SERI PYTHON DARI NOL

Python dari Nol sampai Sangat Advanced

Jilid 2 · Intermediate

Bab 16–43

Badar Mubarok Yogaswara

Daftar Isi

Python dari Nol sampai Sangat Advanced 13

Tentang Buku Ini 15
Untuk siapa buku ini? 15
Apa yang dibahas, dan apa yang tidak 15
Asumsi tentang Anda 15
Cara membaca tulisan di buku ini 15
Ikon di buku ini 16
Peta seri: 13 seri dalam tiga tingkat 16
Pembagian tiga jilid 17
Cara membaca: berurutan atau melompat 18
Di luar buku 18
Versi Python yang dipakai 18
Catatan jujur: apa yang diuji dan tidak diuji 18
Siap? 19

Bab 16 — OOP: Membuat Cetakan untuk Data Lapangan 21
Class dan objek 21
Tiap objek punya data sendiri 22
Inheritance: anak mewarisi sifat induk 23
Polymorphism: satu perintah, banyak cara 24
Encapsulation: menjaga data 25
Dunder method: mengajari objek bergaul 26
dataclass: kelas data tanpa banyak tulisan 27
Proyek Bersambung I1: data lapangan menjadi kelas 28
Latihan 30
Ringkasan Bab 30

Bab 17 — Environment dan Dependency: Satu Kamar Python untuk Tiap Proyek 33
venv: membuat kamar baru 33
pip dan requirements.txt 35
pyproject.toml: kartu identitas proyek 36
uv: pengelola paket generasi baru 37
Proyek Bersambung I1: memasang inventaris 39
Poetry: pesaing uv 39
Anaconda dan conda 40

Latihan . . . 41
Ringkasan Bab . . . 42

Bab 18 — Testing Dasar: Memeriksa Kode Otomatis dengan pytest . . . 43
assert: "pastikan ini benar" . . . 43
pytest dan tes pertama . . . 44
Saat tes gagal . . . 45
Menata folder tes . . . 45
Fixture, parametrize, dan raises . . . 46
Menjalankan sebagian tes . . . 48
Proyek Bersambung I1: tes untuk paket inventaris . . . 49
Latihan . . . 49
Ringkasan Bab . . . 50

Bab 19 — Kualitas Kode: Rapi, Tercatat, dan Terdokumentasi . . . 51
PEP 8: tata bahasa kode Python . . . 51
ruff: pemeriksa sekaligus perapi . . . 52
ruff format: merapikan tampilan . . . 53
ruff check –fix: perbaikan otomatis . . . 54
pre-commit: penjaga pintu otomatis . . . 55
Logging: catatan harian program . . . 57
Logging di paket inventaris . . . 58
Debugging: mencari bug dengan teliti . . . 59
Dokumentasi proyek: MkDocs dan Sphinx . . . 60
Proyek Bersambung I1: merapikan inventaris . . . 61
Latihan . . . 63
Ringkasan Bab . . . 63

Bab 20 — Git, GitHub, dan Alur Kerja Proyek: Mesin Waktu untuk Kode Anda . . . 65
Konsep dasar git . . . 65
Melihat perbedaan dan membatalkan . . . 66
Branch dan merge . . . 67
Konflik: bila dua orang mengubah baris yang sama . . . 68
.gitignore dan menjaga rahasia . . . 69
GitHub: git untuk bekerja bersama . . . 71
Alur kerja harian . . . 71
GitHub Actions: pemeriksaan otomatis di server . . . 72
Proyek Bersambung I1: semua bergabung . . . 73
Latihan . . . 74
Ringkasan Bab . . . 75
Penutup Seri I1 . . . 75

Bab 21 — Bekerja dengan API: Meminta Data dari Komputer Lain . . . 77
HTTP dan JSON dalam lima menit . . . 77
Server tiruan: dapur milik kita sendiri . . . 78
Permintaan pertama dengan requests . . . 78
JSON dan parameter . . . 78
httpx: saudara requests . . . 79
Autentikasi: menunjukkan "kartu anggota" . . . 79
Ketika keadaan tidak mulus . . . 80
Kode status dan raise_for_status() . . . 80
Timeout: jangan menunggu selamanya . . . 81
Coba ulang dengan jeda . . . 81
Satu panggilan ke API nyata: Open-Meteo . . . 83
Proyek Bersambung I2: mengambil data cuaca dan harga . . . 84
Latihan . . . 84
Ringkasan Bab . . . 85

Bab 22 — Regex dan Pengolahan Teks: Menggali Data dari Tulisan Berantakan . . . 87
Mencari: search, match, findall . . . 87
Bahan-bahan pola . . . 88
Grup: mengambil bagian-bagian . . . 89
Mengganti dan memisah . . . 90
compile, flags, dan VERBOSE . . . 90
Jebakan: serakah dan lupa None . . . 91
Proyek Bersambung I2: membersihkan catatan lapangan . . . 92
Latihan . . . 93
Ringkasan Bab . . . 93

Bab 23 — Pengantar numpy dan pandas: Tabel Pintar untuk Data Lapangan . . . 95
numpy: array . . . 95
Operasi sekaligus . . . 96
pandas: DataFrame . . . 97
Membaca dan menulis file . . . 98
Memilih dan menyaring . . . 98
Agregasi: meringkas per kelompok . . . 99
Menggabung tabel dan data kosong . . . 100
Proyek Bersambung I2: menggabungkan lima sektor, cuaca, dan harga . . . 101
Latihan . . . 103
Ringkasan Bab . . . 103

Bab 24 — Database: Lemari Arsip yang Tidak Pernah Lupa . . . 105
SQLite: basis data dalam satu berkas . . . 105

Jangan menempel teks pengguna ke SQL . . . 106
Mengubah, menghapus, dan transaksi . . . 107
Dua tabel: JOIN dan GROUP BY . . . 108
pandas dan SQL . . . 108
SQLAlchemy: satu cara untuk banyak basis data . . . 109
PostgreSQL: basis data tingkat server . . . 111
Proyek Bersambung I2: menyimpan data gabungan . . . 112
Latihan . . . 113
Ringkasan Bab . . . 113

Bab 25 — Data Science dan Analisis: Dari Tabel Menjadi Gambar dan Dugaan . . . 115
pandas lanjut: kolom turunan dan kategori . . . 115
Data berurutan waktu . . . 116
numpy lanjut . . . 117
matplotlib: menggambar . . . 118
scikit-learn: model pertama . . . 119
Jupyter Notebook dan Anaconda . . . 121
Proyek Bersambung I2: dari basis data ke grafik . . . 122
Latihan . . . 125
Ringkasan Bab . . . 125
Penutup Seri I2 . . . 126

Bab 26 — Statistika dengan Python: Dari Angka Menjadi Keputusan . . . 127
Statistika deskriptif: merangkum tanpa menipu . . . 127
Peluang dan distribusi . . . 128
Sampling dan selang kepercayaan . . . 129
Uji hipotesis: dua kelompok . . . 131
Chi-square: dua kategori . . . 132
ANOVA: tiga kelompok atau lebih . . . 132
Dua jebakan p-value . . . 134
Korelasi bukan sebab-akibat . . . 135
Visualisasi statistik dengan seaborn . . . 136
Proyek Bersambung I3: uji beda pertumbuhan antar-plot dan antar-varietas . . . 137
Latihan . . . 138
Ringkasan Bab . . . 138

Bab 27 — Regresi dan Pemodelan Kuantitatif: Membuat Model dari Data . . . 141
Regresi linear sederhana: persamaan alometrik pohon . . . 141
Regresi berganda dan diagnostik . . . 143
Regresi logistik: hasilnya ya atau tidak . . . 144

GLM: data cacah dengan Poisson . . . 145
Ridge dan Lasso: bila penjelas banyak dan saling mirip . . . 146
Regresi non-linear: kurva pertumbuhan . . . 147
Model efek campuran: data berkelompok . . . 148
Deret waktu dan peramalan dengan ARIMA . . . 149
Regresi spasial: GWR . . . 150
Simulasi Monte Carlo: menghitung ketidakpastian . . . 152
Optimisasi: mencari pilihan terbaik . . . 153
Model sistem dinamis dan analisis sensitivitas . . . 154
Proyek Bersambung I3: model untuk sistem informasi kita . . . 156
Latihan . . . 157
Ringkasan Bab . . . 157

Bab 28 — Metode Kualitatif dan Campuran: Mendengar Cerita, Lalu Menghitungnya . . . 159
Kualitatif, kuantitatif, atau campuran? . . . 159
Data wawancara tiruan kita . . . 161
Pengodean tema . . . 162
Analisis isi: frekuensi kata dan KWIC . . . 163
TF-IDF: kata yang khas . . . 165
Topic modeling: topik dari tumpukan teks . . . 166
Sentimen berbasis kamus . . . 167
Embedding: kata menjadi titik di peta makna . . . 168
Mengubah data kualitatif menjadi kuantitatif . . . 169
Validitas dan reliabilitas . . . 171
Proyek Bersambung I3: analisis wawancara dengan mixed methods . . . 172
Latihan . . . 173
Ringkasan Bab . . . 173

Bab 29 — Riset yang Dapat Direproduksi: Analisis yang Bisa Diulang Siapa Saja . . . 175
Notebook Jupyter yang rapi . . . 176
Quarto: laporan yang dibuat dari kode . . . 177
Lingkungan terkunci . . . 178
Snakemake: alur analisis yang tahu apa yang perlu diulang . . . 180
DVC: versi untuk data . . . 182
Berbagi data dan kode . . . 183
Proyek Bersambung I3: analisis yang bisa diulang . . . 185
Latihan . . . 186
Ringkasan Bab . . . 186
Penutup Seri I3 . . . 186

