_sempadan.get(nama, 0) + f.geometry().area() / 10000

print("KPH         luas(ha)  sempadan(ha)  persen  titik_panas")
for f in dengan_titik.getFeatures():
    nama = f["NAMA_KPH"]
    luas = f.geometry().area() / 10000
    sem = luas_sempadan.get(nama, 0)
    print("%-11s %8.2f %13.2f %7.1f %12d" % (nama, luas, sem, 100 * sem / luas, f["JML_HOTSPOT"]))
```

Skrip ini mirip Skrip 21.1, jadi hanya bagian yang **baru** yang dijelaskan. Baris `import`, `DATA`, dan pemuatan layer (baris 3-11) sudah dibahas sebelumnya.

Baris	Tulisan	Artinya
14-16	`sempadan = processing.run("native:buffer", {...})["OUTPUT"]`	Sama dengan Skrip 21.1, tetapi `["OUTPUT"]` ditulis langsung di ujung. Kotak `sempadan` langsung berisi layer hasil.
19-21	`irisan = processing.run("native:intersection", {...})`	Iris layer batas KPH dengan sempadan. `INPUT` adalah batas, `OVERLAY` adalah sempadan. Hasil langkah 1 menjadi masukan langkah 2.
20	`"INPUT_FIELDS": ["NAMA_KPH"]`	Kolom dari `INPUT` yang dibawa ke hasil. Daftar kosong (`[]`) pada `OVERLAY_FIELDS` berarti, menurut keterangan alat, semua kolom dibawa.
24-25	`dengan_titik = processing.run("native:countpointsinpolygon", {...})`	Hitung titik panas di dalam tiap KPH. `POLYGONS` adalah batas, `POINTS` adalah titik panas, dan `FIELD` adalah nama kolom penghitung baru, `JML_HOTSPOT`.
28	`luas_sempadan = {}`	Siapkan kamus kosong.
29-31	`for f in irisan.getFeatures():`	Untuk tiap irisan: ambil nama KPH, lalu tambahkan luasnya (dalam hektare) ke jumlah milik KPH itu. `luas_sempadan.get(nama, 0)` berarti "ambil jumlah sejauh ini, atau 0 bila belum ada".
33	`print("KPH luas(ha) ...")`	Cetak judul tabel. Spasinya diketik tangan agar sejajar dengan baris 38.
34-37	`for f in dengan_titik.getFeatures():`	Untuk tiap KPH: ambil nama, luas, dan luas sempadannya (`.get(nama, 0)`).
38	`print("%-11s %8.2f %13.2f %7.1f %12d" % (...))`	Cetak satu baris tabel. `%-11s` teks rata kiri selebar 11 karakter, `%8.2f` angka dua desimal selebar 8, `%d` bilangan bulat. Persen dihitung dengan `100 * sem / luas`. Format `%` dibahas di Bab 9.

Hasil. Layar menampilkan (berdasarkan uji pada QGIS 4.0.2):

```
HASIL
KPH         luas(ha)  sempadan(ha)  persen  titik_panas
KPH Alpha       4.00          1.18    29.4            2
KPH Beta        4.00          1.39    34.6            5
KPH Gamma       8.00          1.50    18.7            5
```

Dari tabel ini Anda tahu: sempadan paling luas di KPH Gamma (1,50 hektare), tetapi persen terbesar ada di KPH Beta (34,6 persen).

Bagian yang boleh diubah.

- Baris 15, `"DISTANCE": 30`. Lebar sempadan.
- Baris 25, `"FIELD": "JML_HOTSPOT"`. Nama kolom penghitung. Bila Anda mengubahnya, ubah juga `f["JML_HOTSPOT"]` di baris 38.
- Baris 20, `INPUT_FIELDS`. Tambahkan nama kolom lain yang ingin dibawa.
- **Baris 33 dan 38, format tabel.** Lebar kolom boleh disesuaikan, tetapi judul dan baris data harus selaras.

Jebakan. Semua dicoba pada QGIS 4.0.2.

- **Nama kolom tidak sama.** `f["JML_HOTSPOT"]` diganti `f["JML_TITIK"]`: judul tabel tercetak, lalu skrip berhenti dengan `KeyError: 'JML_TITIK'`.
- **Layer sungai gagal dibuka.** Dengan `Sungay.gpkg`, tidak ada galat. Kolom sempadan berisi `0.00` dan persen `0.0` untuk ketiga KPH. Angka tampak wajar, padahal kosong.
- **Jarak nol.** `"DISTANCE": 0` menghentikan alat Intersection dengan `QgsProcessingException: Could not write feature into OUTPUT: Could not add feature with geometry type GeometryCollection to layer of type MultiPolygon`. Jarak nol menghasilkan bentuk yang tidak cocok untuk alat berikutnya.
- **Pembagian dengan nol.** Baris 38 membagi dengan `luas`. Data latihan tidak punya KPH berluas nol, jadi galat `ZeroDivisionError` tidak muncul di sini (Bab 10 membahas penangannya).

Latihan variasi. Ubah lebar sempadan menjadi 50 meter. Bagaimana tabelnya berubah?

Contoh jawaban (sudah diuji). Ganti baris 15 menjadi `"INPUT": sungai, "DISTANCE": 50, "SEGMENTS": 8, "END_CAP_STYLE": 0, "JOIN_STYLE": 0,`. Layar menampilkan:

```
HASIL
KPH         luas(ha)  sempadan(ha)  persen  titik_panas
KPH Alpha       4.00          1.69    42.2            2
KPH Beta        4.00          2.32    58.0            5
KPH Gamma       8.00          2.54    31.8            5
```

> **Pola yang bisa dipakai ulang:** Kerangka rantai: `langkah1 = processing.run("ALAT_1", {...})["OUTPUT"]`, lalu `langkah2 = processing.run("ALAT_2", {"INPUT": langkah1, ...})["OUTPUT"]`. Tiap langkah memakai hasil langkah sebelumnya sebagai masukan. Ganti ID alat dan isiannya.

Skrip 21.3: Alat raster, NDVI dan tinggi tajuk

Raster adalah kisi petak (piksel), jadi alatnya sedikit berbeda dari vektor. Skrip ini memakai dua alat raster dan satu alat statistik. Hasilnya dua angka per KPH.

Dua istilah baru. **NDVI** adalah indeks kehijauan tanaman dari citra, dihitung `(NIR - Merah) / (NIR + Merah)`. Nilai tinggi berarti vegetasi rapat. **CHM** adalah tinggi tajuk pohon, dihitung dari raster tinggi permukaan (DSM) dikurangi raster tinggi tanah (DTM). **Statistik zonal** meringkas nilai raster di dalam tiap poligon.

Tujuan. Menghitung NDVI dan tinggi tajuk, lalu mencetak rata-ratanya untuk tiap KPH.

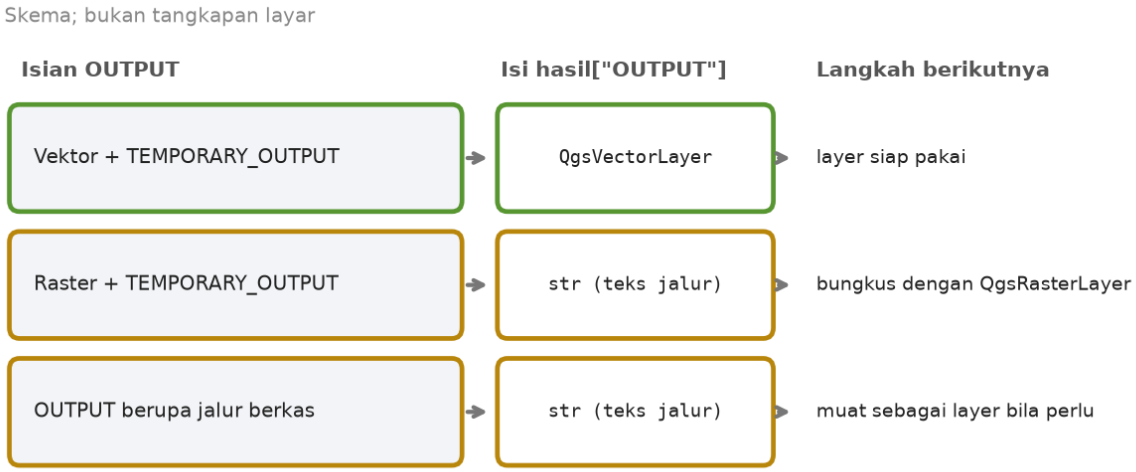
Bahan. Paket data `paket-i4`: `Batas_KPH.gpkg`, `Citra_KPH.tif` (empat band, band 1 Merah dan band 4 NIR), `DSM_Drone.tif`, dan `DTM_Drone.tif`. Berkas `b2_03_raster_alat.py`. Ganti `DATA` di baris 6.

Ilustrasi 21.4: Alur skrip 21.3



Hal yang perlu Anda tahu dulu: keluaran raster sementara berbeda dari vektor sementara.

Ilustrasi 21.5: Isi hasil OUTPUT bergantung pada jenisnya



Skrip utuh:

```
PYTHON
# [SKRIP 2.3: Alat raster dari Python: NDVI, tinggi tajuk (CHM), dan statistik per KPH]
# Penulis: Badar Mubarok Yogaswara
import processing
from qgis.core import QgsProject, QgsVectorLayer, QgsRasterLayer

DATA = "D:/Latihan/paket-i4"
HASIL = "D:/Latihan/hasil_i4"

batas = QgsVectorLayer(DATA + "/Batas_KPH.gpkg", "Batas_KPH", "ogr")
citra = QgsRasterLayer(DATA + "/Citra_KPH.tif", "Citra_KPH", "gdal")   # band 1=Merah, 4=NIR
dsm = QgsRasterLayer(DATA + "/DSM_Drone.tif", "DSM", "gdal")
dtm = QgsRasterLayer(DATA + "/DTM_Drone.tif", "DTM", "gdal")

# 1. NDVI = (NIR - Merah) / (NIR + Merah). Dalam ekspresi, band ditulis "NamaLayer@nomorBand".
jalur_ndvi = processing.run("native:rastercalc", {
    "EXPRESSION": '("Citra_KPH@4" - "Citra_KPH@1") / ("Citra_KPH@4" + "Citra_KPH@1")',
    "LAYERS": [citra], "OUTPUT": "TEMPORARY_OUTPUT"})["OUTPUT"]
print("Keluaran raster sementara berupa:", type(jalur_ndvi).__name__)    # teks jalur, bukan layer
ndvi = QgsRasterLayer(jalur_ndvi, "NDVI_KPH", "gdal")                    # bungkus jadi layer
print("NDVI sah?", ndvi.isValid())

# 2. CHM = DSM - DTM
jalur_chm = processing.run("native:rastercalc", {
    "EXPRESSION": '"DSM@1" - "DTM@1"', "LAYERS": [dsm, dtm], "OUTPUT": "TEMPORARY_OUTPUT"})["OUTPUT"]
chm = QgsRasterLayer(jalur_chm, "CHM", "gdal")

# 3. Statistik zonal: rata-rata nilai raster di dalam tiap poligon KPH
def rata_rata(raster, awalan):
    return processing.run("native:zonalstatisticsfb", {
        "INPUT": batas, "INPUT_RASTER": raster, "RASTER_BAND": 1,
        "COLUMN_PREFIX": awalan, "STATISTICS": [2],      # 2 = Mean
        "OUTPUT": "TEMPORARY_OUTPUT"})["OUTPUT"]

zn = rata_rata(ndvi, "ndvi_")
zc = rata_rata(chm, "chm_")
chm_per_kph = {f["NAMA_KPH"]: f["chm_mean"] for f in zc.getFeatures()}
print("KPH         NDVI rata-rata   CHM rata-rata (m)")
for f in zn.getFeatures():
```

```
    print("%-11s %12.3f %16.2f" % (f["NAMA_KPH"], f["ndvi_mean"], chm_per_kph[f["NAMA_KPH"]]))

QgsProject.instance().addMapLayer(ndvi)
QgsProject.instance().addMapLayer(chm)
```

Penjelasan baris demi baris. Hanya yang baru yang dijelaskan.

Baris	Tulisan	Artinya
9-12	`batas`, `citra`, `dsm`, `dtm`	Buka satu layer vektor dan tiga raster. Raster memakai `QgsRasterLayer(..., "gdal")`. Nama kedua (`"Citra_KPH"`, `"DSM"`, `"DTM"`) penting, karena dipakai di ekspresi.
15-17	`jalur_ndvi = processing.run("native:rastercalc", {...})["OUTPUT"]`	Jalankan kalkulator raster. `EXPRESSION` berisi rumus NDVI. Band ditulis `"NamaLayer@nomorBand"`, misalnya `"Citra_KPH@4"` untuk band 4. `LAYERS` berisi daftar layer yang namanya dipakai di rumus.
18	`print(type(jalur_ndvi).__name__)`	Cetak jenis hasil. Jawabannya `str`: teks jalur, bukan layer.
19	`ndvi = QgsRasterLayer(jalur_ndvi, "NDVI_KPH", "gdal")`	Bungkus jalur itu menjadi layer raster.
20	`print("NDVI sah?", ndvi.isValid())`	Buktikan pembungkusnya berhasil.
23-25	`jalur_chm = ...` dan `chm = ...`	Hal yang sama untuk CHM, dengan rumus `"DSM@1" - "DTM@1"`.
28-32	`def rata_rata(raster, awalan):`	Resep bernama yang menjalankan alat statistik zonal. Dibahas di Bab 7. Ia menerima raster dan awalan nama kolom, lalu mengembalikan layer batas dengan kolom baru.
29-31	`"native:zonalstatisticsfb"`	Alat statistik zonal. `INPUT` poligon, `INPUT_RASTER` raster, `RASTER_BAND` nomor band, `COLUMN_PREFIX` awalan nama kolom hasil, dan `STATISTICS` daftar statistik. Angka 2 berarti rata-rata (Mean).
34-35	`zn = rata_rata(ndvi, "ndvi_")` dan `zc = rata_rata(chm, "chm_")`	Panggil resep dua kali. Kolom hasil bernama awalan ditambah `mean`: `ndvi_mean` dan `chm_mean`.
36	`chm_per_kph = {f["NAMA_KPH"]: f["chm_mean"] for f in zc.getFeatures()}`	Susun kamus: nama KPH sebagai kunci, rata-rata CHM sebagai nilai. Cara menulis ini disebut *dictionary comprehension*.
37-39	`for f in zn.getFeatures():`	Cetak tabel: nama KPH, `ndvi_mean`, dan CHM dari kamus tadi.
41-42	`addMapLayer(ndvi)` dan `addMapLayer(chm)`	Tampilkan dua raster di proyek.

Pada uji, bantuan alat `native:zonalstatisticsfb` mencatat kode statistik: 0 Count, 1 Sum, 2 Mean, 3 Median, 4 St dev, 5 Minimum, 6 Maximum, dan seterusnya.

Hasil. Layar menampilkan (berdasarkan uji pada QGIS 4.0.2):

```
HASIL
Keluaran raster sementara berupa: str
NDVI sah? True
KPH         NDVI rata-rata   CHM rata-rata (m)
KPH Alpha          0.604             2.60
KPH Beta           0.431             2.69
KPH Gamma          0.329             2.56
```

Dari hasil ini Anda tahu: vegetasi paling hijau ada di KPH Alpha (NDVI 0,604), sedangkan tajuk paling tinggi rata-rata di KPH Beta (2,69 meter).

Bagian yang boleh diubah.

- **Baris 16, rumus NDVI.** Nomor band harus sesuai urutan band citra Anda. Citra ini: band 1 Merah, band 4 NIR.
- Baris 31, `"STATISTICS": [2]`. Ganti kode statistik. Perhatikan: resep `rata_rata` dipakai untuk NDVI dan CHM sekaligus, jadi perubahannya berlaku untuk keduanya.
- **Baris 34-36, awalan kolom.** Bila Anda mengubah awalan, ubah juga nama kolom yang dibaca.
- **Baris 24, rumus CHM.** Ganti raster atau rumusnya.

Jebakan. Semua dicoba pada QGIS 4.0.2.

- **Band tertukar, tanpa galat.** Rumus NDVI ditulis terbalik: `"Citra_KPH@1"` di tempat `"Citra_KPH@4"` dan sebaliknya. Skrip jalan mulus, tetapi tabel menampilkan `-0.604`, `-0.431`, dan `-0.329`. Tanda minus adalah petunjuk bahwa band terbalik.
- **Nama layer di rumus tidak cocok.** `"DSM@1"` diganti `"DSM_Drone@1"`, padahal layer bernama `"DSM"`. Hasilnya `QgsProcessingException: Error occurred while performing calculation.`
- **Nama kolom hasil salah.** Menulis `f["chm_rata"]` menghasilkan `KeyError: 'chm_rata'`. Kolom hasil selalu awalan ditambah nama statistik (`mean`, `max`, dan seterusnya).
- **Hasil raster adalah teks.** Memanggil `jalur_ndvi.width()` gagal, karena `str` tidak punya metode itu. Bungkus dulu dengan `QgsRasterLayer`.

Latihan variasi. Ganti kolom CHM menjadi nilai tertinggi per KPH, bukan rata-rata.

Contoh jawaban (sudah diuji). Ubah tiga hal: pada baris 31 jadi `"STATISTICS": [2, 6]` (rata-rata dan tertinggi sekaligus), pada baris 36 jadi `f["chm_max"]`, dan pada baris 37 ganti judul menjadi `CHM tertinggi (m)`. Layar menampilkan:

```
HASIL
KPH         NDVI rata-rata   CHM tertinggi (m)
KPH Alpha          0.604            13.94
KPH Beta           0.431            13.96
KPH Gamma          0.329            13.96
```

> **Pola yang bisa dipakai ulang:** Kerangka alat raster: `jalur = processing.run("native:rastercalc", {"EXPRESSION": '"NAMA@1" + ...', "LAYERS": [LAYER], "OUTPUT": "TEMPORARY_OUTPUT"})["OUTPUT"]`, lalu bungkus `layer = QgsRasterLayer(jalur, "NAMA", "gdal")`. Ganti rumus, layer, dan nama.

Konsep: batch, satu resep untuk banyak berkas

Tiga skrip berikutnya membahas **batch**: pekerjaan yang sama diulang untuk banyak masukan. Bayangkan Anda memasak resep yang sama untuk tiga puluh porsi. Resepnya ditulis sekali, lalu perulangan `for` (Bab 6) memasaknya satu porsi demi satu porsi. Buku I4 Bab 3 menunjukkan cara bawaan QGIS (*Execute as batch process*). Cara itu cocok untuk beberapa berkas. Untuk puluhan berkas, skrip lebih ringkas.

Ilustrasi 21.6: Satu resep, banyak berkas

Skema; bukan tangkapan layar



Istilah baru bagian ini:

- **Batch**: pekerjaan sama, banyak masukan.
- **Pola berkas** (`glob`): cara mencari banyak berkas sekaligus, misalnya semua yang berakhiran `.tif`.
- **Latar belakang**: jalur kerja terpisah, supaya QGIS tidak membeku.

Skrip 21.4: Batch sederhana, NDVI untuk semua ubin

Tujuan. Menghitung NDVI untuk semua ubin ortofoto di satu folder dengan satu perulangan `for`, menyimpan tiap hasil sebagai berkas, dan mencetak rata-ratanya.

Bahan. Paket data `paket-i4`, folder `Ortofoto_Mentah` (tiga ubin; pada data ini band 1 adalah NIR dan band 2 adalah Merah). Berkas `b3_01_batch_ndvi.py`. Ganti `DATA` di baris 8 dan `HASIL` di baris 9. Skrip membuat folder `NDVI_ubin` di dalam `HASIL`.

Ilustrasi 21.7: Alur skrip 21.4

Skema; bukan tangkapan layar



Skrip utuh:

```python
# [SKRIP 3.1: Batch sederhana: NDVI untuk semua ubin ortofoto dalam satu folder]
# Penulis: Badar Mubarok Yogaswara
import glob
import os
import processing
from qgis.core import QgsRasterLayer

DATA = "D:/Latihan/paket-i4"
HASIL = "D:/Latihan/hasil_i4"
folder_masuk = DATA + "/Ortofoto_Mentah"
folder_keluar = HASIL + "/NDVI_ubin"
os.makedirs(folder_keluar, exist_ok=True)       # buat folder bila belum ada

# 1. Kumpulkan semua berkas .tif, diurutkan agar hasilnya selalu berurutan sama
daftar = sorted(glob.glob(folder_masuk + "/*.tif"))
print("Ditemukan", len(daftar), "ubin")

# 2. Ulangi pekerjaan yang sama untuk tiap berkas. Pada ubin ini band 1 = NIR dan band 2 = Merah.
berhasil = []
for jalur in daftar:
    nama = os.path.basename(jalur)                          # contoh: ubin_01.tif
    keluar = folder_keluar + "/NDVI_" + nama
    ubin = QgsRasterLayer(jalur, "ubin", "gdal")            # nama "ubin" dipakai di ekspresi
    processing.run("native:rastercalc", {
        "EXPRESSION": '("ubin@1" - "ubin@2") / ("ubin@1" + "ubin@2")',
        "LAYERS": [ubin], "OUTPUT": keluar})
    hasil = QgsRasterLayer(keluar, "hasil", "gdal")
    rata = processing.run("native:rasterlayerstatistics", {"INPUT": hasil, "BAND": 1})["MEAN"]
    print("  %s -> %s  | NDVI rata-rata %.3f" % (nama, os.path.basename(keluar), rata))
    berhasil.append(keluar)
```

```
print("Selesai:", len(berhasil), "dari", len(daftar), "berkas")
```

Penjelasan baris demi baris.

Baris	Tulisan	Artinya
3-6	`import glob`, `import os`, `import processing`, `from qgis.core ...`	`glob` untuk mencari berkas, `os` untuk urusan folder dan nama berkas.
10-12	`folder_masuk`, `folder_keluar`, `os.makedirs(...)`	Tentukan folder masukan dan keluaran, lalu buat folder keluaran bila belum ada.
15	`daftar = sorted(glob.glob(folder_masuk + "/*.tif"))`	`glob.glob` mengembalikan semua jalur berkas berakhiran `.tif`. `sorted` mengurutkannya agar hasil selalu berurutan sama.
16	`print("Ditemukan", len(daftar), "ubin")`	Cetak jumlah berkas yang ditemukan.
19	`berhasil = []`	Daftar kosong untuk mencatat hasil.
20	`for jalur in daftar:`	Mulai perulangan. Di tiap putaran, `jalur` berisi satu berkas.
21-22	`nama = os.path.basename(jalur)` lalu `keluar = ...`	Ambil nama berkas saja (misalnya `ubin_01.tif`), lalu susun jalur keluaran `NDVI_ubin_01.tif`.
23	`ubin = QgsRasterLayer(jalur, "ubin", "gdal")`	Buka ubin sebagai layer bernama `ubin`. Nama ini dipakai di rumus.
24-26	`processing.run("native:rastercalc", {...})`	Hitung NDVI. Band 1 adalah NIR dan band 2 adalah Merah, jadi rumusnya `("ubin@1" - "ubin@2") / ("ubin@1" + "ubin@2")`. `OUTPUT` berisi jalur berkas, jadi hasilnya ditulis ke sana.
27	`hasil = QgsRasterLayer(keluar, "hasil", "gdal")`	Buka berkas hasil sebagai layer.
28	`rata = processing.run("native:rasterlayerstatistics", {...})["MEAN"]`	Jalankan alat statistik raster. Hasilnya kamus dengan beberapa kunci. Skrip mengambil kunci `MEAN`.
29-30	`print(...)` dan `berhasil.append(keluar)`	Cetak satu baris ringkas, lalu catat berkas ke daftar.
32	`print("Selesai:", len(berhasil), "dari", len(daftar), "berkas")`	Cetak ringkasan akhir.

Hasil. Layar menampilkan (berdasarkan uji pada QGIS 4.0.2):

```
HASIL
Ditemukan 3 ubin
  ubin_01.tif -> NDVI_ubin_01.tif  | NDVI rata-rata 0.471
  ubin_02.tif -> NDVI_ubin_02.tif  | NDVI rata-rata 0.471
```

```
  ubin_03.tif -> NDVI_ubin_03.tif  | NDVI rata-rata 0.467
Selesai: 3 dari 3 berkas
```

Di folder `NDVI_ubin` kini ada tiga berkas NDVI. QGIS dapat menambahkan berkas pendamping berakhiran `.aux.xml`. Berkas itu aman dibiarkan.

Bagian yang boleh diubah.

- **Baris 10, folder masukan.** Ganti `/Ortofoto_Mentah` dengan folder lain.
- Baris 15, pola `"/*.tif"`. Ganti ke `"/ubin_0*.tif"` untuk memilih sebagian berkas.
- **Baris 25, rumus.** Urutan band harus sesuai data Anda. Perhatikan: di `Citra_KPH.tif` (Skrip 21.3), band 4 adalah NIR, sedangkan di ubin ini band 1.
- Baris 22, awalan `"NDVI_"`. Ganti sesuai selera.

Jebakan. Semua dicoba pada QGIS 4.0.2.

- **Folder salah, tanpa galat.** `Ortofoto_Mentah` diganti `Ortofoto_Mentahh`. Layar menampilkan `Ditemukan 0 ubin` dan `Selesai: 0 dari 0 berkas`. Perulangan tidak berputar sama sekali. Baca baris pertama hasil.
- **Satu berkas rusak menghentikan semuanya.** Folder `Ortofoto_Campur` berisi empat ubin, dan ubin keempat rusak. Layar menampilkan `Ditemukan 4 ubin`, tiga baris hasil, lalu `ERROR 1: Attempt to create -2147483648x-2147483648 dataset is illegal,sizes must be larger than zero.` dan berakhir dengan `QgsProcessingException: Error creating output file.` Tiga ubin pertama sudah selesai, tetapi ringkasan akhir tidak pernah tercetak. Penanganannya (`try` dan `except`) dibahas di Bab 10.
- **Band terbalik, tanpa galat.** Rumus ditulis `("ubin@2" - "ubin@1") / ...`. Hasilnya `-0.471`, `-0.471`, dan `-0.467`. Tanda minus menandakan band terbalik.
- **Band yang tidak ada.** Menulis `"ubin@4"` pada ubin dua band menghasilkan `QgsProcessingException: Error occurred while performing calculation.`
- `berhasil` selalu sama panjang. Daftar `berhasil` selalu terisi setiap putaran, sehingga ringkasan "3 dari 3" belum berarti ada penyaringan. Ia baru berguna bila disertai penanganan galat.

Latihan variasi. Tambahkan satu baris di dalam perulangan yang mencetak ukuran piksel tiap hasil.

Contoh jawaban (sudah diuji). Sisipkan sebelum `berhasil.append(keluar)`:

```
PYTHON
    print("     ukuran:", hasil.width(), "x", hasil.height(), "piksel")
```

Di bawah tiap baris NDVI, layar menampilkan `ukuran: 100 x 100 piksel`.

Pola yang bisa dipakai ulang: Kerangka batch berkas: `daftar = sorted(glob.glob(FOLDER + "/*.EKSTENSI"))`, lalu `for jalur in daftar:`, lalu satu atau dua panggilan `processing.run(...)` dengan `OUTPUT` berupa jalur berkas yang disusun dari `os.path.basename(jalur)`. Ganti folder, ekstensi, alat, dan rumus.

Skrip 21.5: Satu putaran untuk tiap fitur

Perulangan tidak harus atas berkas. Skrip ini berputar atas **fitur**: untuk tiap KPH, ia memilih satu poligon lalu memotong raster tinggi tajuk (CHM) dengannya. Hasilnya satu raster per KPH.

Tujuan. Menghasilkan raster CHM terpisah untuk tiap KPH, lalu mencetak ukuran, rata-rata, dan nilai tertingginya.

Bahan. Paket data `paket-i4`: `Batas_KPH.gpkg`, `DSM_Drone.tif`, dan `DTM_Drone.tif`. Berkas `b3_02_batch_per_kph.py`. Ganti `DATA` di baris 7 dan `HASIL` di baris 8. Skrip membuat folder `CHM_per_KPH`.

Ilustrasi 21.8: Alur skrip 21.5



Skrip utuh:

```python
# [SKRIP 3.2: Batch per fitur: potong raster tinggi tajuk (CHM) untuk tiap KPH]
# Penulis: Badar Mubarok Yogaswara
import os
import processing
from qgis.core import QgsVectorLayer, QgsRasterLayer

DATA = "D:/Latihan/paket-i4"
HASIL = "D:/Latihan/hasil_i4"
folder_keluar = HASIL + "/CHM_per_KPH"
os.makedirs(folder_keluar, exist_ok=True)

batas = QgsVectorLayer(DATA + "/Batas_KPH.gpkg", "Batas_KPH", "ogr")
dsm = QgsRasterLayer(DATA + "/DSM_Drone.tif", "DSM", "gdal")
dtm = QgsRasterLayer(DATA + "/DTM_Drone.tif", "DTM", "gdal")

# 1. CHM untuk seluruh area (satu kali saja, di luar pengulangan)
chm_jalur = processing.run("native:rastercalc", {
    "EXPRESSION": '"DSM@1" - "DTM@1"', "LAYERS": [dsm, dtm], "OUTPUT": "TEMPORARY_OUTPUT"})["OUTPUT"]

# 2. Satu putaran untuk tiap KPH: pilih satu poligon, lalu potong CHM dengannya
for fitur in batas.getFeatures():
```

```
    nama = fitur["NAMA_KPH"]                                # contoh: KPH Alpha
    aman = nama.replace(" ", "_")                           # untuk nama berkas: KPH_Alpha
    satu_kph = processing.run("native:extractbyexpression", {
        "INPUT": batas, "EXPRESSION": '"NAMA_KPH" = \'%s\'' % nama,
        "OUTPUT": "TEMPORARY_OUTPUT"})["OUTPUT"]
    keluar = folder_keluar + "/CHM_" + aman + ".tif"
    processing.run("gdal:cliprasterbymasklayer", {
        "INPUT": chm_jalur, "MASK": satu_kph, "CROP_TO_CUTLINE": True,
        "NODATA": -9999, "OUTPUT": keluar})
    hasil = QgsRasterLayer(keluar, aman, "gdal")
    st = processing.run("native:rasterlayerstatistics", {"INPUT": hasil, "BAND": 1})
    print("%-10s %4d x %4d piksel | rata-rata %.2f m | tertinggi %.2f m" % (
        aman, hasil.width(), hasil.height(), st["MEAN"], st["MAX"]))
print("Berkas di folder:", sorted(os.listdir(folder_keluar)))
```

Penjelasan baris demi baris. Hanya yang baru yang dijelaskan.

Baris	Tulisan	Artinya
17-18	`chm_jalur = processing.run("native:rastercalc", {...})["OUTPUT"]`	Hitung CHM untuk seluruh area **satu kali saja**, di luar perulangan. Menghitung ulang di tiap putaran hanya membuang waktu.
21	`for fitur in batas.getFeatures():`	Satu putaran untuk tiap KPH.
22-23	`nama = fitur["NAMA_KPH"]` lalu `aman = nama.replace(" ", "_")`	Ambil nama KPH, lalu ganti spasi dengan garis bawah untuk nama berkas: `KPH_Alpha`.
24-26	`satu_kph = processing.run("native:extractbyexpression", {...})["OUTPUT"]`	Ambil satu poligon saja dengan ekspresi. `'"NAMA_KPH" = \'%s\'' % nama` menyisipkan nama ke dalam teks ekspresi, dengan kutip tunggal di sekelilingnya.
27	`keluar = folder_keluar + "/CHM_" + aman + ".tif"`	Susun nama berkas keluaran.
28-30	`processing.run("gdal:cliprasterbymasklayer", {...})`	Potong CHM dengan poligon tadi. `INPUT` raster yang dipotong, `MASK` poligon pemotong, `CROP_TO_CUTLINE` berarti ukuran hasil mengikuti poligon, dan `NODATA` nilai untuk piksel di luar poligon.
31-32	`hasil = QgsRasterLayer(...)` lalu `st = ...rasterlayerstatistics...`	Buka hasil, lalu hitung statistiknya. Kali ini seluruh kamus disimpan di `st`, bukan hanya satu kuncinya.
33-34	`print("%-10s %4d x %4d piksel ..." % (...))`	Cetak satu baris: nama, lebar, tinggi, rata-rata (`st["MEAN"]`), dan nilai tertinggi (`st["MAX"]`). Tanda garis tegak di dalam tulisan yang dicetak hanyalah pemisah tampilan.
35	`print("Berkas di folder:", sorted(os.listdir(folder_keluar)))`	Cetak isi folder keluaran, diurutkan.

Hasil. Layar menampilkan (berdasarkan uji pada QGIS 4.0.2):

```
HASIL
KPH_Alpha   400 x  400 piksel | rata-rata 2.60 m | tertinggi 13.94 m
KPH_Beta    400 x  400 piksel | rata-rata 2.69 m | tertinggi 13.96 m
KPH_Gamma   800 x  400 piksel | rata-rata 2.56 m | tertinggi 13.96 m
Berkas di folder: ['CHM_KPH_Alpha.tif', 'CHM_KPH_Alpha.tif.aux.xml', 'CHM_KPH_Beta.tif',
'CHM_KPH_Beta.tif.aux.xml', 'CHM_KPH_Gamma.tif']
```

Perhatikan daftar berkas di baris terakhir. Berkas `.aux.xml` dibuat oleh QGIS saat menghitung statistik, bukan oleh skrip secara sengaja. Isi daftar di komputer Anda bisa berbeda, misalnya `CHM_KPH_Gamma.tif.aux.xml` ikut muncul bila skrip dijalankan untuk kedua kalinya. KPH Gamma berukuran 800 kali 400 piksel karena areanya dua kali lebih luas.

Bagian yang boleh diubah.

- **Baris 9-10, folder keluaran.** Ganti nama folder.
- Baris 25, kolom `NAMA_KPH`. Ganti bila data Anda memakai kolom lain.
- Baris 29, `CROP_TO_CUTLINE`. `True` memotong ukuran hasil mengikuti KPH.
- Baris 30, `NODATA`. Nilai -9999 menandai piksel kosong. Nilai ini ikut dikenali saat statistik dihitung.
- **Baris 17-18, rumus CHM.** Ganti rumus atau raster, misalnya untuk NDVI.

Jebakan. Semua dicoba pada QGIS 4.0.2.

- **Nama KPH yang memuat kutip tunggal.** Pada uji, nama `KPH D'Alpha` disisipkan ke baris 25. Alat berhenti dengan `Incomplete expression. You might not have finished the full expression.` Kutip tunggal di tengah nama mengakhiri teks ekspresi lebih awal. Data latihan tidak memuat nama seperti itu.
- **Nama kolom salah.** `fitur["NAMA_KPH"]` diganti `fitur["NAMA"]`: `KeyError: 'NAMA'`.
- `CROP_TO_CUTLINE` bernilai `False`. Tidak ada galat. Ketiga hasil berukuran `800 x 800 piksel`, karena ukuran tidak dipotong mengikuti KPH. Rata-rata Beta dan Gamma di layar tetap `2.69` dan `2.56`.
- **Berkas pendamping di daftar.** Karena baris 35 mencetak semua isi folder, berkas `.aux.xml` ikut tampil.

Latihan variasi. Jalankan hanya untuk KPH Gamma. Bagian mana yang cukup ditambah?

Contoh jawaban (sudah diuji). Sisipkan dua baris setelah baris 22:

```
PYTHON
    if nama != "KPH Gamma":
        continue
```

`continue` berarti "lewati sisa putaran ini, lanjut ke fitur berikutnya". Layar menampilkan satu baris `KPH_Gamma   800 x  400 piksel | rata-rata 2.56 m | tertinggi 13.96 m`, disusul daftar isi folder (yang juga memuat berkas dari uji sebelumnya).

> **Pola yang bisa dipakai ulang:** Kerangka batch per fitur: hitung bahan bersama **sekali** di luar perulangan, lalu `for fitur in LAYER.getFeatures():`, pilih satu fitur dengan `native:extractbyexpression`, lalu proses dan simpan dengan nama yang memuat nama fitur. Ganti layer, kolom nama, dan alat di tengah.

Konsep: latar belakang dan fungsi selesai

Skrip 21.4 dan 21.5 membuat QGIS terasa macet selama bekerja. Bayangkan Anda pelayan restoran yang juga harus memasak. Selama Anda memasak, tamu tidak terlayani. Solusinya: serahkan masakan ke juru masak di dapur belakang, sementara Anda tetap melayani. QGIS menyediakan juru masak ini sebagai **QgsTask**, yaitu tugas latar belakang.

Ilustrasi 21.9: Program utama dan latar belakang



Pembagian kerjanya begini. **Fungsi tugas** berjalan di dapur belakang dan hanya boleh memasak (menghitung). **Fungsi selesai** berjalan di depan setelah masakan jadi, dan aman untuk menyentuh proyek, misalnya menambah layer. Fungsi selesai disebut juga *callback*, yaitu fungsi yang dipanggil kembali setelah tugas berakhir.

Tiga istilah baru:

- **QgsTask**: tugas latar belakang milik QGIS.
- **Fungsi tugas**: fungsi yang dikerjakan di latar belakang.
- **Fungsi selesai**: fungsi yang dipanggil di program utama setelah tugas berakhir.

Skrip 21.6: Batch di latar belakang dengan QgsTask

Tujuan. Menjalankan batch NDVI di latar belakang, lengkap dengan bilah kemajuan dan kemungkinan pembatalan, lalu menambahkan hasilnya ke proyek.

Bahan. Paket data `paket-i4`, folder `Ortofoto_Mentah`. Berkas `b3_03_batch_latar_belakang.py`. Ganti `DATA` di baris 8 dan `HASIL` di baris 9. Skrip membuat folder `NDVI_latar`. Skrip ini paling pas dijalankan di Editor Skrip QGIS yang terbuka, karena tugas latar belakang butuh QGIS yang hidup.

Ilustrasi 21.10: Alur skrip 21.6



Skrip utuh:

```python
# [SKRIP 3.3: Batch di latar belakang dengan QgsTask supaya QGIS tidak macet]
# Penulis: Badar Mubarok Yogaswara
import glob
import os
import processing
from qgis.core import QgsApplication, QgsTask, QgsMessageLog, Qgis, QgsRasterLayer, QgsProject

DATA = "D:/Latihan/paket-i4"
HASIL = "D:/Latihan/hasil_i4"
folder_masuk = DATA + "/Ortofoto_Mentah"
folder_keluar = HASIL + "/NDVI_latar"
os.makedirs(folder_keluar, exist_ok=True)


def hitung_ndvi(task, folder_in, folder_out):
    """Berjalan di LATAR BELAKANG. Jangan menyentuh proyek atau antarmuka di sini."""
    daftar = sorted(glob.glob(folder_in + "/*.tif"))
    keluar = []
    for i, jalur in enumerate(daftar):
        if task.isCanceled():                       # pengguna menekan Cancel
            return None
        tujuan = folder_out + "/NDVI_" + os.path.basename(jalur)
        ubin = QgsRasterLayer(jalur, "ubin", "gdal")
        processing.run("native:rastercalc", {
            "EXPRESSION": '("ubin@1" - "ubin@2") / ("ubin@1" + "ubin@2")',
            "LAYERS": [ubin], "OUTPUT": tujuan})
        keluar.append(tujuan)
        task.setProgress(100 * (i + 1) / len(daftar))    # bilah kemajuan di pojok kanan bawah QGIS
    return keluar
```

```
def selesai(exception, hasil=None):
    """Berjalan di PROGRAM UTAMA setelah tugas berakhir. Aman untuk menyentuh proyek."""
    if exception is not None:
        QgsMessageLog.logMessage("Tugas gagal: %s" % exception, "Batch NDVI", Qgis.Critical)
    elif hasil is None:
        QgsMessageLog.logMessage("Tugas dibatalkan", "Batch NDVI", Qgis.Warning)
    else:
        for jalur in hasil:
            nama = os.path.basename(jalur)[:-4]
            QgsProject.instance().addMapLayer(QgsRasterLayer(jalur, nama, "gdal"))
        QgsMessageLog.logMessage("Selesai: %d berkas" % len(hasil), "Batch NDVI", Qgis.Success)
        print("Tugas selesai:", len(hasil), "berkas NDVI ditambahkan ke proyek")


tugas = QgsTask.fromFunction("Hitung NDVI semua ubin", hitung_ndvi, on_finished=selesai,
                             folder_in=folder_masuk, folder_out=folder_keluar)
QgsApplication.taskManager().addTask(tugas)
print("Tugas dikirim ke latar belakang. QGIS tetap bisa dipakai.")
```

Penjelasan baris demi baris. Skrip ini terdiri atas dua fungsi dan tiga baris pengiriman. Rumus NDVI di baris 25 sama dengan Skrip 21.4.

Blok A: fungsi tugas, yang bekerja di latar belakang (baris 15-29).

Baris	Tulisan	Artinya
15	`def hitung_ndvi(task, folder_in, folder_out):`	Fungsi tugas. Parameter pertama `task` diisi otomatis oleh QGIS. Dua lainnya diisi dari baris 47.
16	`"""Berjalan di LATAR BELAKANG..."""`	Catatan di awal fungsi (disebut *docstring*, Bab 7). Isinya peringatan: jangan menyentuh proyek atau antarmuka di sini.
17-18	`daftar = ...` dan `keluar = []`	Kumpulkan berkas, siapkan daftar hasil.
19	`for i, jalur in enumerate(daftar):`	Ulangi tiap ubin. `enumerate` menambahkan nomor urut `i` mulai dari 0 (Bab 6).
20-21	`if task.isCanceled(): return None`	Bila pengguna menekan Cancel, berhenti dan kembalikan `None`.
22-27	`tujuan = ...` sampai `keluar.append(tujuan)`	Hitung NDVI satu ubin ke berkas, lalu catat jalurnya.
28	`task.setProgress(100 * (i + 1) / len(daftar))`	Isi bilah kemajuan, 0 sampai 100 persen.
29	`return keluar`	Kembalikan daftar berkas hasil. Nilai ini akan diterima fungsi selesai.

Blok B: fungsi selesai, yang bekerja di depan (baris 32-43).

Baris	Tulisan	Artinya
32	`def selesai(exception, hasil=None):`	Fungsi selesai. `exception` berisi galat bila tugas gagal. `hasil` berisi daftar dari `return keluar`. `=None` adalah nilai bawaan (Bab 7).
34-35	`if exception is not None:`	Bila ada galat, catat ke log sebagai `Qgis.Critical`.
36-37	`elif hasil is None:`	Bila tidak ada hasil, tugas dianggap dibatalkan. Catat `Qgis.Warning`.
38-43	`else:`	Bila sukses: untuk tiap jalur, buat layer dan tambahkan ke proyek (baris 39-41), catat `Qgis.Success` (baris 42), lalu cetak pesan (baris 43). `[:-4]` membuang empat huruf terakhir (`.tif`) dari nama berkas.

Blok C: membuat dan mengirim tugas (baris 46-49).

Baris	Tulisan	Artinya
46-47	`tugas = QgsTask.fromFunction("Hitung NDVI semua ubin", hitung_ndvi, on_finished=selesai, folder_in=..., folder_out=...)`	Bungkus fungsi tugas menjadi tugas. Isian: judul, fungsi tugas, fungsi selesai (`on_finished`), lalu argumen bernama yang diteruskan ke fungsi tugas. Fungsi dipakai sebagai nilai tanpa kurung.
48	`QgsApplication.taskManager().addTask(tugas)`	Serahkan tugas ke manajer tugas QGIS. Tugas mulai berjalan.
49	`print("Tugas dikirim ...")`	Cetak pesan. Pesan ini muncul **sebelum** tugas selesai, karena skrip tidak menunggu.

Hasil. Layar menampilkan (berdasarkan uji pada QGIS 4.0.2):

```
HASIL
Tugas dikirim ke latar belakang. QGIS tetap bisa dipakai.
Tugas selesai: 3 berkas NDVI ditambahkan ke proyek
```

Pesan pertama muncul langsung. Pesan kedua muncul beberapa saat kemudian, setelah tugas selesai. Di dalam QGIS, kemajuan tugas tampil di bilah status dan pesan lognya di panel **Log Messages** dengan tag `Batch NDVI` [CEK: tampilan visualnya, tidak dapat diuji tanpa antarmuka QGIS]. Tiga layer baru `NDVI_ubin_01` sampai `NDVI_ubin_03` masuk ke proyek.

Bagian yang boleh diubah.

- **Baris 8-9, jalur.** Sesuaikan dengan komputer Anda.
- **Baris 17, daftar berkas.** Ubah pola berkas atau batasi jumlahnya.
- **Baris 25, rumus.** Sesuaikan urutan band.
- Baris 35, 37, dan 42, tag log `"Batch NDVI"`. Ganti dengan nama yang Anda suka.

- **Baris 46, judul tugas.** Judul ini tampil di daftar tugas QGIS.

Jebakan.

- **Skrip tidak menunggu tugas.** Pesan "Tugas dikirim" tampil sebelum NDVI selesai. Itu normal di dalam QGIS.
- **Skrip mandiri harus menunggu sendiri.** Pada uji, skrip ini dijalankan dengan kerangka mandiri, lalu dibiarkan habis begitu saja. Layar hanya menampilkan `Tugas dikirim ...`, kemudian proses berhenti mendadak (kode keluar 139, tanda proses mogok). Pesan "Tugas selesai" tidak pernah muncul. Perbaikannya: sebelum `app.exitQgis()`, tunggu tugas dengan perulangan berikut. Dengan perulangan ini, uji berjalan normal dan menampilkan kedua pesan.

```python
import time

while QgsApplication.taskManager().countActiveTasks() > 0:
    app.processEvents()
    time.sleep(0.05)
for _ in range(20):
    app.processEvents()
    time.sleep(0.02)
```

- **Fungsi tugas tidak boleh menyentuh proyek atau antarmuka.** Komentar di baris 16 dan 33 dan Buku I4 mengingatkan bahwa menambah layer dari fungsi tugas dapat membuat QGIS macet atau tertutup sendiri. Itu sebabnya `addMapLayer` ada di fungsi selesai. Bagian ini tidak diuji [CEK].
- **Berkas pendamping.** Seperti Skrip 21.4, QGIS dapat membuat berkas `.aux.xml` di folder `NDVI_latar`.

Latihan variasi. Batasi tugas hanya untuk dua ubin pertama.

Contoh jawaban (sudah diuji). Ubah baris 17 menjadi:

```python
    daftar = sorted(glob.glob(folder_in + "/*.tif"))[:2]
```

`[:2]` mengambil dua isi pertama dari daftar. Layar menampilkan `Tugas selesai: 2 berkas NDVI ditambahkan ke proyek`, dan folder `NDVI_latar` berisi `NDVI_ubin_01.tif` dan `NDVI_ubin_02.tif`.

> **Pola yang bisa dipakai ulang:** Kerangka tugas latar belakang: tulis `def kerja(task, ...)` yang hanya menghitung dan mengakhiri dengan `return HASIL`, tulis `def selesai(exception, hasil=None)` yang menyentuh proyek, lalu `QgsApplication.taskManager().addTask(QgsTask.fromFunction("JUDUL", kerja, on_finished=selesai, ...))`. Ganti isi fungsi `kerja`.

Cek paham

1. Apa isi `hasil["OUTPUT"]` bila `OUTPUT` berupa `"TEMPORARY_OUTPUT"` pada alat vektor? Bagaimana bila alatnya raster?

2. Mengapa CHM dihitung di luar perulangan pada Skrip 21.5?
3. Fungsi mana yang boleh menambahkan layer ke proyek pada Skrip 21.6, dan mengapa?

Jawaban.

1. Pada alat vektor, isinya layer (`QgsVectorLayer`) yang siap dipakai. Pada alat raster, isinya teks jalur berkas, yang harus dibungkus dengan `QgsRasterLayer`.
2. CHM seluruh area sama untuk semua KPH. Menghitungnya sekali cukup, dan menghitungnya di tiap putaran hanya membuang waktu.
3. Fungsi selesai. Ia berjalan di program utama, sedangkan fungsi tugas berjalan di latar belakang dan tidak boleh menyentuh proyek.

Kesalahan umum

- **Salah ketik nama isian atau kolom.** Nama isian yang tak dikenal diabaikan diam-diam (Skrip 21.1), sedangkan nama kolom yang salah memicu `KeyError`. Perbaikan: salin nama dari `parameterDefinitions()` dan dari daftar kolom layer.
- **Mengira layer yang gagal dibuka akan menimbulkan galat.** Alat tetap jalan dengan layer kosong dan menghasilkan angka nol. Perbaikan: periksa `isValid()` dan baca baris pertama hasil.
- **Urutan band terbalik.** NDVI menjadi negatif tanpa pesan apa pun. Perbaikan: cek urutan band tiap dataset dan curigai tanda minus.
- **Memakai satuan jarak yang salah.** `DISTANCE` mengikuti CRS, jadi 30 pada CRS derajat berarti 30 derajat. Perbaikan: pastikan CRS metrik.
- **Skrip mandiri yang tidak menunggu tugas latar belakang.** Prosesnya berhenti mendadak sebelum tugas selesai. Perbaikan: tambahkan perulangan tunggu.

Ringkasan dan latihan

- `processing.run("ID_ALAT", {kamus parameter})` memanggil alat Processing. Hasilnya kamus, dan jawabannya diambil dengan kunci seperti `["OUTPUT"]`.
- Vektor sementara berupa layer. Raster sementara dan keluaran berupa berkas berupa teks jalur.
- Merangkai alat berarti memakai hasil satu alat sebagai masukan alat berikutnya.
- Batch berarti satu resep, banyak masukan: perulangan atas berkas (`glob`) atau atas fitur (`getFeatures`).
- `QgsTask` menjalankan pekerjaan berat di latar belakang. Fungsi tugas menghitung, fungsi selesai menyentuh proyek.

Latihan. Gabungkan Skrip 21.2 dan 21.5. Untuk tiap KPH, buat sempadan sungai 30 meter, lalu potong CHM dengan sempadan itu, dan cetak rata-rata tinggi tajuk di sempadan. Alat apa yang Anda butuhkan di setiap langkah, dan bagian mana yang dipinjam dari tiap skrip?

Tabel perbandingan: memanggil alat dengan klik dan dengan skrip

Pekerjaan	Dengan klik di QGIS	Dengan skrip
Memilih alat	Cari di Processing Toolbox	ID alat, misalnya `native:buffer`
Mengisi kolom	Isi dialog	Kamus parameter
Menjalankan	Tombol Run	`processing.run(ID, kamus)`
Mengambil hasil	Layer baru di panel Layers	`hasil["OUTPUT"]`, lalu `addMapLayer`
Banyak berkas	Execute as batch process	Perulangan `for` dengan `glob`
Pekerjaan berat	Alat berjalan, QGIS terasa macet	`QgsTask` di latar belakang

BAB 22: Skrip yang Tidak Mudah Menyerah: Galat, Batch Tahan Galat, dan Ekspresi QGIS

> **Berdasarkan Seri I4:** Skrip bab ini berasal dari Buku I4, *Otomasi Dasar dengan Python* (Intermediate Class), Bab 4 dan Bab 5, berkas `b4_01_galat_dasar.py`, `b4_02_batch_tahan_galat.py`, `b4_03_galat_alat.py`, `b5_01_ekspresi_python.py`, dan `b5_02_ekspresi_di_alat.py`. Di dalam berkasnya tertulis nomor Skrip 4.1 sampai 5.2, mengikuti penomoran buku itu. Di bab ini kelimanya diberi nomor Skrip 22.1 sampai 22.5. Di buku itu, galat dan ekspresi dibahas lewat menu dan klik. Bab ini menunjukkan sisi skripnya.

Studi kasus: "Enam kiriman, satu berkas rusak, dan laporan yang menulis dirinya sendiri"

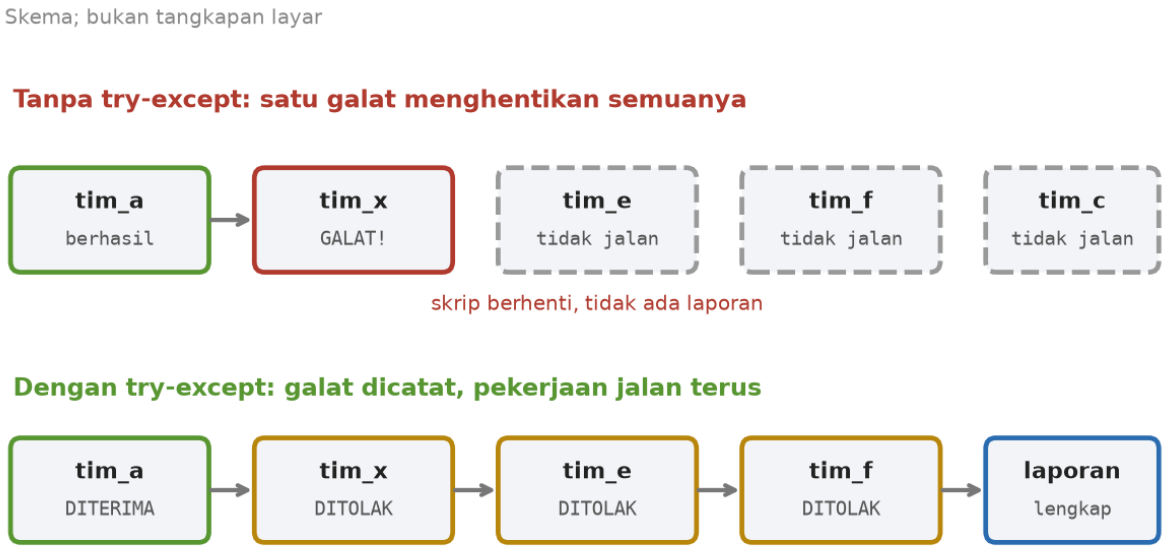
Enam tim lapangan mengirim titik survei. Malam hari, skrip penggabung Anda berhenti di tengah jalan dengan pesan Inggris yang panjang. Anda tidak tahu berkas mana yang bermasalah. Esoknya Kepala Seksi hanya bertanya: berapa kiriman yang diterima, mana yang ditolak, dan mengapa? Ia juga minta satu kalimat ringkas per KPH, misalnya "KPH Beta, 4 ha, 5 titik panas", yang angkanya berubah tiap bulan. Bab ini mengajarkan dua hal. Pertama, membuat skrip yang menjawab pertanyaan itu sendiri. Kedua, menulis rumus laporan satu kali saja.

Konsep: galat dalam tiga kalimat

Galat (*exception*) adalah tanda dari Python bahwa ada yang gagal. Tanpa penanganan, skrip langsung berhenti. Dengan `try` dan `except`, Anda menangkap galat itu, mencatat sebabnya, lalu meneruskan pekerjaan.

Bayangkan petugas gudang yang memeriksa tiap kiriman. Barang rusak dikembalikan dengan catatan. Gudang tidak ditutup hanya karena satu kardus basah. Bab 10 sudah mengajarkan dasarnya. Bab ini memakainya pada data GIS.

Ilustrasi 22.1: Tanpa dan dengan penanganan galat



Lima istilah yang perlu Anda ingat:

- **Galat** (*exception*): peristiwa yang menghentikan skrip bila tidak ditangani.
- `try` dan `except`: coba kerjakan, dan bila galat, tangkap lalu tangani.
- `raise`: perintah untuk menimbulkan galat sendiri dengan pesan yang jelas.
- **Validasi**: memeriksa masukan sebelum dipakai.
- **Log**: catatan hasil tiap langkah, yang bisa dibaca belakangan.

Skrip 22.1: Memeriksa masukan sebelum dipakai

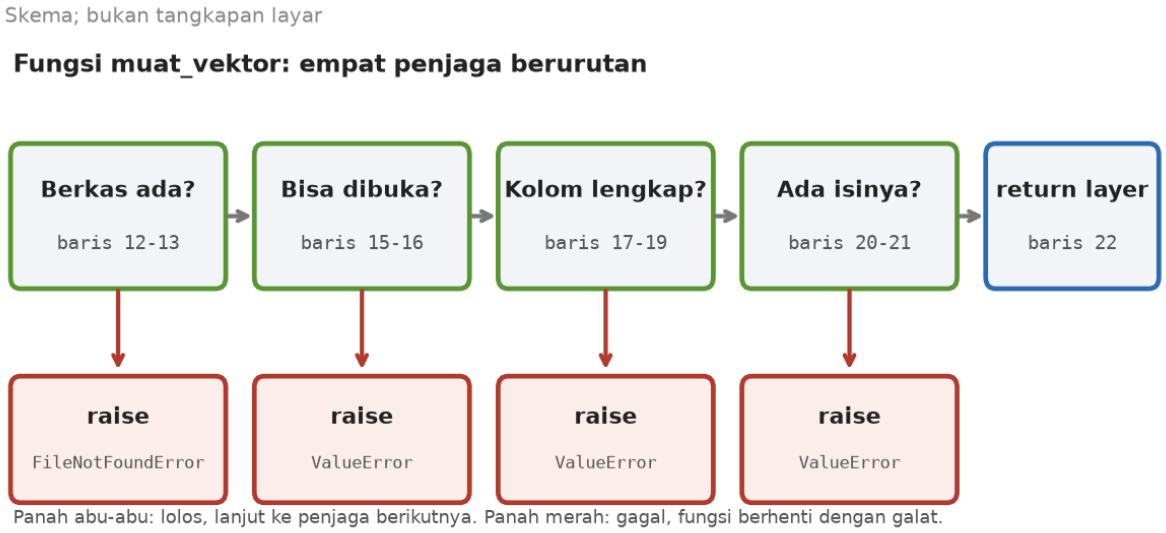
Berkas `b4_01_galat_dasar.py` sama dengan yang dipakai di Buku I4. Di dalam berkasnya tertulis "Skrip 4.1". Skrip ini sudah dibedah baris demi baris di Bab 10. Di sini ia dipakai sebagai **cetakan**: fungsi penjaga di baris 10-22 muncul lagi dalam bentuk lain pada Skrip 22.2 dan 22.3. Karena itu penjelasannya dipadatkan.

Tujuan. Memeriksa empat berkas kiriman satu per satu. Berkas yang bermasalah dilewati dengan pesan jelas, dan berkas yang baik tetap dibaca.

Bahan. Paket data `paket-i4`, folder `Kiriman_Lapangan`. Isinya `tim_a.gpkg` (baik, 6 titik), `tim_e.gpkg` (rusak), dan `tim_f.gpkg` (kolom `Kondisi` hilang). Berkas `tim_x.gpkg` memang tidak ada. Ganti `DATA` di baris 6 sesuai komputer Anda. Dijalankan di Editor Skrip QGIS, seperti di Bab 14.

Gambar berikut menunjukkan empat penjaga di dalam fungsi `muat_vektor`.

Ilustrasi 22.2: Fungsi penjaga muat_vektor



Skrip utuh:

```python
# [SKRIP 4.1: Memeriksa masukan sebelum dipakai dan menangkap galat dengan try-except]
# Penulis: Badar Mubarok Yogaswara
import os
from qgis.core import QgsVectorLayer

DATA = "D:/Latihan/paket-i4"
HASIL = "D:/Latihan/hasil_i4"


def muat_vektor(jalur, nama, field_wajib=()):
    """Membuka layer vektor. Bila ada yang tidak beres, berhenti dengan pesan yang jelas."""
    if not os.path.exists(jalur):
        raise FileNotFoundError("berkas tidak ada: " + jalur)
    layer = QgsVectorLayer(jalur, nama, "ogr")
    if not layer.isValid():
        raise ValueError("berkas tidak bisa dibuka (mungkin rusak): " + os.path.basename(jalur))
    kurang = [f for f in field_wajib if layer.fields().indexOf(f) < 0]
    if kurang:
        raise ValueError("field wajib tidak ada: " + ", ".join(kurang))
    if layer.featureCount() == 0:
        raise ValueError("layer kosong, tidak ada fitur")
    return layer


# Empat berkas dicoba satu per satu. Satu gagal tidak boleh menghentikan yang lain.
kasus = [
    DATA + "/Kiriman_Lapangan/tim_a.gpkg",      # baik
    DATA + "/Kiriman_Lapangan/tim_x.gpkg",      # tidak ada
    DATA + "/Kiriman_Lapangan/tim_e.gpkg",      # rusak
    DATA + "/Kiriman_Lapangan/tim_f.gpkg",      # field Kondisi tidak ada
]
for jalur in kasus:
    nama = os.path.basename(jalur)
    try:
        layer = muat_vektor(jalur, nama, field_wajib=["Kondisi", "Tinggi_Phn"])
    except FileNotFoundError as e:
```

```
        print("[LEWATI] %s -> %s" % (nama, e))
    except ValueError as e:
        print("[LEWATI] %s -> %s" % (nama, e))
    else:
        # blok else: hanya jalan bila TIDAK ada galat
        print("[OK]     %s -> %d fitur" % (nama, layer.featureCount()))
    finally:
        # blok finally: selalu jalan, berhasil atau gagal
        print("         selesai memeriksa", nama)

# Galat hitungan yang sering terjadi: pembagian dengan nol
def persen(bagian, total):
    try:
        return 100 * bagian / total
    except ZeroDivisionError:
        return None          # tidak ada jawaban yang masuk akal

print("Persen 3 dari 12:", persen(3, 12))
print("Persen 3 dari 0:", persen(3, 0))
```

Penjelasan singkat. Nomor baris mengikuti berkas di atas. Rincian tiap baris ada di Bab 10.

Baris	Bagian	Artinya
1-7	Pembuka	Judul, perkakas `os` dan `QgsVectorLayer`, serta dua alamat. Skrip ini hanya memakai `DATA`. `HASIL` tidak dipakai.
10-22	Fungsi penjaga `muat_vektor`	Empat pemeriksaan berurutan. Begitu satu gagal, fungsi memicu `raise` dengan pesan jelas dan berhenti.
25-45	Perulangan dengan `try`	`try` memanggil penjaga. Dua `except` menangkap `FileNotFoundError` dan `ValueError`. `else` jalan bila tidak ada galat. `finally` selalu jalan.
47-55	Fungsi `persen`	Galat `ZeroDivisionError` ditangkap, lalu fungsi mengembalikan `None`.

Dua hal perlu Anda amati di sini. Pertama, baris 36-39 berisi dua `except` dengan isi yang sama. Ini boleh, dan Anda akan melihat cara menggabungkannya pada latihan. Kedua, kolom wajib ditulis di baris 35, bukan di dalam fungsi. Jadi fungsi yang sama bisa dipakai untuk layer lain dengan kolom wajib yang lain.

Hasil. Layar menampilkan (berdasarkan uji pada QGIS 4.0.2):

```
HASIL
[OK]     tim_a.gpkg -> 6 fitur
         selesai memeriksa tim_a.gpkg
[LEWATI] tim_x.gpkg -> berkas tidak ada: D:/Latihan/paket-i4/Kiriman_Lapangan/tim_x.gpkg
         selesai memeriksa tim_x.gpkg
[LEWATI] tim_e.gpkg -> berkas tidak bisa dibuka (mungkin rusak): tim_e.gpkg
         selesai memeriksa tim_e.gpkg
[LEWATI] tim_f.gpkg -> field wajib tidak ada: Kondisi
         selesai memeriksa tim_f.gpkg
Persen 3 dari 12: 25.0
Persen 3 dari 0: None
```

Satu berkas baik dan tiga berkas bermasalah, tetapi keempatnya diperiksa. Perhatikan bahwa baris `selesai memeriksa` muncul untuk semuanya, karena itu tugas `finally`.

Bagian yang boleh diubah.

- **Baris 6, lokasi data.** Ganti dengan lokasi `paket-i4` di komputer Anda.
- **Baris 27-30, daftar berkas.** Tambah atau ganti berkas yang ingin diperiksa. Akibatnya, tiap berkas baru ikut diperiksa dan dilaporkan.
- **Baris 35, kolom wajib.** `field_wajib=["Kondisi", "Tinggi_Phn"]` boleh diganti nama kolom lain. Akibatnya, berkas yang tidak punya kolom itu dilewati.
- **Baris 20-21, syarat kosong.** Hapus dua baris ini bila layer kosong boleh diterima.

Jebakan.

- **Folder salah.** Bila folder di baris 27-30 salah ketik, **semua** berkas dilaporkan tidak ada, termasuk `tim_a.gpkg` yang sebenarnya baik. Pada uji, nama folder `Kiriman_Lapangan` diganti `Kiriman` dan layar menampilkan `[LEWATI] tim_a.gpkg -> berkas tidak ada: D:/Latihan/paket-i4/Kiriman/tim_a.gpkg`. Pesan itu jujur: berkas memang tidak ada di alamat itu. Periksa alamatnya, bukan datanya.
- **Salah ketik di dalam fungsi tidak tertangkap.** Skrip hanya menangkap `FileNotFoundError` dan `ValueError`. Pada uji, `layer.fields()` di baris 17 sengaja diubah menjadi `layer.field()`. Layar menampilkan `selesai memeriksa tim_a.gpkg`, lalu skrip berhenti dengan `AttributeError: 'QgsVectorLayer' object has no attribute 'field'. Did you mean: 'fields'?`. Baris `finally` tetap berjalan sebelum skrip berhenti. Ini bagus: galat yang tidak Anda duga memang harus terlihat.
- `None` tidak bisa dihitung. Fungsi `persen(3, 0)` mengembalikan `None`. Bila hasilnya langsung dipakai hitung, misalnya `persen(3, 0) + 1`, muncul `TypeError: unsupported operand type(s) for +: 'NoneType' and 'int'`. Periksa hasilnya dulu sebelum dihitung.

Latihan variasi.

1. Tambahkan `tim_b.gpkg` dan `tim_d.gpkg` ke daftar `kasus`.
2. Hapus `"Kondisi"` dari daftar kolom wajib. Berkas mana yang berubah status?
3. Gabungkan dua `except` di baris 36-39 menjadi satu.

Contoh jawaban nomor 1, hanya bagian yang berubah (sisipkan sebelum tanda `]` di baris 31):

```python
    DATA + "/Kiriman_Lapangan/tim_b.gpkg",
    DATA + "/Kiriman_Lapangan/tim_d.gpkg",
```

Layar menambah dua baris: `[OK]  tim_b.gpkg -> 5 fitur` dan `[OK]  tim_d.gpkg -> 3 fitur`. Perhatikan bahwa `tim_d.gpkg` lolos walau sistem koordinatnya berbeda. Skrip ini tidak memeriksa CRS. Pemeriksaan itu baru ada di Skrip 22.2.

Contoh jawaban nomor 2: ubah baris 35 menjadi `field_wajib=["Tinggi_Phn"]`. Layar menampilkan `[OK]  tim_f.gpkg -> 3 fitur`, karena kekurangannya hanya kolom `Kondisi`.

Contoh jawaban nomor 3 (sudah diuji, hasilnya sama persis):

```python
    except (FileNotFoundError, ValueError) as e:
        print("[LEWATI] %s -> %s" % (nama, e))
```

> **Pola yang bisa dipakai ulang:** Penjaga masukan: tulis fungsi yang memeriksa syarat satu per satu dan memicu `raise JenisGalat("pesan jelas")` bila gagal. Pemakai penjaga: perulangan berisi `try`, `except JenisGalat as e` yang mencetak pesan, dan `else` untuk jalur sukses. Ganti isi pemeriksaan, daftar berkas, dan kolom wajib. Kerangkanya tetap.

Skrip 22.2: Batch tahan galat dengan laporan

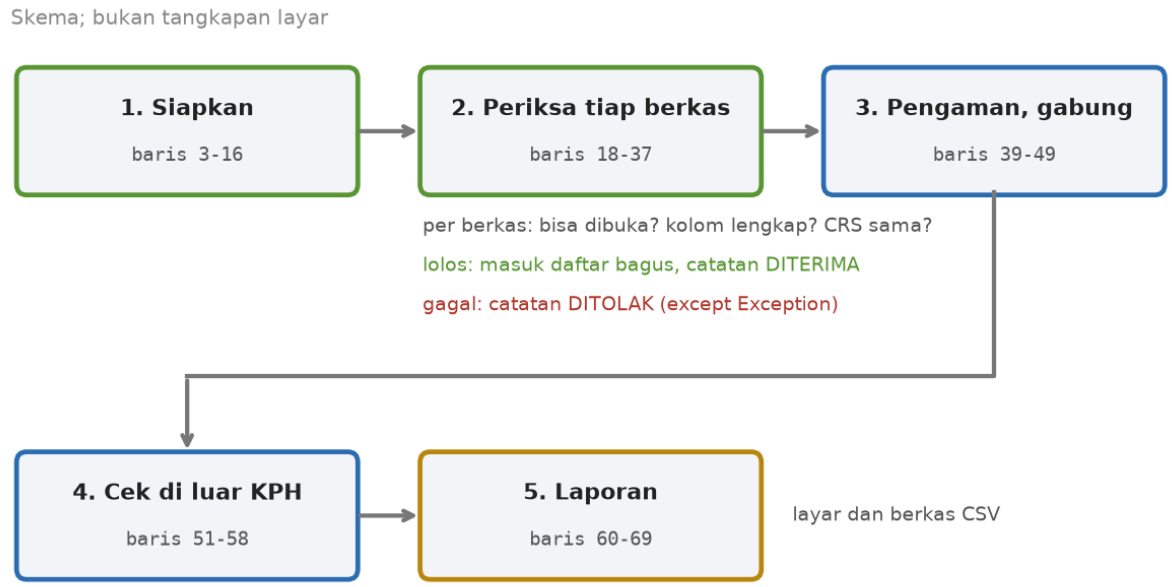
Skrip ini membawa pola penjaga ke pekerjaan nyata: menggabungkan semua kiriman lapangan. Bedanya, galat ditangkap lalu **dicatat**, bukan hanya dicetak. Catatan itu menjadi laporan untuk Kepala Seksi.

Tujuan. Menggabungkan semua kiriman lapangan menjadi satu layer, sambil mencatat berkas yang ditolak, CRS yang diubah, dan titik yang jatuh di luar semua KPH.

Bahan. Paket `paket-i4`: enam berkas `Kiriman_Lapangan/tim_a.gpkg` sampai `tim_f.gpkg`, serta `Batas_KPH.gpkg`. Berkas `b4_02_batch_tahan_galat.py` (di dalamnya tertulis "Skrip 4.2"). Ganti `DATA` di baris 9 dan `HASIL` di baris 10. Skrip membuat folder `HASIL` bila belum ada. Pekerjaan memakai alat Processing, yang dibahas di Bab 21. CRS (sistem koordinat) dibahas di Bab 14.

Skrip ini bekerja dalam lima tahap. Gambar berikut menunjukkan urutannya, lengkap dengan nomor baris.

Ilustrasi 22.3: Alur skrip 22.2



Skrip utuh:

```python
# [SKRIP 4.2: Batch tahan galat: menggabungkan kiriman lapangan, mencatat yang gagal]
# Penulis: Badar Mubarok Yogaswara
import csv
import glob
import os
import processing
from qgis.core import QgsProject, QgsVectorLayer, QgsCoordinateReferenceSystem

DATA = "D:/Latihan/paket-i4"
HASIL = "D:/Latihan/hasil_i4"
os.makedirs(HASIL, exist_ok=True)

CRS_TUJUAN = "EPSG:32749"
FIELD_WAJIB = ["Kondisi", "Tinggi_Phn", "Foto"]
log = []                    # satu baris catatan untuk tiap berkas: (berkas, status, keterangan)
bagus = []                  # layer yang lolos pemeriksaan

for jalur in sorted(glob.glob(DATA + "/Kiriman_Lapangan/*.gpkg")):
    nama = os.path.basename(jalur)
    try:
        layer = QgsVectorLayer(jalur, nama, "ogr")
        if not layer.isValid():
            raise ValueError("berkas rusak atau bukan GeoPackage")
        kurang = [f for f in FIELD_WAJIB if layer.fields().indexOf(f) < 0]
        if kurang:
            raise ValueError("field hilang: " + ", ".join(kurang))
        catatan = "sesuai"
        if layer.crs().authid() != CRS_TUJUAN:
            asal = layer.crs().authid()
            layer = processing.run("native:reprojectlayer", {
                "INPUT": layer, "TARGET_CRS": QgsCoordinateReferenceSystem(CRS_TUJUAN),
                "OUTPUT": "TEMPORARY_OUTPUT"})["OUTPUT"]
            catatan = "CRS %s diubah ke %s" % (asal, CRS_TUJUAN)
        bagus.append(layer)
        log.append((nama, "DITERIMA", "%d titik; %s" % (layer.featureCount(), catatan)))
    except Exception as e:                      # satu berkas gagal, yang lain tetap diproses
        log.append((nama, "DITOLAK", str(e)))

if not bagus:               # tanpa berkas lolos, penggabungan gagal; tampilkan dulu sebab penolakan
    for baris in log:
        print("%-22s %-9s %s" % baris)
    raise RuntimeError("Tidak ada berkas yang lolos pemeriksaan; penggabungan dibatalkan")

# Gabungkan yang lolos
gabungan = processing.run("native:mergevectorlayers", {
    "LAYERS": bagus, "CRS": QgsCoordinateReferenceSystem(CRS_TUJUAN),
    "OUTPUT": "TEMPORARY_OUTPUT"})["OUTPUT"]
gabungan.setName("Titik_Survei_Gabungan")
QgsProject.instance().addMapLayer(gabungan)

# Pemeriksaan keberadaan: titik yang tidak jatuh di dalam KPH mana pun patut dicurigai
batas = QgsVectorLayer(DATA + "/Batas_KPH.gpkg", "Batas_KPH", "ogr")
cek = processing.run("native:joinattributesbylocation", {
    "INPUT": gabungan, "JOIN": batas, "JOIN_FIELDS": ["NAMA_KPH"], "METHOD": 1,
    "PREDICATE": [0], "DISCARD_NONMATCHING": False, "OUTPUT": "TEMPORARY_OUTPUT"})["OUTPUT"]
luar = [f["Foto"] for f in cek.getFeatures() if not f["NAMA_KPH"]]
for foto in luar:
    log.append((foto, "CURIGA", "titik jatuh di luar semua KPH"))

# Laporan: layar dan berkas CSV
with open(HASIL + "/log_kiriman.csv", "w", newline="", encoding="utf-8") as f:
    w = csv.writer(f)
    w.writerow(["berkas", "status", "keterangan"])
```

```
    w.writerows(log)
for baris in log:
    print("%-22s %-9s %s" % baris)
print("Ringkasan: %d diterima, %d ditolak, %d curiga; total titik gabungan %d" % (
    sum(1 for b in log if b[1] == "DITERIMA"), sum(1 for b in log if b[1] == "DITOLAK"),
    len(luar), gabungan.featureCount()))
```

Penjelasan baris demi baris. Skrip ini 69 baris, jadi dibahas dalam enam blok. Nomor baris mengikuti berkas di atas.

Blok 1: persiapan (baris 3-16).

Baris	Tulisan	Artinya
3-6	`import csv, glob, os, processing`	Panggil perkakas: `csv` untuk menulis tabel teks, `glob` untuk mencari berkas dengan pola nama, `os` untuk urusan folder, dan `processing` untuk memanggil alat QGIS.
7	`from qgis.core import ...`	Panggil tiga perkakas QGIS: proyek, pembuka layer vektor, dan pengenal sistem koordinat.
9-11	`DATA`, `HASIL`, `os.makedirs(...)`	Dua alamat di satu tempat. `os.makedirs(HASIL, exist_ok=True)` membuat folder hasil. Bila folder sudah ada, tidak terjadi galat.
13	`CRS_TUJUAN = "EPSG:32749"`	Kode sistem koordinat yang diinginkan. EPSG:32749 adalah UTM zona 49S, sama dengan data paket ini.
14	`FIELD_WAJIB = [...]`	Tiga kolom yang harus ada di setiap berkas.
15-16	`log = []` dan `bagus = []`	Dua daftar kosong. `log` menampung catatan, `bagus` menampung layer yang lolos.

Blok 2: periksa tiap berkas (baris 18-37). Ini jantung skrip. Isi `try` memuat tiga pemeriksaan berlapis. Pemeriksaan yang gagal memicu `raise`, dan `except` di baris 36 mencatatnya.

Baris	Tulisan	Artinya
18	`for jalur in sorted(glob.glob(...)):`	`glob.glob` mencari semua berkas berpola `*.gpkg` di folder kiriman. Tanda bintang mewakili bagian nama apa saja. `sorted` mengurutkannya menurut abjad.
19-20	`nama = ...` lalu `try:`	Ambil nama berkas saja, lalu mulai bagian yang boleh gagal.
21-23	`layer = QgsVectorLayer(...)` lalu `raise ValueError(...)`	Pemeriksaan 1: berkas harus bisa dibuka. Berkas rusak menghasilkan layer tidak valid.
24-26	`kurang = [...]` lalu `raise ValueError(...)`	Pemeriksaan 2: semua kolom wajib ada. Caranya sama dengan Skrip 22.1.
27	`catatan = "sesuai"`	Catatan awal. Teks ini diganti bila CRS diubah.

Baris	Tulisan	Artinya
28-33	`if layer.crs().authid() != CRS_TUJUAN:`	Pemeriksaan 3: `authid()` memberi kode CRS layer, misalnya `EPSG:4326`. Bila beda dari tujuan, alat `native:reprojectlayer` mengubahnya. `"TEMPORARY_OUTPUT"` berarti hasil disimpan sementara. `["OUTPUT"]` mengambil layer hasil dari kamus jawaban alat.
34	`bagus.append(layer)`	Layer yang lolos masuk daftar `bagus`.
35	`log.append((nama, "DITERIMA", ...))`	Tambahkan satu catatan berisi tiga hal: nama, status, dan keterangan. Tiga hal dalam kurung ini disebut tuple (Bab 5).
36-37	`except Exception as e:`	Tangkap galat jenis apa pun. Catat berkas sebagai `DITOLAK` beserta teks galatnya, `str(e)`. `Exception` adalah induk hampir semua galat biasa.

Blok 3: pengaman bila tidak ada yang lolos (baris 39-42). Daftar kosong dianggap salah (`False`) oleh Python. Karena itu `if not bagus:` berarti "bila tidak ada satu pun layer yang lolos". `RuntimeError` adalah jenis galat umum untuk "ada yang tidak beres saat skrip berjalan". Seperti `raise ValueError` di Skrip 22.1, `raise` menghentikan skrip dengan pesan buatan sendiri.

Baris	Tulisan	Artinya
39	`if not bagus:`	Periksa apakah daftar `bagus` masih kosong.
40-41	`for baris in log:` lalu `print(...)`	Cetak dulu seluruh catatan, supaya sebab penolakan terlihat.
42	`raise RuntimeError("Tidak ada berkas yang lolos ...")`	Hentikan skrip dengan pesan jelas. Penggabungan tidak dicoba, karena tidak ada yang bisa digabung.

Blok 4: gabungkan yang lolos (baris 44-49).

Baris	Tulisan	Artinya
45-47	`processing.run("native:mergevectorlayers", {...})`	Gabungkan semua layer di daftar `bagus` menjadi satu layer sementara, dalam CRS tujuan.
48-49	`gabungan.setName(...)` lalu `addMapLayer(...)`	Beri nama yang jelas, lalu tampilkan di proyek.

Blok 5: cek titik di luar KPH (baris 51-58). Titik survei yang tidak jatuh di dalam KPH mana pun patut dicurigai: mungkin salah koordinat.

Baris	Tulisan	Artinya
52	`batas = QgsVectorLayer(...)`	Buka layer batas KPH.
53-55	`processing.run("native:joinattributesbylocation", {...})`	Tempelkan nama KPH ke tiap titik menurut letaknya. `METHOD: 1` berarti ambil pasangan pertama saja. `PREDICATE: [0]` berarti titik dan KPH harus berpotongan (`intersect`). `DISCARD_NONMATCHING: False` berarti titik tanpa pasangan tetap disimpan, dengan kolom `NAMA_KPH` kosong.
56	`luar = [f["Foto"] for f in ... if not f["NAMA_KPH"]]`	Kumpulkan kolom `Foto` dari titik yang `NAMA_KPH`-nya kosong. Kolom `Foto` dipakai sebagai pengenal titik.
57-58	`for foto in luar:`	Untuk tiap titik di luar, tambahkan catatan `CURIGA` ke `log`.

Blok 6: laporan (baris 60-69).

Baris	Tulisan	Artinya
61-64	`with open(... "log_kiriman.csv" ...)`	Tulis seluruh `log` ke berkas CSV: satu baris judul, lalu satu baris per catatan. `newline=""` mencegah baris kosong ekstra di Windows.
65-66	`print("%-22s %-9s %s" % baris)`	Cetak tiap catatan. `%-22s` menyediakan 22 kolom rata kiri, `%-9s` sembilan kolom. Tuple `baris` mengisi tiga tempat sekaligus.
67-69	`print("Ringkasan: ...")`	Hitung dan cetak ringkasan. `sum(1 for b in log if b[1] == "DITERIMA")` menambah angka 1 untuk tiap catatan berstatus itu.

Hasil. Layar menampilkan (berdasarkan uji pada QGIS 4.0.2):

```
HASIL
tim_a.gpkg              DITERIMA  6 titik; sesuai
tim_b.gpkg              DITERIMA  5 titik; sesuai
tim_c.gpkg              DITERIMA  4 titik; sesuai
tim_d.gpkg              DITERIMA  3 titik; CRS EPSG:4326 diubah ke EPSG:32749
tim_e.gpkg              DITOLAK   berkas rusak atau bukan GeoPackage
tim_f.gpkg              DITOLAK   field hilang: Kondisi
b_05.jpg                CURIGA    titik jatuh di luar semua KPH
Ringkasan: 4 diterima, 2 ditolak, 1 curiga; total titik gabungan 18
```

Berkas `log_kiriman.csv` berisi tujuh baris catatan yang sama, ditambah satu baris judul `berkas,status,keterangan`. Dari hasil ini Anda tahu empat hal. Empat berkas diterima. Dua berkas ditolak beserta alasannya. `tim_d.gpkg` diterima setelah diubah dari EPSG:4326. Satu titik, `b_05.jpg`, jatuh di luar semua KPH. Total titik gabungan 18, yaitu 6 + 5 + 4 + 3.

Bagian yang boleh diubah.

- **Baris 9-10, jalur.** Sesuaikan dengan komputer Anda.
- **Baris 13, CRS tujuan.** Ganti kode EPSG sesuai wilayah Anda. Akibatnya, berkas dengan CRS lain diubah ke kode itu.
- **Baris 14, kolom wajib.** Ganti dengan kolom yang menurut Anda harus ada.
- **Baris 18, pola nama.** `*.gpkg` boleh diganti, misalnya `tim_*.gpkg`.
- **Baris 39-42, pengaman.** Pesan di baris 42 boleh diganti kalimat Anda sendiri. Jangan hapus pengaman ini. Tanpa baris itu, daftar `bagus` yang kosong membuat penggabungan gagal dengan pesan yang tidak menyebut sebabnya (lihat Jebakan).
- Baris 54, `METHOD` dan `PREDICATE`. Angkanya mengikuti daftar pilihan alat. Menurut daftar alat di QGIS 4.0.2, `PREDICATE` 0 berarti `intersect`, 1 `contain`, 2 `equal`, 3 `touch`, 4 `overlap`, 5 `are within`, dan 6 `cross`. `METHOD` 0 membuat satu fitur per pasangan, 1 mengambil pasangan pertama, dan 2 mengambil pasangan dengan irisan terbesar.

Jebakan.

- `except Exception` menelan salah ketik Anda. Bila `layer.fields()` di baris 24 salah ditulis menjadi `layer.field()`, setiap berkas yang bisa dibuka dicatat `DITOLAK` dengan teks `'QgsVectorLayer' object has no attribute 'field'`. Daftar `bagus` jadi kosong. Pengaman di baris 39-42 lalu mencetak seluruh `log` dan berhenti dengan `RuntimeError`. Jadi sebab sebenarnya terlihat di layar. Tanpa pengaman itu, skrip berhenti di penggabungan dengan `QgsProcessingException: Could not create memory layer`, dan laporan tidak sempat tercetak. Bila Anda melihat teks galat yang sama pada banyak berkas baik, curigai skrip Anda, bukan datanya.
- **Folder kiriman salah.** Bila pola di baris 18 tidak menemukan berkas, `bagus` juga kosong dan `log` kosong. Layar hanya menampilkan `RuntimeError: Tidak ada berkas yang lolos pemeriksaan; penggabungan dibatalkan`, tanpa daftar berkas. Pesan ini tidak menyebut folder. Bila Anda melihatnya dan `log` tidak berisi catatan apa pun, periksa dulu `DATA` di baris 9 dan pola di baris 18.
- **Kolom pertama laporan bercampur.** Untuk berkas, kolom pertama berisi nama berkas (`tim_a.gpkg`). Untuk titik curiga, kolom pertama berisi nama foto (`b_05.jpg`). Itu bukan galat, tetapi hati-hati saat membaca. Untuk mengetahui berkas asal titik itu, buka kolom `Foto` di layer `Titik_Survei_Gabungan`.
- **Titik curiga tidak dibuang.** Ringkasan menyebut 1 curiga, tetapi total titik tetap 18. Titik itu hanya ditandai.

Latihan variasi.

1. Tambahkan pemeriksaan: tolak berkas yang tidak punya fitur sama sekali.
2. Setelah laporan, cetak ulang hanya catatan yang berstatus `DITOLAK`.

Contoh jawaban nomor 1 (sudah diuji): sisipkan dua baris ini setelah baris 23.

```
PYTHON
        if layer.featureCount() == 0:
            raise ValueError("layer kosong")
```

Pada data paket ini tidak ada berkas kosong, jadi hasilnya tetap sama. Pemeriksaan ini baru bekerja bila ada kiriman kosong.

Contoh jawaban nomor 2 (sudah diuji): tambahkan di akhir skrip.

```
PYTHON
print("--- hanya yang ditolak ---")
for baris in log:
    if baris[1] == "DITOLAK":
        print(baris)
```

Layar menambah tiga baris, yaitu judul dan dua catatan. Misalnya `('tim_f.gpkg', 'DITOLAK', 'field hilang: Kondisi')`.

> **Pola yang bisa dipakai ulang:** Kerangka batch tahan galat: siapkan dua daftar kosong `log` dan `bagus`. Untuk tiap berkas, `try` memeriksa dan memperbaiki, lalu menambah ke `bagus` dan `log`. `except Exception as e` mencatat berkas yang gagal ke `log`. Setelah perulangan, bila `bagus` kosong, cetak `log` lalu `raise` dengan pesan jelas. Bila tidak, proses `bagus` dan tulis `log` ke layar dan CSV. Ganti isi pemeriksaan dan alat yang dipanggil.

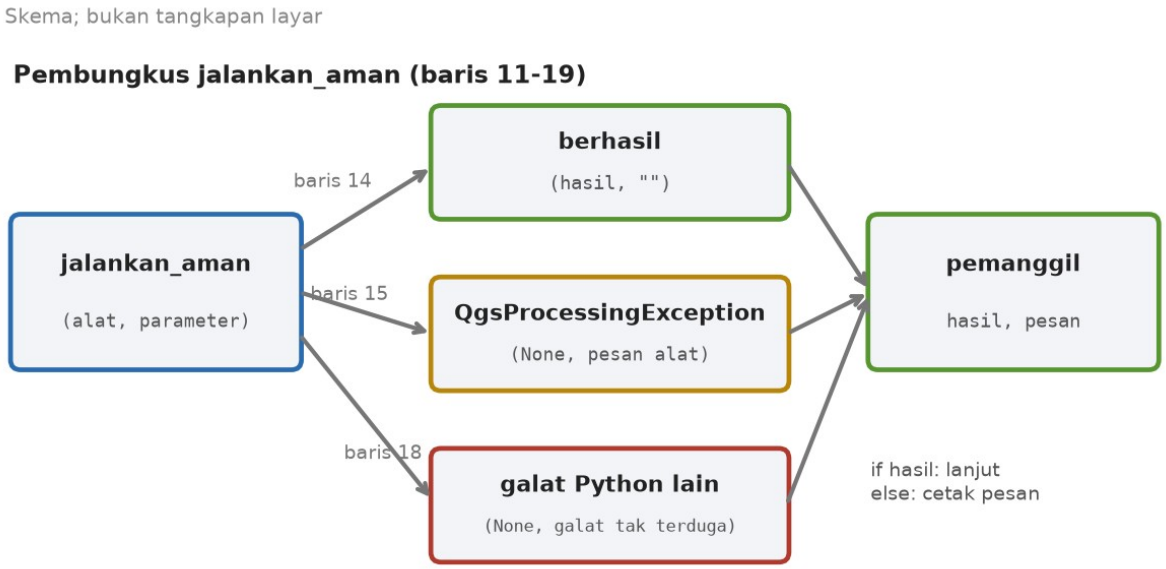
Skrip 22.3: Pembungkus untuk alat Processing

Skrip 22.2 menangkap galat di dalam perulangan. Skrip ini menyiapkan cara yang lebih rapi: **pembungkus** kecil yang memanggil alat Processing dan **tidak pernah** menghentikan skrip. Pembungkus itu selalu mengembalikan dua hal: hasil dan pesan.

Tujuan. Membuat fungsi `jalankan_aman` yang menjalankan alat Processing dan mengembalikan hasil atau pesan galat, lalu mengujinya pada empat kasus.

Bahan. Paket `paket-i4`, berkas `Sungai.gpkg`. Berkas `b4_03_galat_alat.py` (di dalamnya tertulis "Skrip 4.3"). Ganti `DATA` di baris 7. Skrip hanya mencetak teks.

Ilustrasi 22.4: Alur pembungkus jalankan_aman



Skrip utuh:

```python
# [SKRIP 4.3: Menangkap galat dari alat Processing dengan pembungkus kecil]
# Penulis: Badar Mubarok Yogaswara
import traceback
import processing
from qgis.core import QgsVectorLayer, QgsProcessingException, QgsMessageLog, Qgis

DATA = "D:/Latihan/paket-i4"
HASIL = "D:/Latihan/hasil_i4"


def jalankan_aman(alat, parameter):
    """Mengembalikan (hasil, pesan). Bila alat gagal: hasil = None dan pesan berisi sebabnya."""
    try:
        return processing.run(alat, parameter), ""
    except QgsProcessingException as e:          # galat milik Processing
        QgsMessageLog.logMessage("%s gagal: %s" % (alat, e), "Otomasi KPH", Qgis.Warning)
        return None, str(e).replace("\n", " | ")
    except Exception:                            # galat Python lain: simpan jejak untuk diagnosis
        return None, "galat tak terduga: " + traceback.format_exc().splitlines()[-1]


sungai = QgsVectorLayer(DATA + "/Sungai.gpkg", "Sungai", "ogr")

# Kasus 1: berjalan normal
hasil, pesan = jalankan_aman("native:buffer", {
    "INPUT": sungai, "DISTANCE": 30, "OUTPUT": "TEMPORARY_OUTPUT"})
print("Kasus 1:", "OK, %d fitur" % hasil["OUTPUT"].featureCount() if hasil else "GAGAL: " + pesan)

# Kasus 2: masukan tidak ada
hasil, pesan = jalankan_aman("native:buffer", {
    "INPUT": DATA + "/tidak_ada.gpkg", "DISTANCE": 30, "OUTPUT": "TEMPORARY_OUTPUT"})
print("Kasus 2:", "OK" if hasil else "GAGAL: " + pesan)

# Kasus 3: nama alat salah ketik
```

```
hasil, pesan = jalankan_aman("native:buffr", {})
print("Kasus 3:", "OK" if hasil else "GAGAL: " + pesan)

# Kasus 4: parameter wajib (OVERLAY) tidak diisi
hasil, pesan = jalankan_aman("native:clip", {"INPUT": sungai, "OUTPUT": "TEMPORARY_OUTPUT"})
print("Kasus 4:", "OK" if hasil else "GAGAL: " + pesan)
```

Penjelasan baris demi baris. Nomor baris mengikuti berkas di atas.

Blok 1: perkakas dan pembungkus (baris 3-19).

Baris	Tulisan	Artinya
3	`import traceback`	Perkakas untuk membaca jejak galat, yaitu daftar langkah yang berujung pada galat (Bab 10).
5	`from qgis.core import ...`	`QgsProcessingException` adalah galat khusus milik alat Processing. `QgsMessageLog` menulis ke catatan resmi QGIS. `Qgis` menyediakan tingkat pesan seperti `Warning`.
11	`def jalankan_aman(alat, parameter):`	Buat fungsi dengan dua bahan: ID alat dan kamus parameter.
13-14	`try:` lalu `return processing.run(alat, parameter), ""`	Jalankan alat. Bila berhasil, kembalikan dua hal sekaligus: hasil dan teks kosong. Dua hal yang dipisah koma disebut pasangan (tuple).
15-17	`except QgsProcessingException as e:`	Galat dari alat. Tulis pesan ke log QGIS dengan tag `Otomasi KPH` dan tingkat `Warning`. Kembalikan `None` dan teks galat. `.replace(...)` mengganti tiap pindah baris dengan garis tegak berspasi, agar pesan muat dalam satu baris.
18-19	`except Exception:`	Galat Python lain yang tidak terduga. `traceback.format_exc().splitlines()[-1]` mengambil baris terakhir jejak galat, yaitu jenis dan sebabnya.

Blok 2: empat kasus uji (baris 22-40).

Baris	Tulisan	Artinya
22	`sungai = QgsVectorLayer(...)`	Buka layer sungai.
25-26	`hasil, pesan = jalankan_aman(...)`	Pasangan yang dikembalikan dipecah ke dua kotak: `hasil` dan `pesan`. Kasus 1 menjalankan `native:buffer` dengan jarak 30 meter.
27	`print("Kasus 1:", "OK, ..." if hasil else "GAGAL: " + pesan)`	Bentuk `A if syarat else B`. Bila `hasil` ada, cetak bagian `A`. Bila tidak, cetak bagian `B`.
30-32	Kasus 2	Masukan menunjuk berkas yang tidak ada.
35-36	Kasus 3	Nama alat salah ketik: `native:buffr`.
39-40	Kasus 4	Alat `native:clip` dijalankan tanpa parameter wajib `OVERLAY`.

Bentuk `A if syarat else B` mudah dibaca bila dipecah. Baris 27 berarti: "bila `hasil` ada, tulis `OK` dan jumlah fitur; bila tidak, tulis `GAGAL` dan pesannya". Bagian `hasil["OUTPUT"].featureCount()` hanya dihitung bila `hasil` ada. Karena itu tidak ada galat saat `hasil` kosong.

Hasil. Layar menampilkan (berdasarkan uji pada QGIS 4.0.2):

```
HASIL
Kasus 1: OK, 2 fitur
Kasus 2: GAGAL: Unable to execute algorithm | Could not load source layer for INPUT:
D:/Latihan/paket-i4/tidak_ada.gpkg not found
Kasus 3: GAGAL: Error: Algorithm native:buffr not found |
Kasus 4: GAGAL: Unable to execute algorithm | Could not load source layer for OVERLAY: no value specified for
parameter
```

Empat kasus, empat jawaban, dan skrip tidak berhenti di mana pun. Kasus 1 menghasilkan 2 fitur karena dua garis sungai disangga sendiri-sendiri, tanpa digabung. Pesan Kasus 2 dan 4 dimulai dengan `Unable to execute algorithm`, lalu sebabnya setelah garis tegak. Pesan Kasus 3 diakhiri garis tegak kosong, tidak apa-apa.

Bagian yang boleh diubah.

- **Baris 7, lokasi data.** Sesuaikan dengan komputer Anda.
- **Baris 25-26, 30-31, 35, dan 39, kasus uji.** Ganti alat dan parameternya. Cara ini cocok untuk menguji alat baru sebelum dipakai di perulangan besar.
- **Baris 16, tag log.** `"Otomasi KPH"` boleh diganti. Pesan muncul di panel Log Messages QGIS dengan tag itu [CEK: tampilan panel tidak dapat diuji tanpa antarmuka QGIS].
- **Baris 17, pemisah.** `" | "` boleh diganti, misalnya `"; "`.

Jebakan.

- **Galat tak terduga masuk jalur kedua.** Pada uji, parameter Kasus 3 diganti `None`, bukan kamus. Layar menampilkan `Kasus 3: GAGAL: galat tak terduga: TypeError: QgsProcessingAlgorithm.checkParameterValues(): argument 1 has unexpected type 'NoneType'`. Itu galat Python, bukan galat alat, sehingga ditangkap oleh `except Exception` di baris 18.
- **Nilai parameter salah.** Pada uji, `"DISTANCE": "abc"` menghasilkan `GAGAL: Unable to execute algorithm | Incorrect parameter value for DISTANCE`. Pesan menyebut nama parameternya, jadi Anda tahu ke mana mencari.
- **Pesan berbahasa Inggris.** Pesan galat alat berasal dari QGIS dan tidak diterjemahkan. Bila laporan akan dibaca orang lain, ganti dengan kalimat Anda sendiri, misalnya "masukan tidak ditemukan".
- **Hasil kosong berarti gagal.** Skrip yang memakai `jalankan_aman` wajib memeriksa `if hasil` sebelum memakai `hasil["OUTPUT"]`. Bila lupa, muncul `TypeError: 'NoneType' object is not subscriptable`.

Latihan variasi.

1. Tambahkan Kasus 5: jarak berupa teks `"abc"`.
2. Ubah Kasus 1 agar hasil penyangga digabung, dengan parameter `DISSOLVE`.

Contoh jawaban nomor 1 (sudah diuji), tambahkan di akhir skrip:

```python
# Kasus 5: jarak berupa teks
hasil, pesan = jalankan_aman("native:buffer", {"INPUT": sungai, "DISTANCE": "abc", "OUTPUT":
"TEMPORARY_OUTPUT"})
print("Kasus 5:", "OK" if hasil else "GAGAL: " + pesan)
```

Layar menampilkan `Kasus 5: GAGAL: Unable to execute algorithm | Incorrect parameter value for DISTANCE`.

Contoh jawaban nomor 2 (sudah diuji): ubah baris 26 menjadi `"INPUT": sungai, "DISTANCE": 30, "DISSOLVE": True, "OUTPUT": "TEMPORARY_OUTPUT"})`. Layar menampilkan `Kasus 1: OK, 1 fitur`, karena dua penyangga digabung menjadi satu.

> **Pola yang bisa dipakai ulang:** Kerangka pembungkus: `def jalankan_aman(alat, parameter)` berisi `try: return processing.run(alat, parameter), ""`, satu `except QgsProcessingException as e: return None, str(e)`, dan satu `except Exception` untuk sisanya. Pemakainya: `hasil, pesan = jalankan_aman(...)`, lalu `if hasil: ... else: ....` Ganti ID alat dan parameternya.

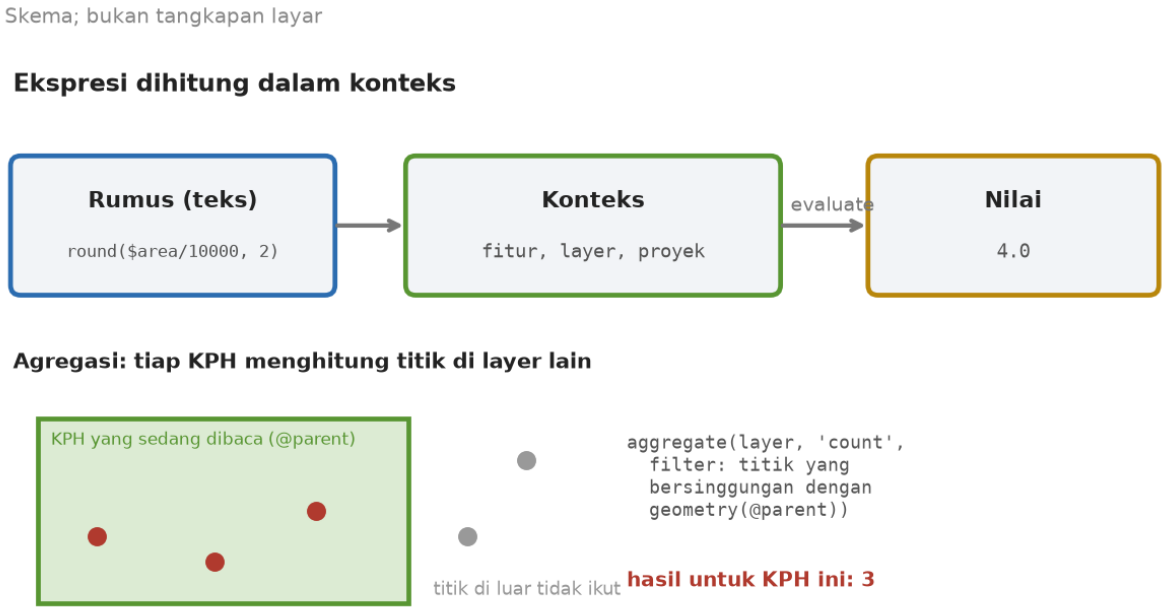
Konsep: ekspresi QGIS dalam tiga kalimat

Ekspresi adalah rumus tertulis yang dihitung QGIS untuk setiap fitur, mirip rumus di sel lembar kerja. Rumus yang sama dipakai di banyak tempat: seleksi, kalkulator kolom, label, dan model. **Agregasi**

menambah satu kemampuan, yaitu menghitung ringkasan dari layer **lain**, misalnya jumlah titik panas di dalam tiap KPH.

Bayangkan surat undangan bertuliskan "Yth. (nama tamu)". Rumusnya sama, tetapi hasilnya berbeda untuk tiap tamu. Tamu yang sedang diurus, lengkap dengan daftar tamunya, disebut **konteks**: fitur yang sedang dibaca, layernya, dan variabel proyek.

Ilustrasi 22.5: Ekspresi dalam konteks dan agregasi



Lima istilah baru:

- **Ekspresi**: rumus yang dihitung per fitur. Kutip ganda menunjuk nama kolom, kutip tunggal menulis teks.
- **Konteks**: fitur, layer, dan proyek tempat rumus dihitung.
- **Agregasi** (`aggregate`): ringkasan dari layer lain, seperti jumlah atau rata-rata.
- `@parent`: fitur yang sedang dibaca di layer luar, dipakai di dalam agregasi.
- `$area`: luas fitur yang sedang dibaca, dihitung menurut pengaturan proyek.

Skrip 22.4: Menghitung ekspresi dari Python

Tujuan. Menghitung ekspresi QGIS dari Python: rumus dasar, agregasi dari layer lain, dan ekspresi yang salah, tanpa menghentikan skrip.

Bahan. Paket `paket-i4`: `Batas_KPH.gpkg` (3 KPH), `Hotspot_Nasional.gpkg` (12 titik panas), dan `Survei.gpkg` (titik survei). Berkas `b5_01_ekspresi_python.py` (di dalamnya tertulis "Skrip 5.1"). Ganti `DATA` di baris 6. Skrip menambah tiga layer ke proyek yang sedang terbuka, dan di akhir mengubah CRS serta ellipsoid proyek itu. Jalankan di proyek kosong.

Skrip ini 77 baris dan terdiri atas enam bagian kecil. Intinya satu fungsi, `hitung`. Gambar berikut menunjukkan isinya.

Ilustrasi 22.6: Fungsi hitung



Skrip utuh:

PYTHON

```
# [SKRIP 5.1: Mengevaluasi ekspresi QGIS dari Python, termasuk agregasi]
# Penulis: Badar Mubarok Yogaswara
from qgis.core import (QgsProject, QgsVectorLayer, QgsExpression, QgsExpressionContext,
                       QgsExpressionContextUtils, QgsCoordinateReferenceSystem)

DATA = "D:/Latihan/paket-i4"
HASIL = "D:/Latihan/hasil_i4"

proyek = QgsProject.instance()
batas = QgsVectorLayer(DATA + "/Batas_KPH.gpkg", "Batas_KPH", "ogr")
hotspot = QgsVectorLayer(DATA + "/Hotspot_Nasional.gpkg", "Hotspot_Nasional", "ogr")
survei = QgsVectorLayer(DATA + "/Survei.gpkg", "Titik_Survei", "ogr")
for lyr in (batas, hotspot, survei):
    proyek.addMapLayer(lyr)          # fungsi aggregate() mencari layer lain berdasarkan NAMA di proyek


def konteks(layer, fitur):
    """Membuat 'lingkungan' tempat ekspresi dihitung: variabel proyek, layer, dan fitur saat ini."""
    c = QgsExpressionContext()
    c.appendScopes(QgsExpressionContextUtils.globalProjectLayerScopes(layer))
    c.setFeature(fitur)
    return c


def hitung(teks, layer, fitur):
    e = QgsExpression(teks)
    if e.hasParserError():                       # salah tulis ekspresi
        return "GALAT TULIS: " + e.parserErrorString()
    nilai = e.evaluate(konteks(layer, fitur))
    if e.hasEvalError():                         # salah saat dihitung
        return "GALAT HITUNG: " + e.evalErrorString()
    return nilai


# 1. Ekspresi dasar: rumus, teks, dan pilihan (CASE)
dasar = {
```

```
    "luas (ha)": "round($area / 10000, 2)",
    "nama huruf besar": 'upper("NAMA_KPH")',
    "ukuran": "CASE WHEN $area > 50000 THEN 'besar' ELSE 'kecil' END",
}
for fitur in batas.getFeatures():
    print(fitur["NAMA_KPH"], {k: hitung(v, batas, fitur) for k, v in dasar.items()})

# 2. Agregasi: menghitung dari LAYER LAIN. @parent = fitur KPH yang sedang dibaca.
titik_panas = ("aggregate(layer:='Hotspot_Nasional', aggregate:='count', expression:=\"ID_Titik\", "
               "filter:=intersects($geometry, geometry(@parent)))")
tinggi_rata = ("round(aggregate(layer:='Titik_Survei', aggregate:='mean', expression:=\"Tinggi_Phn\", "
               "filter:=intersects($geometry, geometry(@parent))), 1)")
print("--- agregasi per KPH ---")
for fitur in batas.getFeatures():
    print("%-10s titik panas = %s | tinggi pohon rata-rata = %s" % (
        fitur["NAMA_KPH"], hitung(titik_panas, batas, fitur), hitung(tinggi_rata, batas, fitur)))

# 3. Agregasi tanpa filter: satu angka untuk seluruh layer
fitur = next(batas.getFeatures())
total = hitung("aggregate('Hotspot_Nasional', 'count', \"ID_Titik\")", batas, fitur)
print("Total titik panas (semua KPH):", total)

# 4. Ekspresi yang salah dikenali dengan aman
print(hitung("round($area / 10000, ", batas, fitur))
print(hitung("to_int('abc') + 1", batas, fitur))

# 5. Dua cara lain menghitung titik: daftar dan gabungan teks
lain = {
    "jumlah lewat overlay": "array_length(overlay_intersects('Hotspot_Nasional', \"ID_Titik\"))",
    "daftar ID": ("aggregate('Hotspot_Nasional', 'concatenate', \"ID_Titik\", "
                  "filter:=intersects($geometry, geometry(@parent)), concatenator:=', ')"),
}
for fitur in batas.getFeatures():
    print(fitur["NAMA_KPH"], {k: hitung(v, batas, fitur) for k, v in lain.items()})

# 6. Awas: $area mengikuti pengaturan ellipsoid proyek, area($geometry) memakai bidang datar peta
fitur = next(batas.getFeatures())
print("Tanpa ellipsoid  :", hitung("$area", batas, fitur), hitung("area($geometry)", batas, fitur))
proyek.setCrs(QgsCoordinateReferenceSystem("EPSG:32749"))
proyek.setEllipsoid("EPSG:7030")                # WGS 84, pengaturan umum pada proyek baru
print("Dengan ellipsoid :", round(hitung("$area", batas, fitur), 2), hitung("area($geometry)", batas, fitur))
```

Penjelasan baris demi baris. Nomor baris mengikuti berkas di atas.

Blok 1: bahan (baris 3-14).

Baris	Tulisan	Artinya
3-4	`from qgis.core import (...)`	Panggil perkakas ekspresi: `QgsExpression` (rumus), `QgsExpressionContext` dan `QgsExpressionContextUtils` (konteks). Tanda kurung memungkinkan daftar panjang ditulis di dua baris.
9	`proyek = QgsProject.instance()`	Ambil proyek yang sedang terbuka (Bab 14).
10-12	Tiga `QgsVectorLayer(...)`	Buka tiga layer. Perhatikan nama layer di isian kedua, misalnya `"Hotspot_Nasional"`. Nama ini dipakai rumus di bawah.
13-14	`for lyr in (...): proyek.addMapLayer(lyr)`	Masukkan ketiganya ke proyek. Ini wajib, karena `aggregate()` mencari layer lain **menurut nama di proyek**.

Blok 2: dua fungsi pembantu (baris 17-32).

Baris	Tulisan	Artinya
17-22	`def konteks(layer, fitur):`	Bangun konteks: wadah kosong, tambahkan variabel global, proyek, dan layer (`appendScopes`), lalu tetapkan fitur yang sedang dibaca (`setFeature`).
26	`e = QgsExpression(teks)`	Ubah teks rumus menjadi objek rumus.
27-28	`if e.hasParserError():`	Periksa **galat tulis**: apakah rumusnya terbaca. Bila tidak, kembalikan teks `GALAT TULIS` berisi sebabnya.
29	`nilai = e.evaluate(konteks(layer, fitur))`	Hitung rumus dalam konteks.
30-31	`if e.hasEvalError():`	Periksa **galat hitung**: rumus terbaca, tetapi nilainya tidak bisa dihitung. Kembalikan teks `GALAT HITUNG`.
32	`return nilai`	Bila tidak ada masalah, kembalikan nilainya.

Perhatikan bahwa fungsi `hitung` tidak memakai `try` dan `except`. QGIS tidak menimbulkan galat Python untuk rumus yang salah. Ia menyimpan tanda galat di objek rumus, dan Andalah yang harus bertanya. Itulah dua pertanyaan di baris 27 dan 30.

Blok 3: ekspresi dasar (baris 35-42).

Baris	Tulisan	Artinya
36-40	`dasar = {...}`	Kamus berisi tiga rumus. Kuncinya label, nilainya teks rumus. `round($area / 10000, 2)` mengubah luas dari meter persegi ke hektare. `upper("NAMA_KPH")` mengubah huruf besar. `CASE WHEN ... END` memilih `besar` atau `kecil` menurut luas.
41-42	`for fitur in batas.getFeatures():`	Untuk tiap KPH, hitung ketiga rumus. `{k: hitung(v, batas, fitur) for k, v in dasar.items()}` membuat kamus baru dalam satu baris: tiap label dipasangkan dengan hasil rumusnya.

Blok 4: agregasi (baris 44-57).

Baris	Tulisan	Artinya
45-48	`titik_panas = (...)` dan `tinggi_rata = (...)`	Dua rumus agregasi yang ditulis di dua baris. Python menyambung teks dalam kurung otomatis. Rumus pertama menghitung (`'count'`) kolom `ID_Titik` di layer `Hotspot_Nasional`. Rumus kedua mengambil rata-rata (`'mean'`) kolom `Tinggi_Phn` di layer `Titik_Survei`, dibulatkan satu desimal.
45-48	`filter:=intersects($geometry, geometry(@parent))`	Syarat: hanya titik yang bersinggungan dengan geometri KPH yang sedang dibaca (`@parent`). Titik di luar tidak dihitung.
50-52	`for fitur in batas.getFeatures():`	Untuk tiap KPH, cetak jumlah titik panas dan tinggi rata-rata. `%-10s` menyediakan sepuluh kolom rata kiri.
55-57	`fitur = next(batas.getFeatures())`	Ambil fitur pertama saja. Lalu hitung agregasi tanpa filter, sehingga hasilnya satu angka untuk seluruh layer.

Blok 5: ekspresi yang salah dan dua cara lain (baris 59-70).

Baris	Tulisan	Artinya
60	`hitung("round($area / 10000, ", ...)`	Rumus sengaja dibiarkan terpotong. Hasilnya `GALAT TULIS`.
61	`hitung("to_int('abc') + 1", ...)`	Rumus benar, tetapi `abc` tidak bisa menjadi bilangan bulat. Hasilnya `GALAT HITUNG`.
64-68	`lain = {...}`	Dua cara lain menghitung titik. `overlay_intersects('Hotspot_Nasional', "ID_Titik")` menghasilkan **daftar** ID titik yang bersinggungan, dan `array_length` menghitung isinya. `'concatenate'` menyambung ID menjadi satu teks, dipisah `', '`.
69-70	`for fitur in batas.getFeatures():`	Cetak hasilnya per KPH.

Blok 6: dua jenis luas (baris 72-77).

Baris	Tulisan	Artinya
73-74	`$area` dan `area($geometry)`	Hitung luas KPH pertama dengan dua rumus, saat proyek belum diberi ellipsoid.
75-76	`proyek.setCrs(...)` lalu `proyek.setEllipsoid("EPSG:7030")`	Ubah CRS proyek menjadi UTM 49S dan beri ellipsoid WGS 84. Ellipsoid adalah bentuk bumi yang dipakai untuk menghitung ukuran. Pengaturan ini umum pada proyek baru.
77	`print("Dengan ellipsoid :", round(hitung("$area", ...), 2), ...)`	Hitung lagi kedua luas. `round(..., 2)` di sini aman karena `$area` menghasilkan angka.

Hasil. Layar menampilkan (berdasarkan uji pada QGIS 4.0.2):

```
HASIL
KPH Alpha {'luas (ha)': 4.0, 'nama huruf besar': 'KPH ALPHA', 'ukuran': 'kecil'}
KPH Beta {'luas (ha)': 4.0, 'nama huruf besar': 'KPH BETA', 'ukuran': 'kecil'}
KPH Gamma {'luas (ha)': 8.0, 'nama huruf besar': 'KPH GAMMA', 'ukuran': 'besar'}
--- agregasi per KPH ---
KPH Alpha  titik panas = 2 | tinggi pohon rata-rata = 11.3
KPH Beta   titik panas = 5 | tinggi pohon rata-rata = 9.6
KPH Gamma  titik panas = 5 | tinggi pohon rata-rata = 8.9
Total titik panas (semua KPH): 12
GALAT TULIS:
Incomplete expression. You might not have finished the full expression.
GALAT HITUNG: Cannot convert 'abc' to int
KPH Alpha {'jumlah lewat overlay': 2, 'daftar ID': 'HS-008, HS-012'}
KPH Beta {'jumlah lewat overlay': 5, 'daftar ID': 'HS-004, HS-005, HS-006, HS-009, HS-011'}
KPH Gamma {'jumlah lewat overlay': 5, 'daftar ID': 'HS-001, HS-002, HS-003, HS-007, HS-010'}
Tanpa ellipsoid  : 40000.0 40000.0
Dengan ellipsoid : 39997.03 40000.0
```

Cara membacanya. Baris pertama: tiga KPH, dua berluas 4 ha dan satu 8 ha, hanya Gamma yang `besar` (di atas 50.000 meter persegi). Agregasi: titik panas 2, 5, dan 5, totalnya 12, sama dengan jumlah titik di layer. Dua baris `GALAT` menunjukkan dua jenis galat ekspresi. Kata `GALAT TULIS:` diikuti baris baru, karena teks sebab dari QGIS memang diawali pindah baris. Baris terakhir: `$area` berubah dari 40000.0 menjadi 39997.03 setelah ellipsoid dipasang, sedangkan `area($geometry)` tetap 40000.0. Untuk satuan hektare dengan dua desimal, selisihnya tidak terlihat. Pada pekerjaan yang butuh ketelitian, pilih satu cara dan catat.

Bagian yang boleh diubah.

- **Baris 6, lokasi data.** Sesuaikan dengan komputer Anda.
- **Baris 36-40, rumus dasar.** Ganti atau tambah rumus Anda sendiri. Kutip ganda untuk kolom, kutip tunggal untuk teks.
- **Baris 39, angka 50000.** Ambang `besar` dan `kecil`, dalam meter persegi.

- **Baris 45-48, nama layer dan kolom.** Nama layer di dalam `aggregate` harus sama dengan nama di proyek, huruf demi huruf.
- **Baris 60-61, rumus yang salah.** Ganti dengan salah ketik buatan Anda untuk melihat pesannya.

Jebakan.

- **Layer tidak ada di proyek.** Pada uji, dua baris `addMapLayer` di baris 13-14 dilewati. Setiap agregasi berubah menjadi `GALAT HITUNG: Cannot find layer with name or ID 'Hotspot_Nasional'`. Skrip tetap berjalan, jadi galatnya tampil sebagai teks di tengah hasil. Bacalah hasilnya, jangan hanya menunggu skrip berhenti.
- **Nama layer salah.** Pada uji, `'Hotspot_Nasional'` di baris 45 diganti `'Hotspot'`. Hasilnya `GALAT HITUNG: Cannot find layer with name or ID 'Hotspot'`, hanya untuk rumus itu.
- **Nama kolom salah.** Pada uji, `"NAMA_KPH"` di baris 38 diganti `"NAMA"`. Hasilnya `GALAT HITUNG: Field 'NAMA' not found.`
- **Kurung kurang.** Pada uji, tanda kurung tutup di baris 37 dihapus. Hasilnya `GALAT TULIS` yang berakhir dengan `expecting COMMA or ')'`. Bila Anda melihat `GALAT TULIS`, periksa tanda kurung dan kutip.
- **Membulatkan hasil yang berupa teks.** Fungsi `hitung` bisa mengembalikan angka atau teks galat. Pada uji, baris 61 dibungkus `round(...)` dan skrip berhenti dengan `TypeError: type str doesn't define __round__ method`. Periksa jenis hasilnya lebih dulu dengan `isinstance(hasil, str)`, yang bertanya: "apakah hasil ini teks?"
- **Proyek Anda berubah.** Baris 13-14 menambah tiga layer, dan baris 75-76 mengubah CRS serta ellipsoid proyek yang terbuka [CEK: tampilan perubahan di jendela QGIS tidak dapat diuji dari sini]. Jalankan di proyek kosong.

Latihan variasi.

1. Tambahkan rumus keliling KPH dalam meter ke kamus `dasar`.
2. Turunkan ambang `besar` menjadi 30000. Apa yang berubah?

Contoh jawaban nomor 1 (sudah diuji): sisipkan setelah baris 39.

```
PYTHON
    "keliling (m)": "round($perimeter, 1)",
```

Layar menambah `'keliling (m)': 800.0` untuk Alpha dan Beta, dan `1200.0` untuk Gamma. Fungsi `$perimeter` memberi keliling fitur yang sedang dibaca.

Contoh jawaban nomor 2 (sudah diuji): ganti `50000` di baris 39 dengan `30000`. Ketiga KPH menjadi `besar`, karena luas terkecil 40000 meter persegi sudah melewati 30000.

> **Pola yang bisa dipakai ulang:** Kerangka menghitung ekspresi: buat `QgsExpression(teks)`, periksa `hasParserError()`, hitung dengan `evaluate(konteks)`, lalu periksa `hasEvalError()`. Konteks dibuat dengan `QgsExpressionContext`, `appendScopes(QgsExpressionContextUtils.globalProjectLayerScopes(layer))`, dan `setFeature`. Ganti teks rumus. Untuk agregasi, pastikan layer yang dirujuk sudah ada di proyek.

Skrip 22.5: Ekspresi di dalam alat Processing

Skrip 22.4 menghitung rumus dengan tangan. Skrip ini menyerahkan rumus kepada **alat Processing**, sehingga hasilnya berupa kolom baru, pilihan fitur, dan teks laporan. Rumus agregasinya sama. Karena itu hanya bagian yang baru yang dijelaskan.

Tujuan. Menambah dua kolom dengan ekspresi, memilih KPH yang titik panasnya banyak, lalu membentuk teks laporan tiga baris untuk tiap KPH.

Bahan. Paket `paket-i4`: `Batas_KPH.gpkg` dan `Hotspot_Nasional.gpkg`. Berkas `b5_02_ekspresi_di_alat.py` (di dalamnya tertulis "Skrip 5.2"). Ganti `DATA` di baris 7. Skrip menambah layer ke proyek yang terbuka, jadi jalankan di proyek kosong.

Ilustrasi 22.7: Alur skrip 22.5



Skrip utuh:

```python
# [SKRIP 5.2: Ekspresi di dalam alat Processing: Field calculator, Extract by expression, label]
# Penulis: Badar Mubarok Yogaswara
import processing
from qgis.core import (QgsProject, QgsVectorLayer, QgsExpression, QgsExpressionContext,
                       QgsExpressionContextUtils)

DATA = "D:/Latihan/paket-i4"
HASIL = "D:/Latihan/hasil_i4"

proyek = QgsProject.instance()
batas = QgsVectorLayer(DATA + "/Batas_KPH.gpkg", "Batas_KPH", "ogr")
hotspot = QgsVectorLayer(DATA + "/Hotspot_Nasional.gpkg", "Hotspot_Nasional", "ogr")
proyek.addMapLayer(batas)
proyek.addMapLayer(hotspot)          # wajib ada di proyek agar aggregate() menemukannya

# 1. Menambah dua kolom baru dengan ekspresi
tahap1 = processing.run("native:fieldcalculator", {
    "INPUT": batas, "FIELD_NAME": "LUAS_HA", "FIELD_TYPE": 0,          # 0 = desimal
    "FIELD_LENGTH": 10, "FIELD_PRECISION": 2,
    "FORMULA": "round($area / 10000, 2)", "OUTPUT": "TEMPORARY_OUTPUT"})["OUTPUT"]
tahap2 = processing.run("native:fieldcalculator", {
    "INPUT": tahap1, "FIELD_NAME": "JML_HOTSPOT", "FIELD_TYPE": 1,     # 1 = bilangan bulat
    "FIELD_LENGTH": 5, "FIELD_PRECISION": 0,
    "FORMULA": "aggregate('Hotspot_Nasional', 'count', \"ID_Titik\", "
               "intersects($geometry, geometry(@parent)))",
```

```
    "OUTPUT": "TEMPORARY_OUTPUT"})["OUTPUT"]
tahap2.setName("Batas_KPH_Ringkas")
proyek.addMapLayer(tahap2)
print("Kolom:", [f.name() for f in tahap2.fields()])
for f in tahap2.getFeatures():
    print("  ", f["NAMA_KPH"], f["LUAS_HA"], "ha;", f["JML_HOTSPOT"], "titik panas")

# 2. Memilih fitur dengan ekspresi lewat alat
rawan = processing.run("native:extractbyexpression", {
    "INPUT": tahap2, "EXPRESSION": '"JML_HOTSPOT" >= 5', "OUTPUT": "TEMPORARY_OUTPUT"})["OUTPUT"]
print("KPH dengan titik panas >= 5:", sorted(f["NAMA_KPH"] for f in rawan.getFeatures()))

# 3. Teks laporan satu KPH, dibentuk dengan ekspresi (cara yang sama dipakai label di layout)
teks = ("'LAPORAN ' || upper(\"NAMA_KPH\") || '\n' || 'Luas: ' || \"LUAS_HA\" || ' ha\n' || "
        "'Titik panas: ' || \"JML_HOTSPOT\" || ' titik'")
ekspresi = QgsExpression(teks)
for f in tahap2.getFeatures():
    c = QgsExpressionContext()
    c.appendScopes(QgsExpressionContextUtils.globalProjectLayerScopes(tahap2))
    c.setFeature(f)
    print(ekspresi.evaluate(c))
    print("---")
```

Penjelasan bagian yang baru. Baris 3-14 sama dengan Skrip 22.4: perkakas, dua layer, dan `addMapLayer`. Komentar di baris 14 mengingatkan bahwa layer `Hotspot_Nasional` wajib ada di proyek agar `aggregate()` menemukannya. Nomor baris mengikuti berkas di atas.

Baris	Tulisan	Artinya
17-20	`processing.run("native:fieldcalculator", {...})`	Panggil alat Field calculator. `FIELD_NAME` nama kolom baru, `FIELD_TYPE: 0` jenis desimal, `FIELD_LENGTH` dan `FIELD_PRECISION` panjang dan jumlah desimal, `FORMULA` rumusnya. Hasilnya layer baru dengan kolom `LUAS_HA`.
21-26	tahap kedua dengan `FIELD_TYPE: 1`	Kolom bilangan bulat `JML_HOTSPOT`. Masukannya hasil tahap pertama (`tahap1`). Rumusnya agregasi seperti di Skrip 22.4. Tanda `\"` menulis kutip ganda di dalam teks Python.
27-28	`tahap2.setName(...)` lalu `addMapLayer`	Beri nama `Batas_KPH_Ringkas` dan tampilkan di proyek.
29-31	`print("Kolom:", [f.name() for f in tahap2.fields()])`	Tampilkan nama kolom, lalu isi tiap KPH. Daftar nama kolom dibuat dengan comprehension (Bab 6).
34-35	`processing.run("native:extractbyexpression", {...})`	Alat Extract by expression: hanya fitur yang memenuhi `"JML_HOTSPOT" >= 5` masuk ke hasil.
36	`sorted(f["NAMA_KPH"] for f in rawan.getFeatures())`	Cetak nama KPH yang lolos, urut abjad.
39-40	`teks = (...)`	Satu rumus panjang yang menyusun laporan tiga baris. Dua garis tegak berurutan pada rumus QGIS menyambung teks. Itu bukan operator Python. `\n` membuat baris baru.
41-47	`QgsExpression(teks)` lalu perulangan	Untuk tiap KPH, bangun konteks seperti di Skrip 22.4, lalu hitung dan cetak teksnya. `---` memisahkan laporan.

Rumus laporan di baris 39-40 terlihat padat. Ditulis sebagai ekspresi QGIS biasa, bentuknya begini:

EKSPRESI

```
'LAPORAN ' || upper("NAMA_KPH") || '
' || 'Luas: ' || "LUAS_HA" || ' ha
' || 'Titik panas: ' || "JML_HOTSPOT" || ' titik'
```

Teks ini sama dengan rumus label di layout peta (Bab 17), sehingga rumus yang Anda uji di sini bisa dipindahkan ke label.

Hasil. Layar menampilkan (berdasarkan uji pada QGIS 4.0.2):

HASIL

```
Kolom: ['fid', 'NAMA_KPH', 'LUAS_HA', 'JML_HOTSPOT']
   KPH Alpha 4.0 ha; 2 titik panas
   KPH Beta 4.0 ha; 5 titik panas
   KPH Gamma 8.0 ha; 5 titik panas
KPH dengan titik panas >= 5: ['KPH Beta', 'KPH Gamma']
LAPORAN KPH ALPHA
Luas: 4 ha
Titik panas: 2 titik
---
LAPORAN KPH BETA
Luas: 4 ha
Titik panas: 5 titik
---
LAPORAN KPH GAMMA
Luas: 8 ha
Titik panas: 5 titik
---
```

Perhatikan dua hal. Pertama, kolom yang muncul adalah `fid`, `NAMA_KPH`, `LUAS_HA`, dan `JML_HOTSPOT`. Kolom `fid` adalah nomor urut bawaan GeoPackage. Kedua, pada laporan tertulis `Luas: 4 ha`, bukan `4.0 ha`. Penyambung teks di QGIS menulis angka bulat tanpa desimal. Bila Anda ingin dua desimal, bungkus dengan `format_number("LUAS_HA", 2)` [CEK: fungsi ini belum diuji di bab ini].

Bagian yang boleh diubah.

- **Baris 7, lokasi data.** Sesuaikan dengan komputer Anda.
- **Baris 18 dan 22, nama kolom baru.** Ganti `LUAS_HA` dan `JML_HOTSPOT` dengan nama Anda.
- **Baris 20 dan 24-25, rumus.** Ganti dengan rumus lain. Akibatnya, isi kolom baru berubah.
- Baris 35, ambang `>= 5`. Ubah syarat pemilihan. Akibatnya, daftar KPH yang terpilih berubah.
- **Baris 39-40, bunyi laporan.** Ganti kata-katanya. Akibatnya, teks laporan berubah.

Jebakan.

- **Layer rujukan tidak ada di proyek.** Pada uji, baris 14 diganti `pass`. Skrip berhenti dengan `QgsProcessingException: Evaluation error in expression "aggregate('Hotspot_Nasional', 'count', "ID_Titik", intersects($geometry, geometry(@parent)))": Cannot find layer with name or ID 'Hotspot_Nasional'.` Berbeda dengan Skrip 22.4, di sini galatnya benar-benar menghentikan skrip. Karena itu pembungkus `jalankan_aman` dari Skrip 22.3 berguna untuk alat jenis ini.

- `FIELD_PRECISION` boleh dihilangkan. Pada uji, `"FIELD_PRECISION": 2` di baris 19 dibuang dan skrip tetap jalan dengan hasil yang sama. Tetap isi supaya bentuk kolomnya jelas.
- **Angka tampil tanpa desimal.** Lihat Hasil di atas. Itu bukan kesalahan data.

Latihan variasi.

1. Pilih KPH dengan titik panas **dua atau lebih**, bukan lima.
2. Pilih KPH yang luasnya lebih dari 5 ha.

Contoh jawaban nomor 1 (sudah diuji): ubah baris 35 menjadi `"INPUT": tahap2, "EXPRESSION": '"JML_HOTSPOT" >= 2', "OUTPUT": "TEMPORARY_OUTPUT"})`. Layar menampilkan `['KPH Alpha', 'KPH Beta', 'KPH Gamma']`.

Contoh jawaban nomor 2 (sudah diuji): ganti syaratnya menjadi `'"LUAS_HA" > 5'`. Layar menampilkan `['KPH Gamma']`, satu-satunya KPH berluas 8 ha.

> **Pola yang bisa dipakai ulang:** Rumus yang sama bisa dipakai di tiga tempat: `native:fieldcalculator` (kolom baru), `native:extractbyexpression` (pilih fitur), dan `QgsExpression.evaluate` (teks laporan). Uji rumus dulu dengan fungsi `hitung` dari Skrip 22.4, lalu pindahkan ke alat. Ganti rumus dan nama kolomnya.

Cek paham

1. Apa beda `else` dan `finally` pada blok `try`? Mengapa `finally` berguna pada Skrip 22.1?
2. Pada Skrip 22.2, apa risikonya memakai `except Exception`, dan bagaimana cara mengurangi risiko itu?
3. Fungsi `hitung` di Skrip 22.4 tidak memakai `try`. Bagaimana ia tahu bahwa rumusnya salah?

Jawaban.

1. `else` hanya berjalan bila tidak ada galat. `finally` selalu berjalan. Pada Skrip 22.1, `finally` mencetak "selesai memeriksa" untuk setiap berkas, bahkan ketika skrip berhenti karena salah ketik.
2. `except Exception` menangkap segala galat, termasuk salah ketik Anda sendiri, sehingga sebab asli tersembunyi. Cara mengurangi risiko: cetak atau simpan teks galatnya (`str(e)`), cetak laporan sebelum langkah berikutnya, dan hentikan skrip dengan pesan jelas bila hasilnya kosong. Skrip 22.2 sudah melakukan dua hal terakhir lewat pengaman di baris 39-42.
3. QGIS tidak memicu galat Python untuk rumus yang salah. Ia menandai objek rumusnya. Fungsi `hitung` bertanya lewat `hasParserError()` (salah tulis) dan `hasEvalError()` (salah hitung).

Kesalahan umum

- **Menangkap galat lalu diam saja.** Perbaikan: selalu cetak atau catat sebabnya, seperti `log` di Skrip 22.2.
- **Lupa memasukkan layer rujukan ke proyek.** Perbaikan: `addMapLayer` untuk setiap layer yang disebut di dalam `aggregate()`, dan samakan nama layer dengan teks di rumus.
- **Memakai hasil alat sebelum diperiksa.** Perbaikan: periksa `if hasil` atau `isValid()` sebelum memakai `hasil["OUTPUT"]`.

- **Menjalankan skrip contoh di proyek penting.** Perbaikan: skrip 22.4 dan 22.5 menambah layer dan mengubah pengaturan proyek yang terbuka. Jalankan di proyek kosong.

Ringkasan dan latihan

- Galat bukan musuh. `try` dan `except` membuat satu berkas rusak tidak menghentikan yang lain.
- Pola penjaga: fungsi memeriksa syarat dan memicu `raise`, pemakainya menangkap lalu mencatat.
- Batch tahan galat menyimpan `log` berisi nama, status, dan keterangan, lalu menampilkannya sebagai laporan. Bila tidak ada berkas yang lolos, laporan dicetak dulu lalu skrip berhenti dengan pesan jelas.
- Ekspresi QGIS dihitung dalam konteks. Dua jenis galatnya: galat tulis dan galat hitung. Keduanya dibaca lewat tanda di objek rumus.
- Rumus yang sama bekerja di kalkulator kolom, pemilihan fitur, dan teks laporan.

Latihan. Buka `Skrip 22.2` dan tambahkan satu pemeriksaan baru: tolak berkas yang jumlah titiknya kurang dari 4. Berkas mana yang akan ditolak pada data paket ini? Jawab dulu dari hasil di bab ini, baru jalankan untuk membuktikan.

Tabel perbandingan: jenis galat dan cara menanganinya

Jenis galat	Contoh di bab ini	Ditangkap dengan	Skrip
`FileNotFoundError`	Berkas `tim_x.gpkg` tidak ada	`except FileNotFoundError`	22.1
`ValueError`	Berkas rusak, kolom hilang	`except ValueError`	22.1
`ZeroDivisionError`	Persen dari total nol	`except ZeroDivisionError`	22.1
`Exception` (induk galat)	Segala galat per berkas	`except Exception as e`	22.2
`QgsProcessingException`	Alat tidak ada, parameter kurang	`except QgsProcessingException`	22.3
Galat tulis rumus	Kurung tidak ditutup	`hasParserError()`	22.4
Galat hitung rumus	`to_int('abc')`	`hasEvalError()`	22.4

BAB 23: Menyatukan Semuanya: Model, Proyek Terpadu, dan Ceklis Masuk M1

> **Berdasarkan Seri I4:** Skrip bab ini berasal dari Buku I4, *Otomasi Dasar dengan Python* (Intermediate Class), Bab 6 dan Bab 7, berkas `b6_01_buat_model.py`, `b6_02_jalankan_model.py`, `b7_01_proyek_terpadu.py`, `b7_02_tugas_ringkasan.py`, dan `b7_03_ceklis_masuk_m1.py`. Di dalam berkasnya tertulis nomor Skrip 6.1 sampai 7.3, mengikuti penomoran buku itu. Di bab ini kelimanya diberi nomor Skrip 23.1 sampai 23.5. Di buku itu, model dibuat dengan klik, dan proyek terpadu dibahas sebagai jembatan ke kelas Master. Bab ini menunjukkan sisi skripnya.

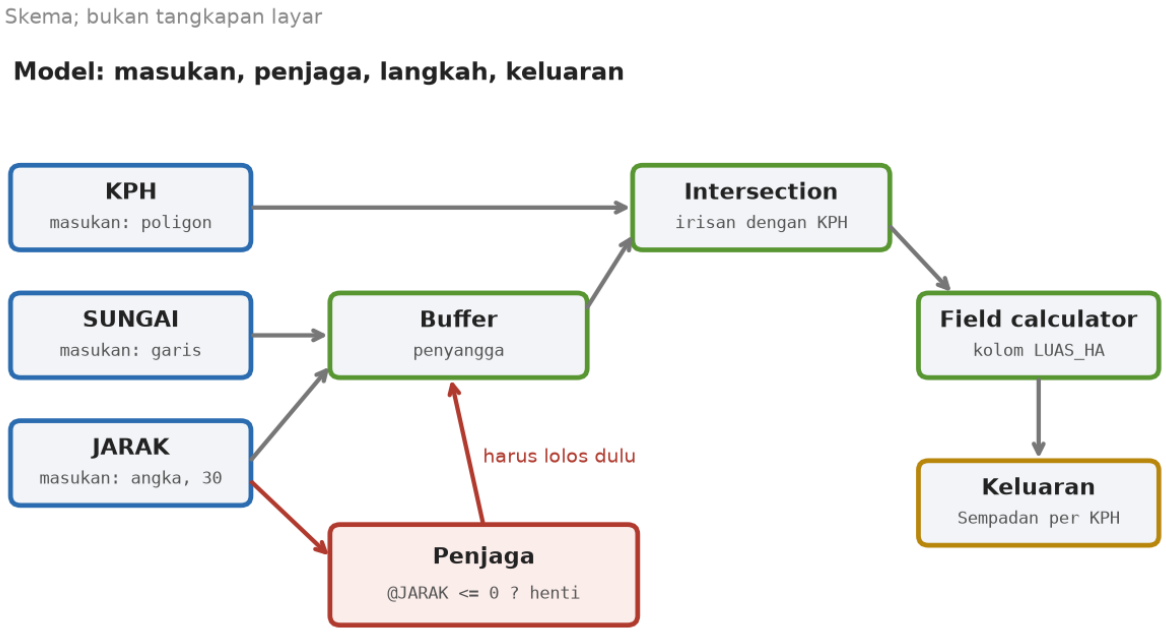
Studi kasus: "Sempadan 20, 30, atau 50 meter, lalu laporan awal bulan"

Kepala Seksi belum memutuskan lebar sempadan sungai. "Coba 20 meter, 30 meter, lalu 50 meter, dan tunjukkan luasnya." Anda ingin satu mesin yang menerima lebar sebagai isian, dan menolak isian yang tidak masuk akal. Awal bulan tiba, dan ia minta laporan lengkap: kiriman lapangan, sempadan, titik panas, dan kondisi vegetasi. Tahun lalu pekerjaan itu memakan dua hari. Kali ini Anda ingin satu skrip yang menyelesaikannya, lalu satu ujian untuk mengetahui apakah Anda siap ke kelas berikutnya.

Konsep: model sebagai berkas, dalam tiga kalimat

Model adalah alur kerja yang disimpan sebagai berkas `.model3`: kotak masukan, kotak alat, dan panah di antaranya. Bayangkan mesin cuci dengan tombol pengaturan. Mesinnya satu, tetapi hasilnya berbeda menurut pengaturan. Karena modelnya hanya berkas teks, ia bisa dibuat dengan klik di Model Designer, atau ditulis dengan Python seperti di bab ini.

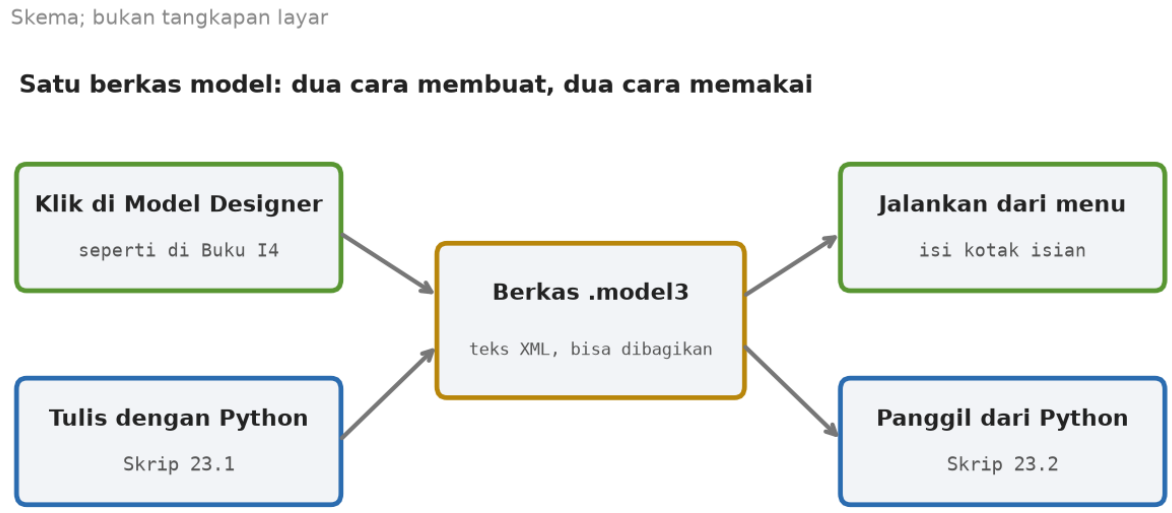
Ilustrasi 23.1: Model: masukan, penjaga, langkah, keluaran



Lima istilah baru:

- **Model**: alur kerja yang disimpan dalam berkas `.model3`.
- **Masukan model**: isian yang diisi pengguna saat model dijalankan.
- **Langkah**: satu alat di dalam model. Di API QGIS disebut *child algorithm*.
- **Penjaga**: langkah yang menghentikan model bila syarat gagal.
- **Sumber nilai**: asal isian sebuah langkah: nilai tetap, masukan model, keluaran langkah lain, atau hasil ekspresi.

Ilustrasi 23.2: Satu berkas model, dua cara membuat, dua cara memakai



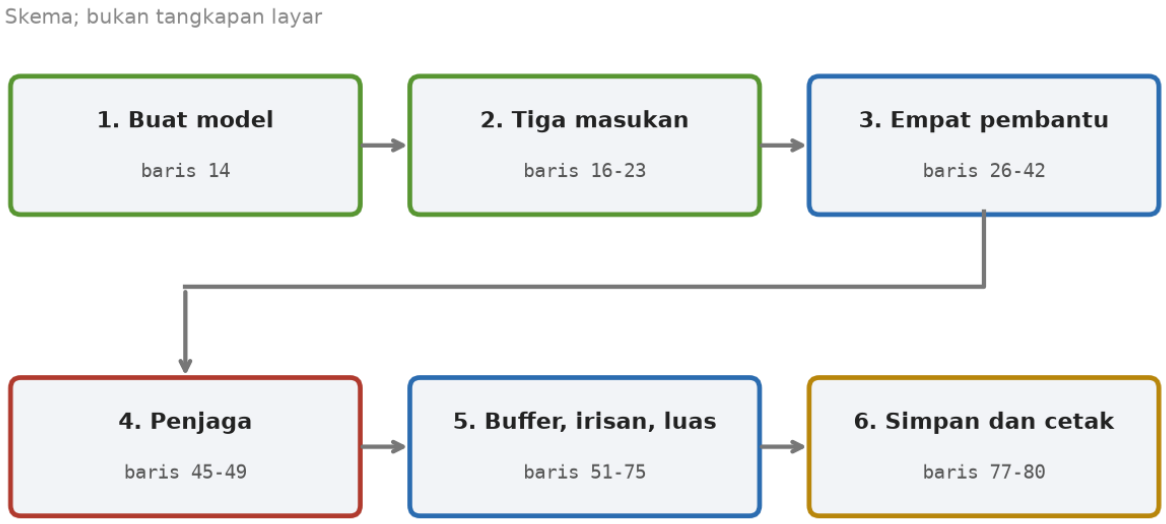
Skrip 23.1: Membangun model lewat Python

Tujuan. Menyusun model "Sempadan Sungai per KPH" dengan kode, lengkap dengan penjaga jarak, lalu menyimpannya sebagai `.model3`.

Bahan. Skrip ini tidak membaca data. Ia hanya menulis satu berkas, `Sempadan_Sungai_per_KPH.model3`, di folder `HASIL`. Berkas `b6_01_buat_model.py` (di dalamnya tertulis "Skrip 6.1"). Ganti `HASIL` di baris 11. Salinan jadi model ini juga ada di folder `skrip/model` pada Seri I4. Dijalankan di Editor Skrip QGIS, seperti di Bab 14.

Skrip ini bekerja dalam enam tahap. Gambar berikut menunjukkan urutannya, lengkap dengan nomor baris.

Ilustrasi 23.3: Alur skrip 23.1



Urutan jalan model ditentukan oleh panah sumber dan ketergantungan, bukan urutan baris skrip.

Skrip utuh:

```python
# [SKRIP 6.1: Membangun model "Sempadan Sungai per KPH" lewat Python, simpan sebagai .model3]
# Penulis: Badar Mubarok Yogaswara
# Model ini sama dengan yang bisa Anda gambar di Graphical Modeler; ini hanya cara lain membuatnya.
import os
from qgis.core import (QgsProcessing, QgsProcessingModelAlgorithm, QgsProcessingModelChildAlgorithm,
                       QgsProcessingModelChildParameterSource as Sumber, QgsProcessingModelParameter,
                       QgsProcessingModelOutput, QgsProcessingModelChildDependency,
                       QgsProcessingParameterVectorLayer, QgsProcessingParameterNumber)

DATA = "D:/Latihan/paket-i4"
HASIL = "D:/Latihan/hasil_i4"
os.makedirs(HASIL, exist_ok=True)

m = QgsProcessingModelAlgorithm("Sempadan Sungai per KPH", "Kehutanan")

# --- Masukan model (yang diisi pengguna saat model dijalankan)
m.addModelParameter(QgsProcessingParameterVectorLayer(
    "KPH", "Layer batas KPH", [QgsProcessing.TypeVectorPolygon]), QgsProcessingModelParameter("KPH"))
m.addModelParameter(QgsProcessingParameterVectorLayer(
    "SUNGAI", "Layer sungai", [QgsProcessing.TypeVectorLine]), QgsProcessingModelParameter("SUNGAI"))
m.addModelParameter(QgsProcessingParameterNumber(
    "JARAK", "Lebar sempadan (m)", QgsProcessingParameterNumber.Double, 30),
    QgsProcessingModelParameter("JARAK"))


def anak(alat, deskripsi):
    c = QgsProcessingModelChildAlgorithm(alat)
    c.generateChildId(m)
    c.setDescription(deskripsi)
    return c


def dari_input(nama):
```

```
    return [Sumber.fromModelParameter(nama)]


def nilai(v):
    return [Sumber.fromStaticValue(v)]


def dari_anak(c, keluaran="OUTPUT"):
    return [Sumber.fromChildOutput(c.childId(), keluaran)]


# --- Penjaga: bila JARAK tidak lebih dari 0, model berhenti dengan pesan jelas
henti = anak("native:raiseexception", "Hentikan bila jarak tidak sah")
henti.addParameterSources("MESSAGE", nilai("Lebar sempadan harus lebih dari 0 meter."))
henti.addParameterSources("CONDITION", [Sumber.fromExpression("@JARAK <= 0")])
m.addChildAlgorithm(henti)

# --- Langkah utama: baru berjalan sesudah penjaga lolos
buf = anak("native:buffer", "Penyangga sungai")
buf.addParameterSources("INPUT", dari_input("SUNGAI"))
buf.addParameterSources("DISTANCE", dari_input("JARAK"))
buf.addParameterSources("DISSOLVE", nilai(True))
buf.setDependencies([QgsProcessingModelChildDependency(henti.childId())])
m.addChildAlgorithm(buf)

iris = anak("native:intersection", "Irisan dengan KPH")
iris.addParameterSources("INPUT", dari_input("KPH"))
iris.addParameterSources("OVERLAY", dari_anak(buf))
m.addChildAlgorithm(iris)

luas = anak("native:fieldcalculator", "Hitung luas (ha)")
luas.addParameterSources("INPUT", dari_anak(iris))
luas.addParameterSources("FIELD_NAME", nilai("LUAS_HA"))
luas.addParameterSources("FIELD_TYPE", nilai(0))
luas.addParameterSources("FIELD_LENGTH", nilai(10))
luas.addParameterSources("FIELD_PRECISION", nilai(2))
# rumus dihitung per fitur, jadi diberikan sebagai nilai tetap (teks)
luas.addParameterSources("FORMULA", nilai("round($area / 10000, 2)"))
keluaran = QgsProcessingModelOutput("SEMPADAN", "Sempadan per KPH")
keluaran.setChildOutputName("OUTPUT")
luas.setModelOutputs({"SEMPADAN": keluaran})
m.addChildAlgorithm(luas)

berkas = HASIL + "/Sempadan_Sungai_per_KPH.model3"
print("Model tersimpan?", m.toFile(berkas))
print("Langkah:", [c.description() for c in m.childAlgorithms().values()])
print("Masukan:", [p.name() for p in m.parameterDefinitions()])
```

Penjelasan baris demi baris. Skrip ini 80 baris, jadi dibahas dalam tujuh blok. Nomor baris mengikuti berkas di atas.

Blok 1: pembuka (baris 1-14).

Baris	Tulisan	Artinya
1-3	`# [SKRIP 6.1 ...`	Komentar. Baris 3 menyebut bahwa model ini sama dengan yang bisa Anda gambar di Model Designer.
4	`import os`	Perkakas untuk urusan folder (Bab 8).
5-8	`from qgis.core import (...)`	Panggil perkakas pembuat model. `QgsProcessingModelChildParameterSource as Sumber` memberi nama panjang itu julukan pendek `Sumber`. Julukan seperti ini disebut alias.
10-12	`DATA, HASIL, os.makedirs(...)`	Dua alamat, lalu buat folder hasil bila belum ada. `DATA` tidak dipakai skrip ini.
14	`m = QgsProcessingModelAlgorithm ("Sempadan Sungai per KPH", "Kehutanan")`	Buat model kosong di kotak `m`. Isian pertama nama model, isian kedua nama kelompoknya.

Blok 2: tiga masukan (baris 16-23). Ini isian yang akan diisi pengguna saat model dijalankan.

Baris	Tulisan	Artinya
17-18	`m.addModelParameter(QgsProcessingParameterVectorLayer("KPH", ...), QgsProcessingModelParameter("KPH"))`	Tambah masukan bernama `KPH`. Isiannya nama internal, label di jendela ("Layer batas KPH"), dan jenis layer yang diterima: poligon. Bagian kedua mendaftarkan masukan itu pada model.
19-20	masukan `SUNGAI`	Sama, tetapi menerima layer garis.
21-23	masukan `JARAK`	Masukan angka desimal (`Double`) dengan nilai bawaan 30.

Blok 3: empat pembantu (baris 26-42). Empat fungsi kecil ini mempersingkat kode di bawahnya. Fungsi dibahas di Bab 7.

Baris	Tulisan	Artinya
26-30	`def anak(alat, deskripsi):`	Buat satu langkah. `generateChildId(m)` memberinya nomor pengenal yang unik di model, dan `setDescription` memberi nama yang terbaca manusia.
33-34	`def dari_input(nama):`	Sumber nilai dari **masukan model**. Dikembalikan dalam daftar.
37-38	`def nilai(v):`	Sumber nilai berupa **nilai tetap**.
41-42	`def dari_anak(c, keluaran="OUTPUT"):`	Sumber nilai dari **keluaran langkah lain**. Isian kedua punya nilai bawaan `"OUTPUT"`.

Sumber keempat, hasil ekspresi, dipakai langsung di baris 48 lewat `Sumber.fromExpression(...)`.

Blok 4: penjaga (baris 45-49).

Baris	Tulisan	Artinya
46	`henti = anak("native:raiseexception", ...)`	Buat langkah dari alat Raise exception, yaitu alat yang sengaja menghentikan model dengan pesan buatan Anda.
47	`henti.addParameterSources( "MESSAGE", nilai(...))`	Isi pesannya dengan nilai tetap: "Lebar sempadan harus lebih dari 0 meter."
48	`... "CONDITION", [Sumber.fromExpression("@JARAK <= 0")]`	Isi syaratnya dengan ekspresi. `@JARAK` membaca masukan model. Alat hanya menghentikan model bila syarat ini benar.
49	`m.addChildAlgorithm(henti)`	Pasang langkah ini ke model.

Blok 5: buffer dan irisan (baris 51-62).

Baris	Tulisan	Artinya
52-55	`buf = anak("native:buffer", ...)` lalu tiga `addParameterSources`	Buat langkah Buffer. `INPUT` diambil dari masukan `SUNGAI`, `DISTANCE` dari masukan `JARAK`, dan `DISSOLVE` bernilai tetap benar (hasil digabung).
56	`buf.setDependencies([QgsProcessingModelChildDependency(henti.childId())])`	Syarat urutan: Buffer baru boleh jalan **setelah** penjaga lolos. Tanpa baris ini, QGIS tidak tahu bahwa penjaga harus lebih dulu.
57	`m.addChildAlgorithm(buf)`	Pasang ke model.
59-62	`iris = anak("native:intersection", ...)`	Langkah Irisan. `INPUT` dari masukan `KPH`, `OVERLAY` dari keluaran Buffer (`dari_anak(buf)`).

Blok 6: luas dan keluaran (baris 64-75).

Baris	Tulisan	Artinya
64-69	`luas = anak("native:fieldcalculator", ...)`	Langkah Field calculator. `INPUT` dari hasil irisan, lalu nama kolom `LUAS_HA`, jenis desimal (0), panjang 10, dua desimal.
70-71	`"FORMULA"` bernilai tetap	Rumus `round($area / 10000, 2)` diberikan sebagai teks tetap, bukan hasil ekspresi. Rumus per fitur harus dihitung oleh alatnya sendiri untuk tiap fitur. Hasil ekspresi dihitung sekali untuk seluruh model, tanpa fitur.
72-74	`keluaran = QgsProcessingModelOutput("SEMPADAN", "Sempadan per KPH")`	Tandai hasil langkah ini sebagai **keluaran model**. `setChildOutputName("OUTPUT")` menyebut keluaran alat mana yang dipakai, dan `setModelOutputs` memasangnya.
75	`m.addChildAlgorithm(luas)`	Pasang langkah terakhir.

Blok 7: simpan dan lapor (baris 77-80).

Baris	Tulisan	Artinya
77-78	`m.toFile(berkas)`	Simpan model ke berkas `.model3`. Hasilnya `True` bila berhasil.
79	`[c.description() for c in m.childAlgorithms().values()]`	Cetak nama semua langkah. `.values()` mengambil isi kamus langkah.
80	`[p.name() for p in m.parameterDefinitions()]`	Cetak nama semua isian model.

Hasil. Layar menampilkan (berdasarkan uji pada QGIS 4.0.2):

```
HASIL
Model tersimpan? True
Langkah: ['Penyangga sungai', 'Hitung luas (ha)', 'Irisan dengan KPH', 'Hentikan bila jarak tidak sah']
Masukan: ['KPH', 'SUNGAI', 'JARAK', 'sempadan_per_kph']
```

Ada tiga hal yang perlu Anda baca dengan teliti. Pertama, `Model tersimpan? True` berarti berkas sudah tertulis. Kedua, urutan langkah yang dicetak tidak sama dengan urutan Anda menyusunnya: penjaga muncul paling akhir. Urutan jalan model ditentukan oleh sumber nilai dan ketergantungan, bukan urutan cetak. Ketiga, daftar masukan memuat empat nama. Yang keempat, `sempadan_per_kph`, adalah keluaran model yang diperlakukan sebagai isian (tempat menyimpan hasil). Namanya dibentuk dari label keluaran di baris 72, yaitu "Sempadan per KPH", dengan huruf kecil dan garis bawah. Pada uji, berkas hasil skrip ini identik dengan salinan di folder `skrip/model` Seri I4.

Bagian yang boleh diubah.

- **Baris 11, lokasi hasil.** Ganti dengan folder Anda. Akibatnya, berkas model tersimpan di sana.
- **Baris 14, nama model dan kelompok.** Nama ini tampil di Processing Toolbox bila model dipasang.
- **Baris 22, angka 30.** Nilai bawaan lebar sempadan.
- **Baris 47, pesan penjaga.** Ganti kalimatnya. Akibatnya, pesan itu muncul bila jarak tidak sah.
- **Baris 48, syarat.** `@JARAK <= 0` boleh diganti, misalnya `@JARAK <= 0 or @JARAK > 500`.
- Baris 55, `nilai(True)`. Ganti `False` bila potongan buffer tidak perlu digabung.

Jebakan.

- **Nama keluaran bergantung pada label.** Pada uji, label di baris 72 diganti dari "Sempadan per KPH" menjadi "Sempadan Akhir". Daftar masukan lalu memuat `sempadan_akhir`. Skrip 23.2, yang masih memakai `sempadan_per_kph`, berhenti dengan `Could not create destination layer for sempadan_akhir: no value specified for parameter`. Bila Anda mengganti label, ganti juga nama itu di Skrip 23.2.
- **Urutan cetak bukan urutan jalan.** Lihat penjelasan Hasil di atas. Jangan menebak urutan model dari daftar `Langkah`.
- `DATA` tidak dipakai. Baris 10 menyiapkan alamat yang tidak terpakai. Tidak merusak, tetapi bisa membingungkan.
- **Memberi rumus per fitur sebagai ekspresi model.** Dibahas di Blok 6. Buku I4 mencatat bahwa dalam ujinya, rumus itu bila diberikan sebagai hasil ekspresi membuat langkah gagal dengan pesan `Error encountered while running Hitung luas (ha)`. Itu sebabnya `FORMULA` berupa nilai tetap. [CEK: bagian ini tidak diuji ulang di bab ini]

Latihan variasi.

1. Ganti pesan penjaga menjadi "Jarak harus lebih besar dari nol."
2. Ubah nilai bawaan `JARAK` menjadi 50.

Contoh jawaban nomor 1 (sudah diuji): ubah baris 47 menjadi

```python
henti.addParameterSources("MESSAGE", nilai("Jarak harus lebih besar dari nol."))
```

Jalankan ulang Skrip 23.1, lalu Skrip 23.2. Pada baris terakhir hasil, pesan penghentian berubah menjadi `Jarak harus lebih besar dari nol.`

Contoh jawaban nomor 2: ubah angka 30 di baris 22 menjadi 50. Nilai itu menjadi isian awal saat model dibuka dari menu. Skrip 23.2 mengisi `JARAK` sendiri, jadi hasilnya tidak berubah.

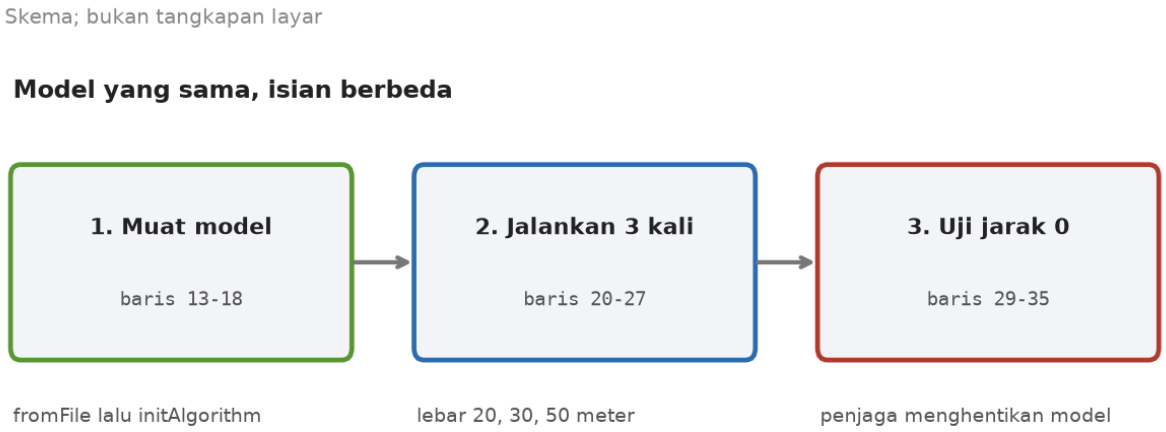
> **Pola yang bisa dipakai ulang:** Kerangka membangun model: `m = QgsProcessingModelAlgorithm(NAMA, KELOMPOK)`, tambahkan masukan dengan `addModelParameter`, buat langkah dengan fungsi `anak(ALAT, DESKRIPSI)`, isi parameternya dengan `addParameterSources` dari salah satu empat sumber nilai, pasang dengan `addChildAlgorithm`, lalu `m.toFile(...)`. Ganti alat, parameter, dan nama.

Skrip 23.2: Menjalankan model dari Python

Tujuan. Memuat model dari berkas, menjalankannya tiga kali dengan lebar sempadan berbeda, lalu menguji penjaga dengan jarak yang tidak sah.

Bahan. Paket `paket-i4`: `Batas_KPH.gpkg` dan `Sungai.gpkg`, serta berkas model dari Skrip 23.1. **Jalankan Skrip 23.1 lebih dulu.** Berkas `b6_02_jalankan_model.py` (di dalamnya tertulis "Skrip 6.2"). Ganti `DATA` di baris 7 dan `HASIL` di baris 8.

Ilustrasi 23.4: Alur skrip 23.2



Skrip utuh:

```python
# [SKRIP 6.2: Menjalankan model .model3 dari Python, berulang dengan jarak berbeda]
# Penulis: Badar Mubarok Yogaswara
# Prasyarat: jalankan SKRIP 6.1 dulu (atau simpan model dari Graphical Modeler dengan nama sama).
import processing
from qgis.core import QgsVectorLayer, QgsProcessingModelAlgorithm, QgsProcessingException

DATA = "D:/Latihan/paket-i4"
HASIL = "D:/Latihan/hasil_i4"

batas = QgsVectorLayer(DATA + "/Batas_KPH.gpkg", "Batas_KPH", "ogr")
sungai = QgsVectorLayer(DATA + "/Sungai.gpkg", "Sungai", "ogr")

# 1. Muat model dari berkas
model = QgsProcessingModelAlgorithm()
print("Model terbaca?", model.fromFile(HASIL + "/Sempadan_Sungai_per_KPH.model3"))
model.initAlgorithm({})
print("Nama model:", model.displayName())
print("Parameter:", [p.name() for p in model.parameterDefinitions()])

# 2. Jalankan berulang dengan tiga lebar sempadan. Model yang sama, angka berbeda.
for jarak in (20, 30, 50):
    hasil = processing.run(model, {
        "KPH": batas, "SUNGAI": sungai, "JARAK": jarak,
        "sempadan_per_kph": "TEMPORARY_OUTPUT"})
    lyr = hasil["sempadan_per_kph"]
    total = sum(f["LUAS_HA"] for f in lyr.getFeatures())
    print("Lebar %2d m -> %d potongan, total %.2f ha" % (jarak, lyr.featureCount(), total))

# 3. Jarak yang tidak sah: model berhenti dan memberi pesan
try:
```

```
    processing.run(model, {"KPH": batas, "SUNGAI": sungai, "JARAK": 0,
                           "sempadan_per_kph": "TEMPORARY_OUTPUT"})
    print("Jarak 0: model berjalan (tidak seharusnya)")
except QgsProcessingException as e:
    print("Jarak 0: model berhenti ->", str(e).replace("\n", " | "))
```

Penjelasan baris demi baris. Nomor baris mengikuti berkas di atas.

Baris	Tulisan	Artinya
4-5	`import processing` dan `from qgis.core import ...`	Perkakas untuk memanggil alat, membuka layer, memuat model, dan menangkap `QgsProcessingException`.
10-11	`batas = ...` dan `sungai = ...`	Buka dua layer masukan.
14-15	`model = QgsProcessingModelAlgorithm()` lalu `model.fromFile(...)`	Buat model kosong, lalu isi dari berkas. `fromFile` menjawab `True` bila berkas terbaca, dan `False` bila tidak.
16	`model.initAlgorithm({})`	Siapkan model supaya isiannya terdaftar. Tanpa baris ini, daftar isian kosong.
17-18	`displayName()` dan `parameterDefinitions()`	Cetak nama model dan nama semua isiannya.
21	`for jarak in (20, 30, 50):`	Ulangi tiga kali, dengan `jarak` berturut-turut 20, 30, dan 50. Tanda kurung di sini membuat tuple (Bab 5).
22-24	`processing.run(model, {...})`	Jalankan model. Bedanya dengan Bab 21: isian pertama adalah **objek model**, bukan teks ID alat. Kamusnya berisi `KPH`, `SUNGAI`, `JARAK`, dan tempat keluaran.
25	`lyr = hasil["sempadan_per_kph"]`	Ambil layer hasil memakai nama keluaran yang Anda lihat di Skrip 23.1. Huruf kecilnya harus persis.
26	`total = sum(f["LUAS_HA"] for f in lyr.getFeatures())`	Jumlahkan kolom `LUAS_HA` dari semua potongan.
27	`print("Lebar %2d m -> %d potongan, total %.2f ha" % ...)`	Cetak lebar, jumlah potongan, dan total luas. `%2d` bilangan bulat dua kolom, `%.2f` desimal dua angka.
30-35	`try: ... except QgsProcessingException as e:`	Jalankan model dengan `JARAK` 0. Penjaga menghentikan model, dan `except` menangkap pesannya. `.replace("\n", ...)` merapikan pesan.

Hasil. Layar menampilkan (berdasarkan uji pada QGIS 4.0.2):

```
HASIL
Model terbaca? True
Nama model: Sempadan Sungai per KPH
Parameter: ['JARAK', 'KPH', 'SUNGAI', 'sempadan_per_kph']
Lebar 20 m -> 3 potongan, total 2.75 ha
Lebar 30 m -> 3 potongan, total 4.07 ha
Lebar 50 m -> 3 potongan, total 6.55 ha
Jarak 0: model berhenti -> Lebar sempadan harus lebih dari 0 meter.
```

Semakin lebar sempadan, semakin besar luasnya: 2,75 ha untuk 20 meter, 4,07 ha untuk 30 meter, dan 6,55 ha untuk 50 meter. Selalu tiga potongan, satu per KPH. Total 4,07 ha untuk 30 meter sama dengan jumlah sempadan per KPH di Skrip 23.3 (1,18 + 1,39 + 1,50). Baris terakhir membuktikan penjaga bekerja: jarak 0 ditolak dengan pesan yang Anda tulis sendiri di Skrip 23.1.

Bagian yang boleh diubah.

- **Baris 7-8, jalur.** Sesuaikan dengan komputer Anda.
- **Baris 15, nama berkas model.** Ganti untuk memuat model lain, misalnya model buatan Anda di Model Designer.
- **Baris 21, daftar jarak.** `(20, 30, 50)` boleh diganti angka lain. Akibatnya, jumlah putaran dan hasilnya berubah.
- **Baris 23-24, nama isian.** Harus sama dengan nama di daftar `Parameter:` pada hasil.
- **Baris 31, jarak 0.** Ganti angka lain untuk menguji batas.

Jebakan.

- **Lupa menjalankan Skrip 23.1.** Pada uji, berkas model dihapus lebih dulu. Layar menampilkan `Model terbaca? False`, `Nama model: model`, dan `Parameter: []`, lalu skrip berhenti dengan `KeyError: 'sempadan_per_kph'` di baris 25. Baris pertama hasil sudah memberi tahu, sebelum galatnya muncul.
- **Huruf besar dan kecil.** Pada uji, `"sempadan_per_kph"` di baris 25 ditulis `"Sempadan_per_KPH"`. Hasilnya `KeyError: 'Sempadan_per_KPH'`. Salin nama dari daftar `Parameter:`.
- **Keluaran tampak sebagai isian.** Daftar `Parameter:` memuat `sempadan_per_kph` walau itu hasil. Di API QGIS, tempat penyimpanan hasil memang dihitung sebagai parameter.
- **Urutan daftar berbeda.** Daftar di sini urut abjad (`JARAK`, `KPH`, `SUNGAI`), sedangkan di Skrip 23.1 urut buatan. Isinya sama.

Latihan variasi.

1. Ganti daftar jarak menjadi 10, 25, 40, dan 100.
2. Cetak juga luas sempadan per KPH di tiap putaran.

Contoh jawaban nomor 1 (sudah diuji): ubah baris 21 menjadi `for jarak in (10, 25, 40, 100):`. Layar menampilkan total 1,40 ha, 3,41 ha, 5,32 ha, dan 11,50 ha.

Contoh jawaban nomor 2 (sudah diuji): sisipkan sebelum baris 27.

```
PYTHON
    for f in lyr.getFeatures():
        print("   ", f["NAMA_KPH"], f["LUAS_HA"])
```

Di putaran 30 meter, layar menambah `KPH Alpha 1.18`, `KPH Beta 1.39`, dan `KPH Gamma 1.5`.

> **Pola yang bisa dipakai ulang:** Kerangka menjalankan model: `model = QgsProcessingModelAlgorithm()`, `model.fromFile(BERKAS)`, `model.initAlgorithm({})`, lalu `processing.run(model, {NAMA_ISIAN: NILAI, ...})` di dalam perulangan. Baca nama isian dari `parameterDefinitions()`. Bungkus dengan `try` dan `except QgsProcessingException` bila ada penjaga.

Konsep: proyek terpadu dan ceklis

Proyek terpadu adalah satu skrip yang memadukan semua yang Anda pelajari di kelas ini. Bayangkan ujian praktik sebelum turun ke jalan raya: yang diuji tangan Anda, bukan hanya ingatan. **Ceklis** adalah daftar pemeriksaan yang memberi tahu apakah bekal Anda sudah cukup untuk kelas berikutnya.

Dua istilah ikut muncul: **keluaran permanen**, yaitu hasil yang disimpan ke berkas, dan **ceklis masuk**, yaitu daftar kemampuan yang dianggap sudah Anda kuasai sebelum masuk M1. M1 adalah Masterclass GIS Modern, yang skripnya dibahas mulai Bab 24.

Skrip 23.3: Proyek mini terpadu

Tujuan. Menjalankan satu alur dari kiriman lapangan sampai tabel ringkasan per KPH, lalu menyimpan hasilnya ke GeoPackage dan proyek QGIS.

Bahan. Seluruh paket `paket-i4`. Berkas `b7_01_proyek_terpadu.py` (di dalamnya tertulis "Skrip 7.1"). Ganti `DATA` di baris 10 dan `HASIL` di baris 11. Skrip menghasilkan lima berkas di folder `HASIL`: `Proyek_Terpadu.qgz`, `Hasil_Terpadu.gpkg`, `ringkasan_kph.csv`, `log_kiriman.csv`, serta `NDVI_KPH.tif` dan `CHM.tif`. **Jalankan di proyek kosong.** Baris 17-19 mengosongkan dan mengubah proyek yang terbuka.

Skrip ini 149 baris dan hampir tidak punya hal yang benar-benar baru. Ia memadukan Skrip 22.2 (data lapangan), Bab 21 (alat vektor dan raster), dan Skrip 22.4 (ekspresi). Karena itu penjelasannya dibagi delapan blok, dan setiap blok menyebut asal pola yang dipakai.

Ilustrasi 23.5: Alur skrip 23.3



Skrip utuh:

```python
# [SKRIP 7.1: Proyek mini terpadu I1-I3: dari kiriman lapangan sampai ringkasan per KPH]
# Penulis: Badar Mubarok Yogaswara
# Keluaran di folder HASIL: Proyek_Terpadu.qgz, Hasil_Terpadu.gpkg, ringkasan_kph.csv, log_kiriman.csv.
import csv
import glob
import os
import processing
from qgis.core import QgsProject, QgsVectorLayer, QgsRasterLayer, QgsCoordinateReferenceSystem

DATA = "D:/Latihan/paket-i4"
HASIL = "D:/Latihan/hasil_i4"
os.makedirs(HASIL, exist_ok=True)

CRS = "EPSG:32749"
SEMENTARA = "TEMPORARY_OUTPUT"
proyek = QgsProject.instance()
proyek.removeAllMapLayers()
proyek.setCrs(QgsCoordinateReferenceSystem(CRS))
proyek.setTitle("KPH Contoh - proyek terpadu")


# ---------------------------------------------------------------- bagian 0: pembantu
def muat_vektor(jalur, nama):
    lyr = QgsVectorLayer(jalur, nama, "ogr")
    if not lyr.isValid():
        raise ValueError("vektor tidak bisa dibuka: " + jalur)
    return lyr
```

```
def muat_raster(jalur, nama):
    lyr = QgsRasterLayer(jalur, nama, "gdal")
    if not lyr.isValid():
        raise ValueError("raster tidak bisa dibuka: " + jalur)
    return lyr


def jalankan(alat, **parameter):
    """Menjalankan alat Processing lalu mengembalikan isi kunci OUTPUT."""
    return processing.run(alat, parameter)["OUTPUT"]


def hitung_kolom(layer, nama, tipe, rumus):
    """Menambah satu kolom dengan ekspresi. tipe: 0 = desimal, 1 = bilangan bulat."""
    return jalankan("native:fieldcalculator", INPUT=layer, FIELD_NAME=nama, FIELD_TYPE=tipe,
                    FIELD_LENGTH=10, FIELD_PRECISION=2 if tipe == 0 else 0, FORMULA=rumus,
                    OUTPUT=SEMENTARA)


batas = muat_vektor(DATA + "/Batas_KPH.gpkg", "Batas_KPH")
sungai = muat_vektor(DATA + "/Sungai.gpkg", "Sungai")
hotspot = muat_vektor(DATA + "/Hotspot_Nasional.gpkg", "Hotspot_Nasional")
citra = muat_raster(DATA + "/Citra_KPH.tif", "Citra_KPH")
dsm = muat_raster(DATA + "/DSM_Drone.tif", "DSM")
dtm = muat_raster(DATA + "/DTM_Drone.tif", "DTM")
for lyr in (batas, sungai, hotspot, citra):
    proyek.addMapLayer(lyr)

# ---------------------------------------------------------------- bagian I3: data lapangan
tujuan_crs = QgsCoordinateReferenceSystem(CRS)
diterima, log = [], []
for jalur in sorted(glob.glob(DATA + "/Kiriman_Lapangan/*.gpkg")):
    nama = os.path.basename(jalur)
    try:
        lyr = muat_vektor(jalur, nama)
        kurang = [f for f in ("Kondisi", "Tinggi_Phn", "Foto") if lyr.fields().indexOf(f) < 0]
        if kurang:
            raise ValueError("field hilang: " + ", ".join(kurang))
        if lyr.crs().authid() != CRS:
            lyr = jalankan("native:reprojectlayer", INPUT=lyr, TARGET_CRS=tujuan_crs, OUTPUT=SEMENTARA)
        diterima.append(lyr)
        log.append((nama, "DITERIMA"))
    except Exception as e:
        log.append((nama, "DITOLAK: %s" % e))

gabung = jalankan("native:mergevectorlayers", LAYERS=diterima, CRS=tujuan_crs, OUTPUT=SEMENTARA)
# Tiap berkas membawa nomor fid sendiri, jadi setelah digabung ada fid kembar. Buang kolom itu;
# GeoPackage akan memberi nomor baru yang unik saat disimpan.
survei = jalankan("native:deletecolumn", INPUT=gabung, COLUMN=["fid"], OUTPUT=SEMENTARA)
survei.setName("Titik_Survei_Gabungan")
proyek.addMapLayer(survei)
with open(HASIL + "/log_kiriman.csv", "w", newline="", encoding="utf-8") as f:
    csv.writer(f).writerows([("berkas", "status")] + log)
ditolak = sum(1 for _, s in log if s != "DITERIMA")
print("Kiriman:", len(log) - ditolak, "diterima,", ditolak, "ditolak;",
      "titik gabungan =", survei.featureCount())

# ---------------------------------------------------------------- bagian I1: analisis vektor
sempadan = jalankan("native:buffer", INPUT=sungai, DISTANCE=30, DISSOLVE=True, OUTPUT=SEMENTARA)
sempadan.setName("Sempadan_30m")
proyek.addMapLayer(sempadan)
irisan = jalankan("native:intersection", INPUT=batas, OVERLAY=sempadan, INPUT_FIELDS=["NAMA_KPH"],
                  OVERLAY_FIELDS=[], OUTPUT=SEMENTARA)
sempadan_ha = {}
for f in irisan.getFeatures():
    sempadan_ha[f["NAMA_KPH"]] = sempadan_ha.get(f["NAMA_KPH"], 0) + f.geometry().area() / 10000

# Kolom baru lewat ekspresi: luas, titik panas, titik survei (agregasi dari layer lain)
```

```
cacah = "aggregate('%s', 'count', \"%s\", intersects($geometry, geometry(@parent)))"
ringkas = hitung_kolom(batas, "LUAS_HA", 0, "round($area / 10000, 2)")
ringkas = hitung_kolom(ringkas, "HOTSPOT", 1, cacah % ("Hotspot_Nasional", "ID_Titik"))
ringkas = hitung_kolom(ringkas, "TITIK_SURVEI", 1, cacah % ("Titik_Survei_Gabungan", "Foto"))
ringkas.setName("Ringkasan_KPH")
proyek.addMapLayer(ringkas)

# ---------------------------------------------------------------- bagian I2: analisis raster
ndvi_jalur = jalankan("native:rastercalc", LAYERS=[citra], OUTPUT=HASIL + "/NDVI_KPH.tif",
                      EXPRESSION='("Citra_KPH@4" - "Citra_KPH@1") / ("Citra_KPH@4" + "Citra_KPH@1")')
chm_jalur = jalankan("native:rastercalc", LAYERS=[dsm, dtm], OUTPUT=HASIL + "/CHM.tif",
                     EXPRESSION='"DSM@1" - "DTM@1"')
ndvi = muat_raster(ndvi_jalur, "NDVI_KPH")
chm = muat_raster(chm_jalur, "CHM")
proyek.addMapLayer(ndvi)
proyek.addMapLayer(chm)


def rata(raster, awalan):
    z = jalankan("native:zonalstatisticsfb", INPUT=batas, INPUT_RASTER=raster, RASTER_BAND=1,
                 COLUMN_PREFIX=awalan, STATISTICS=[2], OUTPUT=SEMENTARA)
    return {f["NAMA_KPH"]: f[awalan + "mean"] for f in z.getFeatures()}


ndvi_rata, chm_rata = rata(ndvi, "n_"), rata(chm, "c_")

# ---------------------------------------------------------------- bagian akhir: tabel dan proyek
judul = ["NAMA_KPH", "LUAS_HA", "SEMPADAN_HA", "HOTSPOT", "TITIK_SURVEI", "NDVI_RATA", "CHM_RATA_M"]
baris = []
for f in ringkas.getFeatures():
    n = f["NAMA_KPH"]
    baris.append([n, f["LUAS_HA"], round(sempadan_ha.get(n, 0), 2), f["HOTSPOT"], f["TITIK_SURVEI"],
                  round(ndvi_rata[n], 3), round(chm_rata[n], 2)])
with open(HASIL + "/ringkasan_kph.csv", "w", newline="", encoding="utf-8") as f:
    w = csv.writer(f)
    w.writerow(judul)
    w.writerows(baris)
print("%-10s %7s %12s %8s %13s %10s %11s" % tuple(judul))
for b in baris:
    print("%-10s %7.2f %12.2f %8d %13d %10.3f %11.2f" % tuple(b))

# Layer sementara hilang saat QGIS ditutup. Simpan yang penting ke satu GeoPackage,
# lalu ganti layer sementara di proyek dengan layer dari GeoPackage itu.
paket = HASIL + "/Hasil_Terpadu.gpkg"
processing.run("native:package", {"LAYERS": [survei, sempadan, ringkas], "OUTPUT": paket,
                                  "OVERWRITE": True})
for lyr in (survei, sempadan, ringkas):
    nama = lyr.name()
    proyek.removeMapLayer(lyr.id())
    proyek.addMapLayer(muat_vektor(paket + "|layername=" + nama, nama))
print("Proyek tersimpan?", proyek.write(HASIL + "/Proyek_Terpadu.qgz"))
print("Isi proyek:", sorted(l.name() for l in proyek.mapLayers().values()))
```

Penjelasan baris demi baris. Nomor baris mengikuti berkas di atas.

Blok 0: menyiapkan proyek (baris 1-20).

Baris	Tulisan	Artinya
1-3	komentar	Judul, penulis, dan daftar keluaran.
4-8	`import ...` dan `from qgis.core import ...`	Perkakas yang sudah Anda kenal: `csv`, `glob`, `os`, `processing`, serta layer vektor, layer raster, dan CRS.
10-12	`DATA`, `HASIL`, `os.makedirs(...)`	Dua alamat dan pembuat folder hasil.
14-15	`CRS` dan `SEMENTARA`	Dua nilai yang dipakai berulang, ditulis sekali di atas. Ganti di satu tempat, berlaku di semua tempat.
16-19	`proyek = QgsProject.instance()` dan tiga baris sesudahnya	Ambil proyek terbuka, **kosongkan semua layernya** (`removeAllMapLayers`), tetapkan CRS, dan beri judul.

Blok 1: empat pembantu (baris 22-47). Pola dari Skrip 22.1 dan 22.3 dipakai lagi dalam bentuk fungsi.

Baris	Tulisan	Artinya
23-27	`def muat_vektor(jalur, nama):`	Buka layer vektor. Bila tidak sah, `raise ValueError`.
30-34	`def muat_raster(jalur, nama):`	Sama untuk raster, dengan penyedia data `"gdal"`.
37-39	`def jalankan(alat, parameter):`	Tanda `parameter` mengumpulkan semua isian bernama menjadi satu kamus (Bab 7). Fungsi lalu menjalankan alat dan langsung mengembalikan isi `["OUTPUT"]`. Hasilnya, pemanggilan jadi pendek: `jalankan("native:buffer", INPUT=..., DISTANCE=30)`.
42-46	`def hitung_kolom(layer, nama, tipe, rumus):`	Tambah satu kolom dengan Field calculator. `tipe` 0 berarti desimal dan 1 bulat. `FIELD_PRECISION=2 if tipe == 0 else 0` memilih dua desimal untuk desimal, nol untuk bulat.

Blok 2: memuat enam layer (baris 49-56).

Baris	Tulisan	Artinya
49-54	enam pemanggilan `muat_...`	Buka tiga layer vektor (batas, sungai, titik panas) dan tiga raster (citra, DSM, DTM). Layer yang gagal dibuka menghentikan skrip dengan pesan jelas.
55-56	`for lyr in (batas, sungai, hotspot, citra):`	Hanya empat yang dimasukkan ke proyek. DSM dan DTM dipakai sebagai bahan hitung, tidak ditampilkan.

Blok 3: data lapangan (baris 58-85). Ini Skrip 22.2 yang diperpendek.

Baris	Tulisan	Artinya
59-60	`tujuan_crs = ...` dan `diterima, log = [], []`	Siapkan CRS tujuan dan dua daftar kosong sekaligus dalam satu baris.
61-73	`for jalur in sorted(glob.glob(...)):`	Untuk tiap kiriman: buka dengan `muat_vektor`, periksa tiga kolom wajib, ubah CRS bila beda, lalu catat `DITERIMA` atau `DITOLAK`. Perbedaan dari Skrip 22.2: tidak ada pemeriksaan titik di luar KPH.
75	`gabung = jalankan("native:mergevectorlayers", ...)`	Gabungkan semua kiriman yang diterima.
76-78	komentar dan `native:deletecolumn`	Tiap berkas membawa nomor `fid` sendiri, jadi setelah digabung ada `fid` kembar. Kolom itu dibuang, dan GeoPackage akan memberi nomor baru yang unik saat menyimpan.
79-80	`survei.setName(...)` lalu `addMapLayer`	Beri nama `Titik_Survei_Gabungan` dan tampilkan. Nama ini dipakai rumus di baris 101.
81-82	`with open(... "log_kiriman.csv" ...)`	Tulis log. `[("berkas", "status")] + log` menyambung satu baris judul di depan daftar `log`.
83	`ditolak = sum(1 for _, s in log if s != "DITERIMA")`	Hitung catatan yang bukan `DITERIMA`. Garis bawah `_` adalah nama kotak untuk nilai yang tidak dipakai.
84-85	`print("Kiriman:", ...)`	Cetak ringkasan diterima dan ditolak.

Blok 4: analisis vektor (baris 87-103). Pola dari Bab 21 dan Skrip 22.4.

Baris	Tulisan	Artinya
88-90	`sempadan = jalankan("native:buffer", ...)`	Sempadan 30 meter di kiri-kanan sungai, digabung jadi satu (`DISSOLVE=True`). Beri nama `Sempadan_30m` dan tampilkan.
91-92	`irisan = jalankan("native:intersection", ...)`	Iris batas KPH dengan sempadan. `INPUT_FIELDS=["NAMA_KPH"]` membawa kolom nama KPH saja, dan `OVERLAY_FIELDS=[]` tidak membawa kolom dari sempadan.
93-95	`sempadan_ha = {}` lalu perulangan	Jumlahkan luas irisan per KPH (dalam hektare). `sempadan_ha.get(nama, 0)` mengambil jumlah sementara, atau 0 bila KPH belum tercatat. Perlu dijumlah karena satu KPH bisa punya beberapa potongan.
98	`cacah = "aggregate('%s', 'count', \"%s\", intersects(...))"`	Cetakan rumus agregasi. Dua tanda `%s` nanti diisi nama layer dan nama kolom.
99-101	tiga `hitung_kolom(...)` berantai	Tambah kolom `LUAS_HA`, `HOTSPOT`, dan `TITIK_SURVEI`. Hasil tiap langkah menjadi masukan langkah berikutnya, dan `cacah % (...)` mengisi cetakan rumusnya.
102-103	`ringkas.setName(...)` lalu `addMapLayer`	Beri nama `Ringkasan_KPH` dan tampilkan.

Blok 5: analisis raster (baris 105-122). Pola dari Skrip 21.3.

Baris	Tulisan	Artinya
106-107	`ndvi_jalur = jalankan("native:rastercalc", ...)`	Hitung NDVI ke berkas `NDVI_KPH.tif`. Rumusnya `(NIR - Merah) / (NIR + Merah)`. Penulisan `"Citra_KPH@4"` berarti band nomor 4 dari layer `Citra_KPH`, yaitu inframerah dekat. Band 1 adalah merah.
108-109	`chm_jalur = jalankan("native:rastercalc", ...)`	Hitung tinggi tajuk (CHM) sebagai DSM dikurangi DTM, ke berkas `CHM.tif`.
110-113	`muat_raster(...)` dan `addMapLayer`	Buka kedua hasil dan tampilkan. Alat raster mengembalikan jalur teks, bukan layer.
116-119	`def rata(raster, awalan):`	Hitung rata-rata raster di dalam tiap KPH dengan `native:zonalstatisticsfb`. `STATISTICS=[2]` berarti rata-rata (*mean*). Kamus hasilnya memasangkan nama KPH dengan rata-ratanya. Kolom hasil bernama `awalan` ditambah `mean`.
122	`ndvi_rata, chm_rata = rata(ndvi, "n_"), rata(chm, "c_")`	Dua pemanggilan dalam satu baris, hasilnya masuk ke dua kotak.

Blok 6: tabel ringkasan (baris 124-137).

Baris	Tulisan	Artinya
125-130	`judul = [...]` dan `baris = []` lalu perulangan	Susun satu baris per KPH: nama, luas, sempadan, titik panas, titik survei, NDVI, dan tinggi tajuk, yang dibulatkan.
131-134	`with open(... "ringkasan_kph.csv" ...)`	Tulis ke CSV: baris judul, lalu semua baris.
135-137	`print("%-10s ..." % tuple(judul))`	Cetak tabel di layar. `tuple(...)` mengubah daftar menjadi tuple, karena operator `%` memintanya. Format `%7.2f` memberi tujuh kolom dengan dua desimal.

Blok 7: kemas dan simpan (baris 139-149).

Baris	Tulisan	Artinya
141-143	`processing.run("native:package", {...})`	Tulis tiga layer sementara ke satu berkas GeoPackage. `OVERWRITE: True` menimpa berkas lama.
144-147	`for lyr in (survei, sempadan, ringkas):`	Untuk tiap layer sementara: hapus dari proyek, lalu tambahkan layer yang sama dari GeoPackage. Alamatnya dibentuk dari jalur berkas, tanda garis tegak, `layername=`, dan nama tabel.
148	`proyek.write(HASIL + "/Proyek_Terpadu.qgz")`	Simpan proyek. Hasilnya `True` bila berhasil.
149	`print("Isi proyek:", sorted(...))`	Cetak daftar nama layer di proyek, urut abjad.

Hasil. Layar menampilkan (berdasarkan uji pada QGIS 4.0.2):

```
HASIL
Kiriman: 4 diterima, 2 ditolak; titik gabungan = 18
NAMA_KPH   LUAS_HA  SEMPADAN_HA  HOTSPOT  TITIK_SURVEI  NDVI_RATA  CHM_RATA_M
KPH Alpha     4.00         1.18        2             6      0.604        2.60
KPH Beta      4.00         1.39        5             4      0.431        2.69
KPH Gamma     8.00         1.50        5             7      0.329        2.56
Proyek tersimpan? True
Isi proyek: ['Batas_KPH', 'CHM', 'Citra_KPH', 'Hotspot_Nasional', 'NDVI_KPH', 'Ringkasan_KPH', 'Sempadan_30m',
'Sungai', 'Titik_Survei_Gabungan']
```

Satu tabel ini menjawab semua pertanyaan Kepala Seksi. KPH Gamma terluas (8 ha). KPH Beta punya sempadan terbesar terhadap luasnya, dan titik panas Beta dan Gamma masing-masing 5. NDVI turun dari Alpha (0,604) ke Gamma (0,329). Nilai itu sama dengan keterangan di `README.txt` paket data.

Ada satu angka yang layak dicermati. `TITIK_SURVEI` berjumlah 6 + 4 + 7 = 17, padahal `titik gabungan = 18`. Selisih satu itu adalah titik yang jatuh di luar semua KPH. Ia sama dengan titik curiga `b_05.jpg` di Skrip 22.2. Jadi agregasi per KPH tidak menghitungnya, dan ini benar.

Bagian yang boleh diubah.

- **Baris 10-11, jalur.** Sesuaikan dengan komputer Anda.
- **Baris 14, CRS.** Ganti sesuai wilayah Anda. Akibatnya, semua kiriman diubah ke CRS itu.
- **Baris 65, kolom wajib.** `("Kondisi", "Tinggi_Phn", "Foto")` boleh diganti kolom lain.
- Baris 88-89, jarak 30 dan nama `Sempadan_30m`. Ubah keduanya bersamaan. Akibatnya, kolom `SEMPADAN_HA` berubah.
- **Baris 99-101, kolom ringkasan.** Tambah kolom baru dengan satu panggilan `hitung_kolom` lagi.
- **Baris 106-109, rumus raster.** Ganti nomor band bila citra Anda berbeda.

Jebakan.

- **Proyek Anda dikosongkan.** Baris 17 menghapus semua layer dari proyek yang terbuka, dan baris 18-19 mengubah CRS serta judulnya. Baris 148 menyimpan proyek ke berkas baru. Jalankan di proyek kosong [CEK: perilaku jendela QGIS setelah penyimpanan tidak dapat diuji dari sini].
- `fid` kembar. Pada uji, baris 78 diganti `survei = gabung`. Skrip berjalan sampai tabel tercetak, lalu berhenti di baris 142 dengan `QgsProcessingException: Error obtained while packaging one or more layers.` Di layar juga muncul `ERROR 1: failed to execute insert : UNIQUE constraint failed: Titik_Survei_Gabungan.fid`. Itu sebabnya kolom `fid` dibuang.
- **Nama layer di rumus harus sama.** Pada uji, `"Hotspot_Nasional"` di baris 100 diganti `"Hotspot"`. Skrip berhenti di baris 100 dengan `Cannot find layer with name or ID 'Hotspot'`. Berbeda dengan Skrip 22.4, galat ini menghentikan skrip.
- **Layer sementara hilang.** Layer hasil alat bersifat sementara. Itu sebabnya baris 139-147 memindahkannya ke GeoPackage. Bila bagian itu dihapus, proyek yang tersimpan menunjuk layer yang tidak ada lagi.
- **Dua cara menghitung luas.** Kolom `LUAS_HA` memakai `$area` (mengikuti ellipsoid proyek), sedangkan `SEMPADAN_HA` memakai `geometry().area()` (bidang datar). Bedanya sangat kecil, dan dibahas pada Skrip 22.4.

Latihan variasi.

1. Ubah lebar sempadan menjadi 50 meter.
2. Cetak tambahan: sempadan sebagai persen dari luas KPH.

Contoh jawaban nomor 1 (sudah diuji): ubah baris 88 dan 89 menjadi

```
PYTHON
sempadan = jalankan("native:buffer", INPUT=sungai, DISTANCE=50, DISSOLVE=True, OUTPUT=SEMENTARA)
sempadan.setName("Sempadan_50m")
```

Kolom `SEMPADAN_HA` berubah menjadi 1,69, 2,32, dan 2,54. Angka ini sesuai dengan hasil model di Skrip 23.2 untuk 50 meter (jumlah 6,55 ha).

Contoh jawaban nomor 2 (sudah diuji): tambahkan di akhir skrip.

```
PYTHON
for b in baris:
    print(b[0], "sempadan = %.1f persen dari luas" % (100 * b[2] / b[1]))
```

Layar menambah `KPH Alpha sempadan = 29.5 persen dari luas`, `KPH Beta sempadan = 34.8 persen dari luas`, dan `KPH Gamma sempadan = 18.8 persen dari luas`.

> **Pola yang bisa dipakai ulang:** Kerangka proyek terpadu: siapkan proyek dan pembantu (`muat_vektor`, `muat_raster`, `jalankan`, `hitung_kolom`), muat bahan, jalankan tahap demi tahap dengan hasil tiap tahap menjadi masukan tahap berikutnya, susun tabel ringkasan, lalu kemas ke GeoPackage dan simpan proyek. Ganti alat dan nama layer. Pembantunya tetap.

Skrip 23.4: Ringkasan sebagai tugas latar belakang

Skrip 23.3 membuat QGIS terasa macet selama bekerja. Skrip ini mengerjakan ringkasan yang sama sebagai **tugas latar belakang**, dengan pola `QgsTask` dari Bab 21 (Skrip 21.6). Karena itu hanya bagian yang baru dijelaskan.

Tujuan. Menghitung ringkasan per KPH di latar belakang, melaporkan kemajuan, lalu menampilkan hasilnya saat tugas selesai.

Bahan. Paket `paket-i4`: batas, sungai, titik panas, citra, DSM, dan DTM. Berkas `b7_02_tugas_ringkasan.py` (di dalamnya tertulis "Skrip 7.2"). Ganti `DATA` di baris 10 dan `HASIL` di baris 11. Skrip ini paling pas dijalankan di Editor Skrip QGIS yang terbuka, karena tugas latar belakang butuh QGIS yang hidup. Hasilnya: `NDVI_KPH.tif`, `CHM.tif`, dan `ringkasan_latar.csv`.

Ilustrasi 23.6: Dua jalur kerja Skrip 23.4



Skrip utuh:

```python
# [SKRIP 7.2: Ringkasan per KPH sebagai tugas latar belakang (QgsTask)]
# Penulis: Badar Mubarok Yogaswara
# Pola: pekerjaan berat di fungsi tugas (latar belakang); proyek dan tabel di fungsi selesai (utama).
import csv
import os
import processing
from qgis.core import (QgsApplication, QgsTask, QgsMessageLog, Qgis, QgsProject,
                       QgsVectorLayer, QgsRasterLayer)

DATA = "D:/Latihan/paket-i4"
```

```
HASIL = "D:/Latihan/hasil_i4"
os.makedirs(HASIL, exist_ok=True)
SEMENTARA = "TEMPORARY_OUTPUT"


def jalankan(alat, **parameter):
    return processing.run(alat, parameter)["OUTPUT"]


def hitung_ringkasan(task, data, hasil):
    """Latar belakang. Hanya membaca berkas dan menjalankan alat; tidak menyentuh QgsProject."""
    batas = QgsVectorLayer(data + "/Batas_KPH.gpkg", "Batas_KPH", "ogr")
    sungai = QgsVectorLayer(data + "/Sungai.gpkg", "Sungai", "ogr")
    hotspot = QgsVectorLayer(data + "/Hotspot_Nasional.gpkg", "Hotspot_Nasional", "ogr")
    citra = QgsRasterLayer(data + "/Citra_KPH.tif", "Citra_KPH", "gdal")
    dsm = QgsRasterLayer(data + "/DSM_Drone.tif", "DSM", "gdal")
    dtm = QgsRasterLayer(data + "/DTM_Drone.tif", "DTM", "gdal")
    for lyr in (batas, sungai, hotspot, citra, dsm, dtm):
        if not lyr.isValid():
            raise ValueError("Layer tidak terbaca: " + lyr.name())

    # tahap 1 (0-25%): sempadan sungai
    sem = jalankan("native:buffer", INPUT=sungai, DISTANCE=30, DISSOLVE=True, OUTPUT=SEMENTARA)
    iris = jalankan("native:intersection", INPUT=batas, OVERLAY=sem, INPUT_FIELDS=["NAMA_KPH"],
                    OVERLAY_FIELDS=[], OUTPUT=SEMENTARA)
    sem_ha = {}
    for f in iris.getFeatures():
        sem_ha[f["NAMA_KPH"]] = sem_ha.get(f["NAMA_KPH"], 0) + f.geometry().area() / 10000
    task.setProgress(25)
    if task.isCanceled():
        return None

    # tahap 2 (25-50%): titik panas
    hit = jalankan("native:countpointsinpolygon", POLYGONS=batas, POINTS=hotspot, FIELD="HOTSPOT",
                   OUTPUT=SEMENTARA)
    titik = {f["NAMA_KPH"]: f["HOTSPOT"] for f in hit.getFeatures()}
    task.setProgress(50)
    if task.isCanceled():
        return None

    # tahap 3 (50-100%): raster
    ndvi = jalankan("native:rastercalc", LAYERS=[citra], OUTPUT=hasil + "/NDVI_KPH.tif",
                    EXPRESSION='("Citra_KPH@4" - "Citra_KPH@1") / ("Citra_KPH@4" + "Citra_KPH@1")')
    chm = jalankan("native:rastercalc", LAYERS=[dsm, dtm], OUTPUT=hasil + "/CHM.tif",
                   EXPRESSION='"DSM@1" - "DTM@1"')
    task.setProgress(75)

    def rata(jalur, awalan):
        raster = QgsRasterLayer(jalur, "r", "gdal")
        z = jalankan("native:zonalstatisticsfb", INPUT=batas, INPUT_RASTER=raster, RASTER_BAND=1,
                     COLUMN_PREFIX=awalan, STATISTICS=[2], OUTPUT=SEMENTARA)
        return {f["NAMA_KPH"]: f[awalan + "mean"] for f in z.getFeatures()}

    n, c = rata(ndvi, "n_"), rata(chm, "c_")
    task.setProgress(100)
    hasil_baris = []
    for f in batas.getFeatures():
        k = f["NAMA_KPH"]
        hasil_baris.append([k, round(f.geometry().area() / 10000, 2), round(sem_ha.get(k, 0), 2),
                            titik[k], round(n[k], 3), round(c[k], 2)])
    return hasil_baris


def selesai(exception, baris=None):
    """Program utama. Aman menyentuh proyek, tabel, dan tampilan."""
    if exception is not None:
        QgsMessageLog.logMessage("Ringkasan gagal: %s" % exception, "Ringkasan KPH", Qgis.Critical)
        return
```

```
    if baris is None:
        QgsMessageLog.logMessage("Ringkasan dibatalkan", "Ringkasan KPH", Qgis.Warning)
        return
    with open(HASIL + "/ringkasan_latar.csv", "w", newline="", encoding="utf-8") as f:
        w = csv.writer(f)
        w.writerow(["NAMA_KPH", "LUAS_HA", "SEMPADAN_HA", "HOTSPOT", "NDVI_RATA", "CHM_RATA_M"])
        w.writerows(baris)
    for nama in ("NDVI_KPH", "CHM"):
        QgsProject.instance().addMapLayer(QgsRasterLayer(HASIL + "/" + nama + ".tif", nama, "gdal"))
    for b in baris:
        print("%-10s luas %5.2f ha | sempadan %4.2f ha | titik panas %d | NDVI %.3f | CHM %.2f m"
              % tuple(b))
    QgsMessageLog.logMessage("Ringkasan selesai: %d KPH" % len(baris), "Ringkasan KPH", Qgis.Success)


tugas = QgsTask.fromFunction("Ringkasan per KPH", hitung_ringkasan, on_finished=selesai,
                             data=DATA, hasil=HASIL)
QgsApplication.taskManager().addTask(tugas)
print("Tugas berjalan di latar belakang; QGIS tetap bisa dipakai.")
```

Penjelasan bagian yang baru. Nomor baris mengikuti berkas di atas.

Baris	Tulisan	Artinya
4-13	impor, `DATA`, `HASIL`, `SEMENTARA`	Sama seperti Skrip 23.3, ditambah `QgsApplication`, `QgsTask`, dan `QgsMessageLog` untuk tugas dan catatan.
16-17	`def jalankan(alat, **parameter):`	Pembantu yang sama seperti di Skrip 23.3.
20	`def hitung_ringkasan(task, data, hasil):`	Fungsi tugas. Isian pertama, `task`, dikirim QGIS otomatis. Dua isian lainnya berasal dari baris 94-95.
22-27	enam pemanggilan `QgsVectorLayer` dan `QgsRasterLayer`	Layer dibuka **di dalam** fungsi tugas, bukan diambil dari proyek. Fungsi tugas tidak boleh menyentuh proyek (komentar di baris 21).
28-30	`for lyr in (...):` lalu `if not lyr.isValid():` dan `raise ValueError(...)`	Pengaman. Tiap layer dicek satu per satu. Bila ada yang tidak terbaca, tugas berhenti dengan pesan `Layer tidak terbaca:` dan nama layernya. `isValid()` dan `raise ValueError` dibahas di Bab 10.
32-41	tahap 1: sempadan	Buffer dan irisan seperti Skrip 23.3. `task.setProgress(25)` melaporkan kemajuan 25 persen. `if task.isCanceled(): return None` berhenti bila pengguna membatalkan tugas.
43-49	tahap 2: titik panas	Alat `native:countpointsinpolygon` menambah kolom `HOTSPOT`, yaitu jumlah titik di tiap KPH. Hasilnya dijadikan kamus nama dan jumlah. Lalu kemajuan 50 persen dan pemeriksaan pembatalan lagi.
51-56	tahap 3a: NDVI dan CHM	Dua `native:rastercalc` seperti Skrip 23.3. Kemajuan 75 persen.
58-62	`def rata(jalur, awalan):`	Fungsi di dalam fungsi, karena hanya dipakai di sini. Ia membuka raster hasil dan menghitung rata-rata per KPH.
64-65	`n, c = rata(...), rata(...)` lalu `task.setProgress(100)`	Hitung dua rata-rata, lalu lapor kemajuan penuh.

Baris	Tulisan	Artinya
66-71	`hasil_baris = []` lalu `return hasil_baris`	Susun satu daftar berisi satu baris per KPH, lalu **kembalikan**. Nilai kembalian ini yang diterima fungsi selesai. Luas KPH dihitung langsung dari geometri (`f.geometry().area()`).
74	`def selesai(exception, baris=None):`	Fungsi selesai. `exception` berisi galat bila tugas gagal. `baris` berisi daftar dari `return`.
76-81	`if exception is not None:` lalu `if baris is None:`	Tiga jalur: gagal (catat `Qgis.Critical`), dibatalkan (catat `Qgis.Warning`), atau sukses.
82-85	`with open(... "ringkasan_latar.csv" ...)`	Tulis CSV. Berbeda dari Skrip 23.3, tidak ada kolom `TITIK_SURVEI`, karena versi ini tidak menggabung kiriman lapangan.
86-87	`for nama in ("NDVI_KPH", "CHM"):`	Tambahkan dua raster hasil ke proyek. Ini aman di sini karena fungsi selesai berjalan di program utama.
88-91	`for b in baris: print(...)`	Cetak satu baris per KPH, lalu catat `Qgis.Success` di log.
94-95	`tugas = QgsTask.fromFunction("Ringkasan per KPH", hitung_ringkasan, on_finished=selesai, data=DATA, hasil=HASIL)`	Bungkus fungsi tugas menjadi tugas. Dua isian bernama di ujung diteruskan ke `hitung_ringkasan`.
96-97	`addTask(tugas)` lalu `print(...)`	Serahkan tugas ke manajer tugas QGIS. Pesan di baris 97 muncul **sebelum** hasil, karena skrip tidak menunggu.

Hasil. Layar menampilkan (berdasarkan uji pada QGIS 4.0.2):

```
HASIL
Tugas berjalan di latar belakang; QGIS tetap bisa dipakai.
KPH Alpha  luas  4.00 ha | sempadan 1.18 ha | titik panas 2 | NDVI 0.604 | CHM 2.60 m
KPH Beta   luas  4.00 ha | sempadan 1.39 ha | titik panas 5 | NDVI 0.431 | CHM 2.69 m
KPH Gamma  luas  8.00 ha | sempadan 1.50 ha | titik panas 5 | NDVI 0.329 | CHM 2.56 m
```

Baris pertama muncul langsung. Tiga baris sisanya muncul belakangan, setelah tugas selesai. Angkanya sama dengan Skrip 23.3: sempadan 1,18, 1,39, dan 1,50 ha; titik panas 2, 5, dan 5; NDVI 0,604, 0,431, dan 0,329; tinggi tajuk 2,60, 2,69, dan 2,56 meter. Dua cara berbeda, jawaban sama. Itulah tanda yang baik. Kemajuan tugas tampil di bilah status QGIS dan pesan lognya di panel Log Messages dengan tag `Ringkasan KPH` [CEK: tampilan visualnya tidak dapat diuji tanpa antarmuka QGIS].

Bagian yang boleh diubah.

- **Baris 10-11, jalur.** Sesuaikan dengan komputer Anda.
- **Baris 33, jarak 30.** Ubah lebar sempadan. Akibatnya, kolom `sempadan` berubah.
- **Baris 39, 47, 56, dan 65, angka kemajuan.** Atur pembagian bilah kemajuan sesuka Anda.
- Baris 77, 80, dan 91, tag log `"Ringkasan KPH"`. Ganti dengan nama yang Anda suka.
- **Baris 94, judul tugas.** Judul ini tampil di daftar tugas QGIS.

Jebakan.

- **Galat di latar belakang tidak terlihat di layar.** Pada uji, `data=DATA` di baris 95 diubah menjadi `data=DATA + "/salah"`. Layar hanya menampilkan `Tugas berjalan di latar belakang; QGIS tetap bisa dipakai.` dan tidak ada pesan lain. Galatnya hanya masuk ke log, dengan teks `Ringkasan gagal: Layer tidak terbaca: Batas_KPH`, sebab pengaman di baris 28-30 menangkap layer yang gagal dibuka (diverifikasi dengan salinan skrip yang mencetak teks log). Bila pengaman itu dihapus, teksnya `Ringkasan gagal: Could not create memory layer.` Bila hasil tidak muncul, buka panel Log Messages.
- **Baris pengaman (28-30) jangan dihapus.** Layer yang gagal dibuka tidak menimbulkan galat sendiri. Tanpa pengaman, tugas tidak berhenti dan mencetak tinggi tajuk yang ngawur: pada uji tercetak `CHM 0.00 m` atau `CHM inf m`, dan nilainya bisa berbeda tiap kali dijalankan. Pada uji, nama berkas DSM di baris 26 diganti `DSM_Hilang.tif`. Dengan pengaman, tugas berhenti, tidak ada CSV, dan log memuat `Ringkasan gagal: Layer tidak terbaca: DSM`. Seperti galat lain di latar belakang, pesannya hanya terlihat di panel Log Messages. Saat menambah layer baru, tambahkan juga ke daftar di baris 28.
- **Jangan menyentuh proyek di fungsi tugas.** Menambah layer dari latar belakang bisa membuat QGIS macet atau tertutup [CEK: tidak diuji di antarmuka QGIS].
- **Dijalankan di luar jendela QGIS.** Bila skrip dijalankan tanpa antarmuka, tugas harus ditunggu dengan perulangan, seperti dibahas pada Skrip 21.6. Pada uji, perulangan itu dipakai.

Latihan variasi.

1. Ubah sempadan menjadi 50 meter.
2. Bandingkan hasilnya dengan latihan serupa di Skrip 23.3.

Contoh jawaban (sudah diuji): ubah `DISTANCE=30` di baris 33 menjadi `DISTANCE=50`. Kolom `sempadan` menjadi 1,69, 2,32, dan 2,54 ha. Angka ini sama dengan hasil Skrip 23.3 yang diubah ke 50 meter, sehingga dua jalur berbeda memberi jawaban yang sama.

> **Pola yang bisa dipakai ulang:** Kerangka tugas ringkasan: `def kerja(task, ...)` membuka layernya sendiri, bekerja tahap demi tahap dengan `task.setProgress(...)` dan `task.isCanceled()`, lalu `return` daftar hasil. `def selesai(exception, hasil=None)` menangani tiga jalur: gagal, dibatalkan, sukses. Ganti isi tahap dan format laporan.

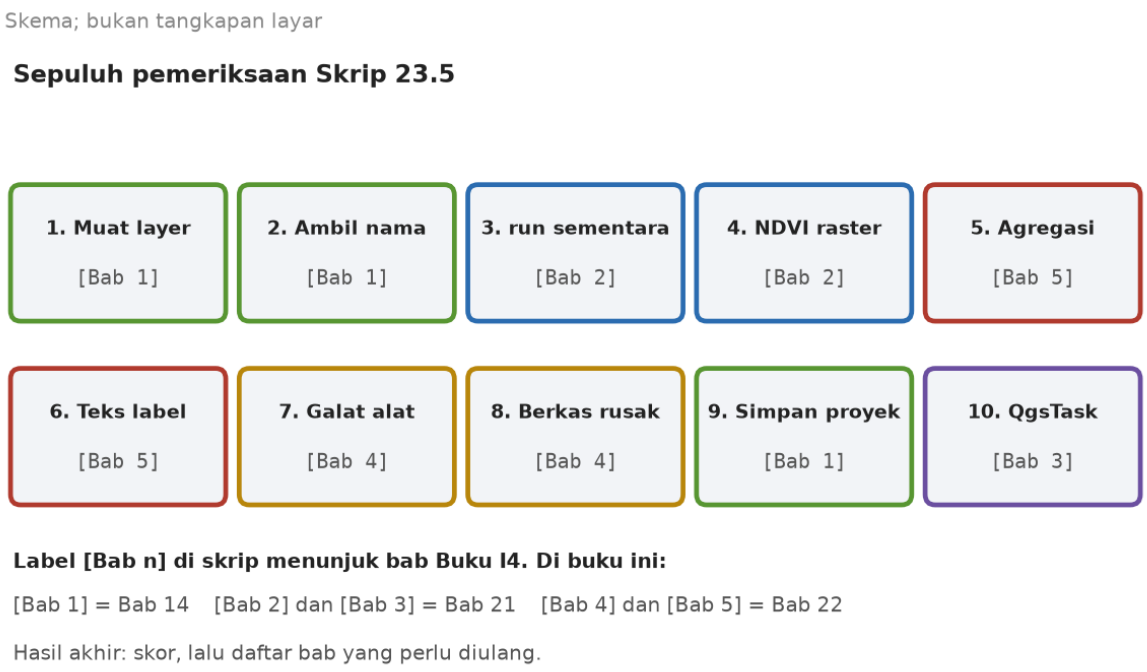
Skrip 23.5: Ceklis masuk M1

Skrip ini unik. Ia bukan pekerjaan analisis, melainkan **ujian untuk Anda**: sepuluh pemeriksaan otomatis yang mencetak LULUS atau GAGAL, lalu skor akhir. Bagian terpentingnya adalah cara tiap pemeriksaan ditulis, karena pola itu bisa Anda pakai untuk menguji skrip sendiri.

Tujuan. Menguji sepuluh keterampilan yang dipakai kelas M1, lalu menyebut bab mana yang perlu diulang.

Bahan. Paket `paket-i4`: `Batas_KPH.gpkg`, `Hotspot_Nasional.gpkg`, `Citra_KPH.tif`, dan `Kiriman_Lapangan/tim_e.gpkg`. Berkas `b7_03_ceklis_masuk_m1.py` (di dalamnya tertulis "Skrip 7.3"). Ganti `DATA` di baris 13 dan `HASIL` di baris 14. Keluarannya `uji_kesiapan.qgz`. **Jalankan di proyek kosong.** Baris 18 dan 122 mengosongkan proyek terbuka. Skor akhir muncul belakangan, karena pemeriksaan ke-10 berjalan di latar belakang.

Ilustrasi 23.7: Sepuluh pemeriksaan Skrip 23.5



Skrip utuh:

```python
# [SKRIP 7.3: Uji kesiapan otomatis: sepuluh keterampilan yang dipakai Masterclass Seri 1 (M1)]
# Penulis: Badar Mubarok Yogaswara
# Tiap pemeriksaan mencetak LULUS atau GAGAL beserta alasannya.
# Skor akhir muncul setelah tugas latar belakang (pemeriksaan 10) selesai.
import os
import re
import time
import processing
from qgis.core import (QgsApplication, QgsTask, QgsProject, QgsVectorLayer, QgsRasterLayer,
                       QgsExpression, QgsExpressionContext, QgsExpressionContextUtils,
                       QgsProcessingException)

DATA = "D:/Latihan/paket-i4"
HASIL = "D:/Latihan/hasil_i4"
os.makedirs(HASIL, exist_ok=True)

proyek = QgsProject.instance()
proyek.removeAllMapLayers()
catatan = []                                    # daftar (nama, lulus?, alasan)
NDVI = '("Citra_KPH@4" - "Citra_KPH@1") / ("Citra_KPH@4" + "Citra_KPH@1")'
CACAH = ("aggregate('Hotspot_Nasional', 'count', \"ID_Titik\", "
         "intersects($geometry, geometry(@parent)))")


def periksa(nama, fungsi):
```

```
    try:
        alasan = fungsi()
        catatan.append((nama, True, alasan or ""))
    except Exception as e:           # pemeriksaan yang gagal dicatat; skrip tidak berhenti
        catatan.append((nama, False, "%s: %s" % (type(e).__name__, e)))
    n, ok, alasan = catatan[-1]
    print("%-46s %s  %s" % (n, "LULUS" if ok else "GAGAL", alasan))


def tegaskan(syarat, pesan):
    if not syarat:
        raise AssertionError(pesan)


batas = QgsVectorLayer(DATA + "/Batas_KPH.gpkg", "Batas_KPH", "ogr")
hotspot = QgsVectorLayer(DATA + "/Hotspot_Nasional.gpkg", "Hotspot_Nasional", "ogr")
citra = QgsRasterLayer(DATA + "/Citra_KPH.tif", "Citra_KPH", "gdal")


def konteks(fitur):
    c = QgsExpressionContext()
    c.appendScopes(QgsExpressionContextUtils.globalProjectLayerScopes(batas))
    c.setFeature(fitur)
    return c


def k1():
    for lyr in (batas, hotspot, citra):
        tegaskan(lyr.isValid(), lyr.name() + " tidak sah")
        proyek.addMapLayer(lyr)
    tegaskan(len(proyek.mapLayers()) == 3, "jumlah layer tidak 3")


def k2():
    tegaskan(proyek.mapLayersByName("Batas_KPH")[0].featureCount() == 3, "Batas_KPH bukan 3 fitur")
    r = proyek.mapLayersByName("Citra_KPH")[0]
    tegaskan(r.bandCount() == 4 and r.rasterUnitsPerPixelX() == 2.0, "raster tidak sesuai")
    tegaskan(proyek.mapLayersByName("Tidak_Ada") == [], "nama tak ada harus memberi daftar kosong")
    return "band=%d, piksel=%.0f m" % (r.bandCount(), r.rasterUnitsPerPixelX())


def k3():
    v = processing.run("native:buffer", {"INPUT": batas, "DISTANCE": 10,
                                         "OUTPUT": "TEMPORARY_OUTPUT"})["OUTPUT"]
    tegaskan(isinstance(v, QgsVectorLayer) and v.featureCount() == 3, "bukan layer vektor")
    r = processing.run("native:rastercalc", {"EXPRESSION": '"Citra_KPH@4" - "Citra_KPH@1"',
                                             "LAYERS": [citra], "OUTPUT": "TEMPORARY_OUTPUT"})["OUTPUT"]
    tegaskan(isinstance(r, str) and os.path.exists(r), "keluaran raster bukan jalur berkas")
    return "vektor -> layer, raster -> jalur teks"


def k4():
    jalur = processing.run("native:rastercalc", {"EXPRESSION": NDVI, "LAYERS": [citra],
                                                 "OUTPUT": "TEMPORARY_OUTPUT"})["OUTPUT"]
    st = processing.run("native:rasterlayerstatistics",
                        {"INPUT": QgsRasterLayer(jalur, "ndvi", "gdal"), "BAND": 1})
    tegaskan(-1 <= st["MIN"] <= st["MAX"] <= 1, "NDVI di luar -1..1")
    return "NDVI %.2f sampai %.2f" % (st["MIN"], st["MAX"])


def k5():
    e = QgsExpression(CACAH)
    hasil = {f["NAMA_KPH"]: e.evaluate(konteks(f)) for f in batas.getFeatures()}
    benar = {f["NAMA_KPH"]: sum(1 for h in hotspot.getFeatures()
                                if h.geometry().intersects(f.geometry())) for f in batas.getFeatures()}
    tegaskan(len(hasil) == 3, "agregasi tidak menghasilkan 3 nilai: %s" % hasil)
    tegaskan(hasil == benar, "agregasi %s beda dari hitungan manual %s" % (hasil, benar))
    return str(hasil)
```

```
def k6():
    e = QgsExpression("'Titik api: ' || " + CACAH + " || ' titik'")
    beta = [x for x in batas.getFeatures() if x["NAMA_KPH"] == "KPH Beta"][0]
    teks = e.evaluate(konteks(beta))
    tegaskan(not e.hasEvalError() and teks == "Titik api: 5 titik", "teks label salah: %s" % teks)
    return teks


def k7():
    try:
        processing.run("native:buffer", {"INPUT": DATA + "/tidak_ada.gpkg", "DISTANCE": 5,
                                         "OUTPUT": "TEMPORARY_OUTPUT"})
    except QgsProcessingException:
        return "galat alat tertangkap"
    raise AssertionError("seharusnya gagal")


def k8():
    lyr = QgsVectorLayer(DATA + "/Kiriman_Lapangan/tim_e.gpkg", "rusak", "ogr")
    tegaskan(not lyr.isValid(), "berkas rusak dianggap sah")
    return "isValid() = False untuk berkas rusak"


def k9():
    berkas = HASIL + "/uji_kesiapan.qgz"
    tegaskan(proyek.write(berkas), "gagal menulis proyek")
    proyek.clear()
    tegaskan(len(proyek.mapLayers()) == 0, "proyek belum kosong")
    tegaskan(proyek.read(berkas) and len(proyek.mapLayers()) == 3, "proyek tidak terbaca utuh")
    return "tulis, kosongkan, baca ulang: 3 layer"


daftar = [("1. Muat vektor dan raster [Bab 1]", k1), ("2. Ambil layer dengan nama [Bab 1]", k2),
          ("3. processing.run sementara [Bab 2]", k3), ("4. Raster calculator NDVI [Bab 2]", k4),
          ("5. Agregasi dan @parent [Bab 5]", k5), ("6. Teks label dari ekspresi [Bab 5]", k6),
          ("7. Menangkap galat alat [Bab 4]", k7), ("8. Mendeteksi berkas rusak [Bab 4]", k8),
          ("9. Simpan dan buka proyek [Bab 1]", k9)]
for nama, fungsi in daftar:
    periksa(nama, fungsi)


# 10. Tugas latar belakang. Hasilnya tiba belakangan, jadi skor akhir dicetak di selesai().
def kerja(task):
    total = 0
    for i in range(1, 6):
        if task.isCanceled():
            return None
        time.sleep(0.05)
        total += i
        task.setProgress(i * 20)
    return total


def selesai(exception, nilai=None):
    ok = exception is None and nilai == 15
    catatan.append(("10. QgsTask sederhana [Bab 3]", ok, "hasil tugas = %s" % nilai))
    print("%-46s %s  hasil tugas = %s" % (catatan[-1][0], "LULUS" if ok else "GAGAL", nilai))
    lulus = sum(1 for _, o, _ in catatan if o)
    gagal = sorted({re.search(r"\[(Bab \d)\]", n).group(1) for n, o, _ in catatan if not o})
    akhir = "Siap masuk M1." if not gagal else "Ulangi: " + ", ".join(gagal) + "."
    print("SKOR: %d dari %d. %s" % (lulus, len(catatan), akhir))


tugas_uji = QgsTask.fromFunction("Uji tugas latar belakang", kerja, on_finished=selesai)
QgsApplication.taskManager().addTask(tugas_uji)
```

Penjelasan baris demi baris. Skrip ini 160 baris, jadi dibahas dalam lima blok. Nomor baris mengikuti berkas di atas.

Blok 1: bahan dan dua pembantu (baris 1-49).

Baris	Tulisan	Artinya
5-11	`import ...` dan `from qgis.core import (...)`	Perkakas yang sudah dikenal, ditambah `re` untuk mencari pola teks dan `time` untuk jeda.
17-19	`proyek = ...`, `removeAllMapLayers()`, `catatan = []`	Ambil proyek, **kosongkan**, dan siapkan daftar catatan. Tiap catatan berisi nama, lulus atau tidak, dan alasan.
20-22	`NDVI` dan `CACAH`	Dua rumus yang dipakai pemeriksaan, ditulis sekali di atas.
25-32	`def periksa(nama, fungsi):`	**Jantung ceklis.** Fungsi pemeriksaan dijalankan di dalam `try`. Bila berhasil, catat lulus dengan alasan yang dikembalikan fungsi itu. Bila galat apa pun, catat gagal dengan `type(e).__name__` (jenis galat) dan pesannya. Baris 31-32 mencetak hasilnya.
35-37	`def tegaskan(syarat, pesan):`	Bila syarat salah, `raise AssertionError(pesan)`. Pemeriksaan lain cukup memanggil fungsi ini.
40-42	tiga layer	Buka layer batas, titik panas, dan citra.
45-49	`def konteks(fitur):`	Pembantu konteks ekspresi, sama dengan Skrip 22.4.

Perhatikan bahwa `periksa` memakai `except Exception`, seperti Skrip 22.2. Di sini itu memang disengaja: ceklis harus tetap berjalan sampai akhir, apa pun yang gagal.

Blok 2: sembilan pemeriksaan, `k1` sampai `k9` (baris 52-125). Tiap fungsi menguji satu keterampilan. Fungsi yang tidak memanggil `raise` dianggap lulus.

Pemeriksaan	Baris	Yang diuji	Asal materi
1	52-56	Tiga layer terbuka (`isValid()`), masuk proyek, jumlahnya 3	Bab 14
2	59-64	`mapLayersByName` memberi layer yang benar. Batas berisi 3 fitur, citra 4 band dan piksel 2 meter. Nama yang tidak ada memberi daftar kosong	Bab 14
3	67-74	Hasil `native:buffer` berupa layer vektor, hasil `native:rastercalc` berupa jalur teks berkas	Bab 21
4	77-83	NDVI dihitung dengan `rastercalc`, lalu nilainya harus antara -1 dan 1 (perbandingan berantai)	Bab 21
5	86-93	Agregasi `@parent` menghasilkan tiga angka, dan sama dengan hitungan manual memakai `intersects`	Bab 22
6	96-101	Teks label `Titik api: 5 titik` untuk KPH Beta	Bab 22

Pemeriksaan	Baris	Yang diuji	Asal materi
7	104-110	Alat dengan masukan yang tidak ada **harus** gagal. Bila tidak gagal, `raise AssertionError("seharusnya gagal")`	Bab 22
8	113-116	Berkas rusak `tim_e.gpkg` memberi `isValid()` bernilai `False`	Bab 22
9	119-125	Tulis proyek, kosongkan (`proyek.clear()`), baca ulang, dan tiga layer kembali	Bab 14

Pada pemeriksaan 3, `isinstance(v, QgsVectorLayer)` bertanya: "apakah `v` berjenis layer vektor?" Jawabannya `True` atau `False`.

Pemeriksaan 7 menarik: ia menguji bahwa **galat memang terjadi**. Kalimat terakhirnya, `raise AssertionError`, hanya tercapai bila alat tidak gagal, dan itu justru kegagalan ujiannya.

Blok 3: daftar fungsi (baris 128-134).

Baris	Tulisan	Artinya
128-132	`daftar = [("1. ...", k1), ...]`	Daftar pasangan: nama pemeriksaan dan **fungsinya**. Fungsi boleh dipakai sebagai nilai. Tulis `k1` tanpa kurung, karena yang disimpan adalah fungsinya, bukan hasilnya.
133-134	`for nama, fungsi in daftar: periksa(nama, fungsi)`	Jalankan semuanya dengan satu perulangan. Menambah pemeriksaan baru cukup menambah satu pasangan.

Blok 4: pemeriksaan 10, tugas latar belakang (baris 137-146).

Baris	Tulisan	Artinya
138-146	`def kerja(task):`	Fungsi tugas sederhana: menjumlahkan 1 sampai 5 dengan jeda singkat (`time.sleep(0.05)`), lalu mengembalikan 15.

Blok 5: skor akhir (baris 149-160).

Baris	Tulisan	Artinya
150-152	`def selesai(exception, nilai=None):`	Pemeriksaan ke-10 lulus bila tidak ada galat dan hasilnya 15. Catat dan cetak seperti pemeriksaan lain.
153	`lulus = sum(1 for _, o, _ in catatan if o)`	Hitung catatan yang lulus.
154	`gagal = sorted({re.search(r"\[(Bab \d)\]", n).group(1) for ...})`	Dari nama pemeriksaan yang gagal, ambil bagian `[Bab n]` dengan pola teks, kumpulkan tanpa kembar (himpunan), lalu urutkan.
155-156	`akhir = ...` lalu `print("SKOR: ...")`	Bila tidak ada yang gagal, cetak "Siap masuk M1." Bila ada, cetak bab yang perlu diulang.
159-160	`QgsTask.fromFunction(...)` lalu `addTask`	Buat dan kirim tugas, seperti Skrip 23.4.

Hasil. Layar menampilkan (berdasarkan uji pada QGIS 4.0.2):

```
HASIL
1. Muat vektor dan raster [Bab 1]              LULUS
2. Ambil layer dengan nama [Bab 1]             LULUS  band=4, piksel=2 m
3. processing.run sementara [Bab 2]            LULUS  vektor -> layer, raster -> jalur teks
4. Raster calculator NDVI [Bab 2]              LULUS  NDVI -0.02 sampai 0.84
5. Agregasi dan @parent [Bab 5]                LULUS  {'KPH Alpha': 2, 'KPH Beta': 5, 'KPH Gamma': 5}
6. Teks label dari ekspresi [Bab 5]            LULUS  Titik api: 5 titik
7. Menangkap galat alat [Bab 4]                LULUS  galat alat tertangkap
8. Mendeteksi berkas rusak [Bab 4]             LULUS  isValid() = False untuk berkas rusak
9. Simpan dan buka proyek [Bab 1]              LULUS  tulis, kosongkan, baca ulang: 3 layer
10. QgsTask sederhana [Bab 3]                  LULUS  hasil tugas = 15
SKOR: 10 dari 10. Siap masuk M1.
```

Sepuluh lulus, skor 10 dari 10. Baris `10.` dan `SKOR` muncul paling akhir, setelah tugas latar belakang selesai.

Label `[Bab n]` menunjuk bab Buku I4, bukan bab buku ini. Skrip ini ditulis untuk buku seri. Jadi `[Bab 1]` berarti Bab 14 buku ini, `[Bab 2]` dan `[Bab 3]` berarti Bab 21, serta `[Bab 4]` dan `[Bab 5]` berarti Bab 22. Bila skor Anda kurang, baca pesan "Ulangi: Bab n" dengan terjemahan itu.

Uji negatif: membuktikan ceklis bisa berkata GAGAL

Ceklis yang selalu LULUS tidak berguna. Karena itu penulisnya menyertakan satu berkas hasil uji negatif, `hasil_uji/b7_03_ceklis_uji_negatif.txt`. Keterangannya menyebut bahwa nama berkas `Batas_KPH.gpkg` pada skrip sengaja diubah menjadi `Batas_KPX.gpkg`. **Skrip uji negatif itu sendiri tidak ada** di folder seri. Yang tersedia hanya hasilnya.

Agar jujur, bagian ini tidak mengandalkan berkas hasil itu saja. Untuk bab ini, uji negatif dibuat ulang: satu salinan skrip diubah di baris 40 saja, yaitu `"/Batas_KPH.gpkg"` menjadi `"/Batas_KPX.gpkg"`, lalu dijalankan. Hasilnya **sama persis**, baris demi baris, dengan berkas hasil uji milik penulis. Apakah

penulis mengubah tepat baris itu, tidak bisa dipastikan dari berkasnya. Tetapi satu perubahan itu menghasilkan keluaran yang sama, jadi penjelasannya masuk akal.

Hasil uji negatif:

```
HASIL
1. Muat vektor dan raster [Bab 1]              GAGAL  AssertionError: Batas_KPH tidak sah
2. Ambil layer dengan nama [Bab 1]             GAGAL  IndexError: list index out of range
3. processing.run sementara [Bab 2]            GAGAL  AssertionError: bukan layer vektor
4. Raster calculator NDVI [Bab 2]              LULUS  NDVI -0.02 sampai 0.84
5. Agregasi dan @parent [Bab 5]                GAGAL  AssertionError: agregasi tidak menghasilkan 3 nilai: {}
6. Teks label dari ekspresi [Bab 5]            GAGAL  IndexError: list index out of range
7. Menangkap galat alat [Bab 4]                LULUS  galat alat tertangkap
8. Mendeteksi berkas rusak [Bab 4]             LULUS  isValid() = False untuk berkas rusak
9. Simpan dan buka proyek [Bab 1]              GAGAL  AssertionError: proyek tidak terbaca utuh
10. QgsTask sederhana [Bab 3]                  LULUS  hasil tugas = 15
SKOR: 4 dari 10. Ulangi: Bab 1, Bab 2, Bab 5.
```

Cara membacanya.

- **Baca GAGAL yang pertama.** Pemeriksaan 1 menyebut sebab sebenarnya: `Batas_KPH tidak sah`. Pemeriksaan 2, 3, 5, 6, dan 9 gagal sebagai akibat, dengan pesan yang tampak berbeda, misalnya `IndexError` atau `bukan layer vektor`. Pesan itu gejala, bukan sebab.
- **Yang tetap LULUS.** Pemeriksaan 4, 7, 8, dan 10 tidak memakai layer batas, jadi tetap lulus. Itu bukti bahwa pemeriksaan berdiri sendiri.
- **Skor dan saran.** `SKOR: 4 dari 10. Ulangi: Bab 1, Bab 2, Bab 5.` Dalam peta bab buku ini: Bab 14, Bab 21, dan Bab 22.

Bagian yang boleh diubah.

- **Baris 13-14, jalur.** Sesuaikan dengan komputer Anda.
- **Baris 60, 62, dan 100, angka harapan.** Angka 3, 4, 2.0, dan teks `"Titik api: 5 titik"` hanya cocok untuk data paket `paket-i4`. Ubah bila data Anda berbeda.
- **Baris 128-132, daftar pemeriksaan.** Tambah pemeriksaan baru di sini.
- **Baris 138-146, tugas ke-10.** Ganti isi `kerja` dengan pekerjaan yang ingin Anda uji.

Jebakan.

- **Angka harapan terkunci pada satu paket data.** Pada uji, `"Titik api: 5 titik"` di baris 100 diganti `"Titik api: 6 titik"`. Hasilnya `6. Teks label dari ekspresi [Bab 5] GAGAL AssertionError: teks label salah: Titik api: 5 titik`, dan skor `9 dari 10. Ulangi: Bab 5.` Ceklis menolak data yang berbeda, walau skripnya benar.
- `proyek.clear()` menghapus objek layer. Pemeriksaan 9 mengosongkan proyek, dan layer yang dibuka di baris 40-42 ikut terhapus. Pada uji, pemeriksaan baru ditambahkan **setelah** pemeriksaan 9 dan memakai layer `hotspot`. Hasilnya `RuntimeError: wrapped C/C++ object of type QgsVectorLayer has been deleted`, dan skor menjadi `10 dari 11. Ulangi: Bab 1.` Letakkan pemeriksaan baru **sebelum** pemeriksaan 9, atau buka layernya lagi.

- **Pemeriksaan 10 hanya menguji penjumlahan.** Tugas latar belakang di baris 138-146 hanya menjumlahkan 1 sampai 5. Lulus berarti `QgsTask` berjalan. Itu tidak membuktikan Anda bisa membuat batch sungguhan seperti Skrip 23.4.
- **Lulus 10 dari 10 bukan berarti mahir.** Ceklis hanya memeriksa sepuluh keterampilan dengan satu paket data. Anggap ini tanda bekal cukup, bukan sertifikat.
- **Proyek Anda dikosongkan.** Baris 18 dan 122 menghapus semua layer di proyek terbuka. Jalankan di proyek kosong.

Latihan variasi.

1. Tambahkan pemeriksaan ke-11: jumlah seluruh titik panas harus 12.
2. Ubah harapan pemeriksaan 6 agar gagal, lalu baca skornya.

Contoh jawaban nomor 1 (sudah diuji). Tulis fungsi ini sebelum baris 128.

```python
def k11():
    tegaskan(sum(1 for _ in hotspot.getFeatures()) == 12, "titik panas bukan 12")
    return "12 titik panas"
```

Lalu tambahkan `("11. Jumlah titik panas [Bab 1]", k11),` ke daftar, **tepat sebelum** pasangan `k9`, sesuai Jebakan di atas. Layar menampilkan `11. Jumlah titik panas [Bab 1]  LULUS  12 titik panas` di antara pemeriksaan 8 dan 9, mengikuti urutan di daftar. Skor akhirnya `SKOR: 11 dari 11. Siap masuk M1.`

Contoh jawaban nomor 2 (sudah diuji): ganti `"Titik api: 5 titik"` di baris 100 dengan `"Titik api: 6 titik"`. Layar menampilkan pemeriksaan 6 GAGAL dan `SKOR: 9 dari 10. Ulangi: Bab 5.`

> **Pola yang bisa dipakai ulang:** Kerangka ceklis otomatis: fungsi `periksa(nama, fungsi)` yang menjalankan fungsi di dalam `try` dan mencatat LULUS atau GAGAL, fungsi `tegaskan(syarat, pesan)` untuk `raise AssertionError`, dan daftar pasangan `(nama, fungsi)`. Beri tiap nama label bab dalam kurung siku agar skor akhir bisa menyebut bab yang perlu diulang. Ganti fungsi pemeriksaannya, sesuai skrip yang ingin Anda uji.

Cek paham

1. Mengapa keluaran model di Skrip 23.2 diambil dengan `hasil["sempadan_per_kph"]`, dan apa yang terjadi bila label keluaran di Skrip 23.1 diubah?
2. Mengapa fungsi tugas di Skrip 23.4 membuka layernya sendiri, dan fungsi mana yang boleh menambah layer ke proyek?
3. Pada hasil uji negatif Skrip 23.5, pemeriksaan 3 gagal dengan pesan "bukan layer vektor". Apa penyebab sebenarnya, dan di mana Anda membacanya?

Jawaban.

1. Nama keluaran model dibentuk dari label keluarannya, yaitu "Sempadan per KPH" menjadi `sempadan_per_kph`. Bila label diganti, nama itu ikut berubah, dan Skrip 23.2 berhenti karena tidak menemukan tempat hasil dengan nama lama.

2. Fungsi tugas berjalan di latar belakang dan tidak boleh menyentuh proyek. Karena itu ia membuka layernya sendiri. Fungsi `selesai` berjalan di program utama, jadi aman menambah layer.
3. Penyebabnya berkas batas tidak bisa dibuka karena nama berkasnya salah. Penyebab itu tertulis pada pemeriksaan 1, yaitu GAGAL yang pertama. Pemeriksaan 3 hanya gejala.

Kesalahan umum

- **Menjalankan Skrip 23.2 sebelum Skrip 23.1.** Perbaikan: jalankan pembuat model lebih dulu, dan baca baris pertama hasil. `Model terbaca? False` berarti berkas model tidak ada.
- **Menjalankan skrip bab ini di proyek yang penting.** Skrip 23.3 dan 23.5 mengosongkan proyek terbuka, dan Skrip 23.4 menambah layer. Perbaikan: buka proyek kosong lebih dulu.
- **Menunggu pesan galat di layar padahal tugas berjalan di latar belakang.** Perbaikan: buka panel Log Messages bila hasil tidak muncul.
- Memakai layer tanpa memeriksa `isValid()`. Perbaikan: periksa dulu, seperti `muat_raster` di Skrip 23.3. Skrip 23.4 yang tidak memeriksa bisa menghasilkan angka ngawur tanpa galat.
- **Mengira skor 10 dari 10 sama dengan mahir.** Perbaikan: pakai ceklis sebagai pengingat, lalu coba skrip sendiri pada data Anda.

Ringkasan dan latihan

- Model Processing hanyalah berkas `.model3`. Ia bisa dibuat dengan klik atau dengan kode, lalu dijalankan berulang dari Python dengan `processing.run(model, {...})`.
- Penjaga di dalam model menghentikan pekerjaan dengan pesan jelas bila isian tidak masuk akal.
- Proyek terpadu memadukan pola dari bab sebelumnya: pembantu, tahap demi tahap, tabel ringkasan, GeoPackage, dan proyek.
- Tugas latar belakang menjaga QGIS tetap hidup. Galatnya hanya masuk ke log, jadi periksa log bila hasil tidak muncul.
- Ceklis otomatis memakai pola `periksa`, `tegaskan`, dan daftar fungsi. Uji negatif membuktikan ceklis bisa berkata GAGAL.

Latihan. Buat salinan Skrip 23.5, lalu tambahkan pemeriksaan ke-11 untuk skrip yang menurut Anda paling penting dari bab ini. Tulis syarat yang benar-benar bisa gagal, misalnya hasil model 23.2 untuk 30 meter harus 4,07 ha. Uji dua kali: sekali dengan data benar, sekali dengan data yang sengaja Anda rusak.

Tabel perbandingan: empat cara menjalankan pekerjaan yang sama

Cara	Kelebihan	Kekurangan	Skrip
Model Designer (klik)	Tanpa kode, bisa dilihat sebagai gambar	Sulit diulang dengan banyak nilai	Buku I4, Bab 6
Model dari Python	Bisa diulang dengan perulangan dan diuji	Perlu hafal nama kelas model	23.1 dan 23.2
Skrip terpadu	Satu jalan dari data sampai proyek	QGIS terasa macet saat bekerja	23.3
Tugas latar belakang	QGIS tetap bisa dipakai	Galat hanya tampak di log	23.4

BAB 24: Satu Perintah, Banyak Pekerjaan: NDVI, Validasi, dan Batch dengan Alat Processing

> **Berdasarkan Seri M1:** Skrip bab ini berasal dari Buku M1, *Masterclass GIS Modern* (Master Class), folder `skrip/qgis`, berkas `skrip_1_1.py`, `skrip_1_1_alternatif.py`, `skrip_2_1.py`, `skrip_3_1.py`, `skrip_3_1_alternatif.py`, dan `ekspresi_6_3.txt`. Di dalam berkas, nomor skrip mengikuti bab buku M1 (misalnya "Skrip 1.1"), jadi bisa berbeda dari nomor di bab ini.

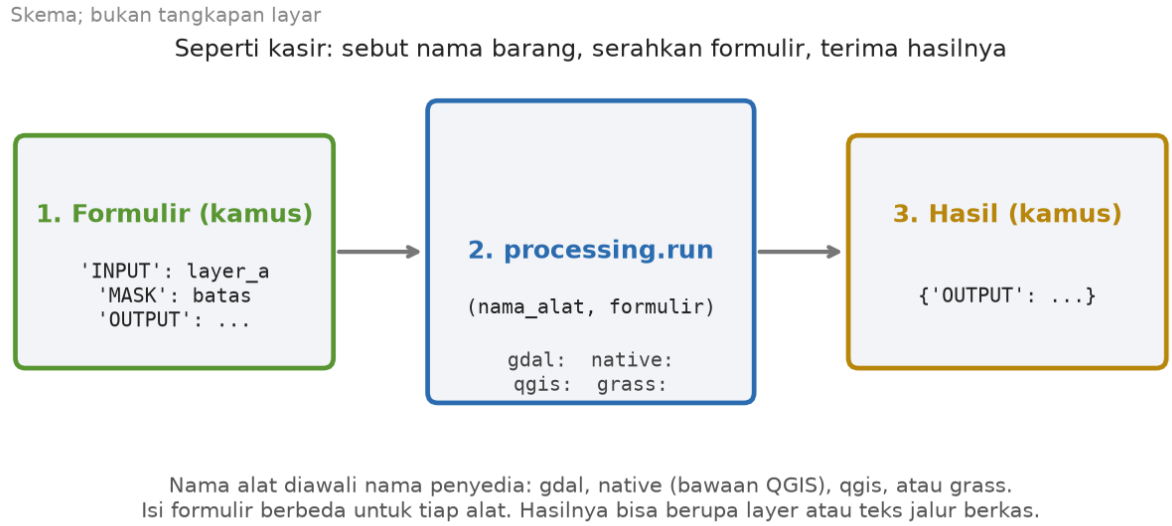
Studi kasus: "Hijaukah KPH kita, dan benarkah petanya?"

Kepala Seksi minta tiga hal sekaligus. Pertama, peta kehijauan tanaman di KPH Contoh dari citra satelit, tanpa mengunduh seluruh citra. Kedua, angka akurasi: seberapa cocok peta klasifikasi dengan hasil survei lapangan? Ketiga, kehijauan dari puluhan foto drone, tanpa membuat QGIS macet. Analis Muda tidak mau mengklik alat satu per satu. Ia memanggil alatnya dari skrip, sehingga pekerjaan yang sama bisa diulang kapan saja.

Konsep: Processing dalam tiga kalimat

QGIS punya ratusan alat analisis yang terkumpul di satu tempat bernama **Processing**. Bayangkan kasir di toko alat: Anda menyebut nama barang, menyerahkan formulir isian, lalu menerima barangnya. Di skrip, "menyebut nama barang" adalah `processing.run("nama_alat", ...)`, dan formulirnya adalah sebuah kamus (dibahas di Bab 5).

Ilustrasi 24.1: Processing seperti kasir



Lima istilah baru di bab ini:

- **Alat Processing** (*algorithm*): satu alat analisis dengan nama tetap, misalnya `gdal:cliprasterbymasklayer`. Awalan sebelum titik dua adalah penyedia alat: `gdal`, `native`, `qgis`, atau `grass`.

- **Kamus parameter**: formulir isian alat. Tiap isian punya nama (misalnya `'INPUT'`) dan nilai.
- **NDVI**: angka kehijauan tanaman dari dua pita citra. Rumusnya `(NIR - Merah) / (NIR + Merah)`. Nilai tinggi berarti tanaman lebat. NIR adalah inframerah dekat.
- **COG** (*Cloud-Optimized GeoTIFF*): citra GeoTIFF yang bisa dibaca sebagian lewat internet. Awalan `/vsicurl/` pada alamat memberi tahu GDAL (perkakas pembaca raster) untuk membaca lewat internet.
- **Tugas latar belakang** (*background task*): pekerjaan yang jalan di belakang layar, sehingga jendela QGIS tetap bisa dipakai.

> **Cara menjalankan skrip bab ini:** Skrip bab ini dibuat untuk Konsol Python QGIS atau Editor Skrip (lihat Bab 3). Skrip 24.1 sampai 24.3 butuh layer tertentu sudah termuat di proyek QGIS. Muat dulu berkas dari paket data `paket-data`, lalu jalankan skripnya. Uji untuk buku ini memakai QGIS 4.0.2 tanpa jendela, dengan layer dimuat lebih dulu.

Skrip 24.1: NDVI dari citra cloud (versi naskah)

Berkas `skrip_1_1.py`. Di dalam berkasnya tertulis "Skrip 1.1". Skrip ini memotong dua pita citra lalu menghitung NDVI dengan `qgis:rastercalculator`. Pelajaran paling berharga di bab ini ada di dua baris terakhirnya: **hasil alat Processing sering berupa teks jalur, bukan layer**. Kita bedah sampai tuntas.

Tujuan. Memuat dua pita citra Sentinel-2 (Merah dan NIR) dari internet, memotongnya dengan batas KPH, lalu menghitung NDVI.

Bahan. Paket data `paket-data`: berkas `Batas_KPH.gpkg` (layer `Batas_KPH`, yang harus sudah dimuat di proyek) dan, sebagai pengganti internet, `Sentinel_Lokal/B04.tif` (Merah) serta `B08.tif` (NIR). Alamat citra pada skrip adalah contoh dan tidak benar-benar ada (lihat Jebakan), jadi untuk berlatih Anda mengganti baris 11 dan 12 dengan jalur berkas lokal.

Skrip memuat dua pita, memotong keduanya, lalu menghitung NDVI. Gambar alurnya ada di Skrip 24.2, yang memakai alat hitung lain di langkah akhir.

Skrip utuh:

```python
# [SKRIP 1.1: Memuat COG & Eksekusi NDVI Virtual di QGIS 4.x]
# Penulis: Badar Mubarok Yogaswara
# Syarat: Pastikan layer batas area (misal: 'Batas_KPH') aktif di QGIS.

import processing
from qgis.core import QgsProject, QgsRasterLayer

print("🔗 Memulai koneksi ke Cloud-Optimized GeoTIFF (COG)...")

# 1. Tentukan URL COG dari STAC API (Contoh URL dummy Sentinel-2)
url_b4 = "/vsicurl/https://sentinel-cogs.s3.us-west-2.amazonaws.com/sentinel-s2-l2a-cogs/2026/T48M/B04.tif"
url_b8 = "/vsicurl/https://sentinel-cogs.s3.us-west-2.amazonaws.com/sentinel-s2-l2a-cogs/2026/T48M/B08.tif"

# 2. Muat Layer secara Virtual
layer_b4 = QgsRasterLayer(url_b4, "B04_Red_Cloud", "gdal")
layer_b8 = QgsRasterLayer(url_b8, "B08_NIR_Cloud", "gdal")

if not layer_b4.isValid() or not layer_b8.isValid():
```

```
    print("❌ Gagal memuat citra dari Cloud. Cek koneksi internet.")
else:
    QgsProject.instance().addMapLayers([layer_b4, layer_b8])
    print("✅ Citra Red dan NIR berhasil dimuat.")

    # 3. Ambil layer Vektor untuk batas pemotongan
    batas_kph = QgsProject.instance().mapLayersByName('Batas_KPH')[0]

    # 4. Potong (Clip) Citra langsung dari Cloud menggunakan Algoritma Native GDAL
    print("✂️ Memotong citra berdasarkan batas KPH...")

    # Clip Band 8 (NIR)
    clip_b8 = processing.run("gdal:cliprasterbymasklayer", {
        'INPUT': layer_b8,
        'MASK': batas_kph,
        'CROP_TO_CUTLINE': True,
        'OUTPUT': 'TEMPORARY_OUTPUT'
    })['OUTPUT']

    # Clip Band 4 (Red)
    clip_b4 = processing.run("gdal:cliprasterbymasklayer", {
        'INPUT': layer_b4,
        'MASK': batas_kph,
        'CROP_TO_CUTLINE': True,
        'OUTPUT': 'TEMPORARY_OUTPUT'
    })['OUTPUT']

    # 5. Kalkulasi NDVI Menggunakan Raster Calculator Native
    # Rumus: (NIR - RED) / (NIR + RED)
    print("📊 Menghitung NDVI...")

    ndvi_result = processing.run("qgis:rastercalculator", {
        'EXPRESSION': '("B08@1" - "B04@1") / ("B08@1" + "B04@1")',
        'LAYERS': [clip_b8, clip_b4],
        'CELLSIZE': 10,
        'OUTPUT': 'TEMPORARY_OUTPUT'
    })

    QgsProject.instance().addMapLayer(QgsRasterLayer(ndvi_result['OUTPUT'], "NDVI_KPH", "gdal"))
    print("🚀 Selesai! Layer NDVI_KPH telah ditambahkan ke kanvas.")
```

Penjelasan baris demi baris. Nomor baris mengikuti berkas di atas.

Baris	Tulisan	Artinya
1-3	`# [SKRIP 1.1 ...`	Komentar: judul, nama penulis, dan syarat: layer batas KPH harus sudah aktif di QGIS.
5	`import processing`	Panggil perkakas `processing`, pintu masuk ke semua alat Processing.
6	`from qgis.core import QgsProject, QgsRasterLayer`	Panggil dua perkakas QGIS: `QgsProject` untuk proyek yang sedang terbuka, `QgsRasterLayer` untuk membuka layer raster.
8	`print("🔗 ...")`	Tampilkan pesan awal. Tanda gambar (emoji) di dalam teks boleh saja, tetapi sebagian konsol menampilkannya sebagai kotak.
11-12	`url_b4 = ..., url_b8 = ...`	Simpan alamat dua pita citra di kotak `url_b4` dan `url_b8`. Awalan `/vsicurl/` berarti "baca lewat internet". B04 adalah pita Merah, B08 adalah NIR.

Baris	Tulisan	Artinya
15-16	`QgsRasterLayer(url_b4, "B04_Red_Cloud", "gdal")`	Buka layer raster. Tiga isian: sumber data, nama layer di QGIS, dan pembaca raster `"gdal"`.
18	`if not layer_b4.isValid() or not layer_b8.isValid():`	Periksa: bila salah satu layer tidak sah (gagal dibuka), jalankan blok di bawahnya. `or` berarti "atau".
19-20	`print("✗ ..."), else:`	Bila gagal, cetak pesan galat. Bila berhasil, jalankan blok `else` (baris 21 dan seterusnya).
21-22	`QgsProject.instance().addMapLayers([...])`	Tambahkan kedua layer ke proyek sekaligus. Tanda kurung siku membuat daftar berisi dua layer.
25	`mapLayersByName('Batas_KPH')[0]`	Cari layer bernama `Batas_KPH` di proyek. Hasilnya daftar, jadi `[0]` mengambil yang pertama.
31-36	`clip_b8 = processing.run("gdal:cliprasterbymasklayer", {...})['OUTPUT']`	Panggil alat potong raster dengan batas KPH. Kamusnya memuat `INPUT` (raster), `MASK` (pemotong), `CROP_TO_CUTLINE` (pangkas tepi) dan `OUTPUT`. Hasilnya kamus, dan `['OUTPUT']` mengambil isi kunci `OUTPUT`.
34	`'CROP_TO_CUTLINE': True`	Pangkas ukuran raster mengikuti batas pemotong.
35	`'OUTPUT': 'TEMPORARY_OUTPUT'`	Simpan hasil di berkas sementara. Yang dikembalikan adalah **teks jalur** berkas itu.
39-44	`clip_b4 = ...`	Ulangi pemotongan untuk pita Merah.
50-55	`processing.run("qgis:rastercalculator", {...})`	Hitung NDVI dengan kalkulator raster. `EXPRESSION` adalah rumus. Nama seperti `"B08@1"` berarti "pita 1 dari layer bernama B08". `LAYERS` adalah daftar layer yang dipakai.
53	`'CELLSIZE': 10`	Ukuran sel hasil: 10 satuan peta (meter pada data UTM).
54	`'OUTPUT': 'TEMPORARY_OUTPUT'`	Simpan hasil di berkas sementara. Kembaliannya **teks jalur** berkas itu, bukan layer. Tulisan `memory:` tidak cocok untuk raster (lihat bawah).
57	`addMapLayer(QgsRasterLayer(ndvi_result['OUTPUT'], "NDVI_KPH", "gdal"))`	Bungkus teks jalur menjadi layer raster bernama `NDVI_KPH`, lalu tambahkan ke proyek. Tanpa pembungkus, `addMapLayer` menerima teks dan gagal.
58	`print("🚀 ...")`	Cetak pesan selesai.

Hasil. Ada dua kemungkinan, bergantung pada baris 11-12.

Dengan alamat bawaan skrip (tidak diubah), layar menampilkan dua baris saja:

HASIL

```
🔗 Memulai koneksi ke Cloud-Optimized GeoTIFF (COG)...
✗ Gagal memuat citra dari Cloud. Cek koneksi internet.
```

Dengan citra lokal (baris 11-12 diganti jalur `Sentinel_Lokal/B04.tif` dan `B08.tif`), skrip berjalan sampai selesai:

HASIL

```
🔗 Memulai koneksi ke Cloud-Optimized GeoTIFF (COG)...
✅ Citra Red dan NIR berhasil dimuat.
✂ Memotong citra berdasarkan batas KPH...
📊 Menghitung NDVI...
🚀 Selesai! Layer NDVI_KPH telah ditambahkan ke kanvas.
```

Layer `NDVI_KPH` muncul di proyek. Ukurannya 40 x 40 piksel, sebab pita masukan 200 x 200 piksel bersel 2 meter dan baris 53 meminta sel 10 meter. Pada uji ini, nilai NDVI terkecil 0,0194, terbesar 0,8246, rata-rata 0,5018. Hasil ini diuji pada QGIS 4.0.2 tanpa jendela, dengan data sintetis, jadi angkanya bukan kondisi lapangan.

Menyelidiki: apa isi hasil Processing? Fungsi `addMapLayer` minta sebuah layer. Namun `ndvi_result['OUTPUT']` berisi teks. Buktinya bisa Anda lihat sendiri. Sisipkan dua baris ini tepat sebelum baris 57 (dengan jorokan yang sama):

PYTHON

```
    print(ndvi_result)
    print(type(ndvi_result['OUTPUT']))
```

Baris pertama menampilkan kamus berisi jalur berkas sementara. Jalurnya berbeda di tiap komputer. Baris kedua menampilkan:

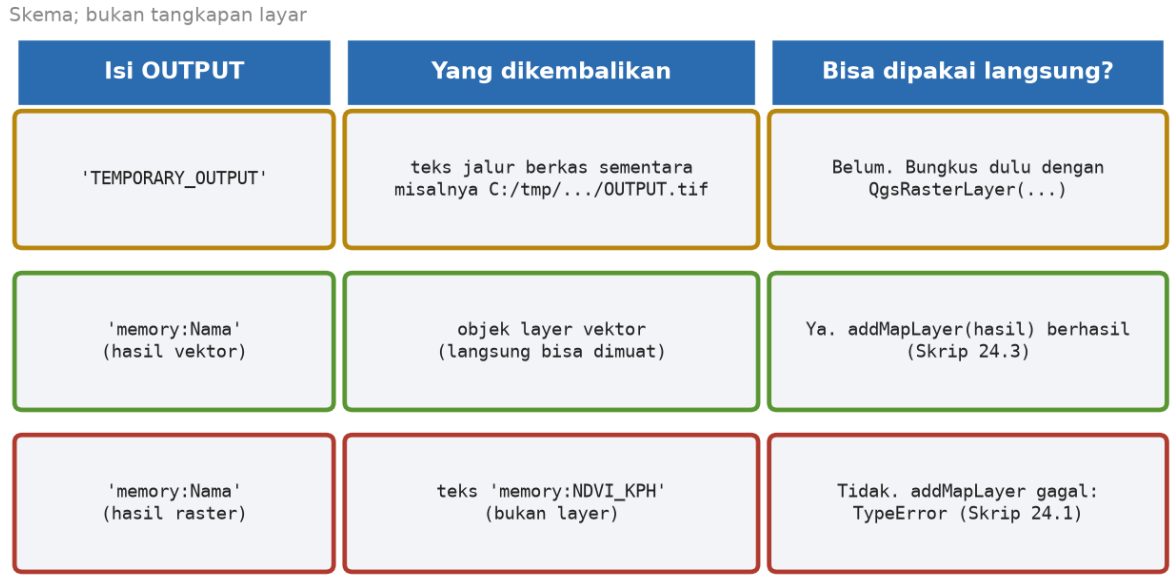
HASIL

```
<class 'str'>
```

Jadi alat mengembalikan **teks** (`'str'` singkatan *string*), yaitu jalur berkas hasil. Karena itu baris 57 membungkusnya dengan `QgsRasterLayer`.

Bagaimana bila baris 54 ditulis `'OUTPUT': 'memory:NDVI_KPH'` dan baris 57 memakai `addMapLayer(ndvi_result['OUTPUT'])`? Uji menunjukkan alat tetap jalan tanpa galat. Namun kamus hasilnya `{'OUTPUT': 'memory:NDVI_KPH'}`, yang juga teks. Baris 57 lalu berhenti dengan `TypeError: QgsProject.addMapLayer(): argument 1 has unexpected type 'str'`. `TypeError` berarti "jenis nilainya salah". Efek sampingnya, sebuah berkas kosong bernama `memory` (0 byte) muncul di folder kerja. Untuk hasil **vektor**, tulisan `memory:` memang berfungsi (lihat Skrip 24.3). Untuk hasil **raster**, tidak.

Ilustrasi 24.2: Tiga cara menulis OUTPUT



Bagian yang boleh diubah.

- **Baris 11-12, alamat citra.** Ganti dengan jalur berkas lokal, misalnya `"D:/Latihan/paket-data/Sentinel_Lokal/B04.tif"`. Tanpa awalan `/vsicurl/`, karena berkasnya ada di komputer.
- **Baris 25, nama layer batas.** Ganti `'Batas_KPH'` dengan nama layer pemotong Anda.
- **Baris 53, ukuran sel.** Ganti `10` sesuai ukuran piksel citra Anda.
- **Baris 54, tempat hasil.** Ganti `'TEMPORARY_OUTPUT'` dengan jalur berkas tetap, misalnya `'D:/Hasil/ndvi.tif'`, bila NDVI harus disimpan. Baris 57 tetap sama. Jangan memakai `memory:` untuk raster.

Jebakan.

- **Alamat contoh tidak ada.** Saat diuji dengan perintah `curl`, alamat pada baris 11 menjawab kode 404 ("tidak ditemukan"). Komentar skrip sendiri menyebutnya "URL dummy". Hasilnya, skrip berhenti di pesan "Gagal memuat citra".
- **Layer batas belum dimuat.** Bila layer `Batas_KPH` tidak ada di proyek, baris 25 gagal dengan `IndexError: list index out of range`. Daftar hasil pencarian kosong, jadi `[0]` tidak punya isi.

- **Memakai `memory:` untuk raster.** Alat tidak galat, tetapi hasilnya teks `memory:NDVI_KPH`, bukan layer. Galat baru muncul di `addMapLayer`, beberapa baris sesudahnya. Pesan "Menghitung NDVI" tidak menjamin NDVI sudah ada.
- **Hasil Processing bukan selalu layer.** Baris 31-36, 39-44, dan 50-55 mengembalikan teks jalur, bukan layer. Teks ini boleh dipakai sebagai sumber data, tetapi bukan sebagai layer. Itu sebabnya baris 57 membungkusnya.

Latihan variasi.

1. Ganti alamat pada baris 11-12 dengan jalur berkas lokal.
2. Tambahkan satu baris setelah baris 36 untuk melihat jenis hasil `clip_b8`.

Contoh jawaban, hanya bagian yang berubah:

```
PYTHON
url_b4 = "D:/Latihan/paket-data/Sentinel_Lokal/B04.tif"
url_b8 = "D:/Latihan/paket-data/Sentinel_Lokal/B08.tif"
```

```
PYTHON
    print(type(clip_b8))
```

Baris `print` menampilkan `<class 'str'>`: hasil pemotongan juga teks jalur.

> **Pola yang bisa dipakai ulang:** Setiap pemanggilan alat Processing punya tiga langkah: **siapkan kamus parameter**, **panggil** `processing.run(...)`, lalu **periksa jenis hasilnya** (teks jalur atau layer) sebelum memakainya. Bila hasilnya teks jalur, bungkus dengan `QgsRasterLayer(jalur, nama, "gdal")` atau `QgsVectorLayer(jalur, nama, "ogr")`.

Skrip 24.2: NDVI cloud, dengan layer bernama

Berkas `skrip_1_1_alternatif.py`. Di dalam berkasnya tertulis "SKRIP 1.1 - ALTERNATIF". Perbedaannya hanya di langkah 5: kalkulator raster `native:rastercalc` (alat yang terdokumentasi) dengan layer yang diberi nama sendiri, dan hasilnya dimuat dari berkas.

Tujuan. Sama dengan Skrip 24.1, yaitu layer `NDVI_KPH`, tetapi dengan nama layer yang Anda atur sendiri.

Bahan. Sama dengan Skrip 24.1 (paket `paket-data`, layer `Batas_KPH` sudah dimuat, baris 13-14 diganti jalur lokal).

Ilustrasi 24.3: Alur Skrip 24.2



Pelajaran: hasil potongan berupa teks jalur. Bungkus jadi layer bernama sebelum dipakai di ekspresi.

Skrip utuh:

PYTHON

```
# [SKRIP 1.1 - ALTERNATIF: Memuat COG & Eksekusi NDVI di QGIS 4.x]
# Penulis: Badar Mubarok Yogaswara
# Syarat: Pastikan layer batas area (misal: 'Batas_KPH') aktif di QGIS.
# Perbedaan dengan Skrip 1.1: langkah 5 memakai native:rastercalc (terdokumentasi) dan hasilnya
# disimpan ke berkas sementara lalu dimuat sebagai layer. Keluaran 'memory:' tidak menghasilkan layer raster.

import processing
from qgis.core import QgsProject, QgsRasterLayer

print("🔗 Memulai koneksi ke Cloud-Optimized GeoTIFF (COG)...")

# 1. Tentukan URL COG dari STAC API (Contoh URL dummy Sentinel-2)
url_b4 = "/vsicurl/https://sentinel-cogs.s3.us-west-2.amazonaws.com/sentinel-s2-l2a-cogs/2026/T48M/B04.tif"
url_b8 = "/vsicurl/https://sentinel-cogs.s3.us-west-2.amazonaws.com/sentinel-s2-l2a-cogs/2026/T48M/B08.tif"

# 2. Muat Layer secara Virtual
layer_b4 = QgsRasterLayer(url_b4, "B04_Red_Cloud", "gdal")
layer_b8 = QgsRasterLayer(url_b8, "B08_NIR_Cloud", "gdal")

if not layer_b4.isValid() or not layer_b8.isValid():
    print("❌ Gagal memuat citra dari Cloud. Cek koneksi internet.")
else:
    QgsProject.instance().addMapLayers([layer_b4, layer_b8])
    print("✅ Citra Red dan NIR berhasil dimuat.")

    # 3. Ambil layer Vektor untuk batas pemotongan
    batas_kph = QgsProject.instance().mapLayersByName('Batas_KPH')[0]

    # 4. Potong (Clip) Citra langsung dari Cloud menggunakan Algoritma Native GDAL
    print("✂️ Memotong citra berdasarkan batas KPH...")

    # Clip Band 8 (NIR)
    clip_b8 = processing.run("gdal:cliprasterbymasklayer", {
        'INPUT': layer_b8,
        'MASK': batas_kph,
```

```
    'CROP_TO_CUTLINE': True,
    'OUTPUT': 'TEMPORARY_OUTPUT'
})['OUTPUT']

# Clip Band 4 (Red)
clip_b4 = processing.run("gdal:cliprasterbymasklayer", {
    'INPUT': layer_b4,
    'MASK': batas_kph,
    'CROP_TO_CUTLINE': True,
    'OUTPUT': 'TEMPORARY_OUTPUT'
})['OUTPUT']

# 5. Kalkulasi NDVI Menggunakan Raster Calculator Native
# Rumus: (NIR - RED) / (NIR + RED)
print("📊 Menghitung NDVI...")

# Beri nama pada hasil potongan agar bisa dirujuk di ekspresi sebagai "B08@1" dan "B04@1"
lyr_b8 = QgsRasterLayer(clip_b8, "B08", "gdal")
lyr_b4 = QgsRasterLayer(clip_b4, "B04", "gdal")
QgsProject.instance().addMapLayers([lyr_b8, lyr_b4], False)

ndvi_path = processing.run("native:rastercalc", {
    'EXPRESSION': '("B08@1" - "B04@1") / ("B08@1" + "B04@1")',
    'LAYERS': [lyr_b8, lyr_b4],
    'OUTPUT': 'TEMPORARY_OUTPUT'
})['OUTPUT']

QgsProject.instance().addMapLayer(QgsRasterLayer(ndvi_path, "NDVI_KPH", "gdal"))
print("🚀 Selesai! Layer NDVI_KPH telah ditambahkan ke kanvas.")
```

Penjelasan yang baru. Sampai baris 50, isi skrip ini sama dengan Skrip 24.1, hanya nomornya lebih besar dua karena komentar tambahan di baris 4-5. Karena itu hanya bagian baru yang dijelaskan. Nomor baris mengikuti berkas di atas.

Baris	Tulisan	Artinya
4-5	`# Perbedaan dengan Skrip 1.1 ...`	Komentar penulis: langkah 5 diganti, dan ia sendiri mencatat bahwa keluaran `memory:` tidak menghasilkan layer raster.
52	`# Beri nama pada hasil potongan ...`	Komentar: nama layer harus cocok dengan nama di rumus.
53-54	`lyr_b8 = QgsRasterLayer(clip_b8, "B08", "gdal")`	Bungkus teks jalur hasil potongan menjadi layer sungguhan. Nama layer `"B08"` dan `"B04"` yang dipakai di rumus.
55	`addMapLayers([lyr_b8, lyr_b4], False)`	Daftarkan kedua layer ke proyek. Isian kedua, `False`, berarti jangan tampilkan di panel Layers. Layer tetap dikenali oleh nama.
57-61	`processing.run("native:rastercalc", {...})['OUTPUT']`	Hitung NDVI dengan `native:rastercalc`. Ia tidak butuh `CELLSIZE`. Hasil ke `TEMPORARY_OUTPUT`, jadi kembaliannya teks jalur berkas sementara.
63	`QgsRasterLayer(ndvi_path, "NDVI_KPH", "gdal")`	Bungkus jalur hasil menjadi layer bernama `NDVI_KPH`, lalu `addMapLayer` memasukkannya ke peta.
64	`print("🚀 ...")`	Pesan selesai.

Hasil. Dengan citra lokal pada baris 13-14, layar menampilkan:

```
HASIL
🔗 Memulai koneksi ke Cloud-Optimized GeoTIFF (COG)...
✅ Citra Red dan NIR berhasil dimuat.
✂ Memotong citra berdasarkan batas KPH...
📊 Menghitung NDVI...
🚀 Selesai! Layer NDVI_KPH telah ditambahkan ke kanvas.
```

Layer `NDVI_KPH` muncul di proyek. Ukurannya 200 x 200 piksel dengan sel 2 meter, sama dengan pita masukan. Skrip 24.1 menghasilkan 40 x 40 piksel, sebab ia meminta sel 10 meter di `CELLSIZE`. Pemeriksaan tambahan pada uji ini: nilai NDVI terkecil 0,002, terbesar 0,836, rata-rata 0,505. Karena datanya sintetis, angka ini bukan kondisi lapangan.

Bagian yang boleh diubah.

- **Baris 13-14, alamat citra.** Jalur lokal atau alamat COG sungguhan.
- **Baris 53-54, nama layer.** Boleh diganti, asal sama dengan nama di rumus baris 58.
- **Baris 58, rumus.** Untuk indeks lain, ganti rumusnya. Contoh: selisih sederhana `"B08@1" - "B04@1"`.
- **Baris 60, tempat hasil.** Ganti `'TEMPORARY_OUTPUT'` dengan jalur berkas tetap bila hasil harus disimpan.

Jebakan.

- **Nama layer di rumus tidak cocok.** Rumus memanggil `"B08@1"`. Bila layer Anda dinamai lain, hasilnya tidak sah. Selalu samakan.
- **Lupa membungkus.** Rumus memakai nama layer `B08`. Teks jalur `clip_b8` tidak punya nama itu. Bila layer bernama B08 tidak terdaftar di proyek dan teks jalur dimasukkan langsung ke `LAYERS`, hasilnya `QgsProcessingException: Error occurred while performing calculation.` (diuji).
- **Butuh internet.** Dengan alamat COG sungguhan, skrip bergantung pada koneksi dan pada alamat yang benar.
- **Satuan sel.** `native:rastercalc` memakai sel dari layer masukan. Cek ukurannya, jangan menebak.

Latihan variasi.

1. Cetak nilai terkecil, terbesar, dan rata-rata NDVI di akhir skrip.
2. Simpan hasil NDVI ke berkas tetap, bukan sementara.

Contoh jawaban 1, ditambahkan di akhir skrip:

```
PYTHON
ndvi = QgsProject.instance().mapLayersByName('NDVI_KPH')[0]
stat = ndvi.dataProvider().bandStatistics(1)
print("Min %.3f | maks %.3f | rata-rata %.3f" % (stat.minimumValue, stat.maximumValue, stat.mean))
```

Layar menampilkan `Min 0.002 | maks 0.836 | rata-rata 0.505`.

Contoh jawaban 2: ganti baris 60 dengan jalur tetap. Berkas `NDVI_KPH_tetap.tif` terbentuk di folder tujuan, dan `ndvi_path` berisi jalur itu.

```
PYTHON
        'OUTPUT': 'D:/Latihan/hasil/NDVI_KPH_tetap.tif'
```

> **Pola yang bisa dipakai ulang:** Untuk hitungan raster dengan nama layer: **bungkus hasil antara menjadi layer bernama** (`QgsRasterLayer(jalur, "NAMA", "gdal")`), **daftarkan ke proyek**, **tulis rumus** dengan `"NAMA@pita"`, lalu **bungkus hasil akhirnya**. Ganti tiga hal: nama layer, rumus, dan tempat keluaran.

Skrip 24.3: Akurasi peta dari titik survei

Berkas `skrip_2_1.py`. Di dalam berkasnya tertulis "SKRIP 2.1". Skrip ini mencocokkan hasil survei lapangan dengan kelas hasil klasifikasi citra. Di dunia penginderaan jauh, ini disebut validasi atau *ground truth* (kebenaran lapangan).

Tujuan. Menarik kelas dari poligon klasifikasi ke tiap titik survei, lalu menghitung berapa persen titik yang kelasnya cocok.

Bahan. Paket data `paket-data`: `Survei.gpkg` (layer `Titik_Survei`, 10 titik, kolom `Kondisi`) dan `Poligon_Sawah.gpkg` (layer `Poligon_Sawah`, 3 poligon, kolom `Kelas_Citra`). Kedua layer harus sudah dimuat di proyek dengan nama itu.

Skrip utuh:

```
PYTHON
# [SKRIP 2.1: Validasi Otomatis Titik Ground-Truth vs Poligon Klasifikasi]
# Penulis: Badar Mubarok Yogaswara

import processing
from qgis.core import QgsProject

print("🔍 Memulai Validasi Lapangan Otomatis...")

layer_survei = QgsProject.instance().mapLayersByName('Titik_Survei')[0]
layer_klasifikasi = QgsProject.instance().mapLayersByName('Poligon_Sawah')[0]

# Melakukan Spatial Join (Menarik atribut poligon ke titik survei yang berada di dalamnya)
hasil_join = processing.run("native:joinattributesbylocation", {
    'INPUT': layer_survei,
    'JOIN': layer_klasifikasi,
    'PREDICATE': [0], # 0 = Intersects
    'JOIN_FIELDS': ['Kelas_Citra'], # Field dari citra satelit
    'METHOD': 0,
    'DISCARD_NONMATCHING': False,
    'PREFIX': 'GIS_',
    'OUTPUT': 'memory:Validasi_Hasil'
})['OUTPUT']

# Menambahkan layer hasil ke QGIS
QgsProject.instance().addMapLayer(hasil_join)

# Menghitung Akurasi
titik_cocok = 0
```

```
total_titik = hasil_join.featureCount()

for fitur in hasil_join.getFeatures():
    if str(fitur['Kondisi']).lower() == str(fitur['GIS_Kelas_Citra']).lower():
        titik_cocok += 1

akurasi = (titik_cocok / total_titik) * 100 if total_titik > 0 else 0

print(f"📊 Laporan Validasi: {titik_cocok} dari {total_titik} titik cocok.")
print(f"🎯 Akurasi Keseluruhan (Overall Accuracy): {akurasi:.2f}%")
```

Penjelasan baris demi baris. Nomor baris mengikuti berkas di atas.

Baris	Tulisan	Artinya
1-2	`# [SKRIP 2.1 ...`	Komentar: judul dan nama penulis.
4-5	`import processing, from qgis.core import QgsProject`	Panggil perkakas Processing dan perkakas proyek.
7	`print("🔍 ...")`	Pesan awal.
9-10	`mapLayersByName('Titik_Survei')[0]`	Ambil layer titik survei dan layer poligon dari proyek, masing-masing ke satu kotak.
13-22	`processing.run("native:joinattributesbylocation", {...})['OUTPUT']`	Gabungkan atribut berdasarkan letak: tiap titik menerima isi kolom dari poligon yang memuatnya. Hasilnya diambil dari kunci `OUTPUT`.
14-15	`'INPUT', 'JOIN'`	Layer yang menerima (titik) dan layer pemberi (poligon).
16	`'PREDICATE': [0]`	Syarat letak. Isiannya daftar. `0` berarti "bersinggungan atau berada di dalam" (*intersects*).
17	`'JOIN_FIELDS': ['Kelas_Citra']`	Hanya kolom ini yang dibawa dari poligon.
18	`'METHOD': 0`	Satu titik boleh menerima banyak poligon cocok, masing-masing menjadi satu baris.
19	`'DISCARD_NONMATCHING': False`	Titik yang tidak jatuh di poligon mana pun tetap disimpan, dengan isi kosong.
20	`'PREFIX': 'GIS_'`	Awalan nama kolom yang dibawa. Kolom baru bernama `GIS_Kelas_Citra`.
21	`'OUTPUT': 'memory:Validasi_Hasil'`	Simpan hasil di memori dengan nama `Validasi_Hasil`. Untuk hasil vektor, ini berfungsi dan mengembalikan layer.
25	`addMapLayer(hasil_join)`	Masukkan layer hasil ke proyek.
28-29	`titik_cocok = 0, total_titik = hasil_join.featureCount()`	Siapkan penghitung cocok di angka 0, lalu hitung jumlah titik.
31-33	`for fitur in hasil_join.getFeatures():`	Ulangi untuk tiap titik. Bila kondisi survei sama dengan kelas citra, tambah penghitung dengan `+= 1`.
32	`str(...).lower()`	Ubah isi kolom menjadi teks huruf kecil, supaya `Sehat` dan `sehat` dianggap sama.

Baris	Tulisan	Artinya
35	`akurasi = (...) * 100 if total_titik > 0 else 0`	Hitung persen. Bagian `if ... else 0` mencegah pembagian dengan nol bila tidak ada titik.
37-38	`print(f"...")`	Cetak laporan. Huruf `f` di depan kutip membuat teks di dalam kurung kurawal diganti nilainya (dibahas di Bab 9). `:.2f` berarti dua angka desimal.

Hasil. Layar menampilkan:

```
HASIL
🔍 Memulai Validasi Lapangan Otomatis...
📊 Laporan Validasi: 9 dari 10 titik cocok.
🎯 Akurasi Keseluruhan (Overall Accuracy): 90.00%
```

Layer `Validasi_Hasil` muncul di proyek dengan kolom `fid`, `Kondisi`, `Tinggi_Phn`, `Foto`, dan `GIS_Kelas_Citra`. Satu titik sengaja dibuat tidak cocok, jadi 9 dari 10.

Bagian yang boleh diubah.

- **Baris 9-10, nama layer.** Ganti dengan nama layer Anda. Harus sama persis dengan nama di panel Layers.
- **Baris 17, kolom yang dibawa.** Ganti `Kelas_Citra` dengan kolom poligon lain.
- **Baris 20 dan 32, awalan.** Bila awalan diganti, nama kolom di baris 32 harus ikut diganti.
- **Baris 16, syarat letak.** Isiannya daftar kode. Menurut daftar pilihan alat di QGIS 4.0.2: `0` bersinggungan (*intersect*), `1` memuat, `2` sama, `3` bersentuhan, `4` tumpang tindih, `5` berada di dalam, `6` menyilang.

Jebakan.

- **Awalan tidak sejalan.** Bila baris 20 diganti `'CITRA_'` tetapi baris 32 dibiarkan, layar menampilkan `KeyError: 'GIS_Kelas_Citra'`. Kolom itu tidak ada lagi.
- **Nama layer salah.** Layer tidak ditemukan, daftar kosong, dan baris 9 berhenti dengan `IndexError: list index out of range`.
- **Titik di luar poligon.** Titik itu bernilai kosong dan dihitung tidak cocok. Angka akurasi turun tanpa peringatan.
- **Banyak poligon cocok.** Dengan `METHOD: 0`, titik yang jatuh di dua poligon tumpang tindih menjadi dua baris, sehingga `total_titik` bisa lebih besar dari jumlah titik asli. Pada uji, satu titik di area tumpang tindih dua poligon (kelas `sehat` dan `rusak`) menghasilkan dua baris dengan `METHOD: 0`, dan satu baris (hanya `sehat`) dengan `METHOD: 1`.

Latihan variasi.

1. Tampilkan kondisi dan kelas citra untuk titik yang tidak cocok.
2. Ganti awalan menjadi `CITRA_` dengan benar.

Contoh jawaban 1, ditambahkan di akhir skrip:

```python
for fitur in hasil_join.getFeatures():
    if str(fitur['Kondisi']).lower() != str(fitur['GIS_Kelas_Citra']).lower():
        print("TIDAK COCOK:", fitur['Kondisi'], "|", fitur['GIS_Kelas_Citra'])
```

Layar menampilkan `TIDAK COCOK: Terserang Hama | Sehat`. Titik itu disurvei sebagai terserang hama, tetapi poligon citranya "sehat".

Contoh jawaban 2: ganti baris 20 menjadi `'PREFIX': 'CITRA_'` **dan** baris 32 menjadi `fitur['CITRA_Kelas_Citra']`. Hasilnya tetap 9 dari 10 titik cocok.

> **Pola yang bisa dipakai ulang:** Validasi dua sumber data punya empat langkah: **gabungkan atribut berdasarkan letak**, **ulangi tiap fitur**, **bandingkan dua kolom**, **hitung persen**. Ganti empat hal: nama dua layer, kolom pembanding, awalan, dan syarat letak.

Skrip 24.4: NDVI banyak ubin di latar belakang

Berkas `skrip_3_1.py`. Di dalam berkasnya tertulis "SKRIP 3.1". Drone menghasilkan puluhan ubin foto. Menghitung semuanya di jendela utama membuat QGIS membeku. Solusinya: jalankan sebagai **tugas latar belakang**.

Tujuan. Menghitung NDVI untuk semua berkas `.tif` di satu folder, satu per satu di belakang layar, lalu mencatat hasilnya di log.

Bahan. Paket data `paket-data`, folder `Ortofoto_Mentah` berisi `ubin_01.tif` sampai `ubin_03.tif` (tiap berkas 2 pita: pita 1 NIR, pita 2 Merah; ukuran 100 x 100 piksel). Folder keluaran, misalnya `NDVI_Hasil`, dibuat otomatis oleh skrip bila belum ada. **Jalur di baris 8 dan 9 adalah jalur komputer penulis**, jadi Anda wajib menggantinya.

Ilustrasi 24.4: Alur tugas latar belakang

Skema; bukan tangkapan layar



Kemajuan dilaporkan lewat setProgress (baris 35). Tombol batal dicek tiap putaran (baris 19).

Skrip utuh:

```python
# [SKRIP 3.1: Batch Processing Citra Drone di Latar Belakang (Non-Blocking)]
# Penulis: Badar Mubarok Yogaswara

import os
import processing
from qgis.core import QgsTask, QgsApplication, QgsMessageLog, Qgis

folder_input = "D:/Data_Drone/Ortofoto_Mentah/"
folder_output = "D:/Data_Drone/NDVI_Hasil/"

def proses_batch_ndvi(task, folder_in, folder_out):
    QgsMessageLog.logMessage("Memulai Batch Processing di Latar Belakang...", "Analisis Drone", Qgis.Info)
    os.makedirs(folder_out, exist_ok=True)

    file_list = [f for f in os.listdir(folder_in) if f.endswith('.tif')]
    total_files = len(file_list)

    for i, file_name in enumerate(file_list):
        if task.isCanceled():
            return False

        path_in = os.path.join(folder_in, file_name)
        path_out = os.path.join(folder_out, f"NDVI_{file_name}")

        processing.run("qgis:rastercalculator", {
            'EXPRESSION': '("{}@1" - "{}@2") / ("{}@1" + "{}@2")'.format(path_in, path_in, path_in, path_in),
            'LAYERS': [path_in],
            'OUTPUT': path_out
        })

        if not os.path.exists(path_out):
            QgsMessageLog.logMessage(f"Keluaran tidak terbentuk: {path_out}", "Analisis Drone", Qgis.Critical)
            return False

        task.setProgress((i + 1) / total_files * 100)

    return True

def task_selesai(exception, result=None):
    if result:
        QgsMessageLog.logMessage("✅ Batch Processing Selesai 100%!", "Analisis Drone", Qgis.Success)
    else:
        QgsMessageLog.logMessage("❌ Proses dibatalkan atau error.", "Analisis Drone", Qgis.Warning)

task_ndvi = QgsTask.fromFunction("Kalkulasi NDVI 50 Ubin", proses_batch_ndvi,
                                 on_finished=task_selesai,
                                 folder_in=folder_input, folder_out=folder_output)

QgsApplication.taskManager().addTask(task_ndvi)
```

Penjelasan baris demi baris. Skrip ini punya dua fungsi (dibahas di Bab 7) dan satu pemanggil. Nomor baris mengikuti berkas di atas.

Baris	Tulisan	Artinya
1-2	`# [SKRIP 3.1 ...`	Komentar: judul dan nama penulis.
4-6	`import os`, `import processing`, `from qgis.core import ...`	Panggil `os` (urusan berkas dan folder), `processing`, dan empat perkakas QGIS untuk tugas dan log.

Baris	Tulisan	Artinya
8-9	`folder_input = "D:/Data_Drone/Ortofoto_Mentah/"`	Dua kotak berisi lokasi folder masukan dan keluaran. Ini jalur komputer penulis dan **harus diganti**.
11	`def proses_batch_ndvi(task, folder_in, folder_out):`	Buat fungsi (resep) bernama `proses_batch_ndvi`. Ia menerima tiga isian: objek tugas, folder masukan, folder keluaran.
12	`QgsMessageLog.logMessage(..., "Analisis Drone", Qgis.Info)`	Tulis catatan ke panel log QGIS, di tab bernama `Analisis Drone`.
13	`os.makedirs(folder_out, exist_ok=True)`	Buat folder keluaran bila belum ada. `exist_ok=True` berarti "jangan galat bila folder sudah ada" (lihat Bab 8). Tanpa baris ini, folder yang tidak ada membuat GDAL gagal menulis.
15	`file_list = [f for f in os.listdir(folder_in) if f.endswith('.tif')]`	Kumpulkan nama berkas di folder masukan yang berakhiran `.tif` (daftar dengan syarat, dibahas di Bab 6). Huruf besar dan kecil dibedakan.
16	`total_files = len(file_list)`	Hitung jumlah berkas.
18	`for i, file_name in enumerate(file_list):`	Ulangi untuk tiap berkas. `enumerate` menambah nomor urut `i` mulai dari 0.
19-20	`if task.isCanceled(): return False`	Bila pengguna membatalkan tugas, berhenti dan kembalikan `False`.
22-23	`path_in = os.path.join(...),path_out = ...`	Susun jalur lengkap berkas masukan dan keluaran. Nama keluaran diawali `NDVI_` lewat f-string.
25-29	`processing.run("qgis:rastercalculator", {...})`	Hitung NDVI untuk satu ubin.
26	`'("{}@1" - "{}@2") / ("{}@1" + "{}@2")'.format(path_in, ...)`	Rumus NDVI. Pita 1 adalah NIR dan pita 2 Merah. Empat tanda `{}` diisi jalur berkas, sebab di sini nama layer adalah jalur berkas itu sendiri.
27	`'LAYERS': [path_in]`	Layer yang dipakai, ditulis sebagai jalur berkas.
28	`'OUTPUT': path_out`	Simpan hasil ke jalur keluaran.
31-33	`if not os.path.exists(path_out):`	Periksa apakah berkas hasil benar-benar terbentuk (`os.path.exists`, lihat Bab 8). Bila tidak, tulis pesan tingkat `Qgis.Critical` (paling berat) ke log, lalu `return False` menghentikan tugas.
35	`task.setProgress((i + 1) / total_files * 100)`	Laporkan kemajuan dalam persen ke QGIS.
37	`return True`	Tanda tugas selesai tanpa dibatalkan dan tanpa kegagalan.

Baris	Tulisan	Artinya
39-43	`def task_selesai(exception, result=None):`	Fungsi penutup. Bila `result` benar (`True`), catat sukses. Bila tidak, catat peringatan. Isian kedua diberi nilai bawaan `result=None` karena pada jalur gagal (tugas mengembalikan `False` atau galat) QGIS memanggil fungsi ini hanya dengan satu isian, `exception`. Tanpa nilai bawaan itu, pesan peringatan tidak pernah tercetak.
45-47	`QgsTask.fromFunction("Kalkulasi NDVI 50 Ubin", proses_batch_ndvi, ...)`	Bungkus fungsi menjadi tugas. Isian: nama tugas, fungsi kerja, fungsi penutup `on_finished`, dan dua isian kata kunci untuk folder.
49	`QgsApplication.taskManager().addTask(task_ndvi)`	Serahkan tugas ke pengelola tugas QGIS. Sejak baris ini, tugas jalan di belakang layar.

Hasil. Panel log QGIS (menu **View**, **Panels**, **Log Messages**), tab `Analisis Drone`, memuat dua pesan. Tiap pesan di panel diawali keterangan waktu. [CEK]

HASIL

```
Memulai Batch Processing di Latar Belakang...
✅ Batch Processing Selesai 100%!
```

Folder keluaran berisi tiga berkas:

HASIL

```
NDVI_ubin_01.tif   41284 byte
NDVI_ubin_02.tif   41284 byte
NDVI_ubin_03.tif   41284 byte
```

Pemeriksaan pada uji ini: ketiga hasil dibandingkan dengan hitungan NumPy `(NIR - Merah) / (NIR + Merah)`. Selisih terbesar hanya 0,00000003 (pembulatan angka desimal). Nilai NDVI tiap ubin berkisar 0,002 sampai 0,817.

Bagian yang boleh diubah.

- **Baris 8-9, folder.** Ganti dengan folder di komputer Anda, dengan garis miring biasa `/` dan garis miring di akhir.
- **Baris 15, jenis berkas.** Ganti `'.tif'` bila berkas Anda berekstensi lain.
- **Baris 26, pita.** Tukar `@1` dan `@2` bila urutan pita di data Anda terbalik.
- **Baris 45, nama tugas.** Ganti teks bebas. Tulisan "50 Ubin" adalah tulisan tetap, tidak dihitung dari data.
- **Baris 13 dan 31-33, pengaman.** Biarkan keduanya. Baris 13 membuat folder keluaran, dan baris 31-33 menghentikan tugas bila satu hasil tidak terbentuk.

Jebakan.

- **Folder keluaran tidak ada.** Baris 13 membuatnya otomatis. Tanpa baris itu, uji menunjukkan hal yang menipu: log tetap menulis "Batch Processing Selesai 100%!", padahal tidak ada berkas terbentuk. Di terminal tampil `ERROR 4: Attempt to create new tiff file ...`

`failed ... No such file or directory`. Karena itu baris 31-33 memeriksa berkas hasil. **Tetap periksa isi folder keluaran.**

- **Satu ubin gagal, sisanya tidak diproses.** Pada baris 31-33, `return False` mengakhiri tugas. Ubin yang belum dikerjakan dilewati. Cek log untuk tahu ubin mana yang gagal.
- **Jalur penulis.** Bila baris 8-9 tidak diganti dan folder masukan tidak ada, tugas berhenti karena galat `FileNotFoundError`. Rincian galat itu tidak tampil di log: yang terlihat pesan "Memulai Batch Processing di Latar Belakang...", lalu "✕ Proses dibatalkan atau error." (peringatan). Lebih dulu, baris 13 sempat membuat folder keluaran. Bila drive `D:` ada di komputer Anda, folder `D:/Data_Drone/NDVI_Hasil/` bisa terbentuk di tempat yang tidak Anda maksud (diuji pada folder tiruan, bukan pada drive `D:` sungguhan). Bila folder masukan ada tetapi kosong, log tetap menulis "Selesai 100%" tanpa membuat berkas. Ini latihan mengganti jalur yang paling sering Anda lakukan.
- **Ekstensi huruf besar.** Berkas `ubin_01.TIF` terlewat, sebab `.tif` peka huruf besar dan kecil.
- **Pesan galat palsu di log.** Pada uji ini, log juga memuat pesan `Cannot open <jalur berkas> ().()` dari pembaca data, tetapi hasilnya tetap benar. Penyebabnya nama layer berupa jalur berkas. Skrip 24.5 menghindarinya.
- **Jalur dengan spasi.** Pada uji, jalur berkas yang memuat spasi (folder bernama `dengan spasi`) tetap diproses benar oleh pola `qgis:rastercalculator` di skrip ini: keluaran terbentuk dan rata-rata NDVI 0,600, sama dengan jalur tanpa spasi. Tanda aneh lain pada jalur tidak diuji.
- **Pesan hanya di log.** Tidak ada tulisan di konsol. Buka panel log untuk melihat kemajuan.

Latihan variasi.

1. Ganti jalur folder sesuai komputer Anda.
2. Tambahkan catatan log untuk tiap berkas yang diproses.

Contoh jawaban 2: sisipkan satu baris sebelum baris 25 (tepat sesudah `path_out`), dengan jorokan yang sama.

```python
        QgsMessageLog.logMessage(f"Memproses {file_name}", "Analisis Drone", Qgis.Info)
```

Log menampilkan `Memproses ubin_01.tif`, `Memproses ubin_02.tif`, dan `Memproses ubin_03.tif` di antara pesan awal dan pesan selesai.

> **Pola yang bisa dipakai ulang:** Pekerjaan berat di latar belakang selalu berbentuk: **fungsi kerja** (menerima `task`, mengulang pekerjaan, membuat folder keluaran, mengecek `task.isCanceled()`, memeriksa bahwa hasil tiap ubin ada, melaporkan `task.setProgress(...)`, mengembalikan `True` atau `False`), **fungsi penutup**, lalu pemanggilan `QgsTask.fromFunction(...)` dan `addTask(...)`. Ganti tiga hal: folder, ekspresi, dan isi pekerjaan di dalam putaran.

Skrip 24.5: Batch NDVI, versi dengan layer bernama

Berkas `skrip_3_1_alternatif.py`. Di dalam berkasnya tertulis "SKRIP 3.1 - ALTERNATIF". Skrip ini mengerjakan hal yang sama dengan Skrip 24.4, tetapi memakai `native:rastercalc` dan layer yang diberi nama. Karena itu hanya perbedaannya yang dijelaskan.

Tujuan. Sama dengan Skrip 24.4, dengan cara yang lebih rapi dan terdokumentasi.

Bahan. Sama dengan Skrip 24.4: folder `Ortofoto_Mentah`, folder keluaran, dan jalur di baris 10-11 yang harus diganti.

Skrip utuh:

```python
# [SKRIP 3.1 - ALTERNATIF: Batch Processing Citra Drone di Latar Belakang (Non-Blocking)]
# Penulis: Badar Mubarok Yogaswara
# Perbedaan dengan Skrip 3.1: memakai native:rastercalc (terdokumentasi) dengan layer bernama,
# bukan qgis:rastercalculator dengan nama layer berupa jalur berkas.

import os
import processing
from qgis.core import QgsTask, QgsApplication, QgsMessageLog, Qgis, QgsRasterLayer

folder_input = "D:/Data_Drone/Ortofoto_Mentah/"
folder_output = "D:/Data_Drone/NDVI_Hasil/"

def proses_batch_ndvi(task, folder_in, folder_out):
    QgsMessageLog.logMessage("Memulai Batch Processing di Latar Belakang...", "Analisis Drone", Qgis.Info)
    os.makedirs(folder_out, exist_ok=True)

    file_list = [f for f in os.listdir(folder_in) if f.endswith('.tif')]
    total_files = len(file_list)

    for i, file_name in enumerate(file_list):
        if task.isCanceled():
            return False

        path_in = os.path.join(folder_in, file_name)
        path_out = os.path.join(folder_out, f"NDVI_{file_name}")

        # Beri nama layer agar bisa dirujuk di ekspresi
        layer_in = QgsRasterLayer(path_in, "ortofoto", "gdal")

        try:
            processing.run("native:rastercalc", {
                'EXPRESSION': '("ortofoto@1" - "ortofoto@2") / ("ortofoto@1" + "ortofoto@2")',
                'LAYERS': [layer_in],
                'OUTPUT': path_out
            })
        except Exception as e:
            QgsMessageLog.logMessage(f"Gagal memproses {file_name}: {e}", "Analisis Drone", Qgis.Critical)
            return False

        if not os.path.exists(path_out):
            QgsMessageLog.logMessage(f"Keluaran tidak terbentuk: {path_out}", "Analisis Drone", Qgis.Critical)
            return False

        task.setProgress((i + 1) / total_files * 100)

    return True

def task_selesai(exception, result=None):
    if result:
        QgsMessageLog.logMessage("✅ Batch Processing Selesai 100%!", "Analisis Drone", Qgis.Success)
```

```
    else:
        QgsMessageLog.logMessage("✗ Proses dibatalkan atau error.", "Analisis Drone", Qgis.Warning)

task_ndvi = QgsTask.fromFunction("Kalkulasi NDVI 50 Ubin", proses_batch_ndvi,
                                 on_finished=task_selesai,
                                 folder_in=folder_input, folder_out=folder_output)

QgsApplication.taskManager().addTask(task_ndvi)
```

Penjelasan yang baru. Bagian lain sama dengan Skrip 24.4, hanya nomor barisnya bergeser. Nomor baris mengikuti berkas di atas.

Baris	Tulisan	Artinya
3-4	`# Perbedaan dengan Skrip 3.1 ...`	Komentar penulis: kini memakai `native:rastercalc` dengan layer bernama, bukan nama layer berupa jalur berkas.
8	`... Qgis, QgsRasterLayer`	Satu perkakas ditambahkan: `QgsRasterLayer`.
15	`os.makedirs(folder_out, exist_ok=True)`	Buat folder keluaran bila belum ada, sama seperti Skrip 24.4.
27-28	`layer_in = QgsRasterLayer(path_in, "ortofoto", "gdal")`	Buka tiap ubin sebagai layer bernama `ortofoto`. Nama sama dipakai untuk semua ubin, sebab layer dibuat baru di tiap putaran.
30-38	`try: ... except Exception as e:`	Coba jalankan perhitungan. Bila terjadi galat jenis apa pun (`Exception`), loncat ke blok `except` (lihat Bab 10). Pesannya tersimpan di `e`.
31-35	`processing.run("native:rastercalc", {...})`	Rumus memakai `"ortofoto@1"` dan `"ortofoto@2"`, jauh lebih pendek daripada rumus berisi jalur. `LAYERS` berisi layer, bukan jalur.
37-38	`QgsMessageLog.logMessage(f"Gagal memproses {file_name}: {e}", ...), return False`	Catat nama ubin dan pesan galatnya sebagai `Qgis.Critical`, lalu hentikan tugas.
40-42	`if not os.path.exists(path_out):`	Sama seperti Skrip 24.4: periksa bahwa berkas hasil benar-benar ada.

Hasil. Log menampilkan dua pesan yang sama seperti Skrip 24.4. Tiga berkas `NDVI_ubin_01.tif` sampai `NDVI_ubin_03.tif` terbentuk, masing-masing 41284 byte. Pesan `Cannot open ...` tidak muncul lagi pada uji ini.

Bagian yang boleh diubah. Baris 10-11 (folder), baris 17 (jenis berkas), dan baris 32 (rumus), seperti pada Skrip 24.4. Satu tambahan: nama `"ortofoto"` di baris 28 boleh diganti, asal rumus di baris 32 ikut diganti.

Jebakan.

- **Nama layer dan rumus tidak cocok.** Bila baris 28 diganti `"foto"` tetapi rumus tetap `"ortofoto@1"`, perhitungan gagal. Blok `try/except` menangkap galatnya. Uji menunjukkan log memuat pesan `Gagal memproses ubin_01.tif: Error occurred while performing calculation.`, dan tugas berhenti. Pesan "Selesai 100%" tidak muncul dan berkas tidak dibuat. Pesan "✗ Proses dibatalkan atau error." muncul sesudah pesan `Gagal memproses`

(tingkat Warning). Tanpa `result=None` pada fungsi penutup, pesan itu tidak pernah tercetak, sebab QGIS memanggil fungsi penutup dengan satu isian pada jalur gagal. Tetap baca pesan `Gagal memproses` untuk tahu ubin mana yang gagal.

- **Jalur dan folder.** Sama dengan Skrip 24.4: ganti jalur. Folder keluaran dibuat otomatis oleh baris 15.

Latihan variasi.

1. Ganti nama layer menjadi `"foto"` di **dua tempat**: baris 28 dan rumus di baris 32. Jalankan lagi. Hasilnya sama seperti sebelumnya.
2. Ganti nama di baris 28 saja, lalu jalankan. Baca panel log dan cari pesan yang muncul.

Contoh jawaban 2: log memuat `Gagal memproses ubin_01.tif: Error occurred while performing calculation.`, dan tidak ada berkas hasil baru.

> **Pola yang bisa dipakai ulang:** Di dalam putaran batch, **buka tiap berkas sebagai layer bernama**, **pakai nama itu di rumus**, dan **tulis hasil ke jalur keluaran**. Ganti nama layer dan rumus bersama-sama.

Ekspresi 24.6: teks laporan untuk Atlas (bukan Python)

Berkas `ekspresi_6_3.txt`. Berkas ini **bukan skrip Python**. Isinya sebuah **ekspresi QGIS**: rumus kecil dengan bahasa tersendiri, yang dipakai di kotak teks atau label. Di buku M1, ekspresi ini ditempel di label tata letak Atlas, yaitu fitur QGIS yang membuat satu halaman peta untuk tiap fitur, misalnya satu halaman per KPH. Bahasa ekspresi tidak mengenal `import`, `print`, atau `def`.

Tujuan. Membuat teks laporan per KPH: nama, luas, dan jumlah titik api aktif.

Bahan. Paket data `paket-data`: `Batas_KPH.gpkg` (kolom `NAMA_KPH`) sebagai layer cakupan Atlas, dan `Hotspot_Nasional.gpkg` (layer `Hotspot_Nasional`, 12 titik, kolom `ID_Titik`). Kedua layer harus ada di proyek. Cara memakainya: tempel di label layout Atlas (bukan di Konsol Python).

Ekspresi utuh:

```
EKSPRESI
'LAPORAN KEBAKARAN HUTAN KPH ' || upper("NAMA_KPH") || '\n' ||
'----------------------------------' || '\n' ||
'Total Luas KPH: ' || round($area / 10000, 2) || ' Ha \n' ||
'Jumlah Titik Api Aktif: ' ||
aggregate(
  layer:= 'Hotspot_Nasional',
  aggregate:= 'count',
  expression:= "ID_Titik",
  filter:= intersects($geometry, geometry(@parent))
) || ' Titik.'
```

Penjelasan baris demi baris. Nomor baris mengikuti berkas di atas.

Baris	Tulisan	Artinya
1	`'LAPORAN KEBAKARAN HUTAN KPH ' ... upper("NAMA_KPH") ...`	Dua tanda garis tegak berurutan merangkai teks, dan di berkas tanda itu ada di sela-sela bagian baris ini. `upper(...)` mengubah nama KPH menjadi huruf besar. Tulisan `"NAMA_KPH"` dengan kutip ganda berarti kolom. Tulisan `'\n'` berarti ganti baris.
2	`'------...'`	Garis pemisah, lalu ganti baris.
3	`round($area / 10000, 2)`	`$area` adalah luas fitur yang sedang diproses, dalam meter persegi. Dibagi 10.000 menjadi hektare, lalu dibulatkan dua desimal.
4-10	`aggregate(layer:='Hotspot_Nasional', ...)`	Ambil ringkasan dari layer lain. `aggregate:= 'count'` berarti hitung jumlah, `expression:= "ID_Titik"` kolom yang dihitung, dan `filter:= intersects($geometry, geometry(@parent))` memilih hanya titik yang berada di dalam KPH yang sedang diproses.
10	`) ... ' Titik.'`	Tutup kurung `aggregate`, lalu sambung dengan kata penutup memakai dua tanda garis tegak.

Tanda `$geometry` menunjuk bentuk titik api yang sedang dicek. Tanda `@parent` menunjuk fitur KPH yang sedang diproses.

Hasil. Ekspresi ini diuji dengan memintanya dihitung untuk tiap KPH di layer `Batas_KPH`. Hasilnya, tiga laporan:

```
HASIL
LAPORAN KEBAKARAN HUTAN KPH KPH ALPHA
----------------------------------
Total Luas KPH: 4 Ha
Jumlah Titik Api Aktif: 2 Titik.