Bab 30 — Iterator dan Generator: Membaca Satu per Satu, Tanpa Menumpuk . . . 187
Apa yang dikerjakan `for`: protokol iterasi . . . 187
Iterator buatan sendiri dengan kelas . . . 188
Generator: cara pendek membuat iterator . . . 189
Ekspresi generator dan bukti hemat memori . . . 190
Generator lebih jauh: `itertools`, `yield from`, dan jebakan . . . 191
Merangkai generator menjadi pipa . . . 191
Proyek Bersambung I4: membaca berkas data besar . . . 192
Latihan . . . 194
Ringkasan Bab . . . 194

Bab 31 — Decorator dan Closure: Membungkus Fungsi dengan Tambahan . . . 195
Fungsi adalah objek . . . 195
Closure: fungsi yang mengingat . . . 196
Decorator pertama . . . 197
`functools.wraps`: menjaga identitas . . . 197
Decorator yang punya argumen . . . 198
Decorator siap pakai dan jebakan . . . 199
Proyek Bersambung I4: log dan coba-ulang . . . 200
Latihan . . . 202
Ringkasan Bab . . . 202

Bab 32 — Context Manager: Membuka, Memakai, dan Pasti Menutup . . . 203
`with` pada berkas . . . 203
Di balik layar: `__enter__` dan `__exit__` . . . 204
Cara singkat: `@contextmanager` . . . 205
`__exit__` boleh menelan galat . . . 206
Alat siap pakai di `contextlib` . . . 206
Jebakan: `with` pada objek yang bukan context manager . . . 207
Proyek Bersambung I4: menulis laporan dengan aman . . . 207
Latihan . . . 209
Ringkasan Bab . . . 209

Bab 33 — Type Hint: Memberi Label pada Data, Memeriksa Sebelum Berjalan . . . 211
Anotasi dasar . . . 211
Tipe untuk koleksi, pilihan, dan fungsi . . . 212
Generics: tipe yang mengikuti isi . . . 213
`TypedDict`, `Literal`, dan `Protocol` . . . 214
Menjalankan mypy sungguhan . . . 215

Pyright, alat pemeriksa lain . . . 216
Python 3.14: anotasi dievaluasi belakangan . . . 216
Proyek Bersambung I4: memberi tipe pada kode proyek . . . 217
Latihan . . . 218
Ringkasan Bab . . . 218

Bab 34 — Validasi Data dan Konfigurasi: Penjaga Pintu dengan Pydantic . . . 221
dataclass rapi, tetapi tidak menjaga . . . 221
Model Pydantic . . . 222
Pesan galat yang jelas . . . 223
Aturan tambahan: `Field` dan validator . . . 223
Pengaturan aplikasi: variabel lingkungan dan `.env` . . . 225
JSON, daftar model, dan skema . . . 226
Dataclass atau Pydantic? . . . 227
Kaitan dengan FastAPI dan agent AI . . . 227
Proyek Bersambung I4: validasi data lapangan lima sektor . . . 227
Latihan . . . 229
Ringkasan Bab . . . 229

Bab 35 — Async dan Concurrency: Mengerjakan Banyak Hal Sekaligus . . . 231
Dua jenis pekerjaan . . . 231
`asyncio`: satu pekerja yang pandai bergantian . . . 232
`gather`: banyak pekerjaan menunggu bersama . . . 232
`TaskGroup`, batas waktu, dan pembatas jumlah . . . 233
Thread: untuk pekerjaan menunggu pada kode biasa . . . 234
Proses: untuk pekerjaan menghitung . . . 235
GIL: mengapa thread tidak selalu mempercepat . . . 236
Mode *free-threaded* . . . 237
Proyek Bersambung I4: mengunduh banyak data cuaca secara async . . . 237
Latihan . . . 240
Ringkasan Bab . . . 241
Penutup Seri I4 . . . 241

Bab 36 — Web Development: Membuka Warung Data di Internet . . . 243
Cara kerja web dalam satu paragraf . . . 243
Flask: warung paling kecil . . . 244
REST API dengan Flask . . . 245
FastAPI: tipe data yang bekerja untuk Anda . . . 246
Tubuh permintaan dengan model Pydantic . . . 247
Dokumentasi yang tertulis sendiri . . . 248
Pengantar Django . . . 249

Server asli, sebentar saja . . . 250
Proyek Bersambung I5: API data unit usaha . . . 251
Latihan . . . 252
Ringkasan Bab . . . 252

Bab 37 — Python di Browser: Tanpa Server, Tanpa Pasang . . . 253
WebAssembly, singkatnya . . . 253
Pyodide: Python di dalam WebAssembly . . . 254
Pyodide langsung di halaman HTML . . . 255
PyScript: Python di dalam HTML . . . 255
JupyterLite . . . 256
Kapan cocok, kapan tidak . . . 256
Proyek Bersambung I5: ringkasan luas petak yang tampil di browser . . . 257
Latihan . . . 258
Ringkasan Bab . . . 258

Bab 38 — GIS 1: Data Vektor: Peta yang Bisa Dihitung . . . 259
Titik, garis, poligon: Shapely . . . 259
GeoPandas: tabel yang punya bentuk . . . 260
CRS: derajat atau meter? . . . 261
Luas petak dan jarak kolam ke sungai . . . 262
Buffer dan overlay . . . 263
Spatial join: menggabung menurut letak . . . 264
Memasang GIS: pip atau conda-forge? . . . 266
Proyek Bersambung I5: peta lokasi usaha . . . 266
Latihan . . . 267
Ringkasan Bab . . . 267

Bab 39 — GIS 2: Data Raster: Foto Udara yang Bisa Dihitung . . . 269
Membaca raster dengan Rasterio . . . 271
NDVI: mengukur kehijauan . . . 271
Memotong citra dengan batas lahan . . . 273
Reproject dan resample . . . 275
Statistik zonal . . . 275
Proyek Bersambung I5: NDVI per petak . . . 276
Latihan . . . 277
Ringkasan Bab . . . 277

Bab 40 — GIS 3: Peta dan Integrasi: PostGIS dan QGIS . . . 279
Peta statis dengan Matplotlib . . . 279
Peta interaktif dengan folium . . . 280
PostGIS: basis data yang mengerti peta . . . 281
Membuat tabel dan memuat data . . . 282
Query spasial . . . 283

PostGIS dan GeoPandas . . . 284
PyQGIS: mengendalikan QGIS dari Python . . . 285
Algoritma Processing, gaya, dan menggambar peta . . . 286
QGIS 4 berbeda dari QGIS 3 . . . 288
Alur kerja Python dan QGIS . . . 289
Proyek Bersambung I5: data lokasi usaha di dua tempat . . . 289
Latihan . . . 290
Ringkasan Bab . . . 290

Bab 41 — Automasi dan Scripting: Biarkan Python Bekerja Sendiri . . . 291
subprocess: memanggil program lain . . . 291
Otomasi berkas: merapikan folder . . . 293
Otomasi infrastruktur: memeriksa layanan . . . 294
Menjadwalkan skrip . . . 294
Web scraping: etika dulu, kode kemudian . . . 296
BeautifulSoup: membaca HTML . . . 297
robots.txt di kode . . . 298
Scraper yang sopan . . . 299
Scrapy: kerangka kerja untuk scraping besar . . . 301
Proyek Bersambung I5: otomasi laporan mingguan . . . 303
Latihan . . . 304
Ringkasan Bab . . . 304

Bab 42 — Antarmuka: Perintah, Jendela, dan Dasbor . . . 305
CLI dengan argparse . . . 305
Bila pengguna salah ketik . . . 306
Subperintah . . . 308
CLI dengan Typer . . . 308
Menguji CLI . . . 311
GUI sederhana dengan tkinter . . . 312
Dasbor data pengantar . . . 314
Proyek Bersambung I5: CLI untuk unit usaha . . . 315
Latihan . . . 316
Ringkasan Bab . . . 316

Bab 43 — Aplikasi Data Cepat dan Dasbor: Dari Skrip ke Layar Orang Lain . . . 317
Streamlit: dasbor dari satu skrip . . . 317
Gradio: demo fungsi dan model . . . 320
Dash: dasbor yang sepenuhnya Anda atur . . . 322
Peta interaktif di dasbor . . . 323
HTMX: web ringan tanpa kerangka besar . . . 325
FastAPI: pintu untuk program lain . . . 326
Menyajikan aplikasi kepada pengguna . . . 328

Proyek Bersambung I5 (penutup): satu data, banyak pintu **330**
Latihan **331**
Ringkasan Bab **331**
Penutup Seri I5 . . . . . **332**

Python dari Nol sampai Sangat Advanced

Dari benar-benar awam sampai tingkat ahli, dengan contoh dari hutan, kebun, sawah, kolam, dan kandang.

Selamat datang. Buku ini mengajarkan Python mulai dari pertanyaan paling dasar ("apa itu komputer?") sampai topik tingkat ahli (internals CPython, GPU, sistem terdistribusi). Isinya **68 bab** dalam **tiga tingkat**, dengan satu proyek yang tumbuh dari bab ke bab: *Sistem Informasi Hutan, Kebun, Pertanian, Perikanan, Peternakan.*

Jilid	Tingkat	Bab
1	Basic	1–15
2	Intermediate	16–43
3	Advanced	44–68

Semua kode dijalankan sungguhan dengan **Python 3.14.8** dan keluaran yang tampil adalah hasil aslinya, kecuali bagian yang ditandai terang-terangan sebagai tidak dijalankan. Penulis: **Badar Mubarok Yogaswara**.

Mulai dari mana? Baca dulu bagian berikutnya, Tentang Buku Ini: untuk siapa buku ini, cara membacanya, dan apa yang diuji dan tidak diuji. Pembaca yang sudah mahir boleh langsung melompat; peta jalurnya ada di Lampiran E.

Tentang Buku Ini

Selamat datang! Buku ini bernama **Python dari Nol sampai Sangat Advanced**, ditulis oleh **Badar Mubarok Yogaswara**. Kalimat "dari nol" tidak main-main: Anda tidak perlu pernah memprogram sebelumnya. Kalimat "sangat advanced" juga tidak main-main: di ujung buku Anda akan menyentuh bagian dalam Python, GPU, AI generatif, dan sistem terdistribusi.

Bayangkan buku ini sebagai **peta perjalanan** dari pondok di tepi kebun sampai ke menara pengamat di puncak hutan. Anda tidak harus berlari. Cukup melangkah, satu bab demi satu bab.

Untuk siapa buku ini?

- **Pemula sungguhan**: belum pernah menulis satu baris kode pun. Bab 1 mulai dari pertanyaan "apa itu komputer?".
- **Orang lapangan**: rimbawan, penyuluh, petani, pembudidaya ikan, peternak, mahasiswa, atau peneliti yang ingin mengolah data sendiri tanpa menunggu orang lain.
- **Pemrogram bahasa lain** yang ingin pindah ke Python: Anda boleh melompat (lihat bagian "Cara membaca" di bawah).
- **Profesional data, GIS, dan AI** yang ingin merapikan dasar dan melihat bagaimana semuanya menyatu dalam satu proyek.

Apa yang dibahas, dan apa yang tidak

Yang dibahas: dasar bahasa Python, kode yang rapi dan teruji, data dan internet, statistik dan pemodelan, aplikasi web, peta (GIS), dasbor, rekayasa perangkat lunak, data besar, machine learning, AI generatif dan agent AI, sampai Python tingkat ahli. Semuanya diikat oleh satu **proyek bersambung**: *Sistem Informasi Hutan, Kebun, Pertanian, Perikanan, Peternakan.* Contoh-contohnya bergantian mengambil dari lima sektor itu.

Yang **tidak** dibahas: buku ini bukan buku kehutanan, pertanian, perikanan, atau peternakan. Angka dan contoh lapangan hanyalah **bahan latihan**. Buku ini juga bukan kamus lengkap semua pustaka Python; kami memilih yang paling berguna dan menunjukkan jalan ke dokumentasi resminya.

Asumsi tentang Anda

Kami hanya berasumsi tiga hal: Anda bisa memakai komputer (membuka berkas, mengetik, menyalin), Anda penasaran, dan Anda bersedia **mencoba sendiri**. Membaca tentang memprogram itu seperti membaca tentang berenang: berguna, tetapi baru terasa setelah masuk ke air.

Cara membaca tulisan di buku ini

Ada beberapa kebiasaan penulisan yang tetap sepanjang buku:

- **Kode dan nama berkas** ditulis dengan huruf `monospace` (huruf berlebar sama), misalnya `print()` dan `data.csv`.
- Setiap contoh kode adalah **skrip** (berkas program) yang punya **nama di atas blok kodenya**. Nama itu juga nama berkas di folder `kode/` pada unduhan Anda.
- **Keluaran selalu asli.** Hasil yang tercetak di buku didapat dengan benar-benar menjalankan skrip itu memakai **Python 3.14.8**. Kami tidak mengetik atau mengedit hasilnya dengan tangan. Contoh di bawah menunjukkannya:

```
# Tentang Buku Ini - Memeriksa versi Python yang menjalankan contoh
import sys
```

```
print("Versi Python:", sys.version.split()[0])
print("Halo, Lahan! Ini keluaran asli, bukan ketikan tangan.")
```

```
Versi Python: 3.14.8
Halo, Lahan! Ini keluaran asli, bukan ketikan tangan.
```

- Skrip siap salin ada di folder `kode/babNN/` (misalnya `kode/bab10/`). Di versi web, tiap bab juga punya **zip unduhan** berisi semua skripnya.
- Istilah teknis dijelaskan **di tempat pertama kali muncul**, dicetak **tebal**.
- Nama variabel dan teks di contoh memakai Bahasa Indonesia agar mudah dibaca. Pesan error dari Python tetap berbahasa Inggris, dan bab-bab awal mengajari cara membacanya.

Ikon di buku ini

Empat jenis kotak muncul di mana-mana. Inilah wujud aslinya:

> **INGAT**
>
> Hal penting yang perlu Anda ingat terus. Kalau hanya sempat membaca kotak, bacalah yang ini.

> **TIPS**
>
> Cara yang lebih mudah, lebih cepat, atau lebih rapi untuk mengerjakan sesuatu.

> **AWAS**
>
> Jebakan yang sering membuat pemula tersandung. Kami menunjukkan error aslinya agar Anda mengenalinya.

> **BAGIAN TEKNIS (boleh dilewati)**
>
> Penjelasan lebih dalam bagi yang penasaran. Aman dilewati; Anda tetap bisa melanjutkan tanpa membacanya. Klik judul kotak ini untuk membukanya.

Peta seri: 13 seri dalam tiga tingkat

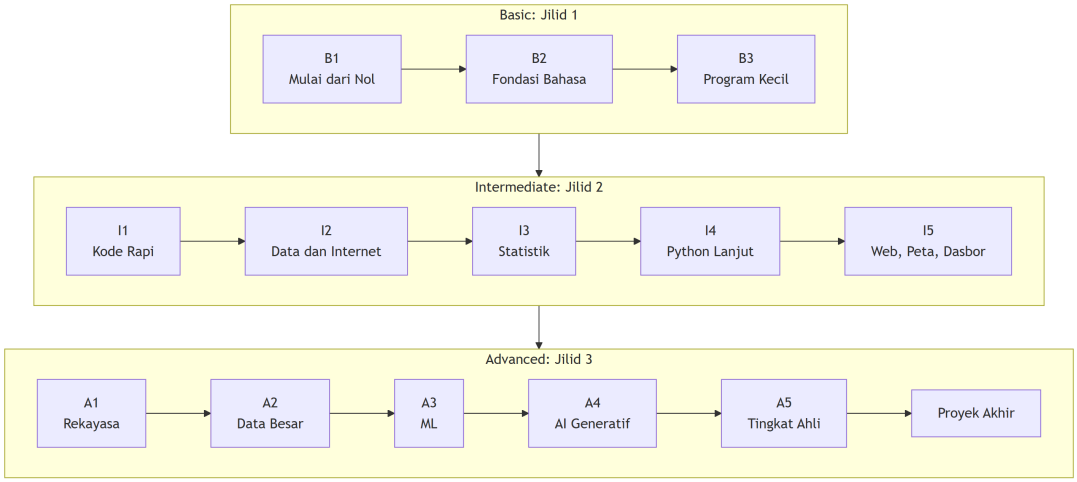
Buku ini berisi **68 bab**. Bab dikelompokkan menjadi **13 seri**, dan seri dikelompokkan menjadi **tiga tingkat**. Tiap seri punya prasyarat, satu proyek bersambung, dan "jembatan" ke seri berikutnya.

Tingkat	Seri	Judul	Bab
Basic	B1	Mulai dari Nol	1–4
Basic	B2	Fondasi Bahasa Python	5–10
Basic	B3	Program Kecil yang Berguna	11–15
Intermediate	I1	Kode yang Rapi dan Terawat	16–20
Intermediate	I2	Data dan Internet	21–25
Intermediate	I3	Statistik dan Pemodelan	26–29
Intermediate	I4	Bahasa Python Tingkat Lanjut	30–35
Intermediate	I5	Aplikasi Pertama: Web, Peta, dan Dasbor	36–43
Advanced	A1	Rekayasa Perangkat Lunak Profesional	44–49

Tingkat	Seri	Judul	Bab
Advanced	A2	Data Skala Besar dan Geospasial Lanjut	50–53
Advanced	A3	Machine Learning untuk Sumber Daya Alam	54–59
Advanced	A4	AI Generatif dan Agent AI	60–63
Advanced	A5	Python Tingkat Ahli	64–68

Setelah Bab 68 ada **Proyek Akhir** (Bab 69), **Bagian Sepuluh** (daftar-daftar sepuluh butir), dan **Lampiran A sampai F** (pemasangan, lembar contekan, glosarium, kunci jawaban, peta jalur belajar, sumber lanjutan).

Buku ini terbit dalam **tiga jilid**: **Jilid 1 (Basic)** memuat Bab 1–15 dan Lampiran A (pemasangan), **Jilid 2 (Intermediate)** memuat Bab 16–43, dan **Jilid 3 (Advanced)** memuat Bab 44–68, Proyek Akhir, Bagian Sepuluh, serta Lampiran A sampai F. Bila Anda memegang satu jilid saja, bab, proyek, dan lampiran yang tidak ada di jilid itu (misalnya Lampiran C glosarium dan Lampiran E peta jalur belajar) ada di Jilid 3.



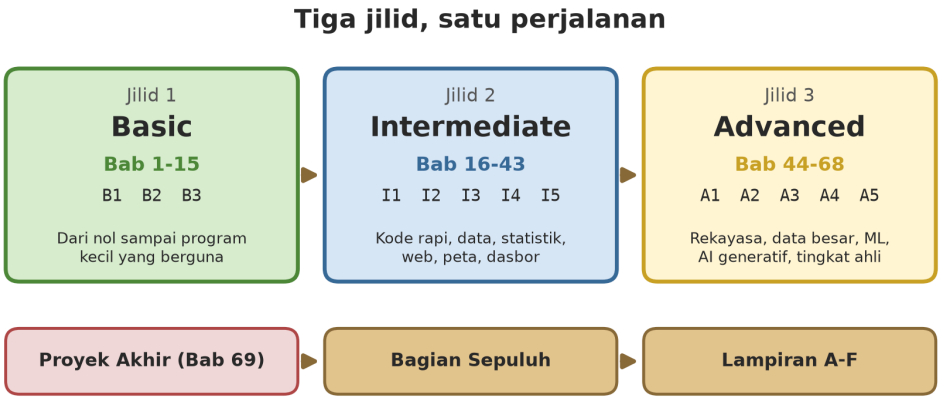
Urutan seri: dari B1 sampai Proyek Akhir.