LAPORAN KEBAKARAN HUTAN KPH KPH BETA
----------------------------------
Total Luas KPH: 4 Ha
Jumlah Titik Api Aktif: 5 Titik.

LAPORAN KEBAKARAN HUTAN KPH KPH GAMMA
----------------------------------
Total Luas KPH: 8 Ha
Jumlah Titik Api Aktif: 5 Titik.
```

Kata "KPH" muncul dua kali di judul, sebab ia ada di teks tetap dan di nama KPH. Itu bukan galat.

Ekspresi juga bisa dicoba dari skrip Python. Contoh berikut menghitungnya untuk satu fitur (batas KPH pertama). Alat yang dipakai adalah `QgsExpression`, yaitu perkakas QGIS yang membaca dan menghitung ekspresi.

```
PYTHON
from qgis.core import QgsExpression, QgsExpressionContext, QgsExpressionContextUtils, QgsProject
batas = QgsProject.instance().mapLayersByName('Batas_KPH')[0]
```

```
teks = open("D:/Latihan/ekspresi_6_3.txt", encoding="utf-8").read()
ekspresi = QgsExpression(teks)
konteks = QgsExpressionContext(QgsExpressionContextUtils.globalProjectLayerScopes(batas))
konteks.setFeature(next(batas.getFeatures()))
print(ekspresi.evaluate(konteks))
```

Layar menampilkan laporan KPH Alpha, empat baris seperti di atas. Dalam uji, ekspresi diparsing tanpa galat.

Bagian yang boleh diubah.

- **Baris 1, judul.** Ganti teks di antara kutip tunggal.
- **Baris 3, satuan.** Ubah pembagi `10000` untuk satuan lain. Ingat: `$area` dalam satuan peta, jadi pada data derajat hasilnya bukan hektare.
- **Baris 6, nama layer.** Ganti `'Hotspot_Nasional'` dengan layer titik lain.
- **Baris 8, kolom hitung.** Ganti `"ID_Titik"` dengan kolom lain dari layer itu.

Jebakan.

- **Layer penghitung tidak ada di proyek.** Bila `Hotspot_Nasional` tidak dimuat, ekspresi tidak menghasilkan teks. Pada uji lewat skrip, hasilnya `None` dan pesan galatnya `Cannot find layer with name or ID 'Hotspot_Nasional'`.
- **Nama layer harus persis.** Huruf besar dan kecil dibedakan, dan nama yang dicari adalah nama di panel Layers.
- **Tanda ganti baris.** Tulisan `\n` hanya bermakna di tempat yang mengerti baris baru. Label Atlas menampilkannya sebagai baris baru. Kotak teks sebaris tidak. [CEK]

Latihan variasi. Buat ekspresi baru yang menampilkan kepadatan titik api per hektare. Contoh jawaban: bagi jumlah titik dengan luas dalam hektare, lalu bulatkan.

```
EKSPRESI
'Titik api per hektare: ' || round(
  aggregate(
    layer:= 'Hotspot_Nasional',
    aggregate:= 'count',
    expression:= "ID_Titik",
    filter:= intersects($geometry, geometry(@parent))
  ) / ($area / 10000), 2)
```

Hasilnya `Titik api per hektare: 0.5` untuk KPH Alpha, `1.25` untuk KPH Beta, dan `0.63` untuk KPH Gamma.

> **Pola yang bisa dipakai ulang:** Laporan teks per fitur: **judul dengan nama kolom**, **luas** dari `$area`, **jumlah** dari `aggregate(...)` dengan filter `intersects(...)`, lalu **rangkai semuanya dengan dua tanda garis tegak**. Ganti nama layer, nama kolom, dan teks tetap.

Cek paham

1. Mengapa baris 57 Skrip 24.1 membungkus `ndvi_result['OUTPUT']` dengan `QgsRasterLayer`? Apa yang terjadi bila baris 54 ditulis `'memory:NDVI_KPH'`?

2. Apa bedanya `'TEMPORARY_OUTPUT'` dan `'memory:Nama'` pada hasil raster?
3. Mengapa Skrip 24.4 sebaiknya dijalankan sebagai tugas latar belakang, dan untuk apa baris 13 serta baris 31-33?

Jawaban.

1. Karena alat mengembalikan teks jalur, bukan layer, dan `addMapLayer` minta layer. Bila baris 54 ditulis `'memory:NDVI_KPH'`, alat mengembalikan teks `'memory:NDVI_KPH'`, sehingga `addMapLayer` gagal dengan `TypeError`.
2. `'TEMPORARY_OUTPUT'` membuat berkas sementara dan mengembalikan jalurnya, yang bisa Anda bungkus menjadi layer. `'memory:Nama'` pada raster hanya mengembalikan teks dan tidak menghasilkan layer.
3. Agar QGIS tidak membeku saat memproses puluhan ubin. Baris 13 membuat folder keluaran bila belum ada. Baris 31-33 menghentikan tugas bila satu hasil tidak terbentuk. Tanpa keduanya, log bisa menulis "Selesai" padahal tidak ada berkas. Tetap cek isi folder keluaran.

Kesalahan umum

- **Memercayai pesan "Selesai".** Pesan itu hanya tulisan di skrip. Perbaikan: cek hasilnya (layer muncul, berkas ada), dan periksa keberadaan berkas di dalam skrip seperti baris 31-33 Skrip 24.4.
- **Memakai tulisan memory untuk hasil raster.** Perbaikan: pakai `'TEMPORARY_OUTPUT'` atau jalur berkas, lalu bungkus dengan `QgsRasterLayer`.
- **Membiarkan jalur komputer penulis.** Perbaikan: ganti baris jalur sesuai komputer Anda dan siapkan folder keluarannya.
- **Nama layer di rumus tidak cocok dengan nama layer sebenarnya.** Perbaikan: ganti keduanya bersama-sama.

Ringkasan dan latihan

- Alat Processing dipanggil dengan `processing.run("nama_alat", {kamus})` dan mengembalikan kamus hasil.
- Hasil alat bisa berupa layer atau **teks jalur**. Periksa jenisnya sebelum memakainya.
- `memory:` berfungsi untuk vektor, tidak untuk raster. Untuk raster, pakai `'TEMPORARY_OUTPUT'` atau jalur berkas, lalu bungkus dengan `QgsRasterLayer` (Skrip 24.1 dan 24.2).
- Pekerjaan berat dijalankan sebagai tugas latar belakang dengan `QgsTask`. Buat folder keluaran, periksa tiap hasil, pantau log, dan cek isi folder.
- Ekspresi QGIS (Ekspresi 24.6) adalah bahasa terpisah dari Python.

Latihan. Gabungkan dua skrip: ubah Skrip 24.5 supaya setelah semua ubin selesai, fungsi penutup mencatat jumlah berkas keluaran di folder. Bagian mana yang Anda ubah, dan fungsi apa dari modul `os` yang membantu?

Tabel perbandingan: tiga cara menulis OUTPUT

Isi OUTPUT	Yang dikembalikan	Cocok untuk
`'TEMPORARY_OUTPUT'`	Teks jalur berkas sementara	Hasil antara, raster maupun vektor
`'memory:Nama'`	Layer (vektor) atau teks yang tidak berguna (raster)	Hasil vektor kecil yang langsung dimuat
Jalur berkas, misalnya `'D:/Hasil/ndvi.tif'`	Teks jalur yang sama	Hasil akhir yang harus disimpan

BAB 25: Dari Lemari Berkas ke Gudang Data: PostGIS, Jaringan, Hidrologi, dan Kesesuaian Lahan dengan Skrip

> **Berdasarkan Seri M2:** Skrip bab ini berasal dari Buku M2, *Basis Data Spasial dan Analisis Lanjut* (Master Class), folder `skrip`, berkas `m2_01b_impor.py` sampai `m2_07d_cadangan.sh`, ditambah berkas pembantu `_inisialisasi.py`. Sembilan berkas SQL di folder yang sama hanya disebut di bagian akhir bab ini. Di buku itu, topik ini dibahas lewat menu, SQL, dan klik. Bab ini menunjukkan sisi skripnya.

Studi kasus: "Datanya ada di mana-mana, hasilnya beda-beda"

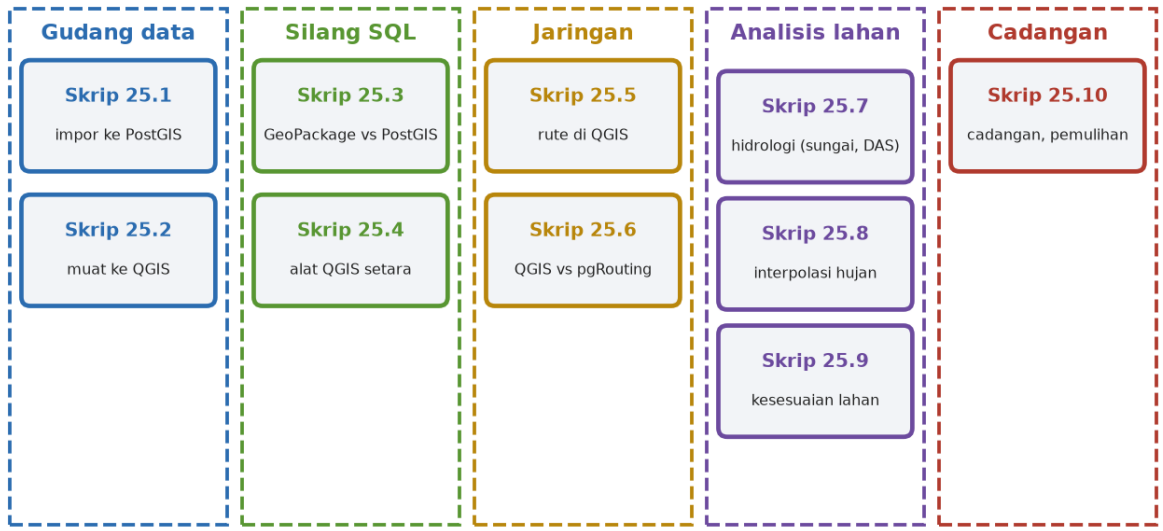
Setiap analis di KPH Contoh menyimpan salinan data sendiri. Petak ada di satu berkas, jalan di berkas lain, dan angka luas dihitung dengan cara yang berbeda-beda. Kepala Seksi minta satu "gudang data" bersama. Ia juga minta jawaban cepat: rute patroli tercepat antar pos, luas daerah aliran sungai, peta curah hujan, dan peta kesesuaian lahan. Analis Muda menjawabnya dengan skrip, supaya setiap angka bisa dicek dan diulang.

Konsep: gudang data dalam tiga kalimat

Berkas GeoPackage itu seperti lemari arsip pribadi: enak dibawa, tetapi tiap orang punya salinannya sendiri. **PostGIS** adalah gudang arsip pusat. Ia berupa program server yang melayani banyak orang sekaligus, dan Anda memintanya lewat surat bernama **SQL** (bahasa untuk bertanya kepada basis data).

Ilustrasi 25.1: Peta bab: sepuluh skrip dalam lima kelompok

Skema; bukan tangkapan layar



Skrip 25.9 memakai keluaran 25.7 (Sungai.gpkg) dan 25.8 (CH_tahunan.tif). Jalankan 25.7, 25.8, lalu 25.9.

Lima istilah baru yang perlu Anda kenal sekarang:

- **Basis data spasial**: tempat menyimpan tabel yang kolomnya boleh berisi bentuk (titik, garis, poligon). PostGIS menambahkan kemampuan itu ke basis data PostgreSQL.
- **Skema**: laci besar di dalam basis data. Semua tabel bab ini ada di laci bernama `kph`.
- **SQL spasial**: SQL yang mengerti bentuk, misalnya `ST_Area` (luas) dan `ST_Intersects` (bersinggungan).
- **Variabel lingkungan**: pengaturan yang disimpan di komputer, di luar skrip, yang boleh dibaca skrip. Cara membacanya `os.environ` (Bab 8).
- **Tanpa jendela** (*headless*): QGIS dijalankan dari terminal, tanpa membuka tampilan grafisnya.

Sebelum mulai: tiga persiapan

Pertama, paket data. Siapkan folder `paket-m2`. Isinya delapan berkas GeoPackage (`Petak.gpkg`, `Tanah.gpkg`, `Jalan.gpkg`, `Jalan_celah.gpkg`, `Fasilitas.gpkg`, `Kejadian.gpkg`, `Stasiun_Hujan.gpkg`, `Batas_Wilayah.gpkg`), satu peta ketinggian `DEM_10m.tif` (200 x 200 sel, sel 10 meter, wilayah 2 x 2 km), dan folder `hasil` untuk keluaran. Semua data sintetis, bukan kondisi lapangan nyata.

Kedua, PostGIS (untuk skrip tertentu). Skrip 25.1, 25.2, 25.6, dan 25.10 butuh server PostgreSQL dengan PostGIS (dan pgRouting untuk 25.6), serta basis data bernama `kph_contoh` dengan skema `kph`. Skrip 25.3 tetap jalan tanpa PostGIS, hanya bagian pembandingnya dilewati. Skrip 25.4 sampai 25.9 tidak butuh PostGIS. Skrip 25.7 dan 25.8 butuh GRASS. Pada pemasangan QGIS 4.0.2 di komputer uji, GRASS sudah ikut terpasang. Uji untuk buku ini memakai QGIS 4.0.2, PostgreSQL 17, PostGIS 3.6, dan pgRouting 4.0. Bila server Anda berbeda versi, cocokkan hasilnya. [CEK]

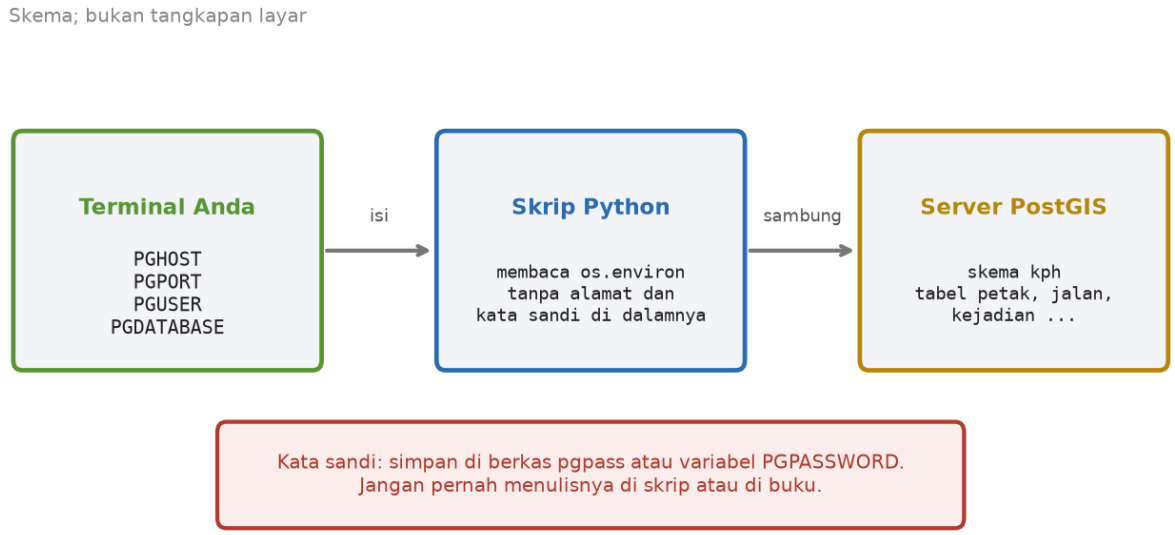
Ketiga, koneksi lewat variabel lingkungan. Skrip tidak memuat alamat server atau kata sandi. Skrip membaca empat pengaturan dari komputer Anda: `PGHOST` (alamat server), `PGPORT` (nomor port), `PGUSER` (nama pengguna), dan `PGDATABASE` (nama basis data). Isi dulu di terminal, lalu jalankan skripnya. Contoh di Windows (terminal Command Prompt):

TERMINAL

```
set PGHOST=localhost
set PGPORT=5432
set PGUSER=nama_pengguna_anda
set PGDATABASE=kph_contoh
"C:\Program Files\QGIS 4.0.2\bin\python-qgis.bat" m2_01b_impor.py D:\Latihan\paket-m2
```

Baris terakhir menjalankan skrip dengan Python milik QGIS (lihat Bab 3). Kata `D:\Latihan\paket-m2` adalah **argumen**: folder paket data, yang dibaca skrip lewat `sys.argv[1]`. Di macOS dan Ubuntu, perintahnya berbeda. [CEK]

Ilustrasi 25.2: Sambungan skrip ke PostGIS



> **Kata sandi tidak ditulis di skrip:** Bila server meminta kata sandi, simpan di berkas `pgpass` milik PostgreSQL, atau di variabel `PGPASSWORD`. Jangan menuliskannya di skrip, di catatan, atau di buku. Dalam uji untuk bab ini, server lokal tidak meminta kata sandi.

Berkas pembantu: _inisialisasi.py

Sebagian besar skrip bab ini dimulai dengan `import _inisialisasi`. Berkas pendek ini menyalakan QGIS tanpa jendela. Bayangkan menyalakan dapur sebelum memasak: bila kompor tidak dinyalakan, resep sebagus apa pun tidak jalan. Di Konsol Python QGIS, dapurnya sudah menyala, jadi berkas ini tidak diperlukan.

Skrip utuh:

```python
# -*- coding: utf-8 -*-
"""Berkas bantu M2: menyalakan QGIS tanpa jendela (headless) agar skrip bisa dijalankan dengan python-qgis.bat.
Penulis: Badar Mubarok Yogaswara. Di Python Console QGIS berkas ini TIDAK diperlukan."""
import os
import sys
from qgis.core import QgsApplication, Qgis

os.environ.setdefault("QT_QPA_PLATFORM", "offscreen")
qgs = QgsApplication([], False)
qgs.initQgis()
sys.path.append(os.path.join(QgsApplication.prefixPath(), "python", "plugins"))
import processing  # noqa: E402
from processing.core.Processing import Processing  # noqa: E402

Processing.initialize()

# GRASS: bila dijalankan tanpa jendela QGIS, beri tahu lokasi GRASS lewat variabel GISBASE
```

```
# (di jendela QGIS sendiri hal ini sudah otomatis dan bagian ini boleh dilewati).
_grass = os.path.join(os.path.dirname(QgsApplication.prefixPath()), "grass", "grass84")
if os.path.isdir(_grass) and "GISBASE" not in os.environ:
    os.environ["GISBASE"] = _grass
```

Penjelasan baris demi baris. Nomor baris mengikuti berkas di atas.

Baris	Tulisan	Artinya
1-3	`# -*- coding: utf-8 -*-` dan teks tiga kutip	Baris 1 memberi tahu Python bahwa berkas memakai huruf UTF-8. Baris 2-3 adalah *docstring* (catatan penjelas di awal berkas, dibahas di Bab 7).
4-6	`import os`, `import sys`, `from qgis.core import ...`	Panggil `os` dan `sys`, lalu dua perkakas QGIS. `Qgis` diimpor tetapi tidak dipakai.
8	`os.environ.setdefault("QT_QPA_PLATFORM", "offscreen")`	Isi pengaturan `QT_QPA_PLATFORM` dengan `offscreen` bila belum ada. Artinya: gambar di layar bayangan, tanpa jendela.
9	`qgs = QgsApplication([], False)`	Buat aplikasi QGIS. Isian `False` berarti tanpa tampilan grafis.
10	`qgs.initQgis()`	Nyalakan mesin QGIS.
11	`sys.path.append(...)`	Tambahkan folder plugin QGIS ke daftar tempat Python mencari modul. Di sinilah `processing` berada.
12-13	`import processing`, `from processing.core.Processing import Processing`	Panggil perkakas Processing. Komentar `noqa` memberi tahu pemeriksa gaya bahwa impor di tengah berkas ini disengaja.
15	`Processing.initialize()`	Daftarkan semua alat Processing, sehingga `processing.run(...)` mengenalnya.
17-21	`_grass = ...`, `os.environ["GISBASE"] = _grass`	Cari folder GRASS di dalam instalasi QGIS. Bila ada dan `GISBASE` belum diisi, isi `GISBASE` dengan folder itu. Tanpa ini, alat `grass:` tidak mau jalan di terminal.

Hasil. Berkas ini tidak mencetak apa-apa. Buktinya bisa Anda lihat dengan dua baris pemeriksaan berikut, yang Anda simpan dalam berkas di folder yang sama:

PYTHON
```
import _inisialisasi
import os
from qgis.core import QgsApplication
print('alat GRASS ada:', QgsApplication.processingRegistry().algorithmById('grass:r.watershed') is not None)
print('GISBASE terisi:', bool(os.environ.get('GISBASE')))
```

HASIL
```
alat GRASS ada: True
GISBASE terisi: True
```

Bagian yang boleh diubah. Baris 19, tulisan `"grass84"`. Itu nama folder GRASS di instalasi QGIS 4.0.2 pada komputer penulis. Bila instalasi Anda memakai versi GRASS lain, ganti angkanya.

Jebakan.

- **Versi GRASS berbeda.** Bila folder `grass84` tidak ada di komputer Anda, `GISBASE` tidak terisi. Alat `grass:` lalu gagal dengan `QgsProcessingException: GRASS GIS folder is not configured. Please configure it before running GRASS GIS algorithms.` (diuji dengan mengganti nama folder menjadi yang tidak ada).
- **Tidak ada penutup.** Berkas menyalakan QGIS tetapi tidak pernah mematikannya (tidak ada `exitQgis()`). Untuk skrip pendek, ini tidak masalah.
- **Satu berkas, satu folder.** Skrip lain menambahkan folder skripnya sendiri ke `sys.path` supaya `import _inisialisasi` ketemu. Pindahkan keduanya bersama-sama.

Skrip 25.1: Memasukkan berkas ke PostGIS dengan GDAL

Berkas `m2_01b_impor.py`. Di dalam berkasnya tertulis "M2 Bab 1". Skrip ini memindahkan tujuh GeoPackage ke gudang. Pekerjaannya sama dengan perintah `ogr2ogr`, tetapi ditulis di Python.

Tujuan. Memasukkan tujuh berkas GeoPackage menjadi tujuh tabel di skema `kph` pada PostGIS.

Bahan. Paket `paket-m2` (tujuh GeoPackage), server PostGIS dengan skema `kph` yang sudah dibuat (oleh `m2_01a_skema.sql`, lihat akhir bab), dan variabel `PGHOST`, `PGPORT`, `PGUSER`, `PGDATABASE`. Perintah: `python-qgis.bat m2_01b_impor.py <folder paket-m2>`.

Skrip utuh:

```python
# -*- coding: utf-8 -*-
"""M2 Bab 1: memasukkan berkas GeoPackage ke PostGIS memakai GDAL (setara ogr2ogr).
Penulis: Badar Mubarok Yogaswara
Pemakaian: python-qgis.bat m2_01b_impor.py <folder paket-m2>
Koneksi dibaca dari variabel lingkungan standar PostgreSQL: PGHOST, PGPORT, PGUSER, PGPASSWORD, PGDATABASE.
Kata sandi tidak pernah ditulis di skrip."""
import os
import sys
from osgeo import gdal

gdal.UseExceptions()
paket = sys.argv[1]
tujuan = "PG:dbname=%s" % os.environ.get("PGDATABASE", "kph_contoh")
# (berkas, layer sumber, nama tabel, jenis geometri)
daftar = [("Petak.gpkg", "Petak", "petak", "MULTIPOLYGON"),
          ("Tanah.gpkg", "Tanah", "tanah", "MULTIPOLYGON"),
          ("Jalan.gpkg", "Jalan", "jalan", "MULTILINESTRING"),
          ("Fasilitas.gpkg", "Fasilitas", "fasilitas", "POINT"),
          ("Kejadian.gpkg", "Kejadian", "kejadian", "POINT"),
          ("Stasiun_Hujan.gpkg", "Stasiun_Hujan", "stasiun_hujan", "POINT"),
          ("Batas_Wilayah.gpkg", "Batas_Wilayah", "batas_wilayah", "MULTIPOLYGON")]
for berkas, layer, tabel, gtype in daftar:
    opsi = gdal.VectorTranslateOptions(
        format="PostgreSQL", layerName=tabel, geometryType=gtype, accessMode="overwrite",
        layerCreationOptions=["SCHEMA=kph", "GEOMETRY_NAME=geom", "FID=gid", "SPATIAL_INDEX=GIST",
"OVERWRITE=YES"],
        srcSRS="EPSG:32749", dstSRS="EPSG:32749", reproject=False)
    gdal.VectorTranslate(tujuan, os.path.join(paket, berkas), options=opsi, layers=[layer])
    print("dimasukkan:", tabel)
```

Penjelasan baris demi baris. Nomor baris mengikuti berkas di atas.

Baris	Tulisan	Artinya
1-6	`# -*- coding ...` dan teks tiga kutip	Huruf UTF-8 dan catatan penjelas: tujuan, cara menjalankan, dan pesan bahwa kata sandi tidak ditulis di skrip.
7-9	`import os`, `import sys`, `from osgeo import gdal`	Panggil `os`, `sys`, dan `gdal`, perkakas pembaca dan penulis data geografis.
11	`gdal.UseExceptions()`	Minta GDAL melaporkan masalah sebagai galat Python yang jelas, bukan diam-diam.
12	`paket = sys.argv[1]`	`sys.argv` adalah daftar kata yang Anda ketik di terminal. Isi ke-0 nama skrip, isi ke-1 folder paket data.
13	`tujuan = "PG:dbname=%s" % os.environ.get("PGDATABASE", "kph_contoh")`	Susun alamat tujuan untuk GDAL: `PG:dbname=` lalu nama basis data. Nama diambil dari `PGDATABASE`. Bila tidak diisi, dipakai `kph_contoh`. Alamat server dan pengguna dibaca otomatis dari `PGHOST` dan kawan-kawan.
15-21	`daftar = [(...), (...), ...]`	Daftar berisi tujuh *tuple* (kelompok isi dalam kurung). Tiap tuple memuat empat hal: nama berkas, nama layer, nama tabel tujuan, dan jenis bentuk.
22	`for berkas, layer, tabel, gtype in daftar:`	Ulangi untuk tiap tuple. Empat isinya langsung masuk ke empat kotak bernama.
23-26	`opsi = gdal.VectorTranslateOptions(...)`	Susun pilihan penerjemahan. `format="PostgreSQL"` tujuan; `layerName=tabel` nama tabel; `geometryType=gtype` jenis bentuk; `accessMode="overwrite"` mode tulis ulang.
25	`layerCreationOptions=[...]`	Pilihan pembuatan tabel: `SCHEMA=kph` (laci kph), `GEOMETRY_NAME=geom` (kolom bentuk bernama geom), `FID=gid` (kolom nomor urut bernama gid), `SPATIAL_INDEX=GIST` (indeks spasial, pencari cepat), dan `OVERWRITE=YES` (bila tabel sudah ada, hapus lalu buat ulang).
26	`srcSRS=..., dstSRS=..., reproject=False`	Sistem koordinat asal dan tujuan sama, EPSG:32749 (UTM zona 49 selatan). Tanpa proyeksi ulang.
27	`gdal.VectorTranslate(tujuan, ..., options=opsi, layers=[layer])`	Kerjakan penerjemahan: baca satu layer dari berkas, tulis ke PostGIS.
28	`print("dimasukkan:", tabel)`	Cetak nama tabel yang selesai.

Hasil. Layar menampilkan tujuh baris:

```
HASIL
dimasukkan: petak
dimasukkan: tanah
dimasukkan: jalan
dimasukkan: fasilitas
dimasukkan: kejadian
dimasukkan: stasiun_hujan
dimasukkan: batas_wilayah
```

Periksa di basis data. Jumlah baris tabel (dihitung dengan `psql`) adalah petak 25, tanah 8, jalan 37, fasilitas 6, kejadian 150, stasiun_hujan 30, dan batas_wilayah 1.

```
SQL
SELECT count(*) FROM kph.petak;
```

Hasilnya `25`.

Bagian yang boleh diubah.

- **Baris 13, nama basis data pengganti.** Ganti `"kph_contoh"` dengan nama basis data Anda, atau isi `PGDATABASE`.
- **Baris 15-21, daftar.** Tambah atau hapus tuple untuk memasukkan berkas lain. Isi keempat bagian tuple.
- **Baris 25, nama skema.** Ganti `SCHEMA=kph` bila skema Anda bernama lain.
- **Baris 25, pengaman ulang.** `OVERWRITE=YES` membuat skrip aman dijalankan berulang. Hapus pilihan ini bila Anda ingin skrip berhenti ketika tabel sudah ada.
- **Baris 26, sistem koordinat.** Ganti kode `EPSG:32749` bila data Anda berada di zona lain.

Jebakan.

- **Lupa argumen folder.** Tanpa `<folder paket-m2>`, baris 12 berhenti dengan `IndexError: list index out of range`.
- **Skema belum ada.** Bila skema `kph` belum dibuat, GDAL berhenti dengan `RuntimeError: Terminating translation prematurely ... May be caused by: Schema "kph" does not exist.` Jalankan `m2_01a_skema.sql` dulu.
- **Dijalankan dua kali.** Berkat `OVERWRITE=YES` di baris 25, jalan kedua berhasil: tabel lama diganti dengan isi berkas asal. Akibatnya, nilai yang sudah Anda ubah langsung di tabel akan hilang (diuji: nilai yang sengaja diubah kembali ke semula). Tanpa pilihan itu, jalan kedua **gagal** dengan `RuntimeError ... Layer kph.petak already exists, CreateLayer failed. Use the layer creation option OVERWRITE=YES to replace it.` Pilihan `accessMode="overwrite"` saja tidak cukup. Percobaannya ada di latihan variasi nomor 2.
- **Nama basis data salah.** Pada komputer penulis, server berbahasa Indonesia mengirim pesan dengan huruf yang bukan UTF-8, sehingga yang tampil `UnicodeDecodeError: 'utf-8' codec can't decode byte 0xab ...`. Penyebab sebenarnya: basis data `PGDATABASE` tidak ada. Cek namanya dengan `psql`.
- **Sandi.** Skrip tidak mengisi kata sandi. Bila server memintanya, siapkan berkas `pgpass`.

Latihan variasi.

1. Masukkan hanya `Petak.gpkg`, dengan nama tabel `petak_uji`.
2. Buktikan sendiri gunanya `OVERWRITE=YES`: hapus pilihan itu, lalu jalankan skrip dua kali.

Contoh jawaban 1: ganti baris 15-21 dengan satu tuple. Tabel `kph.petak_uji` terbentuk dan `SELECT count(*) FROM kph.petak_uji;` menghasilkan `25`.

```python
daftar = [("Petak.gpkg", "Petak", "petak_uji", "MULTIPOLYGON")]
```

Contoh jawaban 2: pakai daftar semula, lalu ubah baris 25 menjadi seperti berikut (tanpa `OVERWRITE=YES`). Jalan pertama pada basis data kosong berhasil. Jalan kedua gagal dengan `Layer kph.petak already exists`, dan tabel dari jalan pertama tetap utuh. Kembalikan `"OVERWRITE=YES"` untuk memulihkan skrip.

```python
        layerCreationOptions=["SCHEMA=kph", "GEOMETRY_NAME=geom", "FID=gid", "SPATIAL_INDEX=GIST"],
```

> **Pola yang bisa dipakai ulang:** Memasukkan banyak berkas ke basis data selalu berbentuk: **daftar tuple (berkas, layer, tabel, jenis bentuk)**, **satu putaran** `for`, **satu panggilan** `gdal.VectorTranslate`. Ganti tiga hal: isi daftar, nama skema, dan pilihan tulis ulang.

Skrip 25.2: Memuat tabel PostGIS di QGIS dengan skrip

Berkas `m2_01c_qgis_muat.py`. Skrip ini membalik arah: kini QGIS yang membaca dari gudang. Ia juga menunjukkan cara memuat sebagian data dengan saringan SQL, dan mengekspor satu layer lewat alat Processing.

Tujuan. Memuat tabel `kph.petak` sebagai layer QGIS, membuat layer berisi petak jati saja, lalu mengekspor `Fasilitas.gpkg` ke tabel baru.

Bahan. Hasil Skrip 25.1 (tabel `kph.petak` sudah ada), `Fasilitas.gpkg`, empat variabel `PG*`. Jalankan dari terminal dengan folder paket sebagai argumen.

Skrip utuh:

```python
# -*- coding: utf-8 -*-
"""M2 Bab 1: memuat tabel PostGIS sebagai layer lewat PyQGIS, dan mengekspor layer ke PostGIS lewat Processing.
Penulis: Badar Mubarok Yogaswara. Koneksi dari variabel lingkungan PGHOST, PGPORT, PGUSER, PGDATABASE (tanpa
kata sandi di skrip)."""
import os
import sys
sys.path.insert(0, os.path.dirname(os.path.abspath(__file__)))
import _inisialisasi  # noqa: F401  (hanya perlu bila dijalankan di luar jendela QGIS)
from qgis.core import QgsDataSourceUri, QgsVectorLayer, QgsProject
import processing

paket = sys.argv[1]
uri = QgsDataSourceUri()
uri.setConnection(os.environ["PGHOST"], os.environ["PGPORT"], os.environ["PGDATABASE"], os.environ["PGUSER"],
"")
uri.setDataSource("kph", "petak", "geom", "", "gid")          # skema, tabel, kolom geometri, filter, kolom
kunci
```

```
lyr = QgsVectorLayer(uri.uri(False), "Petak (PostGIS)", "postgres")
print("sah:", lyr.isValid(), "| jumlah fitur:", lyr.featureCount(), "| CRS:", lyr.crs().authid(), "|
geometri:", lyr.wkbType())
if not lyr.isValid():
    sys.exit("Layer kph.petak tidak sah: periksa PGHOST, PGPORT, PGUSER, PGDATABASE dan skema kph.")

# layer dengan filter SQL (hanya petak jati)
uri.setSql("jenis_tegakan = 'Jati'")
lyr2 = QgsVectorLayer(uri.uri(False), "Petak jati", "postgres")
print("petak jati:", lyr2.featureCount())
QgsProject.instance().addMapLayer(lyr)

# ekspor satu layer GeoPackage ke tabel baru lewat Processing (alat GDAL)
hasil = processing.run("gdal:importvectorintopostgisdatabasenewconnection", {
    "INPUT": os.path.join(paket, "Fasilitas.gpkg|layername=Fasilitas"), "GTYPE": 3,
    "HOST": os.environ["PGHOST"], "PORT": os.environ["PGPORT"], "USER": os.environ["PGUSER"], "DBNAME":
os.environ["PGDATABASE"],
    "PASSWORD": "", "SCHEMA": "kph", "TABLE": "fasilitas_uji", "PK": "gid", "GEOCOLUMN": "geom", "DIM": 0,
    "OVERWRITE": True, "APPEND": False, "ADDFIELDS": False, "LAUNDER": False, "INDEX": False, "SKIPFAILURES":
False,
    "MAKEVALID": False, "PROMOTETOMULTI": False, "PRECISION": True})
print("ekspor selesai:", hasil)
```

Penjelasan baris demi baris. Nomor baris mengikuti berkas di atas.

Baris	Tulisan	Artinya
1-3	`# -*- coding ...` dan teks tiga kutip	Huruf UTF-8 dan catatan penjelas. Koneksi dari variabel `PG*`, tanpa kata sandi di skrip.
4-5	`import os`, `import sys`	Panggil dua perkakas dasar.
6	`sys.path.insert(0, os.path.dirname(os.path.abspath(__file__)))`	Tambahkan folder tempat skrip ini berada ke daftar pencarian modul, supaya `_inisialisasi` ditemukan. `__file__` adalah jalur berkas skrip ini.
7	`import _inisialisasi`	Panggil berkas pembantu, yang menyalakan QGIS. Komentar `noqa` berarti impor ini disengaja walau tidak dipakai langsung.
8-9	`from qgis.core import ...`, `import processing`	Panggil tiga perkakas QGIS dan Processing.
11	`paket = sys.argv[1]`	Ambil folder paket dari terminal.
12-14	`uri = QgsDataSourceUri()` ... `setDataSource(...)`	Siapkan "formulir alamat" sumber data. `setConnection` mengisi server, port, basis data, pengguna, dan sandi kosong (`""`). `setDataSource` mengisi skema, tabel, kolom bentuk, filter (kosong), dan kolom kunci.
15	`QgsVectorLayer(uri.uri(False), "Petak (PostGIS)", "postgres")`	Buka layer. Tiga isian: alamat sumber, nama layer, dan pembaca `"postgres"`. Isian `False` pada `uri(...)` artinya jangan memperluas konfigurasi autentikasi tersimpan.
16	`print("sah:", lyr.isValid(), ...)`	Cetak: apakah layer sah, jumlah fitur, sistem koordinat (`crs().authid()`), dan kode jenis bentuk (`wkbType()`).
17-18	`if not lyr.isValid(): sys.exit("Layer kph.petak tidak sah: ...")`	**Pengaman.** Bila layer tidak sah (misalnya basis data tidak ada), cetak pesan lalu hentikan skrip. `sys.exit(teks)` berarti "berhenti sekarang". Teksnya muncul sebagai pesan galat, dan program berakhir dengan kode 1 (tanda gagal). Tanpa pengaman ini, baris-baris di bawahnya tetap jalan.

Baris	Tulisan	Artinya
21-23	`uri.setSql("jenis_tegakan = 'Jati'")`	Pasang saringan SQL pada formulir, lalu buka layer kedua. Hanya petak jati yang ikut. Cetak jumlahnya.
24	`QgsProject.instance().addMapLayer(lyr)`	Masukkan layer pertama ke proyek. Layer kedua tidak dimasukkan.
27-32	`processing.run("gdal:importvectorintopostgisdatabasenewconnection", {...})`	Panggil alat ekspor ke PostgreSQL. `INPUT` adalah jalur berkas yang disambung tanda garis tegak dan `layername=Fasilitas`. `GTYPE: 3` berarti titik (POINT).
29	`"HOST": os.environ["PGHOST"], ...`	Isi server, port, pengguna, dan basis data dari variabel lingkungan.
30	`"SCHEMA": "kph", "TABLE": "fasilitas_uji", "PK": "gid", "GEOCOLUMN": "geom"`	Tujuan: tabel `fasilitas_uji` di skema `kph`, kolom kunci `gid`, kolom bentuk `geom`.
31-32	`"OVERWRITE": True, "APPEND": False, ...`	Pilihan alat. Tabel lama ditimpa (`OVERWRITE`), tidak ditambah (`APPEND`). Pilihan lain dimatikan.
33	`print("ekspor selesai:", hasil)`	Cetak hasil alat.

Hasil. Layar menampilkan:

```
HASIL
sah: True | jumlah fitur: 25 | CRS: EPSG:32749 | geometri: 6
petak jati: 10
ekspor selesai: {}
```

Angka `6` pada baris pertama adalah kode jenis bentuk *MultiPolygon* (poligon majemuk). Kamus kosong `{}` di baris terakhir wajar, karena alat ekspor tidak punya keluaran. Buktinya, tabel `kph.fasilitas_uji` terisi 6 baris (dicek dengan `psql`). Pada uji ini, QGIS juga mencetak peringatan `DeprecationWarning` (fungsi lama). Peringatan itu boleh diabaikan.

Bagian yang boleh diubah.

- **Baris 14, skema dan tabel.** Ganti `"kph", "petak"` untuk memuat tabel lain.
- **Baris 18, pesan pengaman.** Ganti teks di dalam `sys.exit(...)` dengan kalimat yang paling membantu Anda.
- **Baris 21, saringan.** Ganti `jenis_tegakan = 'Jati'` dengan syarat SQL lain.
- **Baris 28, berkas ekspor.** Ganti berkas dan nama layer.
- **Baris 30, nama tabel.** Ganti `fasilitas_uji` agar tidak menimpa tabel lama.

Jebakan.

- **Variabel belum diisi.** Bila `PGHOST` kosong, baris 13 berhenti dengan `KeyError: 'PGHOST'`.
- **Basis data tidak ada.** Layer tidak sah, jadi layar menampilkan `sah: False | jumlah fitur: 0`. Lalu baris 17-18 menghentikan skrip dengan pesan `Layer kph.petak tidak sah: periksa PGHOST, PGPORT, PGUSER, PGDATABASE dan skema kph.` Tahap ekspor tidak

sempat jalan. Baca baris `sah:` dan pesan itu, lalu periksa keempat variabel `PG*` serta skema `kph`. (Diuji dengan nama basis data yang tidak ada.)

- **Layer kedua tidak dimuat.** `lyr2` hanya dihitung. Tambahkan `addMapLayer(lyr2)` bila ingin melihatnya di peta.
- **Menimpa tabel.** `OVERWRITE: True` mengganti tabel `fasilitas_uji` bila sudah ada.
- **Hanya terminal.** `sys.argv[1]` tidak ada di Konsol QGIS. Di Konsol, ganti baris 6-7 dan 11 dengan jalur manual. [CEK]

Latihan variasi.

1. Muat hanya petak mahoni.
2. Tambahkan layer petak jati ke peta.

Contoh jawaban 1: ganti baris 21 dengan satu baris berikut. Layar menampilkan `petak jati: 5` (tulisan `petak jati` ikut tercetak apa adanya, sebab baris 23 tidak diganti).

```
PYTHON
uri.setSql("jenis_tegakan = 'Mahoni'")
```

Contoh jawaban 2: tambahkan satu baris setelah baris 24.

```
PYTHON
QgsProject.instance().addMapLayer(lyr2)
```

> **Pola yang bisa dipakai ulang:** Memuat tabel PostGIS punya empat langkah: **isi formulir alamat** (`QgsDataSourceUri`), **buka layer** dengan pembaca `"postgres"`, **berhenti bila layer tidak sah** (`sys.exit`), **tambahkan saringan SQL bila perlu** (`setSql`). Ganti tiga hal: skema, tabel, dan saringan.

Skrip 25.3: Kueri yang sama di GeoPackage dan PostGIS

Berkas `m2_02c_silang_geopackage.py`. "Silang" berarti mencocokkan dua cara. Skrip ini menjalankan tiga pertanyaan spasial di GeoPackage, lalu (bila PostGIS ada) di PostGIS, dan membandingkan jawabannya. Bila angkanya sama, Anda boleh percaya keduanya.

Tujuan. Menjawab tiga pertanyaan dengan SQL di GeoPackage, lalu membandingkan dengan jawaban PostGIS.

Bahan. `paket-m2` (tujuh GeoPackage) dan folder `hasil` yang sudah ada. PostGIS dengan hasil Skrip 25.1 bersifat pilihan. Perintah: `python-qgis.bat m2_02c_silang_geopackage.py <folder paket-m2>`.

Skrip utuh:

```
PYTHON
# -*- coding: utf-8 -*-
```

```
"""M2 Bab 2: menjalankan SQL spasial yang sama pada GeoPackage (dialek SQLite/SpatiaLite lewat GDAL),
lalu membandingkan angkanya dengan hasil PostGIS. Penulis: Badar Mubarok Yogaswara.
Pemakaian: python-qgis.bat m2_02c_silang_geopackage.py <folder paket-m2>
Bila PostGIS tidak tersedia, bagian pembanding dilewati (hanya mencetak angka GeoPackage)."""
import os
import sys
from osgeo import gdal, ogr

gdal.UseExceptions()
ogr.UseExceptions()
paket = sys.argv[1]
gpkg = os.path.join(paket, "hasil", "kph_contoh.gpkg")

# 1) kumpulkan semua layer ke satu GeoPackage (sekali saja)
if not os.path.exists(gpkg):
    for berkas, layer in [("Petak", "Petak"), ("Tanah", "Tanah"), ("Jalan", "Jalan"), ("Fasilitas",
"Fasilitas"),
                          ("Kejadian", "Kejadian"), ("Stasiun_Hujan", "Stasiun_Hujan"), ("Batas_Wilayah",
"Batas_Wilayah")]:
        gdal.VectorTranslate(gpkg, os.path.join(paket, berkas + ".gpkg"),
                             options=gdal.VectorTranslateOptions(format="GPKG", layerName=layer,
accessMode="append"))
ds = gdal.OpenEx(gpkg)


def tanya(sql):
    lyr = ds.ExecuteSQL(sql, dialect="SQLite")
    baris = [tuple(f.GetField(i) for i in range(f.GetFieldCount())) for f in lyr]
    ds.ReleaseResultSet(lyr)
    return baris


sqlite = {
    "total_ha": tanya("SELECT ROUND(SUM(ST_Area(geom)) / 10000.0, 2) FROM Petak")[0][0],
    "per_petak_teratas": tanya("SELECT p.kode, COUNT(k.fid) AS n FROM Petak p LEFT JOIN Kejadian k ON
ST_Intersects(p.geom, k.geom) "
                               "GROUP BY p.kode ORDER BY n DESC, p.kode LIMIT 3"),
    "tebangan_dekat_jalan": tanya("SELECT COUNT(*) FROM Kejadian k WHERE k.jenis = 'Tebangan liar' AND "
                                  "EXISTS (SELECT 1 FROM Jalan j WHERE ST_Distance(k.geom, j.geom) <= 100)")[0]
[0],
}
print("GeoPackage/SpatiaLite:", sqlite)

try:
    import psycopg2
    con = psycopg2.connect(dbname=os.environ["PGDATABASE"])      # host, port, user dari PGHOST, PGPORT, PGUSER
    cur = con.cursor()
    cur.execute("SELECT ROUND((SUM(ST_Area(geom)) / 10000.0)::numeric, 2) FROM kph.petak")
    pg_total = float(cur.fetchone()[0])
    cur.execute("SELECT p.kode, COUNT(k.gid) n FROM kph.petak p LEFT JOIN kph.kejadian k ON
ST_Intersects(p.geom, k.geom) "
                "GROUP BY p.kode ORDER BY n DESC, p.kode LIMIT 3")
    pg_top = [tuple(r) for r in cur.fetchall()]
    cur.execute("SELECT COUNT(*) FROM kph.kejadian k WHERE k.jenis = 'Tebangan liar' AND "
                "EXISTS (SELECT 1 FROM kph.jalan j WHERE ST_DWithin(k.geom, j.geom, 100))")
    pg_dekat = cur.fetchone()[0]
    print("PostGIS              :", {"total_ha": pg_total, "per_petak_teratas": pg_top, "tebangan_dekat_jalan":
pg_dekat})
    cocok = (abs(pg_total - sqlite["total_ha"]) < 0.01 and pg_top == sqlite["per_petak_teratas"] and pg_dekat
== sqlite["tebangan_dekat_jalan"])
    print("HASIL SAMA:", cocok)
except Exception as galat:                                       # PostGIS tidak ada atau tidak terhubung
    print("Pembanding PostGIS dilewati:", galat)
```

Penjelasan baris demi baris. Skrip ini punya tiga bagian: menyiapkan satu berkas gabungan, menjawab di GeoPackage, lalu menjawab di PostGIS. Nomor baris mengikuti berkas di atas.

Baris	Tulisan	Artinya
1-5	`# -*- coding ...` dan teks tiga kutip	Catatan penjelas, termasuk pesan: tanpa PostGIS, bagian pembanding dilewati.
6-8	`import os`, `import sys`, `from osgeo import gdal, ogr`	Panggil perkakas, termasuk `ogr` untuk membaca data vektor.
10-11	`gdal.UseExceptions()`, `ogr.UseExceptions()`	Minta GDAL dan OGR melaporkan masalah sebagai galat Python.
12-13	`paket = sys.argv[1]`, `gpkg = os.path.join(paket, "hasil", "kph_contoh.gpkg")`	Ambil folder paket, lalu susun jalur berkas gabungan di folder `hasil`.
15-20	`if not os.path.exists(gpkg):` `for ...` `gdal.VectorTranslate(...)`	Bila berkas gabungan belum ada, gabungkan tujuh layer ke dalamnya, satu per satu (`accessMode="append"` berarti tambahkan). Baris 17-18 adalah daftar pasangan (berkas, layer).
21	`ds = gdal.OpenEx(gpkg)`	Buka berkas gabungan sebagai *dataset* (kumpulan layer).
24-28	`def tanya(sql): ...`	Buat fungsi `tanya` yang menjalankan satu kueri dan mengembalikan hasilnya.
25	`lyr = ds.ExecuteSQL(sql, dialect="SQLite")`	Jalankan SQL dengan dialek SQLite, yang mengerti fungsi spasial seperti `ST_Area`.
26	`baris = [tuple(f.GetField(i) for i in range(f.GetFieldCount())) for f in lyr]`	Untuk tiap baris hasil `f`, kumpulkan isi semua kolomnya menjadi satu tuple. Semua tuple dikumpulkan dalam daftar `baris`.
27-28	`ds.ReleaseResultSet(lyr)`, `return baris`	Lepaskan hasil kueri, lalu kembalikan daftar.
31-37	`sqlite = {...}`	Kamus berisi tiga jawaban. Tulisan `[0][0]` mengambil isi baris pertama, kolom pertama.
38	`print("GeoPackage/SpatiaLite:", sqlite)`	Cetak ketiga jawaban.
40-41	`try:`, `import psycopg2`	Mulai blok percobaan. `psycopg2` adalah penyambung Python ke PostgreSQL.
42-43	`con = psycopg2.connect(dbname=os.environ["PGDATABASE"])`, `cur = con.cursor()`	Sambung ke basis data (server dan pengguna dari `PGHOST` dan kawan-kawan). *Kursor* adalah pena untuk menulis kueri.
44-45	`cur.execute("SELECT ROUND(...)::numeric ...")`	Jalankan kueri total luas di PostGIS. `::numeric` mengubah jenis angka supaya `ROUND` bekerja. `float(cur.fetchone()[0])` mengambil satu hasil.
46-51	`cur.execute(...)` dua kali	Dua kueri berikutnya, setara dengan kueri GeoPackage. Di PostGIS, jarak ditulis `ST_DWithin(..., 100)`.

Baris	Tulisan	Artinya
52	`print("PostGIS ...", {...})`	Cetak jawaban PostGIS dalam bentuk kamus.
53-54	`cocok = (...), print("HASIL SAMA:", cocok)`	Bandingkan: selisih total di bawah 0,01 dan dua jawaban lain persis sama. Cetak `True` atau `False`.
55-56	`except Exception as galat: print(...)`	Bila ada galat apa pun di blok `try`, lewati dan cetak pesannya.

Tiga pertanyaan dalam bahasa biasa: (1) berapa hektare luas seluruh petak? (2) tiga petak mana yang memuat kejadian terbanyak? (3) berapa kejadian "Tebangan liar" yang jaraknya 100 meter atau kurang dari jalan? Baris 33 memakai `LEFT JOIN`: semua petak ikut, walau tanpa kejadian. `GROUP BY` mengelompokkan per petak, `ORDER BY ... LIMIT 3` mengambil tiga teratas.

Hasil. Dengan PostGIS menyala, layar menampilkan:

```
HASIL
GeoPackage/SpatiaLite: {'total_ha': 400.0, 'per_petak_teratas': [('P-04', 13), ('P-09', 12), ('P-12', 12)],
'tebangan_dekat_jalan': 59}
PostGIS              : {'total_ha': 400.0, 'per_petak_teratas': [('P-04', 13), ('P-09', 12), ('P-12', 12)],
'tebangan_dekat_jalan': 59}
HASIL SAMA: True
```

Tanpa PostGIS (misalnya server mati), baris GeoPackage tetap tampil, lalu `Pembanding PostGIS dilewati:` diikuti pesan galat.

Bagian yang boleh diubah.

- **Baris 32-36, isi kueri.** Ganti dengan pertanyaan Anda, lalu ubah kueri PostGIS yang setara (baris 44-51).
- **Baris 36 dan 50, ambang jarak.** Angka `100` (meter) ada di **dua tempat**. Ganti keduanya bersama-sama.
- **Baris 53, toleransi.** Angka `0.01` menentukan seberapa kecil selisih luas yang dianggap sama.

Jebakan.

- **Folder hasil tidak ada.** Skrip tidak membuat folder `hasil`. Penggabungan di baris 19-20 berhenti dengan `RuntimeError: GPKG driver failed to create ...hasil\kph_contoh.gpkg ... unable to open database file.`
- **Server tidak terjangkau.** Pesan yang tampil: `Pembanding PostGIS dilewati: connection to server at "localhost" ... failed: Connection refused.` Perhatikan, skrip tetap berjalan.
- **Penjaring galat terlalu lebar.** Baris `except Exception` menangkap semuanya. Salah ketik di baris 41-54 pun tampil sebagai "dilewati". Baca pesan setelah titik dua.
- **Pesan menyesatkan di komputer berbahasa Indonesia.** Basis data yang tidak ada bisa menampilkan `'utf-8' codec can't decode byte 0xab` (pesan server bukan UTF-8), bukan pesan "database tidak ada".
- **Sambungan tidak ditutup.** `con.close()` tidak ada. Untuk skrip pendek tidak masalah.

Latihan variasi. Ganti ambang jarak dari 100 menjadi 50 meter di kedua kueri. Contoh jawaban: ganti `<= 100` menjadi `<= 50` di baris 36, dan `ST_DWithin(k.geom, j.geom, 100)` menjadi `50` di baris 50. Layar menampilkan `'tebangan_dekat_jalan': 46` di baris GeoPackage dan PostGIS, dan `HASIL SAMA: True`.

> **Pola yang bisa dipakai ulang:** Silang dua sistem: **tulis pertanyaan yang sama di dua dialek**, **ambil hasilnya jadi data Python**, lalu **bandingkan** dengan `==` (angka desimal dengan selisih kecil). Ganti tiga hal: isi pertanyaan, ambang jarak (di dua tempat), dan toleransi.

Skrip 25.4: Hitungan yang sama tanpa basis data

Berkas `m2_02d_qgis_setara.py`. Tanpa PostGIS pun Anda bisa menghitung jumlah kejadian per petak. Skrip ini memakai dua alat Processing QGIS, lalu membandingkan hasilnya. Skrip ini juga memperkenalkan **lambda**.

Tujuan. Menghitung kejadian per petak dengan dua cara di QGIS dan memeriksa bahwa hasilnya sama.

Bahan. `Petak.gpkg` dan `Kejadian.gpkg` dari `paket-m2`. Tidak butuh PostGIS. Tidak menulis berkas.

Konsep singkat: lambda. *Lambda* adalah fungsi mini yang ditulis dalam satu baris, tanpa nama resmi. Bayangkan secarik catatan berisi satu aturan: "ambil angka ini, lalu hitung begini". Bentuknya `lambda masukan: hasil`. Di skrip ini lambda dipakai sebagai aturan pengurutan.

Skrip utuh:

PYTHON
```
# -*- coding: utf-8 -*-
"""M2 Bab 2: padanan Q2 (kejadian per petak) lewat alat Processing QGIS tanpa basis data:
(a) Count points in polygon, (b) Execute SQL (kueri pada layer maya, dialek SQLite; layer masukan bernama
input1, input2, dan seterusnya).
Penulis: Badar Mubarok Yogaswara. Pemakaian: python-qgis.bat m2_02d_qgis_setara.py <folder paket-m2>"""
import os
import sys
sys.path.insert(0, os.path.dirname(os.path.abspath(__file__)))
import _inisialisasi  # noqa: F401
import processing
from qgis.core import QgsVectorLayer

paket = sys.argv[1]
petak = QgsVectorLayer(os.path.join(paket, "Petak.gpkg|layername=Petak"), "Petak", "ogr")
kej = QgsVectorLayer(os.path.join(paket, "Kejadian.gpkg|layername=Kejadian"), "Kejadian", "ogr")

a = processing.run("native:countpointsinpolygon", {"POLYGONS": petak, "POINTS": kej, "FIELD": "jumlah",
"OUTPUT": "memory:"})["OUTPUT"]
hasil_a = sorted(((f["kode"], f["jumlah"]) for f in a.getFeatures()), key=lambda t: (-t[1], t[0]))[:3]
print("Count points in polygon:", hasil_a)

sql = ("SELECT p.kode AS kode, COUNT(k.id) AS jumlah FROM input1 p LEFT JOIN input2 k ON
ST_Intersects(p.geometry, k.geometry) "
       "GROUP BY p.kode ORDER BY jumlah DESC, p.kode LIMIT 3")
b = processing.run("qgis:executesql", {"INPUT_DATASOURCES": [petak, kej], "INPUT_QUERY": sql,
"INPUT_UID_FIELD": "",
                                       "INPUT_GEOMETRY_FIELD": "", "INPUT_GEOMETRY_TYPE": 1,
"INPUT_GEOMETRY_CRS": None,
                                       "OUTPUT": "memory:"})["OUTPUT"]
hasil_b = [(f["kode"], f["jumlah"]) for f in b.getFeatures()]
print("Execute SQL           :", hasil_b)
```

```
print("SAMA:", hasil_a == hasil_b)
```

Penjelasan baris demi baris. Nomor baris mengikuti berkas di atas.

Baris	Tulisan	Artinya
1-4	`# -*- coding ...` dan teks tiga kutip	Catatan penjelas: dua cara menghitung, dan bahwa layer masukan di SQL bernama `input1`, `input2`, dan seterusnya.
5-10	`import os`, `import sys`, `sys.path.insert(...)`, `import _inisialisasi`, …	Panggil perkakas dan nyalakan QGIS, seperti Skrip 25.2.
12	`paket = sys.argv[1]`	Folder paket dari terminal.
13-14	`petak = QgsVectorLayer(...)`, `kej = QgsVectorLayer(...)`	Buka dua layer dari GeoPackage dengan pembaca `"ogr"`.
16	`processing.run("native:countpointsinpolygon", {...})["OUTPUT"]`	Alat pertama: hitung titik di dalam poligon. Hasilnya layer dengan kolom `jumlah`. `memory:` di sini aman karena hasilnya vektor.
17	`sorted(((f["kode"], f["jumlah"]) for f in a.getFeatures()), key=lambda t: (-t[1], t[0]))[:3]`	Dari layer hasil, bentuk pasangan (kode petak, jumlah), urutkan, lalu ambil tiga teratas lewat `[:3]`. Aturan urut ditulis dengan lambda (lihat di bawah).
18	`print("Count points in polygon:", hasil_a)`	Cetak tiga teratas cara pertama.
20-21	`sql = ("SELECT ... FROM input1 p LEFT JOIN input2 k ...")`	Susun teks SQL dalam kurung, dipecah dua baris agar rapi. `input1` adalah layer pertama dan `input2` layer kedua dari daftar di baris 22.
22-24	`processing.run("qgis:executesql", {...})["OUTPUT"]`	Alat kedua: jalankan SQL pada layer-layer QGIS. Isian penting: `INPUT_DATASOURCES` (daftar layer, urutannya menentukan `input1`, `input2`), `INPUT_QUERY` (teks SQL), `INPUT_GEOMETRY_TYPE: 1` (artinya tanpa bentuk, sebab hasilnya hanya tabel).
25	`hasil_b = [(f["kode"], f["jumlah"]) for f in b.getFeatures()]`	Kumpulkan pasangan (kode, jumlah) dari hasil SQL.
26-27	`print(...)`, `print("SAMA:", hasil_a == hasil_b)`	Cetak hasil dan bandingkan dua daftar dengan `==`.

Cara membaca baris 17. `key=lambda t: (-t[1], t[0])` berarti: untuk tiap pasangan `t`, kunci urutnya adalah `(-t[1], t[0])`. `t[1]` adalah jumlah. Tanda minus membalik urutan, sehingga jumlah terbesar tampil lebih dulu. `t[0]` adalah kode petak, dipakai bila jumlahnya sama. Tanpa tanda minus, urutannya dari terkecil.

Hasil. Layar menampilkan:

```
HASIL
Count points in polygon: [('P-04', 13.0), ('P-09', 12.0), ('P-12', 12.0)]
Execute SQL          : [('P-04', 13), ('P-09', 12), ('P-12', 12)]
SAMA: True
```

Angka cara pertama berbentuk `13.0` (desimal), cara kedua `13` (bulat). Python menganggap `13.0 == 13`, jadi hasilnya tetap `True`. QGIS juga mencetak peringatan `DeprecationWarning` yang boleh diabaikan.

Bagian yang boleh diubah.

- **Baris 16, nama kolom hitung.** Ganti `"jumlah"` dengan nama lain, dan ganti juga di baris 17 dan 21.
- **Baris 17 dan 21, angka 3.** `[:3]` dan `LIMIT 3` menentukan berapa petak teratas. Ganti keduanya bersama-sama.
- **Baris 20-21, SQL.** Ganti pertanyaannya. Tetap pakai `input1`, `input2`.

Jebakan.

- **Urutan layer dan nama masukan.** Nama `input1` selalu layer pertama di daftar baris 22. Tukar urutannya, tukar pula maknanya.
- **Pecah hasil seri.** Dua petak sama-sama 12 kejadian. Tanpa `p.kode` di urutan SQL dan `t[0]` di lambda, urutan kedua cara bisa berbeda dan `SAMA` jadi `False`.
- **Jenis bentuk hasil.** Isian `INPUT_GEOMETRY_TYPE` bernilai `1`, yang berarti tanpa bentuk. Daftar pilihan alat di QGIS 4.0.2: `0` otomatis, `1` tanpa bentuk, `2` titik, `3` garis, `4` poligon, lalu `5` sampai `7` untuk versi majemuk. Untuk hasil yang membawa bentuk, pilih nilai yang sesuai.

Latihan variasi. Tampilkan lima petak teratas. Contoh jawaban: ganti `[:3]` menjadi `[:5]` di baris 17 dan `LIMIT 3` menjadi `LIMIT 5` di baris 21. Layar menampilkan lima pasangan di tiap cara, lalu `SAMA: True`:

```
HASIL
Count points in polygon: [('P-04', 13.0), ('P-09', 12.0), ('P-12', 12.0), ('P-08', 9.0), ('P-01', 7.0)]
Execute SQL          : [('P-04', 13), ('P-09', 12), ('P-12', 12), ('P-08', 9), ('P-01', 7)]
SAMA: True
```

> **Pola yang bisa dipakai ulang:** Menghitung ulang dengan dua alat: **jalankan alat A**, **jalankan alat B dengan SQL**, **ubah keduanya menjadi daftar pasangan**, lalu **bandingkan** dengan `==`. Ganti tiga hal: nama layer, kolom hitung, dan jumlah teratas.

Skrip 25.5: Rute terpendek dan tercepat di QGIS

Berkas `m2_03a_rute_qgis.py`. Kepala Seksi ingin tahu jalur patroli dari Pos Jaga 1 ke Pos Jaga 4. Skrip ini mencarinya dengan alat bawaan QGIS, lalu menguji apa yang terjadi bila satu ruas jalan ditutup atau bila jalannya bercelah.

Tujuan. Menghitung rute terpendek (dalam meter) dan tercepat (dalam menit) antar dua pos, dalam lima skenario.

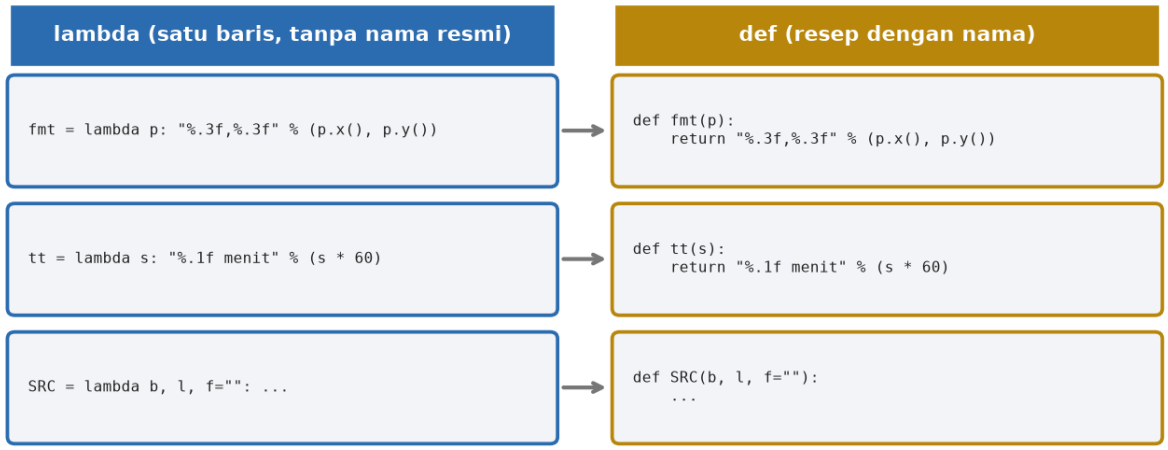
Bahan. `Fasilitas.gpkg` (titik "Pos Jaga 1" dan "Pos Jaga 4"), `Jalan.gpkg` (kolom `status` dan `kecepatan_kmj`), dan `Jalan_celah.gpkg` (jalan yang sengaja dibuat bercelah). Tidak butuh PostGIS. Tidak menulis berkas.

Konsep: jaringan dan biaya. Jaringan jalan itu seperti peta jalur kereta: ada simpul (persimpangan atau ujung jalan) dan ruas (jalan di antara simpul). Tiap ruas punya **biaya** (*cost*), yaitu panjangnya atau waktu tempuhnya. Rute terbaik adalah jalur dengan jumlah biaya terkecil. Dua istilah lain: **strategi** (apakah biaya yang diminimalkan panjang atau waktu) dan **toleransi** (jarak terjauh dua ujung jalan masih dianggap menyambung).

Skrip ini juga memakai tiga lambda. Gambar berikut menunjukkan bahwa lambda hanyalah cara singkat menulis fungsi.

Ilustrasi 25.3: lambda dan def

Skema; bukan tangkapan layar



Hasilnya sama. Kata lambda = "fungsi mini": nama parameter, titik dua, lalu satu ekspresi yang langsung menjadi hasil.

Skrip utuh:

```python
# -*- coding: utf-8 -*-
"""M2 Bab 3: rute terpendek dan rute tercepat dengan alat bawaan QGIS (Processing, algoritma native). Area layanan: lihat m2_03c.
Penulis: Badar Mubarok Yogaswara. Pemakaian: python-qgis.bat m2_03a_rute_qgis.py <folder paket-m2>"""
import os
import sys
sys.path.insert(0, os.path.dirname(os.path.abspath(__file__)))
import _inisialisasi  # noqa: F401
import processing
from qgis.core import QgsVectorLayer, QgsPointXY, QgsProcessingFeedback

paket = sys.argv[1]
SRC = lambda b, l, f="": os.path.join(paket, b + ".gpkg|layername=" + l + (("|subset=" + f) if f else ""))
fas = QgsVectorLayer(SRC("Fasilitas", "Fasilitas"), "Fasilitas", "ogr")
titik = {f["nama"]: f.geometry().asPoint() for f in fas.getFeatures()}
awal, akhir = titik["Pos Jaga 1"], titik["Pos Jaga 4"]
fmt = lambda p: "%.3f,%.3f [EPSG:32749]" % (p.x(), p.y())
```

```
def rute(jaringan, strategi=0, toleransi=0.0, a=awal, b=akhir):
    lyr = QgsVectorLayer(jaringan, "jaringan", "ogr")
    hasil = processing.run("native:shortestpathpointtopoint", {
        "INPUT": lyr, "STRATEGY": strategi, "DIRECTION_FIELD": "", "VALUE_FORWARD": "", "VALUE_BACKWARD": "",
"VALUE_BOTH": "",
        "DEFAULT_DIRECTION": 2, "SPEED_FIELD": "kecepatan_kmj", "DEFAULT_SPEED": 20, "TOLERANCE": toleransi,
        "START_POINT": fmt(a), "END_POINT": fmt(b), "OUTPUT": "memory:"})
    return hasil["TRAVEL_COST"], hasil["OUTPUT"]


jalan = SRC("Jalan", "Jalan")
jalan_buka = SRC("Jalan", "Jalan", "status<>'ditutup'")
celah = SRC("Jalan_celah", "Jalan_celah", "status<>'ditutup'")
tt = lambda s: "%.1f menit" % (s * 60)

c, o = rute(jalan)
print("1) terpendek, semua ruas      : %.1f m" % c)
c0, o0 = rute(jalan_buka)
print("2) terpendek, ruas tutup dibuang: %.1f m  (jumlah segmen keluaran: %d)" % (c0, o0.featureCount()))
cf, of = rute(jalan_buka, strategi=1)
print("3) tercepat (jam -> menit)    : %s" % tt(cf))
cg, og = rute(celah)
print("4) jalan ber-celah, toleransi 0: %.1f m" % cg)
cg5, og5 = rute(celah, toleransi=5.0)
print("5) jalan ber-celah, toleransi 5: %.1f m" % cg5)

print("Catatan: titik awal dan akhir diambil dari fasilitas yang berjarak 25 m di luar jalan; QGIS
menempelkannya ke titik jalan terdekat.")
```

Penjelasan baris demi baris. Nomor baris mengikuti berkas di atas. Bagian awal sama dengan Skrip 25.4, jadi dijelaskan singkat.

Baris	Tulisan	Artinya
1-3	`# -*- coding ...` dan teks tiga kutip	Catatan penjelas. Area layanan dibahas di Skrip 25.6.
4-9	`import os`, `import sys`, ..., `from qgis.core import ...`	Panggil perkakas dan nyalakan QGIS. `QgsPointXY` dan `QgsProcessingFeedback` diimpor tetapi tidak dipakai.
11	`paket = sys.argv[1]`	Folder paket dari terminal.
12	`SRC = lambda b, l, f="": ...`	**Lambda pertama.** Menerima nama berkas `b`, nama layer `l`, dan syarat saringan `f` (bawaan kosong). Hasilnya teks sumber data: folder, nama berkas dengan `.gpkg`, tanda garis tegak, `layername=` dan nama layer. Bila `f` tidak kosong, ditambah tanda garis tegak, `subset=`, dan syarat itu.
13	`fas = QgsVectorLayer(SRC("Fasilitas", "Fasilitas"), ...)`	Buka layer fasilitas memakai lambda tadi.
14	`titik = {f["nama"]: f.geometry().asPoint() for f in fas.getFeatures()}`	Buat kamus nama ke titik, dengan satu baris (*dictionary comprehension*). Kuncinya nama fasilitas, isinya letak titik.
15	`awal, akhir = titik["Pos Jaga 1"], titik["Pos Jaga 4"]`	Ambil dua titik dari kamus. Dua nilai masuk ke dua kotak sekaligus.

Baris	Tulisan	Artinya
16	`fmt = lambda p: "%.3f,%.3f [EPSG:32749]" % (p.x(), p.y())`	**Lambda kedua.** Ubah titik menjadi teks koordinat seperti yang diminta QGIS: `x,y [kode sistem koordinat]`, tiga angka desimal.
19	`def rute(jaringan, strategi=0, toleransi=0.0, a=awal, b=akhir):`	Buat fungsi `rute` dengan nilai bawaan. Bila Anda tidak mengisi `strategi`, dipakai `0` (terpendek). Titik `a` dan `b` bawaannya `awal` dan `akhir`.
20	`lyr = QgsVectorLayer(jaringan, "jaringan", "ogr")`	Buka layer jaringan dari teks sumber.
21-24	`processing.run("native:shortestpathpointtopoint", {...})`	Panggil alat rute terpendek titik ke titik. Isian penting: `STRATEGY` (0 terpendek, 1 tercepat), `DEFAULT_DIRECTION: 2` (jalan dua arah), `SPEED_FIELD` (kolom kecepatan), `DEFAULT_SPEED: 20` (km per jam bila kolom kosong), `TOLERANCE`, dan titik awal akhir dari `fmt(a)`, `fmt(b)`.
25	`return hasil["TRAVEL_COST"], hasil["OUTPUT"]`	Kembalikan dua hal: biaya total dan layer rute.
28-30	`jalan`, `jalan_buka`, `celah`	Tiga sumber jaringan. Dua yang terakhir menyaring ruas yang `status` nya bukan `ditutup`.
31	`tt = lambda s: "%.1f menit" % (s * 60)`	**Lambda ketiga.** Biaya tercepat berupa jam. Aturan ini mengubahnya menjadi teks menit.
33-42	`c, o = rute(jalan)` dan empat pemanggilan lain	Lima percobaan. Tiap hasil dipecah menjadi biaya (`c`) dan layer (`o`). Percobaan 3 memakai `strategi=1`, percobaan 5 memakai `toleransi=5.0`.
36	`o0.featureCount()`	Jumlah segmen di layer rute.
44	`print("Catatan: ...")`	Pengingat: pos jaga berjarak 25 meter di luar jalan, jadi QGIS menempelkannya ke titik jalan terdekat.

Hasil. Layar menampilkan:

```
HASIL
1) terpendek, semua ruas      : 2987.7 m
2) terpendek, ruas tutup dibuang: 3316.2 m  (jumlah segmen keluaran: 1)
3) tercepat (jam -> menit)    : 6.4 menit
4) jalan ber-celah, toleransi 0: 3351.3 m
5) jalan ber-celah, toleransi 5: 3316.2 m
Catatan: titik awal dan akhir diambil dari fasilitas yang berjarak 25 m di luar jalan; QGIS menempelkannya ke
titik jalan terdekat.
```

Cara membacanya: menutup satu ruas menambah rute sekitar 328 meter (2987,7 ke 3316,2). Pada jalan bercelah, tanpa toleransi QGIS tidak melihat ujung yang bercelah itu menyambung, sehingga rutenya memutar (3351,3). Itu kesimpulan dari angkanya, celahnya sendiri tidak diperiksa langsung. [kemungkinan] Dengan toleransi 5 meter, celah dianggap tertutup dan hasilnya kembali sama dengan nomor 2.

Bagian yang boleh diubah.

- **Baris 15, nama pos.** Ganti dengan nama fasilitas lain: `Pos Jaga 2`, `Pos Jaga 3`, `Kantor KPH`, atau `TPK`.
- **Baris 23, kecepatan bawaan.** Ganti `20` (km per jam) untuk ruas tanpa isian kecepatan.
- **Baris 29, saringan.** Ganti `status<>'ditutup'` dengan syarat lain.
- **Baris 37 dan 41, strategi dan toleransi.** Ganti `strategi=1` dan `toleransi=5.0`.

Jebakan.

- **Nama pos salah.** Bila nama tidak ada di kamus, baris 15 berhenti dengan `KeyError: 'Pos Jaga 9'`. Salin nama persis dari hasil baris 14.
- **Satuan biaya.** Untuk strategi tercepat, biayanya jam, bukan menit. Itu sebabnya ada lambda `tt`.
- **Jalan bercelah.** Tanpa toleransi, rute bisa memutar atau gagal. Selalu uji dengan toleransi kecil.
- **Sistem koordinat tertulis tetap.** `EPSG:32749` di baris 16 harus cocok dengan data. Bila data Anda memakai sistem lain, ganti.
- **Nilai bawaan dibaca sekali.** `a=awal` di baris 19 dibaca saat fungsi dibuat. Mengganti `awal` sesudahnya tidak mengubah nilai bawaan.
- **Lambda sulit dibaca pemula.** Tiga lambda di sini boleh ditulis ulang sebagai `def`. Hasilnya sama.

Latihan variasi.

1. Cari rute dari Pos Jaga 1 ke TPK.
2. Tulis ulang `tt` sebagai fungsi `def`.

Contoh jawaban 1: ganti `titik["Pos Jaga 4"]` di baris 15 menjadi `titik["TPK"]`. Layar menampilkan lima baris dengan angka berbeda, antara lain `1) terpendek, semua ruas : 1521.4 m` dan `3) tercepat (jam -> menit) : 5.1 menit`.

Contoh jawaban 2: ganti baris 31 dengan dua baris. Hasil skrip tidak berubah (diuji dengan ketiga lambda ditulis ulang sebagai `def`).

```python
def tt(s):
    return "%.1f menit" % (s * 60)
```

> **Pola yang bisa dipakai ulang:** Rute di QGIS: **siapkan titik awal dan akhir**, **bungkus pemanggilan alat dalam fungsi dengan nilai bawaan**, lalu **panggil fungsi itu untuk tiap skenario**. Ganti tiga hal: nama titik, sumber jaringan, dan strategi atau toleransi.

Skrip 25.6: Rute dan area layanan, QGIS lawan pgRouting

Berkas `m2_03c_silang_jaringan.py`. Ini silang kedua. Rute yang sama dihitung dua kali: sekali di QGIS dan sekali di PostGIS dengan **pgRouting**, ekstensi PostgreSQL untuk rute. Bila angkanya sama, hitungan Anda layak dipercaya.

Tujuan. Membandingkan rute terpendek, rute tercepat, dan area layanan dari QGIS dengan pgRouting, memakai titik yang tepat di simpul.

Bahan. Tabel `kph.jalan` dan `kph.fasilitas` (hasil Skrip 25.1), pgRouting aktif, `Jalan.gpkg`, dan `m2_03b_pgrouting.sql` yang **sudah dijalankan** (membuat tabel `kph.jalan_simpul` dan `kph.jalan_ruas`). Perintah: `python-qgis.bat m2_03c_silang_jaringan.py <folder paket-m2>`.

Konsep singkat. pgRouting memakai simpul bernomor dan ruas yang menyebut simpul asal dan tujuannya. `pgr_dijkstra` mencari rute terpendek, dan `pgr_drivingDistance` mencari semua simpul yang terjangkau dalam batas jarak (area layanan). Skrip ini juga memakai `%s`, tanda tempat untuk nilai yang diisi oleh `psycopg2` secara aman.

Skrip utuh:

PYTHON

```
# -*- coding: utf-8 -*-
"""M2 Bab 3: membandingkan rute dan area layanan QGIS dengan pgRouting memakai titik awal dan akhir yang SAMA
(tepat di simpul).
Penulis: Badar Mubarok Yogaswara. Syarat: m2_03b_pgrouting.sql sudah dijalankan; variabel PGHOST, PGPORT,
PGUSER, PGDATABASE terisi.
Pemakaian: python-qgis.bat m2_03c_silang_jaringan.py <folder paket-m2>"""
import os
import sys
sys.path.insert(0, os.path.dirname(os.path.abspath(__file__)))
import _inisialisasi  # noqa: F401
import processing
import psycopg2
from qgis.core import QgsVectorLayer

paket = sys.argv[1]
con = psycopg2.connect(dbname=os.environ["PGDATABASE"])
cur = con.cursor()


def simpul(vid):
    cur.execute("SELECT ST_X(geom), ST_Y(geom) FROM kph.jalan_simpul WHERE id = %s", (vid,))
    x, y = cur.fetchone()
    return "%.3f,%.3f [EPSG:32749]" % (x, y)


jalan_buka = QgsVectorLayer(os.path.join(paket, "Jalan.gpkg|layername=Jalan|subset=status<>'ditutup'"), "j",
"ogr")
a, b = simpul(5), simpul(24)                      # simpul terdekat Pos Jaga 1 dan Pos Jaga 4 (lihat keluaran
03b)


def qgis_rute(strategi):
    return processing.run("native:shortestpathpointtopoint", {
        "INPUT": jalan_buka, "STRATEGY": strategi, "DIRECTION_FIELD": "", "VALUE_FORWARD": "",
"VALUE_BACKWARD": "", "VALUE_BOTH": "",
        "DEFAULT_DIRECTION": 2, "SPEED_FIELD": "kecepatan_kmj", "DEFAULT_SPEED": 20, "TOLERANCE": 0,
        "START_POINT": a, "END_POINT": b, "OUTPUT": "memory:"})["TRAVEL_COST"]
```

```
cur.execute("""SELECT MAX(agg_cost) FROM pgr_dijkstra('SELECT id, source, target, panjang_m AS cost FROM
kph.jalan_ruas WHERE status <> ''ditutup''', 5, 24, false)""")
pg_m = cur.fetchone()[0]
cur.execute("""SELECT MAX(agg_cost) FROM pgr_dijkstra('SELECT id, source, target, waktu_dtk AS cost FROM
kph.jalan_ruas WHERE status <> ''ditutup''', 5, 24, false)""")
pg_dtk = cur.fetchone()[0]
q_m, q_jam = qgis_rute(0), qgis_rute(1)
print("terpendek : QGIS %.2f m | pgRouting %.2f m | selisih %.4f" % (q_m, pg_m, abs(q_m - pg_m)))
print("tercepat  : QGIS %.3f menit | pgRouting %.3f menit" % (q_jam * 60, pg_dtk / 60))

# area layanan: simpul persimpangan yang terjangkau
cur.execute("SELECT ST_X(geom), ST_Y(geom), id FROM kph.jalan_simpul")
persimpangan = {(round(x, 1), round(y, 1)): i for x, y, i in cur.fetchall()}
cur.execute("SELECT ST_X(geom), ST_Y(geom) FROM kph.jalan_simpul WHERE id = 11")
kx, ky = cur.fetchone()
for batas in (500, 1000):
    h = processing.run("native:serviceareafrompoint", {
        "INPUT": jalan_buka, "STRATEGY": 0, "DIRECTION_FIELD": "", "VALUE_FORWARD": "", "VALUE_BACKWARD": "",
"VALUE_BOTH": "",
        "DEFAULT_DIRECTION": 2, "SPEED_FIELD": "", "DEFAULT_SPEED": 50, "TOLERANCE": 0, "START_POINT": "%.3f,
%.3f [EPSG:32749]" % (kx, ky),
        "TRAVEL_COST": batas, "INCLUDE_BOUNDS": True, "OUTPUT": "memory:"})["OUTPUT"]
    dalam = set()
    for f in h.getFeatures():
        if f["type"] == "within":
            g = f.geometry()
            for p in (g.asMultiPoint() if g.isMultipart() else [g.asPoint()]):
                k = (round(p.x(), 1), round(p.y(), 1))
                if k in persimpangan:
                    dalam.add(persimpangan[k])
    cur.execute("""SELECT COUNT(*) FROM pgr_drivingDistance('SELECT id, source, target, panjang_m AS cost FROM
kph.jalan_ruas WHERE status <> ''ditutup''', 11, %s::float, false)""", (batas,))
    print("area layanan %4d m: persimpangan terjangkau QGIS %d | pgRouting %d" % (batas, len(dalam),
cur.fetchone()[0]))
```

Penjelasan baris demi baris. Nomor baris mengikuti berkas di atas.

Baris	Tulisan	Artinya
1-4	`# -*- coding ...` dan teks tiga kutip	Catatan penjelas dan syarat: 03b sudah dijalankan.
5-11	`import ...`, `import psycopg2`, `from qgis.core import QgsVectorLayer`	Panggil perkakas, nyalakan QGIS, dan panggil `psycopg2`.
13-15	`paket`, `con = psycopg2.connect(...)`, `cur = con.cursor()`	Ambil folder paket, sambung ke PostGIS, dan siapkan kursor.
18-21	`def simpul(vid): ...`	Fungsi yang mengambil koordinat simpul bernomor `vid`. Di SQL, `%s` adalah tempat nilai, dan `(vid,)` adalah nilainya (tuple berisi satu isi, dengan koma). Hasilnya teks koordinat siap dipakai QGIS.
24	`jalan_buka = QgsVectorLayer(... status<>'ditutup' ...)`	Layer jalan tanpa ruas ditutup, seperti Skrip 25.5.
25	`a, b = simpul(5), simpul(24)`	Titik awal dan akhir diambil tepat di simpul 5 dan 24 (simpul terdekat Pos Jaga 1 dan Pos Jaga 4, dari keluaran 03b).

Baris	Tulisan	Artinya
28-32	`def qgis_rute(strategi): ...`	Fungsi pembungkus rute QGIS. Mengembalikan biaya (`TRAVEL_COST`).
35-36	`cur.execute("""SELECT MAX(agg_cost) FROM pgr_dijkstra(...)""")`	Jalankan rute terpendek di pgRouting. Teks tiga kutip memudahkan menulis SQL panjang. Di dalam SQL bersarang, tanda kutip ditulis dua kali (`''ditutup''`). `agg_cost` adalah biaya kumulatif dan `MAX` mengambil yang terbesar, yaitu biaya total.
37-38	`... waktu_dtk AS cost ...`	Hal yang sama, tetapi biayanya waktu (detik).
39	`q_m, q_jam = qgis_rute(0), qgis_rute(1)`	Hitung rute QGIS terpendek (meter) dan tercepat (jam).
40-41	`print("terpendek : ...")`	Cetak perbandingan. `pg_dtk / 60` mengubah detik ke menit, `q_jam * 60` mengubah jam ke menit.
44-45	`persimpangan = {(round(x, 1), round(y, 1)): i for x, y, i in cur.fetchall()}`	Kamus dari koordinat bulat satu desimal ke nomor simpul. Kuncinya pasangan (x, y).
46-47	`... WHERE id = 11, kx, ky = cur.fetchone()`	Ambil koordinat simpul 11 (simpul terdekat Kantor KPH).
48	`for batas in (500, 1000):`	Ulangi untuk dua batas jarak.
49-52	`processing.run("native:serviceareafrompoint", {...})`	Hitung area layanan dari titik kantor dengan batas jarak `TRAVEL_COST`. `INCLUDE_BOUNDS: True` ikut menyertakan titik batas.
53-60	`dalam = set()` dan putaran	Kumpulkan nomor simpul yang terjangkau. `set` adalah himpunan (tanpa isi kembar). Untuk tiap fitur bertipe `within`, ambil titiknya, bulatkan, cocokkan dengan kamus, lalu tambahkan nomornya.
61	`pgr_drivingDistance(...)`	Hitung simpul terjangkau di pgRouting. `%s::float` menyisipkan batas jarak.
62	`print("area layanan ...")`	Cetak jumlah simpul terjangkau menurut QGIS dan pgRouting.

Hasil. Layar menampilkan:

```
HASIL
terpendek : QGIS 3296.93 m | pgRouting 3296.93 m | selisih 0.0000
tercepat  : QGIS 6.373 menit | pgRouting 6.373 menit
area layanan  500 m: persimpangan terjangkau QGIS 3 | pgRouting 3
area layanan 1000 m: persimpangan terjangkau QGIS 10 | pgRouting 10
```

Dua sistem sepakat sampai dua desimal. Angka 3296,93 meter lebih kecil dari 3316,2 meter di Skrip 25.5, karena sekarang rute dihitung dari simpul ke simpul, bukan dari pos jaga yang berjarak dari jalan. [kemungkinan]

Bagian yang boleh diubah.

- **Baris 25, nomor simpul.** `5` dan `24` bisa diganti. Angka yang sama juga tertulis di baris 35 dan 37, jadi ubah **di tiga tempat**.

- **Baris 46, nomor simpul kantor.** `11` untuk titik pusat area layanan.
- **Baris 48, batas jarak.** Ganti `(500, 1000)` dengan batas lain, dalam meter.
- **Baris 31 dan 50, kecepatan bawaan.** Isi `DEFAULT_SPEED` bila ruas tidak punya data kecepatan.

Jebakan.

- **03b belum dijalankan.** Tabel `kph.jalan_simpul` belum ada, dan baris 19 berhenti dengan `psycopg2.errors.UndefinedTable: relasi « kph.jalan_simpul » tidak ada` (pesan mengikuti bahasa server; diuji pada basis data tanpa tabel itu).
- **Nomor simpul kode keras.** Angka 5, 24, dan 11 berasal dari keluaran `m2_03b_pgrouting.sql` pada data ini. Bila data jalan berubah, nomor bisa bergeser. Baca keluaran 03b untuk mencarinya.
- **Kutip ganda dua kali.** `''ditutup''` di dalam SQL bersarang mudah salah ketik.
- **Koneksi tidak ditutup.** `con.close()` tidak ada. Untuk skrip pendek tidak masalah.
- **Pencocokan titik dengan pembulatan.** Titik dicocokkan lewat koordinat yang dibulatkan satu desimal. Bila data berubah, pencocokan bisa meleset. [kemungkinan]

Latihan variasi. Bandingkan rute dari Pos Jaga 1 ke Pos Jaga 2. Pos Jaga 2 berada di simpul 21 menurut keluaran 03b. Contoh jawaban: ganti `simpul(24)` menjadi `simpul(21)` di baris 25, dan `5, 24, false` menjadi `5, 21, false` di baris 35 **dan** 37. Layar menampilkan `terpendek : QGIS 2739.99 m | pgRouting 2739.99 m | selisih 0.0000` dan `tercepat  : QGIS 7.338 menit | pgRouting 7.338 menit`.

> **Pola yang bisa dipakai ulang:** Silang dua mesin rute: **ambil titik yang sama di keduanya**, **hitung di masing-masing**, **ubah satuan agar sama** (detik ke menit, jam ke menit), lalu **cetak berdampingan dengan selisihnya**. Ganti nomor simpul (di semua tempat), batas jarak, dan sumber data.

Skrip 25.7: Hidrologi dari peta ketinggian

Berkas `m2_04a_hidrologi.py`. Dari satu peta ketinggian (DEM), skrip ini mencari ke mana air mengalir, di mana sungainya, dan daerah mana yang airnya bermuara ke satu titik. Hasilnya berguna untuk menilai petak mana yang dekat sungai.

Tujuan. Mengisi lubang pada DEM, menghitung arah dan akumulasi aliran, mengekstrak sungai, mencari titik keluar, dan membentuk daerah aliran sungai (DAS).

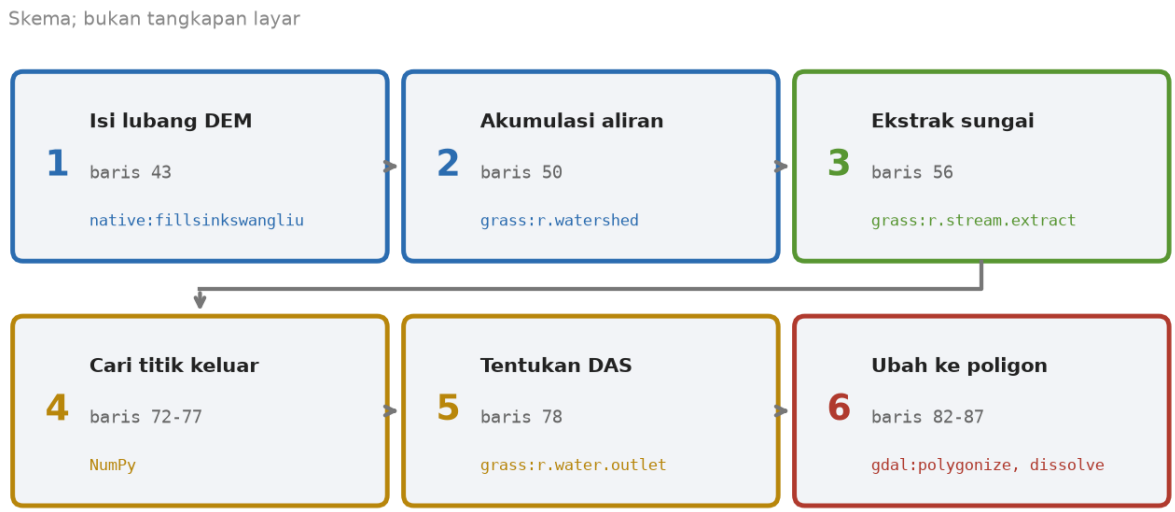
Bahan. `DEM_10m.tif` (200 x 200 sel, sel 10 meter) dan folder `hasil`. Butuh GRASS (lewat `_inisialisasi`). Tidak butuh PostGIS. Waktu jalan pada uji: sekitar 22 detik. Hasilnya disimpan di `hasil`.

Konsep: air mengalir ke bawah. Bayangkan DEM sebagai maket pegunungan dari tanah liat. Hujan yang jatuh di tiap titik mengalir ke tetangga yang paling rendah. Lima istilah baru:

- **DEM** (*Digital Elevation Model*): peta ketinggian berbentuk grid. Tiap sel berisi tinggi.
- **Lubang** (*sink*): sel yang lebih rendah daripada semua tetangganya. Air terjebak di sana. Biasanya itu kesalahan data, jadi diisi.

- **Akumulasi aliran**: untuk tiap sel, jumlah sel di hulu yang airnya lewat sini. Sel dengan angka besar adalah calon sungai.
- **Ambang** (*threshold*): batas akumulasi untuk menyebut sebuah sel "sungai". Di bab ini 500 sel, yaitu 500 x 100 m2 = 5 hektare.
- **DAS dan titik keluar** (*outlet*): DAS adalah seluruh daerah yang airnya bermuara ke satu titik keluar.

Ilustrasi 25.4: Rantai hidrologi



Skrip utuh:

```python
# -*- coding: utf-8 -*-
"""M2 Bab 4: hidrologi dari DEM: isi lubang (fill sinks), arah dan akumulasi aliran, sungai, dan DAS di satu titik keluar (outlet).
Penulis: Badar Mubarok Yogaswara. Pemakaian: python-qgis.bat m2_04a_hidrologi.py <folder paket-m2>
Keluaran disimpan di <paket-m2>/hasil. Ambang sungai: 500 sel (500 x 100 m2 = 5 ha)."""
import os
import sys
import numpy as np
sys.path.insert(0, os.path.dirname(os.path.abspath(__file__)))
import _inisialisasi  # noqa: F401
import processing
from osgeo import gdal

gdal.UseExceptions()
paket = sys.argv[1]
out = os.path.join(paket, "hasil")
os.makedirs(out, exist_ok=True)
dem = os.path.join(paket, "DEM_10m.tif")
P = lambda n: os.path.join(out, n)
BAKU = {"GRASS_REGION_PARAMETER": None, "GRASS_REGION_CELLSIZE_PARAMETER": 0}


def baca(path):
    ds = gdal.Open(path)
    return ds.GetRasterBand(1).ReadAsArray().astype("float64"), ds.GetGeoTransform()


def jumlah_lubang(a):
    """jumlah sel dalam (bukan tepi) yang lebih rendah daripada semua 8 tetangganya"""
```

```
    c = a[1:-1, 1:-1]
    lebih_tinggi = np.ones_like(c, dtype=bool)
    for dy in (-1, 0, 1):
        for dx in (-1, 0, 1):
            if dy == 0 and dx == 0:
                continue
            lebih_tinggi &= a[1 + dy:a.shape[0] - 1 + dy, 1 + dx:a.shape[1] - 1 + dx] > c
    return int(lebih_tinggi.sum())


z0, gt = baca(dem)
print("lubang (sink) pada DEM asli            :", jumlah_lubang(z0))

# 1) isi lubang (Wang & Liu), kemiringan minimum 0,1 derajat
processing.run("native:fillsinkswangliu", {"INPUT": dem, "BAND": 1, "MIN_SLOPE": 0.1, "OUTPUT_FILLED_DEM":
P("DEM_terisi.tif"),
                                           "OUTPUT_FLOW_DIRECTIONS": P("Arah_WL.tif"),
"OUTPUT_WATERSHED_BASINS": P("Basin_WL.tif")})
z1, _ = baca(P("DEM_terisi.tif"))
print("lubang sesudah diisi                   :", jumlah_lubang(z1))
print("sel yang naik / kenaikan terbesar (m)  : %d / %.2f" % (int((z1 > z0 + 1e-6).sum()), float((z1 -
z0).max())))

# 2) akumulasi aliran, arah, dan cekungan dengan GRASS r.watershed (aliran tunggal D8)
processing.run("grass:r.watershed", dict(BAKU, elevation=P("DEM_terisi.tif"), threshold=500, convergence=5,
memory=300, **{"-s": True, "-a": True},
               accumulation=P("Akumulasi.tif"), drainage=P("Arah.tif"), basin=P("Cekungan.tif")))
acc, _ = baca(P("Akumulasi.tif"))
print("akumulasi maksimum (sel) / luas (ha)   : %d / %.1f" % (abs(acc).max(), abs(acc).max() * 100 / 1e4))

# 3) sungai: sel dengan akumulasi >= 500 sel (5 ha)
processing.run("grass:r.stream.extract", dict(BAKU, elevation=P("DEM_terisi.tif"),
accumulation=P("Akumulasi.tif"), threshold=500, mexp=0,
               stream_length=0, memory=300, stream_raster=P("Sungai.tif"),
stream_vector=P("sungai_mentah.gpkg"), direction=P("Arah_stream.tif"),
               GRASS_OUTPUT_TYPE_PARAMETER=2, GRASS_VECTOR_EXPORT_NOCAT=False))
if os.path.exists(P("Sungai.gpkg")):
    os.remove(P("Sungai.gpkg"))
gdal.VectorTranslate(P("Sungai.gpkg"), P("sungai_mentah.gpkg"), options=gdal.VectorTranslateOptions(
    format="GPKG", layerName="Sungai", selectFields=["cat"], geometryType="LINESTRING"))
from osgeo import ogr
ds = ogr.Open(P("Sungai.gpkg"))
lyr = ds.GetLayer(0)
panjang = sum(f.GetGeometryRef().Length() for f in lyr)
print("sungai: %d segmen, total panjang %.2f km" % (lyr.GetFeatureCount(), panjang / 1000.0))
ds = None


# 4) titik keluar (outlet): sel tepi dengan akumulasi terbesar, lalu DAS-nya
acc, gt = baca(P("Akumulasi.tif"))
tepi = np.zeros_like(acc, dtype=bool)
tepi[0, :] = tepi[-1, :] = tepi[:, 0] = tepi[:, -1] = True
r, c = np.unravel_index(np.argmax(np.where(tepi, acc, -1)), acc.shape)
ox, oy = gt[0] + (c + 0.5) * gt[1], gt[3] + (r + 0.5) * gt[5]
print("outlet: baris %d kolom %d, X=%.1f Y=%.1f, akumulasi %d sel" % (r, c, ox, oy, acc[r, c]))
processing.run("grass:r.water.outlet", dict(BAKU, input=P("Arah.tif"), coordinates="%.1f,%.1f [EPSG:32749]" %
(ox, oy), output=P("DAS_outlet.tif")))
das, _ = baca(P("DAS_outlet.tif"))
n_das = int((das == 1).sum())
print("DAS di outlet: %d sel = %.2f ha (%.1f%% wilayah)" % (n_das, n_das * 100 / 1e4, 100.0 * n_das /
das.size))
mentah = processing.run("gdal:polygonize", {"INPUT": P("DAS_outlet.tif"), "BAND": 1, "FIELD": "nilai",
"EIGHT_CONNECTEDNESS": False,
                                            "EXTRA": "", "OUTPUT": "TEMPORARY_OUTPUT"})["OUTPUT"]
dalam = processing.run("native:extractbyexpression", {"INPUT": mentah, "EXPRESSION": '"nilai" = 1', "OUTPUT":
"memory:"})["OUTPUT"]
if os.path.exists(P("DAS_utama.gpkg")):
    os.remove(P("DAS_utama.gpkg"))
```

```
processing.run("native:dissolve", {"INPUT": dalam, "FIELD": [], "SEPARATE_DISJOINT": False, "OUTPUT":
P("DAS_utama.gpkg")})
print("DAS poligon ditulis:", os.path.exists(P("DAS_utama.gpkg")))
```

Skrip ini 88 baris. Penjelasannya dipecah menjadi enam blok.

Blok 1, baris 1-19: persiapan.

Baris	Tulisan	Artinya
1-4	`# -*- coding ...` dan teks tiga kutip	Catatan penjelas, termasuk ambang sungai 500 sel.
5-7	`import os`, `import sys`, `import numpy as np`	Panggil perkakas. `as np` memberi nama singkat `np` pada `numpy`, pustaka hitung angka (Bab 12).
8-11	`sys.path.insert(...)`, `import _inisialisasi`, `import processing`, `from osgeo import gdal`	Nyalakan QGIS lalu panggil Processing dan GDAL.
13-17	`gdal.UseExceptions()` ... `dem = ...`	Siapkan folder `hasil` (`makedirs(..., exist_ok=True)` membuatnya bila belum ada) dan jalur DEM.
18	`P = lambda n: os.path.join(out, n)`	Lambda: aturan "ambil nama berkas `n`, letakkan di folder hasil". `P("Sungai.tif")` menjadi jalur lengkap.
19	`BAKU = {"GRASS_REGION_PARAMETER": None, "GRASS_REGION_CELLSIZE_PARAMETER": 0}`	Pengaturan baku GRASS: wilayah dan ukuran sel ikut data masukan. Dipakai ulang di tiga pemanggilan.

Blok 2, baris 22-36: dua fungsi pembantu.

Baris	Tulisan	Artinya
22-24	`def baca(path): ...`	Buka raster dan kembalikan dua hal: isi sel sebagai larik angka (`ReadAsArray`) dan angka posisi grid (`GetGeoTransform`).
27-28	`def jumlah_lubang(a):` dan teks penjelas	Hitung lubang: sel dalam yang lebih rendah daripada 8 tetangganya.
29	`c = a[1:-1, 1:-1]`	Irisan: ambil sel dalam saja, tanpa baris dan kolom tepi.
30	`lebih_tinggi = np.ones_like(c, dtype=bool)`	Buat larik benar-salah seukuran `c`, semuanya benar dulu.
31-35	`for dy in (-1, 0, 1): ...` `lebih_tinggi &= a[...] > c`	Untuk tiap arah di antara 8 tetangga (kecuali diri sendiri, lewat `continue`), geser peta lalu bandingkan. Tanda `&=` menjaga nilai tetap benar hanya bila tetangga ini lebih tinggi.
36	`return int(lebih_tinggi.sum())`	Jumlah sel yang masih benar, yaitu lubang.

Blok 3, baris 39-47: mengisi lubang.

Baris	Tulisan	Artinya
39-40	`z0, gt = baca(dem), print(... jumlah_lubang(z0))`	Baca DEM asli dan cetak jumlah lubangnya.
43-44	`processing.run("native:fillsinkswangliu", {...})`	Alat isi lubang cara Wang dan Liu. `MIN_SLOPE: 0.1` memberi kemiringan minimum 0,1 derajat agar sel yang diisi tetap mengalir. Tiga keluaran: DEM terisi, arah aliran, dan cekungan.
45-47	`z1, _ = baca(...), print(...)`	Baca DEM terisi. Tanda `_` berarti nilai kedua dibuang. Cetak jumlah lubang baru, jumlah sel yang naik, dan kenaikan terbesar.

Blok 4, baris 49-53: arah dan akumulasi aliran.

Baris	Tulisan	Artinya
50-51	`processing.run("grass:r.watershed", dict(BAKU, elevation=..., threshold=500, ...))`	Panggil alat GRASS. `dict(BAKU, ...)` menyalin kamus `BAKU` lalu menambah isian lain.
50	`**{"-s": True, "-a": True}`	Menyisipkan dua isian yang namanya diawali tanda minus, yang tidak bisa ditulis langsung. `-s` berarti aliran satu arah (D8, delapan tetangga), `-a` berarti akumulasi positif.
51	`accumulation=..., drainage=..., basin=...`	Tiga keluaran: akumulasi, arah aliran, dan cekungan.
52-53	`acc, _ = baca(...), print(... abs(acc).max() ...)`	Cetak akumulasi terbesar dalam sel dan dalam hektare. `abs` dipakai karena GRASS bisa memberi nilai negatif. Angka `100` adalah luas sel dalam meter persegi, dan `1e4` adalah 10.000.

Blok 5, baris 55-68: sungai.

Baris	Tulisan	Artinya
56-58	`processing.run("grass:r.stream.extract", dict(BAKU, ...))`	Ambil sel yang akumulasinya sama atau di atas `threshold=500` sebagai sungai. Keluarannya raster sungai dan garis sungai mentah.
59-60	`if os.path.exists(P("Sungai.gpkg")): os.remove(...)`	Hapus hasil lama agar tidak bentrok.
61-62	`gdal.VectorTranslate(P("Sungai.gpkg"), P("sungai_mentah.gpkg"), ...)`	Salin garis mentah ke berkas bersih bernama `Sungai.gpkg`, hanya kolom `cat`, bentuk garis.
63-66	`from osgeo import ogr ... panjang = sum(...)`	Buka berkas itu dan jumlahkan panjang semua garis lewat generator di dalam `sum`.
67-68	`print(...), ds = None`	Cetak jumlah segmen dan panjang dalam km. Lalu lepaskan berkas dengan mengosongkan kotaknya.

Blok 6, baris 71-88: titik keluar dan DAS.

Baris	Tulisan	Artinya
72-74	`acc, gt = baca(...), tepi = np.zeros_like(...), tepi[0, :] = ... = True`	Buat larik benar-salah yang hanya benar di empat sisi tepi. Penugasan berantai mengisi empat sisi sekaligus.
75	`r, c = np.unravel_index(np.argmax(np.where(tepi, acc, -1)), acc.shape)`	Dari sel tepi saja (sel lain diganti `-1`), cari akumulasi terbesar, lalu ubah nomornya menjadi baris `r` dan kolom `c`.
76-77	`ox, oy = ..., print("outlet: ...")`	Hitung koordinat pusat sel itu dari angka posisi grid, lalu cetak.
78	`processing.run("grass:r.water.outlet", dict(BAKU, input=..., coordinates=..., output=...))`	Tentukan DAS dari titik keluar tadi.
79-81	`das, _ = baca(...), n_das = ..., print(...)`	Hitung sel DAS (nilai 1), luasnya dalam hektare, dan persen wilayah.
82-83	`processing.run("gdal:polygonize", {...})`	Ubah raster DAS menjadi poligon.
84	`processing.run("native:extractbyexpression", {... '"nilai" = 1' ...})`	Ambil poligon bernilai 1, yaitu DAS.
85-88	`os.remove(...), processing.run("native:dissolve", {...}), print(...)`	Hapus hasil lama, gabungkan poligon menjadi satu (`dissolve`), tulis `DAS_utama.gpkg`, dan cetak bahwa berkasnya ada.

Hasil. Layar menampilkan:

```
HASIL
lubang (sink) pada DEM asli            : 167
lubang sesudah diisi                   : 0
sel yang naik / kenaikan terbesar (m)  : 1994 / 13.67
akumulasi maksimum (sel) / luas (ha)   : 25581 / 255.8
sungai: 30 segmen, total panjang 7.75 km
outlet: baris 199 kolom 49, X=312495.0 Y=9996005.0, akumulasi 25581 sel
DAS di outlet: 26211 sel = 262.11 ha (65.5% wilayah)
DAS poligon ditulis: True
```

Cara membaca: ada 167 lubang di DEM asli, semuanya terisi. Sungai terdiri atas 30 segmen sepanjang 7,75 km. DAS yang bermuara di titik keluar menutupi 65,5 persen wilayah (262 ha dari 400 ha). Folder `hasil` kini berisi `DEM_terisi.tif`, `Akumulasi.tif`, `Sungai.gpkg`, `DAS_utama.gpkg`, dan lainnya. QGIS juga mencetak sedikit teks proses GRASS yang boleh diabaikan.

Bagian yang boleh diubah.

- **Baris 56, ambang sungai.** `threshold=500`: makin kecil, sungai makin rapat. Ganti ke `200` untuk sungai lebih rapat.

- **Baris 43, kemiringan minimum.** `MIN_SLOPE: 0.1`. Ganti bila air di tanah datar tidak mengalir sebagaimana mestinya.
- **Baris 50, ambang cekungan.** Isian `threshold=500` di sini terpisah dari ambang sungai di baris 56.
- **Baris 53 dan 81, luas sel.** Angka `100` (m2) benar hanya untuk sel 10 meter. Untuk DEM lain, ganti.
- **Baris 17, DEM.** Ganti `DEM_10m.tif` dengan DEM Anda.

Jebakan.

- **GRASS tidak terdeteksi.** Bila folder GRASS tidak dikenali, baris 50 berhenti dengan `QgsProcessingException: GRASS GIS folder is not configured. Please configure it before running GRASS GIS algorithms.` (diuji dengan mengganti nama folder GRASS). Cek baris 19 `_inisialisasi.py`.
- **Luas sel tertulis tetap.** Baris 53 dan 81 memakai `100`. Dengan DEM berukuran sel lain, hektarenya salah tanpa peringatan.
- **Titik keluar ditentukan otomatis.** Skrip memilih sel tepi dengan akumulasi terbesar. Itu bukan titik pilihan Anda.
- **Hasil lama dihapus.** Baris 59-60 dan 85-86 menghapus berkas lama. Jangan taruh hasil penting dengan nama sama di folder `hasil`.
- **Hasil bergantung versi.** Angka di atas berlaku untuk QGIS 4.0.2 dengan GRASS 8.4. Versi lain bisa sedikit berbeda. [CEK]

Latihan variasi. Buat sungai lebih rapat dengan ambang 200 sel. Contoh jawaban: ganti `threshold=500` menjadi `threshold=200` **hanya di baris 56** (baris `r.stream.extract`). Baris `sungai:` menampilkan `sungai: 105 segmen, total panjang 15.47 km`. DAS dan titik keluar tidak berubah, karena keduanya tidak memakai ambang sungai.

> **Pola yang bisa dipakai ulang:** Hidrologi dari DEM: **isi lubang**, **hitung arah dan akumulasi**, **ambil sel di atas ambang sebagai sungai**, **pilih titik keluar**, lalu **tentukan DAS**. Ganti tiga hal: DEM, ambang, dan titik keluar.

Skrip 25.8: Peta curah hujan dari 24 stasiun

Berkas `m2_05a_interpolasi.py`. Kepala Seksi punya 30 stasiun hujan, tetapi hujan jatuh di mana-mana, bukan hanya di stasiun. Skrip ini menebak curah hujan tahunan di seluruh wilayah dari titik-titik stasiun, memakai sembilan metode, lalu memilih yang paling jitu. Skrip ini panjang (167 baris), jadi dipecah menjadi enam blok.

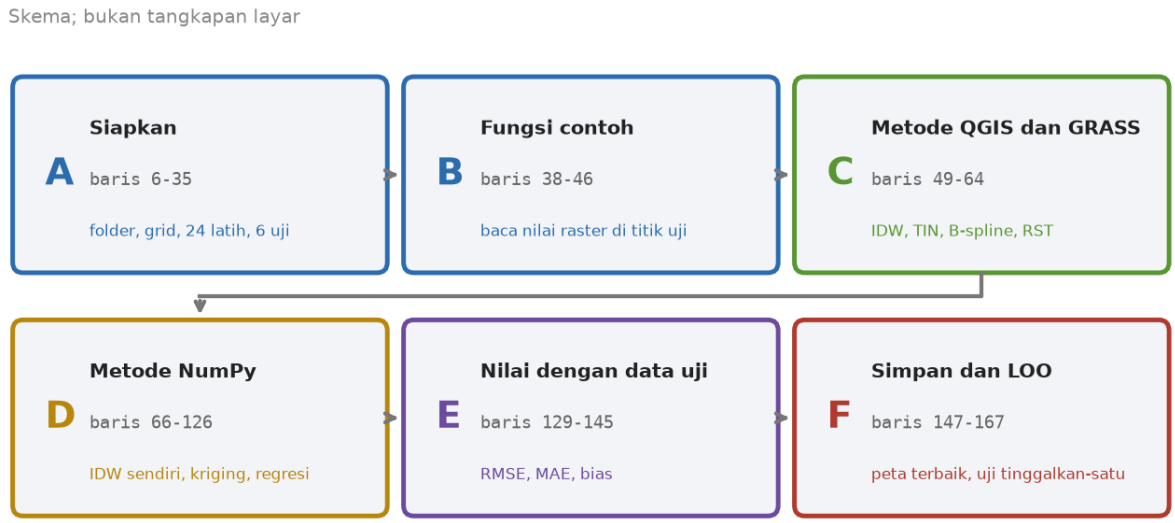
Tujuan. Membuat peta curah hujan tahunan dengan banyak metode interpolasi, menilai tiap metode dengan stasiun yang disembunyikan, dan menyimpan peta metode terbaik sebagai `CH_tahunan.tif`.

Bahan. `Stasiun_Hujan.gpkg` (30 stasiun; kolom `elevasi_m`, `ch_tahunan_mm`, dan `peran` yang bernilai `latih` atau `uji`) dan `DEM_10m.tif`. Butuh GRASS, NumPy, dan SciPy. Tidak butuh PostGIS. Waktu jalan pada uji: sekitar 23 detik.

Konsep: menebak di tempat yang tidak diukur. Bayangkan Anda hanya punya termometer di beberapa desa dan ingin menebak suhu di desa lain. Itulah **interpolasi**: menebak nilai di tempat tak terukur dari titik terukur. Lima istilah baru:

- **IDW** (*inverse distance weighting*): rata-rata berbobot, di mana stasiun yang lebih dekat "bersuara" lebih keras. **Pangkat** `p` menentukan seberapa cepat suara itu mengecil dengan jarak.
- **TIN**: jaring segitiga yang menyambung stasiun. Di dalam tiap segitiga, nilai dihitung dari tiga sudutnya. Di luar jaring, tidak ada nilai.
- **Data latih dan data uji**: 24 stasiun dipakai membuat peta (latih). Enam stasiun lain disembunyikan, lalu dipakai menguji tebakan (uji). Seperti ujian: soalnya belum pernah dilihat.
- **RMSE**: ukuran galat. Rata-rata selisih tebakan dan kenyataan, dengan selisih besar dihitung lebih berat. Makin kecil makin baik. **MAE** adalah rata-rata selisih mutlak, dan **bias** adalah rata-rata selisih bertanda (positif berarti tebakan cenderung kelebihan).
- **Tinggalkan-satu** (*leave-one-out*, LOO): uji dengan menyembunyikan satu stasiun bergantian, sebanyak stasiun yang ada.

Ilustrasi 25.5: Enam blok Skrip 25.8



Skrip utuh:

```python
# -*- coding: utf-8 -*-
"""M2 Bab 5: interpolasi curah hujan tahunan dari 24 stasiun latih, dinilai dengan 6 stasiun uji.
Metode: IDW (p = 1, 2, 3), TIN linear, TIN Clough-Tocher, B-spline (GRASS), RST (GRASS), kriging biasa (NumPy),
dan regresi + residu IDW (NumPy).
Penulis: Badar Mubarok Yogaswara. Pemakaian: python-qgis.bat m2_05a_interpolasi.py <folder paket-m2>
Keluaran: <paket-m2>/hasil/CH_tahunan.tif (metode terbaik) dan tabel galat (RMSE) di layar."""
import os
import sys
import numpy as np
sys.path.insert(0, os.path.dirname(os.path.abspath(__file__)))
import _inisialisasi  # noqa: F401
import processing
from osgeo import gdal, ogr
from scipy.optimize import curve_fit
```

```
gdal.UseExceptions()
ogr.UseExceptions()
paket = sys.argv[1]
out = os.path.join(paket, "hasil")
os.makedirs(out, exist_ok=True)
P = lambda n: os.path.join(out, n)
X0, Y0, L, RES = 312000.0, 9996000.0, 2000.0, 10.0
EXT = "%f,%f,%f,%f [EPSG:32749]" % (X0, X0 + L, Y0, Y0 + L)

# ---- data stasiun: latih dan uji
ds = ogr.Open(os.path.join(paket, "Stasiun_Hujan.gpkg"))
lyr = ds.GetLayer(0)
st = [(f.GetGeometryRef().GetX(), f.GetGeometryRef().GetY(), f["elevasi_m"], f["ch_tahunan_mm"], f["peran"])
for f in lyr]
latih = np.array([s[:4] for s in st if s[4] == "latih"])
uji = np.array([s[:4] for s in st if s[4] == "uji"])
print("stasiun latih: %d, uji: %d" % (len(latih), len(uji)))
processing.run("native:extractbyexpression", {"INPUT": os.path.join(paket, "Stasiun_Hujan.gpkg|
layername=Stasiun_Hujan"),
                                              "EXPRESSION": "\"peran\" = 'latih'", "OUTPUT":
P("stasiun_latih.gpkg")})
from qgis.core import QgsVectorLayer
fi = QgsVectorLayer(P("stasiun_latih.gpkg"), "latih", "ogr").fields().indexOf("ch_tahunan_mm")      # indeks
kolom menurut QGIS (kolom fid ikut dihitung)
data_in = "%s::~::0::~::%d::~::0" % (P("stasiun_latih.gpkg"), fi)


def contoh(raster_path, titik):
    r = gdal.Open(raster_path)
    gt = r.GetGeoTransform()
    a = r.GetRasterBand(1).ReadAsArray()
    nd = r.GetRasterBand(1).GetNoDataValue()
    h = np.array([a[int((gt[3] - y) / -gt[5]), int((x - gt[0]) / gt[1])] for x, y in titik[:, :2]],
dtype="float64")
    if nd is not None:
        h[h == nd] = np.nan                          # di luar segitiga TIN: tidak ada nilai
    return h


hasil = {}
for p in (1, 2, 3):
    processing.run("qgis:idwinterpolation", {"INTERPOLATION_DATA": data_in, "DISTANCE_COEFFICIENT": p,
"EXTENT": EXT, "PIXEL_SIZE": RES,
                                             "OUTPUT": P("CH_idw_p%d.tif" % p)})
    hasil["IDW p=%d" % p] = P("CH_idw_p%d.tif" % p)
for nama, m in (("TIN linear", 0), ("TIN Clough-Tocher", 1)):
    fn = P("CH_tin_%d.tif" % m)
    processing.run("qgis:tininterpolation", {"INTERPOLATION_DATA": data_in, "METHOD": m, "EXTENT": EXT,
"PIXEL_SIZE": RES, "OUTPUT": fn})
    hasil[nama] = fn
BAKU = {"GRASS_REGION_PARAMETER": EXT, "GRASS_REGION_CELLSIZE_PARAMETER": RES}
processing.run("grass:v.surf.bspline", dict(BAKU, input=P("stasiun_latih.gpkg"), column="ch_tahunan_mm",
ew_step=500, ns_step=500, method=1,
               lambda_i=0.01, solver=0, maxit=10000, error=1e-6, memory=300,
raster_output=P("CH_bspline.tif")))
hasil["B-spline bikubik"] = P("CH_bspline.tif")
processing.run("grass:v.surf.rst", dict(BAKU, input=P("stasiun_latih.gpkg"), zcolumn="ch_tahunan_mm",
tension=40, smooth=0.5, segmax=40, npmin=150,
               elevation=P("CH_rst.tif")))
hasil["RST (spline tegangan)"] = P("CH_rst.tif")

# ---- baris dan kolom sel pusat grid, untuk metode NumPy
n = int(L / RES)
gx = X0 + (np.arange(n) + 0.5) * RES
gy = Y0 + L - (np.arange(n) + 0.5) * RES
GX, GY = np.meshgrid(gx, gy)
dem = gdal.Open(os.path.join(paket, "DEM_10m.tif")).ReadAsArray().astype("float64")
```

```
def idw_np(xy, v, tx, ty, p=2):
    d = np.hypot(tx[..., None] - xy[:, 0], ty[..., None] - xy[:, 1])
    d = np.maximum(d, 1e-9)
    w = 1.0 / d ** p
    return (w * v).sum(-1) / w.sum(-1)


# kriging biasa dengan semivariogram eksponensial yang disesuaikan ke data latih
xy = latih[:, :2]
v = latih[:, 3]
ii, jj = np.triu_indices(len(xy), 1)
h = np.hypot(*(xy[ii] - xy[jj]).T)
gam = 0.5 * (v[ii] - v[jj]) ** 2
tepi = np.linspace(0, h.max() * 0.6, 8)
hc, gc = [], []
for a, b in zip(tepi[:-1], tepi[1:]):
    m = (h >= a) & (h < b)
    if m.sum() >= 5:
        hc.append(h[m].mean())
        gc.append(gam[m].mean())
model = lambda hh, c0, c, a: c0 + c * (1 - np.exp(-3 * hh / a))
(c0, c, a), _ = curve_fit(model, hc, gc, p0=[gc[0], max(gc) - gc[0], hc[-1]], bounds=([0, 1, 100], [1e7, 1e7,
5000]))
print("semivariogram eksponensial: nugget=%.0f, sill parsial=%.0f, jangkauan=%.0f m" % (c0, c, a))


def krig(tx, ty):
    N = len(xy)
    D = np.hypot(xy[:, None, 0] - xy[None, :, 0], xy[:, None, 1] - xy[None, :, 1])
    A = np.ones((N + 1, N + 1))
    A[:N, :N] = model(D, 0, c, a) + np.where(D == 0, 0, c0)
    A[N, N] = 0
    Ainv = np.linalg.inv(A)
    shp = tx.shape
    d0 = np.hypot(tx.ravel()[:, None] - xy[:, 0], ty.ravel()[:, None] - xy[:, 1])
    B = np.ones((d0.shape[0], N + 1))
    B[:, :N] = model(d0, 0, c, a) + c0
    lam = B @ Ainv
    return (lam[:, :N] @ v).reshape(shp)


# regresi pada elevasi (dan koordinat X), lalu residunya dengan IDW p=2
def elev_di(x, y):
    return dem[np.clip(((Y0 + L - y) / RES).astype(int), 0, n - 1), np.clip(((x - X0) / RES).astype(int), 0, n
- 1)]


Xr = np.column_stack([np.ones(len(latih)), latih[:, 2], latih[:, 0] - X0])
beta, *_ = np.linalg.lstsq(Xr, v, rcond=None)
res = v - Xr @ beta
print("regresi CH = %.1f + %.2f * elevasi + %.4f * (X - X0)   (R2 = %.3f)" % (beta[0], beta[1], beta[2], 1 -
res.var() / v.var()))


def regresi(tx, ty):
    return beta[0] + beta[1] * elev_di(tx, ty) + beta[2] * (tx - X0) + idw_np(xy, res, tx, ty, 2)


np_metode = {"Kriging biasa (NumPy)": krig, "Regresi elevasi + residu IDW (NumPy)": regresi}
tabel = []
for nama, fn in hasil.items():
    pred = contoh(fn, uji)
    tabel.append((nama, pred))
for nama, fn in np_metode.items():
    pred = fn(uji[:, 0], uji[:, 1])
    tabel.append((nama, pred))
```

```
print("\nGalat pada 6 stasiun uji (mm/tahun), dihitung dari prediksi - pengamatan")
print("%-40s %8s %8s %8s %4s" % ("metode", "RMSE", "MAE", "bias", "n"))
skor = {}
for nama, pred in tabel:
    e = (pred - uji[:, 3])
    e = e[~np.isnan(e)]
    skor[nama] = float(np.sqrt((e ** 2).mean())) if len(e) == len(uji) else 1e9      # metode yang tak mencakup
semua titik uji tidak dipilih
    print("%-40s %8.1f %8.1f %8.1f %4d" % (nama, np.sqrt((e ** 2).mean()), np.abs(e).mean(), e.mean(), len(e)))
print("terbaik menurut RMSE:", min(skor, key=skor.get))

# simpan peta terbaik dari metode NumPy bila memang terbaik; jika tidak, salin raster QGIS terbaik
terbaik = min(skor, key=skor.get)
if terbaik in np_metode:
    arr = np_metode[terbaik](GX, GY).astype("float32")
    d = gdal.GetDriverByName("GTiff").Create(P("CH_tahunan.tif"), n, n, 1, gdal.GDT_Float32,
["COMPRESS=DEFLATE"])
    d.SetGeoTransform((X0, RES, 0, Y0 + L, 0, -RES))
    d.SetProjection(gdal.Open(os.path.join(paket, "DEM_10m.tif")).GetProjection())
    d.GetRasterBand(1).WriteArray(arr)
    d = None
else:
    gdal.Translate(P("CH_tahunan.tif"), hasil[terbaik], creationOptions=["COMPRESS=DEFLATE"])
print("CH_tahunan.tif dibuat dari:", terbaik)

# validasi silang tinggalkan-satu (LOO) untuk pangkat IDW pada 24 stasiun latih
print("\nTinggalkan-satu (LOO) pada 24 stasiun latih, RMSE IDW menurut pangkat:")
for p in (0.5, 1, 2, 3, 4):
    e = []
    for k in range(len(xy)):
        m = np.arange(len(xy)) != k
        e.append(idw_np(xy[m], v[m], np.array([xy[k, 0]]), np.array([xy[k, 1]]), p)[0] - v[k])
    print("  p=%.1f  RMSE=%.1f" % (p, np.sqrt(np.mean(np.square(e)))))
```

Blok A, baris 1-35: persiapan.

Baris	Tulisan	Artinya
1-5	`# -*- coding ...` dan teks tiga kutip	Catatan penjelas: daftar metode dan keluaran.
6-13	`import ...`, `from scipy.optimize import curve_fit`	Panggil perkakas, nyalakan QGIS, lalu panggil `curve_fit` dari SciPy, pustaka sains. `curve_fit` mencocokkan kurva ke titik data.
15-20	`gdal.UseExceptions()` ... `P = lambda n: ...`	Hidupkan laporan galat, ambil folder paket, siapkan folder `hasil`, dan buat lambda `P` untuk jalur berkas keluaran (seperti Skrip 25.7).
21	`X0, Y0, L, RES = 312000.0, 9996000.0, 2000.0, 10.0`	Empat nilai sekaligus: sudut kiri-bawah wilayah (X0, Y0), panjang sisi `L` (2000 m), dan ukuran sel `RES` (10 m).
22	`EXT = "%f,%f,%f,%f [EPSG:32749]" % (...)`	Susun teks cakupan: x terkecil, x terbesar, y terkecil, y terbesar, lalu sistem koordinat.
25-26	`ds = ogr.Open(...)`, `lyr = ds.GetLayer(0)`	Buka layer stasiun.
27	`st = [(... GetX(), GetY(), f["elevasi_m"], f["ch_tahunan_mm"], f["peran"]) for f in lyr]`	Daftar tuple, satu per stasiun: koordinat X dan Y, elevasi, curah hujan, dan peran.

Baris	Tulisan	Artinya
28-29	`latih = np.array([s[:4] for s in st if s[4] == "latih"])`	Pilih stasiun berperan `latih` dan simpan empat nilai pertamanya (`s[:4]`) dalam larik NumPy. `uji` dibuat sama.
30	`print("stasiun latih: %d, uji: %d" % ...)`	Cetak jumlah keduanya.
31-32	`processing.run("native:extractbyexpression", {...})`	Simpan stasiun latih saja ke `stasiun_latih.gpkg`, supaya alat QGIS dan GRASS tidak melihat stasiun uji.
33-35	`fi = ... .fields().indexOf("ch_tahunan_mm"), data_in = ...`	Cari nomor urut kolom hujan menurut QGIS (kolom `fid` ikut dihitung, jadi hasilnya 4). Lalu rakit teks `data_in`, format khusus alat IDW dan TIN: jalur berkas, penanda `::~::`, angka 0, nomor kolom, dan angka 0. Arti tiap angka: [CEK].

Blok B, baris 38-46: fungsi pembaca contoh.

Baris	Tulisan	Artinya
38-41	`def contoh(raster_path, titik): ...`	Fungsi yang membaca nilai raster di tiap titik uji. Buka raster, ambil angka posisi grid (`gt`), isi sel (`a`), dan nilai kosong (`nd`).
43	`h = np.array([a[int((gt[3] - y) / -gt[5]), int((x - gt[0]) / gt[1])] for x, y in titik[:, :2]], dtype="float64")`	Untuk tiap titik (kolom X dan Y), hitung baris dan kolom sel yang memuatnya, lalu ambil isinya. Hasilnya daftar angka di titik-titik uji.
44-46	`if nd is not None: h[h == nd] = np.nan`	Bila raster punya nilai kosong, ganti dengan `np.nan` ("bukan angka"), penanda tidak ada nilai. Nilai itu mencegah TIN yang tidak mencakup titik terhitung.

Blok C, baris 49-64: metode QGIS dan GRASS.

Baris	Tulisan	Artinya
49	`hasil = {}`	Kamus kosong, nanti berisi nama metode dan jalur petanya.
50-53	`for p in (1, 2, 3): processing.run("qgis:idwinterpolation", {...})`	Jalankan IDW tiga kali dengan pangkat 1, 2, dan 3. `DISTANCE_COEFFICIENT` adalah pangkat, `PIXEL_SIZE` ukuran sel. Tiap peta disimpan dan dicatat di `hasil`.
54-57	`for nama, m in (("TIN linear", 0), ("TIN Clough-Tocher", 1)):`	Jalankan TIN dua cara. `METHOD: 0` linear, `1` kubik Clough-Tocher.
58	`BAKU = {"GRASS_REGION_PARAMETER": EXT, ...}`	Pengaturan baku GRASS: wilayah mengikuti `EXT` dan ukuran sel `RES`.
59-61	`processing.run("grass:v.surf.bspline", dict(BAKU, ...))`	Metode B-spline bikubik: permukaan mulus seperti lembaran karet. `ew_step` dan `ns_step` adalah jarak antar simpul lengkung (500 m), `lambda_i` mengatur kehalusan.
62-64	`processing.run("grass:v.surf.rst", dict(BAKU, ...))`	Metode RST (spline tegangan). `tension` mengatur kekakuan lembaran dan `smooth` kehalusan.

Blok D, baris 66-126: metode NumPy. Tiga metode berikutnya ditulis sendiri dengan NumPy.

Baris	Tulisan	Artinya
67-71	`n = int(L / RES) … dem = ...`	Siapkan titik pusat tiap sel peta (200 x 200 sel). `np.meshgrid` membuat dua larik koordinat X dan Y. Baca juga DEM sebagai larik.
74-78	`def idw_np(xy, v, tx, ty, p=2): ...`	IDW buatan sendiri. Hitung jarak ke semua stasiun (`np.hypot`), bobot `1 / jarak ** p`, lalu rata-rata berbobot. `p=2` adalah nilai bawaan. Jarak terkecil dibatasi `1e-9` agar tidak membagi nol.
82-83	`xy = latih[:, :2], v = latih[:, 3]`	Pisahkan koordinat stasiun latih dan nilai hujannya.
84-86	`ii, jj = np.triu_indices(len(xy), 1) … gam = ...`	Untuk semua pasangan stasiun, hitung jarak antar pasangan (`h`) dan setengah kuadrat selisih hujannya (`gam`). Itu bahan semivariogram.
87-93	`tepi = np.linspace(...), for a, b in zip(...)`	Kelompokkan pasangan menurut jarak dalam 7 selang. `zip` menyandingkan batas bawah dan atas tiap selang. Untuk selang berisi minimal 5 pasangan, simpan jarak rata-rata (`hc`) dan nilai `gam` rata-rata (`gc`).
94-96	`model = lambda hh, c0, c, a: ..., curve_fit(...), print(...)`	**Semivariogram**: grafik yang menunjukkan makin jauh dua stasiun, makin beda hujannya. Lambda `model` adalah rumusnya (jenis eksponensial). `curve_fit` mencari tiga angka terbaik: `c0` (nugget), `c` (sill parsial), dan `a` (jangkauan). Tulisan `(c0, c, a), _ =` membongkar tiga angka itu ke tiga kotak dan membuang hasil kedua.
99-111	`def krig(tx, ty): ...`	**Kriging biasa**: seperti IDW, tetapi bobotnya dihitung dari semivariogram tadi lewat persamaan matriks. `np.linalg.inv` membalik matriks, tanda `@` mengalikan matriks. Hasilnya `lam[:, :N] @ v`, yaitu tebakan di tiap titik.
115-116	`def elev_di(x, y): ...`	Fungsi pembantu: ambil elevasi DEM di titik (x, y). `np.clip` menjaga indeks tetap di dalam peta.
119-122	`Xr = np.column_stack(...), beta, *_ = np.linalg.lstsq(...), res = ..., print(...)`	**Regresi**: cari garis lurus yang menghubungkan hujan dengan elevasi dan koordinat X, lewat `lstsq` (kuadrat terkecil). `beta, *_ =` mengambil angka pertama dan membuang sisanya. `res` adalah sisa selisih yang belum terjelaskan.
125-126	`def regresi(tx, ty): ...`	Tebakan akhir: hasil regresi ditambah sisa selisih yang disebar dengan IDW pangkat 2. Dua tahap: "tebak dari ketinggian", lalu "perbaiki dengan tetangga".

Blok E, baris 129-145: menilai semua metode dengan data uji.

Baris	Tulisan	Artinya
129-136	`np_metode = {...}, tabel = []`, dua putaran `for`	Kumpulkan tebakan tiap metode di enam titik uji. Metode QGIS dan GRASS lewat `contoh(...)`, metode NumPy lewat fungsinya. Tiap tebakan disimpan di `tabel`.
137-138	`print("\nGalat pada ..."), print("%-40s %8s ..." % ...)`	Cetak judul tabel. `%-40s` berarti teks rata kiri selebar 40 karakter.
139-144	`skor = {}, for nama, pred in tabel: ...`	Untuk tiap metode, hitung selisih (`e`), buang yang kosong (`~np.isnan(e)`), lalu hitung RMSE, MAE, dan bias. Bila metode tidak mencakup semua titik uji, skornya dibuat `1e9` (sangat buruk) agar tidak terpilih.
143	`skor[nama] = ... if len(e) == len(uji) else 1e9`	Contoh ekspresi bersyarat satu baris.
145	`print("terbaik menurut RMSE:", min(skor, key=skor.get))`	Cari nama metode dengan skor terkecil. `key=skor.get` berarti "bandingkan lewat nilainya".

Blok F, baris 147-167: simpan peta terbaik dan uji tinggalkan-satu.

Baris	Tulisan	Artinya
147-158	`terbaik = min(skor, key=skor.get), if terbaik in np_metode: ... else: ...`	Bila yang terbaik metode NumPy, hitung petanya di seluruh sel dan tulis sebagai GeoTIFF dengan GDAL (`Create`, `SetGeoTransform`, `SetProjection`, `WriteArray`). Bila yang terbaik metode QGIS atau GRASS, cukup salin petanya (`gdal.Translate`).
158	`print("CH_tahunan.tif dibuat dari:", terbaik)`	Cetak metode yang dipakai.
161-167	`for p in (0.5, 1, 2, 3, 4): ... for k in range(len(xy)): ...`	Uji tinggalkan-satu untuk pangkat IDW. Tiap putaran menyembunyikan satu stasiun (`m = np.arange(len(xy)) != k` memilih semua kecuali ke-`k`), menebaknya dari sisanya, lalu mengumpulkan selisihnya. Terakhir cetak RMSE per pangkat.

Hasil. Layar menampilkan (setelah beberapa peringatan GRASS yang boleh diabaikan):

```
HASIL
stasiun latih: 24, uji: 6
semivariogram eksponensial: nugget=0, sill parsial=66870, jangkauan=5000 m
regresi CH = 1944.2 + 3.04 * elevasi + 0.1371 * (X - X0)   (R2 = 0.912)

Galat pada 6 stasiun uji (mm/tahun), dihitung dari prediksi - pengamatan
metode                                   RMSE      MAE     bias    n
IDW p=1                                 183.4    175.2    150.1    6
IDW p=2                                 126.5    119.7     97.4    6
IDW p=3                                  94.6     83.8     61.0    6
TIN linear                               55.9     40.5     38.6    2
TIN Clough-Tocher                        98.3     74.6    -45.6    3
B-spline bikubik                        100.7     84.1      9.5    6
RST (spline tegangan)                    89.9     76.2     41.0    6
```

```
Kriging biasa (NumPy)                        83.7     68.4     33.5    6
Regresi elevasi + residu IDW (NumPy)         42.2     33.3      9.6    6
terbaik menurut RMSE: Regresi elevasi + residu IDW (NumPy)
CH_tahunan.tif dibuat dari: Regresi elevasi + residu IDW (NumPy)

Tinggalkan-satu (LOO) pada 24 stasiun latih, RMSE IDW menurut pangkat:
  p=0.5  RMSE=169.0
  p=1.0  RMSE=149.4
  p=2.0  RMSE=115.3
  p=3.0  RMSE=96.0
  p=4.0  RMSE=90.1
```

Cara membaca: kolom `n` menunjukkan berapa dari enam titik uji yang tercakup. TIN linear hanya mencakup 2 titik, sehingga RMSE-nya (55,9) tampak bagus tetapi tidak adil dibandingkan. Skrip menolaknya. Pemenangnya regresi dengan RMSE 42,2 mm per tahun. Regresinya berbunyi: tiap kenaikan elevasi 100 meter menambah sekitar 304 mm hujan. Pada uji tinggalkan-satu, RMSE IDW terus turun sampai pangkat 4. Peta `CH_tahunan.tif` berukuran 200 x 200 sel, bernilai antara sekitar 2170 dan 3053 mm per tahun. Semua ini data sintetis.

Bagian yang boleh diubah.

- **Baris 21, cakupan dan ukuran sel.** `X0, Y0, L, RES`. Ubah `RES` ke `20` untuk sel lebih kasar dan lebih cepat.
- **Baris 50, pangkat IDW.** Ganti `(1, 2, 3)`. Judul di tabel ikut berubah lewat baris 53.
- **Baris 59-60, B-spline.** `ew_step`, `ns_step`, dan `lambda_i` mengatur kehalusan.
- **Baris 62-63, RST.** `tension` dan `smooth`.
- **Baris 95, batas jangkauan.** Angka `5000` pada `bounds` membatasi jangkauan semivariogram.
- **Baris 162, pangkat uji LOO.** Ganti daftar `(0.5, 1, 2, 3, 4)`.

Jebakan.

- **Indeks kolom pada data masukan.** Baris 34-35 mencari nomor kolom lewat QGIS. Kolom `fid` ikut dihitung, jadi nomornya 4, bukan 3. Salah nomor berarti yang diinterpolasi kolom lain.
- **TIN tidak mencakup semua titik.** Di luar segitiga, TIN tidak punya nilai, sehingga `n` kurang dari 6. Skrip membuang metode itu dengan skor `1e9`, jadi pemenangnya tidak bisa TIN.
- **Jangkauan semivariogram menempel di batas.** Hasil `jangkauan=5000 m` sama dengan batas atas di baris 95. Itu tanda data tidak cukup untuk menentukan jangkauan dengan tegas. [kemungkinan]
- **Matriks kriging bisa singular.** Bila dua stasiun berada di titik yang sama, `np.linalg.inv` gagal. Pada data ini tidak terjadi.
- **Butuh GRASS.** Tanpa GRASS yang terdeteksi, baris 59 gagal (lihat Skrip 25.7).
- **Hasil lama ditimpa.** Berkas bernama sama di folder `hasil` akan diganti.
- **Angka 24 dan 6 hanya di teks.** Hitungan tidak memakainya. Jumlah latih dan uji ditentukan kolom `peran` di data.

Latihan variasi. Uji pangkat IDW 2, 4, dan 6 pada tinggalkan-satu. Contoh jawaban: ganti baris 162 menjadi `for p in (2, 4, 6):`. Layar menampilkan tiga baris terakhir:

```
HASIL
  p=2.0  RMSE=115.3
  p=4.0  RMSE=90.1
  p=6.0  RMSE=92.4
```

Jadi pada data ini, RMSE terendah ada di sekitar pangkat 4, dan naik lagi di pangkat 6.

> **Pola yang bisa dipakai ulang:** Membandingkan banyak metode: **sembunyikan sebagian data sebagai data uji**, **buat peta dengan tiap metode dari data latih**, **baca nilai peta di titik uji**, **hitung RMSE, MAE, dan bias**, lalu **pilih yang terbaik**. Ganti empat hal: data stasiun, daftar metode, ukuran sel, dan kriteria pemilihan.

Skrip 25.9: Peta kesesuaian lahan dengan tumpang tindih berbobot

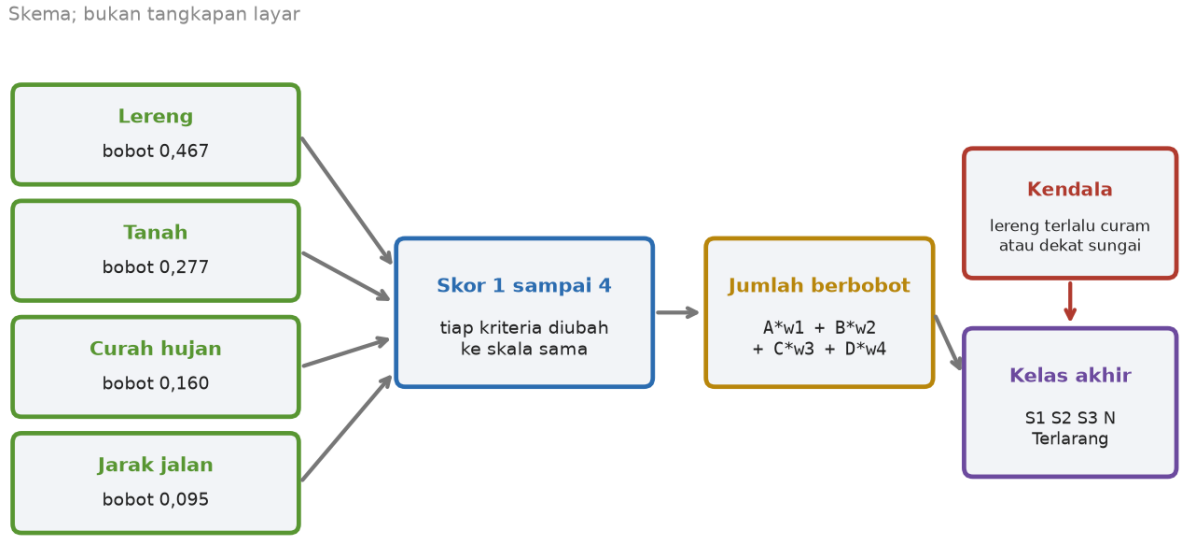
Berkas `m2_06a_kesesuaian.py`. Kepala Seksi bertanya: lahan mana yang paling cocok untuk "jenis pohon contoh"? Skrip ini menilai tiap sel peta dari empat kriteria, menimbangnya, menerapkan larangan, lalu merangkum luasnya per kelas dan per petak. Kriteria dan ambangnya **fiktif untuk latihan**, bukan standar resmi (kata penulis di berkas).

Tujuan. Membuat peta kelas kesesuaian (S1 sangat sesuai sampai N tidak sesuai, plus kelas terlarang), memeriksanya dengan NumPy, menguji kepekaan bobot, dan merangkum per petak.

Bahan. `DEM_10m.tif`, `Tanah.gpkg` (kolom `jenis_tanah`), `Jalan.gpkg`, `Petak.gpkg`, dan dua keluaran bab sebelumnya di folder `hasil`: `Sungai.gpkg` (Skrip 25.7) serta `CH_tahunan.tif` (Skrip 25.8). **Jalankan 25.7 dan 25.8 dulu.** Tidak butuh PostGIS. Waktu jalan pada uji: sekitar 23 detik.

Konsep: skor, bobot, larangan. Bayangkan juri lomba dengan empat penilaian. Tiap sel peta mendapat **skor** 1 sampai 4 untuk tiap kriteria. Tiap kriteria punya **bobot** yang menyatakan seberapa penting. Skor akhir adalah jumlah skor kali bobot. Sebagian sel punya **kendala** (larangan) dan langsung dinyatakan terlarang, berapa pun skornya. Bobotnya tidak ditebak: dihitung dengan **AHP** (*Analytic Hierarchy Process*), metode yang menimbang kriteria lewat perbandingan berpasangan ("lereng lebih penting dari tanah, seberapa?"). Lima istilah baru: skor, bobot, kendala, AHP, dan **reklasifikasi** (mengubah nilai ke kelas, misalnya lereng 0 sampai 8 persen menjadi skor 4).

Ilustrasi 25.6: Skema kesesuaian lahan



Bobot dari AHP (perbandingan berpasangan). Kriteria dan ambang di bab ini fiktif untuk latihan.

Skrip utuh:

```python
# -*- coding: utf-8 -*-
"""M2 Bab 6: kesesuaian lahan multi-kriteria (tumpang tindih berbobot) untuk "jenis pohon contoh" di KPH Contoh.
Kriteria dan ambang di bawah ini FIKTIF untuk latihan; bukan standar kesesuaian lahan resmi.
Penulis: Badar Mubarok Yogaswara. Pemakaian: python-qgis.bat m2_06a_kesesuaian.py <folder paket-m2>
Prasyarat: Bab 4 (hidrologi: Sungai.gpkg) dan Bab 5 (CH_tahunan.tif) sudah dijalankan; keluarannya ada di <paket-m2>/hasil."""
import os
import sys
import numpy as np
sys.path.insert(0, os.path.dirname(os.path.abspath(__file__)))
import _inisialisasi  # noqa: F401
import processing
from osgeo import gdal

gdal.UseExceptions()
paket = sys.argv[1]
out = os.path.join(paket, "hasil")
P = lambda n: os.path.join(out, n)
EXT = "312000,314000,9996000,9998000 [EPSG:32749]"
dem = os.path.join(paket, "DEM_10m.tif")
JARAK_SUNGAI = 30                                   # m: ambang kendala dari sungai (dipakai di rumus QGIS dan pemeriksaan NumPy)


def baca(path):
```

```
    ds = gdal.Open(path)
    return ds.GetRasterBand(1).ReadAsArray().astype("float64")


def reklas(inp, tabel, hasil, batas=1):
    flat = [v for baris in tabel for v in baris]
    processing.run("native:reclassifybytable", {"INPUT_RASTER": inp, "RASTER_BAND": 1, "TABLE": flat,
"NO_DATA": 0, "RANGE_BOUNDARIES": batas,
                                                "NODATA_FOR_MISSING": True, "DATA_TYPE": 0, "OUTPUT": hasil})


# ---------------------------------------------------------------- 1. bobot dengan AHP (perbandingan
berpasangan)
nama = ["lereng", "tanah", "curah hujan", "jarak jalan"]
A = np.array([[1, 2, 3, 4], [1 / 2, 1, 2, 3], [1 / 3, 1 / 2, 1, 2], [1 / 4, 1 / 3, 1 / 2, 1]])
vals, vecs = np.linalg.eig(A)
k = int(np.argmax(vals.real))
w = np.abs(vecs[:, k].real)
w = w / w.sum()
CI = (vals[k].real - len(A)) / (len(A) - 1)
CR = CI / 0.90                                                  # RI untuk n = 4 adalah 0,90 (tabel Saaty)
print("bobot AHP:", dict(zip(nama, np.round(w, 3))), "| lambda_maks = %.3f | CI = %.4f | CR = %.4f" %
(vals[k].real, CI, CR))