Pembagian tiga jilid

Satu buku setebal ini nyaman dipegang bila dipecah. Penulis memutuskan (10 Oktober 2026) membaginya menjadi **tiga jilid** yang mengikuti tiga tingkat:

- **Jilid 1, Basic**: Bab 1–15 (seri B1 sampai B3).
- **Jilid 2, Intermediate**: Bab 16–43 (seri I1 sampai I5).
- **Jilid 3, Advanced**: Bab 44–68 (seri A1 sampai A5).

Proyek Akhir, Bagian Sepuluh, dan Lampiran melengkapi ketiganya.



Gambar 0.1: Tiga jilid, satu perjalanan.

Cara membaca: berurutan atau melompat

- **Pemula**: baca **berurutan** dari Bab 1. Tiap bab menyambung dari bab sebelumnya, dan proyek bersambungnya tumbuh sedikit demi sedikit.
- **Sudah bisa memprogram di bahasa lain**: mulailah dari Seri B2 atau B3 untuk mengenal gaya Python, lalu lompat ke seri yang Anda butuhkan. Cek **prasyarat** di awal tiap seri.
- **Ingin langsung ke topik tertentu** (misalnya peta atau AI): lihat tabel di atas dan **Lampiran E (peta jalur belajar)** yang menyusun jalur untuk awam, pemrogram bahasa lain, GIS, data/AI, dan web.
- **Buntu di satu bab?** Jalankan skrip contohnya apa adanya dulu, lihat keluarannya, baru ubah sedikit-sedikit. Mengubah contoh yang sudah jalan adalah cara belajar tercepat.

Di luar buku

- **Versi web**: bisa dibaca di browser, tiap blok kode punya tombol salin.
- **Unduhan skrip**: tiap bab punya zip berisi semua skripnya (tautannya ada di awal bab).
- **Data contoh**: berkas CSV/JSON kecil dibuat oleh skrip itu sendiri dan tersedia di folder `data/` bab terkait.
- **Diagram**: tersedia dalam bentuk gambar di buku; bagian lampiran menyebutkan bentuk lain bila ada.

Versi Python yang dipakai

Buku ini memakai **Python 3.14.8** sebagai acuan (rilis 30 September 2026; menurut jadwal resmi PEP 745 yang kami baca, dukungan keamanan untuk seri 3.14 berlangsung sampai sekitar Oktober 2030). **Python 3.15.0** baru dirilis pada **9 Oktober 2026**; kami memeriksanya di halaman unduhan resmi python.org pada 10 Oktober 2026. Bahasan fitur 3.15 ada sebagai subbagian di **Bab 47**, dan keluaran buku tetap dari 3.14.8.

> TIPS
>
> Anda tidak wajib memakai 3.14.8. Hampir semua contoh berjalan di Python 3.12 ke atas; bagian yang butuh fitur baru kami tandai di babnya. Cek versi Anda dengan `python --version`. Daftar rilis terbaru selalu ada di python.org/downloads, dan **angka versi bisa sudah berubah** saat Anda membaca ini.

Catatan jujur: apa yang diuji dan tidak diuji

Kami ingin Anda tahu persis seberapa jauh buku ini bisa dipercaya:

- **Diuji**: seluruh kode dijalankan di **Windows 11** dengan **Python 3.14.8**, dan keluarannya diambil dari hasil itu. Sebagian skrip yang tidak bergantung fitur khusus 3.14 juga dibandingkan di Python 3.12.

- **Tidak dijalankan**: bila sebuah kode **tidak dijalankan** saat buku dibuat (misalnya butuh internet, akun, atau perangkat yang tidak ada), kami menulisnya terang-terangan di babnya beserta alasannya, dan keluarannya tidak ditampilkan.
- **Perangkat penulis berbeda dengan perangkat Anda.** Bagian yang memakai **GPU**, **QGIS**, **PostgreSQL**, dan **Redis** dijalankan di komputer penulis (GPU NVIDIA dengan VRAM 6 GB, dan pemasangan lokal yang terisolasi). Komputer Anda mungkin berbeda: versi pustaka, driver, dan sistem operasi bisa membuat hasil sedikit lain. Bila begitu, baca pesan errornya dan cocokkan dengan bab terkait.
- **Data tiruan.** Semua data lapangan adalah **data fiktif** buatan skrip. Angka pohon, panen, ikan, dan ternak hanyalah **bahan latihan**. Angka itu **bukan anjuran teknis** (bukan takaran pakan, jadwal tanam, atau harga sebenarnya). Untuk keputusan di lapangan, rujuklah petunjuk teknis resmi dan tenaga ahli.
- **Tautan dan versi bisa berubah.** Halaman web, nama pustaka, dan nomor versi diperiksa pada saat buku ditulis (Oktober 2026). Bila sebuah tautan mati atau versi sudah berganti, cari nama sumbernya di mesin pencari atau di situs resmi proyeknya, dan periksa dokumentasinya sebelum mengikuti perintah lama.

Siap?

Ambil secangkir kopi (atau teh), buka **Bab 1**, dan mulailah. Kesalahan itu normal; pesan error bukan hukuman, melainkan petunjuk jalan. Selamat menanam kode!

Bab 16 — OOP: Membuat Cetakan untuk Data Lapangan

Seri I1 — Kode yang Rapi dan Terawat

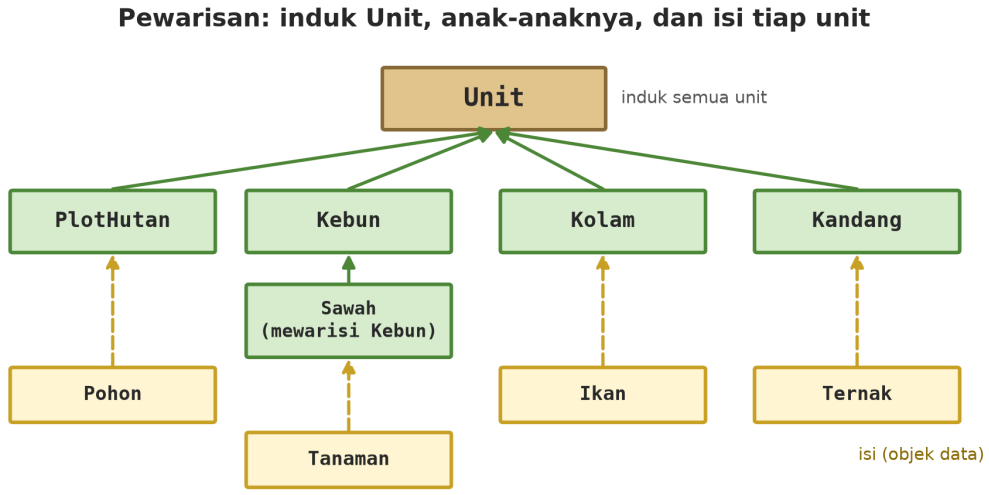
Di bab ini, Anda akan:

- Membuat **class** (cetakan) dan **objek** (hasil cetakan) dengan data dan perilaku sendiri
- Memakai **inheritance** (pewarisan) supaya anak mewarisi sifat induk
- Memahami **encapsulation** (menjaga data) dan **polymorphism** (satu perintah, banyak cara)
- Mengajari objek bergaul dengan Python lewat **dunder method**, lalu menghemat tulisan dengan **dataclass**
- Mengubah data lapangan Seri B menjadi keluarga kelas: induk `Unit` dan anak-anaknya

Semua skrip bab ini ada di folder `kode/bab16/`. Di versi web, Anda juga bisa mengunduh semuanya sekaligus (zip).

Di Seri B, data lapangan Anda tersimpan dalam dict dan list, dan perhitungannya ada di fungsi-fungsi terpisah. Itu berhasil, tetapi semakin besar programnya, semakin sulit menjawab pertanyaan sederhana: *fungsi mana yang cocok untuk data yang mana?* Fungsi `biomassa_kg` hanya masuk akal untuk kolam, tetapi tidak ada yang mencegah Anda memberinya data sapi.

Pemrograman berorientasi objek (*object-oriented programming*, disingkat **OOP**) menyatukan **data** dan **fungsi yang cocok untuknya** dalam satu paket. Analoginya cetakan kue: **class** adalah cetakannya, **objek** adalah kue yang dicetak. Satu cetakan bisa menghasilkan banyak kue, dan tiap kue punya isi sendiri.



Gambar 0.1: Keluarga kelas proyek kita: induk Unit, anak-anaknya, dan data yang disimpan tiap unit.