# ---------------------------------------------------------------- 2. kriteria
processing.run("gdal:slope", {"INPUT": dem, "BAND": 1, "SCALE": 1, "AS_PERCENT": True, "COMPUTE_EDGES": True,
"ZEVENBERGEN": False,
                              "OUTPUT": P("Lereng_pct.tif")})
reklas(P("Lereng_pct.tif"), [[0, 8, 4], [8, 15, 3], [15, 25, 2], [25, 40, 1]], P("Skor_lereng.tif"))
reklas(P("CH_tahunan.tif"), [[0, 2400, 1], [2400, 2600, 2], [2600, 2800, 3], [2800, 99999, 4]],
P("Skor_hujan.tif"))

tanah = processing.run("native:fieldcalculator", {
    "INPUT": os.path.join(paket, "Tanah.gpkg|layername=Tanah"), "FIELD_NAME": "skor", "FIELD_TYPE": 1,
"FIELD_LENGTH": 3, "FIELD_PRECISION": 0,
    "FORMULA": "CASE WHEN \"jenis_tanah\"='Aluvial' THEN 4 WHEN \"jenis_tanah\"='Latosol' THEN 3 WHEN
\"jenis_tanah\"='Podsolik' THEN 2 ELSE 1 END",
    "OUTPUT": "memory:"})["OUTPUT"]
processing.run("gdal:rasterize", {"INPUT": tanah, "FIELD": "skor", "BURN": 0, "USE_Z": False, "UNITS": 0,
"WIDTH": 200, "HEIGHT": 200, "EXTENT": EXT,
                                  "NODATA": 0, "DATA_TYPE": 0, "INIT": 0, "INVERT": False, "OUTPUT":
P("Skor_tanah.tif")})


def jarak(vektor, tmp, hasil):
    processing.run("gdal:rasterize", {"INPUT": vektor, "FIELD": "", "BURN": 1, "USE_Z": False, "UNITS": 0,
"WIDTH": 200, "HEIGHT": 200,
                                      "EXTENT": EXT, "NODATA": 0, "DATA_TYPE": 0, "INIT": 0, "INVERT": False,
"OUTPUT": tmp})
    processing.run("gdal:proximity", {"INPUT": tmp, "BAND": 1, "VALUES": "1", "UNITS": 0, "MAX_DISTANCE": 0,
"REPLACE": 0, "NODATA": -9999,
                                      "DATA_TYPE": 5, "OUTPUT": hasil})


jarak(os.path.join(paket, "Jalan.gpkg|layername=Jalan|subset=status<>'ditutup'"), P("_jalan_sel.tif"),
P("Jarak_jalan.tif"))
jarak(P("Sungai.gpkg"), P("_sungai_sel.tif"), P("Jarak_sungai.tif"))
reklas(P("Jarak_jalan.tif"), [[0, 100, 4], [100, 250, 3], [250, 500, 2], [500, 1e9, 1]], P("Skor_jalan.tif"))

# ---------------------------------------------------------------- 3. kendala (larangan): lereng >= 40% atau
dalam 30 m dari sungai
processing.run("gdal:rastercalculator", {"INPUT_A": P("Skor_lereng.tif"), "BAND_A": 1, "INPUT_B":
P("Jarak_sungai.tif"), "BAND_B": 1,
                                         "FORMULA": "logical_and(A>0, B>%d)" % JARAK_SUNGAI, "NO_DATA": 255,
"RTYPE": 0, "EXTENT_OPT": 0, "EXTRA": "--hideNoData", "OUTPUT": P("Kendala_boleh.tif")})

# ---------------------------------------------------------------- 4. tumpang tindih berbobot
rumus = "A*%.6f + B*%.6f + C*%.6f + D*%.6f" % tuple(w)
```

```
processing.run("gdal:rastercalculator", {"INPUT_A": P("Skor_lereng.tif"), "BAND_A": 1, "INPUT_B":
P("Skor_tanah.tif"), "BAND_B": 1,
                                         "INPUT_C": P("Skor_hujan.tif"), "BAND_C": 1, "INPUT_D":
P("Skor_jalan.tif"), "BAND_D": 1,
                                         "FORMULA": rumus, "NO_DATA": -9999, "RTYPE": 5, "EXTENT_OPT": 0,
"EXTRA": "--hideNoData", "OUTPUT": P("Kesesuaian_skor_mentah.tif")})
processing.run("gdal:rastercalculator", {"INPUT_A": P("Kesesuaian_skor_mentah.tif"), "BAND_A": 1, "INPUT_B":
P("Kendala_boleh.tif"), "BAND_B": 1,
                                         "FORMULA": "(B==1)*(1*(A>=3.2) + 2*logical_and(A>=2.6, A<3.2) +
3*logical_and(A>=2.0, A<2.6) + 4*(A<2.0))",
                                         "NO_DATA": 255, "RTYPE": 0, "EXTENT_OPT": 0, "EXTRA": "--hideNoData",
"OUTPUT": P("Kesesuaian_kelas.tif")})
kelas = baca(P("Kesesuaian_kelas.tif"))
label = {0: "Terlarang (kendala)", 1: "S1 Sangat sesuai", 2: "S2 Cukup sesuai", 3: "S3 Sesuai marginal", 4: "N
Tidak sesuai"}
print("\nLuas per kelas (ha), 1 sel = 100 m2 = 0,01 ha:")
for kd in range(5):
    print("  %-22s %6d sel  %7.2f ha  %5.1f%%" % (label[kd], (kelas == kd).sum(), (kelas == kd).sum() * 0.01,
100.0 * (kelas == kd).mean()))
print("  jumlah: %.2f ha (sel NoData: %d)" % (sum((kelas == kd).sum() for kd in range(5)) * 0.01, (kelas ==
255).sum()))

# ---------------------------------------------------------------- 5. pemeriksaan silang dengan NumPy murni
s = [baca(P("Skor_lereng.tif")), baca(P("Skor_tanah.tif")), baca(P("Skor_hujan.tif")),
baca(P("Skor_jalan.tif"))]
skor = sum(wi * si for wi, si in zip(w, s))
boleh = (s[0] > 0) & (baca(P("Jarak_sungai.tif")) > JARAK_SUNGAI)
np_kelas = np.where(boleh, np.select([skor >= 3.2, skor >= 2.6, skor >= 2.0], [1, 2, 3], 4), 0)
print("\nPemeriksaan NumPy: sel kelas sama = %.4f%%" % (100.0 * (np_kelas == kelas).mean()))

# ---------------------------------------------------------------- 6. kepekaan: bobot sama rata
skor2 = sum(0.25 * si for si in s)
np2 = np.where(boleh, np.select([skor2 >= 3.2, skor2 >= 2.6, skor2 >= 2.0], [1, 2, 3], 4), 0)
print("Kepekaan: luas S1 bobot AHP = %.2f ha; bobot sama rata = %.2f ha; sel yang berubah kelas = %.1f%%" %
      ((np_kelas == 1).sum() * 0.01, (np2 == 1).sum() * 0.01, 100.0 * (np2 != np_kelas).mean()))

# ---------------------------------------------------------------- 7. per petak (histogram zona) dan poligon
untuk basis data
zh = processing.run("native:zonalhistogram", {"INPUT_RASTER": P("Kesesuaian_kelas.tif"), "RASTER_BAND": 1,
                                              "INPUT_VECTOR": os.path.join(paket, "Petak.gpkg|
layername=Petak"), "COLUMN_PREFIX": "k_",
                                              "OUTPUT": "memory:"})["OUTPUT"]
baris = []
for f in zh.getFeatures():
    baris.append((f["kode"], f["jenis_tegakan"], *(f["k_%d" % kd] * 0.01 for kd in (1, 2, 3, 4, 0))))
baris.sort(key=lambda r: -r[2])
print("\nLima petak dengan S1 terluas (ha): kode, jenis, S1, S2, S3, N, Terlarang")
for r in baris[:5]:
    print("  %s %-12s %6.2f %6.2f %6.2f %6.2f %6.2f" % r)
print("  total S1 dari 25 petak = %.2f ha" % sum(r[2] for r in baris))
mentah = processing.run("gdal:polygonize", {"INPUT": P("Kesesuaian_kelas.tif"), "BAND": 1, "FIELD": "kelas",
"EIGHT_CONNECTEDNESS": False, "EXTRA": "",
                                            "OUTPUT": "TEMPORARY_OUTPUT"})["OUTPUT"]
if os.path.exists(P("Kesesuaian_kelas.gpkg")):
    os.remove(P("Kesesuaian_kelas.gpkg"))
processing.run("native:dissolve", {"INPUT": mentah, "FIELD": ["kelas"], "SEPARATE_DISJOINT": False, "OUTPUT":
P("Kesesuaian_kelas.gpkg")})
for tmp in ("_jalan_sel.tif", "_sungai_sel.tif"):
    try:
        os.remove(P(tmp))
    except OSError:
        pass
print("\npoligon kelas ditulis:", os.path.exists(P("Kesesuaian_kelas.gpkg")))
```

Skrip ini 124 baris dan dipecah menjadi tujuh blok.

Blok 1, baris 1-31: persiapan dan fungsi reklasifikasi.

Baris	Tulisan	Artinya
1-5	`# -*- coding ...` dan teks tiga kutip	Catatan penjelas, termasuk peringatan bahwa kriteria fiktif dan syarat menjalankan 25.7 dan 25.8 lebih dulu.
6-12	`import ...`, `from osgeo import gdal`	Panggil perkakas dan nyalakan QGIS.
14-19	`gdal.UseExceptions()` ... `dem = ...`	Siapkan folder, lambda `P`, dan teks cakupan `EXT`.
20	`JARAK_SUNGAI = 30`	Simpan jarak aman dari sungai (30 meter) dalam satu nama. Huruf besar semua adalah kebiasaan Python untuk **konstanta**, yaitu nilai yang tidak diubah selama skrip jalan. Nama ini dipakai di dua tempat: rumus kendala (baris 72) dan pemeriksaan NumPy (baris 92).
23-25	`def baca(path): ...`	Baca raster menjadi larik angka (versi lebih pendek dari Skrip 25.7).
28-31	`def reklas(inp, tabel, hasil, batas=1):`	Fungsi reklasifikasi. Baris 29 meratakan daftar bersarang: `[v for baris in tabel for v in baris]` mengubah `[[0, 8, 4], [8, 15, 3]]` menjadi `[0, 8, 4, 8, 15, 3]`. Alat `native:reclassifybytable` meminta tabel dalam bentuk datar: tiga angka (batas bawah, batas atas, skor) berulang.
30	`"RANGE_BOUNDARIES": batas`	`1` berarti batas bawah ikut, batas atas tidak (`min <= nilai < maks`).
30-31	`"NO_DATA": 0, "NODATA_FOR_MISSING": True`	Nilai yang tidak masuk rentang mana pun diberi skor 0. Itu penanda kendala di Blok 4.

Blok 2, baris 34-43: bobot AHP.

Baris	Tulisan	Artinya
35	`nama = ["lereng", "tanah", "curah hujan", "jarak jalan"]`	Daftar nama empat kriteria.
36	`A = np.array([[1, 2, 3, 4], ...])`	Matriks perbandingan berpasangan 4 x 4. Angka 2 di baris pertama berarti "lereng 2 kali lebih penting daripada tanah". Bagian bawah segitiga adalah kebalikannya (`1 / 2`, dan seterusnya).
37-40	`vals, vecs = np.linalg.eig(A) … w = w / w.sum()`	Cari nilai eigen terbesar dan vektornya, yang menjadi bobot. Lalu bobot dinormalkan supaya jumlahnya 1.
41-42	`CI = ..., CR = CI / 0.90`	Periksa konsistensi penilaian. `CI` adalah indeks konsistensi, `CR` rasionya terhadap angka acuan 0,90 untuk matriks 4 x 4 (tabel Saaty). Nilai `CR` di bawah 0,1 umumnya dianggap konsisten. [CEK]
43	`print("bobot AHP:", dict(zip(nama, np.round(w, 3))), ...)`	Pasangkan nama dan bobot dengan `zip`, jadikan kamus, lalu cetak bersama `lambda_maks`, `CI`, dan `CR`.

Blok 3, baris 45-68: menyiapkan empat kriteria.

Baris	Tulisan	Artinya
46-48	`processing.run("gdal:slope", {...}), reklas(...)`	Hitung lereng dari DEM dalam persen (`AS_PERCENT: True`), lalu ubah ke skor: 0-8 persen skor 4, 8-15 skor 3, 15-25 skor 2, 25-40 skor 1. Lereng 40 persen ke atas tidak masuk rentang, jadi skor 0.
49	`reklas(P("CH_tahunan.tif"), [[0, 2400, 1], ...], ...)`	Ubah curah hujan ke skor: makin deras makin tinggi, 1 sampai 4.
51-54	`tanah = processing.run("native:fieldcalculator", {...})["OUTPUT"]`	Tambah kolom `skor` di layer tanah lewat rumus `CASE WHEN`: Aluvial 4, Latosol 3, Podsolik 2, selain itu 1. `FIELD_TYPE: 1` berarti bilangan bulat.
55-56	`processing.run("gdal:rasterize", {...})`	Ubah poligon tanah menjadi raster 200 x 200 sel, bernilai kolom `skor`.
59-63	`def jarak(vektor, tmp, hasil): ...`	Fungsi dua langkah: ubah garis ke raster (`gdal:rasterize`, nilai 1 di mana ada garis), lalu hitung jarak ke garis terdekat untuk tiap sel (`gdal:proximity`).
66-67	`jarak(...)` dua kali	Panggil fungsi `jarak` untuk jalan yang tidak ditutup (saringan `status<>'ditutup'` ikut di sumber data) dan untuk sungai (hasil Skrip 25.7). Hasilnya `Jarak_jalan.tif` dan `Jarak_sungai.tif`.
68	`reklas(P("Jarak_jalan.tif"), [[0, 100, 4], ...], ...)`	Ubah jarak ke jalan menjadi skor: makin dekat makin tinggi. `1e9` berarti angka sangat besar.

Blok 4 dan 5, baris 70-87: kendala, tumpang tindih, dan kelas.

Baris	Tulisan	Artinya
71-72	`processing.run("gdal:rastercalculator", {... "FORMULA": "logical_and(A>0, B>%d)" % JARAK_SUNGAI ...})`	**Kendala.** Tanda `%d` adalah lubang untuk bilangan bulat, yang diisi dari `JARAK_SUNGAI` sehingga rumusnya menjadi `B>30`. Sel boleh dipakai (nilai 1) bila skor lereng lebih dari 0 (lereng di bawah 40 persen) **dan** jarak ke sungai lebih dari 30 meter. Selain itu 0.
75	`rumus = "A*%.6f + B*%.6f + C*%.6f + D*%.6f" % tuple(w)`	Susun rumus jumlah berbobot, dengan empat bobot dimasukkan ke teks. `tuple(w)` mengubah larik menjadi tuple supaya cocok dengan `%`.
76-78	`processing.run("gdal:rastercalculator", {... "FORMULA": rumus ...})`	Hitung skor akhir: huruf A sampai D adalah empat raster skor. Hasilnya `Kesesuaian_skor_mentah.tif` (`RTYPE: 5` berarti angka desimal).
79-81	`processing.run("gdal:rastercalculator", {... (B==1)*(...) ...})`	Ubah skor ke kelas: skor 3,2 ke atas kelas 1 (S1), 2,6 sampai 3,2 kelas 2 (S2), 2,0 sampai 2,6 kelas 3 (S3), di bawah 2,0 kelas 4 (N). Faktor `(B==1)` membuat sel terlarang menjadi 0.
82	`kelas = baca(P("Kesesuaian_kelas.tif"))`	Baca peta kelas sebagai larik.
83-87	`label = {...}, for kd in range(5): print(...)`	Kamus nama kelas, lalu cetak jumlah sel, luas (hektare), dan persen tiap kelas. 1 sel = 100 m2 = 0,01 ha. `%-22s` rata kiri 22 karakter, `%%` mencetak tanda persen.

Blok 6, baris 89-100: pemeriksaan silang dan kepekaan.

Baris	Tulisan	Artinya
90-93	`s = [baca(...) ...], skor = sum(wi * si for wi, si in zip(w, s)), boleh = ... > JARAK_SUNGAI, np.select(...)`	Hitung ulang semuanya dengan NumPy murni, tanpa alat QGIS. Baris 92 memakai `JARAK_SUNGAI` yang sama dengan rumus QGIS. `np.select` memilih kelas menurut daftar syarat, `np.where` menerapkan kendala.
94	`print("Pemeriksaan NumPy: sel kelas sama = %.4f%%" % ...)`	Cetak berapa persen sel yang kelasnya sama dengan hasil QGIS. Seharusnya 100.
97-100	`skor2 = sum(0.25 * si for si in s) ...`	**Kepekaan.** Ulangi dengan bobot sama rata (0,25 tiap kriteria), lalu cetak luas S1 dan persen sel yang berubah kelas.

Blok 7, baris 102-124: per petak dan poligon.

Baris	Tulisan	Artinya
103-105	`processing.run("native:zonalhistogram", {...}) ["OUTPUT"]`	Hitung berapa sel tiap kelas di dalam tiap petak. Kolom hasil diawali `k_` (`k_1`, `k_2`, dan seterusnya).
106-108	`baris = [], for f in zh.getFeatures(): baris.append((...))`	Kumpulkan satu tuple per petak: kode, jenis, lalu luas lima kelas dalam hektare (sel kali 0,01). Tanda `*` di depan kurung membongkar isi hasil ke dalam tuple.

Baris	Tulisan	Artinya
109	`baris.sort(key=lambda r: -r[2])`	Urutkan petak dari S1 terluas. Lambda mengambil elemen ke-3 (`r[2]`, luas S1) dan membalik tandanya agar menurun.
110-113	`for r in baris[:5]: print(" %s %-12s ..." % r)`	Cetak lima petak teratas, lalu total S1 dari semua petak.
114-118	`processing.run("gdal:polygonize", ...), native:dissolve`	Ubah peta kelas menjadi poligon per kelas, lalu simpan `Kesesuaian_kelas.gpkg`. Hasil lama dihapus lebih dulu.
119-123	`for tmp in (...): try: os.remove(...) except OSError: pass`	Hapus dua berkas sementara. Bila gagal (berkas tidak ada), abaikan saja (`pass`).
124	`print("\npoligon kelas ditulis:", ...)`	Pesan akhir.

Hasil. Layar menampilkan (peringatan GDAL tentang lebar kolom teks boleh diabaikan):

HASIL

```
bobot AHP: {'lereng': np.float64(0.467), 'tanah': np.float64(0.277), 'curah hujan': np.float64(0.16), 'jarak
jalan': np.float64(0.095)} | lambda_maks = 4.031 | CI = 0.0103 | CR = 0.0115

Luas per kelas (ha), 1 sel = 100 m2 = 0,01 ha:
  Terlarang (kendala)      5321 sel    53.21 ha   13.3%
  S1 Sangat sesuai         6733 sel    67.33 ha   16.8%
  S2 Cukup sesuai         12345 sel   123.45 ha   30.9%
  S3 Sesuai marginal      11627 sel   116.27 ha   29.1%
  N Tidak sesuai           3974 sel    39.74 ha    9.9%
  jumlah: 400.00 ha (sel NoData: 0)

Pemeriksaan NumPy: sel kelas sama = 100.0000%
Kepekaan: luas S1 bobot AHP = 67.33 ha; bobot sama rata = 81.76 ha; sel yang berubah kelas = 31.6%

Lima petak dengan S1 terluas (ha): kode, jenis, S1, S2, S3, N, Terlarang
  P-05 Jati          11.95   5.36   0.36   0.01   1.47
  P-03 Mahoni         9.02   5.27   1.63   0.18   3.91
  P-08 Jati           4.46   5.46   2.96   0.00   2.95
  P-17 Jati           4.26   4.78   3.55   1.47   5.84
  P-06 Akasia         4.24   4.95   1.83   0.01   2.53
  total S1 dari 25 petak = 67.33 ha

poligon kelas ditulis: True
```

Cara membaca: lereng paling berpengaruh (bobot 0,467). Penilaian konsisten (`CR` 0,0115, jauh di bawah 0,1). Kelas S1 seluas 67,33 ha (16,8 persen), dan 13,3 persen wilayah terlarang oleh kendala. Pemeriksaan NumPy cocok 100 persen. Bila bobot disama-ratakan, S1 melebar ke 81,76 ha: pilihan bobot sangat menentukan. Petak P-05 adalah yang paling banyak S1.

Bagian yang boleh diubah.

- **Baris 36, matriks AHP.** Ubah angka perbandingan. Jaga agar sisi bawah tetap kebalikan sisi atas.
- **Baris 48-49 dan 68, tabel skor.** Ubah batas rentang dan skor.
- **Baris 20, jarak aman sungai.** Ubah angka `30` (meter) di `JARAK_SUNGAI`. Cukup di satu tempat: rumus kendala (baris 72) dan pemeriksaan NumPy (baris 92) ikut berubah.

- **Baris 80, batas kelas.** Ubah `3.2`, `2.6`, dan `2.0`.
- **Baris 86, luas sel.** `0.01` hektare benar untuk sel 10 meter.

Jebakan.

- **Bergantung pada 25.7 dan 25.8.** Bila `CH_tahunan.tif` belum ada, hasilnya `QgsProcessingException: Unable to execute algorithm` dengan keterangan `Could not load source layer for INPUT_RASTER: ...CH_tahunan.tif not found`.
- **Satu aturan, satu nama.** Syarat sungai 30 meter disimpan di `JARAK_SUNGAI` (baris 20), lalu dipakai di rumus QGIS (baris 72) dan pemeriksaan NumPy (baris 92). Karena itu keduanya selalu sama dan pemeriksaan tetap 100 persen. Jangan menulis angka `30` langsung di dua baris itu. Bila ada yang diganti di satu tempat saja, kedua hitungan berbeda dan angka pemeriksaan turun di bawah 100 persen. Itulah alasan angka dijadikan satu nama.
- **Skor 0 punya dua arti.** `NO_DATA: 0` memberi skor 0 pada nilai di luar rentang, dan baris 72 memakai `A>0` sebagai tanda "lereng terlalu curam". Bila Anda menambah skor sah 0, kendalanya bisa tercampur. [kemungkinan]
- **Jenis tanah tak terdaftar mendapat skor 1.** `ELSE 1` di rumus tanah (baris 53) berlaku untuk nama tanah apa pun yang tidak disebut.
- **CR memakai angka acuan tetap.** `0.90` hanya benar untuk matriks 4 x 4. Matriks lain butuh angka acuan lain.
- **Luas sel tertulis tetap.** `0.01` ha di beberapa tempat. Salah bila sel bukan 10 meter.
- **Bobot dicetak dengan tulisan aneh.** Tulisan `np.float64(...)` itu bentuk cetak NumPy versi baru, bukan galat.
- **Kriteria fiktif.** Jangan memakai ambang ini untuk keputusan nyata.

Latihan variasi. Perlebar jarak aman dari sungai menjadi 50 meter. Contoh jawaban: ganti `JARAK_SUNGAI = 30` menjadi `JARAK_SUNGAI = 50` di baris 20, cukup di satu tempat. Layar menampilkan antara lain `Terlarang (kendala) 8249 sel 82.49 ha 20.6%` dan `S1 Sangat sesuai 6162 sel 61.62 ha 15.4%`, dan `Pemeriksaan NumPy: sel kelas sama = 100.0000%`.

> **Pola yang bisa dipakai ulang:** Kesesuaian lahan: **ubah tiap kriteria jadi skor (reklasifikasi)**, **hitung bobot (AHP)**, **jumlahkan berbobot**, **terapkan kendala**, lalu **periksa silang dengan NumPy**. Ganti empat hal: daftar kriteria, tabel skor, matriks perbandingan, dan ambang kelas.

Skrip 25.10: Cadangan dan pemulihan dengan Bash

Berkas `m2_07d_cadangan.sh`. Ini bukan Python, melainkan skrip **Bash**: bahasa perintah di terminal Linux, macOS, dan Git Bash di Windows. Isinya empat perintah PostgreSQL yang dibungkus supaya bisa diulang. Cadangan yang belum pernah diuji pemulihannya bukan cadangan, jadi skrip ini juga mengujinya.

Tujuan. Menyalin skema `kph` ke berkas cadangan (`pg_dump`), memulihkannya ke basis data baru (`pg_restore`), lalu membandingkan jumlah baris tabel asal dan hasil pulih.

Bahan. Alat PostgreSQL (`pg_dump`, `pg_restore`, `psql`) yang ada di PATH (daftar folder pencarian perintah) atau lewat variabel `PG_BIN`. Basis data `kph_contoh` dengan skema `kph` dan tabel `petak`, `kejadian`, `jalan`, `sungai`, `kesesuaian`, dan `lembar_keputusan` (tiga terakhir berasal dari hasil Skrip

25.7, 25.9, dan `m2_07a_lembar_keputusan.sql`). Hak membuat basis data. Kata sandi lewat `pgpass` atau `PGPASSWORD`, tidak ditulis di skrip. Perintah:

```
TERMINAL
bash m2_07d_cadangan.sh D:/Latihan/cadangan
```

> **Baris 11 menghapus basis data:** Perintah `DROP DATABASE IF EXISTS kph_pulih` menghapus basis data bernama `kph_pulih` bila sudah ada, beserta isinya. Pastikan nama itu tidak dipakai data Anda yang lain. Pada uji untuk bab ini, nama di skrip diganti ke nama basis data uji sementara, karena nama aslinya mungkin dipakai pihak lain.

Skrip utuh:

```
TERMINAL
#!/usr/bin/env bash
# M2 Bab 7: cadangan (pg_dump) dan pemulihan (pg_restore) skema kph, lalu pemeriksaan jumlah baris.
# Penulis: Badar Mubarok Yogaswara. Koneksi lewat PGHOST, PGPORT, PGUSER (kata sandi: berkas pgpass atau
PGPASSWORD, JANGAN ditulis di skrip).
# Pemakaian: bash m2_07d_cadangan.sh <folder tujuan berkas cadangan> ; PG_BIN bisa diisi folder bin PostgreSQL
bila tidak ada di PATH.
set -euo pipefail
B="${PG_BIN:-}"; [ -n "$B" ] && B="$B/"
TUJUAN="${1:-.}"
BERKAS="$TUJUAN/kph_contoh.dump"
"${B}pg_dump" -Fc -n kph -f "$BERKAS" kph_contoh                                    # -Fc = format kustom
(terkompres), -n = hanya skema kph
echo "ukuran cadangan: $(wc -c < "$BERKAS") byte"
"${B}psql" -d postgres -c "DROP DATABASE IF EXISTS kph_pulih" -c "CREATE DATABASE kph_pulih"
"${B}psql" -d kph_pulih -c "CREATE EXTENSION postgis" -c "CREATE EXTENSION pgrouting"   # ekstensi dibuat dulu
di basis data tujuan
"${B}pg_restore" -d kph_pulih "$BERKAS"
for t in petak kejadian jalan sungai kesesuaian lembar_keputusan; do
  a=$("${B}psql" -d kph_contoh -Atc "SELECT count(*) FROM kph.$t"); b=$("${B}psql" -d kph_pulih -Atc "SELECT
count(*) FROM kph.$t")
  echo "$t asal=$a pulih=$b $([ "$a" = "$b" ] && echo SAMA || echo BEDA)"
done
```

Penjelasan baris demi baris. Nomor baris mengikuti berkas di atas.

Baris	Tulisan	Artinya
1	`#!/usr/bin/env bash`	Baris pembuka: skrip ini dijalankan dengan Bash.
2-4	`# M2 Bab 7: ...`	Komentar: tujuan, cara koneksi, dan cara menjalankan.
5	`set -euo pipefail`	Pengaman. Hentikan skrip bila ada perintah gagal (`-e`), bila ada variabel yang belum diisi (`-u`), atau bila salah satu perintah dalam rangkaian gagal (`pipefail`).
6	`B="${PG_BIN:-}"; [ -n "$B" ] && B="$B/"`	Isi `B` dengan isi `PG_BIN` (kosong bila tidak diisi). Bila tidak kosong, tambahkan garis miring di akhir. Jadi `B` adalah awalan folder untuk alat PostgreSQL.

Baris	Tulisan	Artinya
7	`TUJUAN="${1:-.}"`	Folder tujuan adalah argumen pertama. Bila tidak diberikan, pakai titik, yaitu folder sekarang.
8	`BERKAS="$TUJUAN/kph_contoh.dump"`	Nama berkas cadangan.
9	`"${B}pg_dump" -Fc -n kph -f "$BERKAS" kph_contoh`	Buat cadangan. `-Fc` format kustom terkompres, `-n kph` hanya skema `kph`, `-f` nama berkas keluaran, dan `kph_contoh` basis data sumber.
10	`echo "ukuran cadangan: $(wc -c < "$BERKAS") byte"`	Cetak ukuran berkas. Tanda `$(...)` menjalankan perintah di dalamnya dan memakai hasilnya. `wc -c` menghitung byte.
11	`"${B}psql" -d postgres -c "DROP DATABASE IF EXISTS kph_pulih" -c "CREATE DATABASE kph_pulih"`	Hapus lalu buat basis data kosong `kph_pulih` sebagai tempat pemulihan.
12	`"${B}psql" -d kph_pulih -c "CREATE EXTENSION postgis" -c "CREATE EXTENSION pgrouting"`	Aktifkan ekstensi PostGIS dan pgRouting lebih dulu di basis data tujuan, sebab tabel membutuhkannya.
13	`"${B}pg_restore" -d kph_pulih "$BERKAS"`	Pulihkan cadangan ke basis data baru.
14-17	`for t in petak kejadian ...; do ... done`	Untuk enam tabel: hitung baris di basis data asal (`a`) dan hasil pulih (`b`) dengan `psql -Atc "SELECT count(*) ..."` (`-A` tanpa penyejajaran, `-t` tanpa judul, `-c` perintah). Lalu cetak `SAMA` bila `a` dan `b` sama, `BEDA` bila tidak.

Hasil. Pada uji, layar menampilkan:

```
HASIL
ukuran cadangan: 114417 byte
DROP DATABASE
CREATE DATABASE
CREATE EXTENSION
CREATE EXTENSION
petak asal=25 pulih=25 SAMA
kejadian asal=150 pulih=150 SAMA
jalan asal=37 pulih=37 SAMA
sungai asal=30 pulih=30 SAMA
kesesuaian asal=5 pulih=5 SAMA
lembar_keputusan asal=25 pulih=25 SAMA
```

Angka byte berbeda tiap basis data, jadi angka Anda mungkin lain. Bila basis data `kph_pulih` belum ada, `psql` juga mencetak pemberitahuan `NOTICE` bahwa basis data itu tidak ada dan dilewati. Itu normal. Enam tabel sama persis, jadi cadangan terbukti bisa dipulihkan.

Bagian yang boleh diubah.

- **Baris 9 dan 15, nama basis data sumber.** Ganti `kph_contoh`.
- **Baris 11-13 dan 15, nama basis data pulih.** Ganti `kph_pulih`. Ubah di semua tempat.

- **Baris 14, daftar tabel.** Tambah atau kurangi sesuai isi skema Anda.
- **Argumen folder tujuan.** Ganti folder di perintah menjalankan.

Jebakan.

- **Alat PostgreSQL tidak ada di PATH.** Bila `pg_dump` tidak ditemukan, skrip berhenti di baris 9 dengan `pg_dump: command not found`. Isi `PG_BIN` dengan folder `bin` PostgreSQL, atau tambahkan ke PATH. Karena `set -e`, tidak ada perintah penghapus yang sempat jalan.
- **Basis data pulih terhapus.** Baris 11 menghapus basis data dengan nama itu. Ini aksi destruktif.
- **Tabel belum ada.** Tabel `sungai`, `kesesuaian`, dan `lembar_keputusan` berasal dari langkah lain. Bila belum ada di basis data Anda, `psql` berhenti dengan galat. [kemungkinan]
- **Peran dan hak tidak ikut.** Cadangan hanya skema `kph`. Peran pengguna dan hak akses perlu disiapkan terpisah di server tujuan. [kemungkinan]
- **Kata sandi.** Skrip tidak membawanya. Siapkan `pgpass`.
- **Bukan Python.** Skrip ini tidak jalan dengan `python`. Pakai `bash`.

Latihan variasi. Tambahkan tabel `stasiun_hujan` ke daftar pemeriksaan. Contoh jawaban: ubah baris 14 menjadi:

```
TERMINAL
for t in petak kejadian jalan sungai kesesuaian lembar_keputusan stasiun_hujan; do
```

Tabel `kph.stasiun_hujan` berisi 30 baris pada data ini, jadi baris tambahan yang tercetak adalah `stasiun_hujan asal=30 pulih=30 SAMA` (diuji pada salinan skrip).

> **Pola yang bisa dipakai ulang:** Cadangan yang bisa dipercaya: **buat cadangan**, **pulihkan ke basis data baru**, lalu **bandingkan jumlah baris tiap tabel penting**. Ganti tiga hal: nama basis data sumber dan pulih, folder tujuan, dan daftar tabel.

Sembilan berkas SQL yang hanya disebut

Folder skrip M2 juga berisi sembilan berkas SQL. Berkas-berkas ini bukan Python, jadi tidak dibahas baris demi baris di sini. Anda menjalankannya dengan `psql -f nama_berkas.sql`. Dua di antaranya (`m2_03b_pgrouting.sql` dan `m2_07a_lembar_keputusan.sql`) dijalankan pada uji bab ini. Tujuh lainnya hanya dibaca judulnya. [CEK]

Berkas	Isi	Hubungan dengan skrip
`m2_01a_skema.sql`	Mengaktifkan PostGIS, membuat skema `kph`, dan dua peran akses	Dijalankan sebelum Skrip 25.1
`m2_01d_aturan_data.sql`	Menghapus tabel `fasilitas_uji` buatan Skrip 25.2, lalu membuat aturan data (daftar nilai yang sah)	Sesudah Skrip 25.2
`m2_02a_kueri_dasar.sql`	SQL spasial dasar: luas petak, kejadian per petak, dan sebagainya	Sumber kueri yang disilang di Skrip 25.3 dan 25.4

Berkas	Isi	Hubungan dengan skrip
`m2_02b_indeks_dan_view.sql`	Indeks spasial, rencana kueri, view	Satu bab dengan Skrip 25.3 dan 25.4
`m2_03b_pgrouting.sql`	Membuat simpul dan ruas pgRouting dari jalan	Wajib sebelum Skrip 25.6
`m2_04b_sungai_petak.sql`	Menghubungkan sungai dan DAS dengan petak	Memakai hasil Skrip 25.7
`m2_07a_lembar_keputusan.sql`	Tabel lembar keputusan per petak, gabungan semua hasil	Memakai hasil Skrip 25.7 sampai 25.9
`m2_07b_tata_kelola.sql`	Hak akses, jejak perubahan, pemeliharaan	Satu bab dengan Skrip 25.10
`m2_07c_uji_hak.sql`	Menguji hak akses dengan pergantian peran	Sesudah `m2_07b_tata_kelola.sql`

Cek paham

1. Mengapa skrip PostGIS di bab ini tidak memuat alamat server atau kata sandi, dan dari mana skrip tahu ke mana harus tersambung?
2. Apa arti `HASIL SAMA: True` pada Skrip 25.3, dan mengapa membandingkan dua sistem berguna?
3. Mengapa Skrip 25.9 harus dijalankan setelah Skrip 25.7 dan 25.8?

Jawaban.

1. Karena kata sandi dan alamat tidak boleh bocor lewat berkas skrip. Skrip membaca `PGHOST`, `PGPORT`, `PGUSER`, dan `PGDATABASE` dari variabel lingkungan, dan sandi dari berkas `pgpass` atau `PGPASSWORD`.
2. Tiga jawaban (total luas, tiga petak teratas, kejadian dekat jalan) dari GeoPackage persis sama dengan dari PostGIS. Bila dua jalan berbeda memberi angka sama, hitungan kemungkinan besar benar.
3. Karena skrip itu membaca `Sungai.gpkg` (hasil 25.7) dan `CH_tahunan.tif` (hasil 25.8). Tanpa keduanya, alat berhenti dengan galat "tidak ditemukan".

Kesalahan umum

- **Lupa argumen folder.** Hampir semua skrip butuh folder paket. Perbaikan: ketik folder setelah nama skrip, atau ganti `sys.argv[1]` dengan jalur manual bila memakai Konsol QGIS.
- **Mengira Skrip 25.1 tidak menghapus apa-apa.** Skrip ini aman dijalankan ulang karena `OVERWRITE=YES` mengganti tabel lama. Akibatnya, perubahan yang Anda buat langsung di tabel hilang. Perbaikan: simpan perubahan di skrip atau berkas sumber, bukan di tabel hasil impor.
- **Mengganti angka di satu tempat, padahal ada di beberapa tempat.** Contohnya jarak 100 meter (Skrip 25.3) dan nomor simpul (Skrip 25.6). Perbaikan: cari semua tulisan angka itu di skrip. Skrip 25.9 sudah menyimpan jarak sungai dalam satu nama, `JARAK_SUNGAI`, sehingga cukup diganti sekali.
- **Menganggap "dilewati" tidak penting.** Pesan `Pembanding PostGIS dilewati` bisa berarti server mati atau salah ketik. Perbaikan: baca pesan sesudah titik dua.

- **Memberi nama berkas sama dengan modul bawaan Python.** Berkas bernama `enum.py` di folder kerja membuat Python QGIS gagal dengan `ImportError: cannot import name 'NULL' from partially initialized module` (terjadi pada uji bab ini). Perbaikan: beri nama lain.

Ringkasan dan latihan

- **PostGIS** adalah gudang data bersama. Skrip bicara kepadanya lewat GDAL (Skrip 25.1), QGIS (25.2), dan `psycopg2` (25.3, 25.6).
- Koneksi dibaca dari **variabel lingkungan** `PG*`. Kata sandi tidak ditulis di skrip.
- **Menyilang** dua cara (GeoPackage dan PostGIS, QGIS dan pgRouting, alat QGIS dan NumPy) adalah cara terbaik menguji hitungan sendiri.
- **Lambda** adalah fungsi mini satu baris. Boleh ditulis ulang sebagai `def` bila lebih mudah dibaca.
- Analisis lahan (hidrologi, interpolasi, kesesuaian) berantai: keluaran satu skrip menjadi masukan skrip berikutnya.
- Jangan lupa menguji cadangan dengan memulihkannya (Skrip 25.10).

Latihan. Pilih satu skrip di bab ini dan temukan satu angka yang tertulis tetap (misalnya `100` di Skrip 25.3 atau `500`). Tulis di bagian mana saja angka itu muncul, dan apa yang terjadi bila Anda mengubahnya hanya di satu tempat.

Tabel perbandingan: alat apa untuk pekerjaan apa

Pekerjaan	Cara di bab ini	Skrip
Memasukkan berkas ke PostGIS	`gdal.VectorTranslate` dengan tujuan `PG:`	25.1
Memuat tabel PostGIS di QGIS	`QgsDataSourceUri` dan pembaca `"postgres"`	25.2
Menanyakan data dengan SQL	`ExecuteSQL` (GeoPackage), `psycopg2` (PostGIS), `qgis:executesql` (layer QGIS)	25.3, 25.4
Rute dan area layanan	`native:shortestpathpointtopoint`, pgRouting	25.5, 25.6
Sungai dan DAS dari DEM	`native:fillsinkswangliu` dan alat GRASS	25.7
Peta dari titik pengamatan	Alat interpolasi QGIS, GRASS, dan NumPy	25.8
Kesesuaian lahan	Reklasifikasi, AHP, dan kalkulator raster	25.9
Cadangan basis data	`pg_dump` dan `pg_restore` lewat Bash	25.10

BAB 26: Hutan dalam Film Pendek: Deret Waktu, Perubahan, dan Klasifikasi dengan Skrip

> **Berdasarkan Seri M3:** Skrip bab ini berasal dari Buku M3, *Deret Waktu dan Pemodelan* (Master Class), berkas `1_1_susun_ndvi.py` sampai `7_1_validasi.py` (empat belas skrip), ditambah dua berkas pembantu, `m3_umum.py` dan `hutan_mini.py`. Di buku itu, deret waktu dibahas lewat menu dan klik QGIS. Bab ini menunjukkan sisi skripnya.

Studi kasus: "Apa yang terjadi sejak dua tahun lalu?"

Kepala Seksi menaruh dua puluh empat citra satelit KPH Contoh di meja Anda. Pertanyaannya bertubi-tubi. Di mana hutan hilang, kapan, dan seberapa luas? Mana yang terbakar, dan benarkah titik panas itu? Kalau semua dijawab lewat klik, Anda mengulang pekerjaan dua puluh empat kali. Dengan skrip, tiap pertanyaan menjadi satu berkas Python yang bisa dijalankan ulang kapan saja.

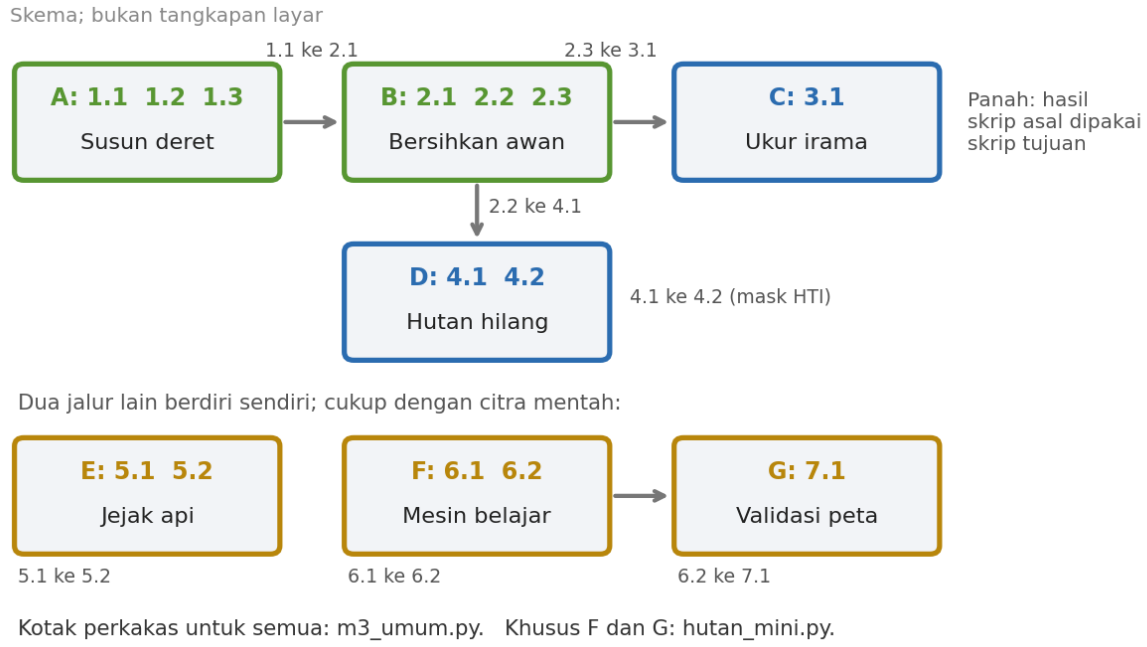
Konsep: deret waktu dalam tiga kalimat

Deret waktu itu seperti **buku lipat bergambar**: tiap lembar hampir sama, dan gerakan baru terlihat saat lembarnya dibalik berurutan. Satu citra hanya memotret satu hari. Puluhan citra yang ditumpuk menurut tanggal memberi tiap piksel sebuah **riwayat angka**, dan riwayat itu bisa dibaca seperti detak jantung.

Lima istilah yang perlu Anda kenal sekarang:

- **Deret waktu**: kumpulan citra lokasi yang sama pada tanggal berbeda. Di bab ini ada 24 tanggal, dari 5 Januari 2024 sampai 25 Desember 2025.
- **NDVI**: angka kehijauan tumbuhan, dihitung dari cahaya merah dan inframerah dekat. Hutan sehat bernilai sekitar 0,85. Awan, tanah terbuka, dan air bernilai jauh lebih rendah.
- **SCL**: peta penanda yang menyertai tiap citra. Isinya kode kelas tiap piksel, misalnya vegetasi, air, atau awan.
- **Fitur**: angka ringkas yang menggambarkan perilaku satu piksel. Fitur menjadi bahan belajar bagi mesin.
- **Validasi**: pemeriksaan apakah peta hasil boleh dipercaya.

Ilustrasi 26.1: Peta jalan bab ini



Bab ini panjang karena isinya empat belas skrip. Bacalah per kelompok, dan jangan terburu-buru ingin menghafal. Tiap kelompok punya **Konsep** dan gambar sendiri.

Kelompok	Skrip	Pertanyaan yang dijawab
A. Menyusun deret	26.1 sampai 26.3	Bagaimana 24 berkas menjadi satu tumpukan bertanggal?
B. Membersihkan awan	26.4 sampai 26.6	Bagaimana membuang awan dan menambal lubangnya?
C. Mengukur irama	26.7	Seperti apa irama normal tiap tutupan lahan?
D. Hutan hilang	26.8 dan 26.9	Di mana, seberapa luas, dan kapan hutan hilang?
E. Jejak api	26.10 dan 26.11	Berapa hektare yang terbakar, dan titik panas mana yang benar?
F. Mesin belajar	26.12 dan 26.13	Bisakah satu peta menyebut semua kelas sekaligus?
G. Validasi	26.14	Seberapa boleh peta itu dipercaya?

Tiap skrip bernomor `Skrip 26.n`. Di dalam berkasnya mungkin tertulis nomor lain, misalnya `SKRIP 1.1`, mengikuti penomoran Buku M3. Bab ini selalu menyebut nama berkas aslinya.

Persiapan: folder, data, dan cara menjalankan

Semua skrip bab ini memakai **satu paket data** dan **satu folder hasil**. Ketiganya harus Anda atur sebelum menjalankan skrip pertama.

Bahan. Paket data `paket-m3`. Isinya 24 citra Sentinel-2 buatan (berkas `_refl.tif` berisi enam band, berkas `_SCL.tif` berisi kode kelas), folder `acuan` (kunci jawaban), folder `vektor` (titik panas, titik latih, kawasan hutan tanaman), dan daftar tanggal `Daftar_Citra.csv`. Semua data sintetis, jadi hasilnya bukan kondisi hutan nyata. Luas area 400 m kali 400 m, atau 40 kali 40 piksel, dengan piksel 10 m.

1. Buat satu folder untuk skrip, misalnya `D:/Latihan_M3/skrip`. Salin ke sana keempat belas skrip dan dua berkas pembantu (`m3_umum.py`, `hutan_mini.py`). Semuanya harus satu folder.
2. Letakkan `paket-m3` di tempat yang Anda tahu, misalnya `D:/Latihan_M3/paket-m3`.
3. Tentukan folder hasil, misalnya `D:/Latihan_M3/hasil`. Skrip membuatnya sendiri bila belum ada.
4. Beri tahu skrip lokasi paket dan hasil lewat dua **variabel lingkungan** (pengaturan di komputer yang bisa dibaca program): `M3_PAKET` dan `M3_HASIL`. Cara kerjanya dibahas di Bab 8.

Dari Konsol Python QGIS (Bab 3), langkah 4 dan menjalankan skrip pertama kira-kira seperti ini. Ketik tiap blok, lalu Enter. Konsol ini belum saya coba di jendela QGIS; urutannya saya ambil dari pola Bab 3 [CEK].

```python
import os, sys
os.environ["M3_PAKET"] = r"D:/Latihan_M3/paket-m3"
os.environ["M3_HASIL"] = r"D:/Latihan_M3/hasil"
sys.path.append(r"D:/Latihan_M3/skrip")
exec(open(r"D:/Latihan_M3/skrip/1_1_susun_ndvi.py", encoding="utf8").read())
```

Urutan itu penting. Kedua variabel harus diisi **sebelum** skrip pertama dijalankan, karena `m3_umum.py` membacanya saat pertama kali diimpor. Bila Python sudah terlanjur mengimpornya, mulai ulang Konsol.

Bila Anda menjalankan lewat terminal, pakai pola "menyalakan QGIS tanpa jendela" dari Bab 3. Semua hasil di bab ini saya uji dengan pola itu, pada QGIS 4.0.2, dengan paket data di folder kerja saya. Karena itu, jalur yang muncul di blok `HASIL` saya tuliskan sebagai `D:/Latihan_M3/hasil`. Jalur di layar Anda akan mengikuti folder Anda sendiri.

Urutan menjalankan. Beberapa skrip memerlukan hasil skrip lain. Jalankan sesuai urutan di tabel ini, dan baca baris pertama keluarannya sebelum lanjut.

Skrip	Memerlukan hasil	Menghasilkan
26.1 `1_1`	tidak ada	`NDVI_deret.tif`, folder `ndvi`
26.2 `1_2`, 26.3 `1_3`	26.1	grafik PNG, layer bertanggal
26.4 `2_1`	26.1	folder `ndvi_bersih`
26.5 `2_2`	26.4	enam komposit `komposit_ndvi_...`
26.6 `2_3`	tidak ada	`NDVI_isi.tif`
26.7 `3_1`	26.6	`metrik_ndvi.tif`, `anomali_ndvi.tif`
26.8 `4_1`	26.5	`mask_hti.tif`, `deforestasi_2waktu.tif`
26.9 `4_2`	26.8	`tanggal_deforestasi.tif`

Skrip	Memerlukan hasil	Menghasilkan
26.10 `5_1`	tidak ada	`bekas_bakar.tif`
26.11 `5_2`	26.10	`bekas_bakar.gpkg`
26.12 `6_1`	tidak ada	`fitur_deret.tif`
26.13 `6_2`	26.12	`peta_rf.tif`
26.14 `7_1`	26.12 dan 26.13	`peta_ndvi_saja.tif`

Dua kebiasaan akan menyelamatkan Anda sepanjang bab ini. Pertama, **baca baris pertama keluaran**. Kedua, **jangan percaya skrip yang tidak berteriak**: skrip yang tidak dijaga bisa tetap jalan dengan masukan yang salah dan menghasilkan angka yang keliru tanpa pesan galat. Skrip 26.4 memberi contoh penjagaannya: ia berhenti dengan pesan bila masukannya tidak terbaca. Bagian Jebakan Skrip 26.4 menjelaskannya.

Kotak perkakas: m3_umum.py

Empat belas skrip bab ini memulai dengan baris `import m3_umum as U`. Berkas `m3_umum.py` adalah **kotak perkakas** buatan penulis: kumpulan fungsi kecil yang dipakai bersama, supaya tiap skrip tidak mengulang pekerjaan yang sama. Bayangkan bengkel dengan satu lemari alat. Semua tukang mengambil obeng dari lemari itu, dan tidak ada yang membuat obeng sendiri.

Kata `U` hanyalah **nama pendek** (alias) untuk kotak itu. `U.daftar_citra()` berarti: "ambil perkakas `daftar_citra` dari kotak `U`". Cara kerja `import` dan alias dibahas di Bab 8, dan cara membuat fungsi sendiri di Bab 7.

Ilustrasi 26.2: Isi kotak perkakas m3_umum

Skema; bukan tangkapan layar



Tujuan. Menyediakan jalur data, daftar citra, pembaca dan penulis GeoTIFF, rumus NDVI dan NBR, serta penambal lubang awan, untuk dipakai semua skrip.

Bahan. Berkas ini tidak dijalankan sendiri. Ia diimpor oleh skrip lain dan harus satu folder dengan mereka. Ia memakai `numpy` dan `osgeo.gdal` (dibahas di Bab 12).

Skrip utuh:

```python
# -*- coding: utf-8 -*-
"""m3_umum.py: fungsi bantu bersama untuk skrip Seri M3. Penulis: Badar Mubarok Yogaswara
Letakkan berkas ini satu folder dengan skrip lain. Ubah PAKET dan HASIL sesuai komputer Anda,
atau isi variabel lingkungan M3_PAKET dan M3_HASIL."""
import csv
import datetime as dt
import os

import numpy as np
from osgeo import gdal

gdal.UseExceptions()

PAKET = os.environ.get("M3_PAKET", r"D:/Latihan_M3/paket-m3")     # folder paket data sintetis
HASIL = os.environ.get("M3_HASIL", r"D:/Latihan_M3/hasil")        # folder keluaran Anda
os.makedirs(HASIL, exist_ok=True)

BAND = {"B02": 0, "B03": 1, "B04": 2, "B08": 3, "B11": 4, "B12": 5}   # urutan band di berkas *_refl.tif
BAIK = (4, 5, 6)                                                       # kode SCL: vegetasi, bukan vegetasi,
air


def daftar_citra():
    """Baca Daftar_Citra.csv: kembalikan daftar (tanggal, jalur_refl, jalur_scl)."""
    hasil = []
    with open(os.path.join(PAKET, "citra", "Daftar_Citra.csv"), encoding="utf8") as f:
        for r in csv.DictReader(f):
            hasil.append((dt.date.fromisoformat(r["tanggal"]),
                          os.path.join(PAKET, "citra", r["berkas_refl"]),
                          os.path.join(PAKET, "citra", r["berkas_scl"])))
    return hasil


def baca_deret():
    """Kembalikan tanggal (daftar), refl (T,6,H,W), scl (T,H,W), geotransform, proyeksi."""
    tgl, refl, scl = [], [], []
    for d, pr, ps in daftar_citra():
        a = gdal.Open(pr)
        refl.append(a.ReadAsArray().astype("float32"))
        scl.append(gdal.Open(ps).ReadAsArray())
        gt, prj = a.GetGeoTransform(), a.GetProjection()
        tgl.append(d)
    return tgl, np.array(refl), np.array(scl), gt, prj


def nilai_ndvi(refl):
    nir, red = refl[:, BAND["B08"]], refl[:, BAND["B04"]]
    return (nir - red) / (nir + red + 1e-9)


def nilai_nbr(refl):
    nir, swir = refl[:, BAND["B08"]], refl[:, BAND["B12"]]
    return (nir - swir) / (nir + swir + 1e-9)


def bersih(scl):
    """True di piksel yang boleh dipakai (bukan awan, bayangan, atau cirrus)."""
    return np.isin(scl, BAIK)


def tulis_tif(path, arr, gt, prj, nodata=None, deskripsi=None, tipe=None):
    """Tulis larik (H,W) atau (B,H,W) ke GeoTIFF."""
    arr = np.asarray(arr)
```

```
    if arr.ndim == 2:
        arr = arr[None]
    if tipe is None:
        tipe = gdal.GDT_Float32 if arr.dtype.kind == "f" else gdal.GDT_Byte
    os.makedirs(os.path.dirname(path), exist_ok=True)
    ds = gdal.GetDriverByName("GTiff").Create(path, arr.shape[2], arr.shape[1], arr.shape[0], tipe,
["COMPRESS=LZW"])
    ds.SetGeoTransform(gt)
    ds.SetProjection(prj)
    for i in range(arr.shape[0]):
        b = ds.GetRasterBand(i + 1)
        b.WriteArray(arr[i])
        if nodata is not None:
            b.SetNoDataValue(nodata)
        if deskripsi:
            b.SetDescription(deskripsi[i])
    ds = None


def baca_tif(path):
    ds = gdal.Open(path)
    return ds.ReadAsArray(), ds.GetGeoTransform(), ds.GetProjection()


def piksel_ke_xy(gt, baris, kolom):
    return gt[0] + (kolom + 0.5) * gt[1], gt[3] + (baris + 0.5) * gt[5]


def xy_ke_piksel(gt, x, y):
    return int((gt[3] - y) / -gt[5]), int((x - gt[0]) / gt[1])


def isi_waktu(x, baik, hari):
    """Interpolasi linear sepanjang waktu. x dan baik berbentuk (T,H,W); hari = hari ke- tiap tanggal."""
    T, H, W = x.shape
    hasil = np.full(x.shape, np.nan, dtype="float64")
    for i in range(H):
        for j in range(W):
            ok = baik[:, i, j]
            if ok.any():
                hasil[:, i, j] = np.interp(hari, hari[ok], x[ok, i, j])
    return hasil
```

Penjelasan baris demi baris. Nomor baris mengikuti berkas di atas.

Baris	Tulisan	Artinya
1-4	`# -*- coding...` dan teks tiga kutip	Baris 1 memberi tahu Python bahwa berkas berisi huruf UTF-8. Baris 2-4 adalah **docstring** (keterangan modul, dibahas di Bab 7).
5-10	`import csv` sampai `from osgeo import gdal`	Panggil lima perkakas: `csv` (membaca tabel teks), `datetime` (tanggal, diberi nama pendek `dt`), `os` (jalur dan folder), `numpy` (hitungan larik, diberi nama pendek `np`), dan `gdal` (membaca serta menulis raster).
12	`gdal.UseExceptions()`	Minta GDAL berteriak (memunculkan galat) bila ada masalah, bukan diam-diam gagal.
14	`PAKET = os.environ.get("M3_PAKET", r"D:/Latihan_M3/paket-m3")`	Isi kotak `PAKET` dengan lokasi paket data. Python mencari pengaturan `M3_PAKET` dulu. Bila tidak ada, ia memakai jalur cadangan di ujung baris, yang menunjuk komputer penulis.

Baris	Tulisan	Artinya
15	`HASIL = os.environ.get("M3_HASIL", ...)`	Sama seperti baris 14, untuk folder hasil.
16	`os.makedirs(HASIL, exist_ok=True)`	Buat folder hasil bila belum ada. `exist_ok=True` berarti: bila sudah ada, tidak apa-apa.
18	`BAND = {"B02": 0, ...}`	**Kamus** (Bab 5): nama band lalu nomor urutnya di berkas `_refl.tif`. Nomor mulai dari 0.
19	`BAIK = (4, 5, 6)`	**Tuple** (Bab 5): kode SCL yang dianggap piksel bersih, yaitu vegetasi (4), bukan vegetasi (5), dan air (6).
22-30	`def daftar_citra():`	Fungsi yang membaca `Daftar_Citra.csv`. Baris 25-26 membuka berkas dan membaca tiap baris tabel sebagai kamus. Baris 27-29 menambahkan satu tuple ke daftar: tanggal, jalur berkas refl, jalur berkas SCL. Baris 30 mengembalikannya.
25	`with open(...) as f:`	Buka berkas dan **tutup otomatis** saat blok selesai (Bab 9).
27	`dt.date.fromisoformat(r["tanggal"])`	Ubah teks seperti `2024-01-05` menjadi objek tanggal sungguhan.
33-42	`def baca_deret():`	Fungsi yang membaca semua citra sekaligus. Hasilnya lima benda: daftar tanggal, larik reflektansi, larik SCL, geotransform, dan proyeksi.
36-41	`for d, pr, ps in daftar_citra():`	Untuk tiap tanggal: buka berkas refl (baris 37), simpan lariknya sebagai bilangan desimal 32 bit (baris 38), simpan larik SCL (baris 39), catat geotransform dan proyeksi (baris 40), simpan tanggal (baris 41).
42	`return tgl, np.array(refl), np.array(scl), gt, prj`	Susun daftar menjadi larik besar. Bentuk `refl` adalah (24, 6, 40, 40): tanggal, band, baris, kolom.
45-47	`def nilai_ndvi(refl):`	NDVI = (inframerah dekat dikurangi merah) dibagi (inframerah dekat ditambah merah). `1e-9` adalah angka sangat kecil agar tidak terjadi pembagian dengan nol.
50-52	`def nilai_nbr(refl):`	NBR = (inframerah dekat dikurangi SWIR2) dibagi (jumlahnya). Memakai band B08 dan B12.
55-57	`def bersih(scl):`	Hasilnya larik benar atau salah: benar bila kode SCL ada di `BAIK`. `np.isin` berarti "apakah angka ini termasuk dalam daftar".
60-78	`def tulis_tif(...)`	Menulis larik ke berkas GeoTIFF. Baris 63-64 menambah sumbu band bila larik hanya dua dimensi. Baris 65-66 memilih tipe data: desimal menjadi Float32, **selain itu menjadi Byte**. Baris 68 membuat berkas terkompres. Baris 71-77 menulis tiap band, nilai kosong (`nodata`), dan namanya. Baris 78 menutup berkas.
81-83	`def baca_tif(path):`	Membaca satu GeoTIFF. Mengembalikan tiga benda: larik, geotransform, proyeksi.
86-87	`def piksel_ke_xy(...)`	Mengubah nomor baris dan kolom piksel menjadi koordinat peta (titik tengah piksel).

Baris	Tulisan	Artinya
90-91	`def xy_ke_piksel(...)`	Kebalikannya: koordinat peta menjadi nomor baris dan kolom.
94-103	`def isi_waktu(x, baik, hari)`	Menambal lubang awan dengan **interpolasi linear** sepanjang waktu, piksel demi piksel. Skrip 26.6 membuat versi sendiri yang hampir sama.

Hasil. Perkakas ini bisa Anda coba langsung di Konsol, setelah dua variabel lingkungan terisi:

```
PYTHON
import m3_umum as U
d = U.daftar_citra()
print(len(d), d[0][0])
tgl, refl, scl, gt, prj = U.baca_deret()
print(refl.shape, scl.shape)
print(U.xy_ke_piksel(gt, 312100, 9996100))
```

Layar menampilkan (berdasarkan uji pada QGIS 4.0.2):

```
HASIL
24 2024-01-05
(24, 6, 40, 40) (24, 40, 40)
(30, 10)
```

Artinya: ada 24 tanggal, tiap tanggal enam band berukuran 40 kali 40, dan titik 312100, 9996100 jatuh di baris 30, kolom 10.

Bagian yang boleh diubah.

- **Baris 14-15, jalur cadangan.** Ganti dengan folder di komputer Anda, supaya tidak perlu mengisi variabel lingkungan tiap kali.
- Baris 19, `BAIK`. Tuple kelas SCL yang dianggap bersih. Menghapus angka 6 membuang piksel air dari analisis. Skrip 26.4 punya ekspresi sendiri, jadi ubahan ini tidak memengaruhinya.
- Baris 68, `COMPRESS=LZW`. Pilihan kompresi berkas hasil. Hapus bila Anda ingin berkas tanpa kompresi.

Jebakan.

- **Folder dibuat saat impor.** Baris 16 berjalan begitu `import m3_umum` dilakukan. Saya mengujinya: dengan `M3_HASIL` menunjuk folder yang belum ada, folder itu langsung muncul setelah impor, walau belum ada hasil apa pun. Salah ketik jalur akan menghasilkan folder baru yang salah.
- **Tipe data diam-diam berubah.** Baris 65-66 hanya mengenal Float32 dan Byte. Larik bilangan bulat besar terpotong tanpa pesan, kecuali Anda menyebut `tipe` sendiri. Kasus nyatanya ada di Skrip 26.9, yang memilih `gdal.GDT_Int32`. Buktinya saya tunjukkan di sana.
- **Jalur cadangan menunjuk komputer penulis.** Bila Anda lupa mengisi `M3_PAKET`, galatnya `FileNotFoundError` saat membuka `Daftar_Citra.csv`. Saya mengujinya dengan jalur paket yang salah, dan galat itu muncul dari baris 25.

- **Semua citra dianggap sebidang.** Baris 40 hanya mengingat geotransform citra terakhir. Bila citra Anda berbeda ukuran atau lokasi, hasilnya keliru tanpa peringatan.

Latihan variasi.

1. Cetak tanggal dan jalur berkas SCL untuk tanggal terakhir.
2. Hitung berapa persen piksel bersih (menurut `bersih`) pada seluruh deret.

Contoh jawaban:

```python
print(d[-1][0], d[-1][2])
print(round(float(U.bersih(scl).mean()), 3))
```

Baris pertama mencetak tanggal terakhir dan jalur SCL-nya. Baris kedua mencetak `0.747` pada data ini: sekitar 75 persen piksel-tanggal bersih, sisanya (25 persen) tertutup awan, bayangan, atau cirrus.

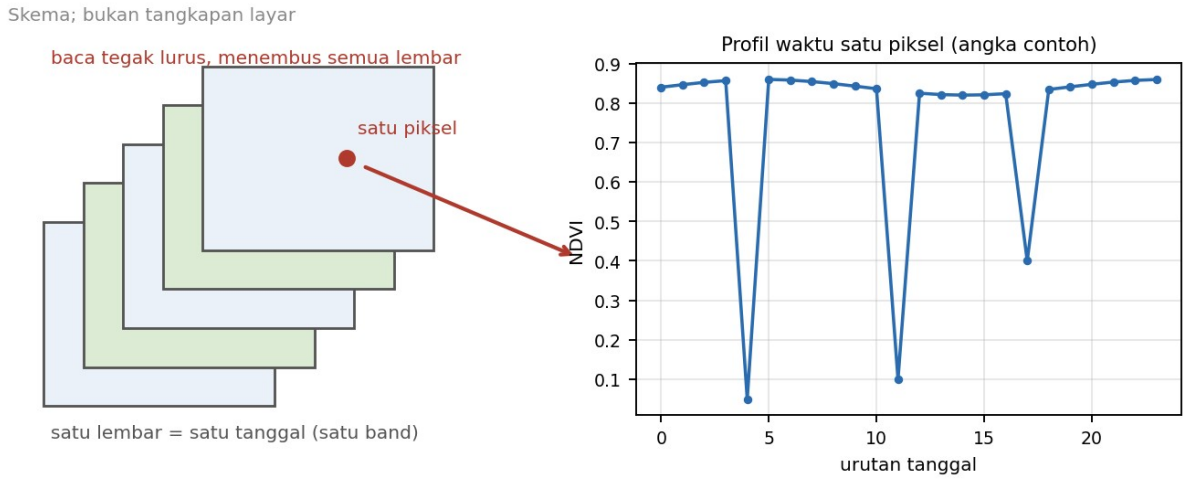
> **Pola yang bisa dipakai ulang:** Kotak perkakas adalah satu berkas berisi **jalur di atas, fungsi kecil di bawah**. Untuk proyek Anda sendiri, ganti dua hal: lokasi data di baris 14-15, dan isi fungsi yang Anda butuhkan. Jangan menyalin fungsi yang sama ke banyak skrip. Letakkan sekali di kotak perkakas, lalu panggil lewat nama pendeknya.

Kelompok A: Menyusun deret, dari 24 berkas menjadi satu tumpukan

Konsep: tumpukan bertanggal dalam tiga kalimat

Bayangkan 24 foto yang dijilid menjadi **album bertanggal**: satu halaman satu tanggal. Di komputer, album itu disebut **tumpukan** (*stack*): satu berkas raster dengan 24 **band**, dan tiap band adalah satu tanggal. Bila Anda menusuk album itu dengan jarum di satu titik, nilai-nilai yang tertusuk membentuk **profil waktu** titik itu.

Ilustrasi 26.3: Tumpukan dan profil waktu



Lima istilah yang perlu Anda kenal sekarang:

- **Tumpukan** (*stack*): satu berkas raster yang berisi banyak band.

- **Band**: satu lembar dalam tumpukan. Di bab ini, satu band berarti satu tanggal.
- **Profil waktu**: deretan nilai satu piksel sepanjang tanggal.
- **Kalkulator raster** (*Raster Calculator*): alat QGIS yang menghitung rumus di atas raster, piksel demi piksel.
- `processing.run`: perintah Python untuk memanggil alat QGIS, seperti menekan tombol alat itu dari skrip (dibahas di Bab 21).

Kelompok ini terdiri dari tiga skrip. Skrip 26.1 membuat tumpukannya. Skrip 26.2 membacanya dan menggambar grafik. Skrip 26.3 menempelkan tanggal supaya QGIS bisa memutarnya seperti film.

Skrip 26.1: Menyusun NDVI menjadi tumpukan 24 band

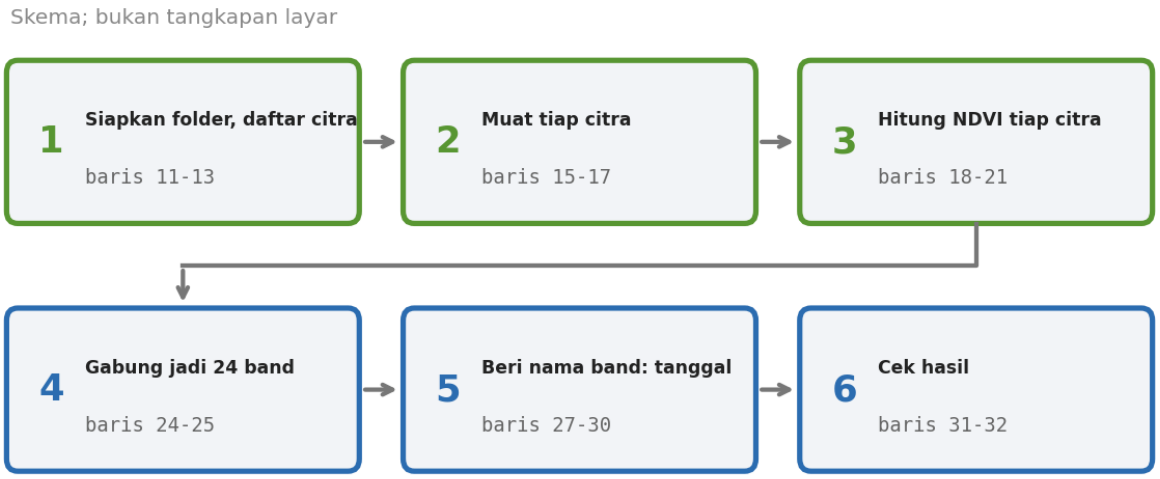
Berkas `1_1_susun_ndvi.py`. Di dalam berkasnya tertulis "SKRIP 1.1", mengikuti penomoran Buku M3.

Tujuan. Menghitung NDVI untuk tiap dari 24 citra, lalu menumpuk semuanya menjadi satu berkas 24 band yang tiap bandnya diberi tanggal.

Bahan. Paket data `paket-m3` (folder `citra`) dan kotak perkakas `m3_umum.py`. Jalankan dari Konsol Python QGIS, atau dari terminal dengan pola Bab 3. Waktu jalan sekitar sepuluh detik.

Skrip ini bekerja dalam enam tahap.

Ilustrasi 26.4: Alur Skrip 26.1



Skrip utuh:

```python
# -*- coding: utf-8 -*-
# SKRIP 1.1: Susun deret waktu NDVI dari 24 citra Sentinel-2 sintetis (jalankan di Python Console QGIS)
# Penulis: Badar Mubarok Yogaswara
# Hasil: NDVI per tanggal (hasil/ndvi/NDVI_YYYYMMDD.tif) dan satu tumpukan 24 band: hasil/NDVI_deret.tif
import os
import processing
from qgis.core import QgsRasterLayer, QgsProject
from osgeo import gdal
import m3_umum as U

os.makedirs(os.path.join(U.HASIL, "ndvi"), exist_ok=True)
daftar = U.daftar_citra()
```

```
keluaran = []
for tgl, pr, ps in daftar:
    nama = "S2_" + tgl.strftime("%Y%m%d") + "_refl"
    lyr = QgsRasterLayer(pr, nama)
    QgsProject.instance().addMapLayer(lyr, False)            # tanpa menampilkan di kanvas
    out = os.path.join(U.HASIL, "ndvi", "NDVI_%s.tif" % tgl.strftime("%Y%m%d"))
    ekspresi = '("%s@4" - "%s@3") / ("%s@4" + "%s@3")' % (nama, nama, nama, nama)   # band 4 = B08, band 3 =
B04
    processing.run("native:rastercalc", {"LAYERS": [lyr], "EXPRESSION": ekspresi, "OUTPUT": out})
    keluaran.append(out)

# gabungkan menjadi satu berkas: tiap berkas menjadi satu band (SEPARATE)
stack = os.path.join(U.HASIL, "NDVI_deret.tif")
processing.run("gdal:merge", {"INPUT": keluaran, "SEPARATE": True, "DATA_TYPE": 5, "OUTPUT": stack})
# beri nama band sesuai tanggal
ds = gdal.Open(stack, gdal.GA_Update)
for i, (tgl, _, _) in enumerate(daftar):
    ds.GetRasterBand(i + 1).SetDescription(tgl.isoformat())
ds = None
lyr = QgsRasterLayer(stack, "NDVI_deret")
print("Tumpukan:", stack, "valid:", lyr.isValid(), "band:", lyr.bandCount())
```

Penjelasan baris demi baris. Nomor baris mengikuti berkas di atas.

Baris	Tulisan	Artinya
1-4	`# -*- coding...` dan komentar	Baris 1 menandai berkas berhuruf UTF-8. Baris 2-4 menyebut judul, penulis, dan hasil yang dibuat.
5-9	`import os` sampai `import m3_umum as U`	Panggil perkakas: `os` (folder), `processing` (alat QGIS), `QgsRasterLayer` dan `QgsProject` (layer dan proyek), `gdal` (raster), dan kotak perkakas kita dengan nama pendek `U`.
11	`os.makedirs(os.path.join(U.HASIL, "ndvi"), exist_ok=True)`	Buat subfolder `ndvi` di dalam folder hasil. `os.path.join` menyambung jalur dengan pemisah yang benar.
12	`daftar = U.daftar_citra()`	Ambil 24 tuple dari kotak perkakas: tanggal, jalur refl, jalur SCL.
13	`keluaran = []`	Siapkan daftar kosong untuk mencatat jalur tiap NDVI yang dibuat.
14	`for tgl, pr, ps in daftar:`	Ulangi 24 kali. Tiap putaran, satu tuple dibongkar ke tiga kotak: `tgl`, `pr`, dan `ps`. Kotak `ps` tidak dipakai di skrip ini.
15	`nama = "S2_" + tgl.strftime("%Y%m%d") + "_refl"`	Susun nama layer, misalnya `S2_20240105_refl`. `strftime` mengubah tanggal menjadi teks menurut pola: tahun, bulan, hari.
16	`lyr = QgsRasterLayer(pr, nama)`	Buka berkas refl sebagai layer raster bernama `nama`.
17	`QgsProject.instance().addMapLayer(lyr, False)`	Daftarkan layer ke proyek supaya alat QGIS bisa menemukannya, tetapi `False` berarti tidak ditampilkan di kanvas.
18	`out = os.path.join(..., "NDVI_%s.tif" % ...)`	Susun jalur keluaran, misalnya `.../ndvi/NDVI_20240105.tif`. Tanda `%s` diganti oleh tanggal (Bab 9).

Baris	Tulisan	Artinya
19	`ekspresi = '("%s@4" - "%s@3") / ...' % (...)`	Susun rumus NDVI untuk kalkulator raster. `nama@4` berarti band 4 (inframerah dekat, B08), `nama@3` berarti band 3 (merah, B04). Empat `%s` diisi nama layer yang sama.
20	`processing.run("native:rastercalc", {...})`	Jalankan alat kalkulator raster dengan tiga isian: `LAYERS` (layer yang dipakai), `EXPRESSION` (rumus), `OUTPUT` (berkas hasil).
21	`keluaran.append(out)`	Catat jalur hasil di daftar.
24	`stack = os.path.join(U.HASIL, "NDVI_deret.tif")`	Tentukan jalur tumpukan akhir.
25	`processing.run("gdal:merge", {...})`	Jalankan alat Merge. `INPUT` berisi daftar 24 berkas. `SEPARATE: True` berarti tiap berkas menjadi band tersendiri, bukan ditempel berdampingan. `DATA_TYPE: 5` memilih Float32 (desimal 32 bit).
27	`ds = gdal.Open(stack, gdal.GA_Update)`	Buka tumpukan untuk **diubah**, bukan hanya dibaca.
28	`for i, (tgl, _, _) in enumerate(daftar):`	Ulangi untuk tiap tanggal. `enumerate` memberi nomor urut `i` (mulai 0). Tanda `_` berarti nilai itu sengaja diabaikan.
29	`ds.GetRasterBand(i + 1).SetDescription(tgl.isoformat())`	Beri band nomor `i + 1` (nomor band di GDAL mulai dari 1) nama berupa tanggal, misalnya `2024-01-05`.
30	`ds = None`	Lepas berkas. Di GDAL, inilah cara **menutup dan menyimpan** perubahan.
31-32	`lyr = QgsRasterLayer(stack, ...)` dan `print(...)`	Buka tumpukan sebagai layer, lalu cetak jalur, apakah valid, dan jumlah band.

Hasil. Layar menampilkan (berdasarkan uji pada QGIS 4.0.2):

```
HASIL
Tumpukan: D:/Latihan_M3/hasil\NDVI_deret.tif valid: True band: 24
```

Dari hasil ini Anda tahu: tumpukan terbaca (`valid: True`) dan memuat 24 band. Folder `ndvi` kini berisi 24 berkas NDVI, satu per tanggal.

Bagian yang boleh diubah.

- **Baris 19, nomor band.** Ganti `@3` menjadi `@6` di seluruh baris untuk menghitung NBR (inframerah dekat dan SWIR2) alih-alih NDVI. Dengan angka 6, rata-rata band pertama berubah dari `0.726` menjadi `0.586`, jadi indeksnya memang lain.
- Baris 25, `DATA_TYPE`. Angka 5 berarti Float32. Saya memeriksanya dengan bantuan alat (`qgis_process help gdal:merge`) dan dengan `gdalinfo` pada hasilnya. Angka lain memberi tipe lain, dan desimal NDVI bisa terpotong.
- **Baris 11 dan 18, nama folder dan berkas.** Ganti bila Anda ingin menyimpan di tempat lain.

Jebakan.

- **Nama layer harus sama persis dengan nama di rumus.** Saya mencoba mengganti satu nama di baris 19 dengan nama lain. Hasilnya galat `QgsProcessingException: Error occurred while performing calculation.` Pesannya tidak menyebut nama mana yang salah, jadi periksa ejaan di baris 15 dan 19.
- `import processing` hanya berjalan sendiri di Konsol QGIS. Saya mencobanya dari terminal biasa tanpa persiapan: layar menampilkan `ModuleNotFoundError: No module named 'processing'`. Nyalakan QGIS tanpa jendela lebih dulu (Bab 3).
- **Menjalankan ulang menimpa berkas lama.** Tidak ada pertanyaan "yakin?". Pindahkan hasil lama bila ingin menyimpannya.
- Kotak `ps` tidak dipakai. Python tidak protes terhadap kotak yang menganggur. Itu tanda skrip ini hanya memerlukan dua dari tiga isi tuple.

Latihan variasi.

1. Hitung **NBR** (inframerah dekat dan SWIR2) alih-alih NDVI.
2. Cetak juga nilai rata-rata band pertama tumpukan, untuk membandingkan dua indeks.

Contoh jawaban, hanya bagian yang berubah:

```python
    ekspresi = '("%s@4" - "%s@6") / ("%s@4" + "%s@6")' % (nama, nama, nama, nama)   # band 6 = B12
```

dan satu baris tambahan di akhir skrip:

```python
print("rata-rata band 1:", round(float(gdal.Open(stack).ReadAsArray()[0].mean()), 3))
```

Saya menjalankan versi NDVI dan versi NBR pada salinan data. Layar menampilkan `rata-rata band 1: 0.726` untuk NDVI dan `rata-rata band 1: 0.586` untuk NBR.

> **Pola yang bisa dipakai ulang:** Pola "ulangi untuk tiap berkas, panggil alat, catat hasil, gabung di akhir" cocok untuk pekerjaan serupa. Ganti tiga hal: **daftar masukan** (baris 12), **rumus** (baris 19), dan **alat yang dipanggil** (baris 20 dan 25). Sisanya, yaitu folder, nama keluaran, dan daftar penampung, hampir tidak berubah.

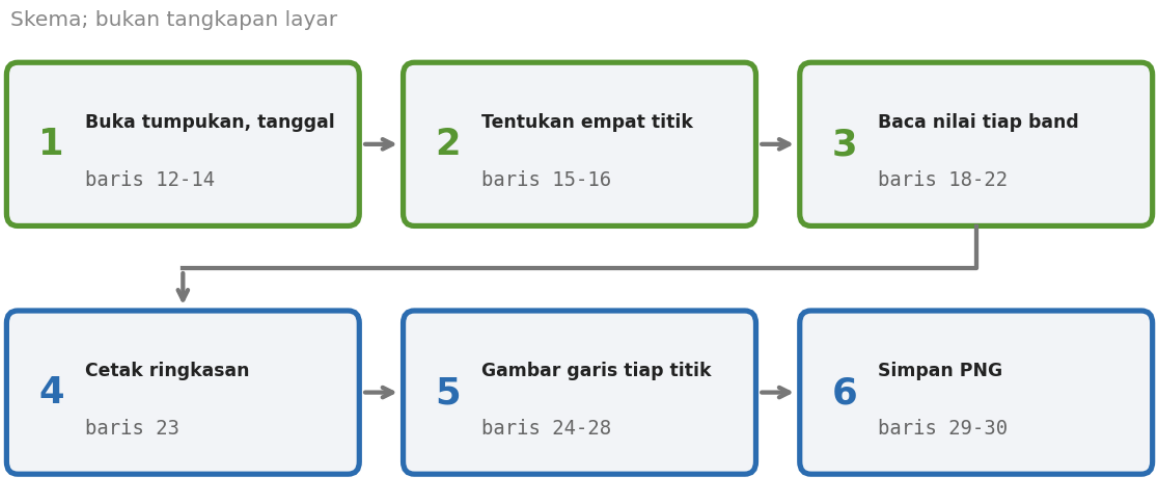
Skrip 26.2: Menggambar profil waktu empat titik

Berkas `1_2_profil_waktu.py`. Di dalam berkasnya tertulis "SKRIP 1.2".

Tujuan. Menggambar grafik NDVI sepanjang 24 tanggal untuk empat titik (hutan alam, sawah, bukaan, bekas bakar), mencetak ringkasannya, dan menyimpan grafik sebagai gambar.

Bahan. Hasil Skrip 26.1 (`NDVI_deret.tif`), `m3_umum.py`, dan `matplotlib` (perkakas membuat grafik, dibahas di Bab 12). Jalankan dari Konsol atau terminal.

Ilustrasi 26.5: Alur Skrip 26.2



Skrip utuh:

```python
# -*- coding: utf-8 -*-
# SKRIP 1.2: Profil waktu satu titik (NDVI sepanjang 24 tanggal) dari tumpukan NDVI_deret.tif
# Penulis: Badar Mubarok Yogaswara
import os
import numpy as np
import matplotlib
matplotlib.use("Agg")
import matplotlib.pyplot as plt
from qgis.core import QgsRasterLayer, QgsPointXY
import m3_umum as U

lyr = QgsRasterLayer(os.path.join(U.HASIL, "NDVI_deret.tif"), "NDVI_deret")
dp = lyr.dataProvider()
tanggal = [t for t, _, _ in U.daftar_citra()]
titik = {"Hutan alam": (312100, 9996100), "Sawah (Beta)": (312365, 9996025),
         "Bukaan (Gamma)": (312100, 9996330), "Bekas bakar (Beta)": (312270, 9996125)}
fig, ax = plt.subplots(figsize=(9, 4))
for nama, (x, y) in titik.items():
    nilai = []
    for b in range(1, lyr.bandCount() + 1):
        v, ok = dp.sample(QgsPointXY(x, y), b)
        nilai.append(v if ok else np.nan)
    print("%-20s NDVI awal %.2f, minimum %.2f, akhir %.2f" % (nama, nilai[0], np.nanmin(nilai), nilai[-1]))
    ax.plot(tanggal, nilai, marker="o", ms=3, label=nama)
ax.set_ylabel("NDVI")
ax.legend(fontsize=8)
ax.grid(alpha=0.3)
fig.autofmt_xdate()
fig.savefig(os.path.join(U.HASIL, "profil_waktu.png"), dpi=130, bbox_inches="tight")
print("Grafik tersimpan di", os.path.join(U.HASIL, "profil_waktu.png"))
```

Penjelasan baris demi baris. Hanya hal baru yang dijelaskan. Perkakas yang sudah dikenal di Skrip 26.1 tidak diulang.

Baris	Tulisan	Artinya
4-10	`import os` sampai `import m3_umum as U`	Panggil perkakas. `numpy` diberi nama pendek `np`. `matplotlib.pyplot` diberi nama pendek `plt`.
6-8	`import matplotlib` lalu `matplotlib.use("Agg")`	Pilih mode **Agg**: matplotlib menggambar di memori dan menyimpan ke berkas, tanpa membuka jendela. Perintah ini harus ditulis **sebelum** `import matplotlib.pyplot`.
9	`from qgis.core import QgsRasterLayer, QgsPointXY`	`QgsPointXY` adalah satu titik dengan koordinat x dan y.
12	`lyr = QgsRasterLayer(...)`	Buka `NDVI_deret.tif` sebagai layer raster.
13	`dp = lyr.dataProvider()`	Ambil **penyedia data**: bagian layer yang bisa membaca nilai piksel.
14	`tanggal = [t for t, _, _ in U.daftar_citra()]`	Kumpulkan hanya tanggalnya, dari 24 tuple. Tanda `_` mengabaikan dua isi lain (Bab 6).
15-16	`titik = {...}`	Kamus: nama titik menjadi pasangan koordinat `(x, y)`. Koordinat memakai satuan meter pada sistem UTM zona 49 selatan (EPSG:32749).
17	`fig, ax = plt.subplots(figsize=(9, 4))`	Siapkan kanvas gambar (`fig`) dan satu bidang grafik (`ax`), berukuran 9 kali 4 inci.
18	`for nama, (x, y) in titik.items():`	Ulangi untuk tiap titik. `.items()` memberi pasangan nama dan koordinat, dan koordinat langsung dibongkar menjadi `x` dan `y`.
19	`nilai = []`	Daftar kosong untuk NDVI titik ini.
20	`for b in range(1, lyr.bandCount() + 1):`	Ulangi untuk band nomor 1 sampai 24. `range` berhenti **sebelum** angka akhir, maka ditambah 1.
21	`v, ok = dp.sample(QgsPointXY(x, y), b)`	Baca nilai piksel di titik itu pada band `b`. Jawabannya dua benda: nilai `v` dan penanda `ok` (benar bila titik ada di dalam raster).
22	`nilai.append(v if ok else np.nan)`	Bila `ok` benar, simpan `v`. Bila tidak, simpan `nan` (tanda "bukan angka").
23	`print("%-20s NDVI awal %.2f, ..." % (...))`	Cetak ringkasan: nilai awal, minimum, dan akhir. `%-20s` memberi nama selebar 20 huruf, rata kiri. `%.2f` berarti dua desimal. `np.nanmin` mencari nilai terkecil dengan mengabaikan `nan`.
24	`ax.plot(tanggal, nilai, marker="o", ms=3, label=nama)`	Gambar garis dengan bulatan kecil di tiap titik data. `label` dipakai untuk legenda.
25-28	`ax.set_ylabel(...)` dan seterusnya	Beri judul sumbu, legenda, garis bantu, dan miringkan tulisan tanggal supaya tidak bertumpuk.

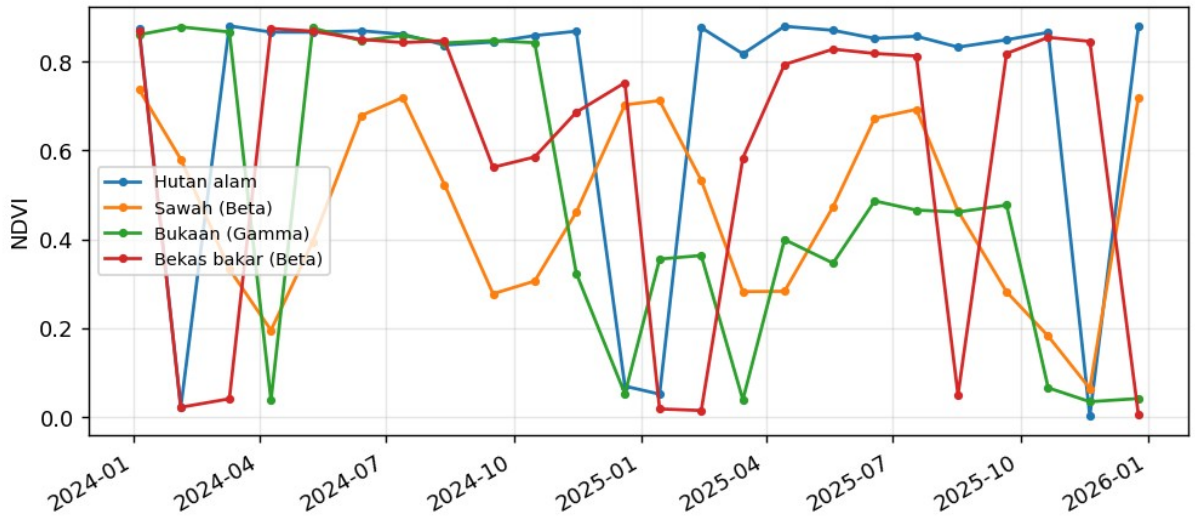
Baris	Tulisan	Artinya
29	`fig.savefig(..., dpi=130, bbox_inches="tight")`	Simpan gambar ke `profil_waktu.png`. `dpi` mengatur ketajaman. `bbox_inches="tight"` memangkas pinggiran kosong.
30	`print("Grafik tersimpan di", ...)`	Beri tahu lokasi gambar.

Hasil. Layar menampilkan (berdasarkan uji pada QGIS 4.0.2):

```
HASIL
Hutan alam           NDVI awal 0.87, minimum 0.00, akhir 0.88
Sawah (Beta)         NDVI awal 0.74, minimum 0.06, akhir 0.72
Bukaan (Gamma)       NDVI awal 0.86, minimum 0.03, akhir 0.04
Bekas bakar (Beta)   NDVI awal 0.87, minimum 0.01, akhir 0.01
Grafik tersimpan di D:/Latihan_M3/hasil\profil_waktu.png
```

Berkas `profil_waktu.png` berisi grafik berikut. Ini keluaran sungguhan skrip, bukan skema.

Ilustrasi 26.6: Profil waktu empat titik



Perhatikan garis biru, hutan alam. Ia hampir datar di sekitar 0,85, lalu **jatuh ke nol** pada beberapa tanggal. Hutannya tidak hilang. Pada tanggal itu, awan menutupi piksel. Itulah mengapa kelompok B ada.

Bagian yang boleh diubah.

- **Baris 15-16, koordinat titik.** Ganti pasangan angka dengan titik Anda sendiri, dalam sistem koordinat yang sama dengan raster. Tambah titik baru dengan menambah satu pasangan nama dan koordinat di kamus.
- Baris 17, `figsize`. Ubah ukuran gambar. `(12, 5)` membuat grafik lebih lebar.
- Baris 29, `dpi`. Naikkan menjadi `200` untuk gambar yang lebih tajam.

Jebakan.

- **Titik di luar raster tidak menghasilkan galat.** Saya mengganti koordinat hutan alam menjadi `(0, 0)`. Layar menampilkan `Hutan alam NDVI awal nan, minimum nan, akhir nan` dan peringatan `RuntimeWarning: All-NaN axis encountered`. Skrip tetap selesai. Selalu baca angka ringkasan, bukan hanya gambarnya.
- **Skrip 26.1 harus sudah dijalankan.** Saya menjalankan skrip ini di folder hasil yang kosong. Layar menampilkan `nan` untuk titik pertama, lalu berhenti dengan `ValueError: x and y must have same first dimension, but have shapes (24,) and (1,)`. Artinya tumpukan tidak ditemukan.
- **Urutan baris 6-8 sengaja begitu.** Aturan matplotlib: pilih mode lebih dulu, baru panggil `pyplot`. Jangan memindahkan baris 7 ke bawah baris 8.
- `matplotlib` harus terpasang di Python QGIS. Skrip tidak memeriksanya (Bab 12). Di QGIS 4.0.2 milik saya, `matplotlib` sudah ada.

Latihan variasi.

1. Tambahkan titik **Air** di koordinat `(312295, 9996195)`.
2. Pastikan hasilnya masuk ke ringkasan dan grafik.

Contoh jawaban, hanya bagian yang berubah (baris 15-16):

```python
titik = {"Hutan alam": (312100, 9996100), "Sawah (Beta)": (312365, 9996025),
         "Bukaan (Gamma)": (312100, 9996330), "Bekas bakar (Beta)": (312270, 9996125),
         "Air": (312295, 9996195)}
```

Layar menampilkan satu baris tambahan: `Air NDVI awal -0.31, minimum -0.52, akhir 0.04`. NDVI air negatif, tanda khas air.

> **Pola yang bisa dipakai ulang:** Untuk membaca profil waktu di titik mana pun, ganti **nama berkas tumpukan** (baris 12) dan **kamus titik** (baris 15-16). Isi perulangan di tengah tidak perlu disentuh.

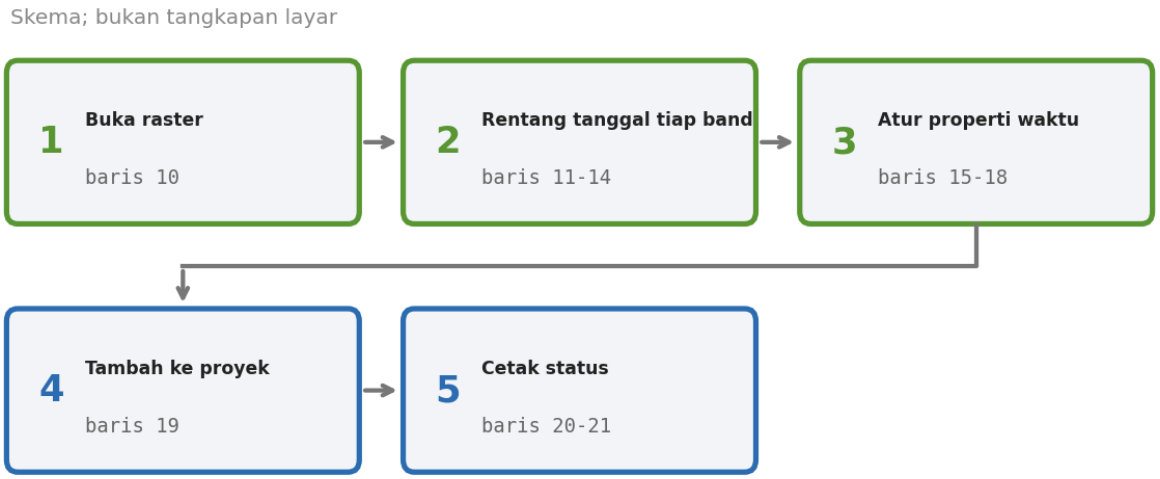
Skrip 26.3: Memberi tanggal agar bisa diputar seperti film

Berkas `1_3_atur_waktu.py`. Di dalam berkasnya tertulis "SKRIP 1.3".

Tujuan. Memberi rentang tanggal pada tiap band `NDVI_deret.tif` supaya QGIS bisa memutarnya di panel Temporal Controller.

Bahan. Hasil Skrip 26.1 dan `m3_umum.py`. Skrip ini hanya bermakna di dalam proyek QGIS yang terbuka, jadi jalankan dari Konsol.

Ilustrasi 26.7: Alur Skrip 26.3



Skrip utuh:

```python
# -*- coding: utf-8 -*-
# SKRIP 1.3: Beri tanggal pada tiap band NDVI_deret.tif agar bisa dimainkan di Temporal Controller
# Penulis: Badar Mubarok Yogaswara
# Mode yang dipakai: Fixed Time Range Per Band (tiap band punya rentang tanggal sendiri).
import os
from qgis.core import QgsRasterLayer, QgsProject, QgsDateTimeRange, Qgis
from qgis.PyQt.QtCore import QDateTime, QDate, QTime
import m3_umum as U

lyr = QgsRasterLayer(os.path.join(U.HASIL, "NDVI_deret.tif"), "NDVI_deret")
rentang = {}
for b, (tgl, _, _) in enumerate(U.daftar_citra(), start=1):
    awal = QDateTime(QDate(tgl.year, tgl.month, tgl.day), QTime(0, 0, 0))
    rentang[b] = QgsDateTimeRange(awal, awal.addDays(1))          # satu hari untuk tiap band
tp = lyr.temporalProperties()
tp.setMode(Qgis.RasterTemporalMode.FixedRangePerBand)
tp.setFixedRangePerBand(rentang)
tp.setIsActive(True)
QgsProject.instance().addMapLayer(lyr)
print("Aktif:", tp.isActive(), "| mode:", tp.mode(), "| band bertanggal:", len(tp.fixedRangePerBand()))
print("Band 9 =", tp.fixedRangePerBand()[9].begin().toString("yyyy-MM-dd"))
```

Penjelasan baris demi baris. Nomor baris mengikuti berkas di atas.

Baris	Tulisan	Artinya
4	`# Mode yang dipakai: ...`	Komentar: tiap band mendapat rentang tanggalnya sendiri.
6	`from qgis.core import ..., QgsDateTimeRange, Qgis`	`QgsDateTimeRange` adalah sepasang tanggal-jam: awal dan akhir. `Qgis` menyimpan daftar pilihan resmi QGIS.
7	`from qgis.PyQt.QtCore import QDateTime, QDate, QTime`	Tiga jenis objek tanggal milik Qt (pustaka di balik antarmuka QGIS): tanggal-jam, tanggal, dan jam.

Baris	Tulisan	Artinya
10	`lyr = QgsRasterLayer(...)`	Buka tumpukan sebagai layer.
11	`rentang = {}`	Kamus kosong. Kuncinya nomor band, isinya rentang tanggal.
12	`for b, (tgl, _, _) in enumerate(U.daftar_citra(), start=1):`	Ulangi untuk tiap tanggal. `start=1` membuat nomor urut `b` dimulai dari 1, sesuai nomor band.
13	`awal = QDateTime(QDate(tgl.year, ...), QTime(0, 0, 0))`	Bentuk objek tanggal-jam: tanggal citra pada pukul 00.00.
14	`rentang[b] = QgsDateTimeRange(awal, awal.addDays(1))`	Simpan rentang satu hari: dari `awal` sampai `awal` ditambah satu hari.
15	`tp = lyr.temporalProperties()`	Ambil pengaturan waktu milik layer.
16	`tp.setMode(Qgis.RasterTemporalMode.FixedRangePerBand)`	Pilih mode "rentang tetap per band". Nama modenya diambil dari daftar resmi, bukan angka.
17	`tp.setFixedRangePerBand(rentang)`	Pasang kamus rentang ke layer.
18	`tp.setIsActive(True)`	Aktifkan fitur waktu pada layer.
19	`QgsProject.instance().addMapLayer(lyr)`	Tambahkan layer ke proyek, kali ini **ditampilkan**.
20	`print("Aktif:", tp.isActive(), ...)`	Cetak status: aktif atau tidak, mode, dan jumlah band yang punya tanggal.
21	`print("Band 9 =", ...[9].begin().toString("yyyy-MM-dd"))`	Cetak tanggal awal rentang band ke-9. Pola `yyyy-MM-dd` berarti tahun-bulan-hari.

Hasil. Layar menampilkan (berdasarkan uji pada QGIS 4.0.2):

```
HASIL
Aktif: True | mode: 3 | band bertanggal: 24
Band 9 = 2024-09-16
```

Band 9 bertanggal 2024-09-16, sesuai urutan di `Daftar_Citra.csv`. Angka `mode: 3` adalah nomor mode `FixedRangePerBand`. Saya memeriksanya dengan `int(Qgis.RasterTemporalMode.FixedRangePerBand)`, yang bernilai 3. Memutar layer di panel Temporal Controller saya tandai [CEK]: pengujian saya berjalan tanpa jendela QGIS.

Bagian yang boleh diubah.

- Baris 14, `addDays(1)`. Panjang rentang tiap band. Saya mengganti menjadi `addDays(10)`, dan tanggal akhir band 9 berubah dari `2024-09-17` menjadi `2024-09-26`. Rentang yang lebih panjang membuat band bertahan lebih lama saat film diputar.
- **Baris 10, nama layer.** Ubah bila Anda memakai tumpukan lain.

Jebakan.

- **Pengaturan hidup di proyek, bukan di berkas.** Tanggal ini tidak tertulis ke `NDVI_deret.tif`. Simpan proyek QGIS (Bab 14) bila ingin menyimpannya.
- **Menjalankan dua kali menambah layer kembar.** Saya mengujinya: setelah dua kali jalan, proyek berisi dua layer bernama `NDVI_deret`. Hapus yang lama sebelum mengulang.
- Nama `Qgis.RasterTemporalMode` mengikuti gaya QGIS baru. Saya mengujinya hanya di QGIS 4.0.2. Di versi lain, nama ini mungkin berbeda [CEK].

Latihan variasi.

1. Cetak juga tanggal **akhir** band ke-9.
2. Buat tiap band berlaku **10 hari**, lalu cetak tanggal akhir band ke-9 lagi.

Contoh jawaban, tambahkan di akhir skrip:

```python
print("Band 9 berakhir:", tp.fixedRangePerBand()[9].end().toString("yyyy-MM-dd"))
```

Dengan `addDays(1)`, layar menampilkan `Band 9 berakhir: 2024-09-17`. Setelah baris 14 diganti menjadi `addDays(10)`, layar menampilkan `Band 9 berakhir: 2024-09-26`.

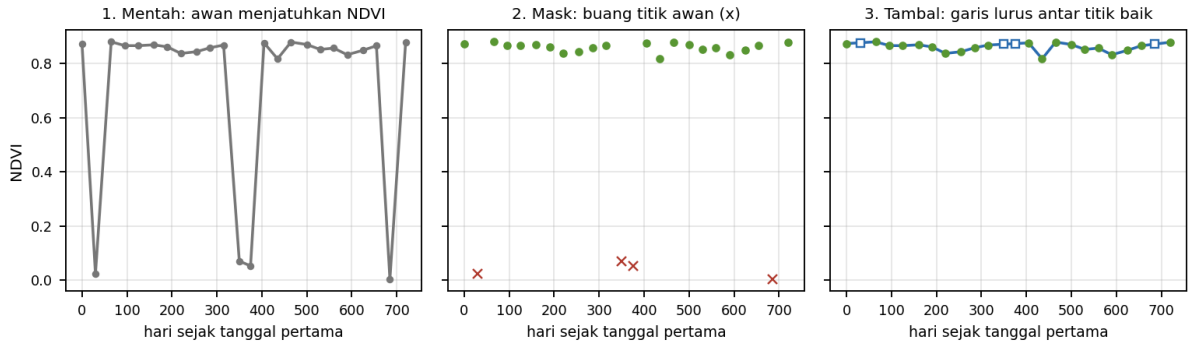
> **Pola yang bisa dipakai ulang:** Untuk memberi tanggal pada tumpukan apa pun: **satu kamus nomor band ke rentang tanggal**, lalu tiga pengaturan (`setMode`, `setFixedRangePerBand`, `setIsActive`). Ganti sumber tanggal di baris 12 dan lamanya rentang di baris 14.

Kelompok B: Membersihkan awan

Konsep: tiga cara menyingkirkan awan

Awan itu seperti **kaca buram di depan kamera**: apa pun yang tampak di baliknya bukan permukaan bumi. Anda punya tiga cara. Pertama, **membuang** piksel yang buram, memakai peta penanda. Kedua, **menggabungkan** beberapa tanggal menjadi satu citra bersih. Ketiga, **menambal** lubangnya dengan nilai dari tanggal sebelum dan sesudah. Ketiganya bisa dipakai bersama.

Ilustrasi 26.8: Awan, mask, dan tambal



Skema; bukan tangkapan layar; angka asli satu piksel hutan alam paket-m3

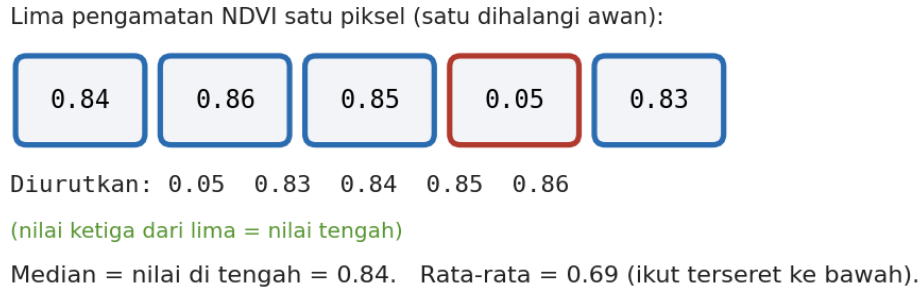
Lima istilah yang perlu Anda kenal sekarang:

- **Mask**: peta penanda piksel mana yang dipakai dan mana yang dibuang.
- **NoData**: nilai khusus yang berarti "piksel ini kosong". Piksel kosong tidak ikut dihitung.
- **Median**: nilai tengah bila semua nilai diurutkan. Lawan kata rata-rata.
- **Komposit median**: citra baru yang tiap pikselnya berisi median dari beberapa tanggal.
- **Interpolasi linear**: menarik garis lurus antara dua nilai yang diketahui, untuk menebak nilai di antaranya.

Median lebih tahan banting daripada rata-rata. Satu nilai ekstrem, misalnya awan yang lolos dari penyaringan, tidak ikut menarik hasilnya.

Ilustrasi 26.9: Median melawan rata-rata

Skema; bukan tangkapan layar



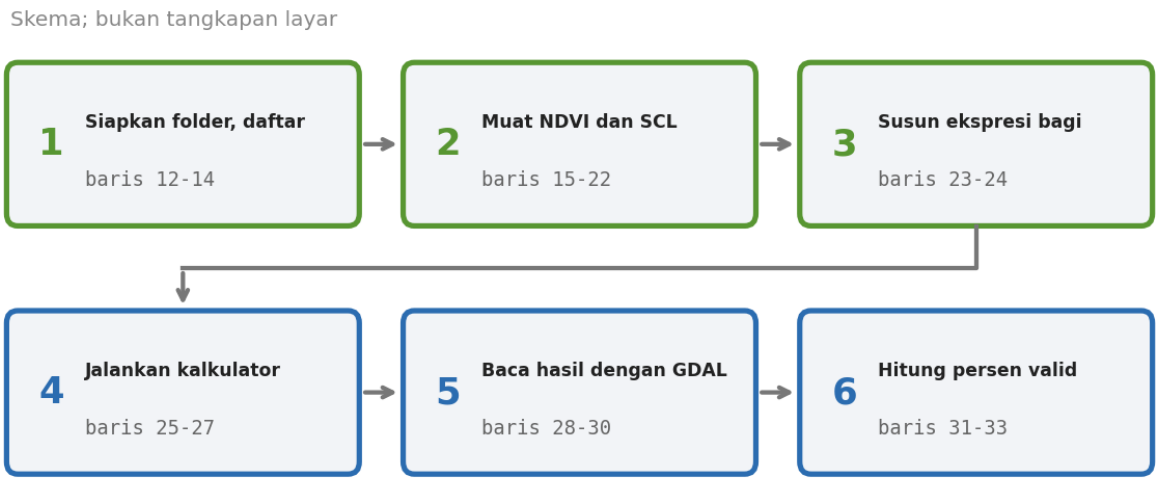
Skrip 26.4: Membuang awan dengan band SCL

Berkas `2_1_masker_awan.py`. Di dalam berkasnya tertulis "SKRIP 2.1".

Tujuan. Membuang piksel berawan, berbayang awan, atau berkabut tipis dari tiap NDVI memakai band SCL. Piksel buruk menjadi NoData.

Bahan. Hasil Skrip 26.1 (folder `ndvi`), berkas `_SCL.tif` dari paket data, dan `m3_umum.py`. Jalankan dari Konsol atau terminal.

Ilustrasi 26.10: Alur Skrip 26.4



Skrip utuh:

```python
# -*- coding: utf-8 -*-
# SKRIP 2.1: Buang awan dan bayangan memakai band SCL, per tanggal, dengan Raster Calculator
# Penulis: Badar Mubarok Yogaswara
# Syarat: Skrip 1.1 sudah dijalankan (hasil/ndvi/NDVI_*.tif ada). Piksel buruk menjadi NoData.
import os
import processing
from qgis.core import QgsRasterLayer, QgsProject
from osgeo import gdal
import numpy as np
import m3_umum as U

os.makedirs(os.path.join(U.HASIL, "ndvi_bersih"), exist_ok=True)
daftar = U.daftar_citra()
keluaran = []
for tgl, pr, ps in daftar:
    k = tgl.strftime("%Y%m%d")
    n_ndvi, n_scl = "NDVI_" + k, "SCL_" + k
    l_ndvi = QgsRasterLayer(os.path.join(U.HASIL, "ndvi", "NDVI_%s.tif" % k), n_ndvi)
    l_scl = QgsRasterLayer(ps, n_scl)
    if not l_ndvi.isValid() or not l_scl.isValid():
        raise SystemExit("Layer tidak terbaca (jalankan Skrip 1.1 dulu): " + n_ndvi)
    QgsProject.instance().addMapLayers([l_ndvi, l_scl], False)
    # piksel baik: SCL 4 (vegetasi), 5 (bukan vegetasi), 6 (air). Nilai baik dibagi 1; nilai buruk dibagi 0 ->
NoData
    ekspresi = '"%s@1" / (("%s@1" = 4) OR ("%s@1" = 5) OR ("%s@1" = 6))' % (n_ndvi, n_scl, n_scl, n_scl)
    out = os.path.join(U.HASIL, "ndvi_bersih", "NDVI_bersih_%s.tif" % k)
    processing.run("native:rastercalc", {"LAYERS": [l_ndvi, l_scl], "EXPRESSION": ekspresi, "OUTPUT": out})
    keluaran.append(out)
    a = gdal.Open(out)
    arr = a.GetRasterBand(1).ReadAsArray()
    nd = a.GetRasterBand(1).GetNoDataValue()
    valid = np.isfinite(arr) if nd is None else (arr != nd) & np.isfinite(arr)
    print(tgl, "piksel valid: %5.1f%%" % (100 * valid.mean()))
print("selesai,", len(keluaran), "berkas")
```

Penjelasan baris demi baris. Hanya hal baru yang diuraikan lebih panjang.

Baris	Tulisan	Artinya
5-10	`import os` sampai `import m3_umum as U`	Perkakas yang sama dengan Skrip 26.1, ditambah `numpy` (nama pendek `np`).
12	`os.makedirs(...ndvi_bersih...)`	Buat folder keluaran `ndvi_bersih`.
13-14	`daftar = ...` dan `keluaran = []`	Ambil 24 tuple tanggal, dan siapkan daftar penampung.
15	`for tgl, pr, ps in daftar:`	Ulangi per tanggal. Kali ini `ps` (jalur SCL) dipakai, dan `pr` tidak.
16	`k = tgl.strftime("%Y%m%d")`	Kode tanggal, misalnya `20240105`.
17	`n_ndvi, n_scl = "NDVI_" + k, "SCL_" + k`	**Penugasan ganda**: dua kotak diisi sekaligus dalam satu baris. Isinya nama layer NDVI dan nama layer SCL.
18-19	`l_ndvi = ...` dan `l_scl = ...`	Buka NDVI hasil Skrip 26.1 dan berkas SCL sebagai dua layer.
20-21	`if not l_ndvi.isValid() or not l_scl.isValid():` lalu `raise SystemExit(...)`	**Pengaman.** `isValid()` menjawab apakah layer berhasil dibuka (Bab 10). Kata `or` berarti satu layer gagal sudah cukup. Bila gagal, `SystemExit` menghentikan skrip saat itu juga dan mencetak pesan beserta nama layer yang bermasalah. Tanpa baris ini, skrip jalan terus dan menulis berkas sampah.
22	`QgsProject.instance().addMapLayers([l_ndvi, l_scl], False)`	Daftarkan **dua layer sekaligus** (dalam bentuk daftar) ke proyek, tanpa ditampilkan.
24	`ekspresi = '"%s@1" / (("%s@1" = 4) OR ... )' % (...)`	Rumus inti. Bagian dalam kurung bernilai 1 bila kode SCL adalah 4, 5, atau 6, dan 0 bila bukan. NDVI dibagi 1 tetap NDVI. NDVI dibagi 0 tidak terdefinisi, dan kalkulator raster mengisinya sebagai NoData.
25	`out = os.path.join(...)`	Jalur keluaran per tanggal.
26	`processing.run("native:rastercalc", {...})`	Jalankan kalkulator raster dengan dua layer sebagai masukan.
27	`keluaran.append(out)`	Catat hasilnya.
28	`a = gdal.Open(out)`	Buka hasil dengan GDAL untuk diperiksa.
29	`arr = a.GetRasterBand(1).ReadAsArray()`	Baca isi band 1 sebagai larik NumPy.
30	`nd = a.GetRasterBand(1).GetNoDataValue()`	Ambil nilai NoData yang dipakai berkas itu.
31	`valid = np.isfinite(arr) if nd is None else (arr != nd) & np.isfinite(arr)`	Susun larik benar atau salah: benar bila piksel punya angka. Bila berkas tidak punya NoData, cukup `isfinite`. Bila punya, piksel juga harus tidak sama dengan `nd`. Tanda `&` berarti "dan", berlaku pada seluruh larik sekaligus.

Baris	Tulisan	Artinya
32	`print(tgl, "piksel valid: %5.1f%%" % (100 * valid.mean()))`	`valid.mean()` adalah pecahan piksel yang benar. Dikali 100 menjadi persen. `%5.1f` berarti lebar lima karakter, satu desimal. `%%` mencetak tanda persen.
33	`print("selesai,", len(keluaran), "berkas")`	Cetak ringkasan akhir.

Hasil. Layar menampilkan (berdasarkan uji pada QGIS 4.0.2):

```
HASIL
2024-01-05 piksel valid:  93.5%
2024-02-04 piksel valid:  61.5%
2024-03-10 piksel valid:  41.6%
2024-04-09 piksel valid:  70.3%
2024-05-09 piksel valid:  81.3%
2024-06-13 piksel valid: 100.0%
2024-07-13 piksel valid: 100.0%
2024-08-12 piksel valid:  93.2%
2024-09-16 piksel valid:  66.9%
2024-10-16 piksel valid:  90.5%
2024-11-15 piksel valid:  75.8%
2024-12-20 piksel valid:  52.1%
2025-01-14 piksel valid:  46.1%
2025-02-13 piksel valid:  67.4%
2025-03-15 piksel valid:  51.0%
2025-04-14 piksel valid:  64.3%
2025-05-19 piksel valid:  88.6%
2025-06-18 piksel valid: 100.0%
2025-07-18 piksel valid:  93.2%
2025-08-17 piksel valid:  83.3%
2025-09-21 piksel valid: 100.0%
2025-10-21 piksel valid:  75.6%
2025-11-20 piksel valid:  57.9%
2025-12-25 piksel valid:  39.5%
selesai, 24 berkas
```

Bacalah angka-angkanya. Pada musim kemarau (Juni dan Juli 2024), 100 persen piksel valid. Pada Desember 2025, hanya 39,5 persen. Tiap tanggal punya porsi awan sendiri, dan itulah yang akan Anda atasi di dua skrip berikutnya.

Bagian yang boleh diubah.

- **Baris 24, kode SCL.** Hapus bagian `OR ("%s@1" = 6)` untuk membuang piksel air, dan kurangi satu `n_scl` di tuple isian. Saya mencobanya: pada 5 Januari 2024, persen valid turun dari `93.5` menjadi `89.4`.
- **Baris 12 dan 25, nama folder dan berkas.** Bebas Anda ganti.
- **Baris 21, teks pesan.** Bebas Anda ubah, misalnya dengan menyebut folder yang Anda pakai. Pesan itu hanya tampil bila ada layer yang gagal dibuka.

Jebakan.

- **Pengaman di baris 20-21 menjaga urutan kerja.** Saya menjalankan skrip di folder hasil yang kosong, sebelum Skrip 26.1. Skrip langsung berhenti dengan pesan `Layer tidak terbaca (jalankan Skrip 1.1 dulu): NDVI_20240105` dan tidak menulis satu berkas pun (hanya folder kosong `ndvi_bersih` yang sudah sempat dibuat). Di berkasnya, Skrip 26.1 tertulis "Skrip

1.1". Tanpa dua baris itu, skrip tetap selesai tanpa galat dan menulis 24 berkas sampah. Pengaman ini hanya memeriksa bahwa layer bisa dibuka, bukan bahwa isinya benar. **Tetap cocokkan angka keluaran dengan angka yang seharusnya.**

- **Trik "bagi nol menjadi NoData" bergantung pada kalkulator raster QGIS.** Saya memeriksanya pada 10 Maret 2024: ada 935 piksel dengan kode SCL buruk, dan tepat 935 piksel NoData di hasil, di posisi yang sama. Pada QGIS 4.0.2 trik ini bekerja. Pada versi lain, periksa dulu [CEK].
- Nilai NoData bukan `-9999`. Berkas hasil memakai bilangan terkecil Float32 (sekitar `-3.4e+38`). Karena itu skrip mengambil `nd` dari berkas, tidak menebaknya.
- **SCL adalah tebakan otomatis.** Tepi awan yang sangat tipis bisa lolos sebagai "bersih". Itu sebabnya ada komposit median di skrip berikutnya.

Latihan variasi.

1. Hanya terima piksel vegetasi dan bukan vegetasi (kode 4 dan 5). Air dibuang.
2. Lihat persen valid pada tiga tanggal pertama.

Contoh jawaban, hanya baris yang berubah (baris 24):

```python
    ekspresi = '"%s@1" / (("%s@1" = 4) OR ("%s@1" = 5))' % (n_ndvi, n_scl, n_scl)
```

Layar menampilkan tiga tanggal pertama: `2024-01-05 piksel valid:  89.4%`, `2024-02-04 piksel valid:  59.7%`, `2024-03-10 piksel valid:  41.2%`. Semuanya lebih rendah daripada hasil asli (93,5; 61,5; 41,6), karena piksel air ikut dibuang.

> **Pola yang bisa dipakai ulang:** Untuk membuang piksel apa pun memakai peta penanda: **nilai dibagi kondisi**. Ganti nama dua layer (baris 18-19) dan **kode yang dianggap baik** (baris 24). Simpan pengaman baris 20-21: skrip berhenti bila layer gagal dibuka. Untuk membuang berdasarkan peta lain, misalnya peta hutan tanaman, cukup ganti sumber penanda dan kodenya.

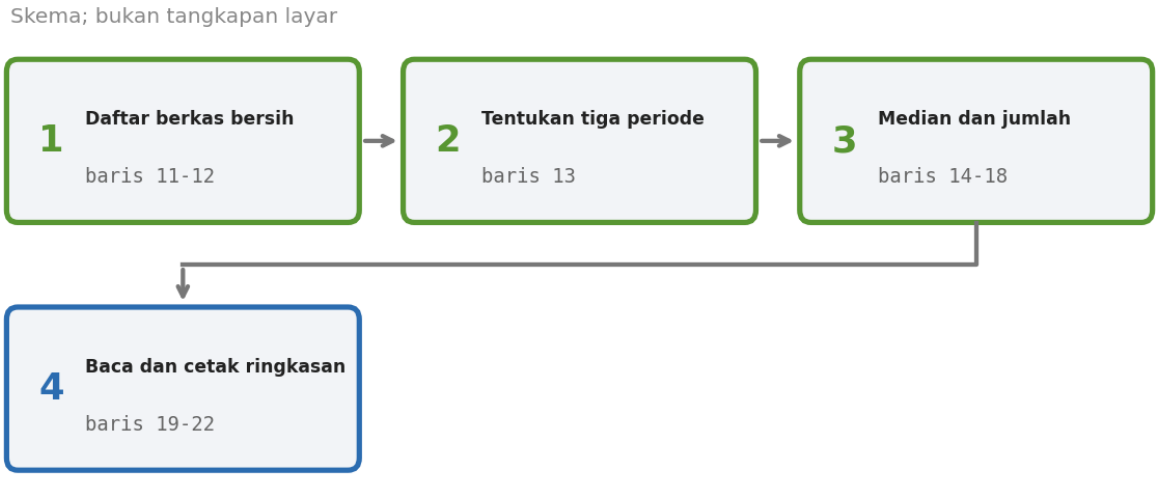
Skrip 26.5: Komposit median dari beberapa tanggal

Berkas `2_2_komposit_median.py`. Di dalam berkasnya tertulis "SKRIP 2.2".

Tujuan. Membuat citra komposit median NDVI dan peta jumlah pengamatan valid untuk tiga periode: sebelum, kemarau 2024, dan sesudah.

Bahan. Hasil Skrip 26.4 (folder `ndvi_bersih`) dan `m3_umum.py`. Jalankan dari Konsol atau terminal.

Ilustrasi 26.11: Alur Skrip 26.5



Skrip utuh:

```python
# -*- coding: utf-8 -*-
# SKRIP 2.2: Komposit median dan jumlah pengamatan valid dengan Cell Statistics
# Penulis: Badar Mubarok Yogaswara
# Syarat: Skrip 2.1 sudah dijalankan.
import os
import numpy as np
import processing
from osgeo import gdal
import m3_umum as U

daftar = U.daftar_citra()
berkas = [os.path.join(U.HASIL, "ndvi_bersih", "NDVI_bersih_%s.tif" % t.strftime("%Y%m%d")) for t, _, _ in
daftar]
periode = {"sebelum": (0, 8), "kemarau2024": (5, 9), "sesudah": (20, 24)}   # indeks [awal, akhir) di daftar
tanggal
for nama, (a, b) in periode.items():
    for kode, stat in (("median", 3), ("jumlah", 1)):
        out = os.path.join(U.HASIL, "komposit_ndvi_%s_%s.tif" % (nama, kode))
        processing.run("native:cellstatistics", {"INPUT": berkas[a:b], "STATISTIC": stat, "IGNORE_NODATA":
True,
                                                 "REFERENCE_LAYER": berkas[a], "OUTPUT_NODATA_VALUE": -9999,
"OUTPUT": out})
    med = gdal.Open(os.path.join(U.HASIL, "komposit_ndvi_%s_median.tif" % nama)).ReadAsArray()
    jml = gdal.Open(os.path.join(U.HASIL, "komposit_ndvi_%s_jumlah.tif" % nama)).ReadAsArray()
    print("%-12s tanggal %s sampai %s | pengamatan valid per piksel: min %d, rata-rata %.1f | piksel tanpa
data: %d"
          % (nama, daftar[a][0], daftar[b - 1][0], jml.min(), jml.mean(), int((med == -9999).sum())))
```

Penjelasan baris demi baris.

Baris	Tulisan	Artinya
5-9	`import os` sampai `import m3_umum as U`	Perkakas, seperti sebelumnya.

Baris	Tulisan	Artinya
11	`daftar = U.daftar_citra()`	Ambil daftar tanggal.
12	`berkas = [os.path.join(...) for t, _, _ in daftar]`	**List comprehension**: satu baris yang membangun daftar 24 jalur berkas NDVI bersih, satu untuk tiap tanggal (Bab 6).
13	`periode = {"sebelum": (0, 8), ...}`	Kamus tiga periode. Isinya pasangan nomor indeks: awal, lalu akhir. Akhir **tidak termasuk**. `(0, 8)` berarti tanggal ke-0 sampai ke-7, yaitu delapan tanggal pertama.
14	`for nama, (a, b) in periode.items():`	Ulangi untuk tiap periode dan bongkar pasangan indeks menjadi `a` dan `b`.
15	`for kode, stat in (("median", 3), ("jumlah", 1)):`	Putaran kedua, di dalam putaran pertama: dua pasangan nama dan nomor statistik. Nomor 3 berarti median. Nomor 1 berarti **jumlah** pengamatan valid per piksel (`Count`).
16	`out = os.path.join(...)`	Nama keluaran, misalnya `komposit_ndvi_sebelum_median.tif`.
17-18	`processing.run("native:cellstatistics", {...})`	Alat Cell statistics menghitung satu nilai per piksel dari tumpukan layer. `INPUT` adalah irisan daftar `berkas[a:b]`. `STATISTIC` memilih jenis hitungan. `IGNORE_NODATA: True` melewati piksel kosong. `REFERENCE_LAYER` memberi ukuran dan lokasi raster hasil. `OUTPUT_NODATA_VALUE` memilih -9999 sebagai tanda kosong.
19-20	`med = gdal.Open(...).ReadAsArray()`	Baca dua hasil tadi sebagai larik.
21-22	`print("%-12s tanggal %s sampai %s ..." % (...))`	Cetak ringkasan: tanggal awal dan akhir periode, jumlah pengamatan valid terkecil (`jml.min()`) dan rata-ratanya, serta banyak piksel tanpa data. `daftar[b - 1][0]` mengambil tanggal terakhir yang benar-benar dipakai. Tanda `%d` mencetak bilangan bulat.

Irisan `berkas[a:b]` adalah cara Python mengambil bagian daftar: dari nomor `a` sampai sebelum nomor `b` (Bab 5).

Hasil. Layar menampilkan (berdasarkan uji pada QGIS 4.0.2):

```
HASIL
sebelum      tanggal 2024-01-05 sampai 2024-08-12 | pengamatan valid per piksel: min 3, rata-rata 6.4 | piksel tanpa data: 0
kemarau2024  tanggal 2024-06-13 sampai 2024-09-16 | pengamatan valid per piksel: min 2, rata-rata 3.6 | piksel tanpa data: 0
sesudah      tanggal 2025-09-21 sampai 2025-12-25 | pengamatan valid per piksel: min 1, rata-rata 2.7 | piksel tanpa data: 0
```

Periode "sebelum" punya rata-rata 6,4 pengamatan valid per piksel. Periode "sesudah" hanya 2,7, dan ada piksel yang hanya punya satu pengamatan. Median dari satu angka sama dengan angka itu sendiri, jadi komposit "sesudah" di piksel tersebut paling rapuh. Tidak ada piksel tanpa data (`0`).

Bagian yang boleh diubah.

- **Baris 13, indeks periode.** Tambah atau ubah periode. Saya menambahkan `"kemarau2025": (17, 21)`. Layar menampilkan satu baris baru: `kemarau2025  tanggal 2025-06-18 sampai 2025-09-21 | pengamatan valid per piksel: min 2, rata-rata 3.8 | piksel tanpa data: 0`.
- **Baris 15, nomor statistik.** Angka adalah **kode**, bukan nama. Menurut bantuan alat: 0 penjumlahan (`Sum`), 1 banyaknya data (`Count`), 2 rata-rata (`Mean`), 3 median (`Median`). Saya memeriksanya lewat `qgis_process help native:cellstatistics`.
- Baris 18, `-9999`. Tanda kosong. Ubah juga pada baris 22 bila Anda mengubahnya.

Jebakan.

- **Indeks akhir tidak termasuk.** `(5, 9)` memakai tanggal ke-5, 6, 7, dan 8, bukan ke-9. Itu sebabnya tanggal akhir di baris 22 memakai `b - 1`.
- **Nomor statistik mudah tertukar.** Kode 1 bukan "satu" dalam arti terkecil, melainkan banyak pengamatan. Selalu cek hasil: peta "jumlah" harus berisi bilangan bulat kecil.
- **Periode di luar daftar menghasilkan galat.** Saya mengganti periode "sesudah" menjadi `(24, 28)`. Layar menampilkan `IndexError: list index out of range` pada baris 18, karena `berkas[24]` tidak ada.
- **Skrip 26.4 harus sudah jalan.** Saya menjalankannya di folder kosong. Layar menampilkan `Incorrect parameter value for INPUT`.

Latihan variasi.

1. Tambahkan periode `kemarau2025` untuk tanggal ke-17 sampai ke-20.
2. Pastikan hasilnya muncul sebagai baris keempat.

Contoh jawaban, hanya baris 13:

```python
periode = {"sebelum": (0, 8), "kemarau2024": (5, 9), "sesudah": (20, 24), "kemarau2025": (17, 21)}
```

Layar menampilkan empat baris, dan baris terakhir sama seperti yang saya sebut di atas.

> **Pola yang bisa dipakai ulang:** Untuk rangkuman statistik beberapa periode, ganti **kamus periode** (baris 13) dan **pasangan statistik** (baris 15). Pola "kamus periode lalu dua putaran" bisa dipakai untuk rata-rata, minimum, atau maksimum: cukup ubah nomor statistiknya.

Skrip 26.6: Menambal lubang awan, lalu menguji tambalannya

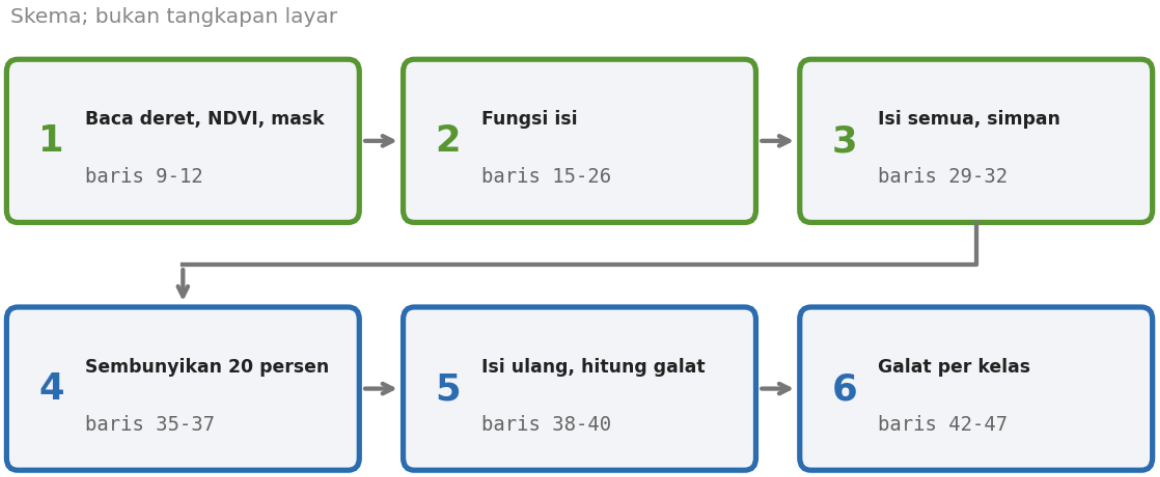
Berkas `2_3_isi_lubang.py`. Di dalam berkasnya tertulis "SKRIP 2.3".

Tujuan. Mengisi lubang akibat awan pada deret NDVI dengan interpolasi linear sepanjang waktu, lalu menguji ketepatannya dengan menyembunyikan 20 persen pengamatan bersih.

Bahan. `m3_umum.py` dan paket data. Skrip ini langsung membaca citra mentah, jadi tidak memerlukan skrip lain. Jalankan dari Konsol atau terminal.

Skrip ini memuat satu fungsi buatan sendiri dan satu uji. Gambar berikut menunjukkan urutannya.

Ilustrasi 26.12: Alur Skrip 26.6



Skrip utuh:

```python
# -*- coding: utf-8 -*-
# SKRIP 2.3: Isi lubang akibat awan dengan interpolasi linear sepanjang waktu, lalu uji kebenarannya
# Penulis: Badar Mubarok Yogaswara
# Cara uji: sembunyikan 20% pengamatan bersih secara acak, isi dengan interpolasi, bandingkan dengan nilai aslinya.
import os
import numpy as np
import m3_umum as U

tgl, refl, scl, gt, prj = U.baca_deret()
ndvi = U.nilai_ndvi(refl)
baik = U.bersih(scl)
hari = np.array([(t - tgl[0]).days for t in tgl], dtype="float64")     # sumbu waktu dalam hari (selang tidak sama)


def isi(ndvi, baik):
    T, H, W = ndvi.shape
    hasil = np.empty_like(ndvi)
    for i in range(H):
        for j in range(W):
            ok = baik[:, i, j]
            if ok.sum() == 0:
                hasil[:, i, j] = np.nan
            else:
                # np.interp: di luar rentang, nilai ujung dipakai (tidak ekstrapolasi)
                hasil[:, i, j] = np.interp(hari, hari[ok], ndvi[ok, i, j])
    return hasil


lengkap = isi(ndvi, baik)
U.tulis_tif(os.path.join(U.HASIL, "NDVI_isi.tif"), lengkap.astype("float32"), gt, prj,
            deskripsi=[t.isoformat() for t in tgl])
print("Piksel-tanggal awan sebelum diisi: %.1f%%, sesudah: %.1f%%" % (100 * (~baik).mean(), 100 * np.isnan(lengkap).mean()))

# uji sembunyi-isi
rng = np.random.default_rng(1)
```

```
sembunyi = baik & (rng.random(baik.shape) < 0.20)
sembunyi[0], sembunyi[-1] = False, False                  # tanggal ujung tidak diuji (tanpa tetangga di satu
sisi)
uji = isi(ndvi, baik & ~sembunyi)
galat = (uji - ndvi)[sembunyi]
print("Uji sembunyi-isi: %d pengamatan; RMSE %.3f; galat mutlak median %.3f" % (sembunyi.sum(),
np.sqrt(np.nanmean(galat ** 2)), np.nanmedian(np.abs(galat))))
# galat per kelas acuan
dinamika, _, _ = U.baca_tif(os.path.join(U.PAKET, "acuan", "Acuan_Dinamika.tif"))
nama = {1: "Hutan alam", 2: "Hutan tanaman", 3: "Pertanian", 4: "Air", 5: "Terbuka", 6: "Deforestasi", 7:
"Terbakar", 8: "Panen HTI"}
for k in range(1, 9):
    m = sembunyi & (dinamika == k)[None]
    e = (uji - ndvi)[m]
    print("  %-14s n=%4d  RMSE %.3f" % (nama[k], m.sum(), np.sqrt(np.nanmean(e ** 2))))
```

Penjelasan baris demi baris.

Baris	Tulisan	Artinya
5-7	`import os`, `import numpy as np`, `import m3_umum as U`	Perkakas. Skrip ini tidak memakai QGIS sama sekali, hanya NumPy dan GDAL lewat kotak perkakas.
9	`tgl, refl, scl, gt, prj = U.baca_deret()`	Baca seluruh deret sekaligus. Lima hasil dibongkar ke lima kotak.
10	`ndvi = U.nilai_ndvi(refl)`	Hitung NDVI untuk semua tanggal. Bentuknya (24, 40, 40).
11	`baik = U.bersih(scl)`	Larik benar atau salah: benar di piksel-tanggal yang bersih.
12	`hari = np.array([(t - tgl[0]).days for t in tgl], dtype="float64")`	Ubah tiap tanggal menjadi **hari ke-berapa** sejak tanggal pertama. Selisih dua tanggal memberi objek yang punya atribut `.days`. Selang antar tanggal tidak sama, jadi waktu harus dalam hari, bukan nomor urut.
15	`def isi(ndvi, baik):`	Mulai fungsi buatan sendiri dengan dua masukan (Bab 7).
16	`T, H, W = ndvi.shape`	Ambil tiga ukuran: jumlah tanggal, tinggi, lebar.
17	`hasil = np.empty_like(ndvi)`	Siapkan larik kosong seukuran `ndvi` untuk menampung hasil.
18-19	`for i in range(H):` dan `for j in range(W):`	Dua putaran bersarang: telusuri tiap baris `i` dan tiap kolom `j`, sehingga tiap piksel dikunjungi sekali.
20	`ok = baik[:, i, j]`	Ambil 24 nilai benar atau salah milik piksel ini. Tanda `:` berarti "semua tanggal".
21-22	`if ok.sum() == 0:` lalu `hasil[:, i, j] = np.nan`	Bila tidak ada satu pun pengamatan bersih, isi seluruh deret piksel dengan `nan`.
23-25	`else:` lalu `np.interp(...)`	Bila ada, tarik garis lurus: `np.interp(hari, hari[ok], ndvi[ok, i, j])` mencari nilai di semua hari berdasarkan hanya hari dan nilai yang bersih.
26	`return hasil`	Kembalikan hasilnya.
29	`lengkap = isi(ndvi, baik)`	Panggil fungsi untuk seluruh data.
30-31	`U.tulis_tif(...)`	Simpan hasil ke `NDVI_isi.tif` sebagai Float32, dengan nama band berupa tanggal.

Baris	Tulisan	Artinya
32	`print("Piksel-tanggal awan sebelum diisi: ...")`	Bandingkan persen awan sebelum dan sesudah. Tanda `~` membalik benar-salah.
35	`rng = np.random.default_rng(1)`	Buat pengundi acak dengan **benih** 1. Benih yang sama selalu menghasilkan undian yang sama.
36	`sembunyi = baik & (rng.random(baik.shape) < 0.20)`	Pilih sekitar 20 persen dari piksel-tanggal bersih secara acak. Hanya yang bersih yang dipilih (`baik &`).
37	`sembunyi[0], sembunyi[-1] = False, False`	Tanggal pertama dan terakhir tidak diuji, karena tidak punya tetangga di satu sisi. Indeks `-1` berarti yang terakhir.
38	`uji = isi(ndvi, baik & ~sembunyi)`	Isi ulang, kali ini seolah-olah titik yang disembunyikan awan.
39	`galat = (uji - ndvi)[sembunyi]`	Selisih tebakan dan nilai asli, hanya di titik yang disembunyikan.
40	`print("Uji sembunyi-isi: ...")`	Cetak jumlah pengamatan, RMSE, dan median galat mutlak.
42-47	`dinamika, _, _ = ...` sampai `print(...)`	Ulangi perhitungan galat per kelas tutupan menurut peta acuan. `(dinamika == k)[None]` menambah satu sumbu agar bentuknya (1, 40, 40) cocok dengan (24, 40, 40).

RMSE adalah ukuran besar galat: selisih dikuadratkan, dirata-ratakan, lalu diakarkan. Selisih besar dihukum lebih berat daripada selisih kecil. Makin kecil, makin tepat.

Hasil. Layar menampilkan (berdasarkan uji pada QGIS 4.0.2):

```
HASIL
Piksel-tanggal awan sebelum diisi: 25.3%, sesudah: 0.0%
Uji sembunyi-isi: 5336 pengamatan; RMSE 0.078; galat mutlak median 0.014
  Hutan alam     n=2572  RMSE 0.038
  Hutan tanaman  n= 304  RMSE 0.039
  Pertanian      n= 625  RMSE 0.158
  Air            n= 277  RMSE 0.093
  Terbuka        n= 136  RMSE 0.020
  Deforestasi    n= 438  RMSE 0.117
  Terbakar       n= 520  RMSE 0.055
  Panen HTI      n= 464  RMSE 0.075
```

Awan sebelumnya 25,3 persen, sesudah ditambal 0,0 persen. Median galat hanya 0,014, jadi tambalan umumnya sangat dekat. RMSE total 0,078 lebih besar karena beberapa kelas sulit ditebak. **Pertanian** punya RMSE 0,158, empat kali hutan alam. Itu masuk akal: sawah naik turun cepat, dan garis lurus antar dua titik tidak menangkap lonjakan di antaranya.

Bagian yang boleh diubah.

- Baris 36, `0.20`. Porsi yang disembunyikan. Saya menggantinya dengan `0.40`: jumlah pengamatan menjadi 10700 dan RMSE naik menjadi 0.091. Makin banyak yang disembunyikan, makin sulit tebakannya.

- Baris 35, benih `1`. Ganti ke `7`. Jumlah pengamatan menjadi 5209 dan RMSE 0.076. Angka bergeser sedikit, tetapi gambaran besarnya sama.
- **Baris 25, fungsi interpolasi.** Fungsi lain bisa dipakai, asal mengembalikan nilai untuk semua hari.

Jebakan.

- Fungsi `isi` menduplikasi `U.isi_waktu`. Isi keduanya hampir sama: fungsi di kotak perkakas (`m3_umum.py` baris 94-103) dipakai Skrip 26.12. Saya menjalankan keduanya pada data yang sama. Hasilnya `np.allclose` benar, selisih terbesar `2.98e-08`, dan posisi `nan` sama. Selisih sekecil itu hanya karena tipe angka (satu 32 bit, satu 64 bit).
- **Putaran per piksel lambat untuk raster besar.** Data contoh hanya 1600 piksel. Raster 10.000 kali 10.000 punya 62.500 kali lebih banyak.
- Di luar rentang, `np.interp` memakai nilai ujung. Ia tidak menarik garis ke luar (tidak ekstrapolasi). Karena itu tanggal ujung dikeluarkan dari uji (baris 37).
- **Ukuran dan luas tidak dikunci di skrip ini.** `T, H, W` diambil dari data. Skrip lain di bab ini mengunci 40 kali 40 piksel dan 0,01 hektare per piksel, dan itu dicatat di tempatnya.

Latihan variasi.

1. Sembunyikan **40 persen** pengamatan bersih.
2. Bandingkan RMSE totalnya dengan hasil asli (0,078).

Contoh jawaban, hanya baris 36:

```python
sembunyi = baik & (rng.random(baik.shape) < 0.40)
```

Layar menampilkan `Uji sembunyi-isi: 10700 pengamatan; RMSE 0.091; galat mutlak median 0.015`. RMSE naik dari 0,078 menjadi 0,091.

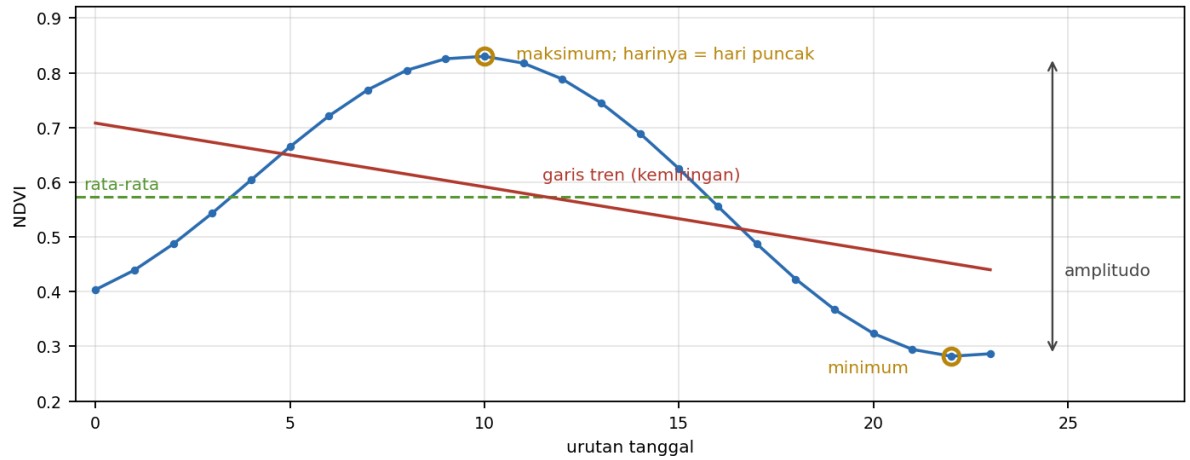
> **Pola yang bisa dipakai ulang:** Pola **sembunyi, isi, bandingkan** menguji metode apa pun. Anda menyembunyikan sebagian data yang nilainya Anda tahu, memakai metode untuk menebaknya, lalu mengukur selisihnya. Ganti **porsi** (baris 36) dan **metode tebak** (fungsi `isi`). Anda tidak perlu data acuan dari luar.

Kelompok C: Mengukur irama tiap tutupan lahan

Konsep: irama dalam tiga kalimat

Tiap tutupan lahan punya **sidik jari musiman**, seperti detak jantung tiap orang berbeda: hutan berdetak tenang, sawah berdetak cepat dan tinggi. Anda bisa meringkas detak itu dalam beberapa angka per piksel: rata-rata, seberapa tinggi naik turunnya, ke mana arah umumnya, dan kapan puncaknya. Angka ringkas ini disebut **metrik deret waktu**.

Ilustrasi 26.13: Metrik pada satu deret



Skema; bukan tangkapan layar

Lima istilah yang perlu Anda kenal sekarang:

- **Amplitudo**: selisih nilai tertinggi dan terendah dalam satu deret.
- **Simpangan baku**: ukuran seberapa jauh nilai biasanya menyimpang dari rata-ratanya.
- **Tren**: kemiringan garis lurus yang paling cocok dengan deret. Di bab ini dinyatakan sebagai perubahan NDVI **per tahun**.
- **Hari puncak**: hari ke-berapa (sejak tanggal pertama) NDVI mencapai maksimum.
- **Anomali**: selisih nilai satu tanggal dari rata-rata piksel itu sendiri.

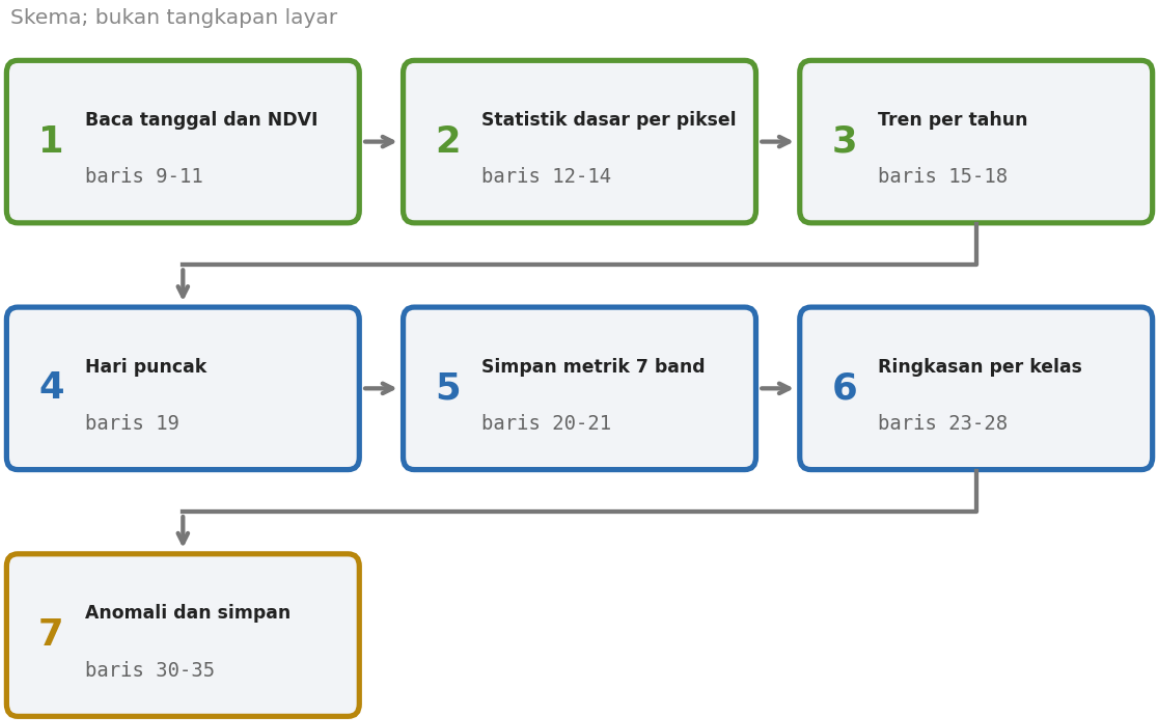
Skrip 26.7: Metrik deret waktu per piksel

Berkas `3_1_metrik_deret.py`. Di dalam berkasnya tertulis "SKRIP 3.1".

Tujuan. Menghitung tujuh metrik NDVI tiap piksel, meringkasnya per kelas tutupan lahan, dan menghitung anomali tiap tanggal.

Bahan. Hasil Skrip 26.6 (`NDVI_isi.tif`), peta acuan `acuan/Acuan_Dinamika.tif` (delapan kelas tutupan), dan `m3_umum.py`. Skrip ini hanya memakai NumPy dan GDAL, jadi bisa dijalankan dari Konsol maupun terminal.

Ilustrasi 26.14: Alur Skrip 26.7



Skrip utuh:

```python
# -*- coding: utf-8 -*-
# SKRIP 3.1: Metrik deret waktu per piksel: rata-rata, minimum, maksimum, amplitudo, simpangan baku, tren per tahun, tanggal puncak
# Penulis: Badar Mubarok Yogaswara
# Syarat: Skrip 2.3 sudah dijalankan (hasil/NDVI_isi.tif). Menghasilkan hasil/metrik_ndvi.tif (7 band).
import os
import numpy as np
import m3_umum as U

tgl = [t for t, _, _ in U.daftar_citra()]
hari = np.array([(t - tgl[0]).days for t in tgl], dtype="float64")
x, gt, prj = U.baca_tif(os.path.join(U.HASIL, "NDVI_isi.tif"))          # (24, H, W)
rata, mini, maks = x.mean(0), x.min(0), x.max(0)
amplitudo = maks - mini
sb = x.std(0)
# tren: kemiringan garis lurus terhadap waktu dalam HARI, lalu dikali 365 -> perubahan NDVI per tahun
hc = hari - hari.mean()
kemiringan = (hc[:, None, None] * (x - rata)).sum(0) / (hc ** 2).sum()
tren_tahun = kemiringan * 365.0
puncak = hari[x.argmax(0)]                                              # hari ke- (sejak tanggal pertama) saat NDVI tertinggi
nama = ["rata", "minimum", "maksimum", "amplitudo", "simpangan_baku", "tren_per_tahun", "hari_puncak"]
U.tulis_tif(os.path.join(U.HASIL, "metrik_ndvi.tif"), np.array([rata, mini, maks, amplitudo, sb, tren_tahun, puncak]).astype("float32"), gt, prj, deskripsi=nama)

din, _, _ = U.baca_tif(os.path.join(U.PAKET, "acuan", "Acuan_Dinamika.tif"))
```

```
kls = {1: "Hutan alam", 2: "Hutan tanaman", 3: "Pertanian", 4: "Air", 5: "Terbuka", 6: "Deforestasi", 7:
"Terbakar", 8: "Panen HTI"}
print("%-14s %6s %9s %9s %12s" % ("Kelas", "rata", "amplitudo", "simp.baku", "tren/tahun"))
for k, n in kls.items():
    m = din == k
    print("%-14s %6.2f %9.2f %9.2f %12.3f" % (n, rata[m].mean(), amplitudo[m].mean(), sb[m].mean(),
tren_tahun[m].mean()))