Class dan objek

```
# Bab 16 - Class dan objek: cetakan dan hasil cetakannya
import math
```

```
class Pohon:
    """Cetakan untuk satu pohon di plot."""

    def __init__(self, jenis, diameter_cm, tinggi_m):
        self.jenis = jenis
        self.diameter_cm = diameter_cm
        self.tinggi_m = tinggi_m

    def luas_penampang_cm2(self):
        return math.pi * (self.diameter_cm / 2) ** 2

    def kategori(self):
        if self.diameter_cm >= 40:
            return "besar"
        if self.diameter_cm >= 30:
            return "sedang"
        return "kecil"


jati = Pohon("Jati", 32.5, 18.0)
meranti = Pohon("Meranti", 45.2, 27.5)

print(jati.jenis, jati.diameter_cm, jati.kategori())
print(meranti.jenis, meranti.diameter_cm, meranti.kategori())
print(f"Luas penampang jati: {jati.luas_penampang_cm2():.1f} cm2")
print(type(jati))
```

```
Jati 32.5 sedang
Meranti 45.2 besar
Luas penampang jati: 829.6 cm2
<class '__main__.Pohon'>
```

- `class Pohon:` membuat cetakan. Nama kelas ditulis dengan huruf besar di awal tiap kata (`PlotHutan`).
- `__init__` adalah metode khusus yang berjalan **otomatis saat objek dibuat**. Di sinilah data awal dipasang. (Dibaca "dunder init": *double underscore*, dua garis bawah di tiap sisi.)
- **`self`** adalah objek yang sedang dikerjakan. `self.jenis = jenis` artinya "simpan nilai ini di dalam objek ini".
- Fungsi di dalam kelas disebut **metode** (*method*). `jati.kategori()` memanggil metode untuk objek `jati`. Python mengisi `self` sendiri.
- `Pohon("Jati", 32.5, 18.0)` membuat satu **objek** (juga disebut *instance*). Luas penampang 829,6 cm² dihitung dari diameter 32,5 cm.

Hasil `type(jati)` menunjukkan bahwa `jati` memang objek kelas `Pohon`.

Tiap objek punya data sendiri

```
# Bab 16 - Setiap objek punya data sendiri
class Kolam:
    jenis_unit = "Kolam"          # milik class: sama untuk semua objek

    def __init__(self, nama, jumlah_ikan):
        self.nama = nama          # milik objek: beda-beda
        self.jumlah_ikan = jumlah_ikan

    def tebar(self, tambahan):
        self.jumlah_ikan = self.jumlah_ikan + tambahan


kolam_a = Kolam("A", 800)
```

```
kolam_b = Kolam("B", 500)
kolam_a.tebar(200)

print(kolam_a.nama, kolam_a.jumlah_ikan)
print(kolam_b.nama, kolam_b.jumlah_ikan)
print(kolam_a.jenis_unit, kolam_b.jenis_unit)
print(isinstance(kolam_a, Kolam), isinstance(kolam_a, str))
```

```
A 1000
B 500
Kolam Kolam
True False
```

Kolam A diisi 800 lalu ditambah 200 ikan, jadi 1000. Kolam B tidak berubah: tetap 500. Dua objek dari cetakan yang sama **tidak saling mengganggu**. Perhatikan juga `jenis_unit`: ia ditulis di tingkat kelas, jadi **dimiliki bersama** oleh semua objek.

⚠ AWAS

Jebakan paling sering saat pertama membuat metode: lupa menulis `self`.

```
# Bab 16 - Jebakan: lupa menulis self
class Pohon:
    def __init__(self, jenis):
        self.jenis = jenis

    def sapa():                    # lupa self!
        print("Halo, saya pohon")


jati = Pohon("Jati")
jati.sapa()
```

```
Traceback (most recent call last):
  File "03_lupa_self.py", line 11, in <module>
    jati.sapa()
    ~~~~~~~~~^^
TypeError: Pohon.sapa() takes 0 positional arguments but 1 was given
```

Pesannya membingungkan: “takes 0 positional arguments but 1 was given” (menerima 0 argumen tetapi diberi 1). Argumen yang dimaksud adalah objek itu sendiri, yang Python kirim otomatis. Tanpa `self` di daftar parameter, tidak ada tempat untuk menerimanya.

Inheritance: anak mewarisi sifat induk

Plot hutan, kebun, kolam, dan kandang sama-sama punya nama dan luas. Menulis ulang itu di tiap kelas itu boros. Solusinya **inheritance**: buat kelas **induk** berisi yang sama, lalu kelas **anak** yang mewarisinya.

```
# Bab 16 - Inheritance: anak mewarisi sifat induk
class Unit:
    """Induk: semua unit punya nama dan luas."""

    def __init__(self, nama, luas_ha):
        self.nama = nama
        self.luas_ha = luas_ha

    def info(self):
        return f"{self.nama} ({self.luas_ha} ha)"
```

```
class PlotHutan(Unit):
    def __init__(self, nama, luas_ha, tegakan):
        super().__init__(nama, luas_ha)     # urus bagian induk dulu
        self.tegakan = tegakan              # lalu bagian khusus anak


class Kebun(Unit):
    pass                                    # tidak menambah apa-apa


plot = PlotHutan("Plot A", 2.5, "Jati")
kebun = Kebun("Kebun Kopi", 1.2)
print(plot.info(), "| tegakan:", plot.tegakan)
print(kebun.info())
print(isinstance(plot, Unit), issubclass(Kebun, Unit), isinstance(kebun, PlotHutan))
```

```
Plot A (2.5 ha) | tegakan: Jati
Kebun Kopi (1.2 ha)
True True False
```

- `class PlotHutan(Unit):` artinya "`PlotHutan` adalah anak dari `Unit`".
- **`super().__init__(...)`** memanggil `__init__` milik induk, supaya bagian induk ikut dikerjakan. Anak lalu menambah bagiannya sendiri (`tegakan`).
- `Kebun` tidak menambah apa-apa, tetapi tetap punya `nama`, `luas_ha`, dan `info()` dari `Unit`.
- `isinstance(plot, Unit)` bernilai `True`: sebuah plot hutan *adalah* sebuah unit. Sebaliknya, kebun bukan plot hutan (`False`).

Polymorphism: satu perintah, banyak cara

Kata ini terdengar menakutkan, padahal idenya sederhana: **anak boleh mengerjakan perintah yang sama dengan cara berbeda**. Anda cukup bilang `ringkasan()`, dan tiap unit tahu sendiri cara menghitungnya.

```
# Bab 16 - Polymorphism: satu perintah, banyak cara
class Unit:
    def __init__(self, nama):
        self.nama = nama

    def hasil(self):
        raise NotImplementedError("anak harus mengisi hasil()")

    def ringkasan(self):
        return f"{self.nama}: {self.hasil()}"


class Kolam(Unit):
    def __init__(self, nama, jumlah, bobot_g):
        super().__init__(nama)
        self.jumlah, self.bobot_g = jumlah, bobot_g

    def hasil(self):
        return f"biomassa {self.jumlah * self.bobot_g / 1000:.1f} kg"


class Kandang(Unit):
    def __init__(self, nama, bobot_sapi_kg):
        super().__init__(nama)
        self.bobot_sapi_kg = bobot_sapi_kg
```

```
    def hasil(self):
        return f"total bobot {sum(self.bobot_sapi_kg):.1f} kg"


daftar = [Kolam("Kolam A", 800, 540.5), Kandang("Kandang 1", [350, 412.5, 389.0])]
for unit in daftar:
    print(unit.ringkasan())      # sama perintahnya, hasilnya menyesuaikan

try:
    Unit("Polos").ringkasan()
except NotImplementedError as e:
    print("Induk saja tidak cukup:", e)
```

```
Kolam A: biomassa 432.4 kg
Kandang 1: total bobot 1151.5 kg
Induk saja tidak cukup: anak harus mengisi hasil()
```

Perulangan `for` memanggil `unit.ringkasan()` tanpa peduli apakah isinya kolam atau kandang. Hitungan kolam (800 × 540,5 g = 432,4 kg) dan kandang (350 + 412,5 + 389,0 = 1151,5 kg) berbeda, tetapi pemanggilnya sama. Induk `Unit` menyerahkan `hasil()` kepada anak dengan `raise NotImplementedError`: "bagian ini wajib diisi anak". Itu sebabnya `Unit("Polos")` gagal dengan pesan yang jelas.

Encapsulation: menjaga data

Encapsulation artinya data dijaga **di dalam** objek, dan akses dari luar lewat pintu yang punya penjaga. Di Python, awalan garis bawah (`_ph_air`) adalah **kesepakatan**: "ini urusan dalam, jangan disentuh dari luar". Penjaga pintunya adalah **`property`**.

```
# Bab 16 - Encapsulation: data dijaga di dalam objek
class Kolam:
    def __init__(self, nama, ph_air):
        self.nama = nama
        self._ph_air = None        # awalan _ : "jangan disentuh dari luar"
        self.ph_air = ph_air       # lewat property, sehingga ikut diperiksa

    @property
    def ph_air(self):
        return self._ph_air

    @ph_air.setter
    def ph_air(self, nilai):
        if not 0 <= nilai <= 14:
            raise ValueError(f"pH harus 0 sampai 14, bukan {nilai}")
        self._ph_air = nilai


kolam = Kolam("A", 7.2)
print("pH awal:", kolam.ph_air)
kolam.ph_air = 6.8
print("pH baru:", kolam.ph_air)

try:
    kolam.ph_air = 17
except ValueError as e:
    print("Ditolak:", e)
print("pH tetap:", kolam.ph_air)
```

```
pH awal: 7.2
pH baru: 6.8
Ditolak: pH harus 0 sampai 14, bukan 17
pH tetap: 6.8
```

Dari luar, `kolam.ph_air` terlihat seperti variabel biasa. Namun tiap kali nilainya diubah, Python menjalankan pemeriksaan di `setter`. pH 17 ditolak dengan `ValueError`, dan nilai lama (6,8) tetap aman.

> **i** INGAT
>
> Python tidak benar-benar mengunci data. Awalan _ hanya kesepakatan antar-programmer. Kekuatan `property` ada pada **pemeriksaan**: data yang salah tidak bisa masuk lewat pintu resmi.

Dunder method: mengajari objek bergaul

Metode ber-garis-bawah-ganda seperti `__init__` disebut **dunder method**. Python memanggilnya otomatis pada saat tertentu. Dengan mengisinya, objek Anda ikut bekerja dengan `print`, `==`, `sorted`, dan `len`.

```python
# Bab 16 - Dunder method: mengajari objek bergaul dengan Python
class Pohon:
    def __init__(self, jenis, diameter_cm):
        self.jenis = jenis
        self.diameter_cm = diameter_cm

    def __repr__(self):                      # tampilan "teknis"
        return f"Pohon({self.jenis!r}, {self.diameter_cm})"

    def __str__(self):                       # tampilan untuk manusia
        return f"{self.jenis} {self.diameter_cm} cm"

    def __eq__(self, lain):
        return (self.jenis, self.diameter_cm) == (lain.jenis, lain.diameter_cm)

    def __lt__(self, lain):                  # supaya bisa diurutkan
        return self.diameter_cm < lain.diameter_cm


class Plot:
    def __init__(self, pohon):
        self.pohon = list(pohon)

    def __len__(self):
        return len(self.pohon)


a, b, c = Pohon("Jati", 32.5), Pohon("Meranti", 45.2), Pohon("Mahoni", 28.0)
print(a)
print(repr(a))
print(a == Pohon("Jati", 32.5), a == b)
print(sorted([a, b, c]))
print(len(Plot([a, b, c])), "pohon")
```

```
Jati 32.5 cm
Pohon('Jati', 32.5)
True False
[Pohon('Mahoni', 28.0), Pohon('Jati', 32.5), Pohon('Meranti', 45.2)]
3 pohon
```

Metode	Dipanggil ketika	Contoh
`__str__`	`print(objek)`	tampilan untuk pembaca
`__repr__`	`repr(objek)`, tampilan di daftar	tampilan untuk programmer
`__eq__`	`a == b`	membandingkan isi, bukan alamat
`__lt__`	`a < b`, `sorted(...)`	mengurutkan menurut diameter
`__len__`	`len(objek)`	jumlah isi

Hasil `sorted` berurutan dari diameter 28,0 ke 32,5 ke 45,2 karena `__lt__` membandingkan diameter.

dataclass: kelas data tanpa banyak tulisan

Kelas yang isinya hanya data (seperti `Pohon`) memaksa kita menulis `__init__`, `__repr__`, dan `__eq__` berulang-ulang. Modul bawaan `dataclasses` membuatkannya otomatis dari **daftar kolom bertipe**.

```
# Bab 16 - dataclass: kelas data tanpa banyak tulisan
from dataclasses import dataclass, field


@dataclass(order=True)
class Pohon:
    jenis: str
    diameter_cm: float
    sehat: bool = True


@dataclass
class Plot:
    nama: str
    pohon: list = field(default_factory=list)    # list baru untuk tiap plot


a = Pohon("Jati", 32.5)
b = Pohon("Mahoni", 28.0, sehat=False)
print(a)                          # __repr__ dibuatkan otomatis
print(a == Pohon("Jati", 32.5))   # __eq__ dibuatkan otomatis
print(a < b)                      # order=True: dibandingkan berurutan per kolom

plot_a, plot_b = Plot("A"), Plot("B")
plot_a.pohon.append(a)
print(len(plot_a.pohon), len(plot_b.pohon))
```

```
Pohon(jenis='Jati', diameter_cm=32.5, sehat=True)
True
True
1 0
```

- Tulis kolom dengan **petunjuk tipe** (`jenis: str`). Nilai bawaan dipasang dengan `=` (`sehat: bool = True`).
- `@dataclass` membuatkan `__init__`, `__repr__`, dan `__eq__`. Opsi `order=True` menambah pembanding urutan. Pohon "Jati" lebih kecil daripada "Mahoni" karena kolom pertama (`jenis`) dibandingkan dulu secara abjad.
- Untuk nilai bawaan berupa list, pakai `field(default_factory=list)` agar tiap objek mendapat list **baru**. Hasil akhir `1 0` membuktikan plot A dan B tidak berbagi satu list.

TIPS

Kebiasaan industri adalah memberi nama kelas, fungsi, dan variabel dalam bahasa Inggris. Di buku ini versi Indonesia memakai nama Indonesia supaya mudah dibaca, tetapi di tim sungguhan, ikuti kesepakatan tim Anda.

Proyek Bersambung I1: data lapangan menjadi kelas

Sekarang semua bahan bekerja bersama. Empat `dataclass` menyimpan data (`Pohon`, `Tanaman`, `Ikan`, `Ternak`), induk `Unit` menyimpan yang sama bagi semua unit, dan lima anak (`PlotHutan`, `Kebun`, `Sawah`, `Kolam`, `Kandang`) mengisi `hasil()` dengan caranya sendiri. `Sawah` adalah anak dari `Kebun`: sawah itu kebun yang juga dihitung per hektare.

```
# Bab 16 - Proyek bersambung I1: data lapangan menjadi kelas
from dataclasses import dataclass


@dataclass
class Pohon:
    jenis: str
    diameter_cm: float
    tinggi_m: float
    sehat: bool = True


@dataclass
class Tanaman:
    komoditas: str
    blok: str
    umur_hari: int
    hasil_kg: float


@dataclass
class Ikan:
    jenis: str
    jumlah: int
    bobot_rata_g: float
    ph_air: float

    def biomassa_kg(self):
        return self.jumlah * self.bobot_rata_g / 1000


@dataclass
class Ternak:
    jenis: str
    umur_bulan: int
    bobot_kg: float


class Unit:
    """Induk semua unit lahan atau usaha."""

    jenis_unit = "Unit"
    satuan = ""

    def __init__(self, nama, luas_ha):
        if luas_ha <= 0:
```

```
            raise ValueError("luas_ha harus lebih dari 0")
        self.nama = nama
        self.luas_ha = luas_ha
        self._isi = []

    def tambah(self, barang):
        self._isi.append(barang)

    @property
    def isi(self):
        return tuple(self._isi)

    def hasil(self):
        raise NotImplementedError

    def ringkasan(self):
        return (f"{self.jenis_unit} {self.nama}: {len(self._isi)} catatan, "
                f"{self.hasil():.1f} {self.satuan}")

    def __len__(self):
        return len(self._isi)


class PlotHutan(Unit):
    jenis_unit, satuan = "Plot hutan", "pohon sehat"

    def hasil(self):
        return sum(1 for p in self._isi if p.sehat)


class Kebun(Unit):
    jenis_unit, satuan = "Kebun", "kg"

    def hasil(self):
        return sum(t.hasil_kg for t in self._isi)


class Sawah(Kebun):                     # sawah = kebun yang juga dihitung per hektare
    jenis_unit = "Sawah"

    def produktivitas(self):
        return self.hasil() / self.luas_ha

    def ringkasan(self):
        return super().ringkasan() + f" ({self.produktivitas():.0f} kg/ha)"


class Kolam(Unit):
    jenis_unit, satuan = "Kolam", "kg biomassa"

    def hasil(self):
        return sum(i.biomassa_kg() for i in self._isi)


class Kandang(Unit):
    jenis_unit, satuan = "Kandang", "kg bobot"

    def hasil(self):
        return sum(t.bobot_kg for t in self._isi)
```

```
plot = PlotHutan("A", 2.5)
for p in [Pohon("Jati", 32.5, 18.0), Pohon("Meranti", 45.2, 27.5),
          Pohon("Mahoni", 28.0, 15.0, sehat=False)]:
    plot.tambah(p)

kebun = Kebun("Kopi", 1.2)
for blok, kg in [("A", 245.5), ("B", 310.0)]:
    kebun.tambah(Tanaman("kopi", blok, 400, kg))

sawah = Sawah("Hilir", 0.5)
sawah.tambah(Tanaman("padi", "1", 110, 1250.0))

kolam = Kolam("A", 0.04)
kolam.tambah(Ikan("lele", 800, 540.5, 7.2))

kandang = Kandang("1", 0.1)
for jenis, bulan, kg in [("sapi", 30, 350), ("sapi", 36, 412.5), ("sapi", 33, 389.0)]:
    kandang.tambah(Ternak(jenis, bulan, kg))

for unit in [plot, kebun, sawah, kolam, kandang]:
    print(unit.ringkasan())
```

```
Plot hutan A: 3 catatan, 2.0 pohon sehat
Kebun Kopi: 2 catatan, 555.5 kg
Sawah Hilir: 1 catatan, 1250.0 kg (2500 kg/ha)
Kolam A: 1 catatan, 432.4 kg biomassa
Kandang 1: 3 catatan, 1151.5 kg bobot
```

Telusuri angkanya: plot A punya 3 pohon, 1 tidak sehat, sehingga 2 pohon sehat. Kebun kopi menjumlahkan 245,5 + 310,0 = 555,5 kg. Sawah 1250 kg di lahan 0,5 ha berarti 2500 kg/ha. Kolam 432,4 kg dan kandang 1151,5 kg sama dengan hitungan Seri B. Angka-angkanya fiktif.

Perhatikan bahwa daftar `[plot, kebun, sawah, kolam, kandang]` berisi lima jenis objek, tetapi satu perulangan dan satu perintah `ringkasan()` menangani semuanya. Menambah jenis unit baru (misalnya `Pembenihan`) cukup menulis satu anak baru. Bab-bab berikut memasang proyek ini di environment sendiri, mengujinya, merapikannya, dan menyimpannya di git.

Latihan

1. Tambahkan metode `info_lengkap()` pada kelas `Pohon` di `01_kelas_pohon.py` yang mengembalikan teks seperti `"Jati, 32.5 cm, kategori sedang"`.
2. Buat kelas `Kebun` sendiri (bukan turunan) dengan `nama` dan daftar `panen_kg`, lalu isi `__len__` dan `__str__`. Cetak satu objek dan panggil `len()` padanya.
3. Pada `04_pewarisan.py`, tambahkan kelas `Kolam(Unit)` dengan atribut `ph_air`. Buat satu objek dan periksa `isinstance(kolam, Unit)`.
4. **Tantangan:** tambahkan kelas `Kandang` pada `05_polimorfisme.py` yang mengisi `hasil()` dengan *rata-rata* bobot, bukan total. Tambahkan juga kelas `Pembenihan(Unit)` baru; pastikan perulangan lama tidak perlu diubah.