# anomali: selisih NDVI tiap tanggal terhadap rata-rata piksel itu
anomali = x - rata[None]
U.tulis_tif(os.path.join(U.HASIL, "anomali_ndvi.tif"), anomali.astype("float32"), gt, prj,
deskripsi=[t.isoformat() for t in tgl])
i_sep = tgl.index(__import__("datetime").date(2024, 9, 16))
m = din == 7
print("Anomali NDVI rata-rata di bekas terbakar pada 2024-09-16: %.3f; di hutan stabil: %.3f" % (anomali[i_sep]
[m].mean(), anomali[i_sep][din == 1].mean()))
```

Penjelasan baris demi baris.

Baris	Tulisan	Artinya
5-7	`import os, import numpy as np, import m3_umum as U`	Perkakas yang sama seperti sebelumnya.
9	`tgl = [t for t, _, _ in U.daftar_citra()]`	Daftar 24 tanggal.
10	`hari = np.array([...], dtype="float64")`	Tiap tanggal menjadi hari ke-berapa sejak tanggal pertama.
11	`x, gt, prj = U.baca_tif(...NDVI_isi.tif)`	Baca tumpukan NDVI yang sudah ditambal. Bentuk `x` adalah (24, tinggi, lebar).
12	`rata, mini, maks = x.mean(0), x.min(0), x.max(0)`	Tiga hitungan sekaligus. Angka `0` berarti "sepanjang sumbu pertama", yaitu sumbu waktu. Hasilnya satu angka per piksel.
13	`amplitudo = maks - mini`	Selisih tertinggi dan terendah.
14	`sb = x.std(0)`	Simpangan baku per piksel.
16	`hc = hari - hari.mean()`	**Pemusatan**: kurangi tiap hari dengan hari rata-rata, supaya hitungan tren lebih rapi.
17	`kemiringan = (hc[:, None, None] * (x - rata)).sum(0) / (hc ** 2).sum()`	Rumus kemiringan garis lurus terbaik. `hc[:, None, None]` mengubah daftar 24 angka menjadi bentuk (24, 1, 1) agar bisa dikalikan dengan `x` yang berbentuk (24, tinggi, lebar). Hasilnya kemiringan per hari.
18	`tren_tahun = kemiringan * 365.0`	Kemiringan per hari dikali 365 menjadi perubahan NDVI per tahun.
19	`puncak = hari[x.argmax(0)]`	`argmax(0)` memberi nomor tanggal tempat NDVI tertinggi. `hari[...]` mengubah nomor itu menjadi hari ke-berapa.
20	`nama = ["rata", ...]`	Tujuh nama band untuk berkas keluaran.
21	`U.tulis_tif(..., np.array([...]).astype("float32"), gt, prj, deskripsi=nama)`	Susun tujuh larik menjadi satu larik tujuh band, ubah ke Float32, lalu simpan ke `metrik_ndvi.tif` dengan nama band.

Baris	Tulisan	Artinya
23	`din, _, _ = U.baca_tif(...Acuan_Dinamika.tif)`	Baca peta acuan kelas tutupan (1 sampai 8). Dua hasil lain diabaikan.
24	`kls = {1: "Hutan alam", ...}`	Kamus nomor kelas menjadi nama kelas.
25	`print("%-14s %6s %9s %9s %12s" % (...))`	Cetak judul kolom tabel dengan lebar tetap.
26	`for k, n in kls.items():`	Ulangi untuk tiap kelas.
27	`m = din == k`	Larik benar atau salah: benar di piksel yang kelas acuannya `k`.
28	`print("%-14s %6.2f ..." % (n, rata[m].mean(), ...))`	`rata[m]` mengambil nilai rata-rata hanya di piksel kelas itu, lalu dirata-ratakan lagi. Hasilnya satu baris ringkasan per kelas.
31	`anomali = x - rata[None]`	Kurangi tiap tanggal dengan rata-rata piksel. `rata[None]` menambah sumbu waktu agar bentuknya cocok.
32	`U.tulis_tif(..."anomali_ndvi.tif"...)`	Simpan 24 band anomali.
33	`i_sep = tgl.index(__import__("datetime").date(2024, 9, 16))`	Cari nomor urut tanggal 16 September 2024 di daftar. `__import__("datetime")` adalah cara tidak lazim memanggil perkakas `datetime` di tempat. Cara yang lebih rapi adalah `import datetime` di bagian atas.
34	`m = din == 7`	Kali ini kelas 7, bekas terbakar.
35	`print("Anomali NDVI rata-rata di bekas terbakar ...")`	Cetak anomali tanggal itu di bekas terbakar, lalu di hutan alam stabil (`din == 1`).

Hasil. Layar menampilkan (berdasarkan uji pada QGIS 4.0.2):

```
HASIL
Kelas            rata amplitudo simp.baku   tren/tahun
Hutan alam       0.86      0.10      0.02       -0.004
Hutan tanaman    0.77      0.10      0.02       -0.006
Pertanian        0.50      0.46      0.15       -0.027
Air             -0.33      0.29      0.07        0.003
Terbuka          0.11      0.06      0.02       -0.001
Deforestasi      0.64      0.59      0.22       -0.281
Terbakar         0.82      0.23      0.06       -0.022
Panen HTI        0.61      0.51      0.17       -0.200
Anomali NDVI rata-rata di bekas terbakar pada 2024-09-16: -0.115; di hutan stabil: -0.015
```

Bacalah tabelnya per kelas. **Hutan alam** stabil: rata-rata 0,86, amplitudo hanya 0,10, hampir tidak punya tren. **Pertanian** naik turun: amplitudo 0,46. **Deforestasi** dan **panen hutan tanaman** punya tren turun tajam (-0,281 dan -0,200 per tahun). Baris terakhir menunjukkan tanda awal kebakaran: pada 16 September 2024, NDVI di bekas terbakar turun 0,115 dari kebiasaannya, sedangkan hutan stabil hanya 0,015.

Bagian yang boleh diubah.

- Baris 18, `365.0`. Pengali satuan tren. Saya mengubahnya menjadi `30.0`, yaitu perubahan per 30 hari. Tren deforestasi di tabel berubah dari `-0.281` menjadi `-0.023`.
- **Baris 33, tanggal anomali.** Ganti ke tanggal lain yang ada di `Daftar_Citra.csv`. Saya mengganti menjadi `date(2025, 8, 17)`. Layar menampilkan `Anomali NDVI rata-rata di bekas terbakar pada 2024-09-16: -0.031; di hutan stabil: -0.020.`
- **Baris 12-14, statistik tambahan.** Tambah `x.std(0)` atau `np.median(x, axis=0)` untuk metrik baru, lalu tambahkan namanya ke daftar baris 20.

Jebakan.

- Teks di `print` tidak ikut berubah. Pada contoh di atas, tanggal diganti tetapi teksnya masih tertulis `2024-09-16`. Angka yang tampil memang milik 17 Agustus 2025, tetapi labelnya salah. Setelah mengganti tanggal, ganti juga tulisannya di baris 35. Pada tanggal itu pun angkanya mencampur dua kebakaran, sebab kelas 7 mencakup bekas bakar tahun 2024 dan 2025.
- **Tanggal harus ada di daftar.** Saya mengganti baris 33 menjadi `date(2024, 9, 17)` dan layar menampilkan `ValueError: datetime.date(2024, 9, 17) is not in list.`
- **Skrip 26.6 harus sudah jalan.** Di folder kosong, layar menampilkan `RuntimeError: ...NDVI_isi.tif: No such file or directory.`
- **Selang antar tanggal tidak sama.** Karena itu tren dihitung terhadap hari, bukan nomor urut. Tren pada data dua tahun hanya cukup untuk belajar, belum untuk laporan resmi [CEK].

Latihan variasi.

1. Cetak juga **hari puncak rata-rata** tiap kelas.
2. Bandingkan kelas yang kemudian berubah (deforestasi, terbakar) dengan hutan alam.

Contoh jawaban, tambahkan di akhir skrip:

```python
for k, n in kls.items():
    print("%-14s hari puncak rata-rata: %5.0f" % (n, puncak[din == k].mean()))
```

Saya menjalankannya. Layar menampilkan delapan baris tambahan. Tiga di antaranya: `Hutan alam hari puncak rata-rata: 303`, `Deforestasi hari puncak rata-rata: 113`, dan `Terbakar hari puncak rata-rata: 87`. Kelas yang kemudian berubah mencapai puncaknya **jauh lebih awal** (sekitar hari ke-100) daripada hutan alam (hari ke-303). Wajar: setelah kejadian, NDVI-nya jatuh dan tidak naik lagi.

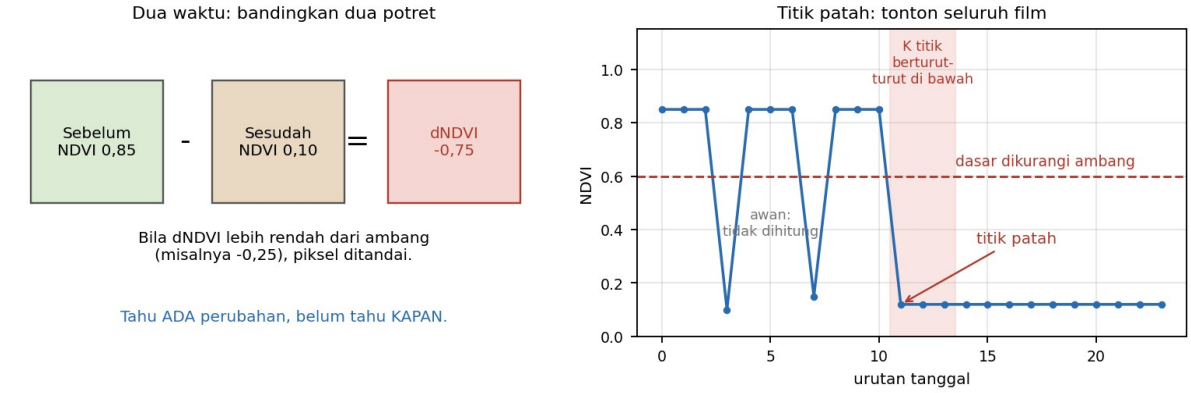
> **Pola yang bisa dipakai ulang:** Pola **ringkas per piksel, lalu ringkas per kelas** berlaku untuk metrik apa saja. Ganti **berkas masukan** (baris 11), **rumus metrik** (baris 12-19), dan **peta kelas** (baris 23-24). Selebihnya, yaitu menulis band dan mencetak tabel, tidak berubah.

Kelompok D: Mendeteksi hutan yang hilang

Konsep: dua foto lawan seluruh film

Deteksi dua waktu itu seperti permainan "temukan perbedaan" antara foto sebelum dan sesudah: cepat dan mudah, tetapi hanya tahu bahwa sesuatu berubah, bukan kapan. **Deteksi titik patah** menonton seluruh film setiap piksel, dan mencatat saat nilainya jatuh lalu tidak kembali. Cara kedua bisa menyebut tanggal, tetapi lebih lambat.

Ilustrasi 26.15: Dua waktu lawan titik patah



Skema; bukan tangkapan layar; angka contoh

Lima istilah yang perlu Anda kenal sekarang:

- **dNDVI**: NDVI sesudah dikurangi NDVI sebelum. Nilai sangat negatif berarti kehijauan hilang.
- **Ambang**: batas angka untuk menyebut sesuatu "berubah".
- **Luas minimum**: gumpalan piksel yang lebih kecil dari batas ini dibuang sebagai derau.
- **Titik patah**: saat sebuah deret berubah nilai secara menetap.
- **Pengamatan berturut-turut**: syarat bahwa penurunan harus bertahan beberapa tanggal, supaya awan yang lolos tidak menipu.

Kepala Seksi berpesan satu hal: **panen hutan tanaman bukan deforestasi.** Itu tebangan terencana. Kedua skrip di bawah membuang kawasan hutan tanaman memakai peta `Kawasan_HTI.gpkg`.

Skrip 26.8: Deforestasi dari dua waktu

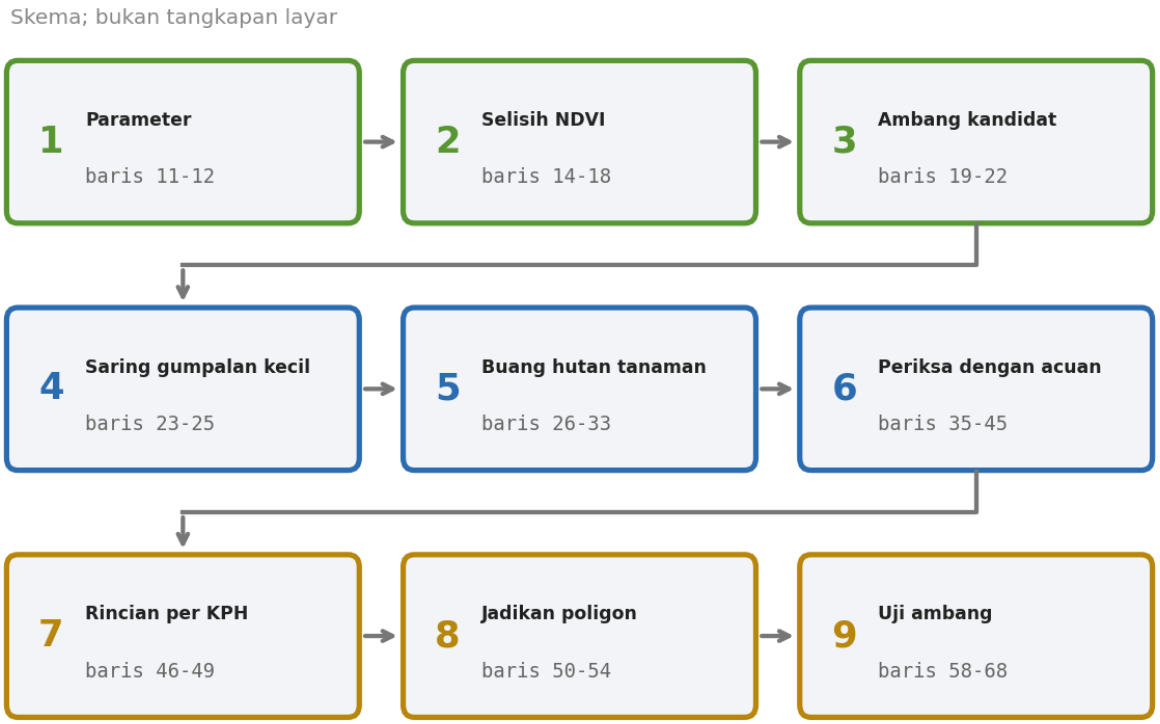
Berkas `4_1_deforestasi_dua_waktu.py`. Di dalam berkasnya tertulis "SKRIP 4.1".

Tujuan. Mendeteksi deforestasi dari selisih komposit NDVI sebelum dan sesudah, dengan ambang dan saringan luas minimum. Lalu membuang kawasan hutan tanaman, dan memeriksa hasilnya terhadap peta acuan.

Bahan. Hasil Skrip 26.5 (komposit median `sebelum` dan `sesudah`), `vektor/Kawasan_HTI.gpkg`, `acuan/Acuan_Perubahan.tif`, `acuan/Acuan_Dinamika.tif`, dan `m3_umum.py`. Jalankan dari Konsol atau terminal.

Skrip ini panjang dan memuat sembilan tahap. Gambar berikut membantu Anda membaginya menjadi bagian kecil.

Ilustrasi 26.16: Alur Skrip 26.8



Skrip utuh:

```python
# -*- coding: utf-8 -*-
# SKRIP 4.1: Deteksi deforestasi dua waktu: selisih komposit NDVI (sesudah dikurangi sebelum), ambang, saringan luas minimum
# Penulis: Badar Mubarok Yogaswara
# Syarat: Skrip 2.2 sudah dijalankan. Ambang dan luas minimum bisa Anda ubah di bagian PARAMETER.
import os
import numpy as np
import processing
from qgis.core import QgsRasterLayer, QgsVectorLayer, QgsProject
import m3_umum as U

AMBANG = -0.25          # selisih NDVI di bawah nilai ini dianggap kehilangan tutupan
LUAS_MIN_PIKSEL = 5     # gumpalan lebih kecil dari ini (5 piksel = 0,05 ha) dibuang sebagai derau

sebelum = QgsRasterLayer(os.path.join(U.HASIL, "komposit_ndvi_sebelum_median.tif"), "sebelum")
sesudah = QgsRasterLayer(os.path.join(U.HASIL, "komposit_ndvi_sesudah_median.tif"), "sesudah")
QgsProject.instance().addMapLayers([sebelum, sesudah], False)
d_ndvi = os.path.join(U.HASIL, "dNDVI.tif")
processing.run("native:rastercalc", {"LAYERS": [sebelum, sesudah], "EXPRESSION": '"sesudah@1" - "sebelum@1"',
"OUTPUT": d_ndvi})
# 1) ambang: hanya piksel yang hutan sebelumnya (NDVI sebelum >= 0.7) dan turun lewat AMBANG
kand = os.path.join(U.HASIL, "kandidat_deforestasi.tif")
processing.run("native:rastercalc", {"LAYERS": [sebelum, sesudah],
    "EXPRESSION": '(("sebelum@1" >= 0.7) AND (("sesudah@1" - "sebelum@1") < %s)) * 1' % AMBANG, "OUTPUT":
kand})
# 2) saring gumpalan kecil
saring = os.path.join(U.HASIL, "kandidat_disaring.tif")
```

```
processing.run("gdal:sieve", {"INPUT": kand, "THRESHOLD": LUAS_MIN_PIKSEL, "EIGHT_CONNECTEDNESS": True,
"NO_MASK": True, "OUTPUT": saring})
# 3) buang kawasan hutan tanaman (panen HTI itu terencana, bukan deforestasi)
hti = os.path.join(U.HASIL, "mask_hti.tif")
processing.run("gdal:rasterize", {"INPUT": os.path.join(U.PAKET, "vektor", "Kawasan_HTI.gpkg"), "BURN": 1,
"UNITS": 1,
    "WIDTH": 10, "HEIGHT": 10, "EXTENT": "312000,312400,9996000,9996400 [EPSG:32749]", "INIT": 0, "NODATA":
255, "DATA_TYPE": 0, "OUTPUT": hti})
final = os.path.join(U.HASIL, "deforestasi_2waktu.tif")
l1, l2 = QgsRasterLayer(saring, "saring"), QgsRasterLayer(hti, "hti")
QgsProject.instance().addMapLayers([l1, l2], False)
processing.run("native:rastercalc", {"LAYERS": [l1, l2], "EXPRESSION": '("saring@1" = 1) AND ("hti@1" = 0)',
"OUTPUT": final})

# pemeriksaan terhadap peta acuan
hasil, gt, _ = U.baca_tif(final)
hasil = hasil == 1
acuan, _, _ = U.baca_tif(os.path.join(U.PAKET, "acuan", "Acuan_Perubahan.tif"))
benar = acuan == 1
tp = (hasil & benar).sum(); fp = (hasil & ~benar).sum(); fn = (~hasil & benar).sum()
print("Terdeteksi: %d piksel = %.2f ha; acuan: %d piksel = %.2f ha" % (hasil.sum(), hasil.sum() * 0.01,
benar.sum(), benar.sum() * 0.01))
print("Tepat %d, salah tambah %d, terlewat %d; presisi %.2f, kepekaan %.2f" % (tp, fp, fn, tp / max(tp + fp,
1), tp / max(tp + fn, 1)))
din, _, _ = U.baca_tif(os.path.join(U.PAKET, "acuan", "Acuan_Dinamika.tif"))
nama = {1: "Hutan alam", 2: "Hutan tanaman", 3: "Pertanian", 4: "Air", 5: "Terbuka", 6: "Deforestasi", 7:
"Terbakar", 8: "Panen HTI"}
print("Piksel terdeteksi menurut kelas acuan:", {nama[k]: int((hasil & (din == k)).sum()) for k in range(1, 9)
if (hasil & (din == k)).sum()})
rr, cc = np.mgrid[0:40, 0:40]
kph = np.where(rr < 20, "Gamma", np.where(cc < 20, "Alpha", "Beta"))
for k in ("Alpha", "Beta", "Gamma"):
    print("  %-6s deforestasi terdeteksi %.2f ha, acuan %.2f ha" % (k, (hasil & (kph == k)).sum() * 0.01,
(benar & (kph == k)).sum() * 0.01))
poli = os.path.join(U.HASIL, "deforestasi_2waktu.gpkg")
if os.path.exists(poli):
    os.remove(poli)                  # polygonize ke GPKG yang sudah ada MENAMBAH fitur, bukan menimpa
processing.run("gdal:polygonize", {"INPUT": final, "BAND": 1, "FIELD": "DN", "EIGHT_CONNECTEDNESS": True,
"OUTPUT": poli})
print("Poligon tersimpan:", poli)



# uji kepekaan ambang (tanpa saringan luas minimum, dengan dan tanpa peta HTI)
dn, _, _ = U.baca_tif(os.path.join(U.HASIL, "dNDVI.tif"))
sb, _, _ = U.baca_tif(os.path.join(U.HASIL, "komposit_ndvi_sebelum_median.tif"))
htim, _, _ = U.baca_tif(hti)
print("Ambang   dengan peta HTI (tepat/salah/lewat)   tanpa peta HTI (tepat/salah/lewat)")
for a in (-0.15, -0.25, -0.35, -0.45):
    k = (sb >= 0.7) & (dn < a)
    for_hti = k & (htim == 0)
    t1, f1, l1_ = (for_hti & benar).sum(), (for_hti & ~benar).sum(), (~for_hti & benar).sum()
    t2, f2, l2_ = (k & benar).sum(), (k & ~benar).sum(), (~k & benar).sum()
    print("%6.2f   %4d / %4d / %4d                   %4d / %4d / %4d" % (a, t1, f1, l1_, t2, f2, l2_))
```

Penjelasan baris demi baris. Baris 55-57 kosong, jadi tidak muncul di tabel.

Baris	Tulisan	Artinya
5-9	`import os` sampai `import m3_umum as U`	Perkakas. `QgsVectorLayer` diimpor tetapi tidak dipakai di skrip ini.
11	`AMBANG = -0.25`	**Konstanta**: kotak yang sengaja ditulis dengan huruf besar sebagai tanda "nilai ini boleh Anda ubah". Selisih NDVI di bawah -0,25 dianggap kehilangan tutupan.

Baris	Tulisan	Artinya
12	`LUAS_MIN_PIKSEL = 5`	Gumpalan lebih kecil dari lima piksel (0,05 hektare) dibuang.
14-15	`sebelum = QgsRasterLayer(...)` dan `sesudah = ...`	Buka dua komposit median sebagai layer bernama `sebelum` dan `sesudah`.
16	`QgsProject.instance().addMapLayers([sebelum, sesudah], False)`	Daftarkan dua layer ke proyek, tanpa ditampilkan.
17-18	`d_ndvi = ...` lalu `processing.run("native:rastercalc", ...)`	Hitung selisih `"sesudah@1" - "sebelum@1"` dan simpan sebagai `dNDVI.tif`.
20-22	`kand = ...` dan `processing.run(...)`	Hitung peta **kandidat**: bernilai 1 bila NDVI sebelum sekurang-kurangnya 0,7 (dulunya hutan) **dan** selisihnya di bawah ambang. `%s` diganti oleh nilai `AMBANG`. Perkalian `* 1` mengubah hasil benar atau salah menjadi angka 1 atau 0.
24-25	`saring = ...` lalu `processing.run("gdal:sieve", {...})`	Alat Sieve membuang gumpalan kecil. `THRESHOLD` diisi `LUAS_MIN_PIKSEL`. `EIGHT_CONNECTEDNESS: True` berarti piksel yang bersentuhan di sudut dianggap satu gumpalan. `NO_MASK: True` berarti tidak memakai penanda validitas bawaan.
27-29	`hti = ...` lalu `processing.run("gdal:rasterize", {...})`	Ubah peta vektor hutan tanaman menjadi raster. `BURN: 1` mengisi 1 di dalam kawasan, `INIT: 0` mengisi 0 di luar. `UNITS: 1` berarti ukuran diberikan dalam satuan peta. `WIDTH` dan `HEIGHT` 10 berarti piksel 10 meter. `EXTENT` menentukan batas area: di sinilah ukuran area dikunci. `DATA_TYPE: 0` memilih Byte.
30-33	`final = ...` sampai `processing.run(...)`	Buka saringan dan mask HTI sebagai layer, lalu hitung hasil akhir: `("saring@1" = 1) AND ("hti@1" = 0)`, yaitu terdeteksi dan **bukan** hutan tanaman.
36-37	`hasil, gt, _ = U.baca_tif(final)` lalu `hasil = hasil == 1`	Baca hasil akhir, lalu jadikan larik benar atau salah.
38-39	`acuan, _, _ = ...` lalu `benar = acuan == 1`	Baca peta acuan perubahan. Nilai 1 berarti deforestasi sebenarnya.
40	`tp = ...; fp = ...; fn = ...`	Tiga pernyataan dalam satu baris, dipisah titik koma. `tp` (tepat) adalah terdeteksi dan benar. `fp` (salah tambah) adalah terdeteksi tetapi salah. `fn` (terlewat) adalah benar tetapi tidak terdeteksi. Tanda `~` membalik benar-salah.
41	`print("Terdeteksi: %d piksel = %.2f ha; ...")`	Cetak jumlah piksel dan luasnya. Luas dihitung dari jumlah piksel kali 0,01 hektare, angka yang dikunci untuk piksel 10 meter.
42	`print("Tepat %d, salah tambah %d, ...")`	Cetak `tp`, `fp`, `fn`, serta **presisi** (`tp` dibagi `tp + fp`) dan **kepekaan** (`tp` dibagi `tp + fn`). `max(tp + fp, 1)` mencegah pembagian dengan nol.

Baris	Tulisan	Artinya
43-44	`din, _, _ = ...` lalu `nama = {...}`	Baca peta kelas tutupan, dan kamus nama kelas.
45	`print("...", {nama[k]: int((hasil & (din == k)).sum()) for k in range(1, 9) if (...).sum()})`	**Dictionary comprehension**: kamus yang dibangun dalam satu baris. Ia menghitung, untuk tiap kelas acuan, berapa piksel terdeteksi, dan hanya mencatat kelas yang jumlahnya lebih dari nol.
46	`rr, cc = np.mgrid[0:40, 0:40]`	Dua larik 40 kali 40 yang berisi nomor baris dan nomor kolom tiap piksel. Ukuran 40 dikunci di sini.
47	`kph = np.where(rr < 20, "Gamma", np.where(cc < 20, "Alpha", "Beta"))`	Menandai KPH tiap piksel. Baris bernomor kurang dari 20 (setengah utara) adalah Gamma. Sisanya, kolom kiri (nomor kurang dari 20) adalah Alpha dan kolom kanan adalah Beta. Ini **tiruan** pembagian wilayah, bukan dibaca dari peta batas.
48-49	`for k in ("Alpha", "Beta", "Gamma"):`	Cetak luas terdeteksi dan luas acuan tiap KPH.
50	`poli = os.path.join(...)`	Jalur berkas GeoPackage untuk poligon hasil.
51-52	`if os.path.exists(poli):` lalu `os.remove(poli)`	**Hapus berkas lama.** `os.path.exists` bertanya apakah berkas sudah ada (Bab 8). Bila ada, `os.remove` menghapusnya. Alat Polygonize menambah poligon ke GeoPackage yang sudah ada dan tidak menimpanya, jadi tanpa dua baris ini jalan kedua menumpuk poligon.
53	`processing.run("gdal:polygonize", {...})`	Ubah hasil akhir menjadi poligon di berkas GeoPackage.
54	`print("Poligon tersimpan:", poli)`	Cetak lokasi berkas poligon.
59-61	`dn, _, _ = ...` sampai `htim, _, _ = ...`	Baca ulang selisih NDVI, komposit sebelum, dan mask HTI untuk uji ambang.
62	`print("Ambang  dengan peta HTI ...")`	Cetak judul tabel uji.
63	`for a in (-0.15, -0.25, -0.35, -0.45):`	Ulangi untuk empat ambang berbeda.
64-65	`k = (sb >= 0.7) & (dn < a)` lalu `for_hti = k & (htim == 0)`	Hitung kandidat tanpa saringan luas, lalu versi yang membuang hutan tanaman.
66-67	`t1, f1, l1_ = ...` dan `t2, f2, l2_ = ...`	Hitung tepat, salah, dan terlewat untuk kedua versi. Nama `l1_` memakai garis bawah di akhir supaya tidak bentrok dengan `l1` di baris 31.
68	`print("%6.2f  %4d / %4d / %4d ...")`	Cetak satu baris tabel uji ambang.

Hasil. Layar menampilkan (berdasarkan uji pada QGIS 4.0.2):

```
HASIL
Terdeteksi: 130 piksel = 1.30 ha; acuan: 130 piksel = 1.30 ha
Tepat 130, salah tambah 0, terlewat 0; presisi 1.00, kepekaan 1.00
```

```
Piksel terdeteksi menurut kelas acuan: {'Deforestasi': 130}
  Alpha  deforestasi terdeteksi 0.30 ha, acuan 0.30 ha
  Beta   deforestasi terdeteksi 0.00 ha, acuan 0.00 ha
  Gamma  deforestasi terdeteksi 1.00 ha, acuan 1.00 ha
Poligon tersimpan: D:/Latihan_M3/hasil\deforestasi_2waktu.gpkg
Ambang   dengan peta HTI (tepat/salah/lewat)   tanpa peta HTI (tepat/salah/lewat)
 -0.15    130 /   13 /    0                     130 /  133 /    0
 -0.25    130 /    1 /    0                     130 /   24 /    0
 -0.35    118 /    1 /   12                     118 /    1 /   12
 -0.45     26 /    0 /  104                      26 /    0 /  104
```

Hasilnya sempurna: 130 piksel (1,30 hektare) terdeteksi, semuanya benar, tidak ada yang terlewat. Rincian per KPH cocok dengan acuan. Data ini sintetis, jadi jangan berharap sebersih ini pada data nyata. Tabel uji ambang adalah bagian paling berguna. Pada ambang -0,15, kedua versi sama-sama menangkap semua 130 piksel. Namun versi dengan peta hutan tanaman salah 13 piksel, sedangkan versi tanpa peta salah 133 piksel. Itulah gunanya membuang kawasan hutan tanaman.

Bagian yang boleh diubah.

- Baris 11, `AMBANG`. Makin negatif, makin ketat. Saya mencoba `-0.35`: layar menampilkan `Terdeteksi: 118 piksel = 1.18 ha` dan `Tepat 118, salah tambah 0, terlewat 12; presisi 1.00, kepekaan 0.91`.
- Baris 12, `LUAS_MIN_PIKSEL`. Ubah menjadi `1`: terdeteksi 131 piksel, dengan satu salah tambah (di kelas Pertanian). Ubah menjadi `60`: terdeteksi hanya 100 piksel, dan bukaan kecil di Alpha yang 30 piksel ikut terbuang (`terlewat 30`, kepekaan 0,77). Saringan yang terlalu besar membuang kejadian nyata.
- Baris 22, `0.7`. Ambang NDVI hutan sebelum perubahan.

Jebakan.

- **Banyak hal dikunci untuk data contoh.** `EXTENT` di baris 29 (`312000,312400,9996000,9996400`), ukuran piksel 10 meter, ukuran 40 kali 40 di baris 46, luas 0,01 hektare per piksel di baris 41 dan 49, serta pembagian KPH `rr < 20` di baris 47. Pada data lain, semuanya harus diganti, atau hasilnya salah tanpa pesan galat.
- **Alat Polygonize menambah, bukan menimpa.** Itu sebabnya baris 51-52 menghapus `deforestasi_2waktu.gpkg` lebih dulu. Saya menjalankan skrip dua kali pada folder yang sama, dan jumlah poligon tetap 3 pada jalan kedua. Tanpa dua baris itu, jumlahnya naik menjadi 6. Bila Windows menolak penghapusan karena berkas itu masih terbuka sebagai layer di QGIS, `os.remove` gagal dengan `PermissionError: [WinError 32]` (diuji; layer di kanvas jendela QGIS tidak diuji). Hapus layernya dari proyek, lalu jalankan ulang.
- **Nama layer harus sama persis dengan nama di rumus.** `"sebelum@1"` mengacu pada layer bernama `sebelum`.
- **Skrip 26.5 harus sudah jalan.** Di folder kosong, layar menampilkan `QgsProcessingException: Error creating output file.` Pesan itu menyesatkan: yang hilang adalah **masukan**, bukan keluaran.
- **Skrip ini panjang dan mencampur tiga pekerjaan**: mendeteksi, memeriksa, dan menguji ambang. Memecahnya menjadi tiga berkas akan lebih rapi, tetapi buku ini membahasnya apa adanya.

Latihan variasi.

1. Ubah ambang menjadi `-0.35`, lalu bandingkan hasilnya dengan ambang asli.
2. Ubah luas minimum menjadi `1`, lalu cari tahu piksel salah tambah itu masuk kelas apa.

Contoh jawaban, hanya baris yang berubah:

```python
AMBANG = -0.35          # selisih NDVI di bawah nilai ini dianggap kehilangan tutupan
LUAS_MIN_PIKSEL = 1     # gumpalan lebih kecil dari ini dibuang sebagai derau
```

Untuk ambang `-0.35`, layar menampilkan 118 piksel (turun dari 130), dengan 12 terlewat. Untuk luas minimum `1`, layar menampilkan `Piksel terdeteksi menurut kelas acuan: {'Pertanian': 1, 'Deforestasi': 130}`, jadi piksel salah itu ada di kelas Pertanian. Tiap variasi saya jalankan terpisah, dengan salah satu konstanta saja yang berubah.

> **Pola yang bisa dipakai ulang:** Deteksi perubahan dua waktu selalu berbentuk **selisih, ambang, saring, topeng, periksa**. Ganti **dua layer masukan** (baris 14-15), **rumus ambang** (baris 22), **berkas topeng** (baris 28), dan **dua konstanta** (baris 11-12). Letakkan konstanta di bagian atas supaya mudah diubah.

Skrip 26.9: Titik patah, kapan hutan mulai hilang

Berkas `4_2_titik_patah.py`. Di dalam berkasnya tertulis "SKRIP 4.2".

Tujuan. Mencari kapan tiap piksel hutan mulai hilang: NDVI turun jauh dari dasarnya pada beberapa pengamatan bersih berturut-turut. Lalu menyimpan tanggalnya dan membandingkannya dengan tanggal sebenarnya.

Bahan. Citra mentah (lewat `m3_umum.py`), peta acuan (`Acuan_Dinamika.tif` dan `Acuan_Perubahan.tif`), dan `mask_hti.tif` dari Skrip 26.8. Skrip ini tidak memakai QGIS. Jalankan dari Konsol atau terminal.

Ilustrasi 26.17: Alur Skrip 26.9



Skrip utuh:

```python
# -*- coding: utf-8 -*-
# SKRIP 4.2: Deteksi titik patah pada deret waktu NDVI: kapan hutan mulai hilang?
# Penulis: Badar Mubarok Yogaswara
# Aturan: dasar = median 6 pengamatan bersih pertama. Perubahan = NDVI turun lebih dari AMBANG dari dasar
# pada K pengamatan bersih berturut-turut. Tanggal perubahan = pengamatan pertama dalam rangkaian itu.
import datetime as dt
import os
import numpy as np
from osgeo import gdal
import m3_umum as U

AMBANG, K, N_DASAR = 0.25, 3, 6
tgl, refl, scl, gt, prj = U.baca_deret()
ndvi = U.nilai_ndvi(refl)
bersih = U.bersih(scl)
T, H, W = ndvi.shape
hari = np.array([(t - tgl[0]).days for t in tgl])


def deteksi(pakai_mask, ambang=AMBANG, k=K):
    ok = bersih if pakai_mask else np.ones_like(bersih)
    hasil = np.full((H, W), -1)            # indeks tanggal awal perubahan, -1 = tidak ada
    for i in range(H):
        for j in range(W):
            idx = np.where(ok[:, i, j])[0]
            if len(idx) < N_DASAR + k:
                continue
            dasar = np.median(ndvi[idx[:N_DASAR], i, j])
            if dasar < 0.7:                # hanya piksel yang awalnya berhutan
                continue
            deret = ndvi[idx[N_DASAR:], i, j]
            turun = deret < dasar - ambang
            for n in range(len(turun) - k + 1):
                if turun[n:n + k].all():
                    hasil[i, j] = idx[N_DASAR + n]
                    break
    return hasil


din, _, _ = U.baca_tif(os.path.join(U.PAKET, "acuan", "Acuan_Dinamika.tif"))
per, _, _ = U.baca_tif(os.path.join(U.PAKET, "acuan", "Acuan_Perubahan.tif"))
hti, _, _ = U.baca_tif(os.path.join(U.HASIL, "mask_hti.tif"))     # dibuat Skrip 4.1
benar = per == 1
print("K  mask-awan  tepat salah(di luar HTI) lewat")
for pakai in (False, True):
    for k in (1, 2, 3):
        h = deteksi(pakai, k=k) >= 0
        h = h & (hti == 0)
        print("%d  %-9s  %5d %5d %5d" % (k, "ya" if pakai else "tidak", (h & benar).sum(), (h & ~benar).sum(),
(~h & benar).sum()))

idx = deteksi(True)
flag = (idx >= 0) & (hti == 0)
keluar = np.where(flag, np.array([int(tgl[i].strftime("%Y%m%d")) if i >= 0 else 0 for i in
idx.ravel()]).reshape(H, W), 0).astype("int32")
U.tulis_tif(os.path.join(U.HASIL, "tanggal_deforestasi.tif"), keluar, gt, prj, nodata=0, tipe=gdal.GDT_Int32)
# tanggal sebenarnya (acuan): bukaan 1 pada 2024-10-28, bukaan 2 pada 2025-06-05
benar_tgl = {"Bukaan 1 (Gamma)": (dt.date(2024, 10, 28), benar & (np.mgrid[0:H, 0:W][0] < 20)),
             "Bukaan 2 (Alpha)": (dt.date(2025, 6, 5), benar & (np.mgrid[0:H, 0:W][0] >= 20))}
for nama, (tg, m) in benar_tgl.items():
    terdeteksi = [tgl[idx[i, j]] for i, j in zip(*np.where(m & flag))]
    tunda = [(d - tg).days for d in terdeteksi]
    pertama = min(terdeteksi)
```

```
    print("%s: tanggal sebenarnya %s | terdeteksi %d piksel | tanggal terdeteksi paling awal %s, paling akhir %s | tunda rata-rata %.0f hari"
          % (nama, tg, len(terdeteksi), pertama, max(terdeteksi), np.mean(tunda)))
```

Penjelasan baris demi baris. Fungsi `deteksi` adalah inti skrip ini, jadi diuraikan lebih rinci.

Baris	Tulisan	Artinya
4-5	komentar `# Aturan: ...`	Aturan dalam kata-kata: dasar adalah median enam pengamatan bersih pertama. Perubahan terjadi bila NDVI turun lebih dari `AMBANG` dari dasar pada `K` pengamatan bersih berturut-turut.
6-10	`import datetime as dt` sampai `import m3_umum as U`	Perkakas. `datetime` diberi nama pendek `dt`. `from osgeo import gdal` (baris 9) dipakai untuk nama tipe data di baris 54.
12	`AMBANG, K, N_DASAR = 0.25, 3, 6`	Tiga konstanta dalam satu baris: ambang penurunan, jumlah pengamatan berturut-turut, dan jumlah pengamatan untuk dasar.
13-15	`tgl, refl, scl, gt, prj = ...` dan dua baris di bawahnya	Baca deret, hitung NDVI, dan buat larik penanda bersih. Kotak `bersih` di sini adalah **larik**, bukan fungsi.
16-17	`T, H, W = ndvi.shape` dan `hari = ...`	Ukuran data, dan hari ke-berapa tiap tanggal.
20	`def deteksi(pakai_mask, ambang=AMBANG, k=K):`	Fungsi dengan tiga masukan. Dua yang terakhir punya **nilai bawaan**: bila tidak diisi, dipakai konstanta di atas (Bab 7).
21	`ok = bersih if pakai_mask else np.ones_like(bersih)`	Bila `pakai_mask` benar, pakai penanda bersih. Bila tidak, anggap semua pengamatan baik (larik berisi benar semua).
22	`hasil = np.full((H, W), -1)`	Larik hasil berisi -1, artinya "belum ada perubahan".
23-24	`for i in range(H):` dan `for j in range(W):`	Kunjungi tiap piksel.
25	`idx = np.where(ok[:, i, j])[0]`	Daftar nomor tanggal yang boleh dipakai di piksel ini.
26-27	`if len(idx) < N_DASAR + k:` lalu `continue`	Bila pengamatan terlalu sedikit, lewati piksel ini dan lanjut ke piksel berikutnya.
28	`dasar = np.median(ndvi[idx[:N_DASAR], i, j])`	Dasar: median dari enam pengamatan bersih pertama.
29-30	`if dasar < 0.7:` lalu `continue`	Hanya piksel yang awalnya berhutan (NDVI dasar sekurang-kurangnya 0,7) yang diperiksa.
31	`deret = ndvi[idx[N_DASAR:], i, j]`	Sisa pengamatan bersih setelah enam yang pertama.
32	`turun = deret < dasar - ambang`	Larik benar atau salah: benar bila nilai lebih rendah dari dasar dikurangi ambang.
33-36	`for n in range(len(turun) - k + 1):` sampai `break`	Geser jendela selebar `k`. Bila semua `k` nilai dalam jendela benar (`.all()`), catat nomor tanggal pertama jendela itu (baris 35) dan hentikan (`break`).

Baris	Tulisan	Artinya
37	`return hasil`	Kembalikan larik nomor tanggal, -1 bila tidak ada.
40-43	`din, _, _ = ...` sampai `benar = per == 1`	Baca peta kelas, peta perubahan, dan mask HTI hasil Skrip 26.8. `benar` adalah piksel deforestasi sebenarnya.
44-49	`print("K  mask-awan ...")` dan dua putaran	Uji enam kombinasi: tanpa dan dengan mask awan, dengan `K` 1, 2, dan 3. Tiap baris menghitung tepat, salah (di luar hutan tanaman), dan terlewat.
51	`idx = deteksi(True)`	Jalankan hasil akhir: dengan mask awan, dengan `K` bawaan (3).
52	`flag = (idx >= 0) & (hti == 0)`	Piksel yang terdeteksi dan bukan hutan tanaman.
53	`keluar = np.where(flag, np.array([...]).reshape(H, W), 0).astype("int32")`	Ubah nomor tanggal menjadi angka berbentuk `YYYYMMDD`, misalnya 20241115. `.ravel()` meratakan larik, `.reshape(H, W)` mengembalikannya. `.astype("int32")` memilih bilangan bulat 32 bit.
54	`U.tulis_tif(..., keluar, gt, prj, nodata=0, tipe=gdal.GDT_Int32)`	Simpan ke `tanggal_deforestasi.tif`. `nodata=0` menandai 0 sebagai "kosong". `gdal.GDT_Int32` adalah tipe data **bilangan bulat 32 bit**, cukup untuk angka sampai sekitar dua miliar. Tanpa `tipe=`, `tulis_tif` memilih Byte (hanya 0 sampai 255) dan tanggal terpotong. Lihat Jebakan.
56-57	`benar_tgl = {...}`	Kamus dua bukaan: nama, tanggal sebenarnya, dan wilayah piksel. `np.mgrid[0:H, 0:W][0] < 20` memilih baris bernomor kurang dari 20 (Gamma). Yang kedua memakai `>= 20` (Alpha).
58-60	`for nama, (tg, m) in benar_tgl.items():`	Untuk tiap bukaan, kumpulkan tanggal terdeteksi tiap piksel dan hitung selisih harinya dari tanggal sebenarnya (`tunda`).
61	`pertama = min(terdeteksi)`	Tanggal terdeteksi paling awal.
62-63	`print("%s: tanggal sebenarnya %s ...")`	Cetak ringkasan: jumlah piksel, tanggal paling awal dan paling akhir, dan rata-rata tundaan.

Hasil. Layar menampilkan (berdasarkan uji pada QGIS 4.0.2):

```
HASIL
K  mask-awan  tepat salah(di luar HTI) lewat
1  tidak        128   873     2
2  tidak        128   593     2
3  tidak        128   156     2
1  ya           130    92     0
2  ya           130     8     0
3  ya           130     0     0
Bukaan 1 (Gamma): tanggal sebenarnya 2024-10-28 | terdeteksi 100 piksel | tanggal terdeteksi paling awal 2024-11-15, paling akhir 2025-01-14 | tunda rata-rata 24 hari
Bukaan 2 (Alpha): tanggal sebenarnya 2025-06-05 | terdeteksi 30 piksel | tanggal terdeteksi paling awal 2025-06-18, paling akhir 2025-06-18 | tunda rata-rata 13 hari
```

Bacalah tabel enam baris itu. Tanpa mask awan dan dengan `K` = 1, ada **873 piksel salah**: satu awan saja sudah dianggap hutan hilang. Dengan mask awan dan `K` = 3, salahnya **nol** dan semua 130 piksel benar. Dua syarat itu, yaitu membuang awan dan menunggu beberapa pengamatan berturut-turut, mengubah hasil total. Tundaan rata-rata 24 hari untuk bukaan Gamma kemungkinan berasal dari selang antar citra yang sekitar 30 hari dan dari tiga pengamatan yang harus ditunggu [kemungkinan].

Bagian yang boleh diubah.

- Baris 12, `K`. Ubah ke `2`. Hasil akhir bertambah dari 130 menjadi 138 piksel bertanggal, karena delapan piksel salah ikut masuk (sama seperti baris `2 ya` di tabel uji). Saya menambahkan `print("Piksel bertanggal:", int(flag.sum()))` di akhir untuk melihat angkanya.
- Baris 12, `AMBANG`. Makin kecil, makin peka tetapi makin mudah tertipu.
- Baris 29, `0.7`. Batas NDVI hutan untuk dasar.

Jebakan.

Tipe data harus disebut sendiri (baris 54). Fungsi `U.tulis_tif` memilih Byte untuk semua larik non-desimal (`m3_umum.py` baris 65-66). Byte hanya memuat 0 sampai 255, jadi bilangan seperti 20241115 akan terpotong menjadi 255. Karena itu baris 54 menyebut `tipe=gdal.GDT_Int32`. Saya memeriksa berkas hasil dengan potongan ini:

```
PYTHON
from osgeo import gdal
import numpy as np
a = gdal.Open(r"D:/Latihan_M3/hasil/tanggal_deforestasi.tif")
print(gdal.GetDataTypeName(a.GetRasterBand(1).DataType), np.unique(a.ReadAsArray()))
```

```
HASIL
Int32 [       0 20241115 20241220 20250114 20250618]
```

Lima nilai itu benar: nol untuk tidak ada perubahan, ditambah empat tanggal. Tanpa `tipe=`, berkas yang sama terbaca `Byte [  0 255]`: semua tanggal rusak menjadi 255, dan skrip tidak mengeluh. Ringkasan di layar (baris 58-63) tidak terpengaruh karena memakai larik di memori, bukan berkas. Hanya berkas `.tif` yang salah. Jebakan lainnya:

- `min()` pada daftar kosong menimbulkan galat. Saya mengganti `AMBANG` menjadi `0.99`, sehingga tidak ada piksel terdeteksi. Layar menampilkan `ValueError: min() iterable argument is empty` di baris 61.
- **Skrip 26.8 harus sudah jalan.** Di folder kosong, layar menampilkan `RuntimeError: ...mask_hti.tif: No such file or directory`.
- **Tanggal sebenarnya dan pembagian wilayah dikunci di kode.** Baris 56-57 hanya cocok dengan data contoh.
- **Putaran per piksel lambat untuk raster besar**, seperti di Skrip 26.6.

Latihan variasi.

1. Ubah `K` menjadi 2 dan lihat berapa piksel bertanggal.

2. Hapus `tipe=gdal.GDT_Int32` dari baris 54, jalankan ulang, lalu periksa tipe berkasnya.

Contoh jawaban nomor 2, satu perubahan pada baris 54. Buang `tipe`:

```python
U.tulis_tif(os.path.join(U.HASIL, "tanggal_deforestasi.tif"), keluar, gt, prj, nodata=0)
```

Layar tidak menampilkan galat. Memeriksa berkas dengan potongan di Jebakan, layar menampilkan `Byte [  0 255]`: semua tanggal terpotong menjadi 255. Kembalikan `tipe=gdal.GDT_Int32` setelah mencoba. Angka K=2 untuk nomor 1 adalah 138.

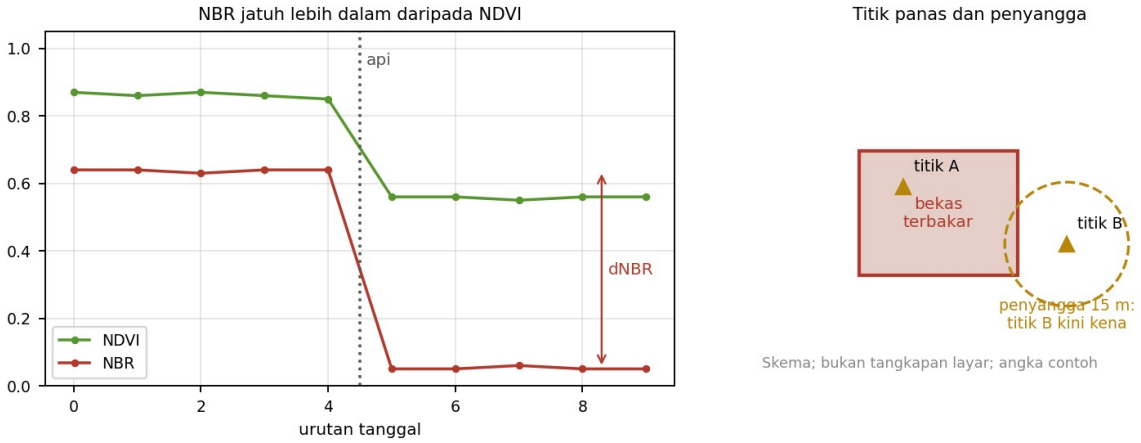
> **Pola yang bisa dipakai ulang:** Deteksi titik patah selalu berbentuk **dasar, ambang, berturut-turut K**. Ganti tiga konstanta di baris 12, ambang hutan di baris 29, dan sumber deret. Bila Anda menyimpan angka besar (tanggal, kode), **sebutkan tipe datanya** pada `tulis_tif`.

Kelompok E: Jejak api

Konsep: arang yang menyerap hijau

Tumbuhan sehat memantulkan banyak inframerah dekat dan sedikit inframerah gelombang pendek. Setelah terbakar, keadaannya terbalik, seperti **arang yang menelan cahaya**. Indeks **NBR** menangkap pembalikan itu. Selisih NBR antara sebelum dan sesudah kebakaran, yaitu **dNBR**, menunjukkan seberapa parah terbakarnya.

Ilustrasi 26.18: NBR dan titik panas



Lima istilah yang perlu Anda kenal sekarang:

- **SWIR**: inframerah gelombang pendek. Di Sentinel-2, SWIR2 adalah band B12.
- **NBR**: (inframerah dekat dikurangi SWIR2) dibagi (inframerah dekat ditambah SWIR2).
- **dNBR**: NBR sebelum dikurangi NBR sesudah. Makin besar, makin parah.
- **Titik panas** (*hotspot*): titik deteksi suhu tinggi oleh satelit. Titik panas adalah petunjuk api, **bukan** peta bekas bakar.
- **Penyangga** (*buffer*): daerah selebar jarak tertentu di sekeliling sebuah objek.

Mengapa NBR, bukan NDVI saja? Pada contoh grafik di atas, NDVI hanya turun sekitar 0,3, sedangkan NBR turun hampir 0,6. Gerakan yang lebih besar lebih mudah dibedakan dari derau.

Skrip 26.10: Bekas kebakaran dengan dNBR

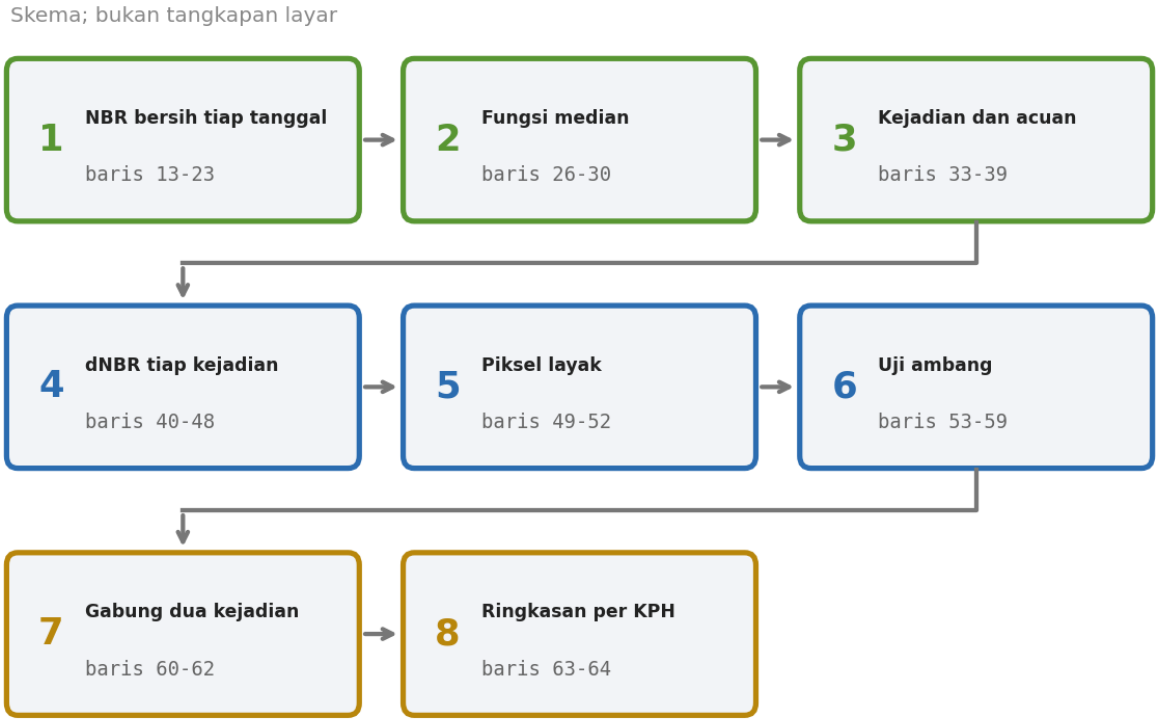
Berkas `5_1_dnbr.py`. Di dalam berkasnya tertulis "SKRIP 5.1".

Tujuan. Memetakan bekas kebakaran dua kejadian (September 2024 di KPH Beta, Agustus 2025 di KPH Gamma) dengan NBR dan dNBR, menguji beberapa ambang, dan menyimpan peta akhir.

Bahan. Citra mentah (berkas `_refl.tif` dan `_SCL.tif`), `acuan/Acuan_Dinamika.tif`, `acuan/Tutupan_Awal.tif`, dan `m3_umum.py`. Skrip ini tidak memerlukan skrip lain. Jalankan dari Konsol atau terminal.

Skrip ini panjang. Gambar berikut membaginya menjadi delapan tahap.

Ilustrasi 26.19: Alur Skrip 26.10



Skrip utuh:

```python
# -*- coding: utf-8 -*-
# SKRIP 5.1: Bekas kebakaran dengan NBR dan dNBR (dua kejadian: September 2024 dan Agustus 2025)
# Penulis: Badar Mubarok Yogaswara
# NBR = (NIR - SWIR2) / (NIR + SWIR2); untuk Sentinel-2: NIR = B08 (band 4 di berkas), SWIR2 = B12 (band 6 di berkas).
# dNBR = NBR sebelum - NBR sesudah. Makin besar, makin parah terbakar.
import os
import numpy as np
import processing
from qgis.core import QgsRasterLayer, QgsProject
import m3_umum as U
```

```
AMBANG_BAKAR = 0.10          # dNBR di atas ini dianggap bekas terbakar (lihat uji ambang di akhir keluaran)
daftar = U.daftar_citra()
os.makedirs(os.path.join(U.HASIL, "nbr_bersih"), exist_ok=True)
nbr = []
for tgl, pr, ps in daftar:
    k = tgl.strftime("%Y%m%d")
    a, s = QgsRasterLayer(pr, "R" + k), QgsRasterLayer(ps, "S" + k)
    QgsProject.instance().addMapLayers([a, s], False)
    ek = '(("R%s@4" - "R%s@6") / ("R%s@4" + "R%s@6")) / (("S%s@1" = 4) OR ("S%s@1" = 5) OR ("S%s@1" = 6))' %
((k,) * 7)
    out = os.path.join(U.HASIL, "nbr_bersih", "NBR_bersih_%s.tif" % k)
    processing.run("native:rastercalc", {"LAYERS": [a, s], "EXPRESSION": ek, "OUTPUT": out})
    nbr.append(out)


def median(idx, nama):
    out = os.path.join(U.HASIL, nama)
    processing.run("native:cellstatistics", {"INPUT": [nbr[i] for i in idx], "STATISTIC": 3, "IGNORE_NODATA":
True,
                                             "REFERENCE_LAYER": nbr[idx[0]], "OUTPUT_NODATA_VALUE": -9999,
"OUTPUT": out})
    return out


tg = [t for t, _, _ in daftar]
kejadian = {"2024": ([5, 6, 7], [8, 9], "Beta"), "2025": ([17, 18], [19, 20], "Gamma")}   # indeks tanggal
sebelum, sesudah, KPH
din, gt, prj = U.baca_tif(os.path.join(U.PAKET, "acuan", "Acuan_Dinamika.tif"))
awal, _, _ = U.baca_tif(os.path.join(U.PAKET, "acuan", "Tutupan_Awal.tif"))
rr, cc = np.mgrid[0:40, 0:40]
kph = np.where(rr < 20, "Gamma", np.where(cc < 20, "Alpha", "Beta"))
sapuan = {}
for nama, (pre, post, wilayah) in kejadian.items():
    a, b = median(pre, "nbr_pre_%s.tif" % nama), median(post, "nbr_post_%s.tif" % nama)
    la, lb = QgsRasterLayer(a, "pre"), QgsRasterLayer(b, "post")
    QgsProject.instance().addMapLayers([la, lb], False)
    d = os.path.join(U.HASIL, "dNBR_%s.tif" % nama)
    processing.run("native:rastercalc", {"LAYERS": [la, lb], "EXPRESSION": '"pre@1" - "post@1"', "OUTPUT": d})
    dn, _, _ = U.baca_tif(d)
    pre_n, _, _ = U.baca_tif(a)
    dn = np.where(dn < -100, np.nan, dn)                      # -9999 = tidak ada data
    acuan = (din == 7) & (kph == wilayah)                     # bekas terbakar kejadian ini (KPH tempat
kejadian)
    # hanya hutan dan hutan tanaman menurut peta tutupan awal, yang NBR-nya tinggi (air dan sawah dibuang: NBR-
nya tidak stabil)
    layak = (pre_n >= 0.5) & np.isin(awal, (1, 2))
    print("Kejadian %s (KPH %s): sebelum %s s.d. %s, sesudah %s s.d. %s" % (nama, wilayah, tg[pre[0]],
tg[pre[-1]], tg[post[0]], tg[post[-1]]))
    print("  dNBR rata-rata di bekas terbakar acuan %.2f; di hutan stabil %.2f; acuan %d piksel (%.2f ha)"
          % (np.nanmean(dn[acuan]), np.nanmean(dn[din == 1]), acuan.sum(), acuan.sum() * 0.01))
    print("  ambang   terdeteksi(ha)  tepat salah lewat  presisi kepekaan")
    for amb in (0.05, 0.10, 0.20, 0.27, 0.44):
        bk = layak & (dn > amb)
        tp, fp, fn = (bk & acuan).sum(), (bk & ~acuan).sum(), (~bk & acuan).sum()
        print("  %5.2f   %8.2f      %4d %5d %4d    %.2f    %.2f" % (amb, bk.sum() * 0.01, tp, fp, fn, tp /
max(tp + fp, 1), tp / max(tp + fn, 1)))
    sapuan[nama] = layak & (dn > AMBANG_BAKAR)
akhir = sapuan["2024"] | sapuan["2025"]
U.tulis_tif(os.path.join(U.HASIL, "bekas_bakar.tif"), akhir.astype("uint8"), gt, prj, nodata=255)
for k in ("Alpha", "Beta", "Gamma"):
    print("KPH %-6s terbakar terdeteksi %.2f ha (acuan %.2f ha)" % (k, (akhir & (kph == k)).sum() * 0.01, ((din
== 7) & (kph == k)).sum() * 0.01))
```

Penjelasan baris demi baris.

Baris	Tulisan	Artinya
4-5	komentar	Rumus dalam kata-kata: NBR memakai band 4 (B08) dan band 6 (B12) di berkas. dNBR adalah NBR sebelum dikurangi NBR sesudah.
6-10	`import os` sampai `import m3_umum as U`	Perkakas, seperti sebelumnya.
12	`AMBANG_BAKAR = 0.10`	Konstanta: dNBR di atas 0,10 dianggap bekas terbakar.
13-15	`daftar = ...`, `os.makedirs(...nbr_bersih...)`, `nbr = []`	Daftar tanggal, folder keluaran, dan daftar penampung jalur.
16-17	`for tgl, pr, ps in daftar:` dan `k = ...`	Ulangi per tanggal dengan kode tanggal.
18	`a, s = QgsRasterLayer(pr, "R" + k), QgsRasterLayer(ps, "S" + k)`	Buka layer refl bernama `R...` dan layer SCL bernama `S...` sekaligus, memakai penugasan ganda.
19	`QgsProject.instance().addMapLayers([a, s], False)`	Daftarkan keduanya ke proyek tanpa ditampilkan.
20	`ek = '...' % ((k,) * 7)`	Rumus NBR dibagi kondisi SCL baik (trik "bagi nol menjadi NoData" seperti Skrip 26.4). Rumus ini memuat tujuh `%s`. `(k,) * 7` membuat tuple berisi tujuh salinan `k`. Jumlahnya harus pas.
21-23	`out = ...`, `processing.run(...)`, `nbr.append(out)`	Jalankan kalkulator raster dan catat jalur hasilnya.
26-30	`def median(idx, nama):`	Fungsi yang menghitung komposit median NBR untuk daftar tanggal `idx`. Baris 28 memakai alat Cell statistics dengan `STATISTIC: 3` (median). `[nbr[i] for i in idx]` mengambil jalur berkas tanggal-tanggal itu. Baris 30 mengembalikan jalur hasil.
33	`tg = [t for t, _, _ in daftar]`	Daftar tanggal saja.
34	`kejadian = {"2024": ([5, 6, 7], [8, 9], "Beta"), ...}`	Kamus dua kejadian. Isi tiap kejadian: nomor tanggal sebelum, nomor tanggal sesudah, dan KPH tempatnya. Nomor-nomor itu dikunci sesuai data contoh.
35-36	`din, gt, prj = ...` dan `awal, _, _ = ...`	Baca peta kelas tutupan dan peta tutupan awal.
37-38	`rr, cc = np.mgrid[0:40, 0:40]` dan `kph = np.where(...)`	Penanda KPH tiruan, seperti di Skrip 26.8. Ukuran 40 dikunci di sini.
39	`sapuan = {}`	Kamus kosong untuk menyimpan hasil tiap kejadian.
40	`for nama, (pre, post, wilayah) in kejadian.items():`	Ulangi per kejadian dan bongkar tiga isinya.
41	`a, b = median(pre, ...), median(post, ...)`	Hitung komposit median NBR sebelum dan sesudah. Memakai fungsi buatan sendiri.

Baris	Tulisan	Artinya
42-45	`la, lb = ...,addMapLayers(...), d = ...,processing.run(...)`	Buka kedua komposit sebagai layer `pre` dan `post`, lalu hitung `"pre@1" - "post@1"` sebagai dNBR.
46-47	`dn, _, _ = U.baca_tif(d)` dan `pre_n, _, _ = U.baca_tif(a)`	Baca dNBR dan NBR sebelum sebagai larik.
48	`dn = np.where(dn < -100, np.nan, dn)`	Nilai -9999 (tanda kosong) diubah menjadi `nan`, supaya tidak ikut dihitung.
49	`acuan = (din == 7) & (kph == wilayah)`	Bekas terbakar sebenarnya untuk kejadian ini: kelas 7 di KPH tempat kejadian.
51	`layak = (pre_n >= 0.5) & np.isin(awal, (1, 2))`	Piksel yang **layak diuji**: NBR sebelum sekurang-kurangnya 0,5, dan tutupan awal hutan alam (1) atau hutan tanaman (2). Air dan sawah dibuang karena NBR-nya tidak stabil.
52	`print("Kejadian %s (KPH %s): ...")`	Cetak tanggal awal dan akhir periode sebelum dan sesudah.
53-54	`print("  dNBR rata-rata di bekas terbakar acuan ...")`	Cetak dNBR rata-rata di bekas bakar sebenarnya dan di hutan stabil. `np.nanmean` mengabaikan `nan`.
55	`print("  ambang terdeteksi(ha)  ...")`	Judul tabel uji.
56-59	`for amb in (0.05, 0.10, 0.20, 0.27, 0.44):`	Untuk tiap ambang uji: `bk` adalah piksel layak dengan dNBR di atas ambang (baris 57), lalu hitung tepat, salah, terlewat, presisi, dan kepekaan seperti di Skrip 26.8.
60	`sapuan[nama] = layak & (dn > AMBANG_BAKAR)`	Simpan hasil kejadian ini, memakai ambang resmi 0,10.
61	`akhir = sapuan["2024"]` diikuti operator atau dan `sapuan["2025"]`	Gabungkan dua kejadian dengan operator **atau** (tanda garis tegak tunggal di berkas): benar bila piksel terbakar pada salah satu kejadian.
62	`U.tulis_tif(..."bekas_bakar.tif"..., akhir.astype("uint8"), ..., nodata=255)`	Simpan peta akhir sebagai Byte berisi 0 dan 1, nilai kosong 255. Byte memang cukup untuk 0 dan 1.
63-64	`for k in ("Alpha", "Beta", "Gamma"):`	Cetak luas terbakar terdeteksi tiap KPH, dibanding luas acuan.

Hasil. Layar menampilkan (berdasarkan uji pada QGIS 4.0.2):

```
HASIL
Kejadian 2024 (KPH Beta): sebelum 2024-06-13 s.d. 2024-08-12, sesudah 2024-09-16 s.d. 2024-10-16
  dNBR rata-rata di bekas terbakar acuan 0.35; di hutan stabil -0.01; acuan 128 piksel (1.28 ha)
  ambang   terdeteksi(ha)  tepat salah lewat  presisi kepekaan
   0.05       1.38       128    10    0    0.93    1.00
   0.10       1.29       128     1    0    0.99    1.00
   0.20       1.14       114     0   14    1.00    0.89
   0.27       0.91        91     0   37    1.00    0.71
   0.44       0.30        30     0   98    1.00    0.23
Kejadian 2025 (KPH Gamma): sebelum 2025-06-18 s.d. 2025-07-18, sesudah 2025-08-17 s.d. 2025-09-21
  dNBR rata-rata di bekas terbakar acuan 0.42; di hutan stabil 0.02; acuan 23 piksel (0.23 ha)
  ambang   terdeteksi(ha)  tepat salah lewat  presisi kepekaan
   0.05       0.30        23     7    0    0.77    1.00
```

```
   0.10       0.23        23     0    0    1.00    1.00
   0.20       0.22        22     0    1    1.00    0.96
   0.27       0.19        19     0    4    1.00    0.83
   0.44       0.09         9     0   14    1.00    0.39
KPH Alpha  terbakar terdeteksi 0.01 ha (acuan 0.00 ha)
KPH Beta   terbakar terdeteksi 1.28 ha (acuan 1.28 ha)
KPH Gamma  terbakar terdeteksi 0.23 ha (acuan 0.23 ha)
```

Baca per kejadian. Pada ambang 0,10, kejadian 2024 mendapat 128 piksel tepat, hanya satu salah, dan tidak ada yang terlewat. Kejadian 2025 mendapat 23 piksel, semuanya tepat. Makin tinggi ambang, makin banyak yang terlewat: pada 0,44 kepekaan tinggal 0,23 dan 0,39. Angka uji 0,27 dan 0,44 berasal dari kelas keparahan literatur yang dimuat di Buku M3, dan literatur itu berasal dari hutan di wilayah lain. Pada data ini, ambang terbaik justru 0,10. **Kalibrasi ambang dengan data Anda sendiri.** Baris `KPH Alpha` mencatat satu piksel salah tambah (0,01 hektare), karena Alpha tidak punya bekas terbakar sama sekali.

Bagian yang boleh diubah.

- Baris 12, `AMBANG_BAKAR`. Saya mengubahnya menjadi `0.20`. Layar menampilkan `KPH Beta terbakar terdeteksi 1.14 ha (acuan 1.28 ha)` dan `KPH Gamma terbakar terdeteksi 0.22 ha (acuan 0.23 ha)`, dan Alpha kini `0.00 ha`. Satu piksel salah di Alpha hilang, tetapi 14 piksel Beta terlewat.
- **Baris 34, nomor tanggal.** Pilih tanggal sebelum dan sesudah untuk kejadian Anda. Nomor mulai dari 0.
- Baris 51, syarat `layak`. Longgarkan atau ketatkan batas 0,5 dan kelas tutupan.
- **Baris 56, daftar ambang uji.** Tambah atau kurangi angkanya.

Jebakan.

- **Rumus di baris 20 sangat panjang.** Jumlah `%s` harus sama dengan jumlah isian. Saya mengubah `* 7` menjadi `* 6`, dan layar menampilkan `TypeError: not enough arguments for format string`. Pesan ini bagus: ia langsung menunjuk masalahnya.
- **Banyak hal dikunci.** Nomor tanggal di baris 34, ukuran 40 kali 40 di baris 37, pembagian KPH di baris 38, dan luas 0,01 hektare per piksel di baris 54, 59, dan 64. Semuanya hanya benar untuk data contoh.
- **Nilai -9999 dianggap kosong.** Baris 48 menganggap setiap dNBR di bawah -100 sebagai kosong. Bila data Anda memakai tanda kosong lain, ganti.
- Variabel `a`, `b`, dan `k` dipakai ulang. Nama `a` dan `b` di baris 41 berbeda artinya dari baris 18. Python mengizinkannya, tetapi mudah membingungkan.
- **Pembaca dari terminal.** Skrip ini memanggil alat QGIS 30 kali, jadi dari terminal butuh persiapan Bab 3.

Latihan variasi.

1. Naikkan `AMBANG_BAKAR` menjadi `0.20`.
2. Bandingkan luas terbakar per KPH dengan hasil asli.

Contoh jawaban, hanya baris 12:

```
PYTHON
AMBANG_BAKAR = 0.20          # dNBR di atas ini dianggap bekas terbakar (lihat uji ambang di akhir keluaran)
```

Tiga baris terakhir layar menjadi `KPH Alpha  terbakar terdeteksi 0.00 ha (acuan 0.00 ha)`, `KPH Beta  terbakar terdeteksi 1.14 ha (acuan 1.28 ha)`, dan `KPH Gamma  terbakar terdeteksi 0.22 ha (acuan 0.23 ha)`. Hasil asli (0,10) adalah 0,01; 1,28; dan 0,23 hektare. Ambang yang lebih tinggi menghilangkan salah tambah, tetapi menambah yang terlewat.

> **Pola yang bisa dipakai ulang:** Pola **median sebelum, median sesudah, selisih, ambang, evaluasi** berlaku untuk kejadian apa pun, termasuk banjir atau hama. Ganti **indeks** (baris 20), **nomor tanggal** (baris 34), **syarat layak** (baris 51), dan **ambang** (baris 12). Tabel uji ambang di baris 55-59 adalah kebiasaan yang patut ditiru: jangan percaya satu ambang tanpa mengujinya.

Skrip 26.11: Mencocokkan titik panas dan memperkirakan tanggal terbakar

Berkas `5_2_titik_panas.py`. Di dalam berkasnya tertulis "SKRIP 5.2".

Tujuan. Mengubah peta bekas bakar menjadi poligon, mencocokkan titik panas dengan poligon itu (dengan dan tanpa penyangga), memperkirakan tanggal terbakar tiap piksel, dan membandingkannya dengan tanggal titik panas.

Bahan. `bekas_bakar.tif` dari Skrip 26.10, `vektor/Hotspot_Bertanggal.gpkg` (12 titik panas, kolom `ID_Titik` dan `TANGGAL`), dan `m3_umum.py`. Jalankan dari Konsol atau terminal.

Ilustrasi 26.20: Alur Skrip 26.11



Skrip utuh:

```
PYTHON
# -*- coding: utf-8 -*-
# SKRIP 5.2: Cocokkan titik panas dengan bekas terbakar, perkirakan tanggal terbakar, hitung luas per KPH
# Penulis: Badar Mubarok Yogaswara
```

```
# Syarat: Skrip 5.1 sudah dijalankan (hasil/bekas_bakar.tif).
import datetime as dt
import os
import numpy as np
import processing
from qgis.core import QgsVectorLayer
import m3_umum as U

bakar = os.path.join(U.HASIL, "bekas_bakar.tif")
poli = os.path.join(U.HASIL, "bekas_bakar.gpkg")
if os.path.exists(poli):
    os.remove(poli)                  # polygonize ke GPKG yang sudah ada MENAMBAH fitur, bukan menimpa
processing.run("gdal:polygonize", {"INPUT": bakar, "BAND": 1, "FIELD": "DN", "EIGHT_CONNECTEDNESS": True,
"OUTPUT": poli})
lp = QgsVectorLayer(poli + "|layername=bekas_bakar", "bekas_bakar")
lp.setSubsetString('"DN" = 1')                                        # hanya gumpalan terbakar (bukan latar)
print("Poligon bekas terbakar:", lp.featureCount())
hs = QgsVectorLayer(os.path.join(U.PAKET, "vektor", "Hotspot_Bertanggal.gpkg"), "hotspot")


def dalam(jarak):
    sumber = hs
    if jarak > 0:
        sumber = processing.run("native:buffer", {"INPUT": hs, "DISTANCE": jarak, "SEGMENTS": 8, "OUTPUT":
"TEMPORARY_OUTPUT"})["OUTPUT"]
    r = processing.run("native:extractbylocation", {"INPUT": sumber, "PREDICATE": [0], "INTERSECT": lp,
"OUTPUT": "TEMPORARY_OUTPUT"})["OUTPUT"]
    return sorted(f["ID_Titik"] for f in r.getFeatures())


for j in (0, 15, 30):
    ids = dalam(j)
    print("Titik panas di dalam bekas terbakar (penyangga %d m): %d dari %d -> %s" % (j, len(ids),
hs.featureCount(), ", ".join(ids)))

# perkiraan tanggal terbakar per piksel: pengamatan bersih pertama dengan NBR turun lebih dari 0.2 dari dasar
(3 pengamatan awal bersih)
tgl, refl, scl, gt, prj = U.baca_deret()
nbr = U.nilai_nbr(refl)
ok = U.bersih(scl)
b, _, _ = U.baca_tif(bakar)
tanggal_bakar = {}
for i, j in zip(*np.where(b == 1)):
    idx = np.where(ok[:, i, j])[0]
    dasar = np.median(nbr[idx[:3], i, j])
    for t in idx[3:]:
        if nbr[t, i, j] < dasar - 0.2:
            tanggal_bakar[(i, j)] = tgl[t]
            break
from collections import Counter
print("Perkiraan tanggal terbakar (jumlah piksel):", dict(sorted(Counter(tanggal_bakar.values()).items())))
# hubungan dengan titik panas
for f in hs.getFeatures():
    p = f.geometry().asPoint()
    i, j = U.xy_ke_piksel(gt, p.x(), p.y())
    tg_h = dt.date.fromisoformat(f["TANGGAL"])
    tb = tanggal_bakar.get((i, j))
    print("%s %s -> %s" % (f["ID_Titik"], tg_h, ("bekas terbakar, tanggal perkiraan %s (selisih %d hari)" %
(tb, (tb - tg_h).days)) if tb else "tidak ada bekas terbakar di pikselnya"))
```

Penjelasan baris demi baris. Baris 11, 21-22, 29-30, dan 34 kosong.

Baris	Tulisan	Artinya
5-10	`import datetime as dt` sampai `import m3_umum as U`	Perkakas. `QgsVectorLayer` adalah layer vektor.
12-13	`bakar = ...` dan `poli = ...`	Jalur peta bekas bakar (raster) dan jalur hasil poligon.
14-15	`if os.path.exists(poli):` lalu `os.remove(poli)`	**Hapus berkas lama.** `os.path.exists` bertanya apakah berkas sudah ada (Bab 8). Bila ada, `os.remove` menghapusnya. Alat Polygonize menambah poligon ke GeoPackage yang sudah ada dan tidak menimpanya, jadi tanpa dua baris ini jalan kedua menggandakan poligon.
16	`processing.run("gdal:polygonize", {...})`	Ubah raster menjadi poligon. `FIELD: "DN"` memberi nama kolom nilai piksel. `EIGHT_CONNECTEDNESS: True` menyatukan piksel yang bersentuhan di sudut.
17	`lp = QgsVectorLayer(poli + "...layername=bekas_bakar", "bekas_bakar")`	Buka poligon sebagai layer. Teks sumber memakai garis tegak lalu nama layer di dalam berkas (pola yang sama dengan Bab 15). Penyedia tidak disebut, dan QGIS memakai bawaannya (`ogr`).
18	`lp.setSubsetString('"DN" = 1')`	Pasang **filter**: hanya poligon bernilai 1 (terbakar), bukan latar belakang.
19	`print("Poligon bekas terbakar:", lp.featureCount())`	Cetak jumlah poligon terbakar.
20	`hs = QgsVectorLayer(...Hotspot_Bertanggal.gpkg, "hotspot")`	Buka layer titik panas.
23	`def dalam(jarak):`	Fungsi dengan satu masukan: jarak penyangga dalam meter.
24	`sumber = hs`	Awalnya sumbernya titik panas apa adanya.
25-26	`if jarak > 0:` lalu `sumber = processing.run("native:buffer", {...})["OUTPUT"]`	Bila jarak lebih dari nol, buat penyangga dari tiap titik. `SEGMENTS: 8` mengatur kehalusan lingkaran. `"TEMPORARY_OUTPUT"` menyimpan hasil di memori. Kurung siku `["OUTPUT"]` mengambil layer dari hasil alat.
27	`r = processing.run("native:extractbylocation", {...})["OUTPUT"]`	Pilih titik atau penyangga yang bersinggungan dengan poligon terbakar. `PREDICATE: [0]` berarti "bersinggungan" (`intersect`). Saya memeriksanya di bantuan alat.
28	`return sorted(f["ID_Titik"] for f in r.getFeatures())`	Kembalikan daftar ID titik terpilih, terurut.
31-33	`for j in (0, 15, 30):` dan dua baris di bawahnya	Uji tiga penyangga: 0, 15, dan 30 meter. Cetak berapa titik panas yang cocok. `", ".join(ids)` menyambung daftar ID dengan koma.
36-38	`tgl, refl, scl, gt, prj = U.baca_deret()` dan dua baris di bawahnya	Baca deret, hitung NBR, dan penanda bersih.

Baris	Tulisan	Artinya
39	`b, _, _ = U.baca_tif(bakar)`	Baca peta bekas bakar sebagai larik.
40	`tanggal_bakar = {}`	Kamus kosong: kuncinya pasangan (baris, kolom), isinya tanggal.
41	`for i, j in zip(*np.where(b == 1)):`	`np.where(b == 1)` memberi dua daftar: nomor baris dan nomor kolom piksel terbakar. `zip(*...)` menyandingkannya menjadi pasangan `(i, j)`.
42-43	`idx = ...` dan `dasar = np.median(nbr[idx[:3], i, j])`	Pengamatan bersih piksel ini, dan dasar NBR dari tiga pengamatan bersih pertama.
44-47	`for t in idx[3:]:` sampai `break`	Cari pengamatan bersih pertama yang NBR-nya turun lebih dari 0,2 dari dasar. Catat tanggalnya dan berhenti.
48	`from collections import Counter`	Impor di tengah berkas. `Counter` menghitung berapa kali tiap nilai muncul. Boleh, tetapi biasanya impor ditaruh di atas.
49	`print("Perkiraan tanggal terbakar ...", dict(sorted(Counter(...).items())))`	Hitung berapa piksel terbakar menurut tiap tanggal perkiraan, lalu cetak sebagai kamus terurut.
51-52	`for f in hs.getFeatures():` dan `p = f.geometry().asPoint()`	Untuk tiap titik panas, ambil koordinatnya.
53	`i, j = U.xy_ke_piksel(gt, p.x(), p.y())`	Ubah koordinat menjadi nomor baris dan kolom piksel.
54	`tg_h = dt.date.fromisoformat(f["TANGGAL"])`	Ubah teks tanggal titik panas menjadi objek tanggal.
55	`tb = tanggal_bakar.get((i, j))`	Cari tanggal perkiraan terbakar di piksel itu. `.get` mengembalikan `None` (kosong) bila tidak ada.
56	`print("%s %s -> %s" % (..., (... if tb else ...)))`	Cetak ID, tanggal titik panas, lalu pesan. Bila `tb` ada: tanggal perkiraan dan selisih harinya. Bila tidak: "tidak ada bekas terbakar di pikselnya".

Hasil. Layar menampilkan (berdasarkan uji pada QGIS 4.0.2):

```
HASIL
Poligon bekas terbakar: 4
Titik panas di dalam bekas terbakar (penyangga 0 m): 4 dari 12 -> HS-006, HS-009, HS-010, HS-011
Titik panas di dalam bekas terbakar (penyangga 15 m): 7 dari 12 -> HS-002, HS-004, HS-005, HS-006, HS-009, HS-010, HS-011
Titik panas di dalam bekas terbakar (penyangga 30 m): 8 dari 12 -> HS-002, HS-004, HS-005, HS-006, HS-009, HS-010, HS-011, HS-012
Perkiraan tanggal terbakar (jumlah piksel): {datetime.date(2024, 9, 16): 122, datetime.date(2024, 10, 16): 2, datetime.date(2025, 8, 17): 23}
HS-001 2024-10-27 -> tidak ada bekas terbakar di pikselnya
HS-002 2025-08-11 -> tidak ada bekas terbakar di pikselnya
HS-003 2024-10-27 -> tidak ada bekas terbakar di pikselnya
HS-004 2024-09-03 -> tidak ada bekas terbakar di pikselnya
HS-005 2024-09-05 -> tidak ada bekas terbakar di pikselnya
HS-006 2024-09-04 -> bekas terbakar, tanggal perkiraan 2024-09-16 (selisih 12 hari)
HS-007 2024-10-28 -> tidak ada bekas terbakar di pikselnya
```

```
HS-008 2025-03-14 -> tidak ada bekas terbakar di pikselnya
HS-009 2024-09-04 -> bekas terbakar, tanggal perkiraan 2024-09-16 (selisih 12 hari)
HS-010 2025-08-12 -> bekas terbakar, tanggal perkiraan 2025-08-17 (selisih 5 hari)
HS-011 2024-09-04 -> bekas terbakar, tanggal perkiraan 2024-09-16 (selisih 12 hari)
HS-012 2025-03-14 -> tidak ada bekas terbakar di pikselnya
```

Dengan penyangga 0 meter, hanya 4 dari 12 titik panas jatuh di dalam bekas terbakar. Dengan penyangga 30 meter, 8 titik. Empat titik terakhir tetap di luar. Saya memeriksa kelas acuan di piksel tiap titik: HS-001 dan HS-007 jatuh di piksel deforestasi, dan bertanggal akhir Oktober 2024, saat bukaan Gamma terjadi (28 Oktober 2024). Jadi **titik panas tidak selalu berarti kebakaran**. Pembukaan lahan bisa ikut menyalakannya [kemungkinan]. Tanggal perkiraan terbakar selalu **sesudah** tanggal titik panas (selisih 12 dan 5 hari), karena tanggal itu adalah citra bersih pertama yang menunjukkan penurunan, sedangkan titik panas dideteksi saat api menyala.

Bagian yang boleh diubah.

- **Baris 31, daftar penyangga.** Saya mengganti menjadi `(0, 50)`: layar menampilkan `penyangga 50 m: 9 dari 12` (ID `HS-002, 004, 005, 006, 008, 009, 010, 011, 012`).
- Baris 45, `0.2`. Ambang penurunan NBR untuk tanggal terbakar.
- **Baris 43 dan 44, angka 3.** Jumlah pengamatan dasar. Jangan lebih besar dari jumlah pengamatan bersih yang ada.

Jebakan.

- **Alat Polygonize menambah, bukan menimpa.** Itu sebabnya baris 14-15 menghapus `bekas_bakar.gpkg` lebih dulu. Saya menjalankan skrip dua kali di folder yang sama, dan kedua putaran mencetak `Poligon bekas terbakar: 4`. Tanpa dua baris itu, putaran kedua mencetak 8, karena poligon baru ditumpuk di atas yang lama. Bila Windows menolak penghapusan karena berkas itu masih terbuka sebagai layer di QGIS, `os.remove` gagal dengan `PermissionError: [WinError 32]` (diuji; layer di kanvas jendela QGIS tidak diuji). Hapus layernya dari proyek, lalu jalankan ulang.
- **Skrip 26.10 harus sudah jalan.** Di folder kosong, layar menampilkan `QgsProcessingException: Unable to execute algorithm` dan `Could not load source layer for INPUT: ...bekas_bakar.tif not found`.
- Nama `j` dipakai dua kali. Baris 31 dan baris 41 sama-sama memakai `j`. Keduanya berurutan, jadi aman. Namun menamai ulang salah satunya mencegah kekacauan bila skrip dikembangkan.
- **Penyangga yang terlalu lebar tidak bermakna.** Area contoh hanya 400 meter kali 400 meter. Saya mencoba `(0, 100, 200)`: layar menampilkan 4, 10, dan 12 dari 12. Pada 200 meter semua titik "cocok", tanpa arti.
- **Butuh minimal tiga pengamatan bersih.** Baris 43 dan 44 mengandaikan `idx` sekurang-kurangnya sepanjang tiga.

Latihan variasi.

1. Ganti penyangga menjadi `(0, 50)`.
2. Catat titik panas yang baru masuk pada 50 meter, dibandingkan 30 meter.

Contoh jawaban, hanya baris 31:

```
PYTHON
for j in (0, 50):
```

Layar menampilkan `penyangga 0 m: 4 dari 12` dan `penyangga 50 m: 9 dari 12 -> HS-002, HS-004, HS-005, HS-006, HS-008, HS-009, HS-010, HS-011, HS-012`. Pada 30 meter daftarnya delapan ID. Titik baru pada 50 meter: **HS-008**. Setelah itu coba ingat: HS-008 bertanggal 14 Maret 2025, sedangkan kebakaran di data ini terjadi September 2024 dan Agustus 2025. Cocok dalam jarak, tidak cocok dalam waktu.

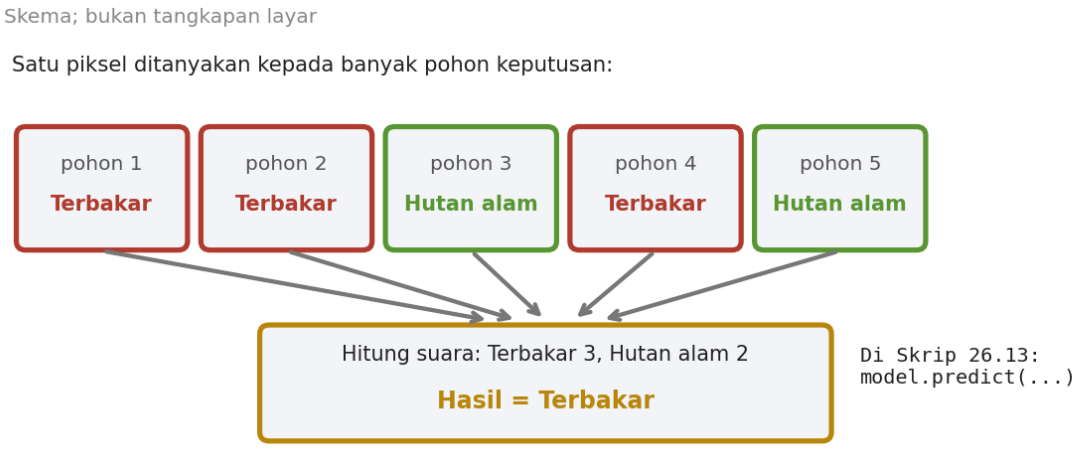
> **Pola yang bisa dipakai ulang:** Untuk mencocokkan titik dengan poligon, ganti **layer poligon** (baris 17-18), **layer titik** (baris 20), dan **daftar jarak** (baris 31). Bila poligon disimpan ke berkas, hapus berkas lama lebih dulu seperti baris 14-15. Selalu uji beberapa jarak dan **bandingkan dengan tanggalnya**, supaya titik yang kebetulan dekat tidak dianggap cocok.

Kelompok F: Mesin yang belajar dari contoh

Konsep: panitia juri dalam tiga kalimat

Sebuah **pohon keputusan** itu seperti permainan tebak-tebakan bercabang: "NBR minimumnya di bawah 0,2? Ya, maka terbakar." Sebuah **hutan acak** adalah ratusan pohon yang masing-masing belajar dari contoh dan fitur yang sedikit berbeda, lalu **memberi suara**. Kelas dengan suara terbanyak menang, seperti keputusan panitia yang lebih dapat dipercaya daripada satu juri.

Ilustrasi 26.21: Hutan acak memberi suara



Lima istilah yang perlu Anda kenal sekarang:

- **Titik latih**: titik yang kelasnya sudah diketahui, dipakai mesin untuk belajar. Paket data punya 200 titik, 25 untuk tiap dari delapan kelas.
- **Pohon keputusan**: aturan bercabang "jika... maka...".
- **Hutan acak** (*random forest*): banyak pohon yang memberi suara.
- **Uji silang**: melatih mesin pada sebagian titik, lalu mengujinya pada titik yang tidak ikut dilatih.

- **Kepentingan fitur**: seberapa sering sebuah fitur dipakai pohon untuk memisahkan kelas.

Di bab ini, bahan belajar mesin adalah **dua belas fitur** per piksel, yang dibuat dari riwayat NDVI, NBR, dan SWIR1. Itu pekerjaan Skrip 26.12.

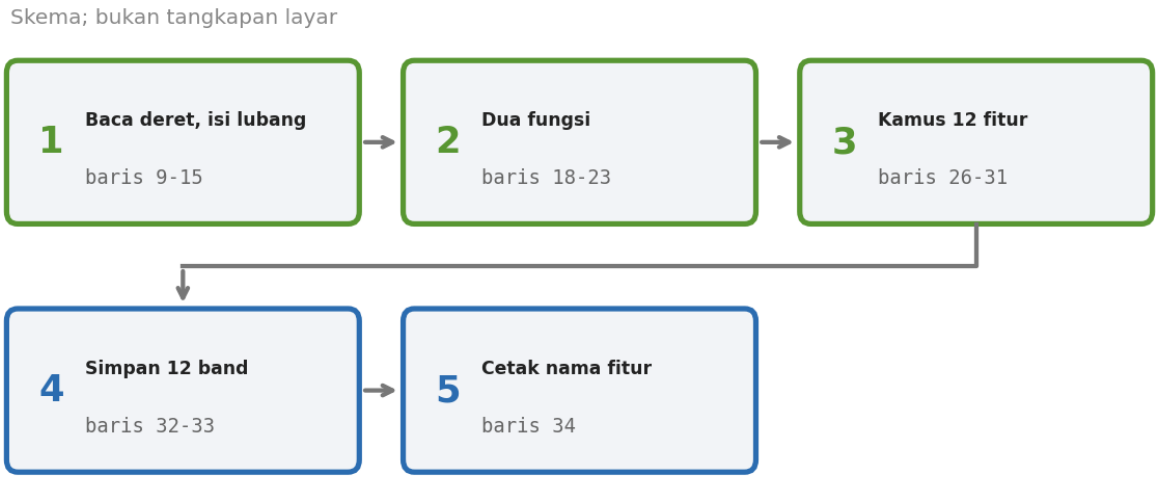
Skrip 26.12: Menyusun dua belas fitur

Berkas `6_1_fitur_deret.py`. Di dalam berkasnya tertulis "SKRIP 6.1".

Tujuan. Menyusun 12 fitur ringkas per piksel dari deret NDVI, NBR, dan SWIR1 yang sudah ditambal, sebagai bahan machine learning.

Bahan. Citra mentah lewat `m3_umum.py`. Skrip ini tidak memakai QGIS dan tidak memerlukan skrip lain. Jalankan dari Konsol atau terminal.

Ilustrasi 26.22: Alur Skrip 26.12



Skrip utuh:

```python
# -*- coding: utf-8 -*-
# SKRIP 6.1: Bangun tumpukan fitur (12 band) dari deret waktu untuk machine learning
# Penulis: Badar Mubarok Yogaswara
# Fitur = angka ringkas yang menggambarkan "perilaku" tiap piksel sepanjang waktu. Hasil: hasil/fitur_deret.tif
import os
import numpy as np
import m3_umum as U

tgl, refl, scl, gt, prj = U.baca_deret()
hari = np.array([(t - tgl[0]).days for t in tgl], dtype="float64")
baik = U.bersih(scl)
ndvi = U.isi_waktu(U.nilai_ndvi(refl), baik, hari)           # NDVI tanpa lubang awan
nbr = U.isi_waktu(U.nilai_nbr(refl), baik, hari)
b11 = U.isi_waktu(refl[:, U.BAND["B11"]], baik, hari)
hc = hari - hari.mean()


def tren_tahun(x):
    return (hc[:, None, None] * (x - x.mean(0))).sum(0) / (hc ** 2).sum() * 365.0


def awal_akhir(x):
```

```
    return np.median(x[-4:], axis=0) - np.median(x[:4], axis=0)      # median 4 tanggal terakhir dikurangi 4
tanggal pertama


fitur = {
    "ndvi_rata": ndvi.mean(0), "ndvi_min": ndvi.min(0), "ndvi_maks": ndvi.max(0), "ndvi_simpang": ndvi.std(0),
    "ndvi_tren": tren_tahun(ndvi), "ndvi_turun_maks": np.diff(ndvi, axis=0).min(0), "ndvi_akhir_awal":
awal_akhir(ndvi),
    "nbr_rata": nbr.mean(0), "nbr_min": nbr.min(0), "nbr_turun_maks": np.diff(nbr, axis=0).min(0),
    "nbr_akhir_awal": awal_akhir(nbr), "swir1_rata": b11.mean(0),
}
nama = list(fitur)
U.tulis_tif(os.path.join(U.HASIL, "fitur_deret.tif"), np.array([fitur[n] for n in nama]).astype("float32"), gt,
prj, deskripsi=nama)
print("Fitur (%d band): %s" % (len(nama), ", ".join(nama)))
```

Penjelasan baris demi baris.

Baris	Tulisan	Artinya
5-7	`import os, import numpy as np, import m3_umum as U`	Perkakas.
9-11	`tgl, refl, scl, gt, prj = ..., hari = ..., baik = ...`	Baca deret, hitung hari ke-berapa tiap tanggal, dan penanda bersih.
12	`ndvi = U.isi_waktu(U.nilai_ndvi(refl), baik, hari)`	Dua fungsi bersarang: hitung NDVI, lalu tambal lubang awannya dengan fungsi kotak perkakas. Ini hampir sama dengan fungsi `isi` di Skrip 26.6.
13	`nbr = U.isi_waktu(U.nilai_nbr(refl), baik, hari)`	Hal yang sama untuk NBR.
14	`b11 = U.isi_waktu(refl[:, U.BAND["B11"]], baik, hari)`	Band B11 (SWIR1) diambil langsung dari larik, lalu ditambal. `U.BAND["B11"]` mencari nomor band dari kamus.
15	`hc = hari - hari.mean()`	Hari yang dipusatkan, seperti di Skrip 26.7.
18-19	`def tren_tahun(x):`	Fungsi tren per tahun. Rumusnya sama dengan Skrip 26.7.
22-23	`def awal_akhir(x):`	Median empat tanggal terakhir dikurangi median empat tanggal pertama. `x[-4:]` berarti empat terakhir, `x[:4]` empat pertama.
26-31	`fitur = {...}`	**Kamus** dua belas fitur. Kuncinya nama, isinya larik hasil hitungan. `ndvi.mean(0)` adalah rata-rata sepanjang waktu, dan seterusnya. `np.diff(ndvi, axis=0).min(0)` mencari penurunan terbesar antara dua tanggal berurutan.
32	`nama = list(fitur)`	Ambil **kunci** kamus sebagai daftar nama, dalam urutan penulisan.
33	`U.tulis_tif(..., np.array([fitur[n] for n in nama]).astype("float32"), gt, prj, deskripsi=nama)`	Susun dua belas larik menjadi satu larik dua belas band, dengan urutan sama dengan `nama`. Simpan ke `fitur_deret.tif`.

Baris	Tulisan	Artinya
34	`print("Fitur (%d band): %s" % (len(nama), ", ".join(nama)))`	Cetak jumlah dan nama fitur. `", ".join(nama)` menyambung daftar dengan koma.

Arti dua belas fitur itu:

Fitur	Arti
`ndvi_rata`, `ndvi_min`, `ndvi_maks`, `ndvi_simpang`	Rata-rata, terendah, tertinggi, dan simpangan baku NDVI.
`ndvi_tren`	Perubahan NDVI per tahun.
`ndvi_turun_maks`	Penurunan NDVI terbesar antara dua tanggal berurutan.
`ndvi_akhir_awal`	NDVI akhir dikurangi NDVI awal.
`nbr_rata`, `nbr_min`	Rata-rata dan terendah NBR.
`nbr_turun_maks`, `nbr_akhir_awal`	Penurunan NBR terbesar, serta NBR akhir dikurangi awal.
`swir1_rata`	Rata-rata band SWIR1.

Hasil. Layar menampilkan (berdasarkan uji pada QGIS 4.0.2):

```
HASIL
Fitur (12 band): ndvi_rata, ndvi_min, ndvi_maks, ndvi_simpang, ndvi_tren, ndvi_turun_maks, ndvi_akhir_awal,
nbr_rata, nbr_min, nbr_turun_maks, nbr_akhir_awal, swir1_rata
```

Berkas `fitur_deret.tif` kini berisi 12 band. Urutan nama di layar adalah urutan band.

Bagian yang boleh diubah.

- **Baris 26-31, isi kamus.** Tambah fitur baru dengan satu entri. Saya menambahkan `"ndvi_median": np.median(ndvi, axis=0)` di **akhir** kamus (setelah `swir1_rata`). Layar menampilkan `Fitur (13 band): ... swir1_rata, ndvi_median`. Skrip 26.13 tetap berjalan: `Titik latih: 200, fitur: 13`.
- **Baris 23, angka 4.** Jumlah tanggal di awal dan akhir yang dibandingkan.

Jebakan.

- **Urutan fitur tidak boleh diubah di tengah.** Skrip 26.14 memakai **tujuh fitur pertama** (`range(7)`), yaitu fitur NDVI. Bila Anda menyisipkan fitur di depan, "tujuh fitur NDVI" itu berubah isi tanpa pesan. Tambahkan fitur baru di **akhir**.
- **Rumus tren sedikit berbeda dari Skrip 26.7.** Di sini memakai `x.mean()` di dalam fungsi, sedangkan di Skrip 26.7 memakai larik `rata` yang sudah dihitung. Penulisannya tidak persis sama, jadi periksa bila Anda menyalinnya.
- **Penambalan per piksel lambat**, seperti Skrip 26.6. Tiga kali pemanggilan `U.isi_waktu` menambah waktu.

Latihan variasi.

1. Tambahkan fitur `ndvi_median` (median NDVI).
2. Pastikan jumlah band menjadi 13.

Contoh jawaban, hanya baris 30:

```python
    "nbr_akhir_awal": awal_akhir(nbr), "swir1_rata": b11.mean(0), "ndvi_median": np.median(ndvi, axis=0),
```

Layar menampilkan `Fitur (13 band): ndvi_rata, ndvi_min, ... swir1_rata, ndvi_median`.

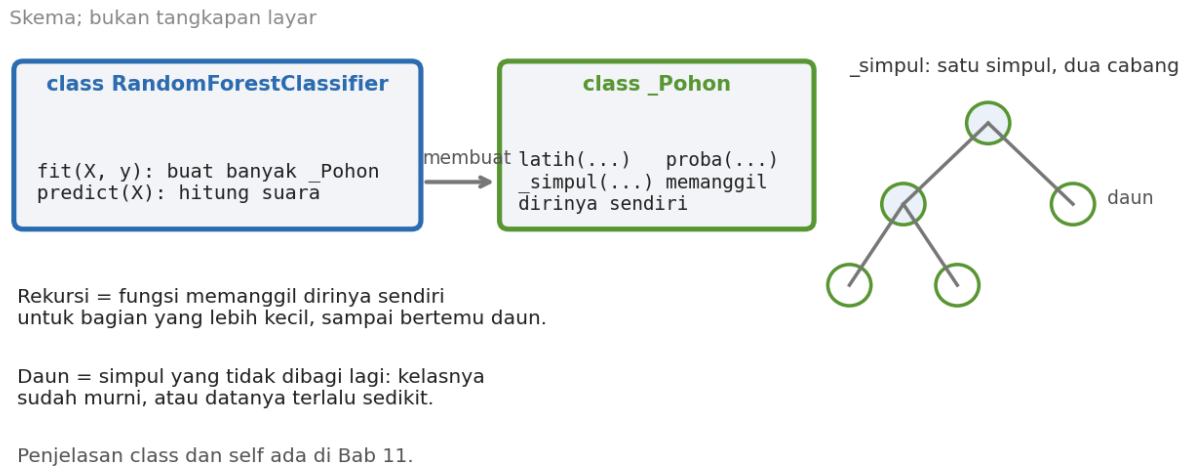
> **Pola yang bisa dipakai ulang:** Fitur disusun dengan pola **kamus nama lalu larik**, lalu satu baris menyusunnya menjadi berkas bernama band. Untuk fitur lain, tambahkan entri di kamus (baris 26-31). Baris 32-34 tidak perlu disentuh. Pegang satu aturan: **fitur baru di akhir**.

Kotak perkakas kedua: hutan_mini.py

Skrip berikutnya melatih hutan acak. Pustaka umumnya adalah **scikit-learn**, tetapi tidak semua QGIS memasangnya. Di QGIS 4.0.2 milik saya, `import sklearn` menghasilkan `ModuleNotFoundError: No module named 'sklearn'`. Karena itu penulis membuat `hutan_mini.py`: hutan acak ringkas yang hanya memakai NumPy.

Berkas ini memuat **kelas** buatan sendiri. Kelas adalah cetakan benda: ia punya data (disebut atribut) dan pekerjaan (disebut metode) sekaligus. Ada dua kelas di sini. `_Pohon` mewakili satu pohon keputusan. `RandomForestClassifier` mewakili hutan yang memegang banyak pohon. Nama terakhir sengaja sama dengan nama di scikit-learn, sehingga skrip lain bisa berganti pustaka tanpa mengubah pemanggilannya. Cara membuat kelas, arti `self`, dan `__init__` dibahas di Bab 11.

Ilustrasi 26.23: Dua kelas di hutan_mini



Bagian yang paling menarik adalah **rekursi**: fungsi yang memanggil dirinya sendiri. Metode `_simpul` membagi data menjadi dua kelompok, lalu memanggil `_simpul` lagi untuk tiap kelompok. Pembagian berhenti di **daun**, yaitu kelompok yang kelasnya sudah murni atau datanya terlalu sedikit. Bayangkan

Anda membagi tumpukan kartu menjadi dua, lalu membagi tiap tumpukan menjadi dua lagi, sampai tiap tumpukan berisi satu jenis kartu.

Tujuan. Menyediakan hutan acak ringkas bagi Skrip 26.13 dan 26.14, sekaligus membuka "kotak hitam" supaya Anda paham cara kerjanya.

Bahan. Berkas ini tidak dijalankan sendiri. Skrip 26.13 dan 26.14 mengimpornya bila `sklearn` tidak ada. Berkas harus satu folder dengan mereka. Ia hanya memakai `numpy`.

Skrip utuh:

PYTHON
```
# -*- coding: utf-8 -*-
"""hutan_mini.py: hutan acak (random forest) ringkas dengan NumPy saja. Penulis: Badar Mubarok Yogaswara
Tujuan: (1) membuka "kotak hitam" supaya Anda paham cara kerjanya, (2) tetap bisa jalan di QGIS yang belum
memasang scikit-learn.
Antarmukanya sengaja dibuat mirip scikit-learn: RandomForestClassifier(...).fit(X,
y), .predict(X), .feature_importances_.
Ini alat belajar; untuk pekerjaan besar pakai scikit-learn."""
import numpy as np


class _Pohon:
    def __init__(self, maks_fitur, kedalaman, daun_min, rng):
        self.mf, self.kd, self.dm, self.rng = maks_fitur, kedalaman, daun_min, rng
        self.fitur, self.ambang, self.kiri, self.kanan, self.nilai = [], [], [], [], []
        self.penting = None

    def _gini(self, hitung):
        n = hitung.sum()
        return 1.0 - ((hitung / n) ** 2).sum() if n else 0.0

    def _simpul(self, X, y, K, kdl):
        hitung = np.bincount(y, minlength=K).astype(float)
        idx = len(self.fitur)
        self.fitur.append(-1); self.ambang.append(0.0); self.kiri.append(-1); self.kanan.append(-1);
self.nilai.append(hitung / hitung.sum())
        if kdl == self.kd or len(y) < 2 * self.dm or (hitung > 0).sum() == 1:
            return idx
        terbaik = (0.0, None, None)
        g0 = self._gini(hitung)
        n = len(y)
        for f in self.rng.choice(X.shape[1], size=self.mf, replace=False):
            urut = np.argsort(X[:, f], kind="stable")
            xs, ys = X[urut, f], y[urut]
            kiri = np.cumsum(np.eye(K)[ys], axis=0)             # jumlah kelas di sisi kiri untuk tiap titik
potong
            kanan = hitung - kiri
            nk = np.arange(1, n + 1)[:, None]
            nr = n - nk
            gk = 1 - ((kiri / nk) ** 2).sum(1)
            gr = np.where(nr[:, 0] > 0, 1 - ((kanan / np.maximum(nr, 1)) ** 2).sum(1), 0)
            skor = g0 - (nk[:, 0] * gk + nr[:, 0] * gr) / n      # penurunan ketidakmurnian
            sah = (xs[:-1] < xs[1:]) & (nk[:-1, 0] >= self.dm) & (nr[:-1, 0] >= self.dm)
            if not sah.any():
                continue
            s = np.where(sah, skor[:-1], -1)
            k = int(s.argmax())
            if s[k] > terbaik[0]:
                terbaik = (s[k], f, (xs[k] + xs[k + 1]) / 2)
        if terbaik[1] is None:
            return idx
        gain, f, t = terbaik
        self.penting[f] += gain * n
```

```
        m = X[:, f] <= t
        self.fitur[idx], self.ambang[idx] = f, t
        self.kiri[idx] = self._simpul(X[m], y[m], K, kdl + 1)
        self.kanan[idx] = self._simpul(X[~m], y[~m], K, kdl + 1)
        return idx

    def latih(self, X, y, K):
        self.penting = np.zeros(X.shape[1])
        self._simpul(X, y, K, 0)
        self.fitur, self.ambang = np.array(self.fitur), np.array(self.ambang)
        self.kiri, self.kanan, self.nilai = np.array(self.kiri), np.array(self.kanan), np.array(self.nilai)

    def proba(self, X):
        simpul = np.zeros(len(X), dtype=int)
        while True:
            daun = self.fitur[simpul] < 0
            if daun.all():
                break
            f = self.fitur[simpul]
            ke_kiri = X[np.arange(len(X)), np.maximum(f, 0)] <= self.ambang[simpul]
            baru = np.where(ke_kiri, self.kiri[simpul], self.kanan[simpul])
            simpul = np.where(daun, simpul, baru)
        return self.nilai[simpul]


class RandomForestClassifier:
    def __init__(self, n_estimators=100, max_features="sqrt", max_depth=None, min_samples_leaf=1,
random_state=0):
        self.n_estimators, self.max_features, self.max_depth = n_estimators, max_features, max_depth
        self.min_samples_leaf, self.random_state = min_samples_leaf, random_state

    def fit(self, X, y):
        X = np.asarray(X, dtype=float)
        self.classes_, yk = np.unique(y, return_inverse=True)
        K, (n, d) = len(self.classes_), X.shape
        mf = max(1, int(np.sqrt(d))) if self.max_features == "sqrt" else (d if self.max_features is None else
int(self.max_features))
        rng = np.random.default_rng(self.random_state)
        self.pohon_, imp = [], np.zeros(d)
        for _ in range(self.n_estimators):
            b = rng.integers(0, n, n)                          # contoh bootstrap: ambil n baris dengan
pengembalian
            p = _Pohon(mf, self.max_depth or 10 ** 6, self.min_samples_leaf, rng)
            p.latih(X[b], yk[b], K)
            self.pohon_.append(p)
            imp += p.penting
        self.feature_importances_ = imp / imp.sum() if imp.sum() else imp
        return self

    def predict_proba(self, X):
        X = np.asarray(X, dtype=float)
        return np.mean([p.proba(X) for p in self.pohon_], axis=0)

    def predict(self, X):
        return self.classes_[self.predict_proba(X).argmax(1)]
```

Penjelasan baris demi baris. Berkas ini padat, jadi dijelaskan per blok.

Baris	Tulisan	Artinya
1-6	docstring dan `import numpy as np`	Keterangan tujuan berkas, dan perkakas NumPy.
9	`class _Pohon:`	Mulai kelas pohon. Awalan garis bawah menandakan kelas ini untuk dipakai di dalam berkas.

Baris	Tulisan	Artinya
10-13	`def __init__(self, maks_fitur, kedalaman, daun_min, rng):`	**Pembuat**: dijalankan saat sebuah pohon dibuat. Ia menyimpan aturan (jumlah fitur yang dicoba, kedalaman maksimum, daun minimum, pengacak) dan menyiapkan lima daftar kosong yang kelak menyimpan isi pohon.
15-17	`def _gini(self, hitung):`	Mengukur **ketidakmurnian** sebuah kelompok. Nilai 0 berarti semua anggota sekelas. Makin dekat ke 1, makin campur.
19	`def _simpul(self, X, y, K, kdl):`	Metode inti: membangun satu simpul pada kedalaman `kdl`.
20-22	`hitung = np.bincount(...)` sampai `self.nilai.append(...)`	Hitung berapa anggota tiap kelas, nomori simpul ini, dan catat simpul baru ke daftar. Baris 22 menaruh beberapa pernyataan satu baris dengan titik koma.
23-24	`if kdl == self.kd or ...: return idx`	Berhenti dan jadikan simpul ini **daun** bila sudah mencapai kedalaman maksimum, datanya terlalu sedikit, atau kelasnya sudah murni.
25-27	`terbaik = (0.0, None, None)` dan dua baris	Siapkan catatan "pembagian terbaik sejauh ini", ketidakmurnian awal, dan jumlah data.
28-44	`for f in self.rng.choice(...):`	Coba beberapa fitur yang dipilih acak. Untuk tiap fitur: urutkan data (baris 29-30), hitung kelas di sisi kiri tiap titik potong (baris 31-32), hitung ketidakmurnian kedua sisi (baris 33-36), dan nilai tiap potongan (baris 37). Baris 38-44 memilih potongan terbaik yang sah.
45-46	`if terbaik[1] is None: return idx`	Bila tidak ada potongan yang berguna, jadikan daun.
47-50	`gain, f, t = terbaik` sampai `self.fitur[idx], ... = f, t`	Pakai potongan terbaik: catat fitur dan ambang di simpul ini, dan tambah kepentingan fitur. `m` memisahkan data menjadi dua kelompok.
51-52	`self.kiri[idx] = self._simpul(...)` dan baris kanan	**Rekursi**: panggil `_simpul` untuk kelompok kiri dan kelompok kanan, pada kedalaman satu tingkat lebih dalam.
53	`return idx`	Kembalikan nomor simpul ini.
55-59	`def latih(self, X, y, K):`	Metode umum: siapkan penampung kepentingan, panggil `_simpul` untuk akar, lalu ubah daftar menjadi larik NumPy.
61-71	`def proba(self, X):`	Menebak kelas untuk banyak titik sekaligus. Baris 63-70 adalah putaran `while True` yang menelusuri pohon dari akar ke daun, dan berhenti (`break`, baris 66) saat semua titik sudah di daun.
74	`class RandomForestClassifier:`	Kelas hutan. Namanya sama dengan di scikit-learn.
75-77	`def __init__(self, n_estimators=100, ...)`	Pembuat hutan. Semua pengaturan punya nilai bawaan. `n_estimators` adalah jumlah pohon.

Baris	Tulisan	Artinya
79-93	`def fit(self, X, y):`	**Melatih** hutan. Baris 81 mengubah label menjadi nomor 0, 1, 2, dan seterusnya. Baris 83 menentukan berapa fitur dicoba tiap simpul (akar jumlah fitur). Baris 86-91 membuat tiap pohon dari contoh `bootstrap` (baris 87): `n` baris diambil acak **dengan pengembalian**, jadi sebagian baris muncul dua kali dan sebagian tidak muncul. Baris 92 menghitung kepentingan fitur rata-rata. Baris 93 mengembalikan hutan itu sendiri, sehingga bisa ditulis berantai `.fit(...)`.
95-97	`def predict_proba(self, X):`	Rata-rata tebakan semua pohon.
99-100	`def predict(self, X):`	Ambil kelas dengan peluang terbesar. Baris 100 adalah "suara terbanyak".

Hasil. Anda bisa mencobanya langsung dengan data mainan (bukan bagian berkas): dua fitur, enam titik.

```
PYTHON
from hutan_mini import RandomForestClassifier
X = [[0.86, 0.02], [0.84, 0.03], [0.88, 0.02], [0.52, 0.15], [0.48, 0.17], [0.55, 0.14]]
y = [1, 1, 1, 2, 2, 2]          # 1 = hutan, 2 = sawah
m = RandomForestClassifier(n_estimators=10, random_state=1).fit(X, y)
print("kelas dikenal:", m.classes_)
print("prediksi      :", m.predict([[0.85, 0.02], [0.50, 0.16]]))
print("kepentingan   :", m.feature_importances_.round(2))
```

Layar menampilkan (berdasarkan uji pada QGIS 4.0.2):

```
HASIL
kelas dikenal: [1 2]
prediksi      : [1 2]
kepentingan   : [0.29 0.71]
```

Titik pertama ditebak hutan, titik kedua sawah. Fitur kedua (simpangan baku) lebih penting daripada fitur pertama pada data mainan ini.

Bagian yang boleh diubah.

- **Baris 75, pengaturan hutan.** `n_estimators` (jumlah pohon), `max_depth` (kedalaman maksimum), `min_samples_leaf` (anggota minimum tiap daun), `random_state` (benih). Skrip pemanggil yang menentukannya.
- Baris 83, `"sqrt"`. Jumlah fitur yang dicoba tiap simpul.

Jebakan.

- **Hasil belum tentu sama dengan scikit-learn.** Meskipun benih sama, cara pengacakannya berbeda. Saya tidak bisa mengujinya karena `sklearn` tidak terpasang di QGIS saya. Bila Anda memasangnya, angka di Skrip 26.13 dan 26.14 bisa berbeda dari yang tertera di bab ini [CEK].
- **Lambat untuk data besar.** Ini alat belajar. Untuk pekerjaan besar, pakai scikit-learn.

- **Label boleh teks.** Saya mencoba label `"hutan"` dan `"sawah"` dengan `max_depth=1`: `classes_` menjadi `['hutan' 'sawah']`, tebakannya benar, dan pohon pertama hanya punya tiga simpul (satu akar, dua daun).

Latihan variasi.

1. Latih dengan `max_depth=1` dan label teks.
2. Hitung simpul pohon pertama.

Contoh jawaban:

```python
y = ["hutan", "hutan", "hutan", "sawah", "sawah", "sawah"]
m = RandomForestClassifier(n_estimators=10, max_depth=1, random_state=1).fit(X, y)
print(m.classes_, m.predict([[0.85, 0.02], [0.50, 0.16]]), len(m.pohon_[0].fitur))
```

Layar menampilkan `['hutan' 'sawah'] ['hutan' 'sawah'] 3`.

> **Pola yang bisa dipakai ulang:** Untuk membuat perkakas yang bisa menggantikan pustaka besar, tiru **antarmuka**-nya: nama kelas, `fit`, `predict`. Skrip pemanggil lalu memakai pola `try ... except ImportError` untuk beralih otomatis, seperti di Skrip 26.13.

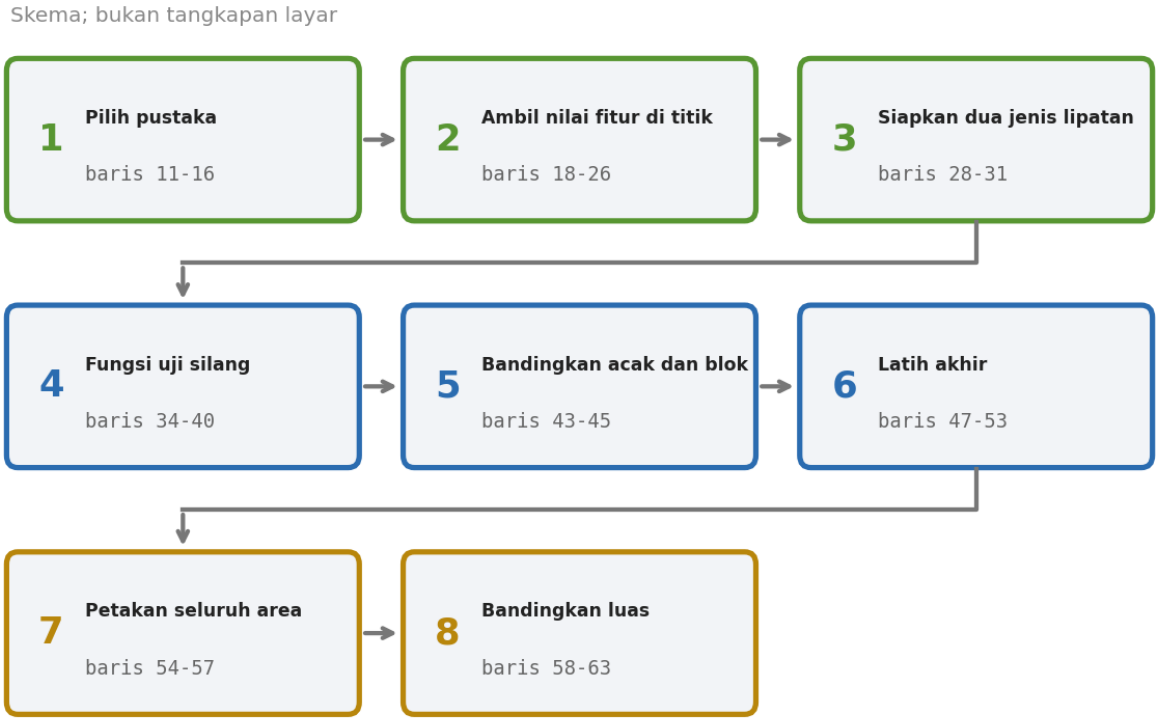
Skrip 26.13: Melatih hutan acak dan memetakan seluruh area

Berkas `6_2_latih_klasifikasi.py`. Di dalam berkasnya tertulis "SKRIP 6.2".

Tujuan. Melatih hutan acak dari titik latih, membandingkan uji silang acak dengan uji silang blok ruang, lalu memetakan seluruh area.

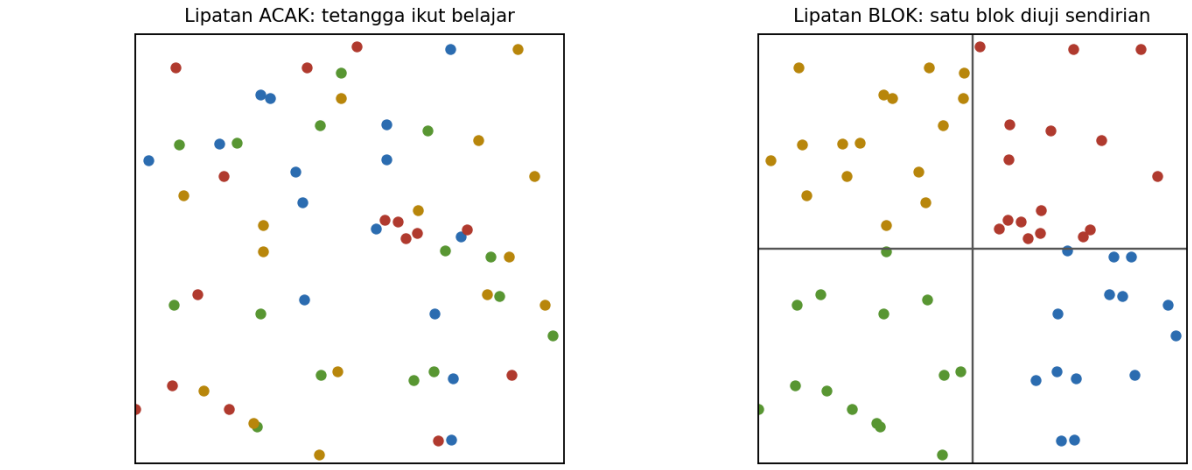
Bahan. `fitur_deret.tif` dari Skrip 26.12, `vektor/Titik_Latih.gpkg` (kolom `KELAS`), `acuan/Acuan_Dinamika.tif`, dan `m3_umum.py`. Bila `sklearn` tidak ada, skrip memakai `hutan_mini.py`. Jalankan dari Konsol atau terminal.

Ilustrasi 26.24: Alur Skrip 26.13



Uji silang perlu dijelaskan sebelum skripnya. Titik latih dibagi menjadi empat kelompok (**lipatan**). Mesin dilatih pada tiga lipatan, lalu diuji pada satu lipatan yang tersisa. Proses diulang empat kali. Pada **lipatan acak**, titik dibagi sembarang, sehingga tetangga dekat bisa ada di kedua sisi. Pada **lipatan blok**, titik dibagi menurut lokasi (empat blok 200 meter), sehingga blok yang diuji benar-benar "asing" bagi mesin.

Ilustrasi 26.25: Dua cara membagi lipatan



Skema; bukan tangkapan layar; warna = nomor lipatan; letak titik contoh

Skrip utuh:

```python
# -*- coding: utf-8 -*-
# SKRIP 6.2: Latih hutan acak dari titik latih, uji silang acak vs uji silang blok ruang, lalu petakan seluruh area
# Penulis: Badar Mubarok Yogaswara
# Syarat: Skrip 6.1 sudah dijalankan. Memakai scikit-learn bila terpasang; bila tidak, memakai hutan_mini.py.
import os
import numpy as np
import processing
from qgis.core import QgsVectorLayer
import m3_umum as U

try:
    from sklearn.ensemble import RandomForestClassifier
    print("Memakai scikit-learn")
except ImportError:
    from hutan_mini import RandomForestClassifier
    print("scikit-learn tidak ada; memakai hutan_mini.py")

fitur_tif = os.path.join(U.HASIL, "fitur_deret.tif")
titik = QgsVectorLayer(os.path.join(U.PAKET, "vektor", "Titik_Latih.gpkg"), "latih")
# ambil nilai 12 fitur pada tiap titik latih (Sample raster values)
s = processing.run("native:rastersampling", {"INPUT": titik, "RASTERCOPY": fitur_tif, "COLUMN_PREFIX": "f",
"OUTPUT": "TEMPORARY_OUTPUT"})["OUTPUT"]
nama_f = [fl.name() for fl in s.fields() if fl.name().startswith("f") and fl.name() != "fid"]
X = np.array([[f[n] for n in nama_f] for f in s.getFeatures()], dtype=float)
y = np.array([f["KELAS"] for f in s.getFeatures()])
xy = np.array([[f.geometry().asPoint().x(), f.geometry().asPoint().y()] for f in s.getFeatures()])
print("Titik latih: %d, fitur: %d, kelas: %s" % (len(y), X.shape[1], dict(zip(*np.unique(y,
return_counts=True)))))

# --- uji silang 1: acak 4 lipatan; uji silang 2: lipatan = blok ruang 200 m x 200 m (4 blok, area 400 m)
rng = np.random.default_rng(5)
lipat_acak = rng.permutation(len(y)) % 4
lipat_blok = (xy[:, 0] >= 312200).astype(int) + 2 * (xy[:, 1] >= 9996200).astype(int)


def uji_silang(lipat):
    benar = []
    for k in range(4):
        tr, te = lipat != k, lipat == k
        m = RandomForestClassifier(n_estimators=100, random_state=1).fit(X[tr], y[tr])
        benar.append((m.predict(X[te]) == y[te]).mean())
    return np.array(benar)


a, b = uji_silang(lipat_acak), uji_silang(lipat_blok)
print("Akurasi uji silang ACAK : %s rata-rata %.3f" % (np.round(a, 2), a.mean()))
print("Akurasi uji silang BLOK : %s rata-rata %.3f" % (np.round(b, 2), b.mean()))

# --- latih akhir dan petakan
model = RandomForestClassifier(n_estimators=200, random_state=1).fit(X, y)
urut = np.argsort(model.feature_importances_)[::-1]
from osgeo import gdal
ds = gdal.Open(fitur_tif)
nama_fitur = [ds.GetRasterBand(i + 1).GetDescription() for i in range(ds.RasterCount)]
print("Kepentingan fitur (5 teratas):", ", ".join("%s %.2f" % (nama_fitur[i], model.feature_importances_[i])
for i in urut[:5]))
st, gt, prj = U.baca_tif(fitur_tif)                  # (12, H, W)
nb, H, W = st.shape
peta = model.predict(st.reshape(nb, -1).T).reshape(H, W).astype("uint8")
U.tulis_tif(os.path.join(U.HASIL, "peta_rf.tif"), peta, gt, prj, nodata=0)
din, _, _ = U.baca_tif(os.path.join(U.PAKET, "acuan", "Acuan_Dinamika.tif"))
```

```
nama = {1: "Hutan alam", 2: "Hutan tanaman", 3: "Pertanian", 4: "Air", 5: "Terbuka", 6: "Deforestasi", 7:
"Terbakar", 8: "Panen HTI"}
print("Luas hasil peta (ha) / luas acuan (ha):")
for k in range(1, 9):
    print("  %-14s %6.2f / %6.2f" % (nama[k], (peta == k).sum() * 0.01, (din == k).sum() * 0.01))
print("Kesesuaian peta dengan acuan di SEMUA piksel (bukan uji mandiri): %.3f" % (peta == din).mean())
```

Penjelasan baris demi baris.

Baris	Tulisan	Artinya
5-9	`import os` sampai `import m3_umum as U`	Perkakas.
11-16	`try:` … `except ImportError:`	**Coba dulu, cadangan kemudian.** Bila `from sklearn.ensemble import RandomForestClassifier` berhasil, pakai scikit-learn dan cetak pesannya. Bila gagal dengan `ImportError`, Python lompat ke blok `except` dan memakai `hutan_mini` (Bab 10).
18-19	`fitur_tif = ...` dan `titik = QgsVectorLayer(...)`	Jalur tumpukan fitur, dan layer titik latih.
21	`s = processing.run("native:rastersampling", {...})["OUTPUT"]`	Alat Sample raster values: mengambil nilai tiap band tumpukan di tiap titik. `COLUMN_PREFIX: "f"` menamai kolom baru `f1`, `f2`, dan seterusnya. Hasilnya layer sementara berisi titik dengan kolom nilai.
22	`nama_f = [fl.name() for fl in s.fields() if fl.name().startswith("f") and fl.name() != "fid"]`	Kumpulkan nama kolom yang berawalan huruf `f`, kecuali `fid` (nomor baris bawaan). Itulah `f1` sampai `f12`.
23	`X = np.array([[f[n] for n in nama_f] for f in s.getFeatures()], dtype=float)`	**Comprehension bersarang**: untuk tiap titik, daftar 12 nilai fitur. Hasilnya larik 200 baris kali 12 kolom. `X` adalah tabel fitur.
24	`y = np.array([f["KELAS"] for f in s.getFeatures()])`	`y` adalah daftar kelas tiap titik.
25	`xy = np.array([[... x(), ... y()] for f in ...])`	Koordinat tiap titik.
26	`print("Titik latih: %d, fitur: %d, kelas: %s" % (...))`	Cetak jumlah titik, jumlah fitur, dan banyak titik tiap kelas. `np.unique(..., return_counts=True)` memberi kelas dan hitungannya, `zip(*...)` menyandingkannya, `dict(...)` membuat kamus.
29	`rng = np.random.default_rng(5)`	Pengacak dengan benih 5.
30	`lipat_acak = rng.permutation(len(y)) % 4`	Acak urutan 200 titik, lalu bagi dengan 4 dan ambil sisanya (0 sampai 3). Hasilnya nomor lipatan tiap titik.
31	`lipat_blok = (xy[:, 0] >= 312200).astype(int) + 2 * (xy[:, 1] >= 9996200).astype(int)`	Nomor lipatan menurut lokasi: 0 untuk barat daya, 1 tenggara, 2 barat laut, 3 timur laut. Batas 312200 dan 9996200 adalah garis tengah area contoh.

Baris	Tulisan	Artinya
34-40	`def uji_silang(lipat):`	Fungsi uji silang, dengan satu masukan: nomor lipatan tiap titik.
35-36	`benar = []` dan `for k in range(4):`	Daftar akurasi, lalu empat putaran.
37	`tr, te = lipat != k, lipat == k`	`tr` (latih) berisi semua titik bukan lipatan `k`. `te` (uji) berisi titik lipatan `k`.
38	`m = RandomForestClassifier(n_estimators=100, random_state=1).fit(X[tr], y[tr])`	Buat hutan 100 pohon dan latih dengan `fit`. Dua panggilan berantai dalam satu baris.
39	`benar.append((m.predict(X[te]) == y[te]).mean())`	Tebak titik uji, bandingkan dengan kelas sebenarnya, dan catat pecahan yang benar.
40	`return np.array(benar)`	Kembalikan empat akurasi.
43-45	`a, b = uji_silang(lipat_acak), uji_silang(lipat_blok)` dan dua `print`	Jalankan uji dengan dua jenis lipatan, dan cetak akurasi tiap lipatan serta rata-ratanya.
48	`model = RandomForestClassifier(n_estimators=200, random_state=1).fit(X, y)`	Latih akhir pada **seluruh** titik, dengan 200 pohon.
49	`urut = np.argsort(model.feature_importances_)[::-1]`	Urutkan nomor fitur dari yang paling penting. `[::-1]` membalik urutan.
50-52	`from osgeo import gdal, ds = ..., nama_fitur = [...]`	Baca nama band dari tumpukan fitur, yaitu nama yang ditulis Skrip 26.12.
53	`print("Kepentingan fitur (5 teratas): ...")`	Cetak lima fitur terpenting dan skornya.
54-55	`st, gt, prj = U.baca_tif(fitur_tif)` dan `nb, H, W = st.shape`	Baca seluruh tumpukan fitur: 12 band kali 40 kali 40.
56	`peta = model.predict(st.reshape(nb, -1).T).reshape(H, W).astype("uint8")`	`st.reshape(nb, -1)` meratakan tiap band menjadi satu baris panjang, `.T` menukar baris dan kolom sehingga tiap baris adalah satu piksel dengan 12 fitur. `predict` menebak kelas semua piksel. `reshape(H, W)` mengembalikan bentuk peta.
57	`U.tulis_tif(..."peta_rf.tif", peta, gt, prj, nodata=0)`	Simpan peta kelas. Byte cukup untuk kelas 1 sampai 8.
58-59	`din, _, _ = ...` dan `nama = {...}`	Peta acuan dan nama kelas.
60-62	`print("Luas hasil peta (ha) / luas acuan (ha):")` dan putaran	Cetak luas tiap kelas pada peta dan pada acuan, piksel kali 0,01 hektare.

Baris	Tulisan	Artinya
63	`print("Kesesuaian peta dengan acuan di SEMUA piksel (bukan uji mandiri): %.3f" % (peta == din).mean())`	Pecahan piksel yang sama dengan acuan. Skrip sendiri menandainya "bukan uji mandiri".

Hasil. Layar menampilkan (berdasarkan uji pada QGIS 4.0.2, tanpa scikit-learn, jadi memakai `hutan_mini`):

```
HASIL
scikit-learn tidak ada; memakai hutan_mini.py
Titik latih: 200, fitur: 12, kelas: {np.int64(1): np.int64(25), np.int64(2): np.int64(25), np.int64(3):
np.int64(25), np.int64(4): np.int64(25), np.int64(5): np.int64(25), np.int64(6): np.int64(25), np.int64(7):
np.int64(25), np.int64(8): np.int64(25)}
Akurasi uji silang ACAK : [0.98 1.   1.   1.  ] rata-rata 0.995
Akurasi uji silang BLOK : [0.5  0.89 0.63 0.44] rata-rata 0.617
Kepentingan fitur (5 teratas): nbr_rata 0.13, ndvi_rata 0.12, ndvi_maks 0.12, swir1_rata 0.12, nbr_min 0.11
Luas hasil peta (ha) / luas acuan (ha):
  Hutan alam       7.99 /   8.00
  Hutan tanaman    0.96 /   0.96
  Pertanian        1.80 /   1.80
  Air              0.80 /   0.80
  Terbuka          0.43 /   0.43
  Deforestasi      1.30 /   1.30
  Terbakar         1.52 /   1.51
  Panen HTI        1.20 /   1.20
Kesesuaian peta dengan acuan di SEMUA piksel (bukan uji mandiri): 0.999
```

Baris `{np.int64(1): np.int64(25), ...}` adalah cara NumPy versi baru menampilkan angka. Artinya tetap: tiap kelas punya 25 titik.

Perhatikan dua baris akurasi. Pada uji silang **acak**, akurasi 0,995, nyaris sempurna. Pada uji silang **blok**, akurasi turun jauh menjadi 0,617, dan satu blok hanya 0,44. Mesin tampak hebat saat tetangga dekat boleh ikut belajar, tetapi jauh lebih lemah di tempat yang belum pernah ia lihat. **Uji silang acak terlalu optimistis untuk data ruang.** Peta akhir sendiri cocok 99,9 persen dengan acuan, tetapi itu bukan ujian mandiri: titik latih ikut dihitung, dan piksel itu juga bagian dari acuan.

Bagian yang boleh diubah.

- Baris 38, `n_estimators`. Saya mengubahnya menjadi `5`: akurasi blok menjadi `[0.5 0.62 0.88 0.44]` (rata-rata 0,612) dan akurasi acak tetap 0,995. Pohon lebih sedikit, hasil sedikit bergeser.
- **Baris 31, batas blok.** Saya mengganti `312200` menjadi `312100`. Akurasi blok menjadi `[0.29 0.92 1.   0.63]` dengan rata-rata 0,707. Blok yang berbeda memberi angka berbeda, jadi uji blok tergantung cara memotong.
- **Baris 29, benih.** Ganti `5` untuk undian lipatan acak yang lain.
- **Baris 48, 200 pohon.** Lebih banyak pohon, lebih lambat tetapi lebih stabil.

Jebakan.

- **Pustaka bisa berganti tanpa Anda sadari.** Baris pertama keluaran memberi tahu: `scikit-learn tidak ada; memakai hutan_mini.py`. Di komputer yang punya scikit-learn, angka akurasi dan peta bisa berbeda [CEK]. Selalu baca baris itu.

- Kolom fitur diambil dari awalan `f`. Saya mencetak daftar kolom hasil pengambilan nilai: `['fid', 'ID', 'KELAS', 'NAMA_KELAS', 'f1', ..., 'f12']`. Daftar `nama_f` berisi tepat `f1` sampai `f12`. Bila data Anda punya kolom lain berawalan `f` (huruf kecil), kolom itu ikut terambil dan merusak `X`.
- **Angka empat dan batas blok dikunci.** `range(4)`, `% 4`, dan dua batas koordinat harus cocok. Saya mengganti `range(4)` menjadi `range(3)`: layar hanya menampilkan tiga angka akurasi, dan satu lipatan tidak pernah diuji. Skrip tidak mengeluh.
- **Skrip 26.12 harus sudah jalan.** Di folder kosong, layar menampilkan `QgsProcessingException: Unable to execute algorithm` dan `Could not load source layer for RASTERCOPY: ...fitur_deret.tif not found`.
- `from osgeo import gdal` di tengah berkas. Boleh, tetapi biasanya impor ditaruh di atas.
- **Luas 0,01 hektare per piksel dikunci** di baris 62.

Latihan variasi.

1. Kurangi pohon uji silang menjadi 5, lalu bandingkan akurasi blok.
2. Geser batas blok barat-timur menjadi 312100.

Contoh jawaban, hanya baris yang berubah:

```
PYTHON
        m = RandomForestClassifier(n_estimators=5, random_state=1).fit(X[tr], y[tr])
```

```
PYTHON
lipat_blok = (xy[:, 0] >= 312100).astype(int) + 2 * (xy[:, 1] >= 9996200).astype(int)
```

Untuk nomor 1, layar menampilkan `Akurasi uji silang BLOK : [0.5  0.62 0.88 0.44] rata-rata 0.612`. Untuk nomor 2, `Akurasi uji silang BLOK : [0.29 0.92 1.   0.63] rata-rata 0.707`. Tiap variasi dijalankan terpisah.

> **Pola yang bisa dipakai ulang:** Klasifikasi dengan titik latih selalu berbentuk **ambil fitur di titik, bagi lipatan, latih, uji, petakan**. Ganti **layer titik** dan **kolom kelas** (baris 19 dan 24), **tumpukan fitur** (baris 18), dan **cara membagi lipatan** (baris 30-31). Biasakan menguji dengan **lipatan blok**, supaya akurasi tidak terlalu manis.

Kelompok G: Seberapa boleh peta ini dipercaya?

Konsep: ujian dengan soal baru

Validasi itu seperti ujian dengan soal yang belum pernah dilihat murid. Nilai ujian yang soalnya sama dengan bahan belajar tidak membuktikan apa-apa. Anda menarik **sampel acak** titik di peta, memeriksa kebenarannya dengan data acuan yang lebih tepercaya, lalu menyusun **matriks kebingungan**. Dari matriks itu keluar akurasi, dan **koreksi luas**: perkiraan luas yang sudah memperhitungkan kesalahan peta.

Ilustrasi 26.26: Membaca matriks kebingungan

Skema; bukan tangkapan layar

KOLOM = kata acuan (kebenaran)

BARIS = kata peta	Hutan alam	Terbakar
Hutan alam	20	0
Terbakar	10	10

Baca satu baris

Baris Terbakar: peta menyebut 'Terbakar' di 20 titik. Hanya 10 yang benar terbakar.
Akurasi pengguna = 10 / 20 = 0,50

Baca satu kolom

Kolom Hutan alam: 30 titik acuan berupa hutan alam. Peta benar di 20 titik, salah di 10 titik.

Angka diambil dari Peta B pada keluaran Skrip 26.14 (hanya dua kelas ditampilkan).

Lima istilah yang perlu Anda kenal sekarang:

- **Matriks kebingungan**: tabel yang menyilangkan kelas peta (baris) dengan kelas acuan (kolom).
- **Akurasi pengguna (UA)**: dari semua piksel yang peta sebut kelas X, berapa bagian yang benar kelas X. Mengukur kesalahan "terlalu banyak menandai".
- **Akurasi produsen (PA)**: dari semua piksel yang sebenarnya kelas X, berapa bagian yang peta temukan. Mengukur kesalahan "terlewat".
- **Sampel acak berstrata**: sampel dengan jumlah tetap pada tiap kelas peta, supaya kelas langka ikut terwakili.
- **Luas terkoreksi**: luas yang disesuaikan dengan kesalahan peta, disertai rentang ketidakpastian (di sini 95 persen).

Skrip 26.14: Validasi dua peta dengan sampel berstrata

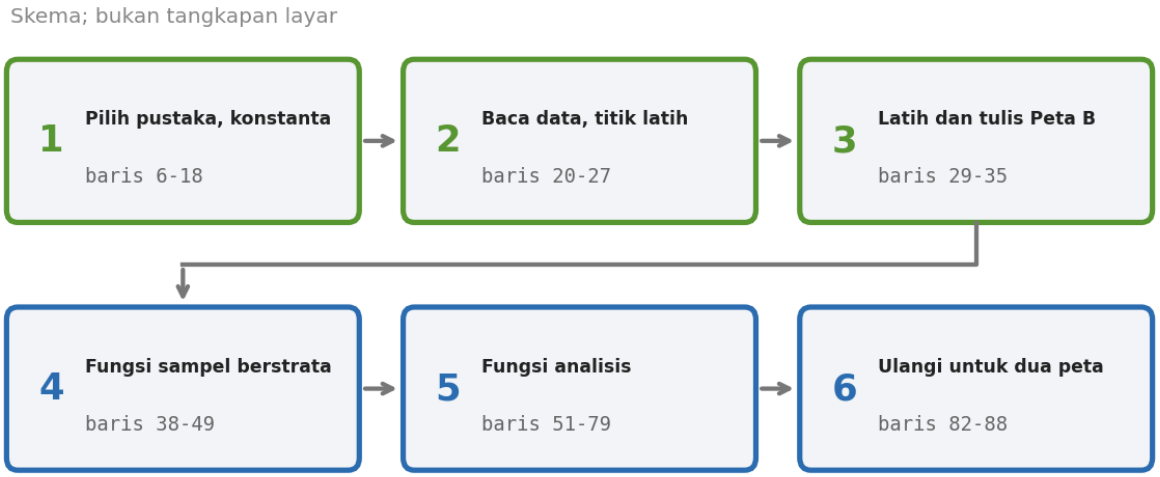
Berkas `7_1_validasi.py`. Di dalam berkasnya tertulis "SKRIP 7.1".

Tujuan. Menguji dua peta klasifikasi dengan sampel acak berstrata: matriks kebingungan, akurasi keseluruhan, akurasi pengguna dan produsen, luas terkoreksi dengan rentang 95 persen, dan kestabilan terhadap pengundian sampel.

Bahan. `fitur_deret.tif` (Skrip 26.12), `peta_rf.tif` (Skrip 26.13), `acuan/Acuan_Dinamika.tif`, `vektor/Titik_Latih.gpkg`, dan `m3_umum.py`. Dua peta diuji. **Peta A** memakai 12 fitur. **Peta B** hanya memakai tujuh fitur NDVI. Dalam praktik, acuan berasal dari tafsir citra resolusi tinggi atau survei lapangan. Di data ini, acuannya kunci jawaban.

Skrip ini yang paling panjang di bab ini, dengan dua fungsi besar.

Ilustrasi 26.27: Alur Skrip 26.14



Skrip utuh:

PYTHON
```
# -*- coding: utf-8 -*-
# SKRIP 7.1: Validasi peta: sampel acak berstrata, matriks kebingungan, akurasi, dan estimasi luas terkoreksi
# Penulis: Badar Mubarok Yogaswara
# Dua peta diuji: (A) hutan acak dengan 12 fitur (hasil Skrip 6.2) dan (B) hutan acak yang hanya memakai fitur
NDVI.
# Acuan: Acuan_Dinamika.tif. Dalam praktik, acuan berasal dari tafsir citra resolusi tinggi atau survei
lapangan.
import os
import numpy as np
from qgis.core import QgsVectorLayer
import m3_umum as U

try:
    from sklearn.ensemble import RandomForestClassifier
except ImportError:
    from hutan_mini import RandomForestClassifier

NAMA = {1: "Hutan alam", 2: "Hutan tanaman", 3: "Pertanian", 4: "Air", 5: "Terbuka", 6: "Deforestasi", 7:
"Terbakar", 8: "Panen HTI"}
N_STRATUM = 20            # sampel per kelas peta
LUAS_PIKSEL = 0.01        # ha

st, gt, prj = U.baca_tif(os.path.join(U.HASIL, "fitur_deret.tif"))
acuan, _, _ = U.baca_tif(os.path.join(U.PAKET, "acuan", "Acuan_Dinamika.tif"))
latih = QgsVectorLayer(os.path.join(U.PAKET, "vektor", "Titik_Latih.gpkg"), "latih")
pl = [(U.xy_ke_piksel(gt, f.geometry().asPoint().x(), f.geometry().asPoint().y()), f["KELAS"]) for f in
latih.getFeatures()]
tr = np.array([p[0] for p in pl])
yl = np.array([p[1] for p in pl])
pakai_latih = np.zeros(acuan.shape, bool)
pakai_latih[tr[:, 0], tr[:, 1]] = True

# peta B: hanya 7 fitur NDVI (band 1 sampai 7 pada tumpukan fitur)
idx = list(range(7))
mB = RandomForestClassifier(n_estimators=100, random_state=1).fit(st[idx][:, tr[:, 0], tr[:, 1]].T, yl)
petaB = mB.predict(st[idx].reshape(len(idx), -1).T).reshape(acuan.shape).astype("uint8")
jB = os.path.join(U.HASIL, "peta_ndvi_saja.tif")
U.tulis_tif(jB, petaB, gt, prj, nodata=0)
```

```
jA = os.path.join(U.HASIL, "peta_rf.tif")


def sampel_berstrata(peta, seed):
    """Sampel acak berstrata: N_STRATUM piksel acak dari tiap kelas peta, di luar titik latih.
    Padanan di antarmuka QGIS: poligonkan peta, lebur per kelas, lalu Random points in polygons (hasil tiap
jalan tidak identik)."""
    rng = np.random.default_rng(seed)
    hasil = {}
    for k in np.unique(peta):
        if k == 0:
            continue
        kand = np.argwhere((peta == k) & ~pakai_latih)
        pilih = rng.choice(len(kand), size=min(N_STRATUM, len(kand)), replace=False)
        hasil[int(k)] = [((int(kand[p][0]), int(kand[p][1])), int(acuan[kand[p][0], kand[p][1]])) for p in
pilih]
    return hasil

def analisis(peta, sampel, nama, cetak=True):
    kelas = sorted(sampel)
    pop = peta[~pakai_latih]                          # populasi sampel = piksel di luar titik latih
    A = pop.size * LUAS_PIKSEL
    W = {k: (pop == k).sum() / pop.size for k in kelas}
    n = {k: len(sampel[k]) for k in kelas}
    nij = np.zeros((len(kelas), len(kelas)))
    for a, k in enumerate(kelas):
        for _, r in sampel[k]:
            if r in kelas:
                nij[a, kelas.index(r)] += 1
    p = np.array([[W[k] * nij[a, b] / n[k] for b in range(len(kelas))] for a, k in enumerate(kelas)])
    oa = np.trace(p)
    ua = np.diag(p) / p.sum(1)
    pa = np.diag(p) / np.maximum(p.sum(0), 1e-12)
    # piksel titik latih sudah punya label pasti: luasnya ditambahkan apa adanya
    luas_peta = np.array([(peta == k).sum() * LUAS_PIKSEL for k in kelas])
    luas_adj = p.sum(0) * A + np.array([(acuan[pakai_latih] == k).sum() * LUAS_PIKSEL for k in kelas])
    se = np.array([np.sqrt(sum(W[ki] ** 2 * (nij[a, b] / n[ki]) * (1 - nij[a, b] / n[ki]) / (n[ki] - 1) for a,
ki in enumerate(kelas))) for b in range(len(kelas))]) * A
    if cetak:
        print("\n=== %s: %d sampel; baris = kelas peta, kolom = kelas acuan ===" % (nama, nij.sum()))
        print("          " + " ".join("%5d" % k for k in kelas))
        for a, k in enumerate(kelas):
            print("kelas %d   " % k + " ".join("%5d" % v for v in nij[a]) + "   | n=%d" % n[k])
        print("Akurasi keseluruhan (berbobot luas): %.3f" % oa)
        print("%-14s %8s %8s %10s %12s %10s" % ("Kelas", "UA", "PA", "luas peta", "luas terkoreksi", "+/-
95%"))
        for a, k in enumerate(kelas):
            print("%-14s %8.2f %8.2f %9.2f ha %11.2f ha %8.2f ha" % (NAMA[k], ua[a], pa[a], luas_peta[a],
luas_adj[a], 1.96 * se[a]))
    return oa, kelas, luas_adj, se


for nama, jalur in (("Peta A (12 fitur)", jA), ("Peta B (NDVI saja)", jB)):
    peta, _, _ = U.baca_tif(jalur)
    oa, kelas, luas_adj, se = analisis(peta, sampel_berstrata(peta, 11), nama)
    print("Luas acuan sebenarnya: " + ", ".join("%s %.2f" % (NAMA[k], (acuan == k).sum() * LUAS_PIKSEL) for k
in (6, 7)))
    # kestabilan terhadap pengundian sampel
    oas = [analisis(peta, sampel_berstrata(peta, s), nama, cetak=False)[0] for s in range(21, 26)]
    print("Akurasi keseluruhan pada 5 undian sampel lain: " + ", ".join("%.3f" % v for v in oas))
```

Penjelasan baris demi baris. Statistiknya sengaja dijelaskan dengan kata-kata dulu, bukan rumus.

Baris	Tulisan	Artinya
6-9	`import os` sampai `import m3_umum as U`	Perkakas.
11-14	`try:` … `except ImportError:`	Seperti Skrip 26.13, tetapi tanpa pesan.
16	`NAMA = {...}`	Kamus nomor kelas ke nama kelas.
17-18	`N_STRATUM = 20` dan `LUAS_PIKSEL = 0.01`	Dua konstanta: sampel per kelas peta, dan luas satu piksel (hektare). Angka kedua dikunci untuk piksel 10 meter.
20-21	`st, gt, prj = ...` dan `acuan, _, _ = ...`	Baca tumpukan fitur dan peta acuan.
22	`latih = QgsVectorLayer(...)`	Layer titik latih.
23	`pl = [(U.xy_ke_piksel(...), f["KELAS"]) for f in latih.getFeatures()]`	Untuk tiap titik latih: pasangan (baris dan kolom piksel, kelas). Hasil: daftar 200 pasangan.
24-25	`tr = np.array([p[0] for p in pl])` dan `yl = ...`	`tr` berisi nomor baris dan kolom, `yl` berisi kelas.
26-27	`pakai_latih = np.zeros(acuan.shape, bool)` lalu `pakai_latih[tr[:, 0], tr[:, 1]] = True`	Larik benar atau salah berukuran peta, benar hanya di 200 piksel titik latih. Piksel itu **tidak boleh** dipakai sebagai sampel uji.
30	`idx = list(range(7))`	Nomor tujuh fitur pertama: fitur NDVI.
31	`mB = RandomForestClassifier(...).fit(st[idx][:, tr[:, 0], tr[:, 1]].T, yl)`	Latih hutan untuk Peta B memakai hanya tujuh fitur, pada piksel titik latih.
32	`petaB = mB.predict(st[idx].reshape(len(idx), -1).T).reshape(acuan.shape).astype("uint8")`	Petakan seluruh area dengan hutan itu, pola yang sama seperti Skrip 26.13.
33-35	`jB = ..., U.tulis_tif(...), jA = ...`	Simpan Peta B, dan catat jalur Peta A (hasil Skrip 26.13).
38-49	`def sampel_berstrata(peta, seed):`	Fungsi pengambil sampel. Docstring baris 39-40 menjelaskan tujuannya.
41-42	`rng = ...` dan `hasil = {}`	Pengacak dengan benih dari masukan, dan kamus hasil.
43-45	`for k in np.unique(peta):` lalu `if k == 0: continue`	Telusuri tiap kelas di peta. Kelas 0 (kosong) dilewati.
46	`kand = np.argwhere((peta == k) & ~pakai_latih)`	Daftar piksel kandidat kelas `k` yang bukan titik latih.
47	`pilih = rng.choice(len(kand), size=min(N_STRATUM, len(kand)), replace=False)`	Undi sampai 20 kandidat tanpa pengembalian. `min(...)` mencegah meminta lebih banyak dari yang ada.

Baris	Tulisan	Artinya
48	`hasil[int(k)] = [((baris, kolom), kelas acuan) for p in pilih]`	Simpan, untuk tiap sampel, lokasinya dan kelas **acuan**-nya. Angka dibungkus `int(...)` agar jadi bilangan Python biasa.
49	`return hasil`	Kembalikan kamus sampel.
51	`def analisis(peta, sampel, nama, cetak=True):`	Fungsi analisis. `cetak=True` adalah nilai bawaan: bila `False`, tidak ada yang dicetak.
52-54	`kelas = sorted(sampel), pop = peta[~pakai_latih], A = pop.size * LUAS_PIKSEL`	Daftar kelas, **populasi** (semua piksel selain titik latih), dan luas populasi dalam hektare.
55	`W = {k: (pop == k).sum() / pop.size for k in kelas}`	**Bobot luas** tiap kelas: pecahan populasi yang dipetakan sebagai kelas itu.
56	`n = {k: len(sampel[k]) for k in kelas}`	Banyak sampel tiap kelas.
57-61	`nij = np.zeros(...)` lalu dua putaran	Isi **matriks kebingungan**: baris kelas peta, kolom kelas acuan. `kelas.index(r)` mencari nomor kolom. Sampel yang kelas acuannya di luar daftar tidak dihitung.
62	`p = np.array([[W[k] * nij[a, b] / n[k] ...]])`	Ubah hitungan menjadi **proporsi luas**: tiap sel dikali bobot kelas petanya, lalu dibagi banyak sampelnya.
63-65	`oa = np.trace(p), ua = ..., pa = ...`	`oa` adalah akurasi keseluruhan (jumlah diagonal). `ua` adalah akurasi pengguna per kelas. `pa` adalah akurasi produsen per kelas. `np.maximum(..., 1e-12)` mencegah bagi nol.
66-68	`luas_peta = ...` dan `luas_adj = ...`	Luas menurut peta, dan **luas terkoreksi**: jumlah kolom `p` dikali luas populasi, ditambah luas titik latih menurut acuan (apa adanya).
69	`se = np.array([np.sqrt(sum(...)) ...]) * A`	**Galat baku** luas tiap kelas. Rumus panjang, dibentuk dari proporsi tiap sel dan banyak sampel.
70-78	`if cetak:` lalu beberapa `print`	Cetak matriks, akurasi keseluruhan, dan tabel UA, PA, luas, serta rentang 95 persen (1,96 kali galat baku).
79	`return oa, kelas, luas_adj, se`	Kembalikan empat hasil.
82	`for nama, jalur in (("Peta A (12 fitur)", jA), ("Peta B (NDVI saja)", jB)):`	Ulangi untuk dua peta.
83-85	`peta, _, _ = ..., oa, kelas, luas_adj, se = analisis(peta, sampel_berstrata(peta, 11), nama), print(...)`	Baca peta, ambil sampel (benih 11), analisis, dan cetak luas acuan sebenarnya untuk deforestasi dan terbakar.
87-88	`oas = [analisis(..., cetak=False)[0] for s in range(21, 26)]` dan `print(...)`	Ulangi pengambilan sampel dengan lima benih lain, dan kumpulkan **hanya** akurasi keseluruhannya (`[0]`). Cetak lima angka itu untuk melihat kestabilan.

Hasil. Layar menampilkan (berdasarkan uji pada QGIS 4.0.2, tanpa scikit-learn):

```
HASIL
=== Peta A (12 fitur): 158 sampel; baris = kelas peta, kolom = kelas acuan ===
              1     2     3     4     5     6     7     8
kelas 1      20     0     0     0     0     0     0     0   | n=20
kelas 2       0    20     0     0     0     0     0     0   | n=20
kelas 3       0     0    20     0     0     0     0     0   | n=20
kelas 4       0     0     0    20     0     0     0     0   | n=20
kelas 5       0     0     0     0    18     0     0     0   | n=18
kelas 6       0     0     0     0     0    20     0     0   | n=20
kelas 7       1     0     0     0     0     0    19     0   | n=20
kelas 8       0     0     0     0     0     0     0    20   | n=20
Akurasi keseluruhan (berbobot luas): 0.995
Kelas                UA       PA  luas peta luas terkoreksi    +/- 95%
Hutan alam         1.00     0.99      7.99 ha        8.05 ha     0.12 ha
Hutan tanaman      1.00     1.00      0.96 ha        0.96 ha     0.00 ha
Pertanian          1.00     1.00      1.80 ha        1.80 ha     0.00 ha
Air                1.00     1.00      0.80 ha        0.80 ha     0.00 ha
Terbuka            1.00     1.00      0.43 ha        0.43 ha     0.00 ha
Deforestasi        1.00     1.00      1.30 ha        1.30 ha     0.00 ha
Terbakar           0.95     1.00      1.52 ha        1.46 ha     0.12 ha
Panen HTI          1.00     1.00      1.20 ha        1.20 ha     0.00 ha
Luas acuan sebenarnya: Deforestasi 1.30, Terbakar 1.51
Akurasi keseluruhan pada 5 undian sampel lain: 0.995, 1.000, 1.000, 1.000, 1.000

=== Peta B (NDVI saja): 158 sampel; baris = kelas peta, kolom = kelas acuan ===
              1     2     3     4     5     6     7     8
kelas 1      20     0     0     0     0     0     0     0   | n=20
kelas 2       0    20     0     0     0     0     0     0   | n=20
kelas 3       0     0    20     0     0     0     0     0   | n=20
kelas 4       0     0     0    20     0     0     0     0   | n=20
kelas 5       0     0     0     0    18     0     0     0   | n=18
kelas 6       0     0     0     0     0    20     0     0   | n=20
kelas 7      10     0     0     0     0     0    10     0   | n=20
kelas 8       0     0     0     0     0     0     0    20   | n=20
Akurasi keseluruhan (berbobot luas): 0.915
Kelas                UA       PA  luas peta luas terkoreksi    +/- 95%
Hutan alam         1.00     0.85      6.87 ha        8.07 ha     0.54 ha
Hutan tanaman      1.00     1.00      0.96 ha        0.96 ha     0.00 ha
Pertanian          1.00     1.00      1.80 ha        1.80 ha     0.00 ha
Air                1.00     1.00      0.80 ha        0.80 ha     0.00 ha
Terbuka            1.00     1.00      0.43 ha        0.43 ha     0.00 ha
Deforestasi        1.00     1.00      1.30 ha        1.30 ha     0.00 ha
Terbakar           0.50     1.00      2.64 ha        1.44 ha     0.54 ha
Panen HTI          1.00     1.00      1.20 ha        1.20 ha     0.00 ha
Luas acuan sebenarnya: Deforestasi 1.30, Terbakar 1.51
Akurasi keseluruhan pada 5 undian sampel lain: 0.940, 0.917, 0.874, 0.889, 0.925
```

Bandingkan dua peta. Peta A hampir sempurna: akurasi keseluruhan 0,995 dan luas terbakar terkoreksi 1,46 hektare, dekat dengan acuan 1,51 hektare. Peta B (hanya NDVI) lebih lemah: akurasi 0,915. Lihat baris kelas 7 pada matriks Peta B: dari 20 sampel yang peta sebut **Terbakar**, hanya 10 yang benar terbakar dan 10 adalah **hutan alam**. Akurasi penggunanya 0,50. Luas terbakar menurut peta B adalah 2,64 hektare, sekitar 1,7 kali luas acuan (1,51 hektare). Setelah dikoreksi, luasnya 1,44 hektare dengan rentang plus minus 0,54. Kesimpulan: **luas dari peta mentah bisa jauh meleset, dan koreksi luas memperbaikinya.** Lima undian ulang menunjukkan Peta B kurang stabil (0,874 sampai 0,940), sedangkan Peta A hampir tidak bergeser.

Bagian yang boleh diubah.

- Baris 17, `N_STRATUM`. Jumlah sampel per kelas. Saya mengubahnya menjadi `10`: sampel Peta A turun menjadi 80 dan akurasi keseluruhan naik menjadi 1,000. Akurasi Peta B menjadi 0,863, dan lima undian ulangnya melebar (0,881 sampai 0,983). Sampel yang lebih sedikit membuat angka lebih goyah.
- **Baris 84 dan 87, benih.** Ganti benih 11 atau rentang `range(21, 26)` untuk undian lain.
- Baris 30, `range(7)`. Daftar fitur untuk Peta B. Ganti untuk menguji kelompok fitur lain.

Jebakan.

- Sampel kecil menghasilkan angka `nan`. Galat baku membagi dengan `n - 1`. Saya mengubah `N_STRATUM` menjadi `1`: layar tetap menampilkan tabel, tetapi kolom rentang berisi `nan ha`, tanpa galat. Skrip diam. Bila ada kelas dengan satu sampel, jangan percaya rentangnya.
- **Titik latih dikeluarkan dari populasi.** Itu sengaja, supaya tidak menguji dengan soal yang sudah dipelajari. Akibatnya, luas terkoreksi menambah luas titik latih "apa adanya" (baris 68). Bentuk ini tidak persis sama dengan rumus bakunya [CEK].
- **Rumus statistik tidak dinamai di skrip.** Pendekatannya mirip penaksir berstrata yang dibahas Olofsson dan kawan-kawan (2014, *Remote Sensing of Environment* 148: 42-57). Saya hanya memastikan artikel itu ada, dan belum mencocokkan rumus baris demi baris dengan skrip [CEK].
- **Dua peta, satu pengujian.** Peta A harus sudah dibuat Skrip 26.13, dan tumpukan fitur harus sudah dibuat Skrip 26.12.
- **Nomor fitur dikunci.** Baris 30 mengandaikan tujuh fitur pertama adalah fitur NDVI.
- **Luas 0,01 hektare per piksel dikunci.**

Latihan variasi.

1. Kurangi sampel per kelas menjadi 10.
2. Lihat apakah akurasi Peta B bertambah goyah.

Contoh jawaban, hanya baris 17:

```python
N_STRATUM = 10            # sampel per kelas peta
```

Layar menampilkan `Akurasi keseluruhan (berbobot luas): 0.863` untuk Peta B, dan lima undian ulang `0.881, 0.915, 0.915, 0.983, 0.915`. Rentang undian ulang (0,881 sampai 0,983) lebih lebar daripada yang asli (0,874 sampai 0,940). Jadi, Peta B menjadi lebih goyah.

> **Pola yang bisa dipakai ulang:** Validasi selalu berbentuk **ambil sampel di luar titik latih, bandingkan dengan acuan, hitung matriks, koreksi luas**. Ganti **peta yang diuji** (baris 82), **acuan** (baris 21), **jumlah sampel** (baris 17), dan **luas piksel** (baris 18). Selalu ulangi pengundian dengan beberapa benih, untuk melihat apakah angka Anda stabil.

Cek paham

1. Mengapa grafik NDVI hutan alam di Skrip 26.2 sering jatuh ke nol, padahal hutannya utuh?
2. Mengapa uji silang blok (Skrip 26.13) memberi akurasi jauh lebih rendah daripada uji silang acak?
3. Skrip 26.9 menyimpan tanggal sebagai bilangan seperti 20241115. Mengapa baris 54 harus menyebut `tipe=gdal.GDT_Int32`, dan apa yang terjadi bila Anda membuang bagian itu?

Jawaban.

1. Awan dan bayangannya ikut terhitung sebagai permukaan, sehingga NDVI jatuh pada tanggal berawan. Hutannya tidak hilang. Kelompok B membuang atau menambal piksel itu.
2. Pada uji acak, tetangga dekat bisa ada di kedua sisi, jadi mesin "mengintip" jawabannya. Pada uji blok, blok yang diuji benar-benar asing bagi mesin. Angka blok lebih jujur untuk data berlokasi.
3. Fungsi `U.tulis_tif` memberi tipe Byte (0 sampai 255) untuk semua larik non-desimal, jadi 20241115 terpotong menjadi 255. Dengan `tipe=gdal.GDT_Int32` di baris 54 Skrip 26.9, berkas menyimpan tanggal utuh. Skrip tidak mengeluh dalam kedua kasus, jadi tipe berkas harus diperiksa sendiri.

Kesalahan umum

- **Menjalankan skrip tanpa skrip pendahulunya.** Skrip 26.4 berhenti dengan pesan bila masukannya tidak ada, tetapi tidak semua skrip dijaga begitu. Perbaikan: ikuti tabel urutan di awal bab, dan cocokkan angka keluaran dengan contoh di bab.
- **Lupa menghapus hasil lama sebelum mengulang.** Alat Polygonize menambah poligon ke berkas yang ada. Skrip 26.8 dan 26.11 sudah menghapus `.gpkg` lama lebih dulu. Bila Anda menulis skrip sendiri dengan alat ini, tambahkan penghapusan yang sama.
- **Mempercayai satu ambang atau satu akurasi.** Ambang literatur belum tentu cocok untuk data Anda (Skrip 26.10), dan uji silang acak terlalu manis (Skrip 26.13). Perbaikan: uji beberapa ambang, dan pakai uji blok serta sampel berstrata.
- **Membiarkan angka yang dikunci untuk data contoh.** Ukuran 40 kali 40, luas 0,01 hektare per piksel, dan batas koordinat tertulis langsung di kode. Perbaikan: gunakan tabel di bawah ini sebagai daftar periksa saat Anda mengganti data.

Skrip	Hal yang dikunci	Letak
26.8 (`4_1`)	Batas area, piksel 10 meter, ukuran 40, luas 0,01 hektare, pembagian KPH tiruan	baris 29, 41, 46-49
26.9 (`4_2`)	Tanggal sebenarnya dua bukaan dan pembagian wilayah	baris 56-57
26.10 (`5_1`)	Nomor tanggal sebelum dan sesudah, ukuran 40, luas 0,01 hektare	baris 34, 37-38, 54, 59, 64
26.13 (`6_2`)	Empat lipatan dan batas blok 312200 dan 9996200, luas 0,01 hektare	baris 31, 36, 62
26.14 (`7_1`)	Luas piksel 0,01 hektare dan tujuh fitur pertama	baris 18, 30
`m3_umum.py`	Jalur cadangan dan tipe data Byte untuk bilangan bulat	baris 14-15, 65-66

Ringkasan dan latihan

- Deret waktu adalah **tumpukan** citra bertanggal. Tiap piksel punya riwayat angka yang bisa dibaca.
- Pembersihan awan memakai tiga cara: **buang** (mask), **gabung** (median), dan **tambal** (interpolasi).
- Deteksi perubahan punya dua cara: **dua waktu** (cepat, tanpa tanggal) dan **titik patah** (menyebut tanggal).
- Kebakaran lebih jelas terlihat di **NBR** daripada di NDVI. Titik panas adalah petunjuk, bukan peta bekas bakar.
- Mesin belajar butuh **fitur**. Uji **blok** memberi angka yang lebih jujur, dan **validasi** dengan sampel berstrata memberi luas terkoreksi.
- Skrip yang jalan tanpa galat belum tentu benar. **Cocokkan angkanya.**

Latihan. Gabungkan tiga skrip. Jalankan Skrip 26.1, 26.4, dan 26.5 untuk periode "kemarau2025" (tanggal ke-17 sampai ke-20). Lalu hitung dNDVI antara komposit "sebelum" dan "kemarau2025" dengan pola Skrip 26.8. Bagian mana dari Skrip 26.8 yang harus Anda ganti, dan bagian mana yang boleh dibiarkan?

Tabel perbandingan: pekerjaan, skrip, dan perkakas Python

Pekerjaan	Skrip	Perkakas Python yang paling berperan
Menyusun tumpukan	26.1 sampai 26.3	`for` dan tuple, `processing.run`, `gdal`
Membersihkan awan	26.4 sampai 26.6	ekspresi kalkulator raster, irisan daftar, fungsi buatan sendiri, `np.interp`
Mengukur irama	26.7	larik NumPy dan sumbu, kamus kelas, mask benar-salah
Hutan hilang	26.8 dan 26.9	konstanta, `continue` dan `break`, fungsi dengan nilai bawaan

Pekerjaan	Skrip	Perkakas Python yang paling berperan
Jejak api	26.10 dan 26.11	kamus berisi tuple, `zip(*...)`, `Counter`
Mesin belajar	26.12 dan 26.13	kamus fitur, `try ... except ImportError`, kelas dan rekursi (`hutan_mini`)
Validasi	26.14	comprehension, fungsi dengan banyak hasil, sampel acak berstrata

BAB 27: Dari Skrip ke Produk: Dasbor, Peta Web, Model 3D, Alat Processing, dan Plugin

> **Berdasarkan Seri M4:** Skrip bab ini berasal dari Buku M4, *Pengembangan dan Publikasi* (Master Class), berkas `buat_dasbor.py`, `buat_peta_web.py`, `dsm_ke_obj.py`, `kerapatan_hotspot.py`, `tambah_luas_ha.py`, `kemas_plugin.py`, dan paket `plugin_ringkas_kph` (dua berkas Python). Di buku itu, topiknya dibahas lewat menu dan klik. Bab ini membahas isi skripnya. Berkas uji penulis di folder yang sama tidak dibahas.

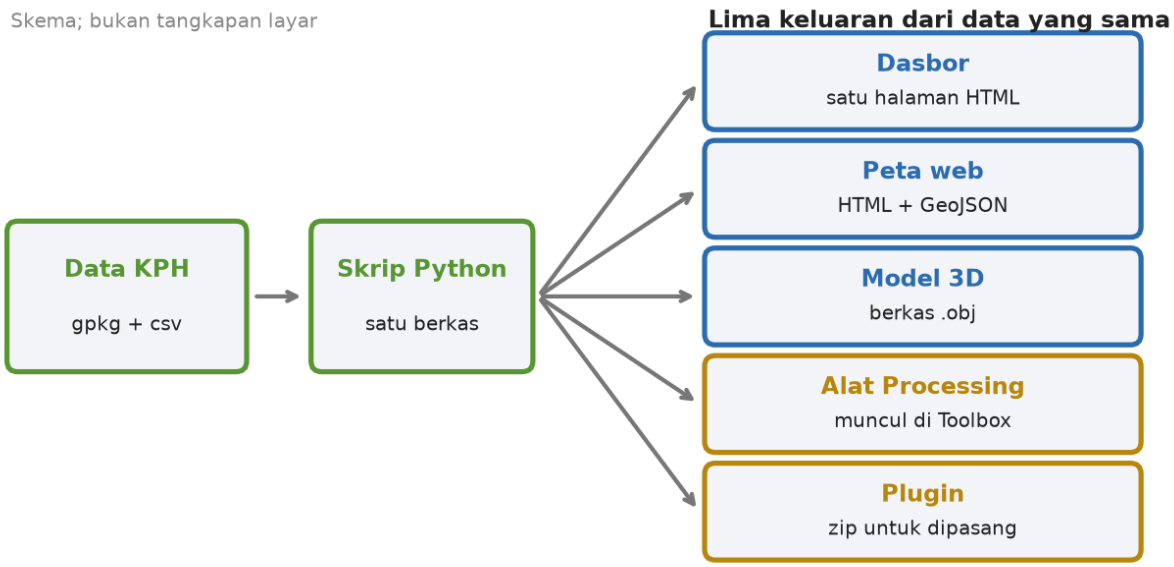
Studi kasus: "Hasilnya harus bisa dipakai orang lain"

Analis Muda sudah pandai menghitung. Kepala Seksi kini meminta hal lain: ringkasan satu halaman untuk rapat, peta yang bisa dibuka dari ponsel, model 3D lahan, dan alat hitung yang bisa dipakai rekan kerja tanpa menyalin skrip. Semua itu bukan analisis baru. Itu pengemasan. Bab ini menunjukkan lima cara mengemas hasil kerja Anda, lengkap dengan skrip yang membuatnya.

Konsep: skrip sebagai pabrik kecil

Bayangkan skrip sebagai **pabrik kecil**. Bahan mentahnya data KPH. Mesinnya Python. Produknya bisa berupa halaman HTML, berkas peta, berkas 3D, alat baru di QGIS, atau satu paket plugin. Anda tidak mengubah cara menghitung. Anda mengubah **bentuk keluaran**.

Ilustrasi 27.1: Satu data, lima keluaran



Lima istilah baru untuk bab ini:

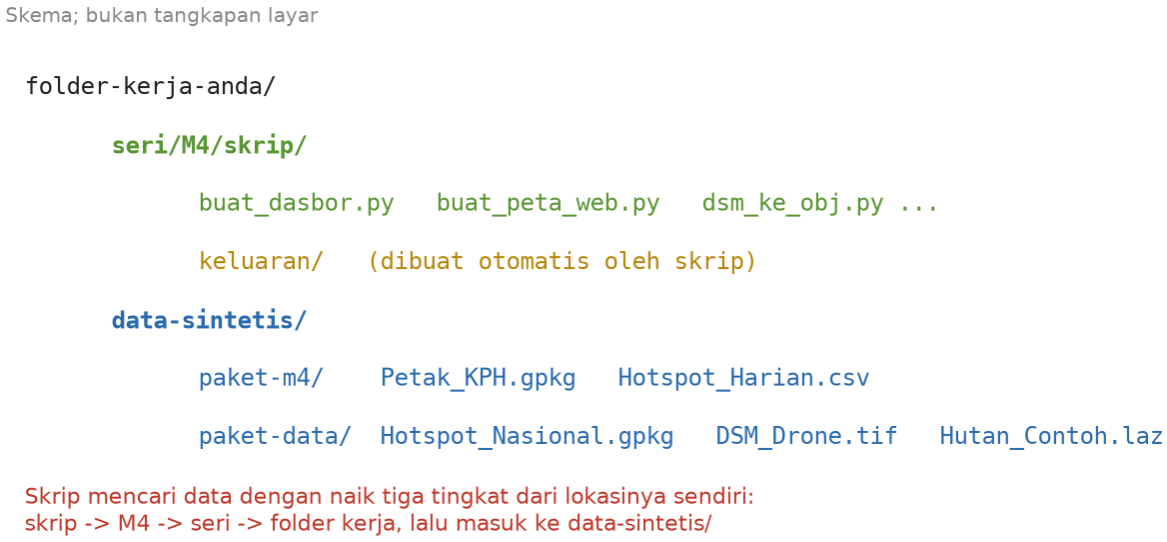
- **HTML**: berkas teks yang dibuka di peramban (*browser*) sebagai halaman web.
- **GeoJSON**: berkas teks berisi objek peta dan atributnya, mudah dibaca peramban.

- **Algoritma Processing**: alat yang tampil di Processing Toolbox QGIS dan bisa dipanggil dari Python.
- **Plugin**: paket kecil yang menambah menu dan tombol di QGIS.
- **Kelas** dan **dekorator**: dua cara menulis alat; keduanya dibahas di Bab 11 dan dibandingkan di bab ini.

Menyiapkan dulu: folder kerja dengan struktur penulis

Skrip M4 punya satu kebiasaan khusus. Mereka menghitung lokasi data dari lokasi **skripnya sendiri**: naik tiga folder, lalu masuk ke `data-sintetis`. Karena itu skrip hanya jalan bila struktur foldernya sama dengan milik penulis. Jangan ubah skripnya. Cukup bangun struktur yang sama di komputer Anda.

Ilustrasi 27.2: Struktur folder yang diharapkan skrip M4



Langkahnya:

1. Buat folder kerja, misalnya `D:/Latihan`.
2. Di dalamnya buat `seri/M4/skrip`, lalu salin semua skrip bab ini ke sana. Salin juga folder `plugin_ringkas_kph`.
3. Di folder kerja yang sama buat folder `data-sintetis`.
4. Salin folder `paket-m4` ke `data-sintetis/paket-m4`.
5. Salin folder `paket-data` ke `data-sintetis/paket-data`.

Pada uji bab ini, struktur di atas dibangun di folder sementara dan semua skrip dijalankan dengan perintah `python-qgis.bat` (lihat Bab 3). Di macOS dan Ubuntu, perintahnya berbeda. [CEK]

TERMINAL

```
cd D:/Latihan/seri/M4/skrip
"C:/Program Files/QGIS 4.0.2/bin/python-qgis.bat" buat_dasbor.py
```

Jalur `C:/Program Files/QGIS 4.0.2` adalah lokasi QGIS di komputer uji. Ganti sesuai pemasangan Anda. Setiap skrip membuat folder `keluaran` sendiri di samping dirinya.

Skrip 27.1: Dasbor satu halaman

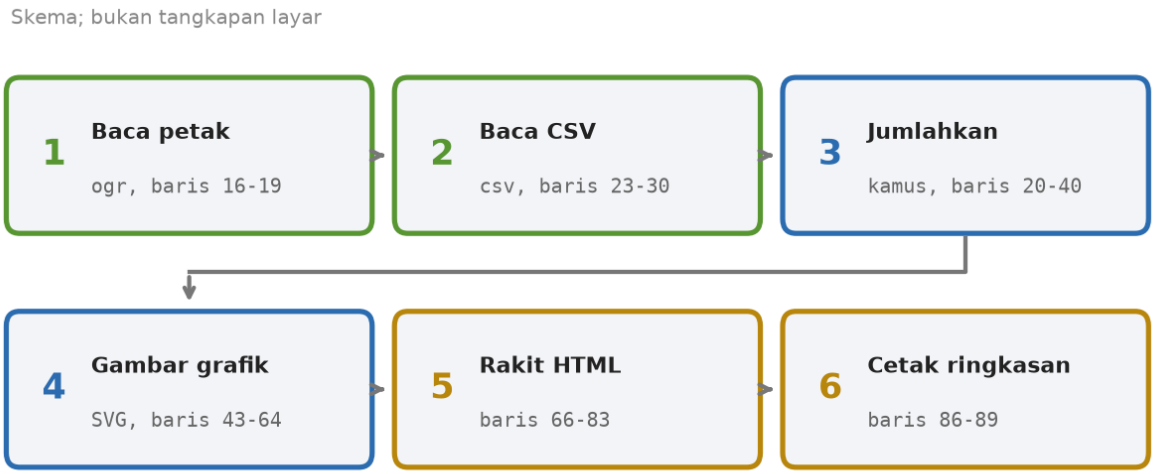
Berkas `buat_dasbor.py`. Bagi pembaca buku M4, ini skrip dasbor di Bab 4.

Tujuan. Membaca petak dan data titik panas harian, lalu menulis satu halaman HTML berisi empat kartu angka dan dua grafik.

Bahan. Paket `paket-m4`: `Petak_KPH.gpkg` (16 petak) dan `Hotspot_Harian.csv` (titik panas per petak per hari, 1 sampai 30 September 2026, data sintetis). Skrip tidak memakai `qgis.core`. Ia hanya butuh GDAL, jadi cukup dijalankan dengan `python-qgis.bat`.

Skrip ini bekerja dalam enam tahap.

Ilustrasi 27.3: Alur skrip 27.1



Skrip utuh:

```python
# -*- coding: utf-8 -*-
"""Dasbor HTML satu berkas (tanpa pustaka luar) untuk KPH Contoh. Penulis: Badar Mubarok Yogaswara
Membaca Petak_KPH.gpkg dan Hotspot_Harian.csv, menulis keluaran/dasbor/dasbor.html. Jalankan: python-qgis.bat
buat_dasbor.py
"""
import csv, json, os
from collections import defaultdict
from osgeo import ogr
ogr.UseExceptions()
BULAN = ["Jan", "Feb", "Mar", "Apr", "Mei", "Jun", "Jul", "Agu", "Sep", "Okt", "Nov", "Des"]
AQ = os.path.dirname(os.path.abspath(__file__))
PAKET = os.path.abspath(os.path.join(AQ, "..", "..", "..", "data-sintetis", "paket-m4"))
OUT = os.path.join(AQ, "keluaran", "dasbor")


def hitung():
    petak = {}
    ds = ogr.Open(os.path.join(PAKET, "Petak_KPH.gpkg"))
    for f in ds.GetLayer(0):
        petak[f["ID_Petak"]] = {"kph": f["NAMA_KPH"], "luas": f["Luas_Ha"]}
```

```
    per_hari = defaultdict(int)
    per_petak = defaultdict(int)
    per_kph = defaultdict(int)
    with open(os.path.join(PAKET, "Hotspot_Harian.csv"), encoding="utf-8") as fh:
        for b in csv.DictReader(fh):
            j = int(b["Jumlah"])
            if b["ID_Petak"] not in petak:
                raise ValueError("ID_Petak %s di Hotspot_Harian.csv tidak ada di Petak_KPH.gpkg" %
b["ID_Petak"])
            per_hari[b["Tanggal"]] += j
            per_petak[b["ID_Petak"]] += j
            per_kph[petak[b["ID_Petak"]]["kph"]] += j
    luas_kph = defaultdict(float)
    for p in petak.values():
        luas_kph[p["kph"]] += p["luas"]
    hari_puncak = max(per_hari, key=lambda k: (per_hari[k], k))
    teratas = sorted(per_petak.items(), key=lambda kv: (-kv[1], kv[0]))[:3]
    return {
        "total": sum(per_hari.values()), "hari": sorted(per_hari.items()), "hari_puncak": hari_puncak,
        "jumlah_puncak": per_hari[hari_puncak], "per_kph": dict(per_kph), "luas_kph": dict(luas_kph),
        "teratas": teratas, "jumlah_petak": len(petak),
    }


def label_tgl(tanggal):
    """'2026-09-30' -> '30 Sep' (hari tanpa nol di depan)."""
    return "%d %s" % (int(tanggal[8:10]), BULAN[int(tanggal[5:7]) - 1])


def garis_svg(hari, w=520, h=160):
    mx = max(v for _, v in hari) or 1
    pts = " ".join("%.1f,%.1f" % (10 + i * (w - 20) / max(len(hari) - 1, 1), h - 15 - v * (h - 30) / mx) for i,
(_, v) in enumerate(hari))
    return ('<svg viewBox="0 0 %d %d" role="img" aria-label="Titik panas per hari"><polyline fill="none"
stroke="#C0392B" '
            'stroke-width="2.5" points="%s"/><text x="10" y="%d" font-size="11" fill="#555">%s</text>'
            '<text x="%d" y="%d" font-size="11" fill="#555" text-anchor="end">%s</text></svg>') % (w, h, pts, h
- 2, label_tgl(hari[0][0]), w - 10, h - 2, label_tgl(hari[-1][0]))


def batang_svg(data, w=520):
    mx = max(data.values()) or 1
    baris = []
    for i, (k, v) in enumerate(sorted(data.items())):
        y = 8 + i * 34
        baris.append('<text x="0" y="%d" font-size="13" fill="#333">%s</text><rect x="90" y="%d" width="%.1f"
height="20" fill="#2B6CB0"/>'
                     '<text x="%.1f" y="%d" font-size="13" fill="#333">%d</text>' % (y + 15, k, y, v * (w -
150) / mx, 96 + v * (w - 150) / mx, y + 15, v))
    return '<svg viewBox="0 0 %d %d" role="img" aria-label="Titik panas per KPH">%s</svg>' % (w, 8 + len(data)
* 34, "".join(baris))


def tulis(d):
    os.makedirs(OUT, exist_ok=True)
    kartu = [("Total titik panas (30 hari)", d["total"]), ("Hari puncak", "%s (%d titik)" % (d["hari_puncak"]
[-2:] + " " + BULAN[int(d["hari_puncak"][5:7]) - 1], d["jumlah_puncak"])),
             ("Jumlah petak", d["jumlah_petak"]), ("Petak teratas", "%s (%d)" % d["teratas"][0])]
    html = """<!doctype html><html lang="id"><head><meta charset="utf-8"><meta name="viewport"
content="width=device-width, initial-scale=1">
<title>Dasbor KPH Contoh</title><style>
body{font-family:system-ui,sans-serif;margin:0;background:#f4f6f9;color:#1F3A5F}
main{max-width:900px;margin:auto;padding:16px}.kartu{display:grid;grid-template-columns:repeat(auto-
fit,minmax(190px,1fr));gap:12px}
.k{background:#fff;border-radius:8px;padding:12px}.k b{display:block;font-size:22px}
.p{background:#fff;border-radius:8px;padding:12px;margin-top:12px}svg{width:100%%;height:auto}
</style></head><body><main><h1>Dasbor KPH Contoh: titik panas September</h1>
<div class="kartu">%s</div>
```

```
<div class="p"><h2>Titik panas per hari</h2>%s</div>
<div class="p"><h2>Titik panas per KPH</h2>%s</div>
<p>Data sintetis. Bukan kondisi lapangan sebenarnya.</p></main></body></html>""" % (
        "".join('<div class="k">%s<b>%s</b></div>' % (a, b) for a, b in kartu), garis_svg(d["hari"]),
batang_svg(d["per_kph"]))
    with open(os.path.join(OUT, "dasbor.html"), "w", encoding="utf-8") as fh:
        fh.write(html)


if __name__ == "__main__":
    d = hitung()
    tulis(d)
    print(json.dumps({k: d[k] for k in ("total", "hari_puncak", "jumlah_puncak", "per_kph", "luas_kph",
"teratas", "jumlah_petak")}, ensure_ascii=False))
```

Penjelasan baris demi baris. Bagian pertama: perkakas dan jalur (baris 1-12).

Baris	Tulisan	Artinya
1	`# -*- coding: utf-8 -*-`	Komentar penanda: berkas ini ditulis dengan huruf UTF-8.
2-4	`"""..."""`	Teks penjelasan di awal berkas (*docstring*). Python tidak menjalankannya.
5	`import csv, json, os`	Panggil tiga perkakas sekaligus: `csv` membaca tabel teks, `json` merakit teks JSON, `os` mengurus jalur folder (Bab 8).
6	`from collections import defaultdict`	Panggil kamus khusus yang otomatis mulai dari nol bila kunci belum ada.
7-8	`from osgeo import ogr` dan `ogr.UseExceptions()`	Panggil pembaca data vektor dari GDAL. Baris 8 membuatnya berteriak (galat) bila gagal, bukan diam saja.
9	`BULAN = ["Jan", "Feb", ..., "Des"]`	Daftar dua belas singkatan bulan. Nomor bulan 1 sampai 12 dipakai untuk memilih namanya (urutan daftar mulai dari 0, jadi bulan 9 ada di nomor 8).
10	`AQ = os.path.dirname(os.path.abspath(__file__))`	Isi kotak `AQ` dengan lokasi folder skrip ini. `__file__` adalah nama berkas yang sedang berjalan.
11	`PAKET = os.path.abspath(os.path.join(AQ, "..", "..", "..", "data-sintetis", "paket-m4"))`	Dari `AQ` naik tiga folder (tiga kali `".."`), lalu masuk `data-sintetis/paket-m4`. Ini sebab struktur folder harus sama.
12	`OUT = ...`	Lokasi folder keluaran: `keluaran/dasbor` di samping skrip.

Bagian kedua: menghitung (baris 15-40). Fungsi `hitung()` membuat tiga "buku catatan" (kamus) lalu mengisinya.

Baris	Tulisan	Artinya
16-19	`petak = {}` lalu `for f in ds.GetLayer(0)`	Buka GeoPackage, lalu catat tiap petak: kodenya jadi kunci, isinya nama KPH dan luas. Kamus di dalam kamus.
20-22	`defaultdict(int)` tiga kali	Siapkan tiga catatan jumlah: per hari, per petak, per KPH. Isi awalnya otomatis 0.

Baris	Tulisan	Artinya
23-24	`with open(...)` dan `csv.DictReader(fh)`	Buka CSV. `DictReader` memberi tiap baris sebagai kamus, kolomnya dipanggil dengan nama (Bab 9).
25	`j = int(b["Jumlah"])`	Ubah teks `"1"` menjadi angka 1 agar bisa dijumlah.
26-27	`if b["ID_Petak"] not in petak: raise ValueError(...)`	Pengaman: sebelum menjumlah, cek apakah kode petak di CSV ada di GeoPackage. Bila tidak ada, skrip berhenti dengan pesan yang menyebut kode petaknya. `raise ValueError` dibahas di Bab 10.
28-30	`per_hari[...] += j`	Tambahkan jumlah ke catatan hari, petak, dan KPH. Baris 30 mencari KPH petak itu lewat kamus `petak`, jadi baris 26-27 harus lebih dulu.
31-33	`luas_kph`	Jumlahkan luas petak per KPH.
34	`max(per_hari, key=lambda k: (per_hari[k], k))`	Cari hari dengan titik terbanyak. `lambda` adalah fungsi mini (Bab 7). Bila ada dua hari sama banyak, tanggal yang lebih akhir menang.
35	`sorted(..., key=lambda kv: (-kv[1], kv[0]))[:3]`	Urutkan petak dari yang terbanyak. Tanda minus membalik urutan. `[:3]` mengambil tiga pertama.
36-40	`return {...}`	Kembalikan semua hasil dalam satu kamus.

Bagian ketiga: menggambar (baris 43-63). Grafik di sini bukan gambar, melainkan teks SVG, yaitu bahasa gambar yang dipahami peramban.

Baris	Tulisan	Artinya
43-45	`def label_tgl(tanggal)` dan `return "%d %s" % (...)`	Ubah tanggal `2026-09-30` menjadi `30 Sep`. Hari ada di huruf ke-9 dan ke-10 (`tanggal[8:10]`), bulan di huruf ke-6 dan ke-7 (`tanggal[5:7]`). Nomor bulan dikurangi 1 lalu dicari namanya di `BULAN`. `int(...)` membuang angka nol di depan hari.
48-49	`def garis_svg(hari, w=520, h=160)` dan `mx = ...`	Fungsi grafik garis. Lebar `w` dan tinggi `h` punya nilai bawaan. `mx` adalah nilai terbesar, untuk skala.
50	`pts = " ".join(... for i, (_, v) in enumerate(hari))`	Hitung titik koordinat tiap hari lalu sambung dengan spasi. `i` adalah urutan hari, `v` jumlah titik panas. Tanda `_` berarti tanggal diabaikan. Jarak antartitik dibagi `max(len(hari) - 1, 1)`: `max` memilih yang terbesar, jadi pembaginya paling kecil 1 dan tidak pernah 0.
51-53	teks `<svg ...>` dengan `%`	Rakit teks gambar: garis merah dan dua label tanggal. Label kiri memakai `label_tgl(hari[0][0])` (hari pertama), label kanan `label_tgl(hari[-1][0])` (hari terakhir). Tanda `%` di ujung mengisi `%d` dan `%s` dengan nilai (Bab 9).
56-63	`def batang_svg(data, w=520)`	Sama, tetapi untuk batang. Tiap KPH mendapat satu batang biru, panjangnya sebanding dengan jumlahnya.

Bagian keempat: merakit dan menjalankan (baris 66-89).

Baris	Tulisan	Artinya
67	`os.makedirs(OUT, exist_ok=True)`	Buat folder keluaran. Bila sudah ada, tidak apa-apa.
68-69	`kartu = [...]`	Daftar empat kartu: judul dan isi. Baris 68 memotong tanggal puncak dengan `[-2:]` (dua huruf terakhir, yaitu tanggalnya) lalu menambah nama bulan dari tanggal itu sendiri, diambil dari `BULAN`.
70-81	`html = """..."""  % (...)`	Teks HTML panjang dengan tiga lubang `%s`: kartu, grafik garis, grafik batang. Tanda `%%` di CSS dibaca sebagai satu tanda persen.
82-83	`with open(..., "w", encoding="utf-8")`	Tulis HTML ke berkas `dasbor.html`.
86	`if __name__ == "__main__":`	Jalankan bagian di bawahnya hanya bila berkas ini dijalankan langsung.
87-89	`d = hitung(), tulis(d), print(json.dumps(...))`	Hitung, tulis, lalu cetak ringkasan sebagai teks JSON. `ensure_ascii=False` menjaga huruf biasa tampil apa adanya.

Hasil. Layar menampilkan satu baris panjang (berdasarkan uji pada QGIS 4.0.2). Baris itu dipecah di sini agar terbaca.

HASIL

```
{"total": 385, "hari_puncak": "2026-09-21", "jumlah_puncak": 34,
 "per_kph": {"KPH Alpha": 91, "KPH Beta": 65, "KPH Gamma": 229},
 "luas_kph": {"KPH Alpha": 4.0, "KPH Beta": 4.0, "KPH Gamma": 8.0},
 "teratas": [["P-14", 33], ["P-12", 31], ["P-13", 31]], "jumlah_petak": 16}
```

Berkas `keluaran/dasbor/dasbor.html` terbentuk (2.309 byte). Isinya empat kartu: total titik panas 385, hari puncak 21 Sep (34 titik), jumlah petak 16, dan petak teratas P-14 (33). Buka dengan klik dua kali di peramban. Halaman ini tidak memuat berkas lain, jadi tidak perlu internet. Halaman ini sudah dibuka di peramban (dilayani lewat server lokal; klik dua kali pada berkas tidak diuji). Yang tampak: judul "Dasbor KPH Contoh: titik panas September", empat kartu (385; 21 Sep (34 titik); 16; P-14 (33)), grafik garis merah berlabel "1 Sep" dan "30 Sep", tiga batang biru (91, 65, 229), dan catatan data sintetis. Pada layar sempit kartu keempat turun ke baris kedua.

Bagian yang boleh diubah.

- Baris 35, angka 3 dalam `[:3]`. Jumlah petak teratas dalam ringkasan. Akibatnya hanya daftar yang dicetak, kartu tetap menampilkan petak pertama.
- Baris 51, warna `#C0392B`. Warna garis grafik. Kode warna enam huruf.
- Baris 61, warna `#2B6CB0`. Warna batang.
- Baris 48 dan 56, `w` dan `h`. Ukuran grafik.
- **Baris 68-69 dan 76, judul dan teks kartu.** Ganti sesuai laporan Anda. Baris 76 memuat kata "September" pada judul halaman.

- Baris 9, daftar `BULAN`. Ganti singkatan bila Anda ingin nama penuh, misalnya "Januari".

Jebakan.

- **Jalur data salah.** Bila skrip dipindah ke folder lain, galatnya: `RuntimeError: ...\data-sintetis\paket-m4\Petak_KPH.gpkg: No such file or directory`. Jalur lengkap yang dicari tercetak, jadi mudah dilacak.
- **Petak yang tak dikenal.** Bila CSV memuat petak yang tidak ada di GeoPackage, misalnya `P-99`, skrip berhenti di baris 27 dengan pesan `ValueError: ID_Petak P-99 di Hotspot_Harian.csv tidak ada di Petak_KPH.gpkg`. (Diuji dengan menambah satu baris ke salinan CSV.) Tanpa pengaman di baris 26-27, galatnya hanya `KeyError: 'P-99'` di baris 30, yang tidak menyebut berkas mana yang bermasalah. Pesan yang menyebut kode dan berkas membuat Anda langsung tahu apa yang harus dibetulkan: tambahkan petaknya ke GeoPackage, atau hapus barisnya dari CSV.
- **Hanya satu hari data.** Jarak antartitik grafik dibagi `max(len(hari) - 1, 1)` di baris 50. Satu hari berarti `len(hari) - 1` sama dengan 0, dan pembagian nol menimbulkan `ZeroDivisionError`. Angka `1` dari `max` mencegahnya: fungsi `garis_svg` yang diuji dengan satu hari menghasilkan gambar berisi satu titik. Satu titik tidak tampak sebagai garis, jadi grafiknya kosong secara kasat mata. Ini diuji di peramban: kotak grafik tampil kosong, hanya dua label tanggal yang tertinggal, dan keduanya sama (misalnya "21 Sep" di kiri dan di kanan).
- **Label tanggal membaca bentuk tetap.** `label_tgl` mengambil huruf ke-9 dan ke-10 sebagai hari, ke-6 dan ke-7 sebagai bulan. Maka kolom `Tanggal` harus berbentuk `2026-09-30`. Bentuk lain, misalnya `30/09/2026`, menghasilkan label yang salah atau galat.
- **Sebagian tulisan masih tetap.** Label grafik dan nama bulan di kartu kini mengikuti data. Namun judul kartu "Total titik panas (30 hari)" (baris 68) dan judul halaman "titik panas September" (baris 76) tetap tertulis. Untuk data Oktober, ubah keduanya sendiri.

Latihan variasi.

1. Cetak **lima** petak teratas, bukan tiga.
2. Ubah warna garis menjadi hijau tua `#2E7D32`.

Contoh jawaban, hanya bagian yang berubah:

```python
    teratas = sorted(per_petak.items(), key=lambda kv: (-kv[1], kv[0]))[:5]
```

Baris 51 ditulis ulang dengan `stroke="#2E7D32"`. Hasil uji: JSON memuat lima petak, yaitu P-14 (33), P-12 (31), P-13 (31), P-16 (30), dan P-15 (28). Berkas HTML memuat `stroke="#2E7D32"`.

> **Pola yang bisa dipakai ulang:** Kerangka dasbor: **baca data, jumlahkan ke kamus, ubah angka menjadi teks gambar, sisipkan ke cetakan HTML, tulis berkas**. Untuk laporan lain, ganti tiga hal: kolom yang dijumlahkan (baris 25-30), judul kartu (baris 68-69), dan teks cetakan (baris 70-80).

Skrip 27.2: Peta web dari data QGIS

Berkas `buat_peta_web.py`. Bagi pembaca buku M4, ini skrip Bab 3.

Tujuan. Mengekspor petak dan titik panas menjadi GeoJSON berkoordinat lintang-bujur, lalu menulis halaman peta web (Leaflet) yang menggambarnya.

Bahan. `paket-m4/Petak_KPH.gpkg` dan `paket-data/Hotspot_Nasional.gpkg`. Skrip memakai QGIS penuh, jadi jalankan dengan `python-qgis.bat`.

Analogi: GeoJSON adalah **lembar pengumuman**, Leaflet adalah **papan** yang menempelkannya. Papannya tidak ikut dalam skrip. Papan itu diambil dari internet saat halaman dibuka.

> **Catatan: halaman ini memuat Leaflet dari internet:** Di baris 42 dan 45, halaman memuat Leaflet versi 1.9.4 dari layanan publik `unpkg.com`. Karena itu halaman perlu internet agar petanya tampil. Skrip hanya menulis berkas di komputer Anda dan tidak mengunggah apa pun. Halaman hasilnya sudah dibuka di peramban lewat server lokal: yang tampak 16 petak berupa poligon biru dan 12 titik panas berupa lingkaran merah, tanpa peta dasar. Membuka berkas dengan klik dua kali tidak diuji.

Skrip utuh:

```python
# -*- coding: utf-8 -*-
"""Membuat web map statis (Leaflet + GeoJSON) dari data KPH Contoh. Penulis: Badar Mubarok Yogaswara
Hanya membuat berkas lokal di folder keluaran/peta_web. Tidak mengunggah apa pun.
Jalankan: python-qgis.bat buat_peta_web.py
"""
import json, math, os
os.environ.setdefault("QT_QPA_PLATFORM", "offscreen")
from qgis.core import (QgsApplication, QgsVectorLayer, QgsCoordinateReferenceSystem,
                       QgsVectorFileWriter, QgsCoordinateTransform, QgsCoordinateTransformContext)

AQ = os.path.dirname(os.path.abspath(__file__))
DATA = os.path.abspath(os.path.join(AQ, "..", "..", "..", "data-sintetis"))
OUT = os.path.join(AQ, "keluaran", "peta_web")


def ke_tile(lon, lat, zoom):
    """Nomor ubin XYZ (skema web: x ke timur, y ke selatan) untuk satu titik."""
    n = 2 ** zoom
    x = int((lon + 180.0) / 360.0 * n)
    y = int((1.0 - math.asinh(math.tan(math.radians(lat))) / math.pi) / 2.0 * n)
    return x, y


def ekspor(layer_path, nama_layer, berkas):
    lyr = QgsVectorLayer(layer_path, nama_layer, "ogr")
    assert lyr.isValid(), layer_path
    opsi = QgsVectorFileWriter.SaveVectorOptions()
    opsi.driverName = "GeoJSON"
    opsi.fileEncoding = "UTF-8"
    opsi.ct = QgsCoordinateTransform(
        lyr.crs(), QgsCoordinateReferenceSystem("EPSG:4326"), QgsCoordinateTransformContext())
    opsi.layerOptions = ["COORDINATE_PRECISION=6", "RFC7946=YES"]
    hasil = QgsVectorFileWriter.writeAsVectorFormatV3(lyr, berkas, QgsCoordinateTransformContext(), opsi)
    assert hasil[0] == QgsVectorFileWriter.WriterError.NoError, hasil
    return lyr.featureCount()
```

```
HTML = """<!doctype html>
<html lang="id"><head><meta charset="utf-8">
<meta name="viewport" content="width=device-width, initial-scale=1">
<title>Peta KPH Contoh</title>
<link rel="stylesheet" href="https://unpkg.com/leaflet@1.9.4/dist/leaflet.css">
<style>html,body,#peta{height:100%;margin:0}</style></head>
<body><div id="peta"></div>
<script src="https://unpkg.com/leaflet@1.9.4/dist/leaflet.js"></script>
<script>
var peta = L.map('peta');
fetch('petak.geojson').then(r => r.json()).then(function (d) {
  var lap = L.geoJSON(d, {
    style: {color: '#2B6CB0', weight: 2, fillOpacity: 0.15},
    onEachFeature: function (f, l) {
      l.bindPopup('<b>' + f.properties.ID_Petak + '</b><br>' + f.properties.NAMA_KPH +
                  '<br>' + f.properties.Jenis + ', ' + f.properties.Luas_Ha + ' ha');
    }
  }).addTo(peta);
  peta.fitBounds(lap.getBounds());
});
fetch('hotspot.geojson').then(r => r.json()).then(function (d) {
  L.geoJSON(d, {pointToLayer: function (f, p) {
    return L.circleMarker(p, {radius: 6, color: '#C0392B', fillOpacity: 0.8});
  }}).addTo(peta);
});
</script></body></html>
"""

if __name__ == "__main__":
    app = QgsApplication([], False)
    app.initQgis()
    os.makedirs(OUT, exist_ok=True)
    n1 = ekspor(os.path.join(DATA, "paket-m4", "Petak_KPH.gpkg"), "Petak_KPH", os.path.join(OUT,
"petak.geojson"))
    n2 = ekspor(os.path.join(DATA, "paket-data", "Hotspot_Nasional.gpkg"), "Hotspot", os.path.join(OUT,
"hotspot.geojson"))
    with open(os.path.join(OUT, "index.html"), "w", encoding="utf-8") as fh:
        fh.write(HTML)
    print("petak:", n1, "hotspot:", n2)
    for b in ("petak.geojson", "hotspot.geojson"):
        d = json.load(open(os.path.join(OUT, b), encoding="utf-8"))
        c = d["features"][0]["geometry"]["coordinates"]
        while isinstance(c[0], list):
            c = c[0]
        print(b, "fitur:", len(d["features"]), "koordinat pertama (lon, lat):", c)
    print("ubin zoom 0 untuk (0,0):", ke_tile(0, 0, 0), "| zoom 1 untuk (0,0):", ke_tile(0, 0, 1),
          "| zoom 10 Jakarta -6.2,106.85:", ke_tile(106.85, -6.2, 10))
    app.exitQgis()
```

Penjelasan baris demi baris. Bagian pertama: perkakas dan penghitung ubin (baris 1-21).

Baris	Tulisan	Artinya
6	`import json, math, os`	Tiga perkakas: JSON, hitungan matematika, dan jalur folder.
7	`os.environ.setdefault("QT_QPA_PLATFORM", "offscreen")`	Pengaturan agar QGIS jalan tanpa membuka jendela. `setdefault` hanya mengisi bila belum ada isinya.
8-9	`from qgis.core import (...)`	Panggil enam perkakas QGIS. Tanda kurung memungkinkan daftar dipecah ke dua baris.
11-13	`AQ, DATA, OUT`	Sama seperti Skrip 27.1: lokasi skrip, folder `data-sintetis` (naik tiga folder), dan folder keluaran `peta_web`.
16-21	`def ke_tile(lon, lat, zoom)`	Hitung nomor ubin peta web untuk satu titik. Ubin adalah kepingan gambar peta dasar. Fungsi ini hanya dicetak sebagai contoh di akhir skrip dan tidak dipakai oleh halaman peta.

Bagian kedua: mengekspor (baris 24-35).

Baris	Tulisan	Artinya
24-26	`def ekspor(...), lyr = QgsVectorLayer(..., "ogr"), assert lyr.isValid(), layer_path`	Buka layer. `assert` menghentikan skrip bila layer gagal dibuka, dan mencetak jalurnya (Bab 10).
27-29	`opsi = QgsVectorFileWriter.SaveVectorOptions()` dst.	Siapkan "formulir pilihan" penulisan: format GeoJSON, huruf UTF-8.
30-31	`opsi.ct = QgsCoordinateTransform(lyr.crs(), QgsCoordinateReferenceSystem("EPSG:4326"), ...)`	Atur perubahan koordinat dari CRS layer (UTM) ke EPSG:4326 (lintang-bujur), karena GeoJSON standar memakainya.
32	`opsi.layerOptions = [...]`	Pilihan tambahan: ketelitian enam desimal dan mengikuti standar GeoJSON (RFC 7946).
33-35	`writeAsVectorFormatV3(...), assert ..., return lyr.featureCount()`	Tulis berkas. Hasilnya diperiksa: bukan `NoError` berarti gagal. Fungsi mengembalikan jumlah fitur.

Bagian ketiga: halaman HTML (baris 38-64). Teksnya panjang, tetapi hanya ada empat hal yang perlu dikenali.

Baris	Tulisan	Artinya
42, 45	`<link ... leaflet.css>` dan `<script src="...leaflet.js">`	Dua alamat internet tempat Leaflet diambil.
47	`var peta = L.map('peta');`	Siapkan papan peta kosong. Ini JavaScript, bukan Python.
48-57	`fetch('petak.geojson')...`	Ambil berkas petak, gambar sebagai poligon biru, beri jendela info (*popup*) berisi ID, KPH, jenis, dan luas, lalu zoom ke seluruh petak.
58-62	`fetch('hotspot.geojson')...`	Ambil titik panas lalu gambar sebagai lingkaran merah berjari-jari 6.

Bagian keempat: menjalankan (baris 66-83).

Baris	Tulisan	Artinya
67-68	`app = QgsApplication([], False)` dan `app.initQgis()`	Nyalakan mesin QGIS tanpa jendela.
69-71	`os.makedirs(...), n1 = ekspor(...), n2 = ekspor(...)`	Buat folder, lalu ekspor dua layer.
72-73	`with open(... "index.html" ...)`	Tulis teks `HTML` ke `index.html`.
75-80	`for b in ("petak.geojson", "hotspot.geojson"):`	Baca balik kedua berkas dan cetak koordinat pertama sebagai pemeriksaan. `while isinstance(c[0], list)` turun lapis demi lapis sampai menemukan angka.
81-82	`print("ubin zoom 0 ...", ke_tile(...))`	Cetak tiga contoh nomor ubin.
83	`app.exitQgis()`	Matikan mesin QGIS.

Hasil. Layar menampilkan (berdasarkan uji pada QGIS 4.0.2):

```
HASIL
petak: 16 hotspot: 12
petak.geojson fitur: 16 koordinat pertama (lon, lat): [109.310738, -0.036173]
hotspot.geojson fitur: 12 koordinat pertama (lon, lat): [109.31149, -0.03301]
ubin zoom 0 untuk (0,0): (0, 0) | zoom 1 untuk (0,0): (1, 1) | zoom 10 Jakarta -6.2,106.85: (815, 529)
```

Dari hasil ini Anda tahu: ada 16 petak dan 12 titik panas, koordinatnya sudah berupa lintang-bujur, dan fungsi `ke_tile` bekerja. Tiga berkas terbentuk di `keluaran/peta_web`: `petak.geojson`, `hotspot.geojson`, dan `index.html`.

Untuk membuka halaman, buku M4 menyarankan melayani folder itu lewat server kecil di komputer sendiri, karena banyak peramban menolak membaca berkas lokal lewat `fetch` bila halaman dibuka

langsung dengan klik dua kali. Itu perilaku peramban, tidak saya uji. [CEK] Bab ini tidak menjalankan server apa pun.

Bagian yang boleh diubah.

- Baris 32, `COORDINATE_PRECISION=6`. Jumlah desimal koordinat. Lebih kecil berarti berkas lebih ringan, tetapi letak kurang teliti.
- **Baris 50, gaya petak.** `color`, `weight` (tebal garis), dan `fillOpacity` (kepekatan isi).
- Baris 60, `radius: 6` dan `color`. Ukuran dan warna titik panas.
- **Baris 52-53, isi popup.** Ganti dengan kolom lain, misalnya `f.properties.Tahun_Tanam`. Nama kolom ditulis persis.
- **Baris 70-71, berkas sumber.** Ganti untuk memetakan layer lain.

Jebakan.

- **Jalur data salah.** Galatnya `AssertionError: ...\data-sintetis\paket-m4\Petak_KPH.gpkg`. Jalur yang gagal ikut tercetak berkat pesan di baris 26. (Diuji dengan menaruh skrip di folder tanpa `data-sintetis`.)
- **Butuh internet.** Tanpa internet, Leaflet tidak termuat dan halaman kosong.
- **Tanpa peta dasar.** JavaScript hanya menggambar petak dan titik. Latar peta (jalan, sungai) tidak ada.
- **Nama kolom di JavaScript.** `f.properties.Jenis` harus sama dengan nama kolom di GeoJSON, termasuk huruf besar dan kecil.

Latihan variasi.

1. Kurangi ketelitian koordinat menjadi **tiga** desimal.
2. Ganti warna petak menjadi hijau tua `#1B5E20` dan jari-jari titik panas menjadi 10.

Contoh jawaban, hanya bagian yang berubah:

```python
    opsi.layerOptions = ["COORDINATE_PRECISION=3", "RFC7946=YES"]
```

Pada HTML (baris 50 dan 60), tulis `color: '#1B5E20'` dan `radius: 10`. Hasil uji: koordinat pertama petak menjadi `[109.311, -0.036]` dan titik panas `[109.311, -0.033]`. Berkas `index.html` memuat `'#1B5E20'` dan `radius: 10`.

> **Pola yang bisa dipakai ulang:** Kerangka peta web: **buka layer, atur transformasi ke EPSG:4326, tulis GeoJSON, tulis HTML yang memuat GeoJSON itu**. Untuk data lain, ganti nama berkas sumber (baris 70-71), kolom popup (baris 52-53), dan warna (baris 50 dan 60).

Skrip 27.3: Model 3D dari DSM

Berkas `dsm_ke_obj.py`. Bagi pembaca buku M4, ini skrip Bab 5.

Tujuan. Mengubah DSM drone (peta tinggi permukaan) menjadi jaring segitiga 3D berformat `.obj`, lalu memeriksa titik awan LAZ dan pengaturan peta 3D QGIS.

Bahan. `paket-data/DSM_Drone.tif` (800 x 800 piksel, 0,5 m) dan `Hutan_Contoh.laz` (15.000 titik). Jalankan dengan `python-qgis.bat`.

Analogi: model 3D adalah **maket dari karton**. Titik sudut adalah paku-pakunya. Segitiga adalah lembar karton yang menghubungkan tiga paku. Makin banyak paku, makin halus maketnya, makin berat pula beratnya.

Skrip utuh:

```python
# -*- coding: utf-8 -*-
"""Mengubah DSM menjadi jaring segitiga 3D (.obj) dan memeriksa pengaturan 3D QGIS. Penulis: Badar Mubarok Yogaswara
Jalankan: python-qgis.bat dsm_ke_obj.py
"""
import os
os.environ.setdefault("QT_QPA_PLATFORM", "offscreen")
from osgeo import gdal
from qgis.core import (QgsApplication, QgsRasterLayer, QgsPointCloudLayer, QgsProject)
gdal.UseExceptions()
AQ = os.path.dirname(os.path.abspath(__file__))
DATA = os.path.abspath(os.path.join(AQ, "..", "..", "..", "data-sintetis", "paket-data"))
OUT = os.path.join(AQ, "keluaran", "tiga_d")


def dsm_ke_obj(tif, obj, langkah=16, skala_z=2.0):
    """Ambil tiap `langkah` piksel; x,y relatif terhadap sudut barat-daya (meter); z dikali skala_z."""
    ds = gdal.Open(tif)
    gt = ds.GetGeoTransform()
    band = ds.GetRasterBand(1)
    data = band.ReadAsArray()
    nodata = band.GetNoDataValue()
    baris = list(range(0, ds.RasterYSize, langkah))
    kolom = list(range(0, ds.RasterXSize, langkah))
    idx = {}
    n = 0
    with open(obj, "w", encoding="utf-8") as fh:
        fh.write("# DSM sintetis KPH Contoh; skala vertikal x%.1f\n" % skala_z)
        for j, r in enumerate(baris):
            for i, c in enumerate(kolom):
                z = float(data[r, c])
                if nodata is not None and z == nodata:
                    z = 0.0
                x = i * langkah * gt[1]
                y = (len(baris) - 1 - j) * langkah * abs(gt[5])    # utara di atas: baris 0 = utara
                n += 1
                idx[(j, i)] = n
                fh.write("v %.3f %.3f %.3f\n" % (x, z * skala_z, y))
        muka = 0
        for j in range(len(baris) - 1):
            for i in range(len(kolom) - 1):
                a, b, c, d = idx[(j, i)], idx[(j, i + 1)], idx[(j + 1, i + 1)], idx[(j + 1, i)]
                fh.write("f %d %d %d\nf %d %d %d\n" % (a, d, c, a, c, b))
                muka += 2
    return n, muka, float(data.min()), float(data.max())


if __name__ == "__main__":
    app = QgsApplication([], False)
    app.initQgis()
    os.makedirs(OUT, exist_ok=True)
    v, f, zmin, zmax = dsm_ke_obj(os.path.join(DATA, "DSM_Drone.tif"), os.path.join(OUT, "dsm.obj"))
    print("OBJ: titik sudut", v, "segitiga", f, "| z min/maks DSM:", round(zmin, 2), round(zmax, 2))
    vs = [l.split()[1:] for l in open(os.path.join(OUT, "dsm.obj"), encoding="utf-8") if l.startswith("v ")]
```

```
    xs, ys = [float(a[0]) for a in vs], [float(a[1]) for a in vs]
    print("OBJ: x dari %.0f ke %.0f m | tinggi (sudah x2) dari %.2f ke %.2f" % (min(xs), max(xs), min(ys),
max(ys)))
    r = QgsRasterLayer(os.path.join(DATA, "DSM_Drone.tif"), "DSM")
    print("DSM valid:", r.isValid(), "ukuran:", r.width(), "x", r.height(), "CRS:", r.crs().authid())
    pc = QgsPointCloudLayer(os.path.join(DATA, "Hutan_Contoh.laz"), "Hutan", "pdal")
    print("point cloud valid:", pc.isValid(), "jumlah titik:", pc.pointCount())
    from qgis._3d import Qgs3DMapSettings, QgsDemTerrainSettings
    s = Qgs3DMapSettings()
    s.setCrs(r.crs())
    s.setExtent(r.extent())
    t = QgsDemTerrainSettings()
    t.setLayer(r)
    t.setVerticalScale(2.0)
    s.setTerrainSettings(t)
    print("3D: terrain =", s.terrainSettings().type() if hasattr(s.terrainSettings(), "type") else "?",
          "| skala vertikal:", s.terrainSettings().verticalScale(), "| ekstensi:", s.extent().toString(1))
    app.exitQgis()
```

Penjelasan baris demi baris. Bagian pertama: persiapan (baris 5-12).

Baris	Tulisan	Artinya
5-6	`import os` dan `os.environ.setdefault(...)`	Perkakas jalur, dan pengaturan QGIS tanpa jendela (sama seperti Skrip 27.2).
7-9	`from osgeo import gdal`, `from qgis.core import (...)`, `gdal.UseExceptions()`	Panggil GDAL untuk membaca raster, dan empat perkakas QGIS. `QgsProject` ikut dipanggil tetapi tidak dipakai.
10-12	`AQ`, `DATA`, `OUT`	Lokasi skrip, folder `paket-data` (naik tiga folder), dan folder keluaran `tiga_d`.

Bagian kedua: fungsi pembuat `.obj` (baris 15-44). Ia bekerja dalam dua langkah: tulis semua titik sudut, lalu tulis segitiganya.

Baris	Tulisan	Artinya
15	`def dsm_ke_obj(tif, obj, langkah=16, skala_z=2.0)`	Fungsi dengan dua pengaturan bawaan: ambil satu titik tiap 16 piksel, dan tinggi dikali 2 agar bukit tampak jelas.
17-21	`gdal.Open(tif)` dst.	Buka raster. `gt` adalah ukuran piksel dan letaknya. `data` adalah larik angka tinggi. `nodata` adalah angka penanda "tidak ada data".
22-23	`baris = list(range(0, ds.RasterYSize, langkah))`	Daftar nomor baris yang dipilih: 0, 16, 32, dan seterusnya. Kolom dibuat sama.
24-25	`idx = {}` dan `n = 0`	`idx` mencatat nomor urut tiap titik sudut. `n` penghitungnya.
26-27	`with open(obj, "w", ...)` dan `fh.write(...)`	Buka berkas `.obj` dan tulis baris komentar di awalnya.
28-30	`for j, r in enumerate(baris):` lalu `for i, c in enumerate(kolom):`	Dua perulangan bersarang menyapu baris dan kolom terpilih. `z` adalah tinggi di titik itu.
31-32	`if nodata is not None and z == nodata: z = 0.0`	Bila piksel tak berdata, tingginya dianggap 0.

Baris	Tulisan	Artinya
33-34	`x = ...` dan `y = ...`	Hitung letak datar dalam meter, dihitung dari sudut barat daya. Baris 0 raster adalah utara, jadi nilainya dibalik.
35-37	`n += 1`, `idx[(j, i)] = n`, `fh.write("v %.3f %.3f %.3f\n" ...)`	Beri nomor, catat di kamus (kuncinya pasangan `(j, i)`), dan tulis satu baris `v` (titik sudut).
38-43	`for j in range(len(baris) - 1)` dst.	Untuk tiap petak persegi empat titik, ambil keempat nomornya (`a, b, c, d`), lalu tulis dua baris `f` (segitiga).
44	`return n, muka, float(data.min()), float(data.max())`	Kembalikan empat nilai: jumlah titik, jumlah segitiga, tinggi terendah dan tertinggi.

Bagian ketiga: menjalankan dan memeriksa (baris 47-70).

Baris	Tulisan	Artinya
48-50	`app = QgsApplication([], False)`, `app.initQgis()`, `os.makedirs(...)`	Nyalakan QGIS tanpa jendela, siapkan folder.
51-52	`v, f, zmin, zmax = dsm_ke_obj(...)` lalu `print(...)`	Panggil fungsi. Empat nilai hasilnya dibongkar ke empat kotak.
53-55	`vs = [l.split()[1:] for l in open(...) if l.startswith("v ")]`	Baca balik berkas `.obj`: ambil hanya baris `v`, buang kata `v`. Huruf `l` mudah dikira angka 1, sebaiknya diganti.
56-57	`r = QgsRasterLayer(...)`	Buka DSM sebagai layer QGIS lalu cetak ukuran dan CRS-nya.
58-59	`QgsPointCloudLayer(..., "pdal")`	Buka titik awan LAZ lalu cetak jumlah titiknya. Pembaca `"pdal"` harus tersedia di QGIS Anda.
60-69	`from qgis._3d import ...` dst.	Buat pengaturan peta 3D: CRS dan batas dari DSM, medan dari DSM dengan skala tinggi 2. Baris 68 mencetak jenis medan dengan `hasattr` sebagai pengaman.

Hasil. Layar menampilkan (berdasarkan uji pada QGIS 4.0.2):

```
HASIL
OBJ: titik sudut 2500 segitiga 4802 | z min/maks DSM: 15.03 38.8
OBJ: x dari 0 ke 392 m | tinggi (sudah x2) dari 33.26 ke 64.42
DSM valid: True ukuran: 800 x 800 CRS: EPSG:32749
point cloud valid: True jumlah titik: 15000
3D: terrain = dem | skala vertikal: 2.0 | ekstensi: 312000.0,9996000.0 : 312400.0,9996400.0
```

Berkas `keluaran/tiga_d/dsm.obj` berukuran 143.790 byte. Lima baris pertamanya:

```
HASIL
# DSM sintetis KPH Contoh; skala vertikal x2.0
v 0.000 44.025 392.000
v 8.000 44.822 392.000
```

```
v 16.000 45.605 392.000
v 24.000 46.359 392.000
```

Perhatikan: angka kedua tiap baris `v` adalah **tinggi**, bukan utara. Format OBJ menaruh tinggi di sumbu kedua. Berkas ini bisa dibuka di program 3D yang membaca `.obj`. [CEK: tidak dibuka pada uji ini]

Dari hasil ini Anda juga bisa menghitung sendiri: 800 piksel dibagi 16 menjadi 50 titik per sisi, jadi 50 x 50 = 2.500 titik sudut. Segitiganya 2 x 49 x 49 = 4.802.

Bagian yang boleh diubah.

- Baris 15, `langkah=16`. Makin kecil, jaring makin rapat dan berkas makin besar.
- Baris 15, `skala_z=2.0`. Pengali tinggi. 1,0 berarti tinggi asli. Baris 66 punya angka 2.0 sendiri untuk peta 3D, ubah juga bila perlu.
- **Baris 51, nama berkas.** Ganti `DSM_Drone.tif` dengan DSM Anda. Pastikan band 1 adalah tinggi.
- Baris 32, `z = 0.0`. Nilai pengganti untuk piksel tanpa data.

Jebakan.

- **Jalur salah.** Galatnya `RuntimeError: ...\data-sintetis\paket-data\DSM_Drone.tif: No such file or directory`. (Diuji.)
- **NoData jadi nol.** Lubang data berubah menjadi dasar datar bertinggi 0, padahal sekitarnya bisa 30 m.
- **Langkah terlalu besar.** Detail kecil hilang karena banyak piksel dilewati.
- **Sumbu tertukar.** Di OBJ, tinggi ada di sumbu kedua. Bila dibuka di program lain, periksa arah "atas".

Latihan variasi.

1. Jadikan jaring lebih jarang: `langkah=32`.
2. Gunakan tinggi asli: `skala_z=1.0`.

Contoh jawaban, hanya bagian yang berubah:

```
PYTHON
def dsm_ke_obj(tif, obj, langkah=32, skala_z=1.0):
```

Hasil uji: untuk `langkah=32`, titik sudut 625 dan segitiga 1.152, berkas jauh lebih kecil (32.651 byte). Untuk `skala_z=1.0`, tinggi berkisar 16,63 sampai 32,21 dan baris `v` kedua menjadi `v 0.000 22.012 392.000`.

> **Pola yang bisa dipakai ulang:** Kerangka raster ke 3D: baca larik tinggi, ambil tiap n piksel, tulis titik `v`, lalu tulis segitiga `f` dari empat tetangga. Untuk DSM lain, ganti berkas (baris 51), `langkah`, dan `skala_z`.

Skrip 27.4: Alat Processing buatan sendiri, gaya kelas

Berkas `kerapatan_hotspot.py`. Bagi pembaca buku M4, ini skrip Bab 2.

Tujuan. Membuat alat baru di Processing Toolbox yang menghitung jumlah titik panas dan kerapatannya (titik per hektare) di tiap petak, lalu menandai petak yang melewati batas.

Bahan. `paket-m4/Petak_KPH.gpkg` (poligon, CRS UTM) dan `paket-data/Hotspot_Nasional.gpkg` (12 titik).

Analogi: alat Processing adalah **formulir resmi**. Siapa pun yang mengisi isiannya akan mendapat hasil yang sama. Gaya kelas adalah formulir **lengkap dengan semua kolom tertulis**. Kelas (*class*) dibahas di Bab 11: sebuah cetakan benda dengan metode, yaitu fungsi yang melekat padanya.

Alat ini bekerja dalam tiga tahap: **identitas** (nama dan kelompok), **formulir** (parameter), dan **pekerjaan** (isi). Tiga tahap itu juga muncul di alat lain.

Skrip utuh:

```python
# -*- coding: utf-8 -*-
"""Processing script: kerapatan titik panas per petak. Penulis: Badar Mubarok Yogaswara"""
from qgis.core import (QgsProcessingAlgorithm, QgsProcessingParameterFeatureSource,
                       QgsProcessingParameterFeatureSink, QgsProcessingParameterNumber,
                       QgsProcessing, QgsFeatureSink, QgsFeature, QgsField, QgsFields,
                       QgsFeatureRequest, QgsCoordinateTransform, QgsProject,
                       QgsProcessingException, QgsSpatialIndex, Qgis)
from qgis.PyQt.QtCore import QMetaType


class KerapatanHotspot(QgsProcessingAlgorithm):
    PETAK = "PETAK"
    TITIK = "TITIK"
    BATAS = "BATAS"
    HASIL = "HASIL"

    def createInstance(self):
        return KerapatanHotspot()

    def name(self):
        return "kerapatan_hotspot"

    def displayName(self):
        return "Kerapatan titik panas per petak"

    def group(self):
        return "Kehutanan"

    def groupId(self):
        return "kehutanan"

    def shortHelpString(self):
        return ("Menghitung jumlah titik panas di tiap petak dan kerapatannya (titik per hektare). "
                "Petak dengan kerapatan di atas batas diberi nilai Rawan = 1.")

    def initAlgorithm(self, config=None):
        self.addParameter(QgsProcessingParameterFeatureSource(
            self.PETAK, "Layer petak (poligon)", [QgsProcessing.SourceType.TypeVectorPolygon]))
        self.addParameter(QgsProcessingParameterFeatureSource(
            self.TITIK, "Layer titik panas (titik)", [QgsProcessing.SourceType.TypeVectorPoint]))
        self.addParameter(QgsProcessingParameterNumber(
            self.BATAS, "Batas rawan (titik per hektare)",
            type=QgsProcessingParameterNumber.Type.Double, defaultValue=2.0, minValue=0.0))
```

```
        self.addParameter(QgsProcessingParameterFeatureSink(self.HASIL, "Petak dengan kerapatan"))

    def processAlgorithm(self, parameters, context, feedback):
        petak = self.parameterAsSource(parameters, self.PETAK, context)
        titik = self.parameterAsSource(parameters, self.TITIK, context)
        batas = self.parameterAsDouble(parameters, self.BATAS, context)
        if petak is None or titik is None:
            raise QgsProcessingException("Layer petak atau titik tidak terbaca.")
        if petak.sourceCrs().mapUnits() != Qgis.DistanceUnit.Meters:
            raise QgsProcessingException("CRS layer petak harus bersatuan meter (luas dibagi 10000 untuk
hektare). Proyeksikan ulang dulu, misalnya ke UTM.")

        # titik disamakan ke CRS petak lebih dulu
        ke_petak = QgsCoordinateTransform(titik.sourceCrs(), petak.sourceCrs(), context.transformContext())
        indeks = QgsSpatialIndex()
        geom = {}
        for i, f in enumerate(titik.getFeatures()):
            g = f.geometry()
            g.transform(ke_petak)
            fb = QgsFeature(i)
            fb.setGeometry(g)
            indeks.addFeature(fb)
            geom[i] = g

        kolom = QgsFields(petak.fields())
        kolom.append(QgsField("Jml_Titik", QMetaType.Type.Int))
        kolom.append(QgsField("Titik_per_Ha", QMetaType.Type.Double))
        kolom.append(QgsField("Rawan", QMetaType.Type.Int))
        sink, dest = self.parameterAsSink(parameters, self.HASIL, context, kolom,
                                          petak.wkbType(), petak.sourceCrs())
        if sink is None:
            raise QgsProcessingException("Keluaran tidak bisa dibuat.")

        total = petak.featureCount() or 1
        for n, f in enumerate(petak.getFeatures()):
            if feedback.isCanceled():
                break
            pg = f.geometry()
            kandidat = indeks.intersects(pg.boundingBox())
            jumlah = sum(1 for k in kandidat if pg.contains(geom[k]))
            luas_ha = pg.area() / 10000.0
            kerapatan = jumlah / luas_ha if luas_ha > 0 else 0.0
            baru = QgsFeature(kolom)
            baru.setGeometry(pg)
            baru.setAttributes(f.attributes() + [jumlah, round(kerapatan, 3), 1 if kerapatan > batas else 0])
            sink.addFeature(baru, QgsFeatureSink.Flag.FastInsert)
            feedback.setProgress(100.0 * (n + 1) / total)
        return {self.HASIL: dest}
```

Penjelasan baris demi baris. Bagian pertama: perkakas dan identitas (baris 1-34).

Baris	Tulisan	Artinya
3-8	`from qgis.core import (...)` dan `QMetaType`	Panggil 15 perkakas QGIS. Dua di antaranya, `QgsFeatureRequest` dan `QgsProject`, tidak dipakai. Yang terakhir, `Qgis`, dipakai untuk memeriksa satuan peta di baris 52.
11	`class KerapatanHotspot(QgsProcessingAlgorithm):`	Buat kelas baru yang **mewarisi** semua sifat algoritma Processing bawaan. Tanda kurung berarti "turunan dari".
12-15	`PETAK = "PETAK"` dst.	Empat nama tetap untuk isian formulir. Dipakai berulang di bawah agar tidak salah ketik.

Baris	Tulisan	Artinya
17-18	`def createInstance(self)`	Pabrik: setiap kali QGIS butuh satu salinan alat, ia memanggil ini. `self` adalah benda yang sedang dibicarakan.
20-24	`name` dan `displayName`	`name` adalah nama teknis untuk Python. `displayName` adalah nama yang terlihat di Toolbox.
26-30	`group` dan `groupId`	Nama kelompok tempat alat muncul ("Kehutanan"), dan kodenya.
32-34	`shortHelpString`	Teks bantuan singkat di kolom kanan jendela alat.

Bagian kedua: formulir (baris 36-44).

Baris	Tulisan	Artinya
36	`def initAlgorithm(self, config=None):`	Metode yang QGIS panggil untuk membangun formulir.
37-38	`QgsProcessingParameterFeatureSource(self.PETAK, "Layer petak (poligon)", [...TypeVectorPolygon])`	Isian layer masukan. Teks di tengah adalah label di formulir. Daftar di ujung membatasi jenis layer: poligon.
39-40	idem untuk `TITIK`	Isian layer titik panas, jenis titik.
41-43	`QgsProcessingParameterNumber(... defaultValue=2.0, minValue=0.0)`	Isian angka: bawaan 2,0 titik per hektare, tidak boleh di bawah 0.
44	`QgsProcessingParameterFeatureSink(self.HASIL, ...)`	Isian keluaran. *Sink* adalah tempat hasil ditulis.

Bagian ketiga: persiapan pekerjaan (baris 46-74).

Baris	Tulisan	Artinya
46	`def processAlgorithm(self, parameters, context, feedback):`	Metode inti. `parameters` berisi isian pengguna, `context` berisi pengaturan QGIS, `feedback` adalah saluran pesan dan kemajuan.
47-49	`self.parameterAsSource(...)` dan `parameterAsDouble(...)`	Ambil isian formulir menjadi benda Python: dua layer dan satu angka.
50-51	`if petak is None or titik is None: raise QgsProcessingException(...)`	Bila layer tidak terbaca, hentikan dengan pesan jelas. `raise` melempar galat (Bab 10).
52-53	`if petak.sourceCrs().mapUnits() != Qgis.DistanceUnit.Meters: raise QgsProcessingException(...)`	Pengaman satuan peta. `mapUnits()` menyebut satuan CRS layer petak: meter atau derajat. Bila bukan meter, alat berhenti dengan pesan yang meminta petak diproyeksikan ulang, misalnya ke UTM. Alasannya: baris 83 membagi luas dengan 10000, dan itu hanya benar bila luas dalam meter persegi.
56	`ke_petak = QgsCoordinateTransform(...)`	Siapkan "penerjemah" CRS dari titik ke petak.

Baris	Tulisan	Artinya
57-65	`indeks = QgsSpatialIndex()` dan `for i, f in enumerate(titik.getFeatures())`	Bangun indeks spasial: semacam daftar isi yang mempercepat pencarian titik. Tiap titik diterjemahkan CRS-nya, dimasukkan ke indeks, dan geometrinya disimpan di kamus `geom`.
67-70	`kolom = QgsFields(petak.fields())` lalu tiga `kolom.append(...)`	Salin kolom petak, tambah tiga kolom baru: `Jml_Titik`, `Titik_per_Ha`, `Rawan`.
71-72	`sink, dest = self.parameterAsSink(...)`	Minta QGIS menyiapkan tempat keluaran dengan kolom itu. Dua nilai dikembalikan: sink untuk menulis, `dest` untuk laporan.
73-74	`if sink is None: raise QgsProcessingException(...)`	Bila tempat keluaran gagal dibuat, hentikan dengan pesan jelas.

Bagian keempat: pekerjaan inti (baris 76-90).

Baris	Tulisan	Artinya
76	`total = petak.featureCount() or 1`	Jumlah petak. `or 1` mencegah pembagian nol di baris 89 bila layer kosong.
77-79	`for n, f in enumerate(petak.getFeatures()):` dan `if feedback.isCanceled(): break`	Telusuri tiap petak. Bila pengguna menekan Batal, berhenti.
81	`kandidat = indeks.intersects(pg.boundingBox())`	Tanya indeks: titik mana yang jatuh di dalam kotak pembatas petak. Ini saringan kasar.
82	`jumlah = sum(1 for k in kandidat if pg.contains(geom[k]))`	Saringan halus: hitung hanya titik yang benar-benar di dalam poligon.
83-84	`luas_ha` dan `kerapatan`	Luas dalam hektare (dibagi 10000). Kerapatan adalah jumlah dibagi luas, dengan pengaman bila luas 0.
85-87	`baru = QgsFeature(kolom)`, `setGeometry`, `setAttributes(...)`	Rakit fitur baru: bentuk lama, atribut lama ditambah tiga nilai baru. Kolom `Rawan` bernilai 1 bila kerapatan lebih besar dari batas, selain itu 0.
88-89	`sink.addFeature(...)` dan `feedback.setProgress(...)`	Tulis fitur dan perbarui bilah kemajuan.
90	`return {self.HASIL: dest}`	Serahkan hasil ke QGIS.

Cara menjalankan. Ada dua jalan.

1. **Dari QGIS.** Simpan berkas ke folder skrip Processing profil Anda: `%APPDATA%\QGIS\QGIS4\profiles\default\processing\scripts`. Alat muncul di Processing Toolbox, kelompok **Kehutanan**. Letak dan kemunculan di jendela QGIS tidak saya uji dari sini. [CEK]
2. **Dari terminal**, dengan `qgis_process`. Untuk skrip berbentuk kelas, jalur berkas boleh langsung diberikan:

```
TERMINAL
cd D:/Latihan/seri/M4/skrip
"C:/Program Files/QGIS 4.0.2/bin/qgis_process-qgis.bat" run kerapatan_hotspot.py -- PETAK=../../../data-sintetis/paket-m4/Petak_KPH.gpkg TITIK=../../../data-sintetis/paket-data/Hotspot_Nasional.gpkg BATAS=2 HASIL=hasil.gpkg
```

Hasil. Layar `qgis_process` menampilkan isian, bilah kemajuan, dan hasilnya (berdasarkan uji pada QGIS 4.0.2):

```
HASIL
Inputs
BATAS:        2
HASIL:        hasil.gpkg
PETAK:        ../../../data-sintetis/paket-m4/Petak_KPH.gpkg
TITIK:        ../../../data-sintetis/paket-data/Hotspot_Nasional.gpkg

0...10...20...30...40...50...60...70...80...90...100 - done.

Results
HASIL:        hasil.gpkg
```

Untuk melihat isinya dari Python, dengan alat yang sudah terdaftar di folder skrip (cara uji bab ini memakai folder skrip sementara):

```
PYTHON
import processing
D = "D:/Latihan/data-sintetis"
r = processing.run("script:kerapatan_hotspot", {
    "PETAK": D + "/paket-m4/Petak_KPH.gpkg",
    "TITIK": D + "/paket-data/Hotspot_Nasional.gpkg",
    "BATAS": 2, "HASIL": "TEMPORARY_OUTPUT"})
hasil = r["HASIL"]
print([f.name() for f in hasil.fields()])
print([f["ID_Petak"] for f in hasil.getFeatures() if f["Rawan"] == 1])
```

```
HASIL
['fid', 'ID_Petak', 'NAMA_KPH', 'Jenis', 'Tahun_Tanam', 'Luas_Ha', 'Jml_Titik', 'Titik_per_Ha', 'Rawan']
['P-07']
```

Hasilnya 16 petak dengan 12 titik terhitung. Hanya P-07 yang rawan (3 titik per hektare). Dengan `BATAS=0.5`, sepuluh petak rawan, karena setiap petak yang punya satu titik sudah melewati 0,5.

Bagian yang boleh diubah.

- Baris 43, `defaultValue=2.0`. Batas rawan bawaan.
- **Baris 21, 24, 27, 30.** Nama alat, nama tampil, dan kelompok. Nama teknis (baris 21) jangan memakai spasi.
- **Baris 68-70, nama kolom keluaran.** Ganti `"Rawan"` dengan nama lain. Skrip lain yang membaca hasilnya harus ikut diganti.

- Baris 83, pembagi `10000.0`. Ubah satuan luas. Bila Anda mengubahnya, ubah juga pesan di baris 53 agar tetap cocok.
- Baris 87, `> batas`. Ganti menjadi `>= batas` bila Anda ingin nilai yang tepat sama dengan batas ikut rawan.

Jebakan.

- **Satuan peta harus meter, dan kini diperiksa.** Luas diambil dari `pg.area()`, dalam satuan CRS petak. Pembagi 10000 hanya benar bila CRS dalam meter. Bila petak di EPSG:4326 (derajat), alat berhenti di baris 52-53 dengan pesan `CRS layer petak harus bersatuan meter ...`. Tanpa pengaman itu, hasilnya diam-diam salah: kerapatan menjadi belasan miliar titik per hektare (misalnya `12309965392.232` untuk satu petak, dan `36929896525.894` untuk P-07) dan sepuluh petak rawan, tanpa pesan galat. Bila muncul pesan itu, proyeksikan ulang petak ke UTM (misalnya EPSG:32749) lalu jalankan lagi. Pengaman ini hanya memeriksa layer petak. Layer titik boleh memakai CRS apa saja, karena baris 56 menerjemahkannya.
- **Tepat di batas tidak rawan.** Uji dengan `BATAS=1`: hanya P-07 rawan, padahal sembilan petak lain tepat 1,0. Alasannya `kerapatan > batas`, bukan `>=`.
- **Batas negatif ditolak.** `BATAS=-1` berhenti dengan `Incorrect parameter value for BATAS`, karena `minValue=0.0`.
- **Layer tertukar diterima.** Dari Python, jenis layer di baris 38 dan 40 tidak diperiksa. Bila titik dan petak tertukar, alat jalan tanpa galat dan hasilnya tidak berarti.
- **Titik di tepi.** `contains` tidak menghitung titik yang tepat di garis tepi atau di sudut petak. Diuji pada poligon contoh: titik di dalam bernilai `True`, titik di tepi dan titik di sudut bernilai `False`. Data bawaan tidak punya titik di tepi, jadi pengaruhnya pada hasil alat tidak terlihat di bab ini.

Latihan variasi.

1. Jalankan dengan batas 1,0, lalu cek berapa petak yang rawan.
2. Ganti nama kolom `Rawan` menjadi `Waspada`.

Contoh jawaban: untuk nomor 1, ganti `"BATAS": 2` menjadi `"BATAS": 1`. Hasil uji: tetap hanya `['P-07']`. Untuk nomor 2, ubah baris 70 menjadi:

```
PYTHON
        kolom.append(QgsField("Waspada", QMetaType.Type.Int))
```

Hasil uji: daftar kolom berakhir `'Titik_per_Ha', 'Waspada'`, dan petak waspada tetap `['P-07']`. Hanya satu baris perlu diubah, karena baris 87 mengisi nilai berdasarkan urutan, bukan nama. Teks bantuan di baris 34 masih menyebut "Rawan"; ganti juga bila perlu.

> **Pola yang bisa dipakai ulang:** Kerangka alat gaya kelas: **identitas** (`createInstance`, `name`, `displayName`, `group`), **formulir** (`initAlgorithm` dengan `addParameter`), **pekerjaan** (`processAlgorithm`: ambil isian, siapkan sink, ulangi tiap fitur, kembalikan hasil). Untuk alat baru, ganti nama kelas, `name`, parameter, dan isi perulangan.

Skrip 27.5: Alat Processing gaya dekorator

Berkas `tambah_luas_ha.py`. Bagi pembaca buku M4, ini bagian "Cara ringkas: dekorator @alg" di Bab 2.

Tujuan. Alat yang menambah satu kolom berisi luas dalam hektare, dihitung di elipsoid WGS 84 (bentuk bumi yang dipakai GPS), pada layer poligon. Ini versi ringkas: 29 baris melawan 90 baris.

Bahan. `paket-m4/Petak_KPH.gpkg`.

Analogi: kalau gaya kelas adalah formulir resmi lengkap, **dekorator** adalah **kartu isian satu lembar**. Dekorator (Bab 11) adalah baris berawalan `@` yang dipasang di atas sebuah fungsi untuk "membungkusnya" dengan tugas tambahan. Di sini bungkusnya mengubah fungsi biasa menjadi alat Processing.

Skrip utuh:

```
PYTHON
# -*- coding: utf-8 -*-
"""Processing script gaya dekorator @alg. Penulis: Badar Mubarok Yogaswara"""
from qgis.processing import alg
from qgis.core import QgsFeature, QgsField, QgsFields, QgsFeatureSink, QgsDistanceArea, QgsProcessingException,
QgsWkbTypes, Qgis
from qgis.PyQt.QtCore import QMetaType


@alg(name="tambah_luas_ha", label="Tambah kolom luas (hektare)", group="kehutanan", group_label="Kehutanan")
@alg.input(type=alg.SOURCE, name="INPUT", label="Layer poligon", types=[2])
@alg.input(type=alg.SINK, name="OUTPUT", label="Hasil")
def tambah_luas_ha(instance, parameters, context, feedback, inputs):
    """Menambah kolom Luas_Ha_Hitung (hektare, dihitung di elipsoid WGS 84)."""
    sumber = instance.parameterAsSource(parameters, "INPUT", context)
    if sumber is None or QgsWkbTypes.geometryType(sumber.wkbType()) != Qgis.GeometryType.Polygon:
        raise QgsProcessingException("Layer masukan harus bertipe poligon.")
    kolom = QgsFields(sumber.fields())
    kolom.append(QgsField("Luas_Ha_Hitung", QMetaType.Type.Double))
    sink, dest = instance.parameterAsSink(parameters, "OUTPUT", context, kolom, sumber.wkbType(),
sumber.sourceCrs())
    if sink is None:
        raise QgsProcessingException("Keluaran tidak bisa dibuat.")
    da = QgsDistanceArea()
    da.setSourceCrs(sumber.sourceCrs(), context.transformContext())
    da.setEllipsoid("WGS84")
    for f in sumber.getFeatures():
        baru = QgsFeature(kolom)
        baru.setGeometry(f.geometry())
        baru.setAttributes(f.attributes() + [round(da.measureArea(f.geometry()) / 10000.0, 3)])
        sink.addFeature(baru, QgsFeatureSink.Flag.FastInsert)
    return {"OUTPUT": dest}
```

Penjelasan baris demi baris.

Baris	Tulisan	Artinya
3	`from qgis.processing import alg`	Panggil dekorator `alg`.

Baris	Tulisan	Artinya
4-5	`from qgis.core import (...)` dan `QMetaType`	Perkakas fitur, kolom, pengukur luas, pelempar galat Processing (`QgsProcessingException`), dan pemeriksa jenis bentuk (`QgsWkbTypes` dan `Qgis`).
8	`@alg(name="tambah_luas_ha", label=..., group=..., group_label=...)`	Identitas alat dalam satu baris: nama teknis, nama tampil, kelompok. Padanan baris 17-34 pada Skrip 27.4.
9	`@alg.input(type=alg.SOURCE, name="INPUT", label="Layer poligon", types=[2])`	Isian layer masukan. Angka 2 artinya poligon (diperiksa: `TypeVectorPolygon` bernilai 2, titik 0, garis 1).
10	`@alg.input(type=alg.SINK, name="OUTPUT", label="Hasil")`	Isian keluaran.
11	`def tambah_luas_ha(instance, parameters, context, feedback, inputs):`	Fungsi pekerjaannya. Isinya padanan `processAlgorithm`.
12	`"""..."""`	Docstring, juga dipakai sebagai teks bantuan.
13	`sumber = instance.parameterAsSource(...)`	Ambil layer masukan dari isian formulir.
14-15	`if sumber is None or QgsWkbTypes.geometryType(...) != Qgis.GeometryType.Polygon: raise QgsProcessingException(...)`	Pengaman jenis layer. Bila layer tidak terbaca, atau bukan poligon (titik atau garis), alat berhenti dengan pesan "Layer masukan harus bertipe poligon." Pengaman ini perlu karena `types=[2]` di baris 9 hanya mengatur pilihan di jendela QGIS, dan tidak menolak layer yang dikirim lewat Python.
16-18	`kolom = ..., kolom.append(...), sink, dest = ...`	Salin kolom, tambah `Luas_Ha_Hitung`, siapkan keluaran. Sama seperti Skrip 27.4.
19-20	`if sink is None: raise QgsProcessingException(...)`	Bila tempat keluaran gagal dibuat, hentikan dengan pesan jelas. Sama seperti baris 73-74 pada Skrip 27.4.
21-23	`da = QgsDistanceArea(), setSourceCrs, setEllipsoid("WGS84")`	Siapkan pengukur luas di permukaan bumi yang melengkung. Hasilnya sedikit berbeda dari luas datar `area()`.
24-28	`for f in sumber.getFeatures():` dst.	Untuk tiap fitur: ukur luas, bagi 10000, bulatkan tiga desimal, tambahkan ke atribut, tulis.
29	`return {"OUTPUT": dest}`	Kembalikan hasil.

Cara menjalankan. Simpan berkas di folder skrip Processing profil Anda (jalur sama seperti Skrip 27.4). Untuk berkas gaya dekorator, perintah `qgis_process run tambah_luas_ha.py` **tidak** menemukan alatnya. Uji bab ini menjawab `Algorithm script:tambah_luas_ha not found!`. Alat harus berada di folder skrip yang dikenal QGIS, lalu dipanggil dengan nama `script:tambah_luas_ha`.

```
PYTHON
import processing
D = "D:/Latihan/data-sintetis"
r = processing.run("script:tambah_luas_ha", {
    "INPUT": D + "/paket-m4/Petak_KPH.gpkg", "OUTPUT": "TEMPORARY_OUTPUT"})
hasil = r["OUTPUT"]
print([f.name() for f in hasil.fields()])
print([(f["ID_Petak"], f["Luas_Ha"], f["Luas_Ha_Hitung"]) for f in list(hasil.getFeatures())[:3]])
```

Hasil. Layar menampilkan (berdasarkan uji pada QGIS 4.0.2, dengan folder skrip sementara):

```
HASIL
['fid', 'ID_Petak', 'NAMA_KPH', 'Jenis', 'Tahun_Tanam', 'Luas_Ha', 'Luas_Ha_Hitung']
[('P-01', 1.0, 1.0), ('P-02', 1.0, 1.0), ('P-03', 1.0, 1.0)]
```

Total kolom `Luas_Ha_Hitung` untuk 16 petak adalah 16,0 ha. Luas elipsoid hampir sama dengan kolom `Luas_Ha`. Selisihnya sekitar 0,00007 ha per petak (luas elipsoid sedikit lebih kecil), sehingga hilang saat dibulatkan ke tiga desimal.

Bagian yang boleh diubah.

- Baris 8, `name`, `label`, `group`. Identitas alat.
- Baris 9, `types=[2]`. Jenis layer yang diterima di formulir. 2 poligon, 1 garis, 0 titik, -1 semua jenis. Baris 14 juga membatasi ke poligon, jadi bila alat ini diubah untuk jenis lain, ubah kedua baris.
- **Baris 17, nama kolom** `Luas_Ha_Hitung`.
- Baris 27, `10000.0` dan `3`. Satuan dan jumlah desimal.
- Baris 23, `"WGS84"`. Elipsoid pengukur. Pilih yang sesuai wilayah, bila lembaga Anda menetapkannya. [CEK]

Jebakan.

- **Layer titik ditolak.** Dari Python, `types=[2]` tidak ditegakkan, jadi baris 14-15 yang menjaganya. Layer `Hotspot_Nasional` (titik) kini berhenti dengan `QgsProcessingException: Layer masukan harus bertipe poligon.` (Diuji.) Tanpa pengaman itu, layer titik diterima dan kolom luasnya bernilai 0,0 tanpa galat, sehingga hasilnya tampak sah padahal tidak berarti.
- **Pemeriksaan keluaran.** Baris 19-20 menjaga bila `parameterAsSink` mengembalikan `None`. Pada uji bab ini, folder keluaran yang tidak ada tidak sampai ke baris itu, karena QGIS lebih dulu menghentikan alat dengan pesannya sendiri (`Could not create layer ...`). Cabang `None` hanya tercapai dengan objek tiruan, jadi belum teruji pada pemakaian nyata.

- **Fitur tanpa bentuk.** Fitur yang geometrinya kosong tidak menimbulkan galat: `measureArea` mengembalikan 0, jadi kolom `Luas_Ha_Hitung` bernilai 0,0 untuk fitur itu. (Diuji pada layer sementara berisi satu poligon dan satu fitur tanpa bentuk.)
- Tidak ada `feedback`. Tidak ada bilah kemajuan dan tombol batal yang bekerja.

Latihan variasi.

1. Ganti nama kolom menjadi `Luas_Ha_Elips`.
2. Bulatkan sampai dua desimal.

Contoh jawaban, hanya bagian yang berubah:

```python
    kolom.append(QgsField("Luas_Ha_Elips", QMetaType.Type.Double))
```

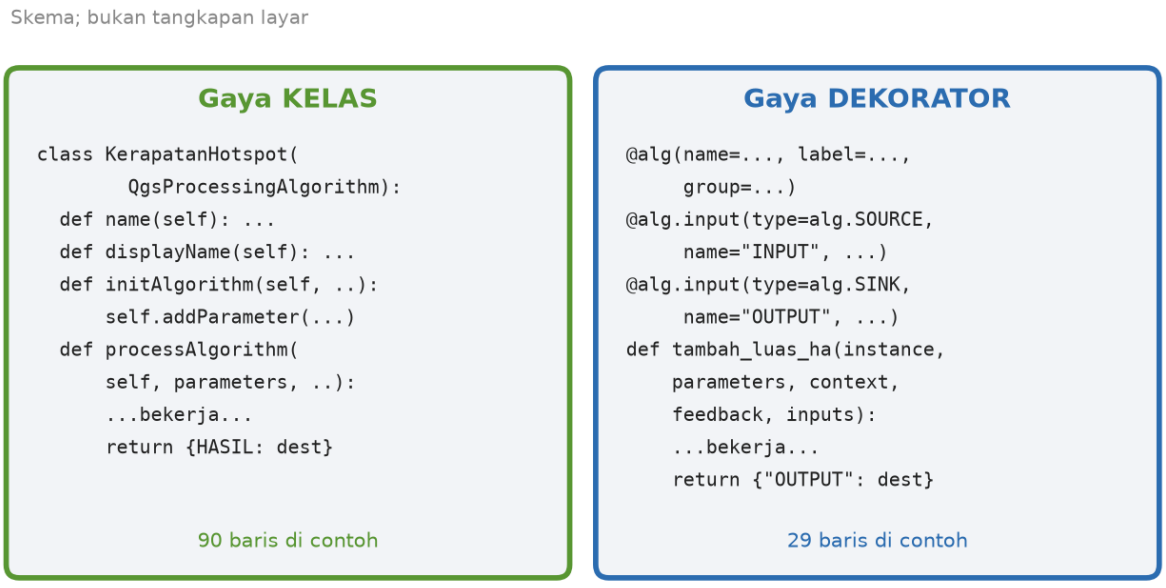
dan di baris 27, ganti `3` menjadi `2`. Hasil uji: kolom bernama `Luas_Ha_Elips`, nilai `1.0` untuk tiga petak pertama.

> **Pola yang bisa dipakai ulang:** Kerangka alat gaya dekorator: **tiga dekorator** (`@alg`, `@alg.input` untuk masukan, `@alg.input` untuk keluaran) di atas **satu fungsi** berparameter tetap `(instance, parameters, context, feedback, inputs)`. Untuk alat baru, ganti identitas, label isian, dan isi fungsi.

Dua gaya, satu pekerjaan: kapan memakai yang mana

Bab 11 sudah memperkenalkan potongan kedua gaya ini. Di sini Anda melihat skrip lengkap dan hasil jalannya. Skrip 27.4 dan 27.5 sama-sama alat Processing. Keduanya terdaftar di Toolbox, bisa dipakai di Modeler dan mode batch, dan bisa dipanggil lewat Python. Bedanya ada di cara menulis.

Ilustrasi 27.4: Dua gaya alat Processing



Hal	Gaya kelas (27.4)	Gaya dekorator (27.5)
Panjang contoh	90 baris	29 baris
Identitas alat	Enam metode terpisah	Satu dekorator `@alg(...)`
Isian formulir	`initAlgorithm` dengan `addParameter`	Satu `@alg.input(...)` per isian
Nama isian	Konstanta kelas (`self.PETAK`)	Teks langsung (`"INPUT"`)
Bilah kemajuan dan batal	Dipakai lewat `feedback`	Tersedia (`feedback`), tetapi contoh tidak memakainya
Pemeriksaan masukan	`QgsProcessingException` dengan pesan jelas: layer terbaca, satuan meter, keluaran jadi	`QgsProcessingException` dengan pesan jelas: jenis poligon, keluaran jadi
`qgis_process run berkas.py`	Berhasil	Tidak menemukan alat
Cocok untuk	Alat dengan banyak isian dan kontrol rinci	Alat kecil, satu tugas

Saran praktis: mulailah dengan gaya dekorator untuk alat kecil. Pindah ke gaya kelas bila Anda butuh banyak isian, kontrol rinci, atau pelaporan kemajuan. Pemeriksaan masukan bisa ditulis di kedua gaya. Isi pekerjaannya, yaitu menyiapkan sink dan menulis fitur, hampir sama di kedua gaya. Jadi belajar satu membantu yang lain.

Skrip 27.6: Plugin sendiri di QGIS

Berkas `plugin_ringkas_kph/__init__.py` dan `plugin_ringkas_kph/ringkas_kph.py`. Bagi pembaca buku M4, ini plugin Bab 1.

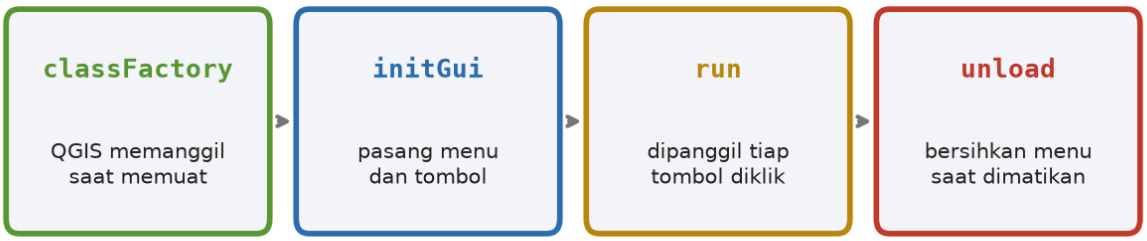
Tujuan. Menambah satu tombol dan satu menu di QGIS. Satu klik menampilkan jumlah objek dan luas total layer yang sedang dipilih.

Bahan. Folder `plugin_ringkas_kph` berisi empat berkas: dua skrip Python di bawah, `metadata.txt` (data diri plugin), dan `icon.png` (gambar tombol). Contoh layer: `paket-m4/Petak_KPH.gpkg`.

Analogi: plugin adalah **tombol baru di dinding**. QGIS adalah gedungnya. Anda memasang tombol itu lewat **pintu depan** (`classFactory`), dan QGIS memberi Anda **remote control** bernama `iface` untuk mengatur menu dan pesan. Plugin punya empat tahap hidup.

Ilustrasi 27.5: Empat tahap hidup plugin

Skema; bukan tangkapan layar



Empat tahap hidup sebuah plugin. run bisa berulang berkali-kali sebelum unload.

Empat istilah baru: `classFactory` (pintu depan), `iface` (remote control QGIS), `initGui` (pasang tombol), `unload` (cabut tombol).

Berkas pertama: `__init__.py`

```python
# -*- coding: utf-8 -*-
"""Pintu masuk plugin. QGIS memanggil classFactory saat plugin dimuat."""


def classFactory(iface):
    from .ringkas_kph import RingkasKPH
    return RingkasKPH(iface)
```

Baris	Tulisan	Artinya
1	`# -*- coding: utf-8 -*-`	Penanda huruf UTF-8.
2	`"""Pintu masuk plugin. ..."""`	Docstring: penjelasan berkas.
5	`def classFactory(iface):`	Fungsi dengan nama **wajib persis** ini. QGIS memanggilnya saat memuat plugin dan memberikan `iface`.
6	`from .ringkas_kph import RingkasKPH`	Panggil kelas dari berkas tetangga. Titik di depan `.ringkas_kph` berarti "dari folder yang sama". Pemanggilan ini sengaja ditaruh di dalam fungsi.
7	`return RingkasKPH(iface)`	Buat satu benda plugin dan serahkan ke QGIS.

Berkas kedua: `ringkas_kph.py`

```python
# -*- coding: utf-8 -*-
"""Plugin Ringkas KPH. Penulis: Badar Mubarok Yogaswara"""
import os

from qgis.core import Qgis, QgsCoordinateTransformContext, QgsDistanceArea, QgsWkbTypes
from qgis.PyQt.QtGui import QAction, QIcon


def ringkas_layer(layer):
    """Mengembalikan (jumlah objek, luas total dalam hektare). Luas dihitung di elipsoid WGS 84."""
    da = QgsDistanceArea()
    da.setSourceCrs(layer.crs(), QgsCoordinateTransformContext())
    da.setEllipsoid("WGS84")
    jumlah = 0
    luas_m2 = 0.0
    for f in layer.getFeatures():
        jumlah += 1
        if layer.geometryType() == QgsWkbTypes.GeometryType.PolygonGeometry:
            luas_m2 += da.measureArea(f.geometry())
    return jumlah, luas_m2 / 10000.0


class RingkasKPH:
    def __init__(self, iface):
        self.iface = iface
        self.action = None

    def initGui(self):
        ikon = os.path.join(os.path.dirname(__file__), "icon.png")
        self.action = QAction(QIcon(ikon), "Ringkas layer aktif", self.iface.mainWindow())
        self.action.triggered.connect(self.run)
        self.iface.addPluginToMenu("&Ringkas KPH", self.action)
        self.iface.addToolBarIcon(self.action)

    def unload(self):
        self.iface.removePluginMenu("&Ringkas KPH", self.action)
        self.iface.removeToolBarIcon(self.action)
        self.action = None

    def run(self):
        layer = self.iface.activeLayer()
        if layer is None or not hasattr(layer, "getFeatures"):
            self.iface.messageBar().pushMessage("Ringkas KPH", "Pilih dulu satu layer vektor di panel Layers.",
                                                level=Qgis.MessageLevel.Warning, duration=6)
            return
        jumlah, luas_ha = ringkas_layer(layer)
        self.iface.messageBar().pushMessage("Ringkas KPH", "%s: %d objek, luas %.2f ha." % (layer.name(),
jumlah, luas_ha),
                                            level=Qgis.MessageLevel.Info, duration=8)
```

Penjelasan baris demi baris. Bagian pertama: perkakas dan penghitung (baris 1-20).

Baris	Tulisan	Artinya
5	`from qgis.core import Qgis, QgsCoordinateTransformContext, QgsDistanceArea, QgsWkbTypes`	Perkakas tingkat pesan, pengukur luas, dan jenis bentuk.

Baris	Tulisan	Artinya
6	`from qgis.PyQt.QtGui import QAction, QIcon`	Perkakas tombol (`QAction`) dan gambar tombol (`QIcon`). Letak `QAction` ini sesuai QGIS 4 (berbasis Qt6). Versi lama menaruhnya di tempat lain.
9-10	`def ringkas_layer(layer):`	Fungsi pembantu, terpisah dari urusan tombol agar mudah diuji. Mengembalikan dua nilai.
11-13	`da = QgsDistanceArea()` dst.	Pengukur luas di elipsoid WGS 84, seperti Skrip 27.5.
14-15	`jumlah = 0` dan `luas_m2 = 0.0`	Dua penghitung awal.
16-19	`for f in layer.getFeatures():` dst.	Tiap fitur menambah `jumlah`. Luas hanya ditambah bila layer berjenis poligon.
20	`return jumlah, luas_m2 / 10000.0`	Kembalikan jumlah dan luas dalam hektare.

Bagian kedua: kelas plugin (baris 23-48).

Baris	Tulisan	Artinya
23-26	`class RingkasKPH:` dan `def __init__(self, iface)`	Cetakan plugin. `__init__` dijalankan sekali saat benda dibuat: menyimpan `iface` dan menyiapkan tempat untuk tombol (`self.action`).
28-30	`def initGui(self):` dan `self.action = QAction(...)`	Dipanggil QGIS saat plugin dinyalakan. Buat tombol dengan ikon dan teks "Ringkas layer aktif".
31	`self.action.triggered.connect(self.run)`	Hubungkan: bila tombol diklik, jalankan metode `run`. `triggered` adalah sinyal klik.
32-33	`addPluginToMenu` dan `addToolBarIcon`	Pasang tombol di menu **Plugins ► Ringkas KPH** dan di bilah alat.
35-38	`def unload(self):`	Saat plugin dimatikan, cabut tombol dari menu dan bilah alat, lalu kosongkan `self.action`.
40-41	`def run(self):` dan `layer = self.iface.activeLayer()`	Dipanggil tiap tombol diklik. Ambil layer yang sedang dipilih.
42-45	`if layer is None or not hasattr(layer, "getFeatures"): ... return`	Bila tidak ada layer, atau layernya bukan vektor, tampilkan peringatan selama 6 detik lalu berhenti.
46-48	`jumlah, luas_ha = ringkas_layer(layer)` lalu `pushMessage(...)`	Hitung, lalu tampilkan hasil di bilah pesan QGIS selama 8 detik. `%s`, `%d`, dan `%.2f` diisi nama, jumlah, dan luas dua desimal.

Isi `metadata.txt` yang diperiksa QGIS (nilai contoh dari paket):

Bidang	Isi contoh	Arti
`name`	Ringkas KPH	Nama plugin di daftar.
`qgisMinimumVersion`	3.34	Versi QGIS terendah yang didukung.
`description` dan `about`	teks singkat dan teks panjang	Penjelasan pendek dan panjang.
`version`	0.1.0	Nomor versi, bentuk mayor.minor.tambal.
`author` dan `email`	nama dan alamat surel penulis	Identitas pembuat.
`repository`	alamat contoh di example.org	Tempat kode; alamat contoh, bukan yang sungguhan.

Cara memasang. Salin folder `plugin_ringkas_kph` ke folder plugin profil QGIS Anda. Di Windows: `%APPDATA%\QGIS\QGIS4\profiles\default\python\plugins`. Lalu buka **Plugins ► Manage and Install Plugins**, tab **Installed**, dan centang **Ringkas KPH**. Pilih layer di panel Layers, lalu klik tombol atau menu **Plugins ► Ringkas KPH ► Ringkas layer aktif**. Langkah klik ini tidak saya uji karena butuh jendela QGIS. [CEK]

Yang saya uji adalah isi kodenya, tanpa jendela, dengan `iface` tiruan yang menyediakan menu dan bilah pesan sederhana (cara menjalankan tanpa jendela dibahas di Bab 3).

Hasil. Pesan yang muncul pada uji (berdasarkan QGIS 4.0.2, `iface` tiruan):

```
HASIL
kelas: RingkasKPH
menu: ['&Ringkas KPH', 'bar']
PESAN: Ringkas KPH | Pilih dulu satu layer vektor di panel Layers. | level 1 | detik 6
PESAN: Ringkas KPH | Petak_KPH: 16 objek, luas 16.00 ha. | level 0 | detik 8
menu setelah unload: []
```

Urutannya mengikuti empat tahap hidup: `classFactory` membuat benda, `initGui` memasang dua tempelan (menu dan bilah), `run` dipanggil dua kali (tanpa layer, lalu dengan layer), dan `unload` mengosongkan semuanya. Luas aslinya 15,9988 ha. Pesan menampilkan 16,00 karena `%.2f` membulatkan dua desimal.

Bagian yang boleh diubah.

- **Baris 30, teks "Ringkas layer aktif".** Label tombol dan menu.
- Baris 32 dan 36, `"&Ringkas KPH"`. Nama menu. Tanda `&` menandai huruf pintasan. Harus sama di baris 32 dan 36.
- Baris 44 dan 48, `duration=6` dan `duration=8`. Lama pesan tampil, dalam detik.
- Baris 13, `"WGS84"`. Elipsoid pengukur.
- `__init__.py` baris 6-7. Bila Anda mengganti nama kelas, ganti di dua tempat ini.

Jebakan.

- Nama `classFactory` wajib persis. Huruf besar dan kecil dihitung. Alat pengemas di Skrip 27.7 memeriksa kata `def classFactory`.

- **Titik di layer titik.** Layer `Hotspot_Nasional` memberi `(12, 0.0)`: dua belas objek, luas 0. Itu benar, bukan galat, karena titik tak punya luas.
- Butuh `iface` sungguhan. Kode tidak bisa dijalankan langsung seperti skrip biasa. `iface` hanya ada di dalam QGIS.
- `QgsDistanceArea` tanpa konteks. Objek ini diberi `QgsCoordinateTransformContext()` yang kosong. Untuk layer UTM sederhana tidak masalah, tetapi untuk transformasi datum yang khusus, hasilnya bisa berbeda. [kemungkinan]

Latihan variasi.

1. Ubah label tombol menjadi "Ringkas layer terpilih".
2. Perpanjang pesan sukses menjadi 12 detik.

Contoh jawaban, hanya baris yang berubah:

```
PYTHON
        self.action = QAction(QIcon(ikon), "Ringkas layer terpilih", self.iface.mainWindow())
```

dan pada `pushMessage` terakhir, `duration=12`. Hasil uji: teks aksi menjadi `Ringkas layer terpilih`.

> **Pola yang bisa dipakai ulang:** Kerangka plugin: `__init__.py` dengan `classFactory`, lalu kelas dengan `__init__`, `initGui`, `unload`, dan `run`. Pekerjaan hitung ditaruh di fungsi terpisah. Untuk plugin baru, ganti nama kelas, label menu, dan isi `run`.

Skrip 27.7: Memeriksa dan mengemas plugin

Berkas `kemas_plugin.py`. Bagi pembaca buku M4, ini bagian pengemasan di Bab 6.

Tujuan. Memeriksa bahwa plugin lengkap (berkas wajib dan data diri), lalu mengemasnya menjadi satu berkas zip.

Bahan. Folder `plugin_ringkas_kph` dari Skrip 27.6. Skrip ini hanya memakai Python biasa, tanpa QGIS, jadi Python sistem pun cukup. Jalankan dengan `python kemas_plugin.py`. Anda juga bisa memberi jalur folder plugin lain sebagai argumen.

Analogi: ini **daftar periksa sebelum mengirim paket**. Alamat lengkap? Isi paket benar? Baru dibungkus.

Skrip utuh:

```
PYTHON
# -*- coding: utf-8 -*-
"""Memeriksa metadata dan mengemas plugin menjadi zip (hanya berkas lokal). Penulis: Badar Mubarok Yogaswara
Jalankan: python kemas_plugin.py [folder_plugin]   (Python biasa sudah cukup; tanpa QGIS)
"""
import configparser, os, re, sys, zipfile

WAJIB = ["name", "qgisMinimumVersion", "description", "about", "version", "author", "email", "repository"]
POLA_VERSI = re.compile(r"^\d+\.\d+\.\d+$")


def periksa(folder):
```

```
    """Kembalikan daftar masalah (kosong = lolos)."""
    masalah = []
    if not os.path.isfile(os.path.join(folder, "__init__.py")):
        masalah.append("tidak ada __init__.py")
    else:
        with open(os.path.join(folder, "__init__.py"), encoding="utf-8") as fh:
            ada_factory = "def classFactory" in fh.read()
        if not ada_factory:
            masalah.append("__init__.py tidak punya classFactory")
    meta = os.path.join(folder, "metadata.txt")
    if not os.path.isfile(meta):
        return masalah + ["tidak ada metadata.txt"]
    cp = configparser.ConfigParser()
    cp.read(meta, encoding="utf-8")
    g = cp["general"] if "general" in cp else {}
    for k in WAJIB:
        if not g.get(k):
            masalah.append("bidang metadata kosong: " + k)
    if g.get("version") and not POLA_VERSI.match(g["version"]):
        masalah.append("versi bukan pola mayor.minor.tambal: " + g["version"])
    return masalah


def kemas(folder, keluar):
    nama = os.path.basename(os.path.normpath(folder))
    induk = os.path.dirname(os.path.abspath(folder))
    ver = configparser.ConfigParser()
    ver.read(os.path.join(folder, "metadata.txt"), encoding="utf-8")
    berkas = os.path.join(keluar, "%s-%s.zip" % (nama, ver["general"]["version"]))
    os.makedirs(keluar, exist_ok=True)
    with zipfile.ZipFile(berkas, "w", zipfile.ZIP_DEFLATED) as zf:
        for r, _, fs in os.walk(folder):
            if "__pycache__" in r:
                continue
            for f in fs:
                p = os.path.join(r, f)
                zf.write(p, os.path.relpath(p, induk))
    return berkas


if __name__ == "__main__":
    aq = os.path.dirname(os.path.abspath(__file__))
    folder = sys.argv[1] if len(sys.argv) > 1 else os.path.join(aq, "plugin_ringkas_kph")
    m = periksa(folder)
    if m:
        print("GAGAL:", m)
        sys.exit(1)
    z = kemas(folder, os.path.join(aq, "keluaran"))
    print("LOLOS. Zip:", os.path.basename(z))
    with zipfile.ZipFile(z) as zf:
        print("isi:", zf.namelist())
    # uji kegagalan: folder tanpa metadata
    import tempfile
    with tempfile.TemporaryDirectory() as t:
        with open(os.path.join(t, "__init__.py"), "w") as fh:
            fh.write("x = 1\n")
        print("uji folder rusak ->", periksa(t))
```

Penjelasan baris demi baris. Bagian pertama: aturan (baris 5-8).

Baris	Tulisan	Artinya
5	`import configparser, os, re, sys, zipfile`	Lima perkakas: baca berkas `metadata.txt`, jalur folder, pola teks, argumen terminal, dan zip.
7	`WAJIB = [...]`	Daftar delapan bidang metadata yang harus terisi.
8	`POLA_VERSI = re.compile(r"^\d+\.\d+\.\d+$")`	Pola versi: tiga kelompok angka dipisah titik, seperti `0.1.0`. `re` adalah perkakas pencocok pola.

Bagian kedua: pemeriksa (baris 11-32).

Baris	Tulisan	Artinya
11-13	`def periksa(folder):` dan `masalah = []`	Fungsi pemeriksa. `masalah` adalah daftar kosong tempat keluhan dikumpulkan.
14-20	`if not os.path.isfile(...)` dan `with open(...) as fh:`	Pastikan `__init__.py` ada, dan memuat kata `def classFactory`. Berkas dibuka dengan `with` (Bab 9), jadi tertutup otomatis begitu isinya selesai dibaca. Hasil pencarian disimpan di `ada_factory`. Bila tidak ada, tambahkan keluhan.
21-23	`meta = ...` dan `return masalah + ["tidak ada metadata.txt"]`	Bila `metadata.txt` tidak ada, langsung kembali dengan keluhan itu.
24-26	`cp = configparser.ConfigParser()` dst.	Baca `metadata.txt`. Ambil bagian `[general]`, atau kamus kosong bila tidak ada.
27-29	`for k in WAJIB:` dan `if not g.get(k):`	Tiap bidang wajib diperiksa. Kosong atau tidak ada berarti keluhan.
30-32	`if g.get("version") and not POLA_VERSI.match(...)`	Periksa bentuk versi, lalu kembalikan semua keluhan. Daftar kosong berarti lolos.

Bagian ketiga: pengemas dan penjalan (baris 35-68).

Baris	Tulisan	Artinya
35-40	`def kemas(folder, keluar):`	Hitung nama folder plugin dan nama zip: `nama-versi.zip`, versinya dibaca dari `metadata.txt`.
41-42	`os.makedirs(...)` dan `with zipfile.ZipFile(berkas, "w", ...)`	Buat folder keluaran dan buka zip untuk ditulisi.
43-48	`for r, _, fs in os.walk(folder):`	Telusuri seluruh isi folder. Folder `__pycache__` dilewati (`continue`). Tiap berkas dimasukkan dengan jalur yang dimulai dari nama folder plugin.
49	`return berkas`	Kembalikan jalur zip.
52-54	`if __name__ == "__main__":` dan `folder = sys.argv[1] if ... else ...`	Pilih folder: argumen terminal bila ada, kalau tidak folder bawaan di samping skrip.

Baris	Tulisan	Artinya
55-58	`m = periksa(folder)` dan `sys.exit(1)`	Bila ada keluhan, cetak `GAGAL` lalu berhenti dengan kode 1 (tanda gagal).
59-62	`z = kemas(...)` lalu `with zipfile.ZipFile(z) as zf:` dan `print(...)`	Kemas, lalu cetak nama zip dan isinya. Zip dibuka dengan `with`, jadi tertutup otomatis setelah daftar isinya dicetak.
63-68	`import tempfile` dan `with tempfile.TemporaryDirectory() as t:`	Uji kegagalan: buat folder sementara tanpa metadata dan lihat keluhannya. `TemporaryDirectory` (dari perkakas `tempfile`) membuat folder sementara, lalu menghapusnya sendiri begitu blok `with` selesai.

Hasil. Layar menampilkan (berdasarkan uji dengan Python 3.12 biasa):

```
HASIL
LOLOS. Zip: plugin_ringkas_kph-0.1.0.zip
isi: ['plugin_ringkas_kph/icon.png', 'plugin_ringkas_kph/metadata.txt', 'plugin_ringkas_kph/ringkas_kph.py',
'plugin_ringkas_kph/__init__.py']
uji folder rusak -> ['__init__.py tidak punya classFactory', 'tidak ada metadata.txt']
```

Zip terbentuk di `keluaran`. Di QGIS, zip ini dipasang lewat **Plugins ► Manage and Install Plugins ► Install from ZIP**. [CEK: tidak diuji]

Bagian yang boleh diubah.

- Baris 7, daftar `WAJIB`. Tambah atau kurangi bidang yang wajib.
- **Baris 8, pola versi.** Ganti bila Anda memakai aturan versi lain.
- **Baris 59, folder keluaran** `os.path.join(aq, "keluaran")`.
- **Baris 54, folder bawaan.** Atau beri argumen saat menjalankan.

Jebakan.

- **Metadata rusak.** Plugin dengan bidang `email` dihapus dan versi `0.1` ditulis: `GAGAL: ['bidang metadata kosong: email', 'versi bukan pola mayor.minor.tambal: 0.1']`, kode keluar 1. (Diuji pada salinan.)
- **Folder tidak ada.** Dengan argumen yang salah: `GAGAL: ['tidak ada __init__.py', 'tidak ada metadata.txt']`.
- `kemas` tidak memeriksa lagi. Fungsi ini langsung membaca `metadata.txt`. Selalu panggil `periksa` lebih dulu, seperti di bagian utama.
- **Folder sementara dibersihkan otomatis.** Baris 65 memakai `TemporaryDirectory` dengan `with`, jadi folder ujinya hilang setelah dipakai. Bila memakai `tempfile.mkdtemp()`, folder sementara tidak pernah dihapus dan menumpuk di folder sementara sistem tiap skrip dijalankan. Pola `with` yang sama menutup berkas di baris 17 dan 61.

Latihan variasi.

1. Tambahkan `"tracker"` sebagai bidang wajib.
2. Uji pada salinan plugin tanpa baris `tracker=`.

Contoh jawaban, baris 7 menjadi:

```python
WAJIB = ["name", "qgisMinimumVersion", "description", "about", "version", "author", "email", "repository",
"tracker"]
```

Hasil uji: plugin asli tetap `LOLOS`. Salinan tanpa baris `tracker=` menghasilkan `GAGAL: ['bidang metadata kosong: tracker']`.

> **Pola yang bisa dipakai ulang:** Kerangka pemeriksa lalu pengemas: **kumpulkan keluhan ke dalam daftar, hentikan bila daftar tidak kosong, baru kemas**. Pola "daftar keluhan" cocok untuk memeriksa data juga. Ganti aturan di `WAJIB` dan fungsi `periksa`.

Cek paham

1. Apa yang membuat skrip M4 hanya jalan di struktur folder tertentu?
2. Apa beda singkat gaya kelas dan gaya dekorator pada alat Processing?
3. Fungsi mana yang QGIS panggil pertama kali saat memuat plugin, dan mengapa namanya tidak boleh diubah?

Jawaban.

1. Skrip menghitung lokasi data dari lokasinya sendiri dengan naik tiga folder lalu masuk `data-sintetis` (baris 11 pada Skrip 27.1).
2. Gaya kelas menulis identitas, formulir, dan pekerjaan sebagai metode terpisah di dalam kelas. Gaya dekorator menaruh identitas dan formulir di baris `@` di atas satu fungsi pekerjaan.
3. `classFactory`. QGIS mencari fungsi dengan nama itu, jadi nama lain tidak akan ditemukan.

Kesalahan umum

- **Memindah skrip tanpa memindah data.** Galatnya tentang berkas tidak ditemukan, dengan jalur lengkap yang dicari. Perbaikan: bangun struktur folder seperti bagian "Menyiapkan dulu".
- **Mengira halaman peta web berjalan offline.** Leaflet diambil dari internet. Perbaikan: sambungkan internet saat membuka halaman, atau simpan salinan Leaflet [CEK].
- Memakai `qgis_process run` untuk alat gaya dekorator. Perbaikan: taruh berkas di folder skrip Processing, lalu panggil `script:tambah_luas_ha`.
- **Menjalankan alat kerapatan pada petak ber-CRS derajat.** Pembagi 10000 hanya benar untuk satuan meter, jadi alat menolaknya dengan pesan. Perbaikan: ubah petak ke CRS UTM sebelum menjalankan alat.
- **Memasukkan layer titik ke alat luas.** Alat `tambah_luas_ha` hanya menerima poligon dan berhenti dengan pesan bila diberi titik. Perbaikan: pilih layer petak.
- **CSV berisi kode petak yang tidak ada di GeoPackage.** Dasbor berhenti dengan pesan yang menyebut kodenya. Perbaikan: samakan kode di kedua berkas.

Ringkasan dan latihan

- Satu data bisa dikemas menjadi dasbor, peta web, model 3D, alat Processing, atau plugin. Pola dasar skripnya tetap: baca, olah, tulis.
- Skrip M4 menghitung jalur data dari lokasinya sendiri, jadi struktur folder penulis harus ditiru.
- Alat Processing punya dua gaya: kelas (lengkap dan rinci) dan dekorator (ringkas).
- Plugin punya empat tahap: `classFactory`, `initGui`, `run`, `unload`.
- Periksa dulu, kemas kemudian: daftar keluhan mencegah paket cacat.
- Skrip yang baik memeriksa masukannya lebih dulu dan berhenti dengan pesan yang jelas: kode petak tak dikenal, satuan peta bukan meter, layer bukan poligon.

Latihan. Jalankan Skrip 27.1 dengan data Anda sendiri, misalnya data bulan Oktober. Label grafik dan nama bulan di kartu "Hari puncak" akan ikut berubah. Lalu ubah judul kartu "Total titik panas (30 hari)" di baris 68 dan judul halaman di baris 76 agar sesuai data Anda. Petunjuk: hitung jumlah hari dengan `len(d["hari"])`.

Tabel perbandingan: lima produk dari satu data

Produk	Skrip	Perlu QGIS penuh?	Keluaran dan catatan
Dasbor	27.1	Tidak, cukup GDAL	`dasbor.html`, mandiri, tanpa internet
Peta web	27.2	Ya	HTML dan GeoJSON, butuh internet untuk Leaflet
Model 3D	27.3	Ya	`dsm.obj`, ukuran bergantung `langkah`
Alat Processing	27.4 dan 27.5	Ya	Muncul di Toolbox, dua gaya penulisan
Plugin	27.6 dan 27.7	Dipakai di dalam QGIS	Menu dan tombol, dikemas sebagai zip

BAB 28: Pola dan Improvisasi: Kerangka Siap Salin dan Cara Mengubahnya

> **Bab penutup Bagian B:** Bab ini tidak berasal dari satu seri. Ia merangkum pola yang berulang di Bab 14 sampai 27, yang berasal dari Seri B2, B3, B4, I1, I2, I3, I4, M1, M2, M3, dan M4. Semua kode di bab ini ditulis khusus untuk merangkum pola itu, dengan data latihan `paket-b2` dan `paket-i4`. Tidak ada skrip lama yang disalin utuh.

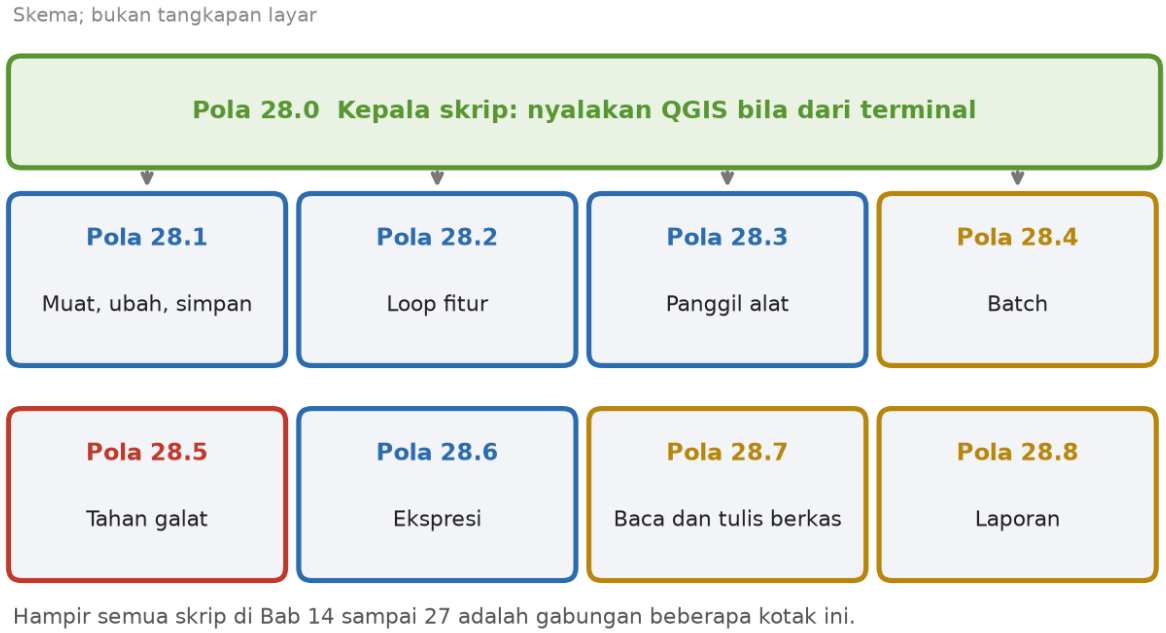
Studi kasus: "Datanya berubah. Skripnya bagaimana?"

Analis Muda sudah menulis puluhan skrip. Lalu datang data baru: folder berbeda, nama kolom berbeda, jumlah berkas bertambah. Ia tidak ingin menulis dari nol. Kepala Seksi berpesan sederhana: "Jangan hafalkan skripnya. Kenali polanya." Bab ini memberi Anda pola-pola itu, siap disalin, lengkap dengan petunjuk bagian mana yang diganti.

Konsep: improvisasi dalam tiga kalimat

Memasak dari resep itu mudah. **Berimprovisasi** artinya Anda tahu fungsi tiap bahan, jadi bisa menggantinya. Skrip GIS sama: hampir semuanya tersusun dari beberapa **pola dasar** yang disambung-sambung. Bila Anda mengenal polanya, data baru hanya berarti mengganti beberapa nama dan angka.

Ilustrasi 28.1: Delapan pola dasar PyQGIS



Setiap pola di bab ini dibangun dari bagian yang sudah Anda kenal. Lima istilah baru:

- **Pola**: kerangka skrip yang berulang di banyak tempat.
- **Kerangka**: pola yang sudah dibersihkan dari isi, siap Anda isi dengan data sendiri.

- **Jebakan**: kesalahan yang sering terjadi dan tidak selalu menimbulkan pesan galat.
- **Konsol Python QGIS**: tempat mengetik dan menjalankan Python di dalam QGIS (Bab 3).
- **python-qgis**: cara menjalankan Python berisi QGIS dari terminal, tanpa membuka jendela QGIS (Bab 3).

Menyiapkan dulu: data dan cara menjalankan

Pola di bab ini memakai dua paket data: `paket-b2` (petak, titik plot, tabel inventarisasi) dan `paket-i4` (batas KPH, sungai, titik panas, kiriman lapangan, ubin foto drone). Letakkan keduanya di satu folder, misalnya `D:/Latihan`.

Di tiap skrip, dua baris ini menentukan lokasi:

- `DATA`: folder yang berisi `paket-b2` dan `paket-i4`.
- `HASIL`: folder untuk berkas keluaran. Dibuat otomatis bila belum ada.

Kedua lokasi dicari lebih dulu di variabel lingkungan `PY_DATA` dan `PY_HASIL`. Bila variabel itu tidak ada, jalur contoh `D:/Latihan` dipakai (Bab 8). **Ganti jalur contoh itu** sesuai komputer Anda. Semua kode di bab ini diuji dengan `python-qgis.bat` di QGIS 4.0.2. Di Konsol Python QGIS, kode yang sama belum saya jalankan dari sini. [CEK]

Pola 28.0: Kepala skrip untuk terminal

Tujuan. Menyalakan QGIS bila skrip dijalankan dari terminal, supaya `processing.run` dan `QgsVectorLayer` bisa dipakai.

Bahan. Tidak ada data. Di Konsol Python QGIS, **tidak perlu** kepala ini, karena QGIS sudah menyala.

Skrip utuh:

```python
# Kepala skrip: hanya untuk terminal (python-qgis). Di Konsol Python QGIS tidak perlu.
import os
from qgis.core import QgsApplication
if QgsApplication.instance() is None:
    os.environ.setdefault("QT_QPA_PLATFORM", "offscreen")
    app = QgsApplication([], False)
    app.initQgis()
    import sys
    sys.path.append(os.path.join(QgsApplication.prefixPath(), "python", "plugins"))
    from processing.core.Processing import Processing
    Processing.initialize()
import processing
print("QGIS siap.")
```

Baris	Tulisan	Artinya
4	`if QgsApplication.instance() is None:`	Tanya: apakah QGIS sudah menyala? Di Konsol jawabannya ya, jadi blok di bawahnya dilewati. Dari terminal jawabannya tidak.
5	`os.environ.setdefault("QT_QPA_PLATFORM", "offscreen")`	Nyalakan tanpa jendela.
6-7	`app = QgsApplication([], False)` dan `app.initQgis()`	Buat dan nyalakan mesin QGIS.
9	`sys.path.append(os.path.join(QgsApplication.prefixPath(), "python", "plugins"))`	Tambahkan folder tempat modul `processing` berada. Lokasi dihitung dari tempat QGIS terpasang, jadi tidak perlu menulis jalur komputer.
10-11	`Processing.initialize()`	Daftarkan semua alat Processing.
12	`import processing`	Panggil perkakas `processing`.

Cara memakainya: tempel isi kepala ini **di atas** pola mana pun yang dijalankan dari terminal. Cara praktis lain: simpan kepala sebagai `kepala.py` dan pola sebagai berkas lain, lalu gabungkan.