Ringkasan Bab

- **Class** = cetakan, **objek** = hasil cetakan. `__init__` mengisi data awal; **`self`** adalah objek yang sedang dikerjakan.
- **Inheritance**: `class Anak(Induk)`; `super().__init__(...)` memanggil induk.
- **Polymorphism**: perintah sama, hasil menyesuaikan jenis objek.
- **Encapsulation**: awalan _ sebagai kesepakatan, `property` sebagai penjaga yang memeriksa.
- **Dunder method** (`__str__`, `__repr__`, `__eq__`, `__lt__`, `__len__`) membuat objek akrab dengan Python.
- **`@dataclass`** membuatkan `__init__`, `__repr__`, dan `__eq__` dari daftar kolom.

Selanjutnya, di Bab 17: environment dan dependency, cara memberi tiap proyek "kamar" Python sendiri dan mengatur paket yang dibutuhkannya.

Bab 17 — Environment dan Dependency: Satu Kamar Python untuk Tiap Proyek

Seri I1 — Kode yang Rapi dan Terawat

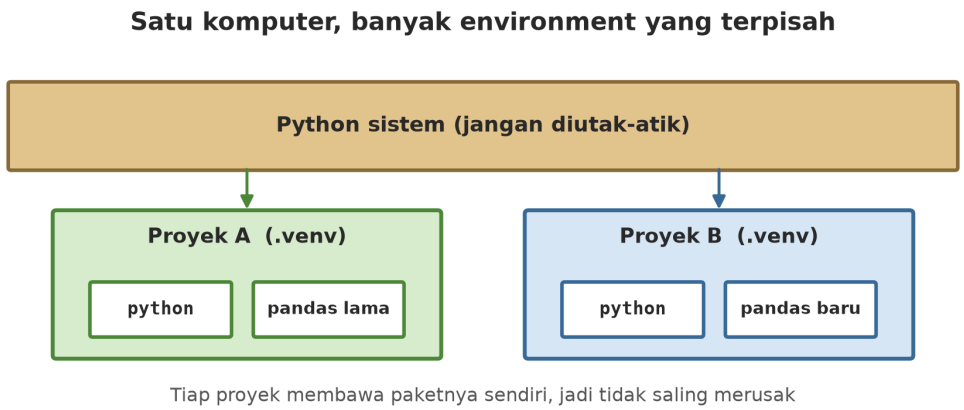
Di bab ini, Anda akan:

- Memahami mengapa tiap proyek perlu **environment** (lingkungan Python) sendiri
- Membuat dan memakai **venv**, memasang paket dengan **pip**, dan menyimpan daftarnya di `requirements.txt`
- Membaca dan menulis `pyproject.toml`, berkas "kartu identitas" proyek
- Mengenal **uv** (pengelola proyek generasi baru), **poetry**, dan **Anaconda/conda**, lengkap dengan catatan lisensi Anaconda
- Memasang paket `inventaris` dari proyek bersambung di environment sendiri

Semua skrip bab ini ada di folder `kode/bab17/`. Di versi web, Anda juga bisa mengunduh semuanya sekaligus (zip).

Di Bab 16, proyek `inventaris` lahir sebagai kumpulan kelas. Mulai Bab 18, ia akan memakai pustaka luar (pytest, ruff). Di sinilah masalah klasik muncul. Proyek A butuh `pandas` versi lama, proyek B butuh yang baru. Bila keduanya dipasang di satu Python yang sama, salah satu rusak. Itu seperti dua petani yang berbagi satu gudang pupuk: begitu yang satu mengganti isinya, yang lain terganggu.

Solusinya **environment** (lingkungan): "kamar" Python terpisah untuk tiap proyek, lengkap dengan paketnya sendiri. **Dependency** (ketergantungan) adalah paket luar yang dibutuhkan proyek Anda.



Gambar 0.1: Satu komputer, banyak environment: tiap proyek membawa paketnya sendiri.

venv: membuat kamar baru

venv adalah modul bawaan Python untuk membuat environment. Perintah di terminal: `python -m venv .venv` (nama `.venv` hanya kebiasaan). Skrip berikut membuatnya lewat kode dan memeriksa isinya.

```
# Bab 17 - Membuat venv: "kamar" Python khusus untuk satu proyek
import os
import shutil
import subprocess
import sys
```

```
import venv
from pathlib import Path

kerja = Path(__file__).parent / "_tmp"
shutil.rmtree(kerja, ignore_errors=True)
kerja.mkdir()
lokasi = kerja / "venv_contoh"

venv.create(lokasi, with_pip=True)          # sama dengan: python -m venv venv_contoh

folder_bin = lokasi / ("Scripts" if os.name == "nt" else "bin")
print("Isi folder venv:", sorted(p.name for p in lokasi.iterdir()))
print("Berkas penting ada:", [
    nama for nama in ("python", "pip", "activate")
    if any(p.stem == nama for p in folder_bin.iterdir())
])

cfg = (lokasi / "pyvenv.cfg").read_text(encoding="utf-8").splitlines()
print([b for b in cfg if b.startswith("include-system-site-packages")])

python_venv = folder_bin / ("python.exe" if os.name == "nt" else "python")
kode = "import sys; print(sys.prefix != sys.base_prefix)"
dalam    =    subprocess.run([python_venv,    "-c",    kode],    capture_output=True,
text=True).stdout.strip()
python_sistem = Path(sys.base_prefix) / ("python.exe" if os.name == "nt" else "bin/
python3")
luar    =    subprocess.run([python_sistem,    "-c",    kode],    capture_output=True,
text=True).stdout.strip()
print("Python di dalam venv terpisah dari sistem?", dalam)
print("Python sistem (induk venv) terpisah?", luar)
```

```
Isi folder venv: ['Include', 'Lib', 'Scripts', 'pyvenv.cfg']
Berkas penting ada: ['python', 'pip', 'activate']
['include-system-site-packages = false']
Python di dalam venv terpisah dari sistem? True
Python sistem (induk venv) terpisah? False
```

- Folder baru berisi `Scripts` (di Linux/macOS bernama `bin`), `Lib` (tempat paket disimpan), `Include`, dan `pyvenv.cfg`. Di `Scripts` ada `python`, `pip`, dan skrip `activate`.
- `include-system-site-packages = false` berarti paket di Python sistem **tidak ikut terlihat**. Kamar ini benar-benar bersih.
- Python di dalam venv melapor `sys.prefix != sys.base_prefix` = `True`: ia tahu dirinya berada di venv. Python sistem melapor `False`.

Untuk memakainya di terminal, **aktifkan** dulu. Aktivasi menaruh folder `Scripts` di depan jalur pencarian perintah, sehingga `python` dan `pip` yang dipanggil adalah milik venv. Menurut dokumentasi resmi Python (dibaca 2026-10-10):

Terminal	Perintah aktivasi
Windows, cmd	`.venv\Scripts\activate.bat`
Windows, PowerShell	`.venv\Scripts\Activate.ps1`
Linux/macOS, bash/zsh	`source .venv/bin/activate`

Untuk keluar, ketik `deactivate`. Aktivasi sifatnya **opsional**: Anda juga bisa memanggil langsung `.venv\Scripts\python.exe`, itulah yang dilakukan skrip-skrip di bab ini.

⚠ AWAS

Jangan **memindah atau menyalin** folder venv ke tempat lain, dan jangan menyimpannya di git. Dokumentasi Python menjelaskan bahwa skrip di dalamnya memuat jalur absolut, jadi venv yang dipindah bisa rusak. Cara yang benar: buang, lalu **buat ulang** dari daftar paket (`requirements.txt` atau `pyproject.toml`). Python 3.13 ke atas bahkan otomatis menaruh `.gitignore` di dalam venv.

pip dan requirements.txt

pip adalah pemasang paket bawaan Python; paketnya diunduh dari **PyPI** (*Python Package Index*, gudang paket resmi). Skrip berikut membuktikan isolasi venv: paket `requests` dipasang di satu venv, lalu daftarnya dibekukan (`pip freeze`) ke **`requirements.txt`** dan dipakai membangun venv kedua.

```
# Bab 17 - pip di dalam venv: pasang, bekukan, pasang ulang dari requirements.txt
import os
import shutil
import subprocess
import venv
from pathlib import Path

kerja = Path(__file__).parent / "_tmp"
shutil.rmtree(kerja, ignore_errors=True)
kerja.mkdir()


def python_dari(lokasi):
    return lokasi / ("Scripts/python.exe" if os.name == "nt" else "bin/python")


def jalan(lokasi, *argumen):
        return subprocess.run([python_dari(lokasi), *argumen], capture_output=True,
text=True)


venv_a = kerja / "venv_a"
venv.create(venv_a, with_pip=True)

hasil = jalan(venv_a, "-c", "import requests")
print("Sebelum dipasang:", hasil.stderr.strip().splitlines()[-1])

jalan(venv_a, "-m", "pip", "install", "--quiet", "--disable-pip-version-check",
"requests")
hasil = jalan(venv_a, "-c", "import requests; print('berhasil diimpor')")
print("Sesudah dipasang:", hasil.stdout.strip())

beku = jalan(venv_a, "-m", "pip", "freeze").stdout
(kerja / "requirements.txt").write_text(beku, encoding="utf-8")
print("--- requirements.txt ---")
print(beku.strip())

venv_b = kerja / "venv_b"
venv.create(venv_b, with_pip=True)
jalan(venv_b, "-m", "pip", "install", "--quiet", "--disable-pip-version-check",
      "-r", str(kerja / "requirements.txt"))
beku_b = jalan(venv_b, "-m", "pip", "freeze").stdout
print("venv_b identik dengan venv_a?", beku_b == beku)
```

```
Sebelum dipasang: ModuleNotFoundError: No module named 'requests'
Sesudah dipasang: berhasil diimpor
--- requirements.txt ---
certifi==2026.7.22
charset-normalizer==3.5.2
idna==3.20
requests==2.34.2
urllib3==2.8.0
venv_b identik dengan venv_a? True
```

Kode ini **dijalankan sungguhan satu kali** oleh penulis, lalu keluarannya disimpan, karena butuh internet dan memuat nomor versi yang terus berubah. Jadi nomor versi di atas adalah keadaan pada saat buku ini ditulis; di komputer Anda akan berbeda.