```
TERMINAL
cat kepala.py pola_28_3.py > jalankan.py
"C:/Program Files/QGIS 4.0.2/bin/python-qgis.bat" jalankan.py
```

(Di Windows PowerShell, gunakan `Get-Content kepala.py, pola_28_3.py | Set-Content jalankan.py`.) Semua hasil di bawah diawali satu baris `QGIS siap.` dari terminal. Baris itu saya hilangkan dari contoh.

Jebakan. Jalur `C:/Program Files/QGIS 4.0.2` di perintah terminal adalah lokasi QGIS di komputer uji. Ganti sesuai pemasangan Anda.

Pola 28.1: Muat, ubah, simpan

Tujuan. Membuka satu layer, mengubahnya (di sini: menyaring), lalu menyimpan hasilnya ke berkas baru. Hampir semua skrip dasar di Bab 14 sampai 17 memakai pola ini.

Bahan. `paket-b2/Petak_KPH.gpkg`, layer `Petak` (16 petak).

Analogi: ini **fotokopi dengan stabilo**. Ambil dokumen (muat), tandai bagian yang Anda perlukan (ubah), simpan salinannya (simpan). Dokumen asli tidak disentuh.

Skrip utuh:

```
PYTHON
import os
from qgis.core import QgsVectorLayer, QgsVectorFileWriter, QgsProject

DATA = os.environ.get("PY_DATA", r"D:/Latihan")
HASIL = os.environ.get("PY_HASIL", r"D:/Latihan/hasil")
os.makedirs(HASIL, exist_ok=True)

# 1. Muat
```

```
petak = QgsVectorLayer(DATA + "/paket-b2/Petak_KPH.gpkg|layername=Petak", "Petak", "ogr")
assert petak.isValid(), "Layer tidak terbuka. Cek DATA dan nama berkas."
print("Muat:", petak.featureCount(), "petak")

# 2. Ubah: saring jenis Jati
petak.setSubsetString("\"JENIS\" = 'Jati'")
print("Setelah disaring:", petak.featureCount(), "petak")

# 3. Simpan
opsi = QgsVectorFileWriter.SaveVectorOptions()
opsi.driverName = "GPKG"
opsi.layerName = "Petak_Jati"
hasil = QgsVectorFileWriter.writeAsVectorFormatV3(
    petak, HASIL + "/petak_jati.gpkg", QgsProject.instance().transformContext(), opsi)
print("Simpan berhasil:", hasil[0] == QgsVectorFileWriter.WriterError.NoError)
petak.setSubsetString("")
```

Baris	Tulisan	Artinya
4-6	`DATA`, `HASIL`, `os.makedirs(...)`	Lokasi data dan keluaran. `exist_ok=True` berarti tidak apa-apa bila folder sudah ada.
9	`QgsVectorLayer(DATA + "/paket-b2/Petak_KPH.gpkg ..." , "Petak", "ogr")`	Buka layer. Tiga isian: sumber, nama layer, dan pembaca `"ogr"` (Bab 15). Sumber berupa jalur berkas, lalu satu tanda garis tegak, lalu `layername=Petak`, yaitu nama layer di dalam berkas GeoPackage.
10	`assert petak.isValid(), "..."`	Hentikan skrip bila layer gagal dibuka, dengan pesan yang jelas (Bab 10).
14	`petak.setSubsetString("\"JENIS\" = 'Jati'")`	Saring: hanya petak berjenis Jati yang terlihat. Teks saringan memakai kutip ganda untuk nama kolom dan kutip tunggal untuk nilai.
18-20	`opsi = QgsVectorFileWriter.SaveVectorOptions()` dst.	"Formulir" penyimpanan: format `GPKG` dan nama layer `Petak_Jati`.
21-22	`QgsVectorFileWriter.writeAsVectorFormatV3(petak, ..., opsi)`	Tulis layer ke berkas. Hasilnya daftar kecil; isian pertamanya adalah kode keberhasilan.
23	`hasil[0] == QgsVectorFileWriter.WriterError.NoError`	Bandingkan kode itu dengan "tidak ada galat".
24	`petak.setSubsetString("")`	Kembalikan layer seperti semula. Saringan kosong berarti semua petak terlihat.

Hasil. Layar menampilkan (berdasarkan uji pada QGIS 4.0.2):

```
HASIL
Muat: 16 petak
Setelah disaring: 5 petak
Simpan berhasil: True
```

Berkas `petak_jati.gpkg` terbentuk di folder `HASIL`.

Bagian yang boleh diubah.

- **Baris 4-5, lokasi data dan hasil.**
- **Baris 9, berkas dan layer.** Ganti untuk layer lain.
- **Baris 14, saringan.** Kolom dan nilai yang disaring.
- Baris 19, `driverName`. Format keluaran. `"GPKG"` untuk GeoPackage. Nama format lain mengikuti daftar format GDAL. [CEK]
- **Baris 20 dan 22, nama layer dan berkas keluaran.**

Jebakan.

- **Saringan menempel pada layer.** `setSubsetString` mengubah layer sampai dikosongkan lagi. Dalam uji: dengan saringan 5 petak, tanpa saringan 16. Bila skrip berhenti sebelum baris 24, saringan tetap menempel.
- **Jalur salah tidak memunculkan galat.** Baca bagian "Jebakan lintas bab" di bawah.

Latihan variasi.

1. Saring petak yang ditanam sebelum tahun 2010.
2. Saring petak berjenis Mahoni.

Contoh jawaban, hanya bagian yang berubah:

```python
petak.setSubsetString("\"TAHUN_TANAM\" < 2010")
```

Hasil uji: 4 petak. Untuk Mahoni, ganti baris itu menjadi `petak.setSubsetString("\"JENIS\" = 'Mahoni'")`; hasil uji: 4 petak.

> **Pola yang bisa dipakai ulang:** Muat, periksa `isValid`, ubah, simpan, kembalikan. Ganti tiga hal: berkas dan layer (baris 9), saringan (baris 14), nama keluaran (baris 20 dan 22).

Pola 28.2: Loop fitur

Tujuan. Memeriksa tiap fitur satu per satu, mengumpulkan hasilnya ke kamus, dan memilih fitur yang memenuhi syarat.

Bahan. Layer `Petak` yang sama.

Analogi: **petugas dengan buku catatan** yang berjalan dari petak ke petak. Tiap petak ia catat, lalu di akhir ia menjumlahkan catatannya.

Skrip utuh:

```python
import os
from qgis.core import QgsVectorLayer, QgsFeatureRequest

DATA = os.environ.get("PY_DATA", r"D:/Latihan")
petak = QgsVectorLayer(DATA + "/paket-b2/Petak_KPH.gpkg|layername=Petak", "Petak", "ogr")
assert petak.isValid(), "Layer tidak terbuka. Cek DATA dan nama berkas."
```

```
# Tiap petak diperiksa satu per satu; hasilnya dikumpulkan ke kamus
luas_per_kph = {}
for f in petak.getFeatures():
    kph = f["NAMA_KPH"]
    ha = f.geometry().area() / 10000
    luas_per_kph[kph] = luas_per_kph.get(kph, 0) + ha
for kph, ha in sorted(luas_per_kph.items()):
    print("%-10s %6.2f ha" % (kph, ha))

# Hanya fitur yang memenuhi syarat
syarat = QgsFeatureRequest().setFilterExpression('"TAHUN_TANAM" < 2010')
tua = [f["KODE_PETAK"] for f in petak.getFeatures(syarat)]
print("Ditanam sebelum 2010:", tua)
```

Baris	Tulisan	Artinya
9	`luas_per_kph = {}`	Buku catatan kosong: kamus yang kuncinya nama KPH.
10	`for f in petak.getFeatures():`	Ulangi untuk tiap petak. `f` adalah petak yang sedang diperiksa (Bab 6 dan 15).
11	`kph = f["NAMA_KPH"]`	Ambil isi kolom `NAMA_KPH` petak ini.
12	`ha = f.geometry().area() / 10000`	Luas bentuk petak, dalam satuan peta, dibagi 10000 menjadi hektare. Benar hanya bila peta berbasis meter.
13	`luas_per_kph[kph] = luas_per_kph.get(kph, 0) + ha`	Tambahkan ke catatan KPH itu. `.get(kph, 0)` berarti "bila belum ada, anggap 0".
14-15	`for kph, ha in sorted(luas_per_kph.items()):`	Cetak catatan berurutan. `%-10s %6.2f` mengatur lebar kolom: `-10s` teks rata kiri 10 huruf, `6.2f` angka dua desimal selebar 6.
18	`QgsFeatureRequest().setFilterExpression('"TAHUN_TANAM" < 2010')`	Buat permintaan khusus: hanya petak yang ditanam sebelum 2010.
19	`[f["KODE_PETAK"] for f in petak.getFeatures(syarat)]`	Ambil kode petak yang lolos saringan, dalam satu baris (*list comprehension*, Bab 6).

Hasil. Layar menampilkan:

```
HASIL
KPH Alpha     4.00 ha
KPH Beta      4.00 ha
KPH Gamma     8.00 ha
Ditanam sebelum 2010: ['P-01', 'P-02', 'P-05', 'P-13']
```

Bagian yang boleh diubah.

- **Baris 11, nama kolom pengelompok.** Ganti `NAMA_KPH` dengan `JENIS` atau `STATUS` untuk pengelompokan lain.
- **Baris 12, pembagi.** `10000` untuk hektare. Hapus pembagi untuk meter persegi.
- **Baris 13, penjumlahan.** Ganti dengan `+ 1` untuk menghitung jumlah petak, bukan luas.

- **Baris 18, syarat.** Ekspresi QGIS biasa.

Jebakan.

- **Nilai kolom kosong.** Bila kolom berisi `NULL` atau kosong, penjumlahan atau perbandingan bisa gagal. Periksa dengan `if nilai is not None`. [CEK]
- **Pembagi sesuai satuan.** Lihat "Jebakan lintas bab", butir satuan luas.

Latihan variasi.

1. Hitung luas per **jenis** tanaman, bukan per KPH.
2. Tampilkan kode petak milik KPH Gamma.

Contoh jawaban:

```
PYTHON
    kph = f["JENIS"]
```

pada baris 11 (nama kotak tidak perlu diganti). Hasil uji: `{'Akasia': 2.94, 'Jati': 4.96, 'Mahoni': 4.04, 'Sengon': 4.06}`. Untuk nomor 2, ganti syarat baris 18 menjadi `'"NAMA_KPH" = \'KPH Gamma\''`. Hasil uji: `['P-09', 'P-10', 'P-11', 'P-12', 'P-13', 'P-14', 'P-15', 'P-16']`.

> **Pola yang bisa dipakai ulang:** Kamus kosong, ulangi tiap fitur, ambil kolom, tambahkan dengan `.get(kunci, 0)`. Ganti kolom pengelompok (baris 11), ukuran yang dijumlahkan (baris 12-13), dan syarat (baris 18).

Pola 28.3: Memanggil alat Processing

Tujuan. Menjalankan alat bawaan QGIS dari Python, dan merantai keluarannya menjadi masukan alat berikutnya. Ini inti Bab 18 sampai 23.

Bahan. `paket-i4/Sungai.gpkg` (2 garis sungai) dan `paket-i4/Batas_KPH.gpkg` (3 KPH).

Analogi: **ban berjalan di pabrik**. Hasil satu mesin langsung jadi bahan mesin berikutnya.

Skrip utuh:

```
PYTHON
import os
import processing

DATA = os.environ.get("PY_DATA", r"D:/Latihan")
sungai = DATA + "/paket-i4/Sungai.gpkg"
batas = DATA + "/paket-i4/Batas_KPH.gpkg"

# Keluaran alat pertama menjadi masukan alat kedua
sempadan = processing.run("native:buffer", {
    "INPUT": sungai, "DISTANCE": 30, "DISSOLVE": True,
    "OUTPUT": "TEMPORARY_OUTPUT"})["OUTPUT"]
irisan = processing.run("native:intersection", {
    "INPUT": batas, "OVERLAY": sempadan,
    "OUTPUT": "TEMPORARY_OUTPUT"})["OUTPUT"]
```

```
for f in irisan.getFeatures():
    print(f["NAMA_KPH"], round(f.geometry().area() / 10000, 2), "ha sempadan")
```

Baris	Tulisan	Artinya
2	`import processing`	Panggil perkakas pemanggil alat.
5-6	`sungai = ...` dan `batas = ...`	Jalur dua berkas. Alat Processing boleh diberi jalur berkas langsung.
9-11	`processing.run("native:buffer", {...})["OUTPUT"]`	Jalankan alat `native:buffer`. Isian alat diberikan sebagai kamus: nama isian lalu nilainya. `"OUTPUT"` di ujung mengambil hasil dari kamus jawaban.
10	`"DISTANCE": 30`	Lebar sempadan: 30 satuan peta (meter bila peta berbasis meter).
11	`"OUTPUT": "TEMPORARY_OUTPUT"`	Simpan hasil di memori, sementara. Hasilnya layer siap pakai.
12-14	`processing.run("native:intersection", {...})`	Alat kedua. Masukannya adalah keluaran alat pertama (`sempadan`).
16-17	`for f in irisan.getFeatures(): print(...)`	Ukur luas tiap irisan lalu cetak.

Hasil. Layar menampilkan:

```
HASIL
KPH Alpha 1.18 ha sempadan
KPH Beta 1.39 ha sempadan
KPH Gamma 1.5 ha sempadan
```

Bagian yang boleh diubah.

- **Baris 9, ID alat.** `"native:buffer"` diganti dengan alat lain. Cara mencari ID dibahas di "Cara mencari fungsi sendiri".
- **Baris 10, isian alat.** Nama isian harus persis seperti yang diminta alat.
- Baris 11, `"OUTPUT"`. Ganti dengan jalur berkas, misalnya `HASIL + "/sempadan.gpkg"`, bila hasil harus tersimpan.
- Baris 10, `"DISSOLVE": True`. `False` menghasilkan satu bentuk per garis.

Jebakan.

- **Hasil bisa teks, bukan layer.** Dengan `"TEMPORARY_OUTPUT"`, hasilnya `QgsVectorLayer`. Dengan jalur berkas, hasilnya hanya teks jalur. Uji: `QgsVectorLayer | str D:/.../buf.gpkg`.
- **Nama alat salah ketik.** `native:buffr` menghasilkan `QgsProcessingException Error: Algorithm native:buffr not found`.
- **Satuan jarak mengikuti CRS.** Lihat "Jebakan lintas bab".

Latihan variasi.

1. Lebarkan sempadan menjadi 50 meter.
2. Apa yang berubah pada luas sempadan?

Contoh jawaban: ganti `"DISTANCE": 30` menjadi `"DISTANCE": 50`. Hasil uji: KPH Alpha 1,69 ha, KPH Beta 2,32 ha, KPH Gamma 2,54 ha. Luas naik karena sempadan makin lebar.

> **Pola yang bisa dipakai ulang:** `hasil = processing.run("id:alat", {isian...})["OUTPUT"]`, lalu masukkan `hasil` ke alat berikutnya. Ganti ID alat, isian, dan nama kunci hasil.

Pola 28.4: Batch, banyak berkas dengan satu putaran

Tujuan. Menjalankan pekerjaan yang sama pada semua berkas dalam satu folder, lalu merangkum hasilnya. Ini pola Bab 21.

Bahan. `paket-i4/Ortofoto_Mentah/` berisi tiga ubin foto drone (`ubin_01.tif` sampai `ubin_03.tif`).

Analogi: **petugas pos memilah surat** satu per satu dengan cara yang sama.

Skrip utuh:

```
PYTHON
import os, glob
import processing

DATA = os.environ.get("PY_DATA", r"D:/Latihan")
daftar = sorted(glob.glob(DATA + "/paket-i4/Ortofoto_Mentah/*.tif"))
print("Ditemukan", len(daftar), "berkas")

ringkas = []
for jalur in daftar:
    hasil = processing.run("native:rasterlayerstatistics", {"INPUT": jalur, "BAND": 1})
    ringkas.append((os.path.basename(jalur), round(hasil["MEAN"], 2)))

for nama, rata in ringkas:
    print(nama, "rata-rata band 1:", rata)
print("Selesai:", len(ringkas), "dari", len(daftar))
```

Baris	Tulisan	Artinya
5	`daftar = sorted(glob.glob(DATA + "/paket-i4/Ortofoto_Mentah/*.tif"))`	`glob` mencari semua berkas dengan pola nama `*.tif` (bintang artinya "apa saja"). `sorted` mengurutkannya.
6	`print("Ditemukan", len(daftar), "berkas")`	Cetak jumlah yang ditemukan. Biasakan memeriksa angka ini.
8	`ringkas = []`	Daftar kosong untuk menampung hasil.
9	`for jalur in daftar:`	Ulangi tiap berkas.

Baris	Tulisan	Artinya
10	`processing.run("native:rasterlayerstatistics", {"INPUT": jalur, "BAND": 1})`	Hitung statistik band 1. Hasilnya kamus berisi `MEAN`, `MAX`, `MIN`, dan lain-lain.
11	`ringkas.append((os.path.basename(jalur), round(hasil["MEAN"], 2)))`	Simpan pasangan: nama berkas (tanpa folder) dan rata-rata dua desimal.
13-15	`for nama, rata in ringkas:` dst.	Cetak ringkasan dan hitung berapa yang selesai.

Hasil. Layar menampilkan:

```
HASIL
Ditemukan 3 berkas
ubin_01.tif rata-rata band 1: 2105.11
ubin_02.tif rata-rata band 1: 2097.24
ubin_03.tif rata-rata band 1: 2095.22
Selesai: 3 dari 3
```

Bagian yang boleh diubah.

- **Baris 5, folder dan pola nama.** `"*.tif"` diganti `"*.gpkg"` untuk vektor.
- Baris 10, `"BAND": 1`. Nomor band. Salah nomor tidak menimbulkan galat, tetapi angkanya lain.
- Baris 11, `hasil["MEAN"]`. Ganti kunci `MAX` untuk nilai tertinggi.
- **Baris 10, alat.** Ganti dengan alat lain yang menerima satu berkas.

Jebakan.

- **Folder kosong.** Bila `daftar` kosong, putaran tidak jalan dan tidak ada pesan galat. Baris 6 ada untuk menangkap kasus ini.
- **Huruf besar pada ekstensi.** Pola `*.tif` di Windows tetap menemukan `.TIF`. Di Linux dan macOS tidak. [kemungkinan: perilaku sistem berkas, tidak diuji]
- **Satu berkas rusak menghentikan semua.** Pola ini belum tahan galat. Baca Pola 28.5.

Latihan variasi.

1. Hitung rata-rata **band 2**.
2. Tampilkan juga nilai tertinggi (`MAX`).

Contoh jawaban: ganti `"BAND": 1` menjadi `"BAND": 2`. Hasil uji: rata-rata 746,82; 743,77; dan 750,6. Nilai `MAX` band 2 untuk ketiganya 1199,0. Kunci lain yang tersedia: `COUNT`, `MIN`, `RANGE`, `STD_DEV`, `SUM`, `SUM_OF_SQUARES`.

> **Pola yang bisa dipakai ulang:** `glob` untuk daftar berkas, `for` untuk putaran, `append` untuk menampung hasil, cetak "selesai x dari y". Ganti folder dan pola nama (baris 5), alat dan isiannya (baris 10), dan kunci hasil (baris 11).

Pola 28.5: Tahan galat, satu berkas rusak tidak menghentikan semua

Tujuan. Memeriksa tiap berkas sebelum dipakai, mencatat alasan penolakan, dan tetap melanjutkan ke berkas berikutnya. Ini pola Bab 22.

Bahan. `paket-i4/Kiriman_Lapangan/`: enam berkas kiriman tim. Satu rusak (`tim_e`), satu kekurangan kolom (`tim_f`).

Analogi: **petugas gerbang** memeriksa tiap tamu. Yang tidak lengkap suratnya ditolak dengan catatan alasan, dan antrean tetap jalan.

Skrip utuh:

```
PYTHON
import os, glob
from qgis.core import QgsVectorLayer

DATA = os.environ.get("PY_DATA", r"D:/Latihan")


def muat_aman(jalur, kolom_wajib):
    layer = QgsVectorLayer(jalur, "uji", "ogr")
    if not layer.isValid():
        raise ValueError("layer tidak terbuka")
    kurang = [k for k in kolom_wajib if layer.fields().indexOf(k) < 0]
    if kurang:
        raise ValueError("kolom kurang: " + ", ".join(kurang))
    return layer


diterima, ditolak = [], []
for jalur in sorted(glob.glob(DATA + "/paket-i4/Kiriman_Lapangan/*.gpkg")):
    nama = os.path.basename(jalur)
    try:
        layer = muat_aman(jalur, ["Kondisi", "Tinggi_Phn"])
        diterima.append((nama, layer.featureCount()))
    except ValueError as galat:
        ditolak.append((nama, str(galat)))

print("DITERIMA:", len(diterima))
for nama, n in diterima:
    print("  ", nama, n, "titik")
print("DITOLAK:", len(ditolak))
for nama, alasan in ditolak:
    print("  ", nama, "->", alasan)
```

Baris	Tulisan	Artinya
7	`def muat_aman(jalur, kolom_wajib):`	Fungsi pemeriksa: menerima jalur dan daftar kolom yang wajib ada (Bab 7).
9-10	`if not layer.isValid(): raise ValueError(...)`	Bila layer tak terbuka, lempar galat berpesan. `raise` melempar, `ValueError` adalah jenis galatnya (Bab 10).
11	`kurang = [k for k in kolom_wajib if layer.fields().indexOf(k) < 0]`	Kumpulkan kolom yang tidak ada. `indexOf` bernilai negatif bila kolom tidak ditemukan.

Baris	Tulisan	Artinya
12-13	`if kurang: raise ValueError(...)`	Bila ada yang kurang, lempar galat memuat daftarnya.
14	`return layer`	Bila lolos semua, kembalikan layernya.
17	`diterima, ditolak = [], []`	Dua daftar kosong sekaligus.
20-24	`try: ... except ValueError as galat: ...`	Coba muat. Bila ada `ValueError`, catat alasannya di `ditolak` dan **lanjutkan** putaran.
26-31	`print(...)`	Cetak laporan: yang diterima dan yang ditolak beserta alasannya.

Hasil. Layar menampilkan:

```
HASIL
DITERIMA: 4
   tim_a.gpkg 6 titik
   tim_b.gpkg 5 titik
   tim_c.gpkg 4 titik
   tim_d.gpkg 3 titik
DITOLAK: 2
   tim_e.gpkg -> layer tidak terbuka
   tim_f.gpkg -> kolom kurang: Kondisi
```

Bagian yang boleh diubah.

- **Baris 21, daftar kolom wajib.** `["Kondisi", "Tinggi_Phn"]` diganti kolom yang harus ada pada data Anda.
- **Baris 18, folder dan pola nama.**
- **Baris 9-14, aturan pemeriksaan.** Tambahkan aturan baru dengan `if ...: raise ValueError("alasan")`.
- **Baris 23, jenis galat.** Tangkap jenis lain bila fungsi Anda melemparnya.

Jebakan.

- **Layer rusak tidak melempar galat.** `QgsVectorLayer` untuk berkas rusak hanya memberi `isValid()` bernilai `False`. Karena itu baris 9-10 perlu.
- `except` yang terlalu lebar. `except Exception` menangkap segalanya, termasuk salah ketik di kode Anda sendiri, yang lalu tampak seperti "berkas ditolak". Tangkap jenis galat yang spesifik.

Latihan variasi.

1. Tolak juga kiriman yang berisi kurang dari 4 titik.
2. Berkas apa yang ditolak sekarang?

Contoh jawaban: tambahkan sebelum `return layer` pada fungsi `muat_aman`:

```
PYTHON
    if layer.featureCount() < 4:
        raise ValueError("terlalu sedikit titik")
```

Hasil uji: `tim_d.gpkg` ditolak dengan alasan `terlalu sedikit titik`, selain `tim_e` (layer tidak terbuka) dan `tim_f` (kolom kurang: Kondisi).

> **Pola yang bisa dipakai ulang:** Fungsi pemeriksa yang melempar `ValueError` berpesan, dibungkus `try/except` di dalam putaran, dengan dua daftar: diterima dan ditolak. Ganti aturan pemeriksaan, kolom wajib, dan lokasi berkas.

Pola 28.6: Ekspresi QGIS dari Python

Tujuan. Memakai bahasa ekspresi QGIS (yang sama dengan Field Calculator) dari Python, untuk menghitung nilai dan memilih fitur. Ini pola Bab 22.

Bahan. Layer `Petak`.

Analogi: ekspresi adalah **rumus Excel** yang dipakai QGIS. Anda menulis rumusnya sebagai teks, QGIS yang menghitung.

Skrip utuh:

```python
import os
from qgis.core import (QgsVectorLayer, QgsExpression, QgsExpressionContext,
                       QgsExpressionContextUtils)

DATA = os.environ.get("PY_DATA", r"D:/Latihan")
petak = QgsVectorLayer(DATA + "/paket-b2/Petak_KPH.gpkg|layername=Petak", "Petak", "ogr")
assert petak.isValid(), "Layer tidak terbuka. Cek DATA dan nama berkas."

# Ekspresi QGIS dihitung dari Python
ekspresi = QgsExpression("round(area($geometry) / 10000, 2) || ' ha, ' || upper(\"JENIS\")")
assert not ekspresi.hasParserError(), ekspresi.parserErrorString()
konteks = QgsExpressionContext()
konteks.appendScopes(QgsExpressionContextUtils.globalProjectLayerScopes(petak))
for i, f in enumerate(petak.getFeatures()):
    konteks.setFeature(f)
    print(f["KODE_PETAK"], "->", ekspresi.evaluate(konteks))
    if i == 2:
        break

# Ekspresi yang sama untuk memilih fitur
petak.selectByExpression("\"STATUS\" = 'Rehabilitasi'")
print("Terpilih:", petak.selectedFeatureCount(), "petak")
petak.removeSelection()
```

Baris	Tulisan	Artinya
10	`ekspresi = QgsExpression("round(area($geometry) / 10000, 2) ... upper(...)")`	Buat ekspresi dari teks: luas dalam hektare, disambung teks " ha, ", disambung nama jenis dalam huruf besar. Penyambung teks dalam ekspresi QGIS berupa dua tanda garis tegak berurutan, bukan operator Python.
11	`assert not ekspresi.hasParserError(), ...`	Periksa tata bahasa ekspresi. Bila salah tulis, hentikan dengan pesan penjelasnya.

Baris	Tulisan	Artinya
12-13	`konteks = QgsExpressionContext()` dan `appendScopes(...)`	Siapkan "ruang kerja" ekspresi: ia perlu tahu layer dan proyek yang dipakai.
14-16	`for i, f in enumerate(...)`, `konteks.setFeature(f)`, `ekspresi.evaluate(konteks)`	Untuk tiap petak, tunjuk petak itu ke ruang kerja, lalu hitung ekspresinya.
17-18	`if i == 2: break`	Berhenti setelah tiga petak (nomor mulai 0).
21-22	`petak.selectByExpression(...)`	Pilih petak dengan ekspresi, lalu cetak jumlah yang terpilih.
23	`petak.removeSelection()`	Hapus pilihan agar layer bersih.

Hasil. Layar menampilkan:

```
HASIL
P-01 -> 1.01 ha, JATI
P-02 -> 0.93 ha, JATI
P-03 -> 1 ha, MAHONI
Terpilih: 2 petak
```

Bagian yang boleh diubah.

- **Baris 10, teks ekspresi.** Satu-satunya tempat rumus ditulis. Uji dulu di Field Calculator QGIS.
- **Baris 21, ekspresi pilihan.** Kolom dan nilai.
- **Baris 17, angka 2.** Jumlah petak yang ditampilkan dikurangi satu.

Jebakan.

- **Ekspresi salah tulis.** `round(area($geometry` menghasilkan `Incomplete expression. You might not have finished the full expression., expecting COMMA or ')'`. Itu sebabnya baris 11 ada.
- **Nama kolom salah.** Ekspresi `"JENISS" || 'x'` tidak menimbulkan galat Python. Kegagalannya baru terbaca lewat `hasEvalError()` dan pesannya `Field 'JENISS' not found`. Bila Anda tidak memeriksa, hasilnya diam-diam kosong.
- `$area` dan `area($geometry)`. Keduanya sama (10100,0) pada proyek tanpa pengaturan elipsoid. Setelah elipsoid proyek diatur ke `EPSG:7030`, `$area` menjadi 10099,2465 sedangkan `area($geometry)` tetap 10100,0.

Latihan variasi.

1. Hitung umur tanaman: 2026 dikurangi `TAHUN_TANAM`.
2. Pilih petak berstatus `Lindung`.

Contoh jawaban: ganti isi `QgsExpression(...)` menjadi `QgsExpression("2026 - \"TAHUN_TANAM\"")`. Hasil uji: P-01 umur 18, P-02 umur 18, P-03 umur 14. Untuk nomor 2, ganti baris 21 menjadi `petak.selectByExpression("\"STATUS\" = 'Lindung'")`; hasil uji: 2 petak. Tahun 2026 tertulis tetap, jadi berubah maknanya pada tahun lain.

> **Pola yang bisa dipakai ulang:** `QgsExpression(teks)`, periksa `hasParserError`, siapkan konteks, `setFeature`, `evaluate`. Ganti teks ekspresi. Bagian lainnya tetap.

Pola 28.7: Baca dan tulis berkas CSV dan JSON

Tujuan. Membaca tabel dari CSV, merangkumnya, lalu menulis hasilnya ke CSV dan JSON. Ini pola yang dipakai di Bab 14, 22, dan 27.

Bahan. `paket-b2/Inventarisasi.csv`: volume pohon per petak (16 baris data), dengan kolom `KODE_PETAK`, `JML_POHON`, `VOLUME_M3`, `TGL_UKUR`, `REGU`.

Analogi: **menyalin dari buku tamu ke buku laporan**. Anda membaca satu halaman, merangkum, lalu menulis lembar baru.

Skrip utuh:

```
PYTHON
import os, csv, json

DATA = os.environ.get("PY_DATA", r"D:/Latihan")
HASIL = os.environ.get("PY_HASIL", r"D:/Latihan/hasil")
os.makedirs(HASIL, exist_ok=True)

# Baca: jumlahkan volume per regu
volume = {}
with open(DATA + "/paket-b2/Inventarisasi.csv", encoding="utf-8") as fh:
    for baris in csv.DictReader(fh):
        regu = baris["REGU"]
        volume[regu] = volume.get(regu, 0) + float(baris["VOLUME_M3"])

# Tulis: CSV
with open(HASIL + "/volume_per_regu.csv", "w", newline="", encoding="utf-8") as fh:
    tulis = csv.writer(fh)
    tulis.writerow(["REGU", "VOLUME_M3"])
    for regu, v in sorted(volume.items()):
        tulis.writerow([regu, round(v, 1)])

# Tulis: JSON
volume = {regu: round(v, 1) for regu, v in volume.items()}
with open(HASIL + "/volume_per_regu.json", "w", encoding="utf-8") as fh:
    json.dump(volume, fh, ensure_ascii=False, indent=2)
print(volume)
```

Baris	Tulisan	Artinya
1	`import os, csv, json`	Tiga perkakas: jalur, tabel teks, dan format JSON.
9-10	`with open(..., encoding="utf-8") as fh:` dan `csv.DictReader(fh)`	Buka CSV dan baca baris demi baris sebagai kamus (Bab 9).
11-12	`regu = baris["REGU"]` dan `volume[regu] = ...`	Jumlahkan `VOLUME_M3` per regu. `float(...)` mengubah teks jadi angka.
15	`open(..., "w", newline="", encoding="utf-8")`	Buka berkas untuk **menulis** (`"w"`). `newline=""` mencegah baris kosong ganda di Windows.

Baris	Tulisan	Artinya
16-19	`csv.writer(fh), writerow(...)`	Tulis baris judul, lalu satu baris per regu.
22	`volume = {regu: round(v, 1) for regu, v in volume.items()}`	Bulatkan semua nilai ke satu desimal. Ini *dictionary comprehension* (Bab 5 dan 6).
23-24	`json.dump(volume, fh, ensure_ascii=False, indent=2)`	Tulis kamus sebagai JSON rapi. `indent=2` memberi lekuk dua spasi.

Hasil. Layar menampilkan:

```
HASIL
{'Regu 1': 1107.6, 'Regu 2': 990.5, 'Regu 3': 702.1}
```

Berkas `volume_per_regu.csv` berisi empat baris (judul dan tiga regu). Berkas `volume_per_regu.json` berisi kamus yang sama dalam bentuk teks JSON.

Bagian yang boleh diubah.

- **Baris 9, nama berkas CSV.**
- **Baris 11-12, kolom pengelompok dan kolom angka.** Nama kolom persis seperti judul CSV.
- Baris 15, `encoding`. `"utf-8-sig"` bila CSV akan dibuka di Excel dan memuat huruf non-ASCII. Diuji: berkas diawali tiga byte `\xef\xbb\xbf` (BOM). [CEK: tampilan di Excel tidak diuji]
- **Baris 23-24, JSON.** Hapus bila tidak diperlukan.

Jebakan.

- **Nama kolom CSV peka huruf besar-kecil.** `baris["kode_petak"]` menghasilkan `KeyError: 'kode_petak'`. (Diuji.)
- **Data kotor.** Di CSV ini, kode petak `P-09` tertulis `p-09`, `P-12` tidak ada, dan `P-17` tidak ada di peta. Pola 28.8 menangani ketiganya.
- **Berkas terkunci.** Lihat "Jebakan lintas bab".

Latihan variasi.

1. Hitung **rata-rata** jumlah pohon (`JML_POHON`) per regu.
2. Simpan CSV dengan `utf-8-sig`.

Contoh jawaban, mengganti baris 11-12 dengan dua kamus:

```
PYTHON
        jml[regu] = jml.get(regu, 0) + int(baris["JML_POHON"])
        n[regu] = n.get(regu, 0) + 1
```

dengan `jml, n = {}, {}` lebih dulu, lalu `{r: round(jml[r] / n[r], 1) for r in sorted(jml)}`. Hasil uji: `{'Regu 1': 886.3, 'Regu 2': 792.2, 'Regu 3': 900.2}`.

> **Pola yang bisa dipakai ulang:** Baca dengan `DictReader`, rangkum ke kamus, tulis dengan `csv.writer` atau `json.dump`. Ganti nama berkas, kolom, dan cara merangkum.

Pola 28.8: Membuat laporan dari beberapa sumber

Tujuan. Menggabungkan layer dan CSV menjadi laporan teks rapi, sambil menangani data kotor. Ini pola penutup Bab 14, 17, dan 22.

Bahan. Layer `Petak` dan `Inventarisasi.csv`.

Analogi: **meja redaksi**. Dua wartawan membawa bahan berbeda. Redaktur mencocokkan, merapikan, lalu menulis satu halaman.

Skrip utuh:

```python
import os, csv
from qgis.core import QgsVectorLayer

DATA = os.environ.get("PY_DATA", r"D:/Latihan")
HASIL = os.environ.get("PY_HASIL", r"D:/Latihan/hasil")
os.makedirs(HASIL, exist_ok=True)

petak = QgsVectorLayer(DATA + "/paket-b2/Petak_KPH.gpkg|layername=Petak", "Petak", "ogr")
assert petak.isValid(), "Layer tidak terbuka. Cek DATA dan nama berkas."

# Bahan 1: KPH tiap petak dan luas per KPH, dari layer
kph_petak, luas = {}, {}
for f in petak.getFeatures():
    kph = f["NAMA_KPH"]
    kph_petak[f["KODE_PETAK"]] = kph
    luas[kph] = luas.get(kph, 0) + f.geometry().area() / 10000

# Bahan 2: volume per petak dari CSV, dijumlahkan per KPH
volume, kode_csv, asing = {}, set(), []
with open(DATA + "/paket-b2/Inventarisasi.csv", encoding="utf-8") as fh:
    for b in csv.DictReader(fh):
        kode = b["KODE_PETAK"].strip().upper()
        if kode not in kph_petak:
            asing.append(kode)
            continue
        kode_csv.add(kode)
        kph = kph_petak[kode]
        volume[kph] = volume.get(kph, 0) + float(b["VOLUME_M3"])

# Rakit teks laporan
baris = ["LAPORAN RINGKAS KPH CONTOH", "-" * 38,
         f"{'KPH':<10}{'Luas (ha)':>12}{'Volume (m3)':>16}"]
for kph in sorted(luas):
    baris.append(f"{kph:<10}{luas[kph]:>12.2f}{volume.get(kph, 0):>16.1f}")
baris.append("-" * 38)
baris.append(f"{'Total':<10}{sum(luas.values()):>12.2f}{sum(volume.values()):>16.1f}")
baris.append("Petak tanpa data volume: " + ", ".join(sorted(set(kph_petak) - kode_csv)))
baris.append("Kode di CSV yang tidak ada di peta: " + ", ".join(asing))

teks = "\n".join(baris)
with open(HASIL + "/laporan_kph.txt", "w", encoding="utf-8") as fh:
    fh.write(teks + "\n")
print(teks)
```

Baris	Tulisan	Artinya
12-16	`kph_petak, luas = {}, {}` dan putaran fitur	Bahan 1: dua kamus. `kph_petak` menyimpan KPH tiap petak, `luas` menjumlahkan luas per KPH.
19	`volume, kode_csv, asing = {}, set(), []`	Bahan 2: kamus volume, himpunan (`set`) kode yang dikenal, dan daftar kode asing.
22	`kode = b["KODE_PETAK"].strip().upper()`	**Rapikan kode**: buang spasi di ujung dan ubah ke huruf besar. Ini yang menyelamatkan `p-09`.
23-25	`if kode not in kph_petak: asing.append(kode); continue`	Bila kode tidak ada di peta, catat sebagai asing dan lompat ke baris berikutnya.
26-28	`kode_csv.add(kode)` dst.	Catat bahwa kode ini ada di CSV, lalu tambahkan volumenya ke KPH-nya.
31-32	`baris = [...]`	Daftar baris laporan: judul, garis, dan judul kolom.
32	`f"{'KPH':<10}{'Luas (ha)':>12}{'Volume (m3)':>16}"`	*f-string*: teks dengan lubang `{...}`. `<10` rata kiri selebar 10, `>12` rata kanan selebar 12 (Bab 9).
33-34	`for kph in sorted(luas):`	Satu baris laporan per KPH. `:>12.2f` berarti angka dua desimal, rata kanan, selebar 12.
36-38	`Total`, petak tanpa data, kode asing	Baris total, lalu dua catatan penting tentang data yang tidak cocok.
40-43	`"\n".join(baris)`, `fh.write(...)`, `print(teks)`	Sambung semua baris dengan pindah baris, simpan ke berkas, dan tampilkan.

Hasil. Layar menampilkan:

```
HASIL
LAPORAN RINGKAS KPH CONTOH
--------------------------------------
KPH          Luas (ha)     Volume (m3)
KPH Alpha         4.00           939.2
KPH Beta          4.00           596.0
KPH Gamma         8.00          1226.5
--------------------------------------
Total            16.00          2761.7
Petak tanpa data volume: P-12
Kode di CSV yang tidak ada di peta: P-17
```

Cek silang: jumlah volume di CSV seluruhnya 2800,2. Angka 2761,7 ditambah volume P-17 (38,5) sama dengan 2800,2. Jadi tidak ada volume yang hilang tanpa terhitung.

Bagian yang boleh diubah.

- **Baris 8 dan 20, sumber data.**
- **Baris 14-16, kolom pengelompok.** `NAMA_KPH` diganti kolom lain.
- **Baris 22, perapian kode.** Tambah `.replace(" ", "")` bila ada spasi di tengah.
- **Baris 31-38, tata letak laporan.** Lebar kolom dalam f-string.

- **Baris 41, nama berkas laporan.**

Jebakan.

- **Tanpa perapian kode, skrip berhenti.** Bila baris 22 ditulis `kode = b["KODE_PETAK"]` dan dipakai langsung sebagai kunci di `kph_petak`, hasilnya `KeyError: 'p-09'`. (Terjadi pada percobaan pertama bab ini.)
- **Data yang hilang tidak menimbulkan galat.** P-12 tidak muncul di CSV dan P-17 tidak muncul di peta. Tanpa baris 37-38 Anda tidak akan tahu.
- **Total tidak sama dengan seluruh CSV.** Laporan hanya menjumlahkan yang cocok dengan peta. Itu disengaja, tetapi harus dicatat.

Latihan variasi.

1. Tambahkan kolom volume per hektare.

Contoh jawaban, mengganti baris 31-34:

```
PYTHON
baris = ["LAPORAN RINGKAS KPH CONTOH", "-" * 50,
         f"{'KPH':<10}{'Luas (ha)':>12}{'Volume (m3)':>16}{'m3 per ha':>12}"]
for kph in sorted(luas):
    vol = volume.get(kph, 0)
    baris.append(f"{kph:<10}{luas[kph]:>12.2f}{vol:>16.1f}{vol / luas[kph]:>12.1f}")
```

Garis pemisah di baris 35 diubah menjadi `"-" * 50`. Hasil uji: KPH Alpha 234,8; KPH Beta 149,0; KPH Gamma 153,3 meter kubik per hektare. Angka ini dihitung dari luas seluruh petak, termasuk P-12 yang tidak punya data volume, jadi cenderung terlalu rendah untuk KPH Gamma.

> **Pola yang bisa dipakai ulang: Kumpulkan bahan ke kamus, rapikan kunci, catat yang tidak cocok, rakit teks dengan f-string, simpan dan cetak.** Ganti sumber data, kolom, dan tata letak.

Tabel: bila data saya berubah, bagian mana yang saya ganti?

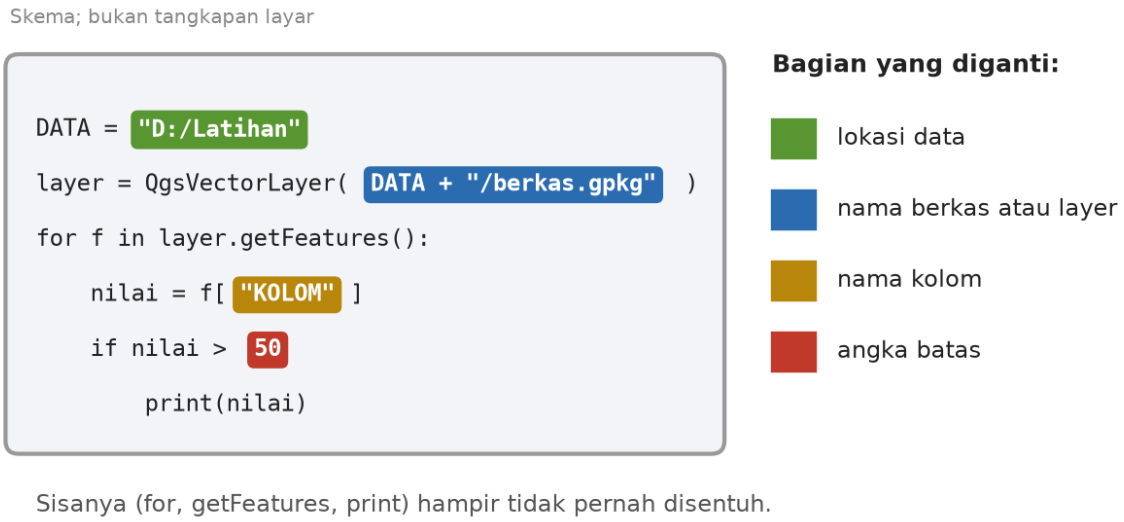
Tabel ini rujukan cepat. Cari baris yang cocok dengan keadaan Anda.

Bila data saya berubah begini	Bagian yang diganti	Ada di pola
Folder data pindah ke komputer lain	`DATA` (dan `HASIL`) di awal skrip	Semua
Berkas atau layer berganti nama	Teks sumber pada `QgsVectorLayer(...)`	28.1, 28.2, 28.6, 28.8
Nama kolom berbeda	Teks di dalam `f["..."]` dan ekspresi	28.2, 28.6, 28.7, 28.8
Satu GeoPackage berisi banyak layer	Bagian sesudah garis tegak: `layername=...`	28.1
Data berada di CRS lain	Ubah dulu ke CRS meter dengan `native:reprojectlayer`	28.3 dan semua yang mengukur
Satuan hasil perlu beda (meter persegi, hektare)	Pembagi `10000`	28.2, 28.8

Bila data saya berubah begini	Bagian yang diganti	Ada di pola
Jumlah berkas bertambah	Tidak ada. Putaran mengikuti isi folder	28.4, 28.5
Ekstensi berkas berbeda	Pola nama di `glob`	28.4, 28.5
Raster punya urutan band lain	Angka `BAND`	28.4
Kolom wajib bertambah	Daftar `kolom_wajib`	28.5
Aturan seleksi berubah	Teks saringan atau ekspresi	28.1, 28.2, 28.6
Alat Processing diganti	ID alat dan nama isiannya	28.3, 28.4
CSV memuat huruf besar-kecil tidak seragam	Rapikan dengan `.strip().upper()`	28.8
Laporan perlu kolom tambahan	Daftar judul dan baris f-string	28.8

Aturan praktisnya: **ubah satu hal, jalankan, periksa hasilnya, baru ubah hal berikutnya.** Bila hasil salah, Anda langsung tahu penyebabnya.

Ilustrasi 28.2: Kerangka dan bagian yang diganti



Jebakan lintas bab

Berikut kesalahan yang berulang di banyak bab. Setiap butir sudah diuji, dengan pesan nyata.

1. Jalur salah, tanpa galat. Membuka layer dari jalur yang salah tidak langsung gagal:

```python
x = QgsVectorLayer(DATA + "/paket-b2/Petak_KPH_X.gpkg|layername=Petak", "Petak", "ogr")
print(x.isValid(), x.featureCount())
print(next(x.getFeatures())["JENIS"])
```

```
HASIL
False -2
StopIteration
```

Angka -2 artinya jumlah tidak diketahui. Skrip baru gagal di baris berikutnya dengan `StopIteration`. Karena itu selalu periksa `isValid()` (Pola 28.1). Dijumpai di Bab 14, 15, dan 16.

2. CRS dan satuan jarak. Jarak `30` pada alat `native:buffer` berarti 30 satuan peta. Pada peta UTM itu 30 meter. Pada peta berderajat (EPSG:4326) itu 30 derajat:

```
HASIL
buffer 30 pada derajat -> extent 79.4,-30.0 : 139.2,29.9
buffer 30 pada UTM -> extent 311970.1,9996030.1 : 312429.9,9996299.8
```

Sungai di data latihan hanya membentang 400 meter. Dengan CRS derajat, sempadannya selebar benua. Tidak ada pesan galat. Perbaikan: ubah dulu ke CRS berbasis meter. Dijumpai di Bab 18, 20, dan 27 (kerapatan titik panas).

3. Huruf besar dan kecil. Aturannya tidak sama di semua tempat:

Tempat	`JENIS` ditulis `jenis`	Hasil pada uji
Kolom fitur QGIS: `f["jenis"]`	Berhasil	`Jati`
Ekspresi QGIS: `"jenis"`	Berhasil	`Jati`
`fields().indexOf("jenis")`	Gagal	`-1`
`fields().lookupField("jenis")`	Berhasil	`3`
Kolom CSV: `baris["kode_petak"]`	Gagal	`KeyError: 'kode_petak'`
Nilai teks: `"p-09" == "P-09"`	Gagal	`False`
Nama kolom salah ketik: `f["JENISS"]`	Gagal	`KeyError: 'JENISS'`

Kesimpulannya: jangan mengandalkan toleransi QGIS. Tulis nama kolom persis seperti aslinya. Nilai teks dari sumber luar, seperti kode di CSV, rapikan dengan `.strip().upper()` (Pola 28.8).

4. Satuan luas. `area()` mengembalikan luas dalam satuan peta. Petak P-01 memberi `10100.0`, dan dibagi 10000 menjadi 1,01 hektare. Kolom `LUAS_LAMA_HA` di data yang sama berisi `1.0`, yaitu angka nominal lama, bukan hasil ukur. Jangan campur keduanya. Pada ekspresi, `$area` ikut pengaturan elipsoid proyek, sedangkan `area($geometry)` tidak (lihat Pola 28.6).

5. Indeks mulai dari 0. Daftar berisi 16 petak punya nomor 0 sampai 15:

```
HASIL
jumlah 16 | pertama: P-01 | terakhir: P-16
IndexError list index out of range
```

Pesan kedua muncul bila Anda meminta `semua[16]`. Indeks `[-1]` mengambil yang terakhir. Dijumpai di Bab 4, 5, dan 15.

6. File terkunci. Selama sebuah berkas masih terbuka di QGIS atau program lain, Windows menolak menghapus atau menimpanya:

```
HASIL
PermissionError [WinError 32] The process cannot access the file because it is being used by another process:
'C:\temp\tmpmd9xbtg9/petak_salin.gpkg'
```

Perbaikan: tutup layer atau lepaskan dari proyek, tutup program lain yang membukanya (misalnya Excel untuk CSV), lalu ulangi. Pesan ini khas Windows.

7. `proyek.clear()` mengosongkan proyek Anda. Beberapa skrip contoh memanggil `QgsProject.instance().clear()`. Di Konsol QGIS, itu membuang semua layer proyek yang sedang terbuka:

```
HASIL
sebelum: 2 layer
sesudah: 0 layer
```

Jangan jalankan skrip seperti itu di proyek penting. Simpan proyek lebih dulu, atau pakai proyek kosong. Dijumpai di Bab 14, 17, dan 23.

8. Hasil yang bukan layer. Hasil `processing.run` dengan `"OUTPUT"` jalur berkas adalah teks, bukan layer. Dengan `"TEMPORARY_OUTPUT"` hasilnya layer: `QgsVectorLayer | str`. Dijumpai di Bab 21.

9. Saringan yang menempel. `setSubsetString` tetap berlaku sampai dikosongkan. Hasil uji: 5 petak dengan saringan, 16 tanpa.

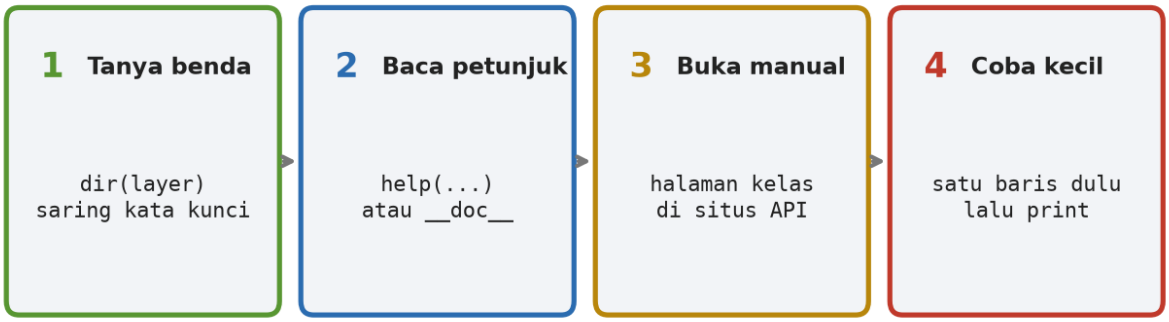
10. Data yang kotor. Kode `p-09` huruf kecil, kode yang hilang, dan kode yang berlebih adalah kejadian sehari-hari. Selalu cetak apa yang tidak cocok (Pola 28.8).

Cara mencari fungsi sendiri

Tidak ada yang hafal semua fungsi QGIS. Pembuat skrip yang mahir hanya tahu cara mencarinya. Bab 13 membahas kebiasaan belajar mandiri ini lebih panjang. Berikut ringkasannya, empat langkah yang cukup.

Ilustrasi 28.3: Empat langkah mencari fungsi

Skema; bukan tangkapan layar



Bila belum ketemu, ulangi dari langkah 1 dengan kata kunci lain.

Skrip berikut menjalankan langkah 1 sampai 3 pada layer petak, dan juga mencari alat Processing.

```python
import os
from qgis.core import QgsVectorLayer, QgsApplication

DATA = os.environ.get("PY_DATA", r"D:/Latihan")
petak = QgsVectorLayer(DATA + "/paket-b2/Petak_KPH.gpkg|layername=Petak", "Petak", "ogr")

# 1. Apa yang bisa dilakukan sebuah layer? Saring daftar namanya dengan kata kunci.
print([n for n in dir(petak) if "select" in n.lower()][:8])

# 2. Baca petunjuk satu fungsi (tiga baris pertama)
print("\n".join(petak.selectByExpression.__doc__.splitlines()[:3]))

# 3. Cari alat Processing dari kata kunci
for alat in QgsApplication.processingRegistry().algorithms():
    if "buffer" in alat.id().lower():
        print(alat.id(), "->", alat.displayName())

# 4. Lihat "formulir" satu alat
alat = QgsApplication.processingRegistry().algorithmById("native:buffer")
for p in alat.parameterDefinitions():
    print(p.name(), "|", p.description())
```

Baris	Tulisan	Artinya
8	`dir(petak)`	Daftar semua nama yang dimiliki benda `petak`.
8	`[n for n in dir(petak) if "select" in n.lower()][:8]`	Saring daftar itu: hanya nama berisi "select". `[:8]` mengambil delapan pertama.
11	`petak.selectByExpression.__doc__`	Teks petunjuk bawaan sebuah fungsi. `splitlines()[:3]` mengambil tiga baris pertama.
14-16	`QgsApplication.processingRegistry().algorithms()`	Daftar semua alat Processing. Saring dengan kata kunci "buffer".
19-21	`algorithmById("native:buffer")` dan `parameterDefinitions()`	Ambil satu alat, lalu lihat semua isian formulirnya.

Hasil. Layar menampilkan (daftar alat bisa berbeda di komputer Anda, bergantung penyedia yang terpasang, misalnya GRASS):

HASIL

```
['AddToSelection', 'IntersectSelection', 'RemoveFromSelection', 'SelectBehavior', 'SetSelection',
'boundingBoxOfSelected', 'deleteSelectedFeatures', 'deselect']
selectByExpression(self, expression: str|None, behavior: Qgis.SelectBehavior =
Qgis.SelectBehavior.SetSelection, context: QgsExpressionContext|None = None)
Selects matching features using an expression.

gdal:buffervectors -> Buffer vectors
gdal:onesidebuffer -> One side buffer
grass:r.buffer -> r.buffer
grass:r.buffer.lowmem -> r.buffer.lowmem
grass:v.buffer -> v.buffer
native:buffer -> Buffer
native:bufferbym -> Variable width buffer (by M value)
native:multiringconstantbuffer -> Multi-ring buffer (constant distance)
native:singlesidedbuffer -> Single sided buffer
native:taperedbuffer -> Tapered buffers
native:wedgebuffers -> Create wedge buffers
qgis:variabledistancebuffer -> Variable distance buffer
INPUT | Input layer
DISTANCE | Distance
SEGMENTS | Segments
END_CAP_STYLE | End cap style
JOIN_STYLE | Join style
MITER_LIMIT | Miter limit
DISSOLVE | Dissolve result
SEPARATE_DISJOINT | Keep disjoint results separate
OUTPUT | Buffered
```

Dari hasil ini Anda tahu: untuk memilih fitur ada fungsi `selectByExpression`, dan alat buffer standar berID `native:buffer` dengan sembilan isian. Nama isian di kolom kiri (`INPUT`, `DISTANCE`, dan seterusnya) adalah kunci kamus pada `processing.run`.

Dua cara lain melihat petunjuk sebuah alat lengkap dengan jenis tiap isian dan nilai yang diterima. Dari Python: `processing.algorithmHelp("native:buffer")`. Dari terminal: `qgis_process-qgis.bat help native:buffer`. Keduanya diuji dan mencetak deskripsi alat serta daftar isian.

Cara membaca dokumentasi API QGIS

API adalah daftar semua fungsi yang boleh Anda panggil. Dokumentasi PyQGIS ada di `https://qgis.org/pyqgis/master/`. Halaman utamanya mengelompokkan modul: Core, GUI, Analysis, Server, Processing, dan 3D, dengan kotak pencarian. Halaman itu menyediakan beberapa versi (pengembangan, rilis stabil, dan LTR). Pilih versi yang sesuai QGIS Anda. [CEK: daftar versi dibaca pada saat penulisan, bisa berubah]

Contoh jalur membaca untuk soal "bagaimana memilih fitur dengan ekspresi?":

1. Dari `dir(petak)` Anda tahu benda itu adalah `QgsVectorLayer`. Cetak `type(petak)` bila ragu.
2. Cari halaman kelas `QgsVectorLayer` di bagian Core: `https://qgis.org/pyqgis/master/core/QgsVectorLayer.html`.
3. Cari metode `selectByExpression` di halaman itu (Ctrl+F). Halaman kelas memuat metode `getFeatures`, `featureCount`, `setSubsetString`, dan `selectByExpression` yang sudah Anda pakai di bab ini.
4. Baca tanda tangan metodenya, seperti yang tadi dicetak oleh `__doc__`.

Cara membaca satu tanda tangan:

Bagian	Artinya
`selectByExpression(`	Nama metode.
`self`	Benda yang memanggil. Anda tidak mengisinya.
`expression: str`	Isian pertama, wajib, berjenis teks.
`behavior: Qgis.SelectBehavior = Qgis.SelectBehavior.SetSelection`	Isian kedua, boleh dilewat. Nilai sesudah tanda sama dengan adalah nilai bawaan.
`context: ... = None`	Isian ketiga, juga boleh dilewat.
`->` sebuah jenis	Bila ada, jenis nilai yang dikembalikan, misalnya `-> int` pada `featureCount`.
`.. seealso::`	Rujukan ke fungsi terkait, berguna untuk menemukan tetangganya.
`since QGIS 3.26`	Fungsi atau isian itu baru ada sejak versi tersebut.

Untuk alat Processing, rujukan terbaik adalah petunjuk alatnya sendiri (`processing.algorithmHelp`). Untuk membangun plugin dan alat sendiri, PyQGIS Developer Cookbook di `https://docs.qgis.org/latest/en/docs/pyqgis_developer_cookbook/index.html` memuat bab "Using Vector Layers", "Expressions, Filtering and Calculating Values", "Tasks", "Developing Python Plugins", dan "Writing a Processing plugin", serta lembar contekan PyQGIS. [CEK: daftar bab dibaca saat penulisan, versi dokumen yang tampil saat itu QGIS 3.44]

Kebiasaan yang membantu:

- **Coba kecil dulu.** Satu fungsi, satu `print`. Jangan menulis tiga puluh baris sebelum mencoba.
- **Baca pesan galat dari bawah.** Baris terakhir memuat jenis dan penjelasannya (Bab 10).
- **Cari contoh di bab lain buku ini.** Gunakan tabel "Bila data saya berubah".

Latihan akhir: rekap petak dari tiga sumber

Saatnya menggabungkan pola. Kepala Seksi meminta satu tabel CSV berisi, untuk tiap petak: KPH, luas, volume dari inventarisasi, jumlah titik panas, dan status waspada. Petak dengan paling sedikit satu titik panas berstatus WASPADA. Petak tanpa data volume harus diberi catatan.

Pola yang dibutuhkan: **muat layer** (28.1), **panggil alat** `native:countpointsinpolygon` (28.3), **baca CSV yang kotor** (28.7), **loop fitur** (28.2), dan **tulis CSV serta cetak ringkasan** (28.7 dan 28.8). Coba kerjakan sendiri lebih dulu. Contoh jawaban yang sudah diuji:

```python
import os, csv
import processing
from qgis.core import QgsVectorLayer

DATA = os.environ.get("PY_DATA", r"D:/Latihan")
HASIL = os.environ.get("PY_HASIL", r"D:/Latihan/hasil")
os.makedirs(HASIL, exist_ok=True)

# 1. Muat petak (Pola 28.1)
petak = QgsVectorLayer(DATA + "/paket-b2/Petak_KPH.gpkg|layername=Petak", "Petak", "ogr")
assert petak.isValid(), "Layer tidak terbuka. Cek DATA dan nama berkas."

# 2. Hitung titik panas per petak dengan alat Processing (Pola 28.3)
hitung = processing.run("native:countpointsinpolygon", {
    "POLYGONS": petak, "POINTS": DATA + "/paket-i4/Hotspot_Nasional.gpkg",
    "FIELD": "TITIK_PANAS", "OUTPUT": "TEMPORARY_OUTPUT"})["OUTPUT"]

# 3. Baca volume dari CSV, dengan kode yang dirapikan (Pola 28.7)
volume = {}
with open(DATA + "/paket-b2/Inventarisasi.csv", encoding="utf-8") as fh:
    for b in csv.DictReader(fh):
        volume[b["KODE_PETAK"].strip().upper()] = float(b["VOLUME_M3"])

# 4. Loop fitur dan rakit baris rekap (Pola 28.2)
rekap = []
for f in hitung.getFeatures():
    kode = f["KODE_PETAK"]
    titik = int(f["TITIK_PANAS"])
    vol = volume.get(kode)
    rekap.append([kode, f["NAMA_KPH"], round(f.geometry().area() / 10000, 2),
                  vol if vol is not None else "",
                  titik,
                  "WASPADA" if titik >= 1 else "aman",
                  "" if vol is not None else "tanpa data volume"])

# 5. Tulis CSV dan cetak ringkasan (Pola 28.7 dan 28.8)
judul = ["KODE_PETAK", "NAMA_KPH", "LUAS_HA", "VOLUME_M3", "TITIK_PANAS", "STATUS", "CATATAN"]
with open(HASIL + "/rekap_petak.csv", "w", newline="", encoding="utf-8") as fh:
    tulis = csv.writer(fh)
    tulis.writerow(judul)
    tulis.writerows(rekap)

print("Petak:", len(rekap), "| titik panas:", sum(r[4] for r in rekap))
print("WASPADA:", [r[0] for r in rekap if r[5] == "WASPADA"])
print("Tanpa data volume:", [r[0] for r in rekap if r[6]])
```

Hasil. Layar menampilkan:

```
HASIL
Petak: 16 | titik panas: 12
WASPADA: ['P-03', 'P-05', 'P-06', 'P-07', 'P-08', 'P-10', 'P-11', 'P-12', 'P-13', 'P-14']
Tanpa data volume: ['P-12']
```

Berkas `rekap_petak.csv` memuat tujuh kolom dan 16 baris data. Contoh dua baris pertamanya:

```
HASIL
KODE_PETAK,NAMA_KPH,LUAS_HA,VOLUME_M3,TITIK_PANAS,STATUS,CATATAN
P-01,KPH Alpha,1.01,246.8,0,aman,
P-02,KPH Alpha,0.93,235.0,0,aman,
```

Pemeriksaan hasil: 12 titik panas terhitung, sama dengan jumlah titik di `Hotspot_Nasional`. P-12 punya titik panas tetapi tidak punya data volume, jadi ia muncul di dua daftar.

Tantangan lanjutan.

1. Ubah ambang WASPADA menjadi dua titik atau lebih. Petak mana yang tersisa?
2. Tambahkan kolom `VOLUME_PER_HA`.
3. Bagaimana skrip ini dibuat tahan galat bila `Inventarisasi.csv` tidak ada?

Petunjuk jawaban: untuk nomor 1, ganti `titik >= 1` pada baris 33 menjadi `titik >= 2`. Hanya P-07 (3 titik) yang lolos, karena petak lain berisi satu titik (lihat hasil `kerapatan` di Skrip 27.4). Untuk nomor 3, bungkus bagian baca CSV dengan `try/except FileNotFoundError` dan siapkan `volume = {}` kosong di bagian `except`.

Cek paham

1. Sebutkan tiga hal yang biasanya diganti bila data baru datang.
2. Mengapa pola batch perlu dibungkus tahan galat?
3. Apa beda `QgsVectorLayer` yang jalurnya salah dengan layer yang jalurnya benar?

Jawaban.

1. Lokasi data (`DATA`), nama berkas atau layer, dan nama kolom. Kadang ditambah angka batas atau nomor band.
2. Satu berkas rusak bisa menghentikan seluruh putaran. Dengan `try/except` dan daftar ditolak, putaran jalan terus dan alasannya tercatat.
3. Layer berjalur salah tidak menimbulkan galat. `isValid()` bernilai `False` dan `featureCount()` bernilai -2. Galat baru muncul di baris berikutnya.

Kesalahan umum

- **Menyalin pola tanpa mengganti jalur contoh.** Perbaikan: ganti `DATA` dan `HASIL` di awal setiap skrip.

- **Mengubah banyak hal sekaligus.** Bila gagal, Anda tidak tahu penyebabnya. Perbaikan: ubah satu hal, jalankan, periksa.
- **Percaya pada data tanpa memeriksa.** Kode huruf kecil, baris hilang, dan kolom kurang itu biasa. Perbaikan: cetak yang tidak cocok (Pola 28.5 dan 28.8).
- **Menjalankan skrip bekas di proyek penting.** Skrip dengan `clear()` atau `removeAllMapLayers()` mengosongkan proyek. Perbaikan: simpan proyek dulu atau pakai proyek kosong.

Ringkasan dan latihan

- Skrip GIS tersusun dari delapan pola: muat-ubah-simpan, loop fitur, panggil alat, batch, tahan galat, ekspresi, baca-tulis berkas, laporan.
- Data baru biasanya hanya menuntut penggantian lokasi, nama berkas, nama kolom, dan angka.
- Jebakan lintas bab: jalur, CRS dan satuan, huruf besar-kecil, indeks 0, file terkunci, dan `clear()`.
- Mencari fungsi: `dir`, `__doc__`, daftar alat Processing, lalu dokumentasi API.
- Mengubah satu hal, menjalankan, memeriksa, adalah kebiasaan paling berharga.

Latihan. Pilih satu skrip dari Bab 14 sampai 27 yang paling Anda sukai. Tandai dengan pensil: mana bagian pola 28.1 sampai 28.8, dan mana bagian yang khas skrip itu. Lalu jalankan dengan data Anda sendiri, dengan hanya mengganti `DATA`, nama berkas, dan nama kolom.

Tabel perbandingan: delapan pola sekilas

Pola	Pekerjaan	Alat utama	Contoh di bab
28.1	Muat, ubah, simpan	`QgsVectorLayer`, `setSubsetString`, `QgsVectorFileWriter`	14, 15, 16
28.2	Loop fitur	`getFeatures`, kamus, `.get`	15, 17, 18
28.3	Panggil alat	`processing.run`	18, 19, 21
28.4	Batch	`glob`, `for`	21, 26
28.5	Tahan galat	`try/except`, `raise`	22
28.6	Ekspresi	`QgsExpression`	22, 23
28.7	Baca dan tulis berkas	`csv`, `json`	14, 22, 27
28.8	Laporan	f-string, `join`	17, 23, 27

BAGIAN C | ARCPY (WINDOWS)

BAB 29: Kenalan dengan ArcPy: Dua Generasi, Satu Sistem Windows

> **Berdasarkan Seri I4 dan M1:** Bab ini pengantar untuk dua bab sesudahnya. Skrip ArcPy di Bab 30 berasal dari Buku M1, *Masterclass QGIS Modern* (Master Class). Skrip di Bab 31 berasal dari Buku I4, *Otomasi Dasar dengan Python* (Intermediate Class). Bab ini sendiri tidak punya skrip utuh. Isinya bersumber dari dokumentasi resmi Esri dan dari pasangan skrip kedua buku itu.

> **Bagian C hanya untuk Windows:** ArcGIS Pro dan ArcMap 10.8 hanya berjalan di Windows. Di macOS dan Ubuntu, Anda tetap bisa membaca skrip bagian ini, tetapi tidak bisa menjalankannya. Penulis belum pernah menjalankan satu pun skrip ArcPy di buku ini, jadi semuanya bertanda **[CEK]**: cocokkan dengan versi ArcGIS di komputer Anda.

Studi kasus: "Skrip itu jalan di kantor pusat, mati di kantor cabang"

Kepala Seksi menyerahkan sebuah skrip dari kantor pusat. Ia berpesan: "Tolong dicoba di kantor cabang." Anda membukanya di komputer cabang, lalu muncul pesan galat pada baris pertama. Ternyata kantor pusat memakai ArcGIS Pro, sedangkan komputer cabang masih memakai ArcMap. Keduanya sama-sama memakai ArcPy. Tetapi tulisan di dalamnya tidak sama. Bab ini membekali Anda supaya tidak kaget: apa itu ArcPy, mana Pro dan mana ArcMap, di mana mengetiknya, dan apa saja yang berbeda.

Konsep: ArcPy dalam tiga kalimat

ArcPy adalah paket Python bawaan ArcGIS. Bayangkan ArcGIS sebagai panel dengan ratusan tombol: penyangga, irisan, hitung luas, dan seterusnya. Dengan ArcPy, Anda menekan tombol itu cukup dengan menulis namanya di sebuah skrip. Dokumentasi Esri menyebut ArcPy sebagai paket Python untuk analisis data geografis, konversi data, pengelolaan data, dan otomasi peta.

Ilustrasi 29.1: ArcPy melayani dua aplikasi



Lima istilah baru yang perlu Anda kenal sekarang:

- **ArcGIS Pro**: aplikasi ArcGIS generasi baru. Ia memakai Python 3.
- **ArcMap 10.8**: aplikasi ArcGIS generasi lama. Ia memakai Python 2.7.
- `import arcpy`: perintah untuk memanggil ArcPy ke dalam skrip. Perintah `import` dibahas di Bab 8.
- **Jendela Python**: kotak di dalam aplikasi ArcGIS untuk mengetik dan menjalankan perintah Python.
- **Proyek** (di Pro, berkas `.aprx`) dan **dokumen peta** (di ArcMap, berkas `.mxd`): berkas yang menyimpan peta Anda.

Dua generasi: Pro lawan ArcMap

Pro dan ArcMap itu seperti dua generasi telepon. Fungsinya mirip, tetapi tombolnya tidak persis sama. Perbedaan terbesarnya ada di versi Python. ArcGIS Pro memakai Python 3, menurut dokumentasi Esri tentang Python di ArcGIS Pro. ArcMap 10.8 memerlukan Python 2.7.18 dan NumPy 1.16.6. Bila belum ada, pemasangan lengkap ArcMap menambahkannya sendiri, menurut halaman persyaratan sistem ArcGIS Desktop 10.8.x.

Hal	ArcGIS Pro	ArcMap 10.8
Versi Python	Python 3	Python 2.7
Berkas peta	proyek `.aprx`	dokumen peta `.mxd`
Modul untuk peta	`arcpy.mp`	`arcpy.mapping`
Sistem operasi	hanya Windows	hanya Windows
Nama alat di skrip l4	`arcpy.analysis.Buffer(...)`	`arcpy.Buffer_analysis(...)`
Ruang sementara di memori	`memory\nama`	`in_memory\nama`

Dokumentasi Esri menjelaskan asal perbedaan modul peta. Perubahan di ArcGIS Pro cukup besar sehingga nama modulnya diganti, dari `arcpy.mapping` menjadi `arcpy.mp`. Daftar peta, layer, dan tata letak kini menjadi **metode** (perintah yang menempel pada sebuah objek) milik objek proyek. Di ArcMap, daftar itu berupa fungsi tersendiri yang harus diberi dokumen peta sebagai masukan pertama.

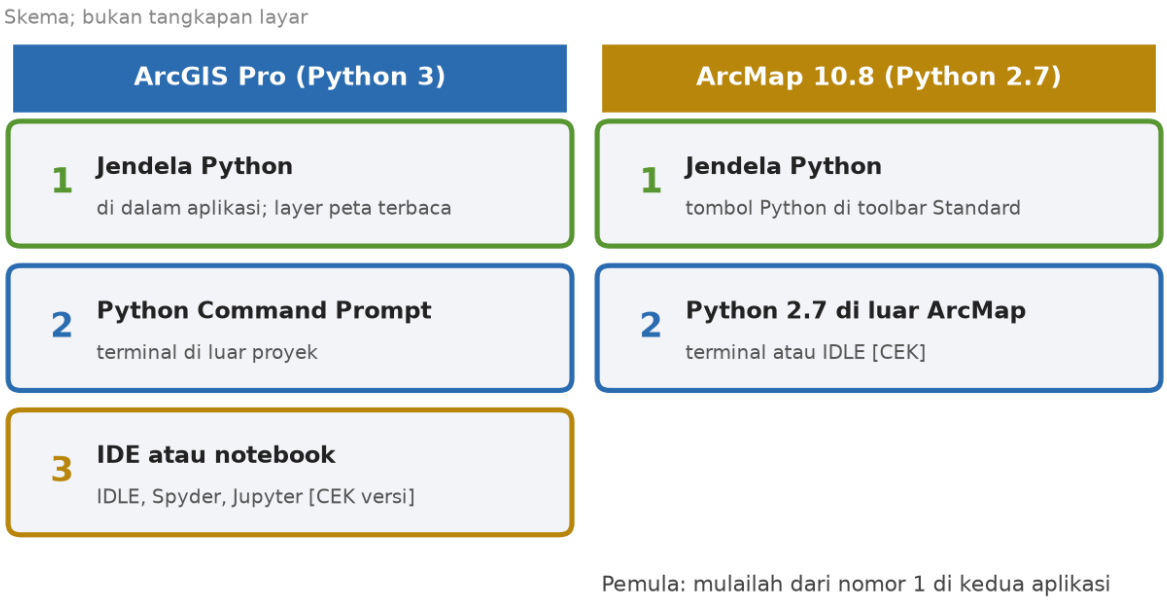
Hanya Windows: apa artinya bagi Anda

Halaman persyaratan sistem ArcGIS Pro menyebut hanya Windows yang didukung: Windows 11, serta Windows Server 2025, 2022, dan 2019. Halaman persyaratan ArcGIS Desktop 10.8.x menyebut Windows 10, Windows 11 (hanya untuk 10.8.1 dan 10.8.2), serta Windows Server 2022, 2019, dan 2016. Tidak ada macOS atau Ubuntu di kedua daftar itu.

Akibat praktisnya: skrip di Bagian C tidak bisa Anda jalankan di Mac atau Ubuntu. Anda tetap bisa membacanya, menerjemahkannya ke PyQGIS (lihat Bab 32), dan menyalin pola pikirnya. Daftar sistem operasi di atas dicatat per tanggal pembacaan dokumen. Versi ArcGIS yang lebih baru bisa mengubahnya, jadi cek ulang di situs Esri. [CEK]

Di mana mengetik dan menjalankan skrip

Ilustrasi 29.2: Tempat mengetik skrip ArcPy



Di ArcGIS Pro, ada tiga tempat. Mulailah dari yang pertama.

1. **Jendela Python.** Buka tab **Analysis**, grup **Geoprocessing**, klik panah di tombol **Python**, lalu pilih **Python window**. Jendela ini menjalankan satu baris atau banyak baris tanpa berkas skrip, dan bisa bekerja dengan layer di peta yang terbuka. Tombol **Load Code** memasukkan berkas `.py` ke jendela, dan transkrip jendela bisa disimpan ke berkas.
2. **Python Command Prompt.** Ini terminal yang sudah diarahkan ke lingkungan Python milik ArcGIS Pro. Ketik `python` untuk sesi interaktif.
3. **IDE atau notebook.** Dokumentasi Esri menyebut IDLE, Spyder, dan Jupyter bisa dipakai bila diarahkan ke Python milik Pro.

Untuk menjalankan berkas skrip dari terminal, dokumentasi Esri memberi contoh perintah ini. Alamat pemasangan Anda bisa berbeda: untuk pemasangan per pengguna, ganti `%PROGRAMFILES%` dengan `%LOCALAPPDATA%`.

TERMINAL
```
"C:\Program Files\ArcGIS\Pro\bin\Python\Scripts\propy" skrip.py
```

Di ArcMap 10.8, tempat termudah adalah jendela Python. Klik tombol **Python** di toolbar **Standard**. Jendela itu punya dua bagian: tempat mengetik di kiri dan bantuan di kanan. Tanda `>>>` berarti siap menerima perintah baru, dan tanda `...` berarti perintah masih berlanjut. Menjalankan skrip dari luar ArcMap memakai Python 2.7 yang ikut terpasang. Cara persisnya (IDLE atau terminal) belum penulis cocokkan. [CEK]

Percobaan pertama: cek versi

Coba satu baris ini di jendela Python, di Pro maupun di ArcMap. Baris `print` dengan satu isian berjalan di kedua versi. [CEK]

```python
import arcpy
print(arcpy.GetInstallInfo()["Version"])
```

Layar menampilkan nomor versi ArcGIS Anda, misalnya `3.x` di Pro atau `10.8.x` di ArcMap. Penulis belum menjalankannya, dan fungsi `GetInstallInfo` belum penulis cocokkan dengan dokumentasi. [CEK]

> **Skrip yang memakai CURRENT:** Banyak skrip di Bab 30 dan 31 memakai kata `"CURRENT"`. Artinya "proyek atau dokumen peta yang sedang terbuka". Dokumentasi Esri menyebut kata ini berlaku bila skrip dijalankan di dalam aplikasi, misalnya di jendela Python. Dari terminal biasa tidak ada proyek yang terbuka, jadi bagian itu gagal.

Beda penulisan yang paling sering menjebak

Ilustrasi 29.3: Empat beda penulisan Pro dan ArcMap

Hal yang berbeda	ArcGIS Pro	ArcMap 10.8
print	`print(a, b)`	`print("%s %s" % (a, b))`
bagi bilangan bulat	`7 / 2` hasilnya `3.5`	`7 / 2` hasilnya `3`; pakai `7 / 2.0`
huruf Indonesia di berkas	UTF-8 bawaan	tambah baris coding di atas
modul peta	`arcpy.mp`	`arcpy.mapping`

Skema; bukan tangkapan layar

Anda akan menemui empat beda ini hampir di setiap skrip. Penjelasan bahasa Python-nya diambil dari panduan resmi Python tentang peralihan Python 2 ke 3.

1. Perintah `print`. Di Python 3, `print` adalah fungsi: `print("A:", x)`. Di Python 2.7, `print` adalah pernyataan, dan tata bahasanya menerima daftar isian tanpa tanda kurung. Akibatnya, `print("A:", x)` di Python 2.7 kemungkinan besar mencetak sepasang isian lengkap dengan tanda kurung dan tanda kutipnya, karena kurung itu dibaca sebagai satu pasangan nilai. [CEK] Karena itu skrip ArcMap di buku ini mencetak satu teks hasil rakitan: `print("A: %s" % x)`. Pilihan lain adalah menambahkan `from __future__ import print_function` di awal berkas, seperti pada skrip bab M1.

2. Pembagian bilangan bulat. Di Python 3, `7 / 2` menghasilkan `3.5`. Di Python 2.7, `7 / 2` menghasilkan `3`, sebab pembagian dua bilangan bulat menghasilkan bilangan bulat. Obatnya: tulis salah satu angka sebagai desimal, `7 / 2.0`, seperti `luas / 10000.0` di skrip ArcMap. Hasil di Python 3.12 sudah diuji, sedangkan hasil di Python 2.7 dibaca dari panduan resmi.

```
HASIL
3.5 3 3.5
```

Baris di atas adalah hasil `print(7 / 2, 7 // 2, 7 / 2.0)` di Python 3.12. Tanda `//` selalu membulatkan ke bawah.

3. Merakit teks. Skrip Pro memakai f-string, misalnya `f"{i}/{total} selesai"`. F-string baru ada sejak Python 3.6, jadi tidak ada di Python 2.7. Skrip ArcMap memakai `"{}/{} selesai".format(i, total)` atau tanda persen: `"%s: luas %.2f ha" % (nama, luas)`. Format `%` berjalan di kedua versi dan dibahas di Bab 9.

4. Huruf Indonesia dan baris `coding`. Panduan Python menyebut berkas sumber Python 2 memakai penyandian ASCII bila tidak ada keterangan lain. Karena itu skrip ArcMap dari Buku I4 membuka berkas dengan `# -*- coding: utf-8 -*-`. Di Python 3, UTF-8 sudah bawaan. Isi skrip itu sebenarnya hanya huruf ASCII, jadi baris itu berjaga-jaga saja. Begitu Anda menulis `é` atau tanda khusus lain di komentar skrip ArcMap, baris itu menjadi wajib.

> **Bedanya di ArcMap:** Setiap skrip ArcPy di buku ini hadir dalam dua versi, dan versi ArcMap selalu diberi kotak seperti ini. Di bab-bab berikutnya, Anda akan terus menemui delapan beda: **pertama**, `print` dengan satu teks hasil rakitan `%` atau `from __future__ import print_function`; **kedua**, pembagian dengan `10000.0`; **ketiga**, `.format` atau `%` sebagai ganti f-string; **keempat**, baris `# -*- coding: utf-8 -*-`; **kelima**, `arcpy.mapping` sebagai ganti `arcpy.mp`; **keenam**, nama alat gaya `Buffer_analysis` sebagai ganti `analysis.Buffer`; **ketujuh**, ruang sementara `in_memory` sebagai ganti `memory`; **kedelapan**, hal kecil Python 2.7 seperti `enumerate(daftar, 1)` dan `open(..., "wb")` untuk CSV.

Jalur Windows dan huruf r

Jalur folder Windows memakai garis miring terbalik: `D:\Latihan\paket-i4`. Masalahnya, Python memakai tanda yang sama sebagai tanda khusus. Di dalam teks biasa, `\n` berarti baris baru dan `\t` berarti tab. Sebagian urutan lain bahkan membuat skrip gagal dimuat. Obatnya: tulis huruf `r` tepat sebelum kutip pembuka. Huruf `r` berarti *raw* atau mentah: garis miring dibaca apa adanya.

Lihat bedanya pada percobaan di Python 3.12. Pada baris pertama, `\t` berubah menjadi tab, sehingga `tim_a` terpotong. Python 3.12 juga memberi peringatan `SyntaxWarning` untuk `\L`.

```
PYTHON
print("D:\Laporan\tim_a")
print(r"D:\Laporan\tim_a")
```

HASIL

```
D:\Laporan    im_a
D:\Laporan\tim_a
```

Kasus lain yang lebih keras: `print("C:\Users\anda")` tanpa huruf `r` langsung gagal di Python 3.12 dengan pesan `SyntaxError: (unicode error) 'unicodeescape' codec can't decode bytes in position 2-3: truncated \UXXXXXXXX escape`. Tanda `\U` dianggap awal penulisan huruf khusus. Semua jalur di skrip Bagian C ditulis dengan `r"..."`. Biasakan hal yang sama. Satu catatan: teks mentah tidak boleh berakhir dengan satu garis miring terbalik sebelum kutip penutup. Baris `x = r"D:\"` gagal di Python 3.12 dengan `SyntaxError: unterminated string literal`. Bila perlu garis miring di ujung, gabungkan teks biasa: `DATA + "\\"`.

Kebiasaan kerja yang aman

- **Salin dulu datanya.** Banyak skrip menyimpan, menimpa, atau menulis keluaran. Pelajari bagian "Jebakan" tiap skrip sebelum menjalankannya.
- **Cocokkan dengan versi Anda.** Nama alat dan parameter bisa berubah antarversi. Setiap skrip di buku ini membawa komentar `[CEK]`.
- **Mulailah dari baris kecil.** Di jendela Python, coba satu baris dulu. Bila hasilnya sesuai harapan, baru jalankan skrip utuh.
- Hati-hati dengan `save()`. Perintah `aprx.save()` di Pro menyimpan perubahan ke proyek yang sedang dibuka. Dokumentasi Esri juga menyediakan `saveACopy(nama_berkas)` untuk menyimpan salinan. [CEK]

Cek paham

1. Apa dua generasi ArcGIS yang memakai ArcPy, dan versi Python masing-masing?
2. Mengapa skrip ArcPy di buku ini tidak bisa Anda jalankan di Ubuntu?
3. Mengapa jalur Windows di skrip ditulis dengan huruf `r` di depan kutip?

Jawaban.

1. ArcGIS Pro memakai Python 3. ArcMap 10.8 memakai Python 2.7.
2. Karena ArcGIS Pro dan ArcMap hanya didukung di Windows menurut halaman persyaratan sistem Esri.
3. Huruf `r` membuat garis miring terbalik dibaca apa adanya. Tanpa `r`, `\n` dan `\t` menjadi tanda khusus, dan `\U` bisa membuat skrip gagal.

Kesalahan umum

- **Menyalin gaya Python 3 ke ArcMap.** Contohnya `print("A:", x)` atau f-string. Perbaikan: ikuti versi ArcMap di Bab 30 dan 31.
- Lupa huruf `r` pada jalur. Perbaikan: tulis `r"D:\Latihan\paket-i4"`.
- Menjalankan skrip ber-`CURRENT` dari terminal. Perbaikan: jalankan dari jendela Python di dalam aplikasi.

- Menyamakan `arcpy.mp` dengan `arcpy.mapping`. Keduanya mirip tujuannya, tetapi namanya dan cara kerjanya berbeda. Perbaikan: pakai `arcpy.mp` di Pro dan `arcpy.mapping` di ArcMap.

Ringkasan dan latihan

- **ArcPy** adalah paket Python bawaan ArcGIS untuk analisis, pengelolaan data, dan otomasi peta.
- ArcGIS Pro memakai Python 3 dan `arcpy.mp`. ArcMap 10.8 memakai Python 2.7 dan `arcpy.mapping`.
- Keduanya hanya berjalan di Windows. Tempat termudah menjalankan skrip adalah jendela Python di dalam aplikasi.
- Empat beda yang paling sering menjebak: `print`, pembagian bilangan bulat, baris `coding`, dan modul peta.
- Tulis jalur Windows dengan huruf `r` di depan kutip.

Latihan. Tulis dua baris `print` yang menampilkan nama KPH dan luasnya di Python 3, lalu tulis ulang untuk Python 2.7 dengan tanda persen. Contoh jawaban: `print("KPH Alpha", 4.0)` menjadi `print("%s %.1f" % ("KPH Alpha", 4.0))`. Hasil tulisan kedua penulis cek di Python 3 saja: `KPH Alpha 4.0`. Di Python 2.7 belum diuji. [CEK]

Tabel perbandingan: Pro dan ArcMap sekilas

Pertanyaan	ArcGIS Pro	ArcMap 10.8
Python versi berapa	3	2.7
Bagaimana membuka peta yang sedang terbuka	`arcpy.mp.ArcGISProject("CURRENT")`	`arcpy.mapping.MapDocument("CURRENT")`
Di mana mengetik cepat	tab Analysis, tombol Python, Python window	tombol Python di toolbar Standard
Cara aman mencetak	`print("A:", x)`	`print("A: %s" % x)`
Cara aman membagi	`luas / 10000`	`luas / 10000.0`

BAB 30: Empat Pekerjaan Klasik di ArcGIS: Skrip Masterclass dalam Bahasa ArcPy

> **Berdasarkan Seri M1:** Skrip bab ini berasal dari Buku M1, *Masterclass QGIS Modern* (Master Class). Berkasnya `bab1_skrip_1_1_pro.py`, `bab2_skrip_2_1_pro.py`, `bab3_skrip_3_1_pro.py`, dan `bab6_skrip_6_1_pro.py` di folder `skrip/arcgis_pro`, serta pasangannya berakhiran `_arcmap.py` di folder `skrip/arcgis_arcmap`. Di dalam berkas, nomor skrip tertulis 1.1, 2.1, 3.1, dan 6.1 mengikuti bab buku M1. Di buku itu, tiap skrip ArcPy dipasangkan dengan skrip PyQGIS yang sepadan. Padanan PyQGIS-nya ada di Bab 24.

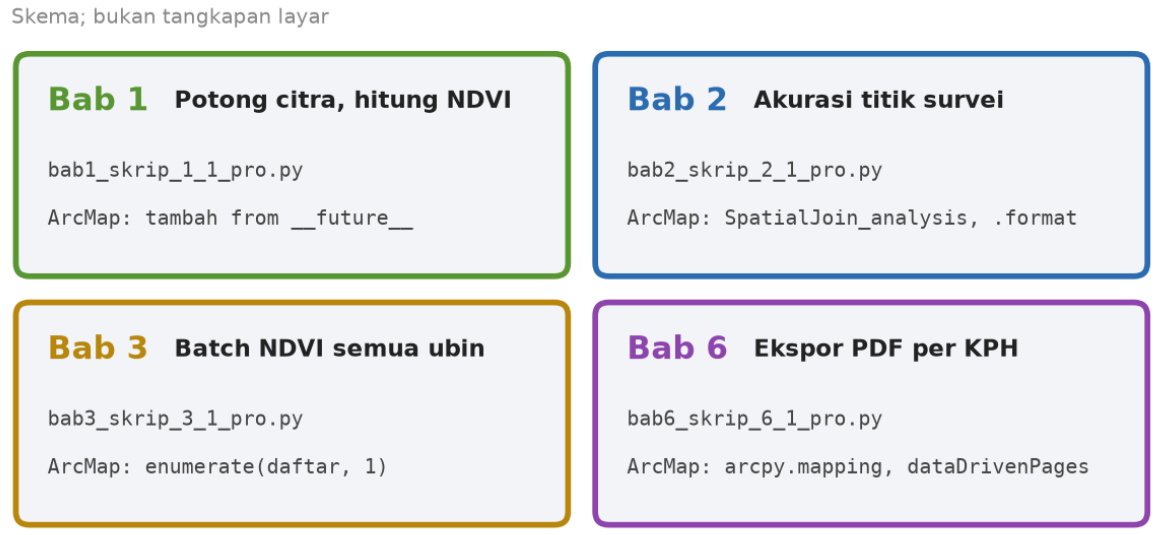
Studi kasus: "Empat tugas rutin, satu kantor cabang"

Kantor cabang memakai ArcGIS, dan Kepala Seksi punya daftar tugas rutin. Pertama, mengubah dua band citra satelit menjadi peta kehijauan (NDVI). Kedua, menghitung seberapa tepat peta klasifikasi dibanding titik survei lapangan. Ketiga, menghitung NDVI untuk banyak foto drone sekaligus. Keempat, mencetak satu PDF laporan kebakaran untuk setiap KPH. Anda sudah mengenal empat pekerjaan ini di versi QGIS. Sekarang Anda menuliskannya dalam bahasa ArcPy, di ArcGIS Pro dan di ArcMap.

Konsep: pekerjaan sama, bahasa beda

Bayangkan satu resep nasi goreng yang ditulis dalam dua buku masak. Bahannya sama dan urutannya sama. Hanya nama alat dapurnya yang berbeda. Begitu pula skrip di bab ini. Setiap pekerjaan hadir dalam dua versi: **ArcGIS Pro** (Python 3) lebih dulu, lalu **ArcMap 10.8** (Python 2.7).

Ilustrasi 30.1: Empat pekerjaan dan beda utama ArcMap



Lima istilah baru yang perlu Anda kenal sekarang:

- **Ekstensi** (*extension*): izin tambahan di ArcGIS untuk kelompok alat tertentu. Alat raster di bab ini memerlukan ekstensi **Spatial Analyst**. Anda "meminjam" izin itu di awal skrip dan "mengembalikannya" di akhir.
- **Objek raster**: nilai di Python yang mewakili satu raster. Objek ini bisa dijumlah, dikurangi, dan dibagi seperti angka biasa. Cara kerja hitungan ini dinamakan *map algebra*.
- `arcpy.env.workspace`: folder kerja bawaan. Nama data tanpa jalur dicari di folder ini.
- **Geodatabase** (berkas `.gdb`): satu wadah untuk banyak layer. Anda memakainya di Skrip 30.2.
- **Map Series** (di Pro) dan **Data Driven Pages** (di ArcMap): fitur yang membuat satu halaman peta untuk setiap fitur di sebuah layer indeks. Anda memakainya di Skrip 30.4.

Semua keluaran di bab ini **perkiraan**. Penulis belum pernah menjalankan skrip ArcPy ini di ArcGIS. Angka di bagian "Hasil" dihitung dengan alat lain (GDAL dan Python) dari data sintetis di paket data, jadi tulisan persisnya di komputer Anda bisa sedikit berbeda. Semuanya bertanda **[CEK]**.

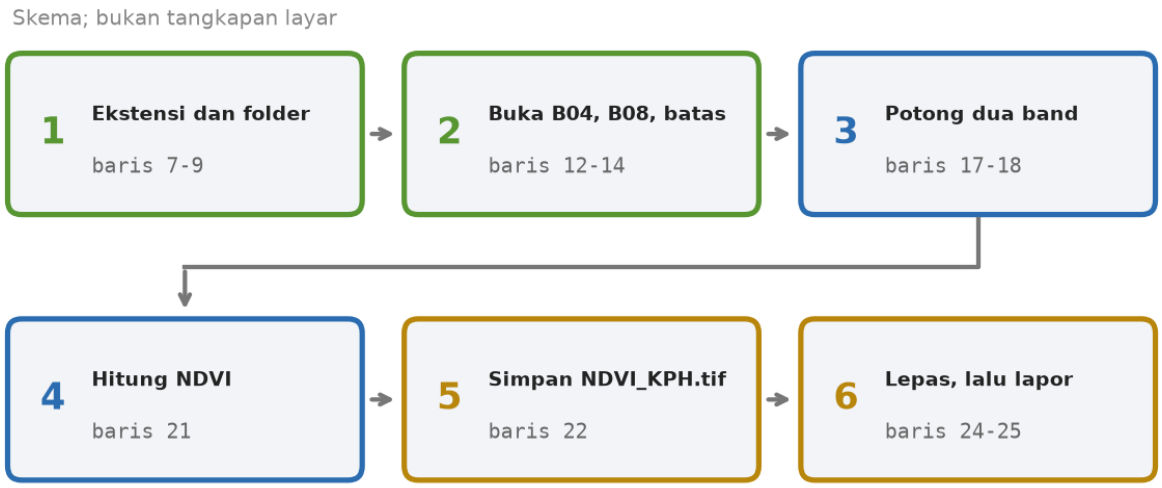
Skrip 30.1: Memotong citra dan menghitung NDVI

Berkas `bab1_skrip_1_1_pro.py` dan `bab1_skrip_1_1_arcmap.py`. Di dalam berkasnya tertulis "Skrip 1.1".

Tujuan. Memotong dua band citra dengan batas KPH, lalu menghitung NDVI dari kedua band itu dan menyimpannya.

Bahan. Tiga data di folder `C:\data_kph`: `B04.tif` (band merah), `B08.tif` (band inframerah dekat, NIR), dan `Batas_KPH.shp`. Di paket data buku ini, `B04.tif` dan `B08.tif` ada di `paket-data\Sentinel_Lokal`, sedangkan `Batas_KPH.shp` ada di `paket-i4\shp` (bersama berkas pendampingnya `.dbf`, `.shx`, dan `.prj`). Salin ketiganya ke satu folder. Ekstensi Spatial Analyst harus tersedia. Jalankan dari jendela Python, atau dari terminal Python milik ArcGIS (lihat Bab 29).

Ilustrasi 30.2: Alur Skrip 30.1



Versi ArcGIS Pro (Python 3)

```python
# [SKRIP 1.1-Pro: Memotong citra dan menghitung NDVI dengan ArcPy]
# Penulis: Badar Mubarok Yogaswara
# [CEK] Cocokkan dengan versi Anda. Butuh ekstensi Spatial Analyst atau Image Analyst.
import arcpy
from arcpy.sa import Raster, Float, ExtractByMask

arcpy.CheckOutExtension("Spatial")
arcpy.env.overwriteOutput = True
arcpy.env.workspace = r"C:\data_kph"      # ganti dengan folder Anda

# 1. Masukan: band Red, band NIR, dan batas area (CRS harus sama, lihat 1.2)
red = Raster("B04.tif")
nir = Raster("B08.tif")
batas = "Batas_KPH.shp"

# 2. Potong kedua band sesuai batas
red_clip = ExtractByMask(red, batas)
nir_clip = ExtractByMask(nir, batas)

# 3. NDVI = (NIR - RED) / (NIR + RED)
ndvi = (Float(nir_clip) - Float(red_clip)) / (Float(nir_clip) + Float(red_clip))
ndvi.save("NDVI_KPH.tif")

arcpy.CheckInExtension("Spatial")
print("Selesai: NDVI_KPH.tif tersimpan.")
```

Penjelasan baris demi baris. Nomor baris mengikuti berkas di atas.

Baris	Tulisan	Artinya
1-3	`# [SKRIP 1.1-Pro ...`	Komentar: judul, penulis (tertulis `Badar Mubarok Yogaswara`), dan pengingat **[CEK]** bahwa skrip perlu ekstensi.
4	`import arcpy`	Panggil perkakas ArcPy.
5	`from arcpy.sa import Raster, Float, ExtractByMask`	Panggil tiga perkakas dari bagian Spatial Analyst (`sa`): `Raster` untuk membuka raster, `Float` untuk mengubah ke angka desimal, `ExtractByMask` untuk memotong.
7	`arcpy.CheckOutExtension("Spatial")`	"Pinjam" izin Spatial Analyst.
8	`arcpy.env.overwriteOutput = True`	Izinkan keluaran menimpa berkas lama dengan nama sama, tanpa bertanya.
9	`arcpy.env.workspace = r"C:\data_kph"`	Tetapkan folder kerja. Huruf `r` di depan kutip menjaga garis miring terbalik (Bab 29).
12-13	`red = Raster("B04.tif"), nir = Raster("B08.tif")`	Buka dua band dan simpan di kotak `red` dan `nir`. Nama tanpa jalur dicari di folder kerja.
14	`batas = "Batas_KPH.shp"`	Simpan nama berkas batas di kotak `batas`. Ini hanya teks.
17-18	`red_clip = ExtractByMask(red, batas)`	Potong tiap band mengikuti bentuk batas KPH. Hasilnya objek raster.
21	`ndvi = (Float(nir_clip) - Float(red_clip)) / (...)`	Rumus NDVI: (NIR kurang merah) dibagi (NIR tambah merah). `Float` mengubah angka menjadi desimal lebih dulu.

Baris	Tulisan	Artinya
22	`ndvi.save("NDVI_KPH.tif")`	Simpan hasil ke berkas `NDVI_KPH.tif`.
24	`arcpy.CheckInExtension("Spatial")`	"Kembalikan" izin Spatial Analyst.
25	`print("Selesai: ...")`	Tampilkan pesan selesai.

Dokumentasi Esri menyebut `ExtractByMask` mengembalikan objek raster, dan objek itu disimpan dengan `.save()`. Karena itu baris 22 diperlukan. [CEK]

Versi ArcMap 10.8 (Python 2.7)

PYTHON
```
# [SKRIP 1.1-ArcMap: Memotong citra dan menghitung NDVI dengan ArcPy]
# Penulis: Badar Mubarok Yogaswara
# [CEK] Cocokkan dengan versi Anda. Butuh ekstensi Spatial Analyst.
from __future__ import print_function   # agar print() berlaku di Python 2.7
import arcpy
from arcpy.sa import Raster, Float, ExtractByMask

arcpy.CheckOutExtension("Spatial")
arcpy.env.overwriteOutput = True
arcpy.env.workspace = r"C:\data_kph"      # ganti dengan folder Anda

red = Raster("B04.tif")
nir = Raster("B08.tif")
batas = "Batas_KPH.shp"

red_clip = ExtractByMask(red, batas)
nir_clip = ExtractByMask(nir, batas)

ndvi = (Float(nir_clip) - Float(red_clip)) / (Float(nir_clip) + Float(red_clip))
ndvi.save("NDVI_KPH.tif")

arcpy.CheckInExtension("Spatial")
print("Selesai: NDVI_KPH.tif tersimpan.")
```

Bedanya di ArcMap: Hampir tidak ada beda isi, hanya satu baris tambahan. Baris 4, `from __future__ import print_function`, membuat `print(...)` dengan tanda kurung berlaku sebagai fungsi di Python 2.7, seperti di Python 3. Karena baris itu, semua nomor baris sesudahnya bergeser satu: `ndvi.save` kini di baris 20 dan `print` di baris 23. Komentar judul memuat "ArcMap", dan komentar pengingat hanya menyebut Spatial Analyst (versi Pro menyebut juga Image Analyst). Pada skrip ini tidak ada pembagian bilangan bulat yang berbahaya, sebab `Float` sudah mengubah kedua raster menjadi desimal. Dokumentasi alat Divide menjelaskan: bila kedua masukan bilangan bulat, hasilnya bilangan bulat (3 dibagi 2 menghasilkan 1). [CEK]

Hasil. Layar menampilkan satu baris pesan. Berkas `NDVI_KPH.tif` muncul di `C:\data_kph`. Berikut perkiraannya, dihitung dengan GDAL dari data contoh. [CEK]

HASIL
```
Selesai: NDVI_KPH.tif tersimpan.
```

Batas KPH di data contoh menutupi seluruh citra (400 m kali 400 m, 200 kali 200 piksel pada 2 m). Rata-rata NDVI seluruh piksel kira-kira **0,505**. Kisaran nilai NDVI selalu antara -1 dan 1, dan semakin tinggi berarti semakin hijau.

Bagian yang boleh diubah. Nomor baris mengikuti versi Pro.

- **Baris 9, folder kerja.** Ganti `C:\data_kph` dengan folder data Anda. Pertahankan huruf `r`.
- **Baris 12-14, nama berkas.** Ganti `B04.tif`, `B08.tif`, dan `Batas_KPH.shp` dengan data Anda. Pastikan band merah dan NIR tidak tertukar.
- **Baris 21, rumus.** Rumus ini boleh diganti rumus indeks lain yang memakai dua band.
- **Baris 22, nama keluaran.** Ganti `NDVI_KPH.tif` untuk nama lain.

Jebakan.

- **Sistem koordinat harus sama.** Komentar di baris 11 mengingatkan hal ini. Bila citra dan batas berbeda sistem koordinat, hasil potongan bisa kosong atau salah tempat. Cek dulu sifat kedua data. [CEK]
- **Band tertukar.** Bila `B04` dan `B08` tertukar, NDVI berubah tanda tanpa pemberitahuan. Pastikan `B04` merah dan `B08` NIR.
- **Ekstensi tidak dikembalikan saat galat.** Baris 24 hanya tercapai bila baris-baris sebelumnya berhasil. Bila skrip berhenti di tengah, izin Spatial Analyst tetap dipinjam sampai Anda menutup sesi. Skrip I4 di Bab 31 memperbaikinya dengan `try` dan `finally`.
- **Nama keluaran tanpa jalur.** `ndvi.save("NDVI_KPH.tif")` mengandalkan folder kerja. Periksa bahwa berkas muncul di `C:\data_kph`. [CEK]
- **Keluaran menimpa tanpa tanya.** Karena `overwriteOutput = True`, berkas lama bernama sama hilang.

Latihan variasi. Tampilkan ukuran raster merah sebelum NDVI dihitung. Contoh jawaban, satu baris tambahan sebelum baris 21:

```python
print("Ukuran:", red.width, "x", red.height, "piksel")
```

Dari data contoh, layar menampilkan `Ukuran: 200 x 200 piksel`, lalu pesan selesai. [CEK]

> **Pola yang bisa dipakai ulang:** Kerangka "hitung indeks dari dua band": **pinjam ekstensi**, **buka dua band**, **potong dengan batas**, hitung rumus dengan `Float`, **simpan**, **kembalikan ekstensi**. Untuk data baru, ganti tiga hal: folder kerja (baris 9), nama berkas masukan (baris 12-14), dan nama keluaran (baris 22).

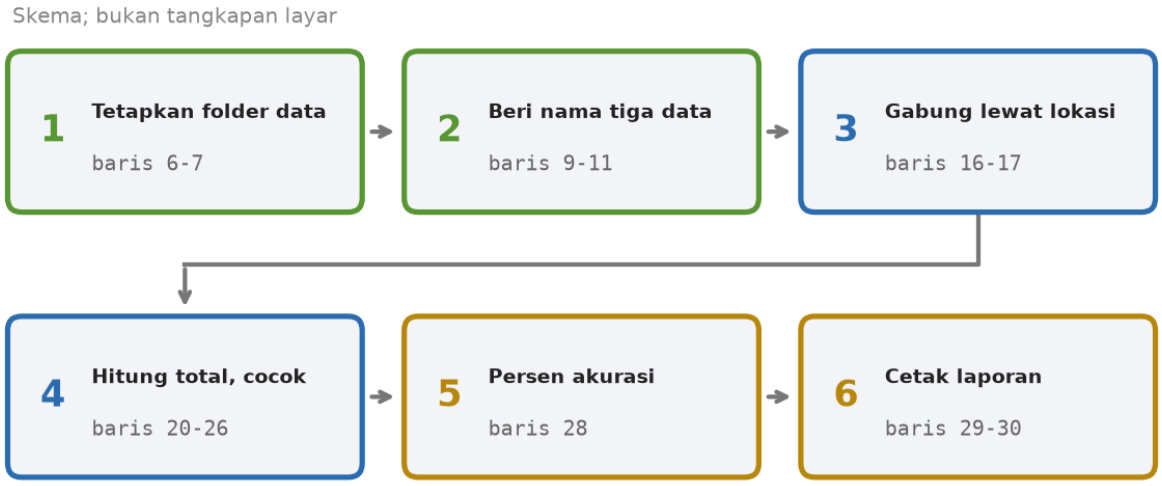
Skrip 30.2: Akurasi titik survei lawan peta klasifikasi

Berkas `bab2_skrip_2_1_pro.py` dan `bab2_skrip_2_1_arcmap.py`. Di dalam berkasnya tertulis "Skrip 2.1".

Tujuan. Menarik kelas peta klasifikasi ke setiap titik survei, lalu menghitung berapa persen titik yang kondisinya cocok dengan kelas peta.

Bahan. Geodatabase `C:\data_kph\Survei.gdb` berisi dua layer: `Titik_Survei` (10 titik, kolom `Kondisi`) dan `Poligon_Sawah` (kolom `Kelas_Citra`). Di paket data buku ini, keduanya berbentuk GeoPackage di `paket-data`: `Survei.gpkg` (layer `Titik_Survei`) dan `Poligon_Sawah.gpkg`. Anda perlu memasukkan keduanya ke geodatabase buatan Anda dengan alat impor bawaan ArcGIS. Skrip tidak melakukan itu. [CEK] Skrip ini sepadan dengan skrip PyQGIS di Bab 24.

Ilustrasi 30.3: Alur Skrip 30.2



Versi ArcGIS Pro (Python 3)

PYTHON

```
# [SKRIP 2.1-Pro: Validasi Titik Ground-Truth vs Poligon Klasifikasi dengan ArcPy]
# Penulis: Badar Mubarok Yogaswara
# [CEK] Cocokkan dengan versi Anda. Spatial Join tersedia pada semua level lisensi.
import arcpy

arcpy.env.workspace = r"C:\data_kph\Survei.gdb"      # ganti dengan lokasi Anda
arcpy.env.overwriteOutput = True

survei = "Titik_Survei"
klasifikasi = "Poligon_Sawah"
hasil = "Validasi_Hasil"

print("Memulai validasi lapangan otomatis...")

# 1. Spatial Join: tarik atribut poligon ke titik survei yang berada di dalamnya
arcpy.analysis.SpatialJoin(survei, klasifikasi, hasil,
                           "JOIN_ONE_TO_ONE", "KEEP_ALL", "", "INTERSECT")

# 2. Hitung akurasi: bandingkan Kondisi dengan Kelas_Citra
total_titik = 0
titik_cocok = 0
with arcpy.da.SearchCursor(hasil, ["Kondisi", "Kelas_Citra"]) as kursor:
    for kondisi, kelas in kursor:
        total_titik += 1
        if str(kondisi).lower() == str(kelas).lower():
            titik_cocok += 1

akurasi = (titik_cocok / total_titik) * 100 if total_titik > 0 else 0
print(f"Laporan Validasi: {titik_cocok} dari {total_titik} titik cocok.")
print(f"Akurasi Keseluruhan (Overall Accuracy): {akurasi:.2f}%")
```

Penjelasan baris demi baris. Nomor baris mengikuti berkas di atas.

Baris	Tulisan	Artinya
1-3	`# [SKRIP 2.1-Pro ...`	Komentar: judul, penulis, dan catatan bahwa Spatial Join tersedia pada semua level lisensi.
4	`import arcpy`	Panggil ArcPy.
6	`arcpy.env.workspace = r"C:\data_kph\Survei.gdb"`	Folder kerja adalah geodatabase itu sendiri. Nama layer di bawah dicari di dalamnya.
7	`arcpy.env.overwriteOutput = True`	Izinkan keluaran menimpa yang lama.
9-11	`survei = "Titik_Survei"` dan dua baris sesudahnya	Simpan nama dua layer masukan dan nama keluaran di kotak `survei`, `klasifikasi`, dan `hasil`.
13	`print("Memulai ...")`	Tampilkan pesan awal.
16-17	`arcpy.analysis.SpatialJoin(survei, klasifikasi, hasil, ...)`	**Spatial Join**: tarik kolom poligon ke titik yang jatuh di dalamnya. Tiga isian pertama: titik, poligon, nama keluaran.
17	`"JOIN_ONE_TO_ONE", "KEEP_ALL", "", "INTERSECT"`	Empat isian pilihan, berurutan: satu baris keluaran per titik; titik tanpa pasangan tetap dipertahankan; kolom bawaan (teks kosong); dua bentuk yang bersinggungan dianggap cocok.
20-21	`total_titik = 0`, `titik_cocok = 0`	Dua penghitung, mulai dari nol.
22	`with arcpy.da.SearchCursor(hasil, ["Kondisi", "Kelas_Citra"]) as kursor:`	Buka kursor pada dua kolom hasil gabungan. Kursor dibahas di Bab 31.
23	`for kondisi, kelas in kursor:`	Ambil satu baris tiap putaran. Dua isiannya masuk ke `kondisi` dan `kelas`.
24	`total_titik += 1`	Tambah satu ke penghitung total.
25-26	`if str(kondisi).lower() == str(kelas).lower():`	Ubah keduanya menjadi teks huruf kecil, lalu bandingkan. Bila sama, tambah satu ke penghitung cocok.
28	`akurasi = (...) * 100 if total_titik > 0 else 0`	Hitung persen. Bagian `if ... else 0` menjaga agar tidak membagi dengan nol.
29-30	`print(f"...")`	Cetak laporan dengan f-string. `{akurasi:.2f}` berarti dua angka desimal.

Dokumentasi Esri mencatat urutan isian `SpatialJoin`: `target_features`, `join_features`, `out_feature_class`, lalu isian pilihan `join_operation`, `join_type`, `field_mapping`, `match_option`. Urutan itu cocok dengan baris 16-17. Setelah gabungan, alat menambahkan kolom `Join_Count`. [CEK]

Versi ArcMap 10.8 (Python 2.7)

PYTHON
```
# [SKRIP 2.1-ArcMap: Validasi Titik Ground-Truth vs Poligon Klasifikasi dengan ArcPy]
# Penulis: Badar Mubarok Yogaswara
# [CEK] Cocokkan dengan versi Anda. Spatial Join tersedia pada semua level lisensi.
from __future__ import print_function   # agar print() berlaku di Python 2.7
import arcpy

arcpy.env.workspace = r"C:\data_kph\Survei.gdb"      # ganti dengan lokasi Anda
arcpy.env.overwriteOutput = True

survei = "Titik_Survei"
klasifikasi = "Poligon_Sawah"
hasil = "Validasi_Hasil"

print("Memulai validasi lapangan otomatis...")

# 1. Spatial Join: tarik atribut poligon ke titik survei yang berada di dalamnya
arcpy.SpatialJoin_analysis(survei, klasifikasi, hasil,
                           "JOIN_ONE_TO_ONE", "KEEP_ALL", "", "INTERSECT")

# 2. Hitung akurasi: bandingkan Kondisi dengan Kelas_Citra
total_titik = 0
titik_cocok = 0
with arcpy.da.SearchCursor(hasil, ["Kondisi", "Kelas_Citra"]) as kursor:
    for kondisi, kelas in kursor:
        total_titik += 1
        if str(kondisi).lower() == str(kelas).lower():
            titik_cocok += 1

akurasi = (float(titik_cocok) / total_titik) * 100 if total_titik > 0 else 0
print("Laporan Validasi: {} dari {} titik cocok.".format(titik_cocok, total_titik))
print("Akurasi Keseluruhan (Overall Accuracy): {:.2f}%".format(akurasi))
```

Bedanya di ArcMap: Ada empat beda. **Pertama**, baris 4 menambah `from __future__ import print_function`, sehingga nomor baris bergeser satu. **Kedua**, alat ditulis `arcpy.SpatialJoin_analysis(...)` (nama alat, garis bawah, nama kotak alat) sebagai ganti `arcpy.analysis.SpatialJoin(...)`. **Ketiga**, baris 29 menulis `float(titik_cocok) / total_titik`. Di Python 2.7, 9 dibagi 10 menghasilkan 0, sehingga akurasi akan tercetak 0 padahal seharusnya 90. `float(...)` memaksa hasil desimal. **Keempat**, cetakan memakai `.format` dengan `{}` sebagai ganti f-string, sebab f-string baru ada sejak Python 3.6. `{:.2f}` bermakna sama dengan `:.2f` di f-string.

Hasil. Layar menampilkan tiga baris. Dari data contoh, 9 dari 10 titik cocok (satu titik sengaja dibuat tidak cocok). Perkiraan ini dihitung dengan GDAL dan Python dari `Survei.gpkg` dan `Poligon_Sawah.gpkg`. [CEK]

HASIL
```
Memulai validasi lapangan otomatis...
Laporan Validasi: 9 dari 10 titik cocok.
Akurasi Keseluruhan (Overall Accuracy): 90.00%
```

Layer `Validasi_Hasil` juga muncul di dalam `Survei.gdb`. Logika Python skrip ini sudah diuji dengan modul ArcPy tiruan buatan sendiri yang membaca paket data, dan hasilnya cocok dengan perkiraan di atas. Perilaku alat ArcGIS yang sebenarnya belum teruji. [CEK]

Bagian yang boleh diubah. Nomor baris mengikuti versi Pro.

- **Baris 6, geodatabase.** Ganti dengan geodatabase milik Anda.
- **Baris 9-11, nama layer.** Ganti `Titik_Survei`, `Poligon_Sawah`, dan `Validasi_Hasil` sesuai data Anda.
- **Baris 22, nama kolom.** Ganti `Kondisi` dan `Kelas_Citra` dengan dua kolom yang ingin Anda bandingkan.
- **Baris 17, pengaturan gabungan.** Ganti `"INTERSECT"` dengan cara pencocokan lain yang didukung alat. Daftarnya ada di dokumentasi alat. [CEK]

Jebakan.

- **Titik di luar poligon.** Karena memakai `KEEP_ALL`, titik tanpa pasangan tetap ada di hasil, dengan `Kelas_Citra` kosong. `str(None).lower()` menghasilkan teks `none`, yang tidak sama dengan kondisi mana pun. Titik itu dihitung tidak cocok. Hasil `str(None).lower()` sudah diuji di Python 3.
- **Nama kolom kembar.** Bila kedua layer punya kolom bernama sama, nama di hasil gabungan bisa berubah, dan baris 22 gagal menemukan kolomnya. [CEK]
- Pembagian di ArcMap tanpa `float`. Di Python 3, `(9 / 10) * 100` menghasilkan `90.0`. Pembagian bilangan bulat gaya Python 2.7 (diperagakan dengan `(9 // 10) * 100` di Python 3) menghasilkan `0`.
- **Geodatabase belum ada.** Skrip tidak membuat `Survei.gdb` dan tidak mengimpor data. Bila geodatabase kosong, kursor gagal.
- **Hasil gabungan menimpa layer lama.** `overwriteOutput = True` membuat `Validasi_Hasil` yang lama diganti.

Latihan variasi. Tambahkan satu baris laporan: jumlah titik yang tidak cocok. Contoh jawaban, tepat sebelum baris cetak akurasi (baris 30):

```python
print(f"Tidak cocok: {total_titik - titik_cocok} titik.")
```

Dari data contoh, layar menampilkan `Tidak cocok: 1 titik.` Untuk ArcMap, ganti dengan `print("Tidak cocok: {} titik.".format(total_titik - titik_cocok))`. [CEK]

> **Pola yang bisa dipakai ulang:** Kerangka "hitung kecocokan dua kolom": **gabungkan layer lewat lokasi**, **ulangi tiap baris dengan kursor**, **hitung total dan yang cocok**, **bagi dengan penjagaan nol**. Untuk data baru, ganti nama layer (baris 9-11) dan dua nama kolom (baris 22).

Skrip 30.3: NDVI untuk semua ubin foto drone

Berkas `bab3_skrip_3_1_pro.py` dan `bab3_skrip_3_1_arcmap.py`. Di dalam berkasnya tertulis "Skrip 3.1".

Tujuan. Menghitung NDVI untuk setiap ubin foto drone dalam satu folder, lalu menyimpan hasilnya di folder lain, dengan laporan kemajuan.

Bahan. Folder `D:\Data_Drone\Ortofoto_Mentah` berisi ubin `.tif` dua band (band 1 inframerah dekat, band 2 merah). Di paket data, tiga ubin ada di `paket-i4\Ortofoto_Mentah`. Siapkan juga folder kosong `D:\Data_Drone\NDVI_Hasil`: **skrip tidak membuat folder ini**. Ekstensi Spatial Analyst harus tersedia. Padanan PyQGIS-nya ada di Bab 24.

Ilustrasi 30.4: Alur Skrip 30.3



Versi ArcGIS Pro (Python 3)

```python
# [SKRIP 3.1-Pro: Batch Processing Citra Drone dengan ArcPy]
# Penulis: Badar Mubarok Yogaswara
# [CEK] Cocokkan dengan versi Anda. Butuh ekstensi Spatial Analyst atau Image Analyst.
import os
import arcpy
from arcpy.sa import Raster, Float

arcpy.CheckOutExtension("Spatial")
arcpy.env.overwriteOutput = True

folder_input = r"D:\Data_Drone\Ortofoto_Mentah"
folder_output = r"D:\Data_Drone\NDVI_Hasil"
arcpy.env.workspace = folder_input

daftar = arcpy.ListRasters("*", "TIF")
total = len(daftar)
print(f"Memulai batch {total} ubin...")

for i, nama in enumerate(daftar, start=1):
    # Urutan band mengikuti Skrip 3.1 QGIS: pastikan Band_1 = NIR dan Band_2 = Red
    band1 = Float(Raster(os.path.join(folder_input, nama, "Band_1")))
    band2 = Float(Raster(os.path.join(folder_input, nama, "Band_2")))
    ndvi = (band1 - band2) / (band1 + band2)
    ndvi.save(os.path.join(folder_output, "NDVI_" + nama))
    print(f"{i}/{total} selesai: {nama}")

arcpy.CheckInExtension("Spatial")
print("Batch selesai.")
```

Penjelasan baris demi baris. Nomor baris mengikuti berkas di atas. Hanya bagian baru yang dijelaskan. Baris `import`, ekstensi, dan `overwriteOutput` sudah Anda kenal di Skrip 30.1.

Baris	Tulisan	Artinya
4	`import os`	Panggil perkakas `os` untuk merakit jalur folder (Bab 8).
6	`from arcpy.sa import Raster, Float`	Hanya dua perkakas yang diperlukan.
8-9	`arcpy.CheckOutExtension("Spatial")` dan `overwriteOutput`	Pinjam ekstensi dan izinkan menimpa.
11-13	`folder_input`, `folder_output`, `arcpy.env.workspace`	Simpan dua lokasi folder. Folder masukan juga dijadikan folder kerja.
15	`daftar = arcpy.ListRasters("*", "TIF")`	Ambil daftar nama semua raster berformat TIF di folder kerja. Tanda bintang berarti "nama apa saja".
16	`total = len(daftar)`	Hitung banyak ubin.
17	`print(f"Memulai batch {total} ubin...")`	Cetak pesan awal. Nilai `total` ikut tercetak.
19	`for i, nama in enumerate(daftar, start=1):`	Ulangi untuk tiap ubin. `enumerate` memberi nomor urut `i` mulai dari 1 (Bab 6).
20	komentar	Pengingat: Band_1 harus NIR dan Band_2 harus merah.
21-22	`band1 = Float(Raster(os.path.join(folder_input, nama, "Band_1")))`	Buka satu band dari ubin ini. Jalurnya berbentuk `...\ubin_01.tif\Band_1`: nama berkas, garis miring, lalu nama band. `Float` mengubahnya ke desimal.
23	`ndvi = (band1 - band2) / (band1 + band2)`	Rumus NDVI.
24	`ndvi.save(os.path.join(folder_output, "NDVI_" + nama))`	Simpan di folder keluaran, dengan nama `NDVI_` ditambah nama ubin.
25	`print(f"{i}/{total} selesai: {nama}")`	Cetak kemajuan, misalnya `1/3 selesai: ubin_01.tif`.
27-28	`arcpy.CheckInExtension("Spatial")` dan `print("Batch selesai.")`	Kembalikan ekstensi dan tutup dengan pesan.

Dokumentasi Esri menyebut `ListRasters` memerlukan folder kerja yang sudah diatur lebih dulu, dan filter namanya tidak peka huruf besar-kecil. Karena itu baris 13 harus mendahului baris 15. [CEK]

Versi ArcMap 10.8 (Python 2.7)

```python
# [SKRIP 3.1-ArcMap: Batch Processing Citra Drone dengan ArcPy]
# Penulis: Badar Mubarok Yogaswara
# [CEK] Cocokkan dengan versi Anda. Butuh ekstensi Spatial Analyst.
```

```
from __future__ import print_function   # agar print() berlaku di Python 2.7
import os
import arcpy
from arcpy.sa import Raster, Float

arcpy.CheckOutExtension("Spatial")
arcpy.env.overwriteOutput = True

folder_input = r"D:\Data_Drone\Ortofoto_Mentah"
folder_output = r"D:\Data_Drone\NDVI_Hasil"
arcpy.env.workspace = folder_input

daftar = arcpy.ListRasters("*", "TIF")
total = len(daftar)
print("Memulai batch {} ubin...".format(total))

for i, nama in enumerate(daftar, 1):
    # Urutan band mengikuti Skrip 3.1 QGIS: pastikan Band_1 = NIR dan Band_2 = Red
    band1 = Float(Raster(os.path.join(folder_input, nama, "Band_1")))
    band2 = Float(Raster(os.path.join(folder_input, nama, "Band_2")))
    ndvi = (band1 - band2) / (band1 + band2)
    ndvi.save(os.path.join(folder_output, "NDVI_" + nama))
    print("{}/{} selesai: {}".format(i, total, nama))

arcpy.CheckInExtension("Spatial")
print("Batch selesai.")
```

Bedanya di ArcMap: Tiga beda kecil. **Pertama**, baris 4 menambah `from __future__ import print_function`, sehingga nomor baris bergeser satu. **Kedua**, `enumerate(daftar, 1)` di baris 20: angka 1 ditulis sebagai isian biasa, bukan `start=1`. **Ketiga**, semua f-string diganti `.format`: `"Memulai batch {} ubin...".format(total)` dan `"{}/{} selesai: {}".format(i, total, nama)`. Tanda `{}` kosong diisi berurutan dari isian `.format`.

Hasil. Layar menampilkan lima baris, dan tiga berkas `NDVI_ubin_01.tif` sampai `NDVI_ubin_03.tif` muncul di `D:\Data_Drone\NDVI_Hasil`. Berikut perkiraannya. Nama ubin dan urutannya mengikuti folder contoh. [CEK]

HASIL

```
Memulai batch 3 ubin...
1/3 selesai: ubin_01.tif
2/3 selesai: ubin_02.tif
3/3 selesai: ubin_03.tif
Batch selesai.
```

Rata-rata NDVI tiap ubin, dihitung dengan GDAL dari data contoh, adalah kira-kira 0,471 untuk `ubin_01`, 0,471 untuk `ubin_02`, dan 0,467 untuk `ubin_03`. [CEK] Logika Python skrip sudah diuji dengan modul ArcPy tiruan buatan sendiri yang membaca paket data. Tiga berkas keluaran terbentuk dan cetakan kemajuannya sama dengan perkiraan di atas. Perilaku alat ArcGIS yang sebenarnya belum teruji.

Bagian yang boleh diubah. Nomor baris mengikuti versi Pro.

- **Baris 11-12, dua folder.** Ganti dengan folder Anda. Pertahankan huruf `r`.

- **Baris 15, pola nama dan format.** Ganti `"TIF"` untuk format lain, atau ganti `"*"` dengan pola nama, misalnya `"ubin_0*"`.
- **Baris 21-22, nomor band.** Tukar `Band_1` dan `Band_2` bila urutan band data Anda berbeda.
- **Baris 24, awalan nama.** Ganti `"NDVI_"` untuk awalan lain.

Jebakan.

- **Folder keluaran tidak dibuat.** Bila `D:\Data_Drone\NDVI_Hasil` belum ada, simpan di baris 24 kemungkinan besar gagal. Bab 31 menunjukkan cara membuat folder lewat `os.makedirs`. [CEK]
- **Band tertukar.** Bila band 1 sebenarnya merah, NDVI berubah tanda dan tidak ada peringatan. Cek urutan band dulu.
- Hanya `.tif`. Berkas berakhiran lain tidak ikut diproses karena filter `"TIF"`. [CEK]
- **Berjalan di depan.** Skrip ini menahan ArcGIS sampai selesai. Untuk puluhan ubin, jalankan dari terminal Python di luar aplikasi [CEK]. Versi QGIS-nya memakai tugas latar belakang (Bab 24).
- **Ekstensi tidak dikembalikan saat galat.** Sama dengan Skrip 30.1.

Latihan variasi. Proses hanya dua ubin pertama untuk uji coba. Contoh jawaban, ganti baris 15 dengan:

```python
daftar = arcpy.ListRasters("*", "TIF")[:2]
```

Tanda `[:2]` mengambil dua isi pertama daftar. Nilai `total` otomatis menjadi 2. Layar menampilkan `Memulai batch 2 ubin...`, `1/2 selesai: ubin_01.tif`, `2/2 selesai: ubin_02.tif`, lalu `Batch selesai.` Jawaban ini sudah diuji dengan modul ArcPy tiruan; perilaku ArcGIS yang sebenarnya belum teruji. [CEK]

> **Pola yang bisa dipakai ulang:** Kerangka "batch semua berkas di folder": **daftar berkas**, ulangi dengan `enumerate`, **kerjakan satu berkas**, **simpan dengan nama baru**, **lapor kemajuan**. Untuk data baru, ganti dua folder (baris 11-12), pola daftar (baris 15), dan rumus (baris 23).

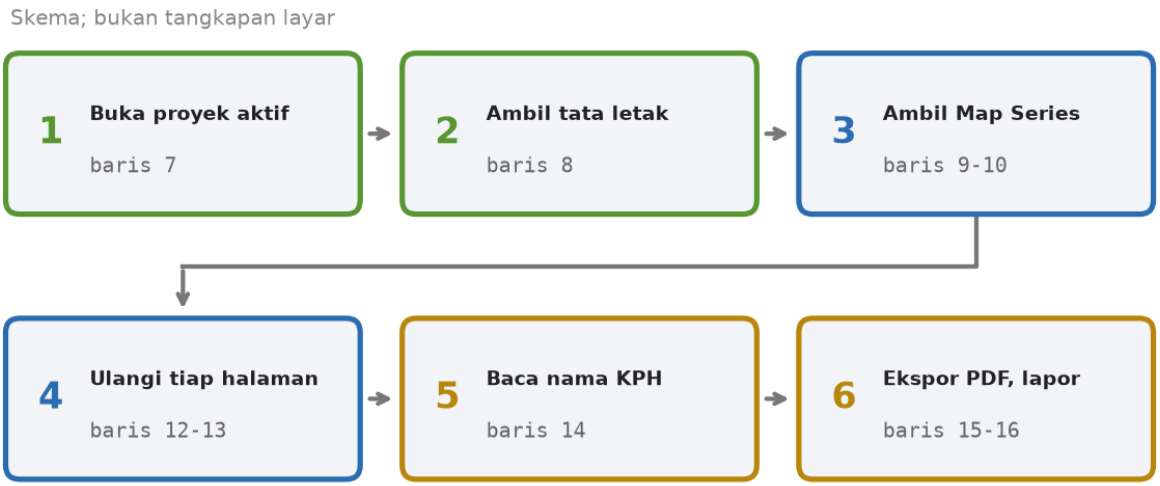
Skrip 30.4: Satu PDF laporan untuk setiap KPH

Berkas `bab6_skrip_6_1_pro.py`, `bab6_skrip_6_1_arcmap.py`, dan teks pendamping `bab6_teks_dinamis.txt`. Di dalam berkasnya tertulis "Skrip 6.1".

Tujuan. Mengekspor satu PDF per KPH dari tata letak peta yang halamannya dibuat satu per satu untuk tiap KPH.

Bahan. Skrip ini **tidak bisa dijalankan dari terminal biasa**. Ia memakai proyek (atau dokumen peta) yang sedang terbuka, jadi jalankan dari jendela Python di dalam aplikasi. Di Pro, proyek Anda harus punya tata letak bernama persis `Laporan Kebakaran Hutan` dengan Map Series aktif, memakai layer indeks yang punya kolom `NAMA_KPH` (misalnya `Batas_KPH`, 3 fitur). Di ArcMap, dokumen peta Anda harus punya Data Driven Pages aktif dengan layer indeks yang sama. Siapkan folder `D:\Laporan`. Skrip tidak membuatnya. Padanan PyQGIS-nya (Atlas) ada di Bab 24.

Ilustrasi 30.5: Alur Skrip 30.4



Versi ArcGIS Pro (Python 3)

```python
# [SKRIP 6.1-Pro: Ekspor PDF per KPH dari Map Series]
# Penulis: Badar Mubarok Yogaswara
# [CEK] Cocokkan dengan versi Anda.
import os
import arcpy

aprx = arcpy.mp.ArcGISProject("CURRENT")
lyt = aprx.listLayouts("Laporan Kebakaran Hutan")[0]
ms = lyt.mapSeries
folder_out = r"D:\Laporan"

for no in range(1, ms.pageCount + 1):
    ms.currentPageNumber = no
    nama = ms.pageRow.NAMA_KPH
    lyt.exportToPDF(os.path.join(folder_out, f"{nama}_Laporan_Bulan_Ini.pdf"))
    print(f"Selesai: {nama}")
```

Penjelasan baris demi baris. Nomor baris mengikuti berkas di atas.

Baris	Tulisan	Artinya
4-5	`import os`, `import arcpy`	Panggil dua perkakas.
7	`aprx = arcpy.mp.ArcGISProject("CURRENT")`	Ambil proyek ArcGIS Pro yang sedang terbuka.

Baris	Tulisan	Artinya
8	`lyt = aprx.listLayouts("Laporan Kebakaran Hutan")[0]`	Cari tata letak dengan nama itu. Hasilnya berupa daftar, jadi `[0]` mengambil yang pertama.
9	`ms = lyt.mapSeries`	Ambil Map Series milik tata letak itu.
10	`folder_out = r"D:\Laporan"`	Lokasi folder untuk PDF.
12	`for no in range(1, ms.pageCount + 1):`	Ulangi dari halaman 1 sampai halaman terakhir. `pageCount` adalah jumlah halaman. `range` berhenti satu sebelum angka akhir, jadi ditambah 1.
13	`ms.currentPageNumber = no`	Pindahkan Map Series ke halaman nomor `no`.
14	`nama = ms.pageRow.NAMA_KPH`	Baca isi kolom `NAMA_KPH` dari fitur halaman ini. Nama kolom ditulis setelah tanda titik.
15	`lyt.exportToPDF(os.path.join(folder_out, f"{nama}_Laporan_Bulan_Ini.pdf"))`	Ekspor tata letak ke PDF. Nama berkasnya nama KPH ditambah `_Laporan_Bulan_Ini.pdf`.
16	`print(f"Selesai: {nama}")`	Cetak nama KPH yang selesai.

Dokumentasi Esri mencatat `pageCount` bersifat baca saja, `currentPageNumber` boleh dibaca dan diisi, dan `pageRow` memberi atribut fitur indeks halaman yang aktif. Dokumentasi yang sama mencatat `Layout.exportToPDF(out_pdf, ...)` dan `ArcGISProject.listLayouts({wildcard})`. Semua itu cocok dengan skrip. [CEK]

Versi ArcMap 10.8 (Python 2.7)

```python
# [SKRIP 6.1-ArcMap: Ekspor PDF per KPH dari Data Driven Pages]
# Penulis: Badar Mubarok Yogaswara
# [CEK] Cocokkan dengan versi Anda.
from __future__ import print_function   # agar print() berlaku di Python 2.7
import os
import arcpy

mxd = arcpy.mapping.MapDocument("CURRENT")
ddp = mxd.dataDrivenPages
folder_out = r"D:\Laporan"

for no in range(1, ddp.pageCount + 1):
    ddp.currentPageID = no
    nama = ddp.pageRow.getValue("NAMA_KPH")
    arcpy.mapping.ExportToPDF(mxd, os.path.join(folder_out, nama + "_Laporan_Bulan_Ini.pdf"))
    print("Selesai: {}".format(nama))

del mxd
```

> **Bedanya di ArcMap:** Konsepnya sama, tetapi namanya lain. **Pertama**, baris 4 `from __future__ import print_function`, sehingga nomor baris bergeser satu. **Kedua**, dokumen peta dibuka dengan `arcpy.mapping.MapDocument("CURRENT")` (baris 8), bukan `arcpy.mp.ArcGISProject("CURRENT")`. **Ketiga**, tidak ada objek tata letak terpisah. Halaman per KPH diatur lewat `mxd.dataDrivenPages` (baris 9), dan tidak perlu mencari tata letak berdasarkan nama. **Keempat**, pindah halaman dengan `ddp.currentPageID = no` (baris 13), bukan `currentPageNumber`. **Kelima**, nama KPH dibaca dengan `ddp.pageRow.getValue("NAMA_KPH")` (baris 14), bukan `ms.pageRow.NAMA_KPH`. **Keenam**, ekspor memakai fungsi `arcpy.mapping.ExportToPDF(mxd, jalur)` (baris 15). **Ketujuh**, nama berkas disambung dengan tanda `+` dan bukan f-string. **Kedelapan**, baris terakhir `del mxd` melepas dokumen peta dari memori.

Dokumentasi ArcMap menyebut `DataDrivenPages` diakses lewat `MapDocument`, dengan sifat `pageCount`, `currentPageID`, dan `pageRow`. Fungsi `ExportToPDF(map_document, out_pdf, ...)` juga tercatat di sana. Cara membaca nilai lewat `getValue` belum penulis cocokkan dengan dokumentasi. [CEK]

Teks dinamis untuk tata letak ArcMap

Berkas `bab6_teks_dinamis.txt` bukan skrip Python. Isinya empat baris teks untuk elemen teks di tata letak ArcMap. Tanda `<dyn .../>` diganti isi kolom milik halaman yang tampil.

EKSPRESI

```
LAPORAN KEBAKARAN HUTAN KPH <dyn type="page" property="attribute" field="NAMA_KPH"/>
-----------------------------------
Total Luas KPH: <dyn type="page" property="attribute" field="Luas_Ha"/> Ha
Jumlah Titik Api Aktif: <dyn type="page" property="attribute" field="Join_Count"/> Titik.
```

Tiap tanda `<dyn type="page" property="attribute" field="..."/>` mengambil isi satu kolom dari layer indeks. Tiga kolom yang dipakai: `NAMA_KPH`, `Luas_Ha`, dan `Join_Count`. Kolom `Luas_Ha` dan `Join_Count` harus sudah ada di layer indeks. Skrip di bab ini tidak membuatnya. Di data contoh, luas ketiga KPH adalah 4, 4, dan 8 hektare, dan jumlah titik panas masing-masing 2, 5, dan 5. [CEK] Di Pro, teks dinamis serupa tersedia, tetapi berkas pendampingnya tidak ada di folder skrip. [CEK]

Hasil. Layar menampilkan satu baris untuk tiap halaman, dan PDF muncul di `D:\Laporan`. Dari data contoh, ada tiga halaman. Berikut perkiraannya. Urutan halaman mengikuti urutan fitur di layer indeks. [CEK]

HASIL

```
Selesai: KPH Alpha
Selesai: KPH Beta
Selesai: KPH Gamma
```

Tiga berkas PDF terbentuk: `KPH Alpha_Laporan_Bulan_Ini.pdf`, `KPH Beta_Laporan_Bulan_Ini.pdf`, dan `KPH Gamma_Laporan_Bulan_Ini.pdf`. Nama berkas memuat

spasi karena nama KPH memuat spasi. Di versi ArcMap, cetakannya sama persis, sebab `print("Selesai: {}".format(nama))` menghasilkan teks yang sama.

Bagian yang boleh diubah. Nomor baris mengikuti versi Pro.

- **Baris 8, nama tata letak.** Ganti dengan nama tata letak Anda, persis sama termasuk huruf besar dan spasi.
- **Baris 10, folder keluaran.** Ganti `D:\Laporan` dan pastikan folder itu ada.
- **Baris 14, nama kolom.** Ganti `NAMA_KPH` dengan kolom lain di layer indeks.
- **Baris 15, nama berkas.** Ganti `_Laporan_Bulan_Ini.pdf` untuk akhiran lain.

Jebakan.

- **Nama tata letak harus persis.** Bila tidak ada tata letak dengan nama itu, `listLayouts(...)` menghasilkan daftar kosong dan `[0]` gagal dengan `IndexError: list index out of range`. Pesan itu sudah diuji pada daftar kosong di Python 3. [CEK]
- **Map Series harus aktif.** Dokumentasi Esri menyebut `lyt.mapSeries` bernilai `None` bila Map Series tidak ada atau tidak aktif. Maka baris 12 gagal dengan `AttributeError: 'NoneType' object has no attribute 'pageCount'`. Pesan Python itu sudah diuji pada nilai `None`.
- Folder `D:\Laporan` harus ada. Skrip tidak membuatnya. [CEK]
- **Dari terminal biasa gagal.** Kata `"CURRENT"` hanya berarti bila ada proyek atau dokumen peta yang terbuka.
- **Nama berkas bisa tertimpa.** Dua KPH dengan nama sama menghasilkan satu PDF.

Latihan variasi. Beri nomor urut di depan nama berkas, supaya PDF terurut rapi. Contoh jawaban, ganti baris 15 dengan:

```python
    lyt.exportToPDF(os.path.join(folder_out, f"{no:02d}_{nama}.pdf"))
```

Bagian `:02d` memaksa nomor memakai dua angka dengan nol di depan. Nama berkasnya menjadi `01_KPH Alpha.pdf`, `02_KPH Beta.pdf`, dan `03_KPH Gamma.pdf`. Format `:02d` sudah diuji di Python 3, tetapi ekspor PDF-nya belum teruji. [CEK]

> **Pola yang bisa dipakai ulang:** Kerangka "satu keluaran per fitur": **buka proyek aktif**, **ambil seri halaman**, **ulangi tiap halaman**, **baca nama dari kolom indeks**, **ekspor dengan nama itu**. Untuk laporan lain, ganti nama tata letak (baris 8), folder (baris 10), dan nama kolom (baris 14).

Cek paham

1. Mengapa skrip Pro dan skrip ArcMap pada Skrip 30.2 menghitung persen akurasi dengan cara sedikit berbeda?
2. Apa yang terjadi bila folder `D:\Data_Drone\NDVI_Hasil` belum ada saat Skrip 30.3 dijalankan?
3. Mengapa Skrip 30.4 tidak bisa dijalankan dari terminal biasa?

Jawaban.

1. Di Python 2.7 (ArcMap), pembagian dua bilangan bulat menghasilkan bilangan bulat, sehingga 9 dibagi 10 menjadi 0. Versi ArcMap memakai `float(titik_cocok)` agar hasilnya desimal.
2. Skrip tidak membuat folder itu. Simpan hasil di baris 24 kemungkinan besar gagal. Buat foldernya lebih dulu. [CEK]
3. Karena memakai `"CURRENT"`, yaitu proyek atau dokumen peta yang sedang terbuka. Di terminal biasa tidak ada yang terbuka.

Kesalahan umum

- **Lupa meminjam ekstensi.** Alat raster seperti `ExtractByMask` memerlukan Spatial Analyst. Perbaikan: cek baris `arcpy.CheckOutExtension("Spatial")` ada di awal.
- **Menyalin gaya Python 3 ke ArcMap.** Contohnya f-string. Perbaikan: pakai `.format` atau `%` di ArcMap.
- Memakai jalur tanpa huruf `r`. Perbaikan: tulis `r"C:\data_kph"`.
- Menjalankan skrip ber-`CURRENT` di luar aplikasi. Perbaikan: jalankan dari jendela Python.

Ringkasan dan latihan

- Empat pekerjaan klasik dituliskan dalam ArcPy: NDVI dari dua band, akurasi titik survei, batch NDVI, dan PDF per KPH.
- Pola raster: pinjam ekstensi, buka band, hitung dengan `Float`, simpan, kembalikan ekstensi.
- Pola vektor: `SpatialJoin`, lalu kursor `SearchCursor` per baris.
- Beda ArcMap yang utama: `from __future__ import print_function`, `float(...)` untuk pembagian, `.format`, nama alat bergaya `_analysis`, dan `arcpy.mapping`.
- Semua skrip di bab ini belum dijalankan di ArcGIS dan bertanda **[CEK]**.

Latihan. Ubah Skrip 30.1 supaya memotong dan menghitung NDVI untuk data milik Anda sendiri. Bagian mana yang Anda ganti, dan bagian mana yang tetap?

Tabel perbandingan: PyQGIS dan ArcPy pada empat pekerjaan

Pekerjaan	PyQGIS (Bab 24)	ArcPy (Pro)
Potong raster dengan batas	alat Processing pemotong raster	`ExtractByMask(raster, batas)`
Hitung NDVI	kalkulator raster dengan ekspresi	`(Float(nir) - Float(red)) / (...)`
Gabung titik dengan poligon	alat gabung atribut menurut lokasi	`arcpy.analysis.SpatialJoin(...)`
Daftar raster di folder	daftar berkas lewat Python	`arcpy.ListRasters("*", "TIF")`
Satu PDF per KPH	Atlas pada tata letak	`lyt.mapSeries` dan `lyt.exportToPDF(...)`

BAB 31: Dua Gaya, Satu Pekerjaan: ArcPy di ArcGIS Pro dan ArcMap

> **Berdasarkan Seri I4:** Skrip bab ini berasal dari Buku I4, *Otomasi Dasar dengan Python* (Intermediate Class). Berkasnya `b1_arcpy_pro.py`, `b2_arcpy_pro.py`, `b3_arcpy_pro.py`, `b4_arcpy_pro.py`, dan `b7_arcpy_pro.py` di folder `arcpy_pro`, serta pasangannya `b1_arcpy_arcmap.py`, `b2_arcpy_arcmap.py`, `b3_arcpy_arcmap.py`, `b4_arcpy_arcmap.py`, dan `b7_arcpy_arcmap.py` di folder `arcpy_arcmap`. Di buku itu, keduanya dipasangkan dengan skrip QGIS yang sepadan. Padanan PyQGIS-nya ada di Bab 14, 21, 22, dan 23. Bab 5 dan 6 buku I4 tidak punya pasangan ArcPy.

> **Bagian C hanya untuk Windows:** ArcGIS Pro dan ArcMap 10.8 hanya berjalan di Windows. Di macOS dan Ubuntu, skrip bab ini tidak bisa Anda jalankan. Semua skrip di bagian ini bertanda **[CEK]**: cocokkan dengan versi ArcGIS di komputer Anda.

Studi kasus: "Kantor cabang memakai ArcGIS"

Kepala Seksi membawa kabar: kantor cabang tidak memakai QGIS. Sebagian komputer di sana memakai ArcGIS Pro, sebagian lagi masih ArcMap 10.8. Pekerjaannya sama persis: membaca batas KPH, menghitung luas, dan memasukkan layer ke peta. Yang berbeda hanya cara menuliskannya. Anda perlu tahu dua gaya itu supaya skrip bisa berpindah dari satu komputer ke komputer lain.

Konsep: ArcPy dalam tiga kalimat

ArcPy adalah perkakas Python bawaan ArcGIS. Bayangkan ArcGIS sebagai kantor dengan banyak loket, dan ArcPy sebagai buku telepon tiap loket: satu panggilan, satu pekerjaan selesai. ArcGIS Pro dan ArcMap punya buku telepon yang hampir sama, tetapi sebagian nomornya ditulis berbeda.

Ilustrasi 31.1: Pekerjaan yang sama, dua gaya penulisan

Pekerjaan	ArcGIS Pro	ArcMap 10.8
Versi Python	`Python 3`	`Python 2.7`
Buka dokumen peta	`arcpy.mp.ArcGISProject("CURRENT")`	`arcpy.mapping.MapDocument("CURRENT")`
Hitung jumlah fitur	`arcpy.management.GetCount(x)[0]`	`arcpy.GetCount_management(x).getOutput(0)`
Cetak teks	`print(nama, luas)`	`print("%s %.2f" % (nama, luas))`
Bagi angka	`luas / 10000`	`luas / 10000.0`

Skema; bukan tangkapan layar

Perbedaan terbesarnya ada di versi Python. ArcGIS Pro memakai **Python 3**. ArcMap 10.8 memakai **Python 2.7**, versi lama yang cara menulis beberapa hal berbeda. Karena itu setiap skrip di bab ini hadir dalam dua versi.

Lima istilah yang perlu Anda kenal sekarang:

- `arcpy.env.workspace`: folder kerja bawaan. Nama data yang Anda tulis dicari di folder ini.
- `Describe`: perintah untuk melihat sifat sebuah data, misalnya jenis bentuk dan sistem koordinatnya.
- **Kursor** (*cursor*): penunjuk yang membaca tabel baris demi baris, seperti jari yang menelusuri daftar.
- **Proyek** (di Pro) dan **dokumen peta** (di ArcMap): berkas yang menyimpan peta Anda.
- **Layer**: satu lapis data di peta. Istilah ini sama dengan di QGIS.

Skrip 31.1: Membaca data, atribut, dan proyek

Tujuan. Melihat daftar data di sebuah folder, memeriksa sifat batas KPH, membaca nama dan luas tiap KPH, membuka sebuah raster, lalu memasukkan layer ke peta.

Bahan. Paket data `paket-i4`: folder `shp` (berisi `Batas_KPH.shp`) dan berkas `Citra_KPH.tif`. Bagian 4 skrip hanya berjalan dari **jendela Python di dalam ArcGIS Pro**, karena memakai proyek yang sedang terbuka. [CEK]

Versi ArcGIS Pro (Python 3)

PYTHON

```
# [SKRIP 1.1-Pro: ArcPy dasar: membaca data, atribut, dan proyek]
# Penulis: Badar Mubarok Yogaswara
# [CEK] Cocokkan dengan versi ArcGIS Pro Anda.
import arcpy

DATA = r"D:\Latihan\paket-i4"
arcpy.env.workspace = DATA + r"\shp"          # folder Shapefile; daftar data dibaca dari sini

# 1. Daftar data dan sifatnya
print(arcpy.ListFeatureClasses())              # contoh: ['Batas_KPH.shp', 'Hotspot_Nasional.shp', ...]
batas = "Batas_KPH.shp"
d = arcpy.Describe(batas)
print(d.shapeType, d.spatialReference.name, d.spatialReference.factoryCode)   # Polygon, ..., 32749
print("Jumlah fitur:", int(arcpy.management.GetCount(batas)[0]))
print("Field:", [f.name for f in arcpy.ListFields(batas)])

# 2. Membaca tiap baris dengan kursor (padanan getFeatures() di QGIS)
with arcpy.da.SearchCursor(batas, ["NAMA_KPH", "SHAPE@AREA"]) as kursor:
    for nama, luas in kursor:
        print(nama, round(luas / 10000, 2), "ha")          # luas dalam satuan peta (m2) dibagi 10.000

# 3. Raster
r = arcpy.Raster(DATA + r"\Citra_KPH.tif")
print("Raster:", r.width, "x", r.height, "piksel,", r.bandCount, "band; piksel", r.meanCellWidth, "m")

# 4. Proyek: tambah data lalu ambil layer dengan nama
aprx = arcpy.mp.ArcGISProject("CURRENT")      # proyek yang sedang terbuka
peta = aprx.listMaps()[0]
peta.addDataFromPath(DATA + r"\shp\Batas_KPH.shp")
layer = peta.listLayers("Batas_KPH")[0]       # listLayers() juga mengembalikan DAFTAR
print("Layer ditemukan:", layer.name)
```

```
aprx.save()
```

Alur skrip ini mirip skrip QGIS yang sudah Anda kenal: siapkan lokasi data, baca sifatnya, ulangi tiap baris, lalu masukkan ke peta.

Penjelasan baris demi baris. Nomor baris mengikuti skrip di atas.

Baris	Tulisan	Artinya
1-3	`# [SKRIP 1.1-Pro ...`	Komentar: judul, nama penulis, dan pengingat **[CEK]**.
4	`import arcpy`	Panggil perkakas ArcPy. Tanpa baris ini, perintah `arcpy...` tidak dikenal.
6	`DATA = r"D:\Latihan\paket-i4"`	Isi kotak `DATA` dengan lokasi folder data. Huruf `r` di depan kutip membuat garis miring terbalik `\` dibaca apa adanya.
7	`arcpy.env.workspace = ...`	Tetapkan folder kerja ke subfolder `shp`. Mulai sekarang, nama berkas seperti `"Batas_KPH.shp"` dicari di sana.
10	`print(arcpy.ListFeatureClasses())`	Tampilkan daftar data vektor di folder kerja.
11-12	`batas = ...,d = arcpy.Describe(batas)`	Simpan nama data di kotak `batas`. Lalu `Describe` mengumpulkan sifatnya ke kotak `d`.
13	`print(d.shapeType, ...)`	Tampilkan jenis bentuk, nama sistem koordinat, dan kode angkanya.
14	`arcpy.management.GetCount(batas)[0]`	Hitung jumlah fitur. Hasilnya berupa teks, jadi `int(...)` mengubahnya menjadi angka.
15	`[f.name for f in arcpy.ListFields(batas)]`	Kumpulkan nama semua kolom ke dalam satu daftar.
18-20	`with arcpy.da.SearchCursor(...) as kursor:`	Buka kursor pada dua kolom: `NAMA_KPH` dan `SHAPE@AREA` (luas bentuk). Perulangan `for` membaca satu baris tiap putaran, dan dua isiannya langsung masuk ke kotak `nama` dan `luas`.
20	`round(luas / 10000, 2)`	Ubah luas dari meter persegi ke hektare, lalu bulatkan dua desimal.
23-24	`arcpy.Raster(...)`	Buka sebuah raster, lalu tampilkan lebar, tinggi, jumlah band, dan ukuran piksel.
27	`arcpy.mp.ArcGISProject("CURRENT")`	Ambil proyek ArcGIS Pro yang sedang terbuka. Kata `CURRENT` berarti "yang sekarang".
28-29	`aprx.listMaps()[0], peta.addDataFromPath(...)`	Ambil peta pertama dari proyek, lalu tambahkan data batas sebagai layer.
30-31	`peta.listLayers("Batas_KPH")[0]`	Cari layer bernama `Batas_KPH`. Hasil pencarian berupa daftar, jadi `[0]` mengambil yang pertama.
32	`aprx.save()`	Simpan proyek.

Hasil yang diharapkan. Berikut perkiraan keluarannya. Angka-angka ini dihitung dari data dengan alat lain, bukan dijalankan di ArcGIS, jadi tulisan persisnya bisa sedikit berbeda. [CEK]

```
HASIL
Jumlah fitur: 3
KPH Alpha 4.0 ha
KPH Beta 4.0 ha
KPH Gamma 8.0 ha
Raster: 200 x 200 piksel, 4 band; piksel 2.0 m
Layer ditemukan: Batas_KPH
```

Baris sistem koordinat memuat angka `32749`, yaitu kode UTM zona 49 Selatan.

Versi ArcMap 10.8 (Python 2.7)

```
PYTHON
# -*- coding: utf-8 -*-
# [SKRIP 1.1-ArcMap: ArcPy dasar: membaca data, atribut, dan dokumen peta (Python 2.7)]
# Penulis: Badar Mubarok Yogaswara
# [CEK] Cocokkan dengan versi ArcMap Anda.
import arcpy

DATA = r"D:\Latihan\paket-i4"
arcpy.env.workspace = DATA + r"\shp"

# 1. Daftar data dan sifatnya
print(arcpy.ListFeatureClasses())
batas = "Batas_KPH.shp"
d = arcpy.Describe(batas)
print("%s %s %s" % (d.shapeType, d.spatialReference.name, d.spatialReference.factoryCode))
print("Jumlah fitur: %d" % int(arcpy.GetCount_management(batas).getOutput(0)))
print("Field: %s" % [f.name for f in arcpy.ListFields(batas)])

# 2. Membaca tiap baris dengan kursor
with arcpy.da.SearchCursor(batas, ["NAMA_KPH", "SHAPE@AREA"]) as kursor:
    for nama, luas in kursor:
        print("%s %.2f ha" % (nama, luas / 10000.0))        # 10000.0: hasil bagi desimal di Python 2

# 3. Raster
r = arcpy.Raster(DATA + r"\Citra_KPH.tif")
print("Raster: %d x %d piksel, %d band, piksel %s m" % (r.width, r.height, r.bandCount, r.meanCellWidth))

# 4. Dokumen peta: tambah layer lalu ambil dengan nama
mxd = arcpy.mapping.MapDocument("CURRENT")
df = arcpy.mapping.ListDataFrames(mxd)[0]
arcpy.mapping.AddLayer(df, arcpy.mapping.Layer(DATA + r"\shp\Batas_KPH.shp"))
layer = arcpy.mapping.ListLayers(mxd, "Batas_KPH", df)[0]    # juga mengembalikan DAFTAR
print("Layer ditemukan: %s" % layer.name)
mxd.save()
```

> **Bedanya di ArcMap:** Strukturnya sama dengan versi Pro: empat bagian, urutan yang sama. Hanya penulisannya yang berbeda, dan ada empat beda yang perlu Anda hafal. **Pertama**, baris 1 menambah `# -*- coding: utf-8 -*-` agar huruf Indonesia dibaca benar di Python 2.7. **Kedua**, `print` hanya diberi satu teks hasil rakitan dengan tanda `%`, karena `print` Python 2.7 berperilaku lain bila diberi beberapa isian. **Ketiga**, perintah hitung ditulis `arcpy.GetCount_management(...).getOutput(0)`, bukan `arcpy.management.GetCount(...)[0]`. **Keempat**, urusan peta memakai modul `arcpy.mapping` dan ada satu perantara bernama *data frame*, sedangkan Pro memakai `arcpy.mp`.

Bagian yang boleh diubah. Nomor baris mengikuti versi ArcGIS Pro.

- **Baris 6, lokasi data.** Ganti `D:\Latihan\paket-i4` dengan lokasi folder `paket-i4` di komputer Anda. Pertahankan huruf `r` di depan kutip.
- **Baris 11, nama data.** Ganti `Batas_KPH.shp` dengan data lain di folder `shp`, misalnya `Sungai.shp`.
- **Baris 18, nama kolom.** Ganti `NAMA_KPH` dengan kolom lain. Daftar kolom ada di hasil baris 15.
- **Baris 23, raster.** Ganti `Citra_KPH.tif` dengan raster lain di folder yang sama.
- **Baris 30, nama layer.** Ganti `Batas_KPH` sesuai nama layer di peta Anda.

Jebakan.

- **Satuan luas.** `SHAPE@AREA` memakai satuan peta. Pada data UTM dalam meter, hasilnya meter persegi, dan pembagian 10.000 mengubahnya menjadi hektare. Pada data berkoordinat derajat, angkanya bukan hektare.
- `CURRENT` di luar ArcGIS. Bagian 4 butuh proyek (atau dokumen peta) yang sedang terbuka. Dari terminal biasa, bagian ini gagal. Jalankan dari jendela Python di dalam aplikasinya. [CEK]
- `print` dengan banyak isian di ArcMap. Gaya `print(nama, luas)` milik Python 3 bisa mencetak tanda kurung dan koma di Python 2.7. Pakai satu teks dengan `%`.
- **Garis miring terbalik.** Lokasi data tanpa huruf `r` di depan kutip bisa dibaca salah, karena `\` dipakai Python untuk tanda khusus.

Latihan variasi. Ubah bagian 2 supaya hanya KPH yang lebih luas dari 5 hektare yang tercetak. Contoh jawaban untuk ArcGIS Pro:

```python
with arcpy.da.SearchCursor(batas, ["NAMA_KPH", "SHAPE@AREA"]) as kursor:
    for nama, luas in kursor:
        if luas / 10000 > 5:
            print(nama, round(luas / 10000, 2), "ha")
```

Dari data contoh, hanya satu baris yang tercetak: `KPH Gamma 8.0 ha`. [CEK]

> **Pola yang bisa dipakai ulang:** Skrip ArcPy hampir selalu berbentuk empat tahap: **tetapkan lokasi data**, **cek sifat data**, **ulangi tiap baris dengan kursor**, **masukkan hasil ke peta**. Untuk data baru, ganti tiga hal: lokasi data (baris 6), nama data (baris 11), dan nama kolom (baris 18).

Skrip 31.2: Rantai alat, dari sempadan sungai sampai NDVI per KPH

Berkas `b2_arcpy_pro.py` dan `b2_arcpy_arcmap.py`. Di dalam berkasnya tertulis "Skrip 2.1" mengikuti penomoran Buku I4. Padanan PyQGIS-nya ada di Bab 21.

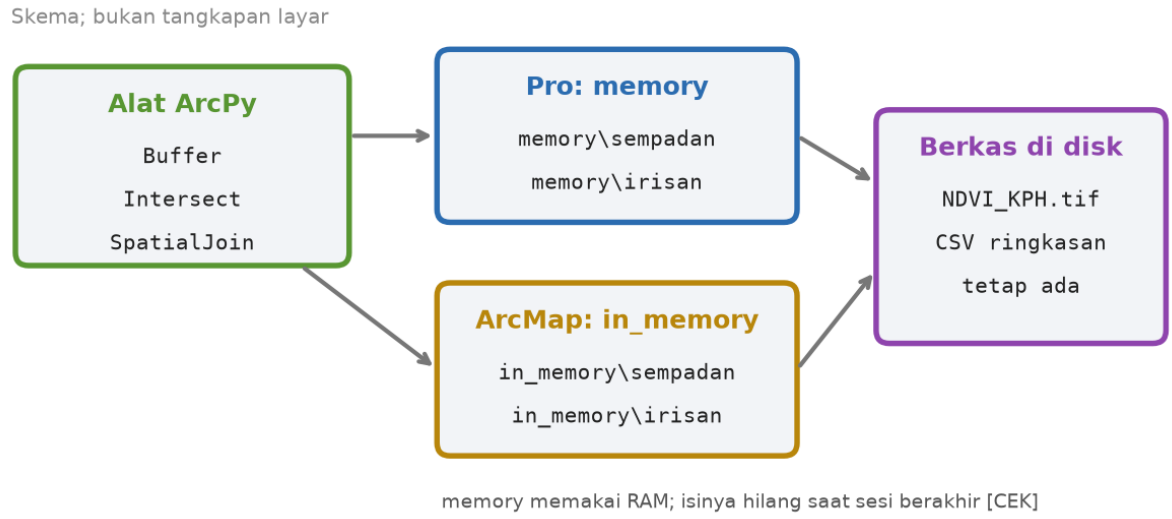
Tujuan. Menjalankan beberapa alat ArcPy berurutan: membuat sempadan sungai 30 m, memotongnya dengan batas KPH, menghitung titik panas per KPH, lalu menghitung NDVI dan rata-ratanya untuk tiap KPH.

Bahan. Paket data `paket-i4`: folder `shp` (`Sungai.shp`, `Batas_KPH.shp`, `Hotspot_Nasional.shp`) dan `Citra_KPH.tif` (empat band: merah, hijau, biru, inframerah dekat). Siapkan juga folder `D:\Latihan\hasil_i4`: **skrip tidak membuatnya**. Ekstensi Spatial Analyst harus tersedia.

Enam istilah baru:

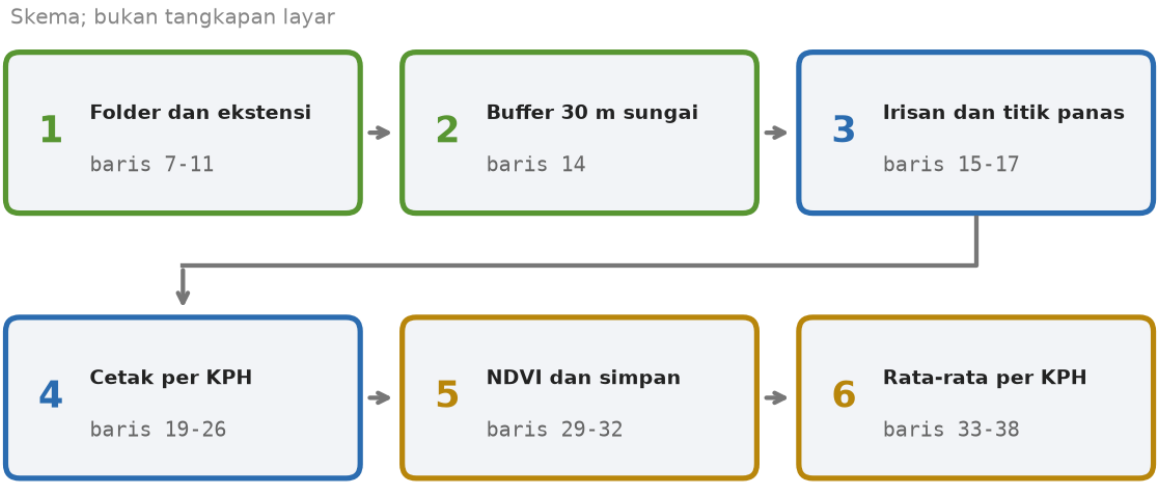
- **Alat geoprocessing**: satu pekerjaan siap pakai di ArcGIS, misalnya `Buffer`. Di skrip, alat ditulis sebagai perintah ArcPy.
- **Sempadan** (*buffer*): daerah selebar jarak tertentu di kiri-kanan sebuah garis.
- **Ruang sementara** (`memory` di Pro): tempat menyimpan hasil antara di memori komputer, tanpa menulis berkas.
- **Kamus** (*dictionary*): daftar pasangan "nama dan nilai". Dibahas di Bab 5.
- **Statistik zonal**: menghitung angka ringkasan, misalnya rata-rata, untuk nilai raster di dalam tiap poligon.
- `Join_Count`: kolom jumlah pasangan yang ditambahkan alat Spatial Join.

Ilustrasi 31.2: Ruang sementara dan berkas di disk



Skrip ini bekerja dalam enam tahap.

Ilustrasi 31.3: Alur Skrip 31.2



Versi ArcGIS Pro (Python 3)

PYTHON

```
# [SKRIP 2.1-Pro: Rantai alat: sempadan sungai, titik panas, dan NDVI per KPH dengan ArcPy]
# Penulis: Badar Mubarok Yogaswara
# [CEK] Cocokkan dengan versi Anda. Zonal Statistics as Table dan Float butuh ekstensi Spatial Analyst.
import arcpy
from arcpy.sa import Raster, Float, ZonalStatisticsAsTable

DATA = r"D:\Latihan\paket-i4"
HASIL = r"D:\Latihan\hasil_i4"
arcpy.env.workspace = DATA + r"\shp"
arcpy.env.overwriteOutput = True
arcpy.CheckOutExtension("Spatial")

# Keluaran sementara: ruang kerja "memory" (padanan TEMPORARY_OUTPUT di QGIS)
arcpy.analysis.Buffer("Sungai.shp", r"memory\sempadan", "30 Meters", dissolve_option="ALL")
arcpy.analysis.Intersect(["Batas_KPH.shp", r"memory\sempadan"], r"memory\irisan")
arcpy.analysis.SpatialJoin("Batas_KPH.shp", "Hotspot_Nasional.shp", r"memory\kph_titik",
                           "JOIN_ONE_TO_ONE", "KEEP_ALL", match_option="INTERSECT")   # menambah Join_Count

sempadan = {}
with arcpy.da.SearchCursor(r"memory\irisan", ["NAMA_KPH", "SHAPE@AREA"]) as k:
    for nama, luas in k:
        sempadan[nama] = sempadan.get(nama, 0) + luas / 10000
with arcpy.da.SearchCursor(r"memory\kph_titik", ["NAMA_KPH", "Join_Count", "SHAPE@AREA"]) as k:
    for nama, jml, luas in k:
        print("%s: luas %.2f ha, sempadan %.2f ha, titik panas %d"
              % (nama, luas / 10000, sempadan.get(nama, 0), jml))

# NDVI = (NIR - Merah) / (NIR + Merah); band 4 = NIR, band 1 = Merah. Nama band ditulis jalur\Band_n.
nir = Float(Raster(DATA + r"\Citra_KPH.tif\Band_4"))
merah = Float(Raster(DATA + r"\Citra_KPH.tif\Band_1"))
ndvi = (nir - merah) / (nir + merah)
ndvi.save(HASIL + r"\NDVI_KPH.tif")
ZonalStatisticsAsTable("Batas_KPH.shp", "NAMA_KPH", ndvi, r"memory\ndvi_zonal", "DATA", "MEAN")
with arcpy.da.SearchCursor(r"memory\ndvi_zonal", ["NAMA_KPH", "MEAN"]) as k:
    for nama, rata in k:
        print("%s: NDVI rata-rata %.3f" % (nama, rata))
arcpy.management.Delete(r"memory\sempadan")      # ruang memory memakai RAM: bersihkan setelah selesai
```

```
arcpy.CheckInExtension("Spatial")
```

Penjelasan baris demi baris. Nomor baris mengikuti berkas di atas. Baris `import`, ekstensi, dan folder kerja sudah Anda kenal di Skrip 31.1 dan Bab 30.

Baris	Tulisan	Artinya
1-3	`# [SKRIP 2.1-Pro ...`	Komentar: judul, penulis, dan pengingat bahwa Zonal Statistics dan `Float` butuh Spatial Analyst.
4-5	`import arcpy` dan `from arcpy.sa import ...`	Panggil ArcPy, lalu tiga perkakas bagian raster: `Raster`, `Float`, dan `ZonalStatisticsAsTable`.
7-8	`DATA = ...`, `HASIL = ...`	Simpan lokasi data dan lokasi hasil.
9-11	`arcpy.env.workspace = DATA + r"\shp"` dan dua baris sesudahnya	Folder kerja diarahkan ke `shp`, keluaran boleh menimpa, dan ekstensi Spatial dipinjam.
14	`arcpy.analysis.Buffer("Sungai.shp", r"memory\sempadan", "30 Meters", dissolve_option="ALL")`	Buat sempadan sungai. Isian: data masukan, tempat keluaran (di ruang sementara), jarak (angka dan satuan dalam satu teks), lalu `dissolve_option="ALL"` yang menggabungkan semua sempadan jadi satu bentuk.
15	`arcpy.analysis.Intersect([...], r"memory\irisan")`	Irisan: ambil bagian yang tumpang tindih antara `Batas_KPH.shp` dan sempadan. Dua masukan ditulis dalam satu daftar `[...]`.
16-17	`arcpy.analysis.SpatialJoin(... match_option="INTERSECT")`	Gabungkan titik panas ke KPH. Alat menambahkan kolom `Join_Count`: jumlah titik panas per KPH. Dua teks pilihan, `JOIN_ONE_TO_ONE` dan `KEEP_ALL`, sama dengan Skrip 30.2. Pilihan `match_option` ditulis dengan namanya, sehingga isian yang dilewati tidak perlu diisi `""`.
19	`sempadan = {}`	Siapkan kamus kosong untuk luas sempadan per KPH.
20-22	`with arcpy.da.SearchCursor(r"memory\irisan", [...]) as k:`	Baca irisan baris demi baris. Baris 22 menambah luas sempadan: `sempadan.get(nama, 0)` mengambil nilai lama (atau 0 bila belum ada), lalu ditambah luas baru dalam hektare. Satu KPH bisa punya beberapa potongan, jadi luasnya dijumlahkan.
23-26	`with arcpy.da.SearchCursor(r"memory\kph_titik", [...]) as k:`	Baca hasil gabungan. Tiap baris dicetak dengan tiga isian: `%s` untuk teks, `%.2f` untuk desimal dua angka, `%d` untuk bilangan bulat. Tanda kurung membuat cetakan boleh berlanjut ke baris berikutnya.
28-30	`nir = Float(Raster(DATA + r"\Citra_KPH.tif\Band_4"))`	Buka band 4 (NIR) dan band 1 (merah). Nama band ditulis setelah nama berkas: `\Band_4`.
31	`ndvi = (nir - merah) / (nir + merah)`	Rumus NDVI.
32	`ndvi.save(HASIL + r"\NDVI_KPH.tif")`	Simpan NDVI ke folder hasil.

Baris	Tulisan	Artinya
33	`ZonalStatisticsAsTable("Batas_KPH.shp", "NAMA_KPH", ndvi, r"memory\ndvi_zonal", "DATA", "MEAN")`	Hitung rata-rata (`"MEAN"`) NDVI di dalam tiap poligon KPH. Isian: poligon zona, kolom pembeda zona, raster nilai, tabel keluaran, `"DATA"` (abaikan piksel tanpa nilai), dan jenis statistik.
34-36	`with arcpy.da.SearchCursor(r"memory\ndvi_zonal", ["NAMA_KPH", "MEAN"]) as k:`	Baca tabel hasil dan cetak rata-rata NDVI tiap KPH dengan tiga angka desimal.
37	`arcpy.management.Delete(r"memory\sempadan")`	Hapus satu objek sementara. Ruang memory memakai RAM, jadi dibersihkan.
38	`arcpy.CheckInExtension("Spatial")`	Kembalikan ekstensi.

Dokumentasi Esri mencatat bahwa `ZonalStatisticsAsTable` memerlukan Spatial Analyst atau Image Analyst, dengan urutan isian `in_zone_data, zone_field, in_value_raster, out_table, ignore_nodata, statistics_type`. Urutan itu cocok dengan baris 33. Dokumentasi `Buffer` memberi contoh jarak berformat `"30 Meters"`, dan nilai `"ALL"` pada `dissolve_option` menggabungkan semua sempadan menjadi satu fitur. Dokumentasi `SpatialJoin` menyebut kolom `Join_Count` ditambahkan otomatis. [CEK]

Versi ArcMap 10.8 (Python 2.7)

PYTHON

```
# -*- coding: utf-8 -*-
# [SKRIP 2.1-ArcMap: Rantai alat: sempadan sungai, titik panas, dan NDVI per KPH (Python 2.7)]
# Penulis: Badar Mubarok Yogaswara
# [CEK] Cocokkan dengan versi Anda. Zonal Statistics as Table dan Float butuh ekstensi Spatial Analyst.
import arcpy
from arcpy.sa import Raster, Float, ZonalStatisticsAsTable

DATA = r"D:\Latihan\paket-i4"
HASIL = r"D:\Latihan\hasil_i4"
arcpy.env.workspace = DATA + r"\shp"
arcpy.env.overwriteOutput = True
arcpy.CheckOutExtension("Spatial")

# Keluaran sementara di ArcMap: ruang kerja in_memory (di Pro namanya memory)
arcpy.Buffer_analysis("Sungai.shp", r"in_memory\sempadan", "30 Meters", dissolve_option="ALL")
arcpy.Intersect_analysis(["Batas_KPH.shp", r"in_memory\sempadan"], r"in_memory\irisan")
arcpy.SpatialJoin_analysis("Batas_KPH.shp", "Hotspot_Nasional.shp", r"in_memory\kph_titik",
                           "JOIN_ONE_TO_ONE", "KEEP_ALL", match_option="INTERSECT")   # menambah Join_Count

sempadan = {}
with arcpy.da.SearchCursor(r"in_memory\irisan", ["NAMA_KPH", "SHAPE@AREA"]) as k:
    for nama, luas in k:
        sempadan[nama] = sempadan.get(nama, 0) + luas / 10000.0
with arcpy.da.SearchCursor(r"in_memory\kph_titik", ["NAMA_KPH", "Join_Count", "SHAPE@AREA"]) as k:
    for nama, jml, luas in k:
        print("%s: luas %.2f ha, sempadan %.2f ha, titik panas %d"
              % (nama, luas / 10000.0, sempadan.get(nama, 0), jml))

# NDVI = (NIR - Merah) / (NIR + Merah); band 4 = NIR, band 1 = Merah
nir = Float(Raster(DATA + r"\Citra_KPH.tif\Band_4"))
merah = Float(Raster(DATA + r"\Citra_KPH.tif\Band_1"))
ndvi = (nir - merah) / (nir + merah)
```

```
ndvi.save(HASIL + r"\NDVI_KPH.tif")
ZonalStatisticsAsTable("Batas_KPH.shp", "NAMA_KPH", ndvi, r"in_memory\ndvi_zonal", "DATA", "MEAN")
with arcpy.da.SearchCursor(r"in_memory\ndvi_zonal", ["NAMA_KPH", "MEAN"]) as k:
    for nama, rata in k:
        print("%s: NDVI rata-rata %.3f" % (nama, rata))
arcpy.Delete_management("in_memory")      # bersihkan seluruh ruang in_memory
arcpy.CheckInExtension("Spatial")
```

> **Bedanya di ArcMap:** Ada lima beda. **Pertama**, baris 1 menambah `# -*- coding: utf-8 -*-`, sehingga semua nomor baris bergeser satu. **Kedua**, nama alat ditulis bergaya lama: `arcpy.Buffer_analysis(...)`, `arcpy.Intersect_analysis(...)`, dan `arcpy.SpatialJoin_analysis(...)`. **Ketiga**, ruang sementara bernama `in_memory`, bukan `memory`. **Keempat**, pembagian ke hektare ditulis `luas / 10000.0` (baris 23 dan 27), agar hasilnya desimal di Python 2.7. **Kelima**, pembersihan di baris 38 memakai `arcpy.Delete_management("in_memory")`, yang menghapus **seluruh** ruang `in_memory`, bukan satu objek. Keluarannya tetap tercetak dengan satu teks hasil rakitan `%`, jadi aman di Python 2.7.

Hasil. Layar menampilkan enam baris, dan `NDVI_KPH.tif` muncul di folder hasil. Berikut perkiraannya. Luas dan sempadan dihitung dengan GDAL dan OGR dari data contoh, dan sejalan dengan `README.txt` paket data. Urutan baris mengikuti urutan tabel. [CEK]

```
HASIL
KPH Alpha: luas 4.00 ha, sempadan 1.18 ha, titik panas 2
KPH Beta: luas 4.00 ha, sempadan 1.39 ha, titik panas 5
KPH Gamma: luas 8.00 ha, sempadan 1.50 ha, titik panas 5
KPH Alpha: NDVI rata-rata 0.604
KPH Beta: NDVI rata-rata 0.431
KPH Gamma: NDVI rata-rata 0.329
```

Skrip PyQGIS padanannya di Bab 21 menghasilkan angka yang sama. Logika Python skrip ini sudah diuji dengan modul ArcPy tiruan buatan sendiri yang membaca paket data. Perilaku alat ArcGIS yang sebenarnya belum teruji. [CEK]

Bagian yang boleh diubah. Nomor baris mengikuti versi Pro.

- **Baris 7-8, lokasi data dan hasil.** Ganti sesuai komputer Anda. Pertahankan huruf `r`.
- **Baris 14, jarak.** Ganti `"30 Meters"` untuk jarak lain. Satuan ikut ditulis dalam teksnya.
- **Baris 29-30, nomor band.** Ganti `Band_4` dan `Band_1` bila urutan band citra Anda berbeda.
- **Baris 33, jenis statistik.** Ganti `"MEAN"` untuk statistik lain yang didukung alat. Daftarnya ada di dokumentasi alat. [CEK]

Jebakan.

- **Folder hasil tidak dibuat.** Bila `D:\Latihan\hasil_i4` belum ada, baris 32 kemungkinan besar gagal. Skrip 31.3 menunjukkan cara membuat folder lebih dulu. [CEK]
- **Ruang sementara menumpuk.** Di Pro, hanya `memory\sempadan` yang dihapus. Objek `irisan`, `kph_titik`, dan `ndvi_zonal` bertahan sampai sesi berakhir. [CEK]

- **Satuan luas.** Pembagian 10.000 benar hanya bila data berukuran meter (UTM). Pada data berkoordinat derajat, angkanya bukan hektare.
- **Band tertukar.** Bila band 1 dan 4 tertukar, NDVI berubah tanda tanpa peringatan.
- **Nama band dalam jalur.** Penulisan `Citra_KPH.tif\Band_4` mengandaikan ArcGIS mengenali band lewat jalur seperti itu. [CEK]
- **Angka antaralat sedikit berbeda.** Sempadan bergantung pada jumlah segmen yang dipakai di ujung bulat. Alat yang berbeda bisa selisih sekitar 0,01 hektare. Itu bukan galat.

Latihan variasi. Ubah sempadan menjadi 50 m. Contoh jawaban, ganti baris 14 dengan:

```python
arcpy.analysis.Buffer("Sungai.shp", r"memory\sempadan", "50 Meters", dissolve_option="ALL")
```

Dari data contoh, baris pertama sampai ketiga menjadi sempadan kira-kira `1.69`, `2.32`, dan `2.54` hektare. Baris NDVI tidak berubah. Perkiraan ini dihitung dengan OGR. [CEK]

> **Pola yang bisa dipakai ulang:** Kerangka "rantai alat": **tiap alat menulis ke ruang sementara**, **alat berikutnya membaca keluaran sebelumnya**, **kursor merangkum hasil ke layar**, **hapus yang sementara**. Untuk data baru, ganti nama data (baris 14-17), jarak (baris 14), dan nomor band (baris 29-30).

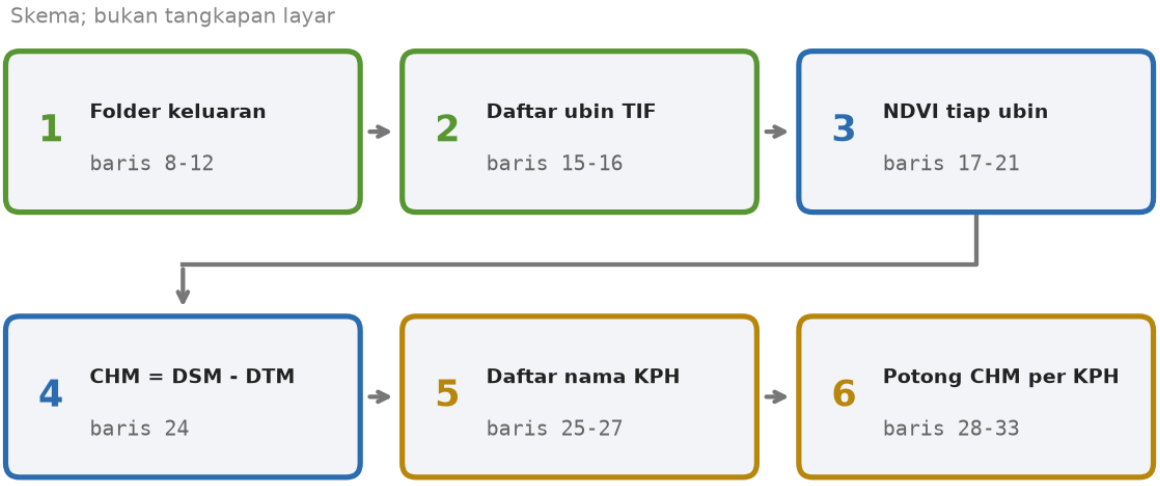
Skrip 31.3: Batch NDVI semua ubin dan potongan CHM per KPH

Berkas `b3_arcpy_pro.py` dan `b3_arcpy_arcmap.py`. Di dalam berkasnya tertulis "Skrip 3.1". Padanan PyQGIS-nya ada di Bab 21.

Tujuan. Menghitung NDVI untuk semua ubin foto drone dalam satu folder, lalu memotong peta tinggi tajuk (CHM) satu per satu untuk tiap KPH.

Bahan. Paket data `paket-i4`: folder `Ortofoto_Mentah` (tiga ubin; band 1 inframerah dekat, band 2 merah), `DSM_Drone.tif`, `DTM_Drone.tif`, dan `shp\Batas_KPH.shp`. Berbeda dari Skrip 31.2, skrip ini **membuat folder hasil sendiri**. Ekstensi Spatial Analyst harus tersedia. CHM (*canopy height model*) adalah tinggi tajuk pohon: permukaan tertinggi (DSM) dikurangi permukaan tanah (DTM).

Ilustrasi 31.4: Alur Skrip 31.3



Versi ArcGIS Pro (Python 3)

PYTHON

```
# [SKRIP 3.1-Pro: Batch: NDVI semua ubin dan potongan CHM per KPH dengan ArcPy]
# Penulis: Badar Mubarok Yogaswara
# [CEK] Cocokkan dengan versi Anda. Butuh ekstensi Spatial Analyst.
import os
import arcpy
from arcpy.sa import Raster, Float, ExtractByMask

DATA = r"D:\Latihan\paket-i4"
HASIL = r"D:\Latihan\hasil_i4"
arcpy.env.overwriteOutput = True
arcpy.CheckOutExtension("Spatial")
os.makedirs(HASIL + r"\NDVI_ubin", exist_ok=True)

# A. Semua ubin dalam satu folder (band 1 = NIR, band 2 = Merah)
arcpy.env.workspace = DATA + r"\Ortofoto_Mentah"
daftar = arcpy.ListRasters("*", "TIF")
for i, nama in enumerate(daftar, start=1):
    b1 = Float(Raster(os.path.join(DATA, "Ortofoto_Mentah", nama, "Band_1")))
    b2 = Float(Raster(os.path.join(DATA, "Ortofoto_Mentah", nama, "Band_2")))
    ((b1 - b2) / (b1 + b2)).save(os.path.join(HASIL, "NDVI_ubin", "NDVI_" + nama))
    print("%d/%d selesai: %s" % (i, len(daftar), nama))

# B. Satu putaran per KPH: pilih satu poligon, potong CHM dengannya
chm = Raster(DATA + r"\DSM_Drone.tif") - Raster(DATA + r"\DTM_Drone.tif")
batas = DATA + r"\shp\Batas_KPH.shp"
with arcpy.da.SearchCursor(batas, ["NAMA_KPH"]) as k:
    nama_kph = [b[0] for b in k]
for nama in nama_kph:
    arcpy.analysis.Select(batas, r"memory\satu_kph", "NAMA_KPH = '%s'" % nama)
    keluar = os.path.join(HASIL, "CHM_" + nama.replace(" ", "_") + ".tif")
    ExtractByMask(chm, r"memory\satu_kph").save(keluar)
    print("CHM tersimpan untuk", nama)
arcpy.CheckInExtension("Spatial")
```

Penjelasan baris demi baris. Hanya bagian yang baru dijelaskan lengkap. Baris yang mirip Skrip 30.3 dan 31.2 dirangkum.

Baris	Tulisan	Artinya
1-3	`# [SKRIP 3.1-Pro ...`	Komentar judul, penulis, dan pengingat ekstensi.
4-6	`import os`, `import arcpy`, `from arcpy.sa import ...`	Tiga perkakas. `ExtractByMask` dipakai untuk memotong raster.
8-11	`DATA`, `HASIL`, `overwriteOutput`, `CheckOutExtension`	Lokasi, izin menimpa, dan ekstensi.
12	`os.makedirs(HASIL + r"\NDVI_ubin", exist_ok=True)`	Buat folder `NDVI_ubin` beserta folder induknya. `exist_ok=True` berarti: bila sudah ada, jangan galat.
15-16	`arcpy.env.workspace = ...`, `daftar = arcpy.ListRasters("*", "TIF")`	Folder kerja diarahkan ke ubin. Lalu ambil daftar nama raster TIF.
17	`for i, nama in enumerate(daftar, start=1):`	Ulangi tiap ubin dengan nomor urut mulai dari 1.
18-19	`b1 = Float(Raster(os.path.join(DATA, "Ortofoto_Mentah", nama, "Band_1")))`	Buka band 1 dan band 2 dari ubin ini sebagai desimal.
20	`((b1 - b2) / (b1 + b2)).save(...)`	Hitung NDVI dan langsung simpan, tanpa kotak perantara. Nama keluaran: `NDVI_` ditambah nama ubin, di folder `NDVI_ubin`.
21	`print("%d/%d selesai: %s" % (i, len(daftar), nama))`	Cetak kemajuan. `%d` untuk bilangan bulat, `%s` untuk teks.
24	`chm = Raster(DATA + r"\DSM_Drone.tif") - Raster(DATA + r"\DTM_Drone.tif")`	CHM adalah DSM dikurangi DTM. Dilakukan sekali, di luar perulangan.
25-27	`with arcpy.da.SearchCursor(batas, ["NAMA_KPH"]) as k:`	Kumpulkan nama semua KPH ke dalam daftar. `[b[0] for b in k]` berarti: dari tiap baris `b`, ambil isian pertama.
28	`for nama in nama_kph:`	Ulangi untuk tiap KPH.
29	`arcpy.analysis.Select(batas, r"memory\satu_kph", "NAMA_KPH = '%s'" % nama)`	Pilih satu KPH ke ruang sementara. Klausa pilihnya berbentuk `NAMA_KPH = 'KPH Alpha'`, dengan nama KPH disisipkan oleh `%s`.
30	`keluar = os.path.join(HASIL, "CHM_" + nama.replace(" ", "_") + ".tif")`	Susun nama berkas. `replace` mengganti spasi dengan garis bawah: `CHM_KPH_Alpha.tif`.

Baris	Tulisan	Artinya
31	`ExtractByMask(chm, r"memory\satu_kph").save(keluar)`	Potong CHM dengan bentuk satu KPH, lalu simpan.
32	`print("CHM tersimpan untuk", nama)`	Cetak pesan.
33	`arcpy.CheckInExtension("Spatial")`	Kembalikan ekstensi.

Versi ArcMap 10.8 (Python 2.7)

PYTHON

```
# -*- coding: utf-8 -*-
# [SKRIP 3.1-ArcMap: Batch: NDVI semua ubin dan potongan CHM per KPH (Python 2.7)]
# Penulis: Badar Mubarok Yogaswara
# [CEK] Cocokkan dengan versi Anda. Butuh ekstensi Spatial Analyst.
import os
import arcpy
from arcpy.sa import Raster, Float, ExtractByMask

DATA = r"D:\Latihan\paket-i4"
HASIL = r"D:\Latihan\hasil_i4"
arcpy.env.overwriteOutput = True
arcpy.CheckOutExtension("Spatial")
if not os.path.exists(HASIL + r"\NDVI_ubin"):          # Python 2.7 tidak punya exist_ok
    os.makedirs(HASIL + r"\NDVI_ubin")

# A. Semua ubin dalam satu folder (band 1 = NIR, band 2 = Merah)
arcpy.env.workspace = DATA + r"\Ortofoto_Mentah"
daftar = arcpy.ListRasters("*", "TIF")
for i, nama in enumerate(daftar, 1):
    b1 = Float(Raster(os.path.join(DATA, "Ortofoto_Mentah", nama, "Band_1")))
    b2 = Float(Raster(os.path.join(DATA, "Ortofoto_Mentah", nama, "Band_2")))
    ((b1 - b2) / (b1 + b2)).save(os.path.join(HASIL, "NDVI_ubin", "NDVI_" + nama))
    print("%d/%d selesai: %s" % (i, len(daftar), nama))

# B. Satu putaran per KPH: pilih satu poligon, potong CHM dengannya
chm = Raster(DATA + r"\DSM_Drone.tif") - Raster(DATA + r"\DTM_Drone.tif")
batas = DATA + r"\shp\Batas_KPH.shp"
with arcpy.da.SearchCursor(batas, ["NAMA_KPH"]) as k:
    nama_kph = [b[0] for b in k]
for nama in nama_kph:
    arcpy.Select_analysis(batas, r"in_memory\satu_kph", "\"NAMA_KPH\" = '%s'" % nama)
    keluar = os.path.join(HASIL, "CHM_" + nama.replace(" ", "_") + ".tif")
    ExtractByMask(chm, r"in_memory\satu_kph").save(keluar)
    print("CHM tersimpan untuk %s" % nama)
arcpy.CheckInExtension("Spatial")
```

> **Bedanya di ArcMap:** Ada lima beda. **Pertama**, baris 1 menambah baris `coding`, sehingga nomor baris bergeser satu. **Kedua**, Python 2.7 tidak punya `exist_ok`, jadi folder ditulis dengan penjaga manual: `if not os.path.exists(...)` lalu `os.makedirs(...)` (baris 13-14). **Ketiga**, `enumerate(daftar, 1)` (baris 19). **Keempat**, alat bergaya lama `arcpy.Select_analysis(...)` dan ruang `in_memory\satu_kph` (baris 31 dan 33). **Kelima**, klausa pilih ditulis dengan tanda kutip ganda di sekeliling nama kolom: `"\"NAMA_KPH\" = '%s'" % nama`, dan cetakan memakai `%s` tunggal (baris 34). Penulisan kutip ganda pada nama kolom untuk Shapefile di ArcMap belum penulis cocokkan dengan dokumentasi. [CEK]

Hasil. Layar menampilkan enam baris. Folder hasil berisi tiga berkas NDVI di `NDVI_ubin` dan tiga berkas CHM. Berikut perkiraannya. Nama ubin dan KPH mengikuti paket data. [CEK]

HASIL

```
1/3 selesai: ubin_01.tif
2/3 selesai: ubin_02.tif
3/3 selesai: ubin_03.tif
CHM tersimpan untuk KPH Alpha
CHM tersimpan untuk KPH Beta
CHM tersimpan untuk KPH Gamma
```

Berkas yang terbentuk: `NDVI_ubin\NDVI_ubin_01.tif` sampai `NDVI_ubin_03.tif`, dan `CHM_KPH_Alpha.tif`, `CHM_KPH_Beta.tif`, `CHM_KPH_Gamma.tif` langsung di folder hasil. Dengan GDAL dari data contoh, rata-rata NDVI ketiga ubin kira-kira 0,471, 0,471, dan 0,467. Rata-rata tinggi tajuk per KPH kira-kira 2,60 m, 2,69 m, dan 2,56 m, sama dengan hasil skrip PyQGIS di Bab 21. [CEK] Logika Python skrip sudah diuji dengan modul ArcPy tiruan buatan sendiri, dan cetakannya sama dengan perkiraan di atas. Perilaku ArcGIS yang sebenarnya belum teruji.

Bagian yang boleh diubah. Nomor baris mengikuti versi Pro.

- **Baris 8-9, lokasi.** Ganti sesuai komputer Anda.
- **Baris 16, pola dan format.** Ganti `"*"` atau `"TIF"` untuk memilih ubin lain.
- **Baris 18-19, nomor band.** Tukar `Band_1` dan `Band_2` bila urutan band berbeda.
- **Baris 29, klausa pilih.** Ganti `NAMA_KPH` dengan kolom pembeda lain.
- **Baris 30, awalan nama keluaran.** Ganti `"CHM_"` untuk awalan lain.

Jebakan.

- **Band terbalik.** Bila band 1 sebenarnya merah, NDVI negatif dan tidak ada peringatan.
- **Kutip dalam klausa pilih.** Pro memakai `NAMA_KPH = '%s'` tanpa kutip ganda, ArcMap memakai kutip ganda di nama kolom. Dokumentasi Esri untuk `Select` memberi contoh dengan nama kolom diapit kutip ganda. Mana yang perlu untuk Shapefile bergantung pada versi. [CEK]
- **Nama KPH yang memuat tanda kutip tunggal.** Teks `'KPH D'Arc'` merusak klausa pilih, karena kutip tunggal di tengah nama dianggap penutup. [CEK]
- **Objek sementara dipakai ulang.** `memory\satu_kph` ditulis ulang tiap putaran. Itu aman karena `overwriteOutput = True`.

- **Lokasi CHM berbeda dari versi QGIS.** Di sini CHM disimpan langsung di folder hasil. Versi PyQGIS di Bab 21 memakai subfolder `CHM_per_KPH`.

Latihan variasi. Potong CHM hanya untuk KPH Beta. Contoh jawaban, tambahkan satu baris sebelum baris 28:

```python
nama_kph = [n for n in nama_kph if n == "KPH Beta"]
```

Daftar disaring sehingga hanya berisi `KPH Beta`. Layar menampilkan tiga baris kemajuan ubin, lalu `CHM tersimpan untuk KPH Beta`. Jawaban ini sudah diuji dengan modul ArcPy tiruan. [CEK]

> **Pola yang bisa dipakai ulang:** Kerangka "batch per berkas" dan "batch per fitur": **daftar dulu**, ulangi dengan `for`, **kerjakan satu per satu**, **beri nama keluaran dari nama masukan**. Untuk data baru, ganti pola daftar (baris 16), nomor band (baris 18-19), dan kolom pembeda (baris 26 dan 29).

Skrip 31.4: Menangkap galat dengan ArcPy

Berkas `b4_arcpy_pro.py` dan `b4_arcpy_arcmap.py`. Di dalam berkasnya tertulis "Skrip 4.1". Padanan PyQGIS-nya ada di Bab 22.

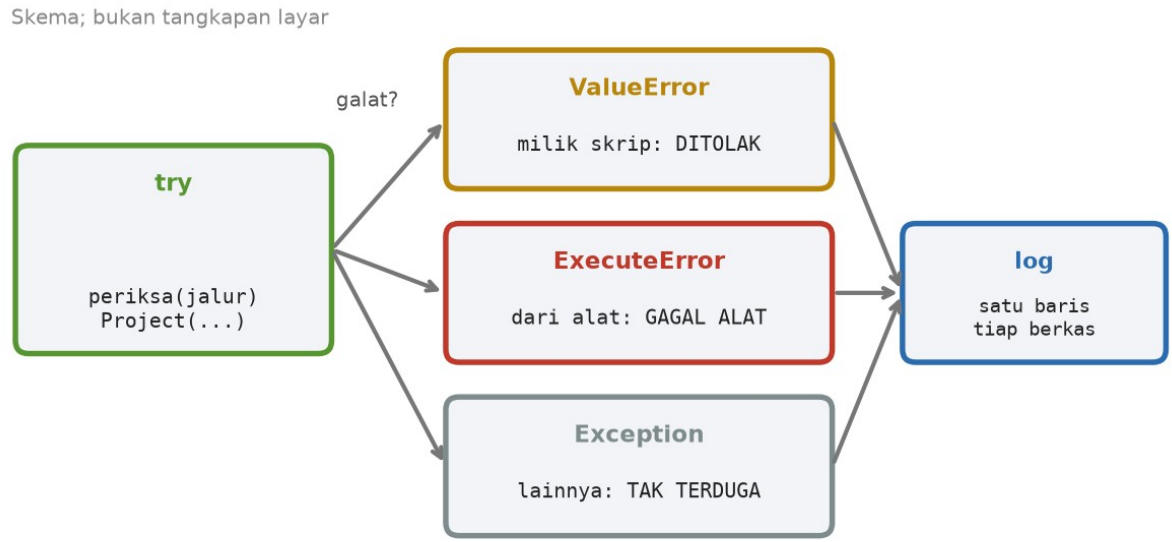
Tujuan. Memeriksa tiap kiriman Shapefile dari tim lapangan: apakah datanya ada, apakah kolom wajib lengkap, dan apakah sistem koordinatnya sudah EPSG 32749. Yang berbeda diproyeksikan, dan semua hasil dicatat dalam satu daftar log.

Bahan. Paket data `paket-i4`, folder `Kiriman_Lapangan\shp` berisi enam kiriman, `tim_a.shp` sampai `tim_f.shp`. Satu kiriman (`tim_d`) memakai sistem koordinat derajat (EPSG 4326), satu (`tim_e`) berkas rusak, dan satu (`tim_f`) tanpa kolom `Kondisi`. Skrip ini hanya **menguji**, tidak menyimpan hasil proyeksi. Tidak perlu Spatial Analyst.

Lima istilah baru, semuanya dibahas di Bab 7 dan 10:

- `def`: membuat fungsi, yaitu potongan kerja yang diberi nama.
- `raise`: sengaja memunculkan galat dengan pesan yang jelas.
- `try` dan `except`: "coba kerjakan ini; bila galat jenis tertentu, kerjakan itu".
- `arcpy.ExecuteError`: jenis galat yang dimunculkan ArcPy ketika sebuah alat gagal.
- `glob`: perkakas untuk mencari berkas dengan pola nama, misalnya semua `*.shp`.

Ilustrasi 31.5: Tiga penangkap galat



Versi ArcGIS Pro (Python 3)

PYTHON

```
# [SKRIP 4.1-Pro: Penanganan galat dengan ArcPy: periksa masukan lalu tangkap ExecuteError]
# Penulis: Badar Mubarok Yogaswara
# [CEK] Cocokkan dengan versi Anda. Kiriman lapangan berbentuk Shapefile di folder Kiriman_Lapangan\shp.
import glob
import os
import sys
import arcpy

DATA = r"D:\Latihan\paket-i4"
FIELD_WAJIB = ["Kondisi", "Tinggi_Phn", "Foto"]
log = []


def periksa(jalur):
    """Berhenti dengan pesan jelas bila masukan tidak layak. Mengembalikan kode EPSG data."""
    if not arcpy.Exists(jalur):
        raise ValueError("data tidak ada atau tidak bisa dibuka")
    nama_field = [f.name for f in arcpy.ListFields(jalur)]
    kurang = [f for f in FIELD_WAJIB if f not in nama_field]
    if kurang:
        raise ValueError("field hilang: " + ", ".join(kurang))
    return arcpy.Describe(jalur).spatialReference.factoryCode


for jalur in sorted(glob.glob(DATA + r"\Kiriman_Lapangan\shp\*.shp")):
    nama = os.path.basename(jalur)
    try:
        epsg = periksa(jalur)
        if epsg != 32749:
            arcpy.management.Project(jalur, "memory\\" + nama[:-4], arcpy.SpatialReference(32749))
            log.append((nama, "DITERIMA", "CRS %s diubah ke 32749" % epsg))
        else:
            log.append((nama, "DITERIMA", "sesuai"))
    except ValueError as e:                     # galat milik skrip kita
        log.append((nama, "DITOLAK", str(e)))
    except arcpy.ExecuteError:                  # galat dari alat geoprocessing
        log.append((nama, "GAGAL ALAT", arcpy.GetMessages(2)))
    except Exception:                           # galat Python lain
```

```
        log.append((nama, "TAK TERDUGA", str(sys.exc_info()[1])))
for baris in log:
    print("%-14s %-12s %s" % baris)
```

Penjelasan baris demi baris. Nomor baris mengikuti berkas di atas.

Baris	Tulisan	Artinya
1-3	`# [SKRIP 4.1-Pro ...`	Komentar judul, penulis, dan pengingat bahwa kiriman berbentuk Shapefile di `Kiriman_Lapangan\shp`.
4-7	`import glob, os, sys, arcpy`	Empat perkakas. `sys` dipakai untuk membaca pesan galat.
9-11	`DATA, FIELD_WAJIB, log = []`	Lokasi data, daftar tiga kolom wajib, dan daftar kosong untuk catatan.
14-15	`def periksa(jalur):` dan teks di bawahnya	Buat fungsi `periksa`. Teks tiga kutip di bawahnya adalah penjelasan fungsi.
16-17	`if not arcpy.Exists(jalur): raise ValueError(...)`	Bila data tidak ada atau tidak bisa dibuka, berhenti dengan pesan jelas.
18	`nama_field = [f.name for f in arcpy.ListFields(jalur)]`	Kumpulkan nama semua kolom ke dalam daftar.
19-21	`kurang = [f for f in FIELD_WAJIB if f not in nama_field]`	Cari kolom wajib yang tidak ada. Bila ada, berhenti dengan pesan yang memuat namanya, disambung dengan `", ".join(...)`.
22	`return arcpy.Describe(jalur).spatialReference.factoryCode`	Kembalikan kode sistem koordinat data, misalnya 32749.
25	`for jalur in sorted(glob.glob(DATA + r"\Kiriman_Lapangan\shp\*.shp")):`	Ulangi untuk tiap berkas `.shp` di folder itu, diurutkan menurut nama.
26	`nama = os.path.basename(jalur)`	Ambil nama berkas saja, tanpa folder.
27-28	`try:` dan `epsg = periksa(jalur)`	Mulai blok "coba". Jalankan pemeriksaan.
29-31	`if epsg != 32749:`	Bila sistem koordinat berbeda, proyeksikan ke `memory\` dengan nama berkas tanpa `.shp` (`nama[:-4]` membuang empat huruf terakhir). Catat "CRS ... diubah ke 32749".
32-33	`else:`	Bila sama, catat "sesuai".
34-35	`except ValueError as e:`	Tangkap galat milik skrip (dari `raise` di fungsi). Catat DITOLAK beserta pesannya.
36-37	`except arcpy.ExecuteError:`	Tangkap galat dari alat ArcGIS. Catat GAGAL ALAT dengan `arcpy.GetMessages(2)`, yaitu pesan tingkat galat dari alat itu.
38-39	`except Exception:`	Tangkap semua galat Python lain. Catat TAK TERDUGA dengan pesan dari `sys.exc_info()[1]`.

Baris	Tulisan	Artinya
40-41	`for baris in log:` `print("%-14s %-12s %s" % baris)`	Cetak log. Tiga bagian tiap baris: teks rata kiri selebar 14 karakter, teks rata kiri selebar 12, lalu sisanya.

Dokumentasi Esri menyebut: alat yang gagal memunculkan `arcpy.ExecuteError`, `GetMessages(2)` mengambil pesan berderajat galat, dan blok `try` dengan beberapa `except` dipakai untuk membedakan galat alat dari galat Python lain. Ketiga hal itu cocok dengan baris 36-39. [CEK]

Versi ArcMap 10.8 (Python 2.7)

PYTHON

```
# -*- coding: utf-8 -*-
# [SKRIP 4.1-ArcMap: Penanganan galat dengan ArcPy (Python 2.7)]
# Penulis: Badar Mubarok Yogaswara
# [CEK] Cocokkan dengan versi Anda. Kiriman lapangan diandaikan berbentuk Shapefile di folder
Kiriman_Lapangan\shp.
import glob
import os
import sys
import arcpy

DATA = r"D:\Latihan\paket-i4"
FIELD_WAJIB = ["Kondisi", "Tinggi_Phn", "Foto"]
log = []


def periksa(jalur):
    """Berhenti dengan pesan jelas bila masukan tidak layak. Mengembalikan kode EPSG data."""
    if not arcpy.Exists(jalur):
        raise ValueError("data tidak ada atau tidak bisa dibuka")
    nama_field = [f.name for f in arcpy.ListFields(jalur)]
    kurang = [f for f in FIELD_WAJIB if f not in nama_field]
    if kurang:
        raise ValueError("field hilang: " + ", ".join(kurang))
    return arcpy.Describe(jalur).spatialReference.factoryCode


for jalur in sorted(glob.glob(DATA + r"\Kiriman_Lapangan\shp\*.shp")):
    nama = os.path.basename(jalur)
    try:
        epsg = periksa(jalur)
        if epsg != 32749:
            arcpy.Project_management(jalur, "in_memory\\" + nama[:-4], arcpy.SpatialReference(32749))
            log.append((nama, "DITERIMA", "CRS %s diubah ke 32749" % epsg))
        else:
            log.append((nama, "DITERIMA", "sesuai"))
    except ValueError as e:                     # galat milik skrip kita
        log.append((nama, "DITOLAK", str(e)))
    except arcpy.ExecuteError:                  # galat dari alat geoprocessing
        log.append((nama, "GAGAL ALAT", arcpy.GetMessages(2)))
    except Exception:                           # galat Python lain
        log.append((nama, "TAK TERDUGA", str(sys.exc_info()[1])))
for baris in log:
    print("%-14s %-12s %s" % baris)
```

> **Bedanya di ArcMap:** Hanya dua beda kecil. **Pertama**, baris 1 menambah `# -*- coding: utf-8 -*-`, sehingga nomor baris bergeser satu. **Kedua**, proyeksi memakai nama alat gaya lama dan ruang `in_memory`: `arcpy.Project_management(jalur, "in_memory\\" + nama[:-4], ...)` (baris 31). Dua garis miring terbalik `\\` di dalam teks biasa dibaca sebagai satu garis miring. Komentar baris 4 menyebut kiriman "diandaikan" berbentuk Shapefile, sedangkan versi Pro menyatakannya lugas. Sisanya sama persis: `except ValueError as e`, `arcpy.ExecuteError`, `sys.exc_info()[1]`, dan cetakan dengan `%`. Semuanya sah di Python 2.7.

Hasil. Layar menampilkan satu baris untuk tiap kiriman. Berikut perkiraannya. Dari enam kiriman, empat diterima dan dua ditolak, sama dengan hasil skrip PyQGIS di Bab 22. [CEK]

```
HASIL
tim_a.shp      DITERIMA     sesuai
tim_b.shp      DITERIMA     sesuai
tim_c.shp      DITERIMA     sesuai
tim_d.shp      DITERIMA     CRS 4326 diubah ke 32749
tim_e.shp      DITOLAK      data tidak ada atau tidak bisa dibuka
tim_f.shp      DITOLAK      field hilang: Kondisi
```

Perkiraan ini disusun dengan membaca sistem koordinat dan kolom tiap Shapefile lewat OGR, lalu menjalankan logika Python skrip dengan modul ArcPy tiruan. Dua hal bisa berbeda di ArcGIS sungguhan. Pertama, kode sistem koordinat yang dibaca dari berkas `.prj`. Kedua, apakah `arcpy.Exists` menganggap `tim_e.shp` (berkas rusak) ada atau tidak. Bila `Exists` mengatakan ada, kiriman itu tertangkap di cabang lain. [CEK]

Bagian yang boleh diubah. Nomor baris mengikuti versi Pro.

- **Baris 9, lokasi data.** Ganti sesuai komputer Anda.
- **Baris 10, kolom wajib.** Tambah atau kurangi nama kolom di daftar `FIELD_WAJIB`.
- **Baris 25, folder dan pola.** Ganti `Kiriman_Lapangan\shp` dan `*.shp` untuk folder atau format lain.
- **Baris 29-31, kode tujuan.** Ganti `32749` (tiga tempat, termasuk teks di baris 31) bila sistem koordinat tujuan Anda lain.

Jebakan.

- Urutan `except` penting. Python memeriksa dari atas. Bila `except Exception:` diletakkan **paling atas**, ia menangkap semuanya, dan cabang `ValueError` serta `ExecuteError` tidak pernah dipakai. Semua kiriman akan berlabel TAK TERDUGA. Perilaku ini sudah diuji di Python 3.12. Tulis penangkap yang paling khusus di atas, yang paling umum di bawah.
- **Hasil proyeksi tidak disimpan.** Keluarannya ke `memory\` dan tidak dipakai lagi. Skrip hanya menguji kelayakan. Berbeda dari versi PyQGIS di Bab 22, tidak ada penggabungan data dan tidak ada pemeriksaan titik di luar KPH.
- **Data tanpa sistem koordinat.** Bila `factoryCode` bernilai 0, baris 29 memicu `Project` yang gagal. Kasusnya tertangkap sebagai GAGAL ALAT. [CEK]
- **Pesan galat tidak diperiksa lebih jauh.** Cabang terakhir hanya mencatat teksnya. Untuk menyelidiki, cetak seluruh isi galat.

- **Kolom wajib dibaca dari Shapefile.** Nama kolom Shapefile maksimal 10 karakter. `Tinggi_Phn` pas 10 karakter, jadi aman. Nama yang lebih panjang terpotong. [CEK]

Latihan variasi. Tambahkan kolom wajib `Catatan`, yang tidak ada di kiriman mana pun. Contoh jawaban, ganti baris 10 dengan:

```python
FIELD_WAJIB = ["Kondisi", "Tinggi_Phn", "Foto", "Catatan"]
```

Dari data contoh, keempat kiriman yang terbaca (`tim_a` sampai `tim_d`) berubah menjadi DITOLAK dengan keterangan `field hilang: Catatan`. Kiriman `tim_f` ditolak dengan keterangan `field hilang: Kondisi, Catatan`. Kiriman `tim_e` tetap ditolak karena datanya tidak terbuka. Jawaban ini sudah diuji dengan modul ArcPy tiruan. [CEK]

> **Pola yang bisa dipakai ulang:** Kerangka "periksa dulu, tangkap galat bertingkat": **fungsi pemeriksa yang melempar galat berpesan jelas**, `try` dengan penangkap dari yang paling khusus ke yang paling umum, **catat hasil tiap berkas dalam satu log**. Untuk data baru, ganti folder dan pola (baris 25), kolom wajib (baris 10), dan kode tujuan (baris 29-31).

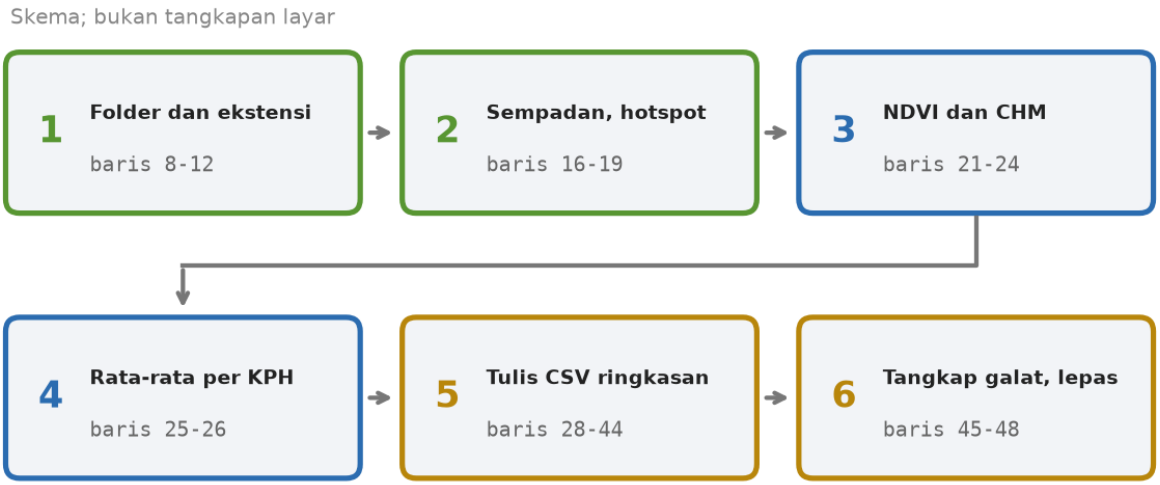
Skrip 31.5: Proyek mini terpadu, ringkasan per KPH ke CSV

Berkas `b7_arcpy_pro.py` dan `b7_arcpy_arcmap.py`. Di dalam berkasnya tertulis "Skrip 7.1". Padanan PyQGIS-nya ada di Bab 23.

Tujuan. Menyatukan alat-alat sebelumnya menjadi satu proyek kecil: menghitung luas, sempadan sungai, titik panas, NDVI rata-rata, dan tinggi tajuk rata-rata per KPH, lalu menulisnya ke berkas CSV.

Bahan. Paket data `paket-i4`: folder `shp`, `Citra_KPH.tif`, `DSM_Drone.tif`, dan `DTM_Drone.tif`. Siapkan folder `D:\Latihan\hasil_i4` lebih dulu: **skrip tidak membuatnya**. Ekstensi Spatial Analyst harus tersedia. Skrip ini memakai Shapefile, supaya sejajar dengan bagian QGIS, menurut komentar di baris 3.

Ilustrasi 31.6: Alur Skrip 31.5



Versi ArcGIS Pro (Python 3)

PYTHON

```
# [SKRIP 7.1-Pro: Proyek mini terpadu dengan ArcPy: ringkasan per KPH]
# Penulis: Badar Mubarok Yogaswara
# [CEK] Cocokkan dengan versi Anda. Butuh Spatial Analyst. Data Shapefile dipakai agar sejajar dengan Bagian
QGIS.
import csv
import arcpy
from arcpy.sa import Raster, Float, ZonalStatisticsAsTable

DATA = r"D:\Latihan\paket-i4"
HASIL = r"D:\Latihan\hasil_i4"
arcpy.env.workspace = DATA + r"\shp"
arcpy.env.overwriteOutput = True
arcpy.CheckOutExtension("Spatial")

try:
    # I1: sempadan 30 m dan titik panas
    arcpy.analysis.Buffer("Sungai.shp", r"memory\sem", "30 Meters", dissolve_option="ALL")
    arcpy.analysis.Intersect(["Batas_KPH.shp", r"memory\sem"], r"memory\iris")
    arcpy.analysis.SpatialJoin("Batas_KPH.shp", "Hotspot_Nasional.shp", r"memory\ringkas",
                               "JOIN_ONE_TO_ONE", "KEEP_ALL", match_option="INTERSECT")
    # I2: NDVI dan CHM, lalu rata-rata per KPH
    nir = Float(Raster(DATA + r"\Citra_KPH.tif\Band_4"))
    merah = Float(Raster(DATA + r"\Citra_KPH.tif\Band_1"))
    ndvi = (nir - merah) / (nir + merah)
    chm = Raster(DATA + r"\DSM_Drone.tif") - Raster(DATA + r"\DTM_Drone.tif")
    ZonalStatisticsAsTable("Batas_KPH.shp", "NAMA_KPH", ndvi, r"memory\z_ndvi", "DATA", "MEAN")
    ZonalStatisticsAsTable("Batas_KPH.shp", "NAMA_KPH", chm, r"memory\z_chm", "DATA", "MEAN")

    def tabel(jalur, kolom):
        with arcpy.da.SearchCursor(jalur, ["NAMA_KPH", kolom]) as k:
            return {n: v for n, v in k}

    sempadan = {}
    with arcpy.da.SearchCursor(r"memory\iris", ["NAMA_KPH", "SHAPE@AREA"]) as k:
        for n, luas in k:
            sempadan[n] = sempadan.get(n, 0) + luas / 10000
    n_rata, c_rata = tabel(r"memory\z_ndvi", "MEAN"), tabel(r"memory\z_chm", "MEAN")
    with open(HASIL + r"\ringkasan_kph_pro.csv", "w", newline="") as f:
        w = csv.writer(f)
        w.writerow(["NAMA_KPH", "LUAS_HA", "SEMPADAN_HA", "HOTSPOT", "NDVI_RATA", "CHM_RATA_M"])
        with arcpy.da.SearchCursor(r"memory\ringkas", ["NAMA_KPH", "SHAPE@AREA", "Join_Count"]) as k:
            for n, luas, jml in k:
                w.writerow([n, round(luas / 10000, 2), round(sempadan.get(n, 0), 2), jml,
                            round(n_rata[n], 3), round(c_rata[n], 2)])
    print("Ringkasan tersimpan.")
except arcpy.ExecuteError:
    print("Alat gagal:", arcpy.GetMessages(2))
finally:
    arcpy.CheckInExtension("Spatial")
```

Penjelasan baris demi baris. Bagian alat dan raster sudah Anda kenal di Skrip 31.2. Yang baru adalah `try/finally`, fungsi di dalam `try`, dan penulisan CSV.

Baris	Tulisan	Artinya
1-3	`# [SKRIP 7.1-Pro ...`	Komentar judul, penulis, dan pengingat bahwa skrip butuh Spatial Analyst.

Baris	Tulisan	Artinya
4-6	`import csv`, `import arcpy`, `from arcpy.sa import ...`	Tiga perkakas: `csv` untuk menulis tabel teks.
8-12	`DATA`, `HASIL`, folder kerja, `overwriteOutput`, ekstensi	Persiapan, sama seperti Skrip 31.2.
14	`try:`	Mulai blok yang dilindungi. Bila ada alat gagal, loncat ke baris 45.
16-19	`arcpy.analysis.Buffer(...)`, `Intersect(...)`, `SpatialJoin(...)`	Sempadan 30 m, irisan dengan KPH, dan gabungan titik panas, semuanya ke `memory\`.
21-24	`nir`, `merah`, `ndvi`, `chm`	NDVI dari band 4 dan band 1, serta CHM dari DSM dikurangi DTM.
25-26	`ZonalStatisticsAsTable(...)` dua kali	Rata-rata NDVI dan rata-rata CHM per KPH, ke dua tabel sementara.
28-30	`def tabel(jalur, kolom):`	Fungsi pembantu. Ia membaca tabel dan mengembalikan kamus `{nama KPH: nilai}`. Bentuk `{n: v for n, v in k}` disebut *dictionary comprehension*: kamus yang dirakit lewat satu baris perulangan.
32-35	`sempadan = {}` dan kursor irisan	Jumlahkan luas sempadan per KPH, seperti Skrip 31.2.
36	`n_rata, c_rata = tabel(...), tabel(...)`	Dua kamus sekaligus: rata-rata NDVI dan rata-rata CHM. Tanda koma di kiri dan kanan membuat dua nilai terisi bersamaan.
37-39	`with open(HASIL + r"\ringkasan_kph_pro.csv", "w", newline="") as f:`	Buka berkas CSV untuk ditulis. `csv.writer(f)` menyiapkan penulis, lalu `writerow([...])` menulis baris judul.
40-43	kursor `memory\ringkas` dan `w.writerow([...])`	Untuk tiap KPH, tulis satu baris: nama, luas (hektare), sempadan (hektare), titik panas, NDVI rata-rata, CHM rata-rata. Fungsi `round` membulatkan angka.
44	`print("Ringkasan tersimpan.")`	Cetak pesan sukses.
45-46	`except arcpy.ExecuteError:`	Bila alat gagal, cetak pesan galat alat itu.
47-48	`finally:`	Bagian ini **selalu** dijalankan, berhasil atau galat. Ekstensi dikembalikan di sini.

Dokumentasi Esri mencatat pola `try/except/finally` untuk galat alat, dan memberi contoh `arcpy.AddError(arcpy.GetMessages(2))` pada cabang `except`. Skrip ini mencetak pesannya, bukan memakai `AddError`. [CEK]

Versi ArcMap 10.8 (Python 2.7)

```python
# -*- coding: utf-8 -*-
# [SKRIP 7.1-ArcMap: Proyek mini terpadu dengan ArcPy: ringkasan per KPH (Python 2.7)]
# Penulis: Badar Mubarok Yogaswara
# [CEK] Cocokkan dengan versi Anda. Butuh Spatial Analyst.
import csv
import arcpy
```

```
from arcpy.sa import Raster, Float, ZonalStatisticsAsTable

DATA = r"D:\Latihan\paket-i4"
HASIL = r"D:\Latihan\hasil_i4"
arcpy.env.workspace = DATA + r"\shp"
arcpy.env.overwriteOutput = True
arcpy.CheckOutExtension("Spatial")

try:
    # I1: sempadan 30 m dan titik panas
    arcpy.Buffer_analysis("Sungai.shp", r"in_memory\sem", "30 Meters", dissolve_option="ALL")
    arcpy.Intersect_analysis(["Batas_KPH.shp", r"in_memory\sem"], r"in_memory\iris")
    arcpy.SpatialJoin_analysis("Batas_KPH.shp", "Hotspot_Nasional.shp", r"in_memory\ringkas",
                               "JOIN_ONE_TO_ONE", "KEEP_ALL", match_option="INTERSECT")
    # I2: NDVI dan CHM, lalu rata-rata per KPH
    nir = Float(Raster(DATA + r"\Citra_KPH.tif\Band_4"))
    merah = Float(Raster(DATA + r"\Citra_KPH.tif\Band_1"))
    ndvi = (nir - merah) / (nir + merah)
    chm = Raster(DATA + r"\DSM_Drone.tif") - Raster(DATA + r"\DTM_Drone.tif")
    ZonalStatisticsAsTable("Batas_KPH.shp", "NAMA_KPH", ndvi, r"in_memory\z_ndvi", "DATA", "MEAN")
    ZonalStatisticsAsTable("Batas_KPH.shp", "NAMA_KPH", chm, r"in_memory\z_chm", "DATA", "MEAN")

    def tabel(jalur, kolom):
        with arcpy.da.SearchCursor(jalur, ["NAMA_KPH", kolom]) as k:
            return dict((n, v) for n, v in k)

    sempadan = {}
    with arcpy.da.SearchCursor(r"in_memory\iris", ["NAMA_KPH", "SHAPE@AREA"]) as k:
        for n, luas in k:
            sempadan[n] = sempadan.get(n, 0) + luas / 10000.0
    n_rata, c_rata = tabel(r"in_memory\z_ndvi", "MEAN"), tabel(r"in_memory\z_chm", "MEAN")
    with open(HASIL + r"\ringkasan_kph_arcmap.csv", "wb") as f:      # Python 2: mode "wb" untuk csv
        w = csv.writer(f)
        w.writerow(["NAMA_KPH", "LUAS_HA", "SEMPADAN_HA", "HOTSPOT", "NDVI_RATA", "CHM_RATA_M"])
        with arcpy.da.SearchCursor(r"in_memory\ringkas", ["NAMA_KPH", "SHAPE@AREA", "Join_Count"]) as k:
            for n, luas, jml in k:
                w.writerow([n, round(luas / 10000.0, 2), round(sempadan.get(n, 0), 2), jml,
                            round(n_rata[n], 3), round(c_rata[n], 2)])
    print("Ringkasan tersimpan.")
except arcpy.ExecuteError:
    print("Alat gagal: %s" % arcpy.GetMessages(2))
finally:
    arcpy.CheckInExtension("Spatial")
```

Bedanya di ArcMap: Ada enam beda. **Pertama**, baris 1 menambah `coding`, sehingga nomor baris bergeser satu. **Kedua**, nama alat gaya lama (`Buffer_analysis`, `Intersect_analysis`, `SpatialJoin_analysis`) dan ruang `in_memory\...`. **Ketiga**, pembagian `luas / 10000.0` (baris 36 dan 43). **Keempat**, kamus dirakit dengan `dict((n, v) for n, v in k)` (baris 31) sebagai ganti `{n: v for n, v in k}`. Bentuk lama ini masih sah dan hasilnya sama. **Kelima**, berkas CSV dibuka dengan mode `"wb"` (baris 38), karena modul `csv` di Python 2 menulis byte. Di Python 3, mode `"w"` dengan `newline=""` yang dipakai. **Keenam**, nama berkas `ringkasan_kph_arcmap.csv`, dan pesan galat dicetak dengan `"Alat gagal: %s" % arcpy.GetMessages(2)` (baris 47). Tidak ada `Delete_management("in_memory")` di skrip ini, berbeda dari Skrip 31.2.

Hasil. Layar menampilkan satu baris, dan berkas CSV terbentuk di folder hasil. Berikut perkiraannya. Angka dihitung dengan GDAL dan OGR dari data contoh, dan sama dengan hasil skrip PyQGIS di Bab 23. Isi CSV tampak seperti ini, dengan angka ditulis apa adanya oleh `csv.writer` tanpa nol di belakang. [CEK]

HASIL

```
Ringkasan tersimpan.
```

HASIL

```
NAMA_KPH,LUAS_HA,SEMPADAN_HA,HOTSPOT,NDVI_RATA,CHM_RATA_M
KPH Alpha,4.0,1.18,2,0.604,2.6
KPH Beta,4.0,1.39,5,0.431,2.69
KPH Gamma,8.0,1.5,5,0.329,2.56
```

Ringkasan ArcPy punya enam kolom. Ringkasan di Bab 23 punya satu kolom lebih banyak, yaitu `TITIK_SURVEI`. Logika Python skrip sudah diuji dengan modul ArcPy tiruan buatan sendiri, dan isi CSV di atas adalah keluarannya. Perilaku alat ArcGIS yang sebenarnya belum teruji.

Bagian yang boleh diubah. Nomor baris mengikuti versi Pro.

- **Baris 8-9, lokasi.** Ganti sesuai komputer Anda.
- **Baris 16, jarak sempadan.** Ganti `"30 Meters"`.
- **Baris 37, nama berkas CSV.** Ganti `ringkasan_kph_pro.csv`.
- **Baris 39, judul kolom.** Ubah daftar judul, lalu ubah isi baris 42-43 agar urutannya sama.

Jebakan.

- **Folder hasil tidak dibuat.** Bila `D:\Latihan\hasil_i4` belum ada, penulisan CSV gagal. [CEK]
- **Nama KPH tidak ada di tabel zonal.** Baris 43 memakai `n_rata[n]` tanpa `.get`. Nama yang tidak ada memicu galat `KeyError`, misalnya `KeyError: 'KPH Delta'`. Galat ini tidak ditangkap oleh `except arcpy.ExecuteError`, jadi skrip berhenti dengan pesan galat Python. Pesan `KeyError` sudah diuji pada kamus biasa di Python 3.
- **Penyandian CSV.** Berkas ditulis tanpa `encoding`. Di Windows, Python memakai penyandian bawaan komputer. Nama KPH di contoh hanya huruf ASCII, jadi aman. [CEK]
- `newline=""` di Python 3. Dokumentasi modul `csv` menganjurkan pembukaan dengan `newline=""` agar tidak muncul baris kosong ganda di Windows. [CEK]
- **Pembagian dengan nol pada latihan.** Bila luas sebuah KPH 0, hitungan persen gagal.

Latihan variasi. Tambahkan kolom `SEMPADAN_PERSEN`, yaitu persen luas sempadan terhadap luas KPH. Contoh jawaban: ganti baris 39 sampai 43 dengan:

PYTHON

```
        w.writerow(["NAMA_KPH", "LUAS_HA", "SEMPADAN_HA", "HOTSPOT", "NDVI_RATA", "CHM_RATA_M",
"SEMPADAN_PERSEN"])
        with arcpy.da.SearchCursor(r"memory\ringkas", ["NAMA_KPH", "SHAPE@AREA", "Join_Count"]) as k:
            for n, luas, jml in k:
                w.writerow([n, round(luas / 10000, 2), round(sempadan.get(n, 0), 2), jml,
```

```
                                round(n_rata[n], 3), round(c_rata[n], 2),
                                round(100 * sempadan.get(n, 0) / (luas / 10000), 1)])
```

Dari data contoh, kolom baru bernilai kira-kira `29.4`, `34.7`, dan `18.7`. Skrip PyQGIS di Bab 21 menampilkan 34.6 untuk KPH Beta. Selisih 0,1 berasal dari perbedaan jumlah segmen pada sempadan di dua alat. Jawaban ini sudah diuji dengan modul ArcPy tiruan. [CEK]

> **Pola yang bisa dipakai ulang:** Kerangka "proyek mini terpadu": `try` membungkus semua alat, **fungsi pembantu mengubah tabel menjadi kamus**, **satu loop menulis satu baris CSV per KPH**, `finally` mengembalikan ekstensi. Untuk data baru, ganti lokasi (baris 8-10), alat yang dirantai (baris 16-26), dan judul kolom (baris 39-43).

Cek paham

1. Mengapa skrip ArcMap mencetak dengan satu teks hasil rakitan `%`, bukan beberapa isian dipisah koma?
2. Apa fungsi `arcpy.env.workspace`?
3. Mengapa bagian 4 Skrip 31.1 gagal bila dijalankan dari terminal biasa?
4. Apa beda `memory\sempadan` dan `in_memory\sempadan`?
5. Mengapa `except arcpy.ExecuteError` harus ditulis di atas `except Exception`?
6. Apa fungsi blok `finally` di Skrip 31.5?

Jawaban.

1. Karena `print` pada Python 2.7 berperilaku lain bila diberi beberapa isian. Satu teks hasil rakitan aman di kedua versi.
2. Menetapkan folder kerja. Nama data yang ditulis tanpa lokasi dicari di folder itu.
3. Karena `CURRENT` berarti proyek atau dokumen peta yang sedang terbuka. Di terminal biasa tidak ada proyek yang terbuka.
4. Keduanya ruang sementara di memori komputer. `memory` dipakai di ArcGIS Pro, sedangkan `in_memory` adalah nama gaya lama yang dipakai di ArcMap.
5. Python memeriksa penangkap dari atas. `Exception` mencakup semua jenis galat, jadi bila ditulis lebih dulu, penangkap yang lebih khusus tidak pernah dipakai.
6. Menjalankan bagian itu selalu, berhasil atau galat. Di skrip itu, ia mengembalikan ekstensi Spatial Analyst.

Kesalahan umum

- Lupa huruf `r` pada lokasi data. Perbaikan: tulis `r"D:\Latihan\paket-i4"`.
- **Memakai gaya Python 3 di ArcMap.** Perbaikan: ikuti versi ArcMap di bab ini, terutama untuk `print`, `exist_ok`, dan modul `arcpy.mapping`.
- **Menganggap luas selalu hektare.** Perbaikan: cek satuan peta lewat sistem koordinat data.
- **Lupa membuat folder hasil.** Skrip 31.2 dan 31.5 tidak membuatnya. Perbaikan: buat foldernya dulu, atau tambahkan `os.makedirs` seperti di Skrip 31.3.

- Menaruh `except Exception` di paling atas. Perbaikan: urutkan dari yang paling khusus ke yang paling umum.

Ringkasan dan latihan

- **ArcPy** adalah perkakas Python untuk ArcGIS. ArcGIS Pro memakai Python 3, ArcMap 10.8 memakai Python 2.7.
- Pola skrip: lokasi data, sifat data, kursor per baris, lalu peta. Untuk rantai alat: tiap alat menulis ke ruang sementara.
- Lima skrip l4 di bab ini: membaca data dan proyek, rantai alat, batch, penanganan galat, dan proyek terpadu.
- Beda ArcMap yang utama: `print` dengan `%`, `10000.0`, nama alat gaya `_analysis`, `in_memory`, `arcpy.mapping`, `exist_ok` yang tidak ada, dan mode `"wb"` untuk CSV.
- Skrip bagian ini hanya bisa dijalankan di Windows, dan bertanda **[CEK]**.

Latihan. Ganti data pada Skrip 31.1 menjadi `Sungai.shp`. Cetak jumlah fitur dan daftar kolomnya saja. Bagian mana dari skrip yang boleh Anda hapus? Lalu, di Skrip 31.4, tambahkan catatan jumlah kiriman yang diterima di akhir log. Cukup satu baris `print`.

Tabel perbandingan: QGIS dan ArcPy

Pekerjaan	QGIS (PyQGIS)	ArcGIS (ArcPy)
Membaca tiap baris	`layer.getFeatures()`	`arcpy.da.SearchCursor(...)`
Mengambil isi kolom	`fitur["NAMA_KPH"]`	isian pada baris kursor
Mengakses peta yang terbuka	`QgsProject.instance()`	`arcpy.mp.ArcGISProject("CURRENT")` (Pro)
Memanggil alat	`processing.run("native:buffer", {...})`	`arcpy.analysis.Buffer(...)`
Keluaran sementara	`"TEMPORARY_OUTPUT"`	`memory\nama` (Pro) atau `in_memory\nama` (ArcMap)
Jumlah titik per poligon	alat `native:countpointsinpolygon`	`SpatialJoin` lalu kolom `Join_Count`
NDVI	ekspresi dengan `"Citra_KPH@4"`	`Float(Raster(...\Band_4))` lalu rumus
Menangkap galat	`except Exception as e`	`except arcpy.ExecuteError` dan `arcpy.GetMessages(2)`

BAB 32: Dari QGIS ke ArcPy dan Kembali: Pola, Padanan, dan Cara Menerjemahkan

> **Berdasarkan Seri I4 dan M1:** Bab ini tidak memperkenalkan skrip baru. Ia merangkum pola dari skrip yang sudah Anda kenal: skrip PyQGIS Buku I4 (Bab 14, 21, 22, dan 23), skrip PyQGIS Buku M1 (Bab 24), serta pasangan ArcPy-nya di Bab 30 dan 31. Semua padanan di bab ini dibaca dari skrip-skrip itu. Dokumentasi Esri dibuka untuk mencocokkan nama alat, urutan isian, dan perilaku modul peta.

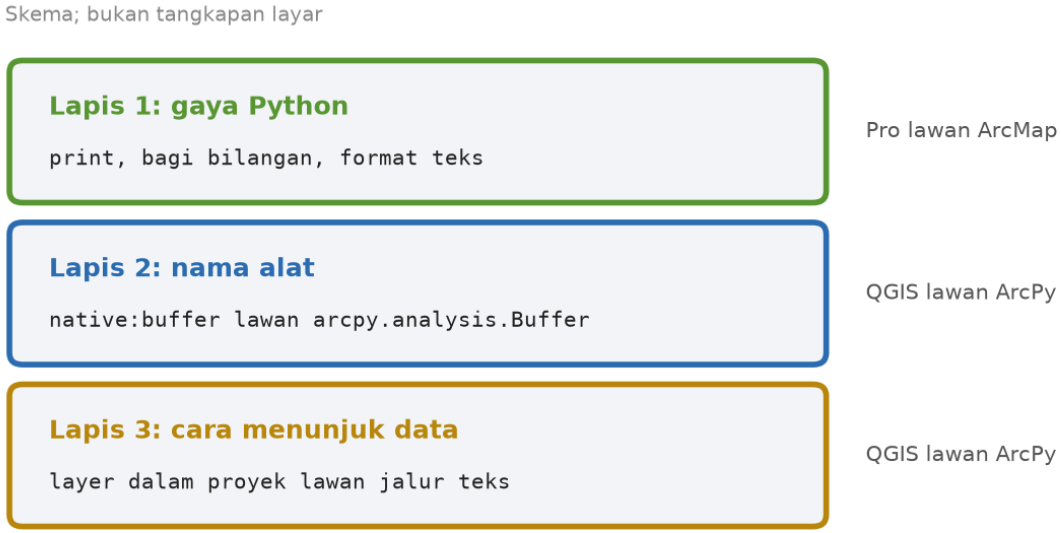
Studi kasus: "Skrip ini jalan di QGIS. Bisakah dibuat versi ArcGIS-nya?"

Anda sudah punya skrip PyQGIS yang menghitung luas, sempadan sungai, dan titik panas per KPH. Kantor cabang meminta versi yang jalan di ArcGIS. Anda tidak perlu menulis dari nol. Pekerjaannya sama, hanya "bahasa alat"-nya yang berbeda. Kabar baiknya: setelah Bab 30 dan 31, Anda sudah punya kamus kecil di kepala. Bab ini merapikan kamus itu menjadi tabel, lalu mengajarkan **cara menerjemahkan** dengan langkah yang bisa diulang.

Konsep: menerjemahkan skrip dalam tiga kalimat

Menerjemahkan skrip itu seperti menerjemahkan resep dari bahasa lain. Bahan dan urutan masaknya sama, tetapi nama bumbu dan alat dapurnya berbeda. Ada tiga lapis yang perlu Anda terjemahkan: gaya Python (Pro lawan ArcMap), nama alat (QGIS lawan ArcPy), dan cara menunjuk data (layer dalam proyek lawan jalur teks).

Ilustrasi 32.1: Tiga lapis beda saat menerjemahkan



Empat istilah baru yang perlu Anda kenal sekarang:

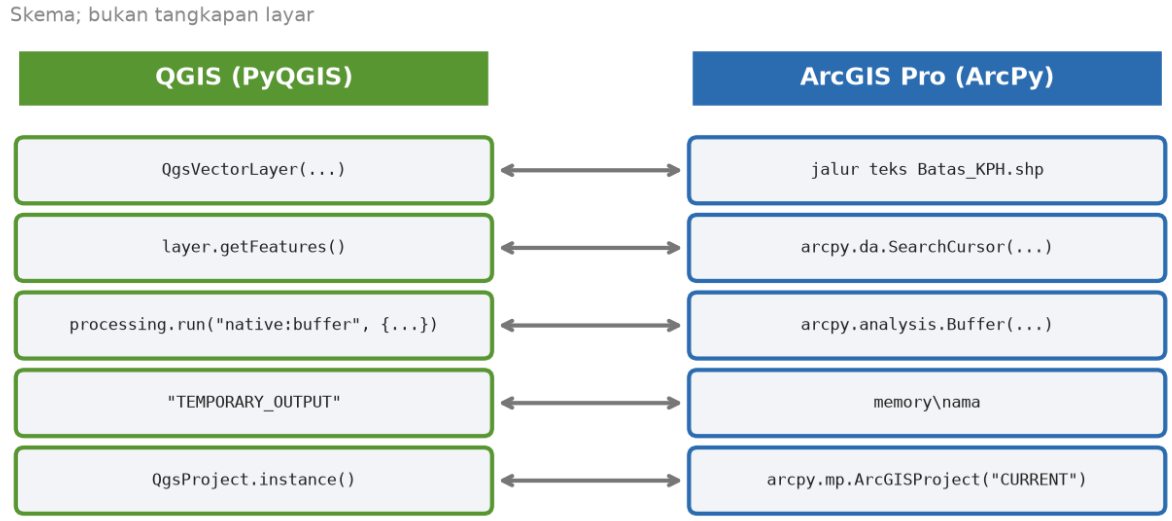
- **Padanan**: dua penulisan di dua alat yang mengerjakan hal yang sama.

- **Alat** (*tool*): satu pekerjaan siap pakai, misalnya penyangga. Di QGIS dipanggil lewat `processing.run`, di ArcGIS lewat `arcpy.<kotak alat>.<Alat>`.
- **Keluaran sementara**: hasil antara yang tidak ditulis ke berkas. Di QGIS `"TEMPORARY_OUTPUT"`, di ArcGIS `memory\nama`.
- **Angka acuan**: hasil yang Anda ketahui benar dari data contoh, misalnya luas KPH Alpha 4,0 hektare. Angka ini dipakai untuk menguji terjemahan Anda.

Tabel padanan: membaca data dan atribut

Tabel ini diambil dari Skrip 31.1 dan skrip PyQGIS `b1_01` dan `b1_02` (Bab 14). Kolom ArcPy memakai penulisan ArcGIS Pro.

Ilustrasi 32.2: Lima pasangan konsep QGIS dan ArcPy



Pekerjaan	PyQGIS	ArcPy (Pro)
Membuka layer vektor	`QgsVectorLayer(DATA + "/Batas_KPH.gpkg", "Batas_KPH", "ogr")`	nama teks `"Batas_KPH.shp"` di folder kerja `arcpy.env.workspace`
Memeriksa data terbuka	`layer.isValid()`	`arcpy.Exists(jalur)`
Jumlah fitur	`layer.featureCount()`	`int(arcpy.management.GetCount(batas)[0])`
Nama kolom	`[f.name() for f in layer.fields()]`	`[f.name for f in arcpy.ListFields(batas)]`
Sistem koordinat	`layer.crs().authid()`	`arcpy.Describe(batas).spatialReference.factoryCode`
Jenis geometri	`QgsWkbTypes.displayString(layer.wkbType())`	`arcpy.Describe(batas).shapeType`
Membaca tiap baris	`for fitur in batas.getFeatures():`	`with arcpy.da.SearchCursor(batas, [...]) as kursor:`

Pekerjaan	PyQGIS	ArcPy (Pro)
Isi satu kolom	`fitur["NAMA_KPH"]`	satu isian di baris kursor
Luas satu fitur	`fitur.geometry().area()`	kolom khusus `"SHAPE@AREA"`

Dua hal yang mudah luput. **Pertama**, `authid()` mengembalikan teks `EPSG:32749`, sedangkan `factoryCode` mengembalikan angka `32749`. Perbandingan `== "EPSG:32749"` di PyQGIS menjadi `== 32749` di ArcPy. **Kedua**, kursor ArcPy mengembalikan isian menurut **urutan kolom yang Anda daftarkan**, sedangkan fitur QGIS dibaca dengan nama kolom.

Tabel padanan: raster

Tabel ini diambil dari Skrip 31.1, 31.2, dan 31.3, serta skrip PyQGIS `b2_03` dan `b3_02` (Bab 21).

Pekerjaan	PyQGIS	ArcPy (Pro)
Membuka raster	`QgsRasterLayer(jalur, nama, "gdal")`	`arcpy.Raster(jalur)`
Lebar dan tinggi	`r.width()` dan `r.height()`	`r.width` dan `r.height`
Jumlah band	`r.bandCount()`	`r.bandCount`
Ukuran piksel	`r.rasterUnitsPerPixelX()`	`r.meanCellWidth`
Satu band	`"Citra_KPH@4"` di dalam ekspresi	`Raster(DATA + r"\Citra_KPH.tif\Band_4")`
NDVI	ekspresi `("Citra_KPH@4" - "Citra_KPH@1") / (...)` lewat `native:rastercalc`	`(nir - merah) / (nir + merah)` dengan `Float(...)`
Tinggi tajuk (CHM)	ekspresi `"DSM@1" - "DTM@1"`	`Raster(DSM) - Raster(DTM)`
Rata-rata per poligon	`native:zonalstatisticsfb`, kolom hasil `ndvi_mean`	`ZonalStatisticsAsTable(..., "DATA", "MEAN")`, kolom hasil `MEAN`
Memotong raster	`gdal:cliprasterbymasklayer`	`ExtractByMask(raster, batas).save(...)`

Perhatikan tanda kurung pada baris kedua sampai keempat. Di PyQGIS, lebar raster adalah **fungsi**: `r.width()`. Di ArcPy, ia **sifat**: `r.width` tanpa kurung. Salah menulis kurung memberi galat. Dokumentasi Esri mencatat bahwa `ExtractByMask` mengembalikan objek raster yang disimpan lewat `.save()`, sedangkan hasil `native:rastercalc` di skrip Bab 21 berupa teks jalur yang kemudian dibungkus jadi layer. [CEK]

Tabel padanan: alat vektor

Tabel ini diambil dari Skrip 31.2, 31.3, dan 31.4, serta skrip PyQGIS `b2_02`, `b3_02`, dan `b4_02` (Bab 21 dan 22).

Pekerjaan	PyQGIS	ArcPy (Pro)
Memanggil alat	`processing.run("native:buffer", {...})["OUTPUT"]`	`arcpy.analysis.Buffer(...)`

Pekerjaan	PyQGIS	ArcPy (Pro)
Cara memberi parameter	kamus: nama parameter lalu nilai	isian berurutan atau kata kunci
Penyangga 30 m	`"DISTANCE": 30`	`"30 Meters"`
Gabungkan semua penyangga	`"DISSOLVE": True`	`dissolve_option="ALL"`
Irisan	`native:intersection` dengan `INPUT` dan `OVERLAY`	`arcpy.analysis.Intersect([a, b], keluaran)`
Jumlah titik per poligon	`native:countpointsinpolygon`	`arcpy.analysis.SpatialJoin(...)` lalu kolom `Join_Count`
Memilih satu KPH	`native:extractbyexpression`	`arcpy.analysis.Select(batas, keluaran, "NAMA_KPH = '%s'" % nama)`
Mengubah sistem koordinat	`native:reprojectlayer`	`arcpy.management.Project(...)`
Keluaran sementara	`"TEMPORARY_OUTPUT"`	`memory\nama`

Dua hal lain yang mudah luput. **Pertama**, hasil alat QGIS berupa kamus, jadi Anda mengambil `["OUTPUT"]`. Di ArcPy tidak ada kamus: Anda sendiri yang menyebut nama keluaran, lalu memakai nama itu di langkah berikutnya. **Kedua**, `"DISTANCE": 30` di QGIS mengikuti satuan sistem koordinat layer. Skrip PyQGIS Anda mencatatnya dalam komentar: satuannya meter karena sistem koordinatnya UTM. Di ArcPy, satuan ikut ditulis di dalam teks jarak.

Tabel padanan: peta, galat, dan berkas

Tabel ini diambil dari Skrip 31.1, 31.4, dan 31.5, skrip M1 di Bab 30, serta skrip PyQGIS `b1_03`, `b4_02`, dan `b7_01` (Bab 14, 22, dan 23).

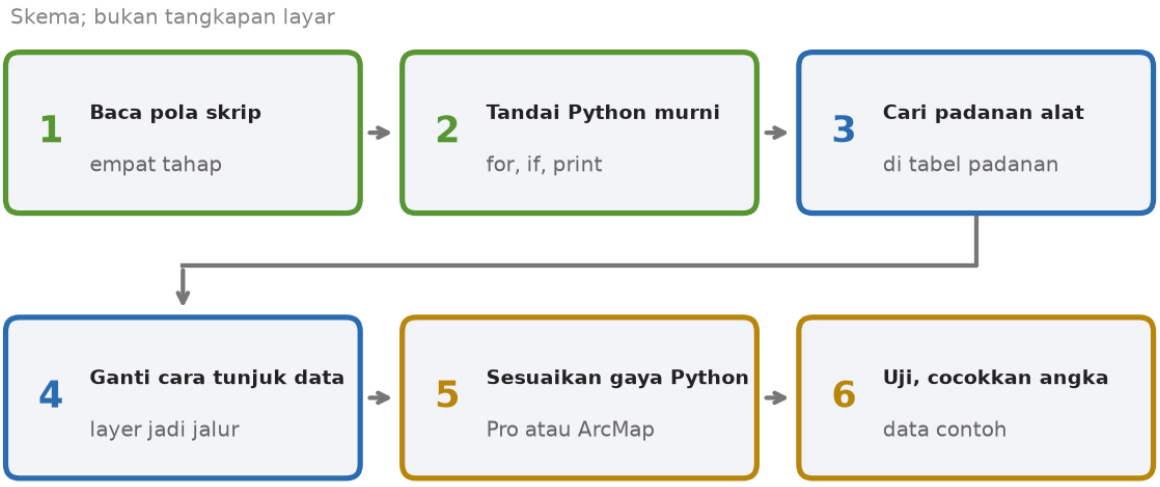
Pekerjaan	PyQGIS	ArcPy (Pro)	ArcPy (ArcMap)
Proyek yang terbuka	`QgsProject.instance()`	`arcpy.mp.ArcGISProject("CURRENT")`	`arcpy.mapping.MapDocument("CURRENT")`
Menambah layer	`proyek.addMapLayer(layer)`	`peta.addDataFromPath(jalur)`	`arcpy.mapping.AddLayer(df, arcpy.mapping.Layer(jalur))`
Mengambil layer	`proyek.mapLayersByName("Batas_KPH")[0]`	`peta.listLayers("Batas_KPH")[0]`	`arcpy.mapping.ListLayers(mxd, "Batas_KPH", df)[0]`
Menyimpan	`proyek.write(berkas)`	`aprx.save()`	`mxd.save()`
Menangkap galat alat	`except Exception as e:`	`except arcpy.ExecuteError:`	`except arcpy.ExecuteError:`
Pesan galat alat	`str(e)`	`arcpy.GetMessages(2)`	`arcpy.GetMessages(2)`

Pekerjaan	PyQGIS	ArcPy (Pro)	ArcPy (ArcMap)
Menulis CSV	`open(..., "w", newline="", encoding="utf-8")`	`open(..., "w", newline="")`	`open(..., "wb")`
Membuat folder	`os.makedirs(HASIL, exist_ok=True)`	`os.makedirs(..., exist_ok=True)`	`if not os.path.exists(...) : os.makedirs(...)`

Bedanya penting pada baris "Menyimpan". `proyek.write(berkas)` menulis ke **berkas yang Anda sebut**. Perintah `aprx.save()` menyimpan perubahan ke **proyek yang sedang terbuka**, yaitu menimpa. Dokumentasi Esri menyediakan `saveACopy(nama_berkas)` untuk menyimpan salinan. [CEK]

Cara menerjemahkan skrip: enam langkah

Ilustrasi 32.3: Enam langkah menerjemahkan skrip



1. **Baca pola skrip dulu.** Hampir semua skrip di buku ini berbentuk empat tahap: tetapkan lokasi data, cek sifat data, ulangi tiap baris, lalu simpan atau tampilkan. Tulis keempat tahap itu di kertas.
2. **Tandai bagian Python murni.** Perulangan `for`, `if`, kamus, daftar, `round`, dan `print` tidak perlu diterjemahkan. Cukup pindahkan.
3. **Cari padanan tiap alat.** Pakai tabel di bab ini. Bila alat tidak ada di tabel, buka dokumentasi alat itu. Cek urutan isiannya, jangan menebak.
4. **Ganti cara menunjuk data.** QGIS memakai layer yang dimuat, ArcPy memakai jalur teks atau nama di folder kerja. Karena itu tentukan `arcpy.env.workspace` dan nama keluaran sementara.
5. **Sesuaikan gaya Python.** Bila tujuannya ArcMap, terapkan beda Python 2.7: `print` dengan `%`, `10000.0`, `.format`, dan `arcpy.mapping`.
6. **Uji kecil dan cocokkan angka acuan.** Jalankan dengan data contoh. Bandingkan hasil dengan angka yang sudah Anda ketahui.

Contoh terjemahan 1: membaca luas tiap KPH

Bagian ini dari skrip PyQGIS `b1_02_baca_atribut.py`, baris 16 sampai 20:

```python
for fitur in batas.getFeatures():
    nama = fitur["NAMA_KPH"]
    luas_ha = fitur.geometry().area() / 10000      # m2 menjadi hektare (CRS harus berupa meter)
    baris.append((nama, round(luas_ha, 2)))
    print(nama, round(luas_ha, 2), "ha")
```

Bagian Python murni: `for`, `print`, `round`. Bagian yang diterjemahkan: cara membaca baris dan cara mendapat luas. `getFeatures()` menjadi kursor, `fitur["NAMA_KPH"]` menjadi isian kursor, dan `fitur.geometry().area()` menjadi kolom `"SHAPE@AREA"`. Padanan ArcPy-nya ada di Skrip 31.1, baris 18 sampai 20:

```python
with arcpy.da.SearchCursor(batas, ["NAMA_KPH", "SHAPE@AREA"]) as kursor:
    for nama, luas in kursor:
        print(nama, round(luas / 10000, 2), "ha")          # luas dalam satuan peta (m2) dibagi 10.000
```

Dua baris PyQGIS tidak punya baris kembaran. Baris `nama = fitur["NAMA_KPH"]` tidak perlu, sebab kursor sudah menaruh nama langsung ke penampung `nama`. Baris `baris.append(...)` hanya dipakai skrip QGIS untuk menulis CSV nanti. Angka acuan untuk menguji terjemahan: luas KPH Alpha 4,0 hektare.

Contoh terjemahan 2: penyangga sungai

Bagian ini dari skrip PyQGIS `b2_02_rantai_vektor.py`, baris 14 sampai 16:

```python
sempadan = processing.run("native:buffer", {
    "INPUT": sungai, "DISTANCE": 30, "SEGMENTS": 8, "END_CAP_STYLE": 0, "JOIN_STYLE": 0,
    "MITER_LIMIT": 2, "DISSOLVE": True, "OUTPUT": "TEMPORARY_OUTPUT"})["OUTPUT"]
```

Semua alat QGIS memakai kamus parameter. Padanan ArcPy-nya di Skrip 31.2, baris 14:

```python
arcpy.analysis.Buffer("Sungai.shp", r"memory\sempadan", "30 Meters", dissolve_option="ALL")
```

Empat parameter QGIS tidak muncul di versi ArcPy: `SEGMENTS`, `END_CAP_STYLE`, `JOIN_STYLE`, dan `MITER_LIMIT`. Artinya versi ArcPy memakai nilai bawaan alat untuk hal-hal itu. Dokumentasi Esri menyebut parameter pilihan `line_side` dan `line_end_type` pada `Buffer` bernilai bawaan `FULL` dan `ROUND`. Karena pengaturan ujung dan sambungan tidak sama persis, sempadan di dua alat bisa selisih sedikit di desimal terakhir. Itu bukan galat. [CEK]

Contoh terjemahan 3: dari ArcGIS Pro ke ArcMap

Terjemahan Pro ke ArcMap lebih mekanis. Ia hanya soal lapis pertama dan kedua. Ambil dua baris dari Skrip 31.2. Versi Pro, baris 15:

```
PYTHON
arcpy.analysis.Intersect(["Batas_KPH.shp", r"memory\sempadan"], r"memory\irisan")
```

Versi ArcMap, baris 16:

```
PYTHON
arcpy.Intersect_analysis(["Batas_KPH.shp", r"in_memory\sempadan"], r"in_memory\irisan")
```

Dua penggantian: `arcpy.analysis.Intersect(...)` menjadi `arcpy.Intersect_analysis(...)`, dan `memory\` menjadi `in_memory\`. Pelajari polanya: nama kotak alat pindah ke belakang dan disambung dengan garis bawah.

> **Pola yang bisa dipakai ulang:** Resep menerjemahkan: **tulis empat tahap skrip**, **tandai yang Python murni**, **cari padanan alat satu per satu di tabel**, **ganti cara menunjuk data**, **sesuaikan gaya Python**, **uji dengan angka acuan**. Hindari menerjemahkan seluruh skrip sekaligus. Terjemahkan satu tahap, uji, lalu lanjut ke tahap berikutnya.

Jebakan lintas alat

Hal-hal ini tidak muncul bila Anda hanya memakai satu alat. Baru terasa ketika Anda berpindah.

Jebakan	Yang terjadi	Cara aman
Urutan kolom di kursor	Isian dibaca menurut urutan daftar kolom. Menukar urutan daftar tanpa menukar nama penampung membuat nilai tertukar diam-diam.	Tulis daftar kolom dan nama penampung berdampingan, dan cetak satu baris untuk memeriksa.
Jarak dan satuan	QGIS mengikuti satuan sistem koordinat layer. ArcPy membaca satuan dari teks `"30 Meters"`.	Periksa sistem koordinat data, dan tulis satuan di ArcPy.
Hasil raster	Hasil `native:rastercalc` berupa teks jalur. Hasil operasi raster ArcPy berupa objek raster yang harus `.save()`.	Ingat: satu bungkus jadi layer, satu lagi disimpan dengan `.save()`.
Kutip dalam klausa pilih	QGIS: `'"NAMA_KPH" = \'%s\''`. Pro: `"NAMA_KPH = '%s'"`. ArcMap: `"\"NAMA_KPH\" = '%s'"`.	Uji klausa dengan satu data kecil dulu. [CEK]
Nama kolom bawaan	Daftar kolom Shapefile di ArcPy memuat kolom bantu seperti `FID` dan `Shape`, sedangkan GeoPackage di QGIS memuat `fid`.	Jangan mengandalkan urutan atau jumlah kolom. Sebut nama kolom. [CEK]

Jebakan	Yang terjadi	Cara aman
Angka antaralat	Dua alat bisa beda di desimal terakhir. Di skrip PyQGIS, persen sempadan KPH Beta tercetak 34.6, sedangkan hitungan dengan OGR memberi 34.7.	Bandingkan dengan toleransi, bukan dengan tanda `==`.
Ekstensi ArcGIS	Alat raster seperti `ZonalStatisticsAsTable` perlu ekstensi Spatial Analyst atau Image Analyst. QGIS tidak memakai ekstensi berbayar untuk itu.	Cek lisensi sebelum menyerahkan skrip ke orang lain.
Tempat menjalankan	Skrip PyQGIS jalan di Windows, macOS, dan Ubuntu. Skrip ArcPy hanya di Windows. Kata `"CURRENT"` hanya berarti di dalam aplikasi ArcGIS.	Putuskan lebih dulu di komputer mana skrip akan dipakai.

Satu contoh nyata untuk baris pertama. Anggap Anda menulis kursor dengan urutan `["SHAPE@AREA", "NAMA_KPH"]`, tetapi tetap membuka dengan `for nama, luas in kursor:`. Penampung `nama` kini berisi luas, dan `luas` berisi nama. Baris `round(luas / 10000, 2)` gagal dengan `TypeError: unsupported operand type(s) for /: 'str' and 'int'`. Pesan itu sudah diuji pada pasangan nilai yang dibalik, di Python 3.

Latihan menerjemahkan

Kerjakan di kertas atau di berkas teks. Tidak ada yang perlu dijalankan di ArcGIS. Jawaban memakai skrip yang sudah ada di buku ini.

1. **PyQGIS ke ArcPy.** Terjemahkan dua baris PyQGIS ini ke ArcGIS Pro: `print("Jumlah fitur:", layer.featureCount())` dan `print("Nama field:", [f.name() for f in layer.fields()])`. Anggap data bernama `batas` di folder kerja.
2. **ArcPy ke PyQGIS.** Terjemahkan satu baris ArcPy Pro ini ke PyQGIS: `arcpy.analysis.Select(batas, r"memory\satu_kph", "NAMA_KPH = '%s'" % nama)`.
3. **Pro ke ArcMap.** Terjemahkan dua baris ArcGIS Pro ini ke ArcMap 10.8: `for i, nama in enumerate(daftar, start=1):` dan `print(f"{i}/{total} selesai: {nama}")`.
4. **Menyusun sendiri.** Tambahkan satu baris ke tabel padanan Anda: pekerjaan "menghitung jumlah fitur" untuk ArcMap. Gunakan Skrip 31.1.

Contoh jawaban.

1. Dari Skrip 31.1, baris 14 dan 15:

```python
print("Jumlah fitur:", int(arcpy.management.GetCount(batas)[0]))
print("Field:", [f.name for f in arcpy.ListFields(batas)])
```

2. Dari skrip PyQGIS `b3_02_batch_per_kph.py`, baris 24 sampai 26:

```
PYTHON
    satu_kph = processing.run("native:extractbyexpression", {
        "INPUT": batas, "EXPRESSION": '"NAMA_KPH" = \'%s\'' % nama,
        "OUTPUT": "TEMPORARY_OUTPUT"})["OUTPUT"]
```

3. Dari skrip M1 versi ArcMap, baris 20 dan 26 (nomor baris ArcMap bergeser satu karena baris `from __future__`):

```
PYTHON
for i, nama in enumerate(daftar, 1):
```

```
PYTHON
    print("{}/{} selesai: {}".format(i, total, nama))
```

4. Dari Skrip 31.1 versi ArcMap, baris 15: `int(arcpy.GetCount_management(batas).getOutput(0))`. Di Pro, penulisannya `int(arcpy.management.GetCount(batas)[0])`.

Semua jawaban disalin langsung dari berkas sumbernya. Hasil menjalankannya di ArcGIS belum teruji. [CEK]

Cek paham

1. Sebutkan tiga lapis beda saat menerjemahkan skrip QGIS ke ArcPy ArcMap.
2. Mengapa `r.width()` di PyQGIS menjadi `r.width` di ArcPy?
3. Mengapa sempadan di dua alat bisa selisih di desimal terakhir?
4. Mengapa hasil alat QGIS diambil dengan `["OUTPUT"]`, sedangkan di ArcPy tidak?

Jawaban.

1. Gaya Python (Pro lawan ArcMap), nama alat (QGIS lawan ArcPy), dan cara menunjuk data (layer dalam proyek lawan jalur teks).
2. Di PyQGIS, lebar raster adalah fungsi, jadi dipanggil dengan kurung. Di ArcPy, ia sifat objek raster, jadi ditulis tanpa kurung.
3. Parameter ujung dan sambungan penyangga tidak persis sama. Itu wajar, dan bukan galat.
4. Alat QGIS mengembalikan kamus berisi semua keluaran. Alat ArcPy menulis ke nama keluaran yang Anda tentukan sendiri, jadi Anda memakai nama itu langsung.

Kesalahan umum

- **Menerjemahkan seluruh skrip sekaligus.** Hasilnya sulit diperiksa. Perbaikan: terjemahkan satu tahap, uji dengan angka acuan, lalu lanjut.

- **Menyalin nama alat dari ingatan.** Nama dan urutan isian alat bisa keliru. Perbaikan: cek dokumentasi alat.
- **Lupa mengganti cara menunjuk data.** Di ArcPy, layer tidak "terbuka" di proyek. Perbaikan: tentukan folder kerja dan nama keluaran.
- **Menganggap angka dua alat harus persis sama.** Perbaikan: bandingkan dengan toleransi kecil.

Ringkasan dan latihan

- Menerjemahkan skrip berarti menerjemahkan tiga lapis: gaya Python, nama alat, dan cara menunjuk data.
- Python murni (`for`, `if`, kamus, daftar) dipindahkan tanpa diubah.
- Tabel padanan membantu mencari nama alat. Dokumentasi alat membantu memeriksa urutan isian.
- Enam langkah: baca pola, tandai Python murni, cari padanan, ganti cara menunjuk data, sesuaikan gaya Python, uji dengan angka acuan.
- Semua skrip ArcPy di buku ini belum dijalankan di ArcGIS dan bertanda **[CEK]**.

Latihan. Ambil skrip PyQGIS dari Bab 21 yang menghitung NDVI per KPH. Tulis ulang rencana terjemahannya ke ArcPy dengan enam langkah di atas, tanpa menjalankan apa pun. Mana yang paling banyak berubah: bagian Python murni, nama alat, atau cara menunjuk data?

Tabel perbandingan: tiga lapis beda

Lapis	Yang berbeda	Contoh
Gaya Python	Pro lawan ArcMap	`print(nama, luas)` lawan `print("%s %s" % (nama, luas))`
Nama alat	QGIS lawan ArcPy	`native:buffer` lawan `arcpy.analysis.Buffer`
Cara menunjuk data	layer dalam proyek lawan jalur teks	`QgsVectorLayer(...)` lawan `"Batas_KPH.shp"`

Glosarium

Glosarium ini memuat istilah yang dipakai di buku ini, diurutkan menurut abjad. Bila Anda bertemu kata yang belum dikenal, cari di sini, lalu buka bab yang disebut di ujung tiap entri.

Agregasi: ringkasan (jumlah, rata-rata, gabungan teks) dari layer lain di dalam ekspresi. *Bab 22, 23.*

AHP: metode menghitung bobot kriteria dari perbandingan berpasangan. *Bab 25.*

Akumulasi aliran: jumlah sel di hulu yang airnya melewati sebuah sel. *Bab 25.*

Akurasi keseluruhan, akurasi pembuat, akurasi pengguna: ketepatan total, dari sisi lapangan, dan dari sisi peta. *Bab 19, 24, 26.*

Alat (tool) geoprocessing: satu pekerjaan siap pakai di ArcGIS atau QGIS, dipanggil dari skrip. *Bab 3, 4, 10.*

Alat Processing: satu fungsi analisis QGIS yang bisa dipanggil dari skrip. *Bab 3, 17, 18.*

Algoritma Processing: alat yang tampil di Processing Toolbox dan bisa dipanggil lewat `processing.run`. *Bab 11, 27.*

Alias (impor): julukan pendek untuk nama panjang, ditulis dengan `as`. *Bab 18, 20, 23.*

Angka acuan: hasil yang diketahui benar dari data contoh, dipakai untuk menguji terjemahan skrip. *Bab 25, 32.*

API: daftar perintah resmi sebuah program, seperti buku menu, yang boleh dipanggil dari program lain. *Bab 4, 7, 8.*

Arah lereng (aspect): ke mana lereng menghadap, dari utara searah jarum jam. *Bab 16, 19.*

ArcGIS Pro: aplikasi ArcGIS generasi baru, memakai Python 3 dan modul `arcpy.mp`. *Bab 1, 9, 29.*

ArcMap: aplikasi ArcGIS generasi lama (versi 10.8 di buku ini), memakai Python 2.7 dan modul `arcpy.mapping`. *Bab 1, 2, 9.*

ArcPy: perkakas Python bawaan ArcGIS Pro dan ArcMap. *Bab 1, 11, 13.*

Area latih: lokasi yang kelas sebenarnya sudah diketahui, dipakai untuk melatih atau memberi nama. *Bab 19.*

Area layanan: wilayah yang terjangkau dalam batas jarak atau waktu dari satu titik. *Bab 25.*

Argumen: nilai sungguhan yang diberikan kepada fungsi saat fungsi dipanggil. *Bab 7, 11, 14.*

Array: tabel angka berbentuk kotak dengan baris dan kolom, dikelola oleh numpy. *Bab 12, 19, 25.*

assert: perintah untuk memastikan sebuah syarat benar; bila salah, skrip berhenti. *Bab 10, 17, 20.*

Atlas: fitur tata letak QGIS yang membuat satu halaman per fitur. *Bab 24, 30.*

Atribut: isi yang menempel pada sebuah objek. *Bab 5, 6, 7.*

Aturan batas: apakah nilai tepat di batas masuk kelas bawah atau atas. *Bab 16, 17, 19.*

Aturan isian (tegas dan lunak): syarat sebuah isian; tegas menolak simpan, lunak hanya memperingatkan. *Bab 20.*

Band: satu lapis angka dalam raster. *Bab 9, 12, 14.*

Baris coding: komentar `# -*- coding: utf-8 -*-` di awal berkas Python 2 agar huruf non-ASCII terbaca. *Bab 29.*

Bash dan set -euo pipefail: bahasa perintah terminal, dan pengaman agar skrip berhenti saat ada galat. *Bab 25.*

Batch: pekerjaan sama yang diulang untuk banyak masukan. *Bab 21, 22, 23.*

Bayangan bukit (hillshade): gambar bayangan seolah bukit disinari matahari. *Bab 16, 17, 19.*

Bentuk (shape): ukuran sebuah array, misalnya 3 baris dan 4 kolom. *Bab 1, 5, 7.*

Bias, presisi, galat RMS, CEP50, R95: ukuran akurasi dan sebaran pembacaan GNSS. *Bab 3, 7, 8.*

Bilangan bulat (integer): angka tanpa desimal, misalnya 7. *Bab 4, 15, 16.*

Bilangan desimal (float): angka dengan desimal, ditulis dengan titik, misalnya 4.0. *Bab 4, 26.*

Blok: kelompok baris yang menjorok ke dalam dan menjadi isi dari if, for, atau def di atasnya. *Bab 10, 19, 20.*

BOM (`utf-8-sig`): tiga byte penanda di awal berkas UTF-8 agar Excel mengenali huruf.

break: perintah untuk menghentikan perulangan lebih awal. *Bab 6, 15, 17.*

Categorized: penyaji yang memberi satu warna untuk tiap nilai teks pada sebuah kolom. *Bab 17.*

Ceklis masuk: daftar pemeriksaan otomatis kemampuan sebelum masuk kelas lanjutan. *Bab 23.*

CHM: tinggi tajuk, yaitu DSM dikurangi DTM. *Bab 19, 21, 23.*

classFactory: fungsi pintu masuk plugin yang dipanggil QGIS saat memuat. *Bab 27.*

COG (Cloud-Optimized GeoTIFF): GeoTIFF yang bisa dibaca sebagian lewat internet.

commit: menyimpan hasil edit layer ke berkas. *Bab 15, 20.*

Comprehension: cara menulis perulangan dalam satu baris untuk membuat daftar atau set baru. *Bab 6, 10, 14.*

Counter, setdefault, lambda, dictionary comprehension: penghitung kemunculan; ambil atau buat isi kunci kamus; fungsi mini satu baris; menyusun kamus dalam satu baris. *Bab 18, 20, 26.*

CRS: sistem koordinat layer. *Bab 13, 14, 16.*

CSV: berkas teks berisi tabel dengan kolom dipisah koma, bisa dibuka di Excel. *Bab 8, 9, 14.*

CURRENT: kata yang berarti proyek atau dokumen peta yang sedang terbuka. *Bab 29, 30, 31.*

Daftar (list): kumpulan nilai berurutan dalam kurung siku, dengan nomor urut yang dimulai dari 0. *Bab 1, 2, 5.*

DAS dan titik keluar (outlet): daerah yang airnya bermuara ke satu titik keluar. *Bab 4, 5, 6.*

Dasbor (statis): satu halaman ringkasan angka dan grafik dari data tertentu. *Bab 27.*

Data latih dan data uji; RMSE, MAE, bias; tinggalkan-satu (LOO): cara menilai tebakan dengan data yang disembunyikan, dan tiga ukuran galatnya. *Bab 25.*

Dekorator: pembungkus fungsi yang dipasang di atas def dengan tanda @. *Bab 11, 27.*

DEM: raster yang angkanya tinggi tanah. *Bab 16, 19, 23.*

Deret waktu: kumpulan citra lokasi sama pada tanggal berbeda. *Bab 11, 12, 26.*

Dictionary comprehension: pembuatan kamus dalam satu baris dengan `{kunci: nilai for ...}`. *Bab 18, 21, 25.*

digitasi: menambah atau mengubah gambar fitur. *Bab 5, 6, 7.*

dir, help, type: tiga perintah Python untuk menampilkan isi, penjelasan, dan jenis sebuah nilai. *Bab 8, 9, 10.*

Docstring: catatan tiga tanda kutip di bawah def yang menjelaskan sebuah fungsi. *Bab 7, 10, 15.*

Dokumentasi: halaman resmi yang menjelaskan tiap perintah sebuah perkakas. *Bab 2, 3, 13.*

Domain (daftar dan rentang): daftar atau rentang nilai sah untuk satu kolom. *Bab 20.*

dpi: banyaknya titik gambar tiap inci pada PNG hasil ekspor. *Bab 12, 17, 26.*

DSM: model tinggi permukaan (puncak pohon dan atap); DTM: model tinggi tanah. *Bab 14, 19, 21.*

Editor skrip: tempat di dalam QGIS untuk menulis dan menyimpan berkas skrip. *Bab 3, 14, 15.*

ekspresi: kalimat syarat atau rumus yang dipahami QGIS. *Bab 14, 15, 17.*

Ekspresi QGIS: bahasa rumus kecil milik QGIS untuk label dan kolom, terpisah dari Python. *Bab 15, 18, 20.*

Ekstensi (extension): izin tambahan di ArcGIS untuk sekelompok alat, misalnya Spatial Analyst. *Bab 17, 21, 24.*

enumerate: perkakas yang menambahkan nomor urut, mulai dari 0, pada setiap putaran perulangan. *Bab 6, 15, 17.*

Equal Interval, Quantile (Equal Count), Natural Breaks (Jenks): tiga cara membagi angka menjadi kelas (rentang sama lebar, jumlah petak hampir sama, dan potong di celah besar). *Bab 17.*

ExecuteError: jenis galat ArcPy ketika sebuah alat gagal. *Bab 31, 32.*

extent: batas luar sebuah raster. *Bab 16, 19, 25.*

Faktor Z: pengali tinggi bila satuan tinggi dan satuan peta berbeda. *Bab 19.*

Feedback: saluran pesan, kemajuan, dan pembatalan pada algoritma Processing. *Bab 11, 27.*

filter layer (subset string): menyembunyikan fitur yang tidak memenuhi syarat sampai filter dikosongkan. *Bab 15, 16.*

fitur: satu objek di layer vektor, misalnya satu petak. *Bab 10, 11, 13.*

for: perintah untuk mengulang pekerjaan pada setiap isi sebuah kumpulan. *Bab 5, 6, 7.*

from __future__ import print_function: baris yang membuat `print(...)` berlaku sebagai fungsi di Python 2.7. *Bab 29, 30.*

f-string: teks bertanda f di depan yang boleh memuat nilai dalam kurung kurawal; hanya ada di Python 3.6 ke atas. *Bab 9, 24, 28.*

f-string dan .format: dua cara merakit teks; f-string baru ada sejak Python 3.6, jadi tidak di Python 2.7. *Bab 9, 24, 28.*

Fungsi: resep bernama yang dibuat dengan def; diberi bahan, ia mengerjakan langkahnya lalu mengembalikan hasil. *Bab 6, 7, 8.*

Fungsi tugas / fungsi selesai (callback): fungsi yang bekerja di latar belakang, dan fungsi yang dipanggil kembali di program utama setelah tugas berakhir. *Bab 21, 23.*

Fungsi tugas dan fungsi selesai (QgsTask): fungsi yang bekerja di latar belakang dan fungsi yang dipanggil di program utama setelah tugas berakhir. *Bab 21, 23.*

Galat (exception): peristiwa yang menghentikan skrip, disertai pesan tentang jenis dan letaknya. *Bab 1, 3, 4.*

Galat tulis dan galat hitung (ekspresi): ekspresi yang tidak terbaca, lawan ekspresi yang terbaca tetapi tidak bisa dihitung. *Bab 22.*

Gaya tersimpan: pengaturan tampilan dan formulir yang disimpan di dalam GeoPackage. *Bab 20.*

GDAL (osgeo): pustaka untuk membaca dan menulis raster dan vektor. *Bab 12, 14, 16.*

Geodatabase (`.gdb`): satu wadah untuk banyak layer dan tabel. *Bab 30.*

GeoJSON: berkas teks berisi objek peta dan atributnya, dibaca peramban. *Bab 27.*

geometri: bentuk sebuah fitur (titik, garis, poligon). *Bab 14, 15, 18.*

geometri sah: bentuk yang tidak menyilang dirinya sendiri. *Bab 15.*

GeoPackage: satu berkas `.gpkg` berisi banyak layer, tabel, dan pengaturan. *Bab 15, 16, 18.*

GNSS: sistem satelit penentu posisi. *Bab 20.*

Graduated: penyaji yang memberi satu warna untuk tiap rentang angka pada sebuah kolom. *Bab 17.*

Hillshade (bayangan bukit): tampilan relief dari arah dan tinggi cahaya. *Bab 16, 17, 19.*

HTML: berkas teks yang dibuka peramban sebagai halaman web. *Bab 27, 28.*

ID alat: nama unik alat untuk skrip, misalnya `native:buffer`. *Bab 21, 22, 23.*

if, elif, else: perintah untuk mengambil keputusan menurut syarat. *Bab 2, 3, 6.*

iface: kotak bernama khusus di dalam QGIS untuk mengendalikan jendela QGIS dari skrip; tidak ada di luar QGIS. *Bab 3, 27.*

import: perintah untuk memanggil sebuah modul. *Bab 3, 8, 9.*

Indeks: nomor urut sebuah nilai di dalam daftar; indeks pertama adalah 0. *Bab 5, 12, 16.*

Indeks spasial: daftar isi geometri yang mempercepat pencarian titik di dalam poligon. *Bab 25, 27.*

Indentasi: jarak menjorok yang menandai sebuah baris termasuk blok di atasnya; Python memakai empat spasi. *Bab 6.*

initGui dan unload: metode plugin untuk memasang dan mencabut menu serta tombol. *Bab 27.*

Interpolasi, IDW, TIN: menebak nilai di tempat tak terukur; IDW memberi bobot menurut jarak, TIN memakai jaring segitiga. *Bab 12, 20, 25.*

Interpolasi linear: menarik garis lurus antara dua nilai yang diketahui untuk menebak nilai di antaranya. *Bab 20, 26.*

Jalur (path): alamat lengkap sebuah berkas atau folder. *Bab 8, 10, 14.*

Jarak minimum: aturan klasifikasi yang memilih kelas dengan rata-rata paling dekat. *Bab 19.*

Jendela Python: kotak di dalam ArcGIS untuk mengetik dan menjalankan perintah Python. *Bab 29, 30, 31.*

join, kunci join, join permanen, join virtual: menempelkan kolom dari tabel lain berdasarkan kolom kunci; permanen menghasilkan layer baru, virtual hanya tampilan. *Bab 7, 8, 12.*

JSON: berkas teks berisi data bersusun yang mirip kamus dan daftar. *Bab 9, 20, 27.*

Kamus (dictionary): kumpulan pasangan kunci dan nilai yang dicari lewat kunci. *Bab 5, 6, 7.*

Kamus parameter: formulir isian alat Processing berupa kamus Python berisi nama isian dan nilainya. *Bab 18, 21, 22.*

Kamus parameter / kamus hasil: isian alat dan jawaban alat, berupa pasangan nama dan nilai. *Bab 18, 21, 22.*

Kappa: kecocokan setelah dikurangi kecocokan karena kebetulan. *Bab 19.*

Kelas (class): cetakan untuk membuat objek. *Bab 11, 13, 14.*

Keluaran sementara (`TEMPORARY_OUTPUT`): hasil perantara yang tidak ditulis ke berkas pilihan pengguna. *Bab 21, 31, 32.*

Kemiringan (slope): seberapa miring permukaan, dalam derajat atau persen. *Bab 12, 16, 19.*

Kemiripan maksimum: aturan klasifikasi yang memilih kelas paling cocok dengan memperhitungkan sebarannya. *Bab 19.*

Klasifikasi: mengubah citra berisi angka pantulan menjadi peta kelas. *Bab 12, 17, 19.*

Kluster: kelompok piksel yang angka bandnya mirip. *Bab 19.*

K-means: cara mengelompokkan dengan memindah pusat kelompok berulang kali. *Bab 19.*

kolom (field): satu jenis keterangan pada tabel atribut. *Bab 3, 9, 10.*

Kolom virtual: kolom yang dihitung dari ekspresi dan tidak disimpan di berkas. *Bab 20.*

Komentar: catatan untuk manusia yang diawali tanda pagar; Python mengabaikannya. *Bab 4, 5, 6.*

komposit: citra multi-band hasil menumpuk band. *Bab 16, 19, 26.*

Konsol Python: kotak ketik di dalam QGIS yang menjalankan perintah satu per satu. *Bab 3, 4, 13.*

konstanta: kotak variabel yang isinya tidak diubah, ditulis huruf besar. *Bab 15, 26, 27.*

Konteks (ekspresi): fitur, layer, dan proyek tempat sebuah ekspresi QGIS dihitung. *Bab 16, 17, 20.*

Konteks ekspresi: lingkungan tempat ekspresi dihitung. *Bab 20, 23.*

Kovarians: ukuran sebaran band satu kelas dan hubungan antarband. *Bab 19.*

Kualitas data, kelengkapan, keabsahan, konsistensi, keunikan: lima istilah pemeriksaan data. *Bab 20.*

Kunci: bagian pasangan di kamus yang dipakai untuk mencari nilai. *Bab 5, 10, 13.*

Label dan halo: tulisan di atas fitur peta, dan garis tepi putih di sekeliling huruf agar terbaca. *Bab 4, 11, 16.*

lambda: fungsi satu baris tanpa nama, biasa dipakai sebagai kriteria pendek. *Bab 7, 15, 17.*

Layer: satu data di dalam proyek, vektor atau raster. *Bab 9, 10, 11.*

Layer indeks: layer yang tiap fiturnya menjadi satu halaman Map Series atau Data Driven Pages. *Bab 30.*

Layer memori: layer sementara di memori komputer. *Bab 20.*

Layer sementara (`memory:`): hasil alat yang hanya ada di memori dan tidak tersimpan ke berkas. *Bab 15, 17, 18.*

Layout dan item: lembar peta untuk dicetak, dan kotak-kotak isinya (peta, judul, legenda, skala, panah). *Bab 17, 20, 22.*

Leaflet: pustaka JavaScript untuk menggambar peta di peramban. *Bab 27.*

Log: catatan hasil tiap langkah skrip yang dapat dibaca belakangan. *Bab 7, 15, 21.*

Lubang (sink): sel DEM yang lebih rendah dari semua tetangganya. *Bab 15, 19, 25.*

Map algebra: hitungan antar raster memakai operator biasa seperti pengurangan dan pembagian. *Bab 30.*

Map Series (Pro) dan Data Driven Pages (ArcMap): fitur yang membuat satu halaman peta untuk tiap fitur di layer indeks. *Bab 30.*

Mask: peta penanda piksel yang dipakai dan dibuang. *Bab 16, 21, 24.*

mask dan crop to cutline: poligon pemotong, dan memangkas kotak raster mengikuti poligon. *Bab 16, 21, 24.*

Masukan model, langkah (child algorithm), penjaga, sumber nilai: isian yang diisi pengguna, satu alat di dalam model, langkah yang menghentikan model bila syarat gagal, dan asal nilai sebuah isian langkah. *Bab 18, 23.*

matplotlib: pustaka untuk membuat grafik dan gambar. *Bab 12, 26.*

Matriks kesalahan: tabel kelas sebenarnya (baris) lawan kelas pada peta (kolom). *Bab 19.*

Median / komposit median: nilai tengah; citra dari nilai tengah beberapa tanggal. *Bab 15, 21, 26.*

Menjalankan: menyuruh komputer membaca skrip dari atas ke bawah dan mengerjakan tiap perintah. *Bab 1, 2, 3.*

Metode: fungsi yang menempel pada sebuah objek atau kelas. *Bab 11, 14, 17.*

Model (Processing): alur kerja yang disimpan dalam berkas `.model3`. *Bab 16, 17, 18.*

Model Processing: resep tertulis dari rangkaian alat, disimpan sebagai berkas `.model3`. *Bab 18, 23.*

Modul: berkas Python berisi perkakas siap pakai yang dipanggil dengan import. *Bab 3, 8, 9.*

Multipart dan singlepart: satu objek berisi banyak bagian terpisah, lawan satu objek satu bagian. *Bab 18.*

NDVI: indeks vegetasi dari pantulan inframerah dekat dan merah. *Bab 16, 19, 21.*

Nilai: isi yang disimpan dalam sebuah variabel, misalnya angka 4.0 atau teks "KPH Alpha". *Bab 1, 4, 5.*

Nilai bawaan: isian yang terisi sendiri. *Bab 7, 10, 17.*

NIR: band inframerah dekat. *Bab 16, 19, 21.*

NoData: nilai penanda kotak raster yang kosong. *Bab 16, 19, 21.*

numpy: pustaka untuk hitungan angka massal. *Bab 12, 19, 20.*

OBJ: format teks sederhana untuk jaring segitiga 3D. *Bab 5, 11, 13.*

Objek: barang hasil cetakan sebuah kelas; membawa isi dan kemampuannya sendiri. *Bab 3, 11, 12.*

Objek raster: nilai Python yang mewakili sebuah raster dan bisa dijumlah, dikurangi, dan dibagi. *Bab 30, 32.*

Padanan: dua penulisan di dua alat yang mengerjakan hal yang sama. *Bab 14, 15, 16.*

Parameter: nama bahan yang diterima sebuah fungsi saat fungsi dibuat. *Bab 7, 11, 15.*

@parent: fitur yang sedang dibaca di layer luar, dipakai di dalam agregasi. *Bab 22, 23, 24.*

Pembungkus (wrapper): fungsi kecil yang memanggil fungsi lain dengan penanganan galat di dalamnya. *Bab 11, 15, 22.*

Penyaji (renderer): objek QGIS yang memegang aturan simbologi satu layer. *Bab 17.*

Perintah: satu langkah dalam sebuah skrip, misalnya menghitung luas. *Bab 1, 2, 3.*

Pewarisan: kemampuan sebuah kelas mengambil isi dan kemampuan kelas induknya. *Bab 11.*

pg_dump dan pg_restore: alat PostgreSQL untuk membuat dan memulihkan cadangan. *Bab 25.*

pgRouting: ekstensi PostgreSQL untuk menghitung rute dan area layanan. *Bab 25.*

Piksel: satu petak raster yang menyimpan satu angka. *Bab 12, 14, 16.*

Plugin: paket kecil yang menambah menu dan tombol di QGIS. *Bab 2, 21, 25.*

Pola (kerangka): bentuk skrip yang berulang, dengan bagian isian yang diganti. *Bab 7, 10, 15.*

Pola berkas (`glob`): cara mencari banyak berkas sekaligus berdasarkan pola nama. *Bab 21.*

PostGIS: ekstensi PostgreSQL yang menambahkan jenis dan fungsi spasial. *Bab 12, 25.*

print: perintah untuk menampilkan sesuatu di layar. *Bab 1, 3, 4.*

processing.run: perintah memanggil alat Processing dari skrip; hasilnya kamus. *Bab 7, 15, 16.*

Profil waktu: nilai satu piksel sepanjang tanggal. *Bab 26.*

Proyek: wadah semua layer, dilihat dari Python lewat `QgsProject`. *Bab 14, 17, 18.*

Proyek (`.aprx`) dan dokumen peta (`.mxd`): berkas penyimpan peta di ArcGIS Pro dan ArcMap. *Bab 14, 17, 18.*

Proyek terpadu: satu skrip yang memadukan beberapa tahap analisis dan menyimpan hasil permanen. *Bab 23, 31.*

Pseudocolor: pewarnaan raster dengan satu warna untuk tiap rentang nilai. *Bab 17.*

Pustaka (library): kotak perkakas tambahan yang ditulis orang lain dan boleh dipakai. *Bab 12, 16, 17.*

PyQGIS: Python yang dipakai untuk mengendalikan QGIS. *Bab 1, 2, 4.*

Python: bahasa tulisan yang dipahami komputer, dengan aturan yang sengaja dibuat mudah dibaca. *Bab 1, 2, 3.*

QgsExpression, konteks ekspresi: cara menghitung ekspresi QGIS dari Python. *Bab 17, 20, 22.*

QgsFeatureRequest: permintaan data berfilter yang mengembalikan baris cocok ke skrip tanpa menyorotnya. *Bab 20, 27, 28.*

QgsTask: tugas latar belakang milik QGIS. *Bab 21, 23, 24.*

raise: perintah untuk menimbulkan galat sendiri dengan pesan yang jelas. *Bab 10, 18, 20.*

Rantai alat: keluaran satu alat dipakai sebagai masukan alat berikutnya. *Bab 21, 31.*

raster, piksel, band: gambar dari kotak berangka; satu kotak; satu lembar angka. *Bab 12, 14, 15.*

Rasterisasi: mengubah poligon menjadi raster. *Bab 19.*

rekap per kategori: statistik untuk tiap kelompok. *Bab 15.*

reklasifikasi: mengganti rentang nilai menjadi nomor kelas. *Bab 16, 19, 25.*

Reklasifikasi, skor, bobot, kendala: mengubah nilai ke skor, tingkat penting tiap kriteria, dan larangan mutlak pada analisis kesesuaian. *Bab 16, 19, 25.*

Reproject, buffer, clip, dissolve, Intersect, Difference, Union: ubah CRS; zona selebar jarak; potong dengan layer lain; lebur batas; ambil bagian yang bertumpuk; ambil bagian yang tidak tertutup; gabung keduanya. *Bab 17, 25.*

return: perintah untuk mengembalikan hasil sebuah fungsi kepada pemanggilnya. *Bab 7, 10, 11.*

RMSE: akar rata-rata kuadrat selisih; ukuran besar galat. *Bab 20, 25, 26.*

Ruang sementara (`memory` di Pro, `in_memory` di ArcMap): tempat hasil antara disimpan di memori, tanpa berkas. *Bab 29, 31.*

Satu-ke-banyak dan satu-ke-satu: satu objek bisa mendapat beberapa baris hasil, lawan hanya satu baris. *Bab 18.*

SCL: peta penanda kelas piksel (awan, bayangan, vegetasi, air) pada Sentinel-2. *Bab 17, 26.*

seleksi: menandai sebagian fitur tanpa menghapus yang lain. *Bab 5, 6, 7.*

self: kata yang menunjuk objek yang sedang dipakai di dalam sebuah metode. *Bab 11, 15, 26.*

Semivariogram dan kriging biasa: pola kemiripan nilai menurut jarak, dan interpolasi yang memakainya untuk bobot. *Bab 25.*

Set (himpunan): kumpulan nilai tanpa kembar dan tanpa urutan. *Bab 4, 5, 6.*

setSubsetString: saringan yang menempel pada layer sampai dikosongkan. *Bab 15, 16, 17.*

Simbologi: aturan warna dan bentuk gambar sebuah layer di peta. *Bab 17.*

simpul (vertex): titik sudut sebuah garis atau poligon. *Bab 15, 17, 25.*

Simpul dan ruas (jaringan): persimpangan atau ujung jalan, dan potongan jalan di antaranya. *Bab 15, 17, 25.*

Sink: tempat keluaran algoritma Processing ditulis. *Bab 11, 25, 27.*

Skema: daftar kolom beserta jenisnya. *Bab 1, 2, 3.*

Skrip: berkas teks berisi urutan perintah untuk komputer, dijalankan dari atas ke bawah. Berkas skrip Python berakhiran .py. *Bab 1, 2, 3.*

Skrip mandiri: skrip yang dijalankan dari terminal, di luar QGIS. *Bab 3, 9, 14.*

Spatial join dan predikat: gabung atribut menurut hubungan letak, dan aturan hubungan letaknya (misalnya intersect, are within). *Bab 18, 24, 30.*

SQL spasial: SQL dengan fungsi bentuk seperti `ST_Area` dan `ST_Intersects`. *Bab 25.*

SQL, WHERE, GROUP BY, HAVING, JOIN: bahasa tanya basis data beserta bagian utamanya. *Bab 20, 25.*

statistik zonal: statistik raster per poligon. *Bab 16, 21, 31.*

SVG: bahasa gambar berbentuk teks yang dipahami peramban (dipakai grafik dasbor). *Bab 17, 27.*

Syarat: pertanyaan yang jawabannya hanya benar atau salah, misalnya luas lebih dari 5. *Bab 6, 10, 15.*

sys.argv: daftar kata yang diketik di terminal setelah nama skrip. *Bab 25, 27.*

Tabel tanpa geometri: tabel biasa tanpa titik, garis, atau poligon. *Bab 20.*

Tanda spektral: ciri angka band satu kelas (rata-rata dan sebaran). *Bab 19.*

Tanpa jendela (headless): menjalankan QGIS dari terminal tanpa tampilan grafis. *Bab 12, 17, 18.*

Teks (string): rangkaian huruf, angka, dan tanda baca yang selalu diapit tanda kutip. *Bab 4, 5, 7.*

Teks mentah (`r"..."`): teks yang garis miring terbaliknya dibaca apa adanya. *Bab 15, 29.*

TEMPORARY_OUTPUT: nilai isian `OUTPUT` yang membuat berkas sementara dan mengembalikan teks jalurnya. *Bab 16, 19, 21.*

Terminal: jendela untuk mengetik perintah berbentuk teks. *Bab 1, 2, 3.*

tipe data raster (UInt16, Byte): jenis angka yang disimpan tiap piksel.

Titik referensi: titik berkoordinat benar yang dipakai sebagai kunci jawaban. *Bab 20.*

Titik sudut (vertex) dan segitiga (face): bagian jaring 3D; baris `v` dan `f` di berkas OBJ. *Bab 15, 27.*

Titik uji: titik berkelas pasti yang tidak ikut melatih. *Bab 19, 20, 25.*

Toleransi topologi: jarak terjauh dua ujung jalan masih dianggap menyambung.

topologi: aturan kerapian antar bentuk (tidak ganda, tidak tumpang tindih, tidak berlubang). *Bab 15.*

Traceback: tulisan jejak galat; dibaca dari baris paling bawah ke atas. *Bab 10, 22, 24.*

Trigger (pagar): aturan di basis data yang berjalan otomatis saat data masuk atau berubah. *Bab 20.*

try dan except: perintah untuk mencoba suatu pekerjaan dan menangani galat bila terjadi, tanpa menghentikan skrip. *Bab 10, 14, 15.*

Tugas latar belakang (QgsTask): pekerjaan yang jalan di belakang layar sehingga jendela QGIS tidak membeku. *Bab 21, 23, 24.*

tumpang tindih (overlap) dan celah (gap): dua bentuk berbagi area, dan lubang di antara bentuk bertetangga. *Bab 15, 18, 21.*

Tumpukan (stack): satu berkas raster berisi banyak band, satu band satu tanggal. *Bab 6, 17, 18.*

Tuple: daftar yang tidak bisa diubah setelah dibuat, ditulis dengan kurung biasa. *Bab 5, 8, 10.*

Ubin (tile): kepingan gambar peta dasar bernomor x, y, zoom. *Bab 21, 24, 27.*

Uji negatif: uji yang sengaja dibuat gagal untuk membuktikan alat uji benar-benar bisa menyatakan GAGAL. *Bab 23.*

Validasi: pemeriksaan masukan sebelum dipakai. *Bab 22, 24, 25.*

Value Map, Value Relation, Range: kotak pilihan daftar tetap, kotak pilihan dari tabel acuan, dan kotak angka berbatas. *Bab 20.*

Variabel: kotak bernama yang menyimpan satu nilai. *Bab 4, 8, 14.*

Variabel lingkungan: pengaturan bernama yang disimpan sistem operasi dan bisa dibaca skrip. *Bab 8, 25, 26.*

Versi: nomor edisi sebuah perangkat lunak, misalnya 3.12.13. *Bab 2, 3, 9.*

Widget: kotak isian di formulir. *Bab 20.*

with open: cara aman membuka berkas; Python menutupnya otomatis setelah blok selesai. *Bab 8, 9, 14.*

Zona dan histogram zonal: poligon tempat raster dirangkum, dan jumlah piksel tiap kelas di dalam tiap zona. *Bab 16, 18, 19.*

Daftar Pustaka

Daftar ini memuat sumber yang dipakai menyusun buku ini. Semua tautan dibuka pada 7 Oktober 2026. Alamat halaman bisa berubah di kemudian hari. Skrip dan data latihan berasal dari seri buku tutorial GIS karya penulis yang sama, dan tidak dicantumkan lagi di sini.

Dokumentasi QGIS dan PyQGIS

- QGIS Project. (t.t.). *Installation Guide*. qgis.org. https://qgis.org/resources/installation-guide/.
- QGIS Project. (t.t.). *PyQGIS Developer Cookbook*. QGIS Documentation. https://docs.qgis.org/latest/en/docs/pyqgis_developer_cookbook/index.html.
- QGIS Project. (t.t.). *Introduction* (PyQGIS Developer Cookbook). QGIS Documentation. https://docs.qgis.org/latest/en/docs/pyqgis_developer_cookbook/intro.html.
- QGIS Project. (t.t.). *Python Console and Code Editor*. Dalam *QGIS User Manual*, QGIS Documentation. https://docs.qgis.org/latest/en/docs/user_manual/plugins/python_console.html.
- QGIS Project. (t.t.). *QGIS Python API Documentation*. qgis.org. https://qgis.org/pyqgis/.
- QGIS Project. (t.t.). *QgsApplication Class Reference*. QGIS Python API. https://qgis.org/pyqgis/master/core/QgsApplication.html.
- QGIS Project. (t.t.). *QgsVectorLayer Class Reference*. QGIS Python API. https://qgis.org/pyqgis/master/core/QgsVectorLayer.html.

Dokumentasi Python

- Python Software Foundation. (t.t.). *Using Python on macOS*. Python Documentation. https://docs.python.org/3/using/mac.html.
- Python Software Foundation. (t.t.). *Using Python on Unix platforms*. Python Documentation. https://docs.python.org/3/using/unix.html.
- Python Software Foundation. (t.t.). *Using Python on Windows*. Python Documentation. https://docs.python.org/3/using/windows.html.
- Python Software Foundation. (t.t.). *Porting Python 2 Code to Python 3*. Python Documentation. https://docs.python.org/3/howto/pyporting.html.
- Python Software Foundation. (t.t.). *Simple statements*. Dalam *The Python 2.7 Language Reference*. https://docs.python.org/2.7/reference/simple_stmts.html.
- Python Software Foundation. (t.t.). *PEP 263: Defining Python Source Code Encodings*. https://peps.python.org/pep-0263/.
- Python Software Foundation. (t.t.). *PEP 498: Literal String Interpolation*. https://peps.python.org/pep-0498/.

Dokumentasi ArcGIS dan ArcPy

- Esri. (t.t.). *ArcGIS Desktop 10.8.x system requirements*. System Requirements. https://desktop.arcgis.com/en/system-requirements/latest/arcgis-desktop-system-requirements.htm.

- Esri. (t.t.). *ArcGIS Pro system requirements*. ArcGIS Pro documentation. https://doc.esri.com/en/arcgis-pro/latest/get-started/arcgis-pro-system-requirements.html.
- Esri. (t.t.). *ArcGIS Pro Python reference*. ArcGIS Pro documentation. https://doc.esri.com/en/arcgis-pro/latest/arcpy/main/arcgis-pro-arcpy-reference.html.
- Esri. (t.t.). *ArcGISProject class*. ArcGIS Pro documentation. https://doc.esri.com/en/arcgis-pro/latest/arcpy/mapping/arcgisproject-class.html.
- Esri. (t.t.). *DataDrivenPages class*. ArcMap Documentation. https://desktop.arcgis.com/en/arcmap/latest/analyze/arcpy-mapping/datadrivenpages-class.htm.
- Esri. (t.t.). *Error handling with Python*. ArcGIS Pro documentation. https://pro.arcgis.com/en/pro-app/3.5/arcpy/get-started/error-handling-with-python.htm.
- Esri. (t.t.). *Get Count*. ArcGIS Pro documentation. https://pro.arcgis.com/en/pro-app/3.5/tool-reference/data-management/get-count.htm.
- Esri. (t.t.). *Installing Python for ArcGIS Pro*. ArcGIS Pro documentation. https://doc.esri.com/en/arcgis-pro/latest/arcpy/get-started/installing-python-for-arcgis-pro.html.
- Esri. (t.t.). *Introduction to arcpy.mp*. ArcGIS Pro documentation. https://doc.esri.com/en/arcgis-pro/latest/arcpy/mapping/introduction-to-arcpy-mp.html.
- Esri. (t.t.). *Layout class*. ArcGIS Pro documentation. https://doc.esri.com/en/arcgis-pro/latest/arcpy/mapping/layout-class.html.
- Esri. (t.t.). *ListRasters*. ArcGIS Pro documentation. https://doc.esri.com/en/arcgis-pro/latest/arcpy/functions/listrasters.html.
- Esri. (t.t.). *Map class*. ArcGIS Pro documentation. https://doc.esri.com/en/arcgis-pro/latest/arcpy/mapping/map-class.html.
- Esri. (t.t.). *MapSeries class*. ArcGIS Pro documentation. https://doc.esri.com/en/arcgis-pro/latest/arcpy/mapping/mapseries-class.html.
- Esri. (t.t.). *Python window*. ArcGIS Pro documentation. https://doc.esri.com/en/arcgis-pro/latest/arcpy/get-started/python-window.html.
- Esri. (t.t.). *SearchCursor class*. ArcGIS Pro documentation. https://pro.arcgis.com/en/pro-app/3.5/arcpy/data-access/searchcursor-class.htm.
- Esri. (t.t.). *The Python window*. ArcMap Documentation. https://desktop.arcgis.com/en/arcmap/latest/analyze/executing-tools/what-is-the-python-window-.htm.
- Esri. (t.t.). *What is ArcPy?* ArcGIS Pro documentation. https://doc.esri.com/en/arcgis-pro/latest/arcpy/get-started/what-is-arcpy-.html.

Tabel Semua Skrip

Tabel ini mendaftar setiap skrip yang dibahas di buku, lengkap dengan seri asal dan nomor bab. Bila Anda memegang buku seri GIS, gunakan tabel ini untuk mencari bab yang membedah sebuah skrip.

Seri	Skrip	Bab	Bagian
B2	`b2_01_atribut.py`	6, 15	B
B2	`b2_02_seleksi.py`	15	B
B2	`b2_03_digitasi.py`	5, 8, 15	B
B2	`b2_04_topologi.py`	7, 15	B
B2	`b2_05_kalkulasi.py`	7, 15	B
B2	`b2_06_join.py`	7, 15	B
B2	`b2_07_laporan.py`	7, 9, 15	B
B3	`b3_01_info.py`	16	B
B3	`b3_02_komposit.py`	16	B
B3	`b3_03_dem.py`	16	B
B3	`b3_04_clip.py`	16	B
B3	`b3_05_reklas.py`	16	B
B3	`b3_06_ndvi.py`	16	B
B3	`b3_07_proyek.py`	16	B
B4	`skrip_4_1_kategori.py`	17	B
B4	`skrip_4_2_klasifikasi.py`	17	B
B4	`skrip_4_2b_raster.py`	17	B
B4	`skrip_4_3_label.py`	17	B
B4	`skrip_4_4_layout.py`	17	B
B4	`skrip_4_5_ekspor.py`	17	B
B4	`skrip_4_6_jembatan.py`	17	B
I1	`1_1_cek_data.py`	18	B
I1	`2_1_buffer_sempadan.py`	18	B
I1	`3_1_overlay.py`	18	B
I1	`4_1_dissolve_rekap.py`	18	B
I1	`5_1_spatial_join.py`	18	B
I1	`6_1_buat_model.py`	18	B
I1	`7_1_laporan_terpadu.py`	18	B
I2	`i2_01_aljabar_raster.py`	19	B

Seri	Skrip	Bab	Bagian
I2	i2_02_turunan_dem.py	19	B
I2	i2_03_kelas_lereng.py	19	B
I2	i2_04_tak_terbimbing.py	12, 19	B
I2	i2_05_terbimbing.py	19	B
I2	i2_07_terpadu.py	19	B
I3	i3_01_statistik_gnss.py	20	B
I3	i3_02_buat_geopackage.py	20	B
I3	i3_03_domain_geopackage.py	20	B
I3	i3_04_formulir_widget.py	20	B
I3	i3_05_pagar_di_berkas.py	20	B
I3	i3_06_cek_kualitas.py	20	B
I3	i3_07_sql_dasar.py	20	B
I3	i3_08_ekspresi_kualitas.py	20	B
I3	i3_09_gabung_dan_bentrok_id.py	20	B
I4	b1_00_python_dasar.py	4, 5, 6, 7, 14	B
I4	b1_01_muat_layer.py	14	B
I4	b1_02_baca_atribut.py	14, 32	B
I4	b1_03_simpan_proyek.py	14	B
I4	b1_arcpy_arcmap.py	31	C
I4	b1_arcpy_pro.py	31	C
I4	b2_02_rantai_vektor.py	21, 32	B
I4	b2_03_raster_alat.py	21	B
I4	b2_arcpy_arcmap.py	31	C
I4	b2_arcpy_pro.py	31	C
I4	b3_01_batch_ndvi.py	21	B
I4	b3_02_batch_per_kph.py	21, 32	B
I4	b3_03_batch_latar_belakang.py	21	B
I4	b3_arcpy_arcmap.py	31	C
I4	b3_arcpy_pro.py	31	C
I4	b4_01_galat_dasar.py	10, 22	B
I4	b4_02_batch_tahan_galat.py	22	B
I4	b4_03_galat_alat.py	22	B
I4	b4_arcpy_arcmap.py	31	C
I4	b4_arcpy_pro.py	31	C

Seri	Skrip	Bab	Bagian
I4	b5_01_ekspresi_python.py	22	B
I4	b5_02_ekspresi_di_alat.py	22	B
I4	b6_01_buat_model.py	23	B
I4	b7_01_proyek_terpadu.py	23	B
I4	b7_02_tugas_ringkasan.py	23	B
I4	b7_03_ceklis_masuk_m1.py	23	B
I4	b7_arcpy_arcmap.py	31	C
I4	b7_arcpy_pro.py	31	C
M1	bab1_skrip_1_1_arcmap.py	30	C
M1	bab1_skrip_1_1_pro.py	30	C
M1	bab2_skrip_2_1_arcmap.py	30	C
M1	bab2_skrip_2_1_pro.py	30	C
M1	bab3_skrip_3_1_arcmap.py	30	C
M1	bab3_skrip_3_1_pro.py	30	C
M1	bab6_skrip_6_1_arcmap.py	30	C
M1	bab6_skrip_6_1_pro.py	30	C
M1	skrip_1_1.py	24	B
M1	skrip_1_1_alternatif.py	24	B
M1	skrip_2_1.py	24	B
M1	skrip_3_1.py	24	B
M1	skrip_3_1_alternatif.py	24	B
M2	_inisialisasi.py	25	B
M2	m2_01b_impor.py	25	B
M2	m2_01c_qgis_muat.py	25	B
M2	m2_02c_silang_geopackage.py	25	B
M2	m2_02d_qgis_setara.py	25	B
M2	m2_03a_rute_qgis.py	25	B
M2	m2_03c_silang_jaringan.py	25	B
M2	m2_04a_hidrologi.py	25	B
M2	m2_05a_interpolasi.py	25	B
M2	m2_06a_kesesuaian.py	25	B
M2	m2_07d_cadangan.sh	25	B
M3	1_1_susun_ndvi.py	26	B
M3	1_2_profil_waktu.py	26	B

Seri	Skrip	Bab	Bagian
M3	1_3_atur_waktu.py	26	B
M3	2_1_masker_awan.py	26	B
M3	2_2_komposit_median.py	26	B
M3	2_3_isi_lubang.py	26	B
M3	3_1_metrik_deret.py	26	B
M3	4_1_deforestasi_dua_waktu.py	26	B
M3	4_2_titik_patah.py	26	B
M3	5_1_dnbr.py	26	B
M3	5_2_titik_panas.py	26	B
M3	6_1_fitur_deret.py	26	B
M3	6_2_latih_klasifikasi.py	26	B
M3	7_1_validasi.py	26	B
M3	hutan_mini.py	11, 12, 26	B
M3	m3_umum.py	26	B
M4	buat_dasbor.py	27	B
M4	buat_peta_web.py	27	B
M4	dsm_ke_obj.py	27	B
M4	kemas_plugin.py	27	B
M4	kerapatan_hotspot.py	11, 27	B
M4	ringkas_kph.py	27	B
M4	tambah_luas_ha.py	11, 27	B

PYTHON UNTUK GIS

Dari Nol sampai Bisa Berimprovisasi

Buku ini mengajak Anda menulis perintah untuk peta. Anda tidak perlu pernah memrogram sebelumnya, dan tidak perlu jago matematika.

Setelah selesai, Anda bisa membaca skrip GIS milik orang lain, mengubahnya untuk data Anda sendiri, dan menulis skrip baru dari kerangka yang sudah Anda kenal.

Tujuannya bukan menghafal, melainkan paham polanya.

ISI BUKU

Bagian A

Python dari nol: memasang, menjalankan, dan semua dasar bahasanya (Bab 1-13).

Bagian B

PyQGIS: sekitar 100 skrip QGIS, dibedah satu per satu (Bab 14-28).

Bagian C

ArcPy: skrip ArcGIS Pro dan ArcMap 10.8, khusus Windows (Bab 29-32).

CARA BUKU INI MENGAJAR

- Studi kasus dari kantor pengelola hutan fiktif
- Skrip dibedah baris demi baris, dengan bagian yang boleh diubah
- Cek paham, kesalahan umum, dan latihan variasi di tiap bab

Versi web, skrip, dan unduhan

badarmubarok.id/buku-gis/python/

Pindai kode di samping dengan ponsel.

Penulis: Badar Mubarok Yogaswara