Telusuri hasilnya:

- **Sebelum dipasang**, `import requests` gagal dengan `ModuleNotFoundError`. Kamar baru memang kosong.
- **Sesudah** `pip install requests`, impor berhasil.
- `pip freeze` mencatat **lima** paket, bukan satu: `requests` membawa empat dependensi lain (`certifi`, `charset-normalizer`, `idna`, `urllib3`). Masing-masing dikunci pada nomor versi persis (`==`).
- Venv kedua yang dibangun dari `requirements.txt` identik dengan yang pertama (`True`). Itulah gunanya berkas ini: rekan kerja atau server dapat membuat kamar yang sama.

> TIPS
>
> Pasang paket dengan `python -m pip install ...`, bukan hanya `pip install ...`. Bentuk pertama memastikan pip yang dipakai adalah milik Python yang sedang Anda panggil. Itu menghindari kesalahan "terpasang di Python yang salah".

pyproject.toml: kartu identitas proyek

`requirements.txt` hanya daftar paket. Proyek modern menyimpan **semuanya** dalam satu berkas, **`pyproject.toml`**: nama, versi, Python yang dibutuhkan, dependensi, dan pengaturan alat (pytest, ruff). Formatnya **TOML**: bagian ditulis `[nama]`, isinya `kunci = nilai`. Ini berkas milik paket `inventaris`:

```toml
[build-system]
requires = ["setuptools>=61"]
build-backend = "setuptools.build_meta"

[project]
name = "inventaris"
version = "0.1.0"
requires-python = ">=3.12"
dependencies = []

[project.optional-dependencies]
dev = ["pytest", "ruff"]

[tool.ruff]
line-length = 88
```

- `[build-system]` memberi tahu alat pemasang cara "membangun" paket (di sini memakai setuptools).
- `[project]` memuat identitas proyek. `dependencies` adalah paket wajib (kosong: `inventaris` hanya memakai pustaka bawaan). `optional-dependencies` memuat kelompok tambahan, di sini `dev` untuk alat pengembang.
- `[tool.xxx]` adalah ruang pengaturan untuk alat, seperti `[tool.ruff]` dan `[tool.pytest.ini_options]` (Bab 18).

Python 3.11 ke atas punya modul bawaan **`tomllib`** untuk membacanya:

```python
# Bab 17 - Membaca pyproject.toml dengan tomllib (pustaka bawaan Python 3.11+)
import tomllib
from pathlib import Path

berkas = Path(__file__).parent / "proyek" / "pyproject.toml"
with open(berkas, "rb") as f:             # tomllib mewajibkan mode biner
    data = tomllib.load(f)

proyek = data["project"]
print("Nama proyek      :", proyek["name"])
print("Versi            :", proyek["version"])
print("Butuh Python     :", proyek["requires-python"])
print("Dependensi       :", proyek["dependencies"])
print("Dependensi dev   :", proyek["optional-dependencies"]["dev"])
print("Alat build       :", data["build-system"]["build-backend"])
print("Bagian [tool.*]  :", sorted(data["tool"]))
print("Panjang baris ruff:", data["tool"]["ruff"]["line-length"])
```

```
Nama proyek      : inventaris
Versi            : 0.1.0
Butuh Python     : >=3.12
Dependensi       : []
Dependensi dev   : ['pytest', 'ruff']
Alat build       : setuptools.build_meta
Bagian [tool.*]  : ['ruff', 'setuptools']
Panjang baris ruff: 88
```

Ingat: TOML dibaca sebagai dict biasa. `data["project"]["name"]` adalah dict di dalam dict (Bab 9).

uv: pengelola paket generasi baru

uv adalah pengelola paket dan proyek Python buatan Astral (pembuat ruff yang kita pakai di Bab 19). Ia melakukan pekerjaan `venv` + `pip` + kunci versi dalam satu perintah, dan sangat cepat. Berdasarkan dokumentasi resmi uv (dibaca 2026-10-10) dan percobaan di komputer penulis:

Perintah	Fungsi
`uv init`	membuat proyek baru (`pyproject.toml`, `.python-version`, `main.py`)
`uv add paket`	menambah dependensi ke `pyproject.toml`, membuat `.venv` dan `uv.lock`
`uv run perintah`	menjalankan perintah di environment proyek, memastikan environment mutakhir
`uv lock` / `uv sync`	memperbarui berkas kunci / menyelaraskan environment dengan kunci
`uv tree`	menampilkan pohon dependensi
`uv venv`, `uv pip ...`	pengganti `python -m venv` dan `pip` yang kompatibel

```python
# Bab 17 - uv: proyek baru dari nol (init, add, run, tree)
import shutil
import subprocess
from pathlib import Path

kerja = Path(__file__).parent / "_tmp" / "uv_demo"
shutil.rmtree(kerja, ignore_errors=True)
kerja.mkdir(parents=True)
```

```
def uv(*argumen):
   hasil = subprocess.run(["uv", *argumen], cwd=kerja, capture_output=True, text=True)
    return hasil.stdout.strip()


uv("init", "--name", "inventaris-demo", "--no-readme", "--vcs", "none", "--python",
"3.12")
print("Berkas setelah uv init:", sorted(p.name for p in kerja.iterdir()))

uv("add", "requests")
print("Berkas setelah uv add :", sorted(p.name for p in kerja.iterdir()))

baris = (kerja / "pyproject.toml").read_text(encoding="utf-8").splitlines()
print("--- pyproject.toml (dependencies) ---")
print("\n".join(baris[baris.index("dependencies = [") : baris.index("]")+1]))

print("--- uv run ---")
print(uv("run", "python", "-c", "import requests; print('requests siap dipakai')"))

print("--- uv tree ---")
print(uv("tree"))
```

```
Berkas setelah uv init: ['.python-version', 'main.py', 'pyproject.toml']
Berkas setelah uv add : ['.python-version', '.venv', 'main.py', 'pyproject.toml',
'uv.lock']
--- pyproject.toml (dependencies) ---
dependencies = [
    "requests>=2.34.2",
]
--- uv run ---
requests siap dipakai
--- uv tree ---
inventaris-demo v0.1.0
└── requests v2.34.2
    ├── certifi v2026.7.22
    ├── charset-normalizer v3.5.2
    ├── idna v3.20
    └── urllib3 v2.8.0
```

Ini juga dijalankan sungguhan dan keluarannya disimpan (butuh internet dan memuat nomor versi). Perhatikan:

- `uv init` hanya membuat tiga berkas. **`.venv`** dan **`uv.lock`** baru lahir ketika `uv add` dijalankan. `uv.lock` adalah **berkas kunci** berisi versi persis semua paket, dikelola uv dan tidak boleh disunting tangan (dokumentasi uv).
- `uv add requests` menulis `"requests>=2.34.2"` ke `pyproject.toml`: batas bawah, bukan harga mati. Versi persis ada di `uv.lock`.
- `uv tree` memperlihatkan pohonnya: `requests` di atas, empat dependensi di bawahnya, sama dengan hasil `pip freeze` tadi.

> **i** INGAT
>
> Dua jenis daftar punya tugas berbeda. **`pyproject.toml`** menyatakan "paket apa yang saya butuhkan" (longgar: `>=2.34.2`). **Berkas kunci** (`uv.lock`, `requirements.txt` hasil freeze) menyatakan "versi persis yang terpasang" supaya semua orang mendapat hasil yang sama.

Proyek Bersambung I1: memasang inventaris

Sekarang paket `inventaris` dari Bab 16 dipasang di environment sendiri memakai uv. Opsi `-e` (*editable*) membuat Python membaca kode langsung dari folder proyek, jadi perubahan kode langsung terlihat tanpa memasang ulang.

```
# Bab 17 - Proyek bersambung I1: memasang paket inventaris di environment sendiri
import os
import shutil
import subprocess
from pathlib import Path

induk = Path(__file__).parent
kerja = induk / "_tmp" / "pasang"
shutil.rmtree(kerja, ignore_errors=True)
kerja.mkdir(parents=True)
salinan = kerja / "proyek"
shutil.copytree(induk / "proyek", salinan)

lingkungan = kerja / "venv_inventaris"
subprocess.run(["uv", "venv", str(lingkungan), "--quiet"], check=True)
python = lingkungan / ("Scripts/python.exe" if os.name == "nt" else "bin/python")

# -e = "editable": kode di folder proyek langsung terbaca tanpa memasang ulang
subprocess.run(["uv", "pip", "install", "--quiet", "--python", str(python), "-e",
str(salinan)],
               check=True)

kode = """
from importlib.metadata import version
import inventaris
from inventaris.contoh import buat_contoh

print("Terpasang: inventaris", version("inventaris"))
from pathlib import Path
print("Kode dibaca dari folder proyek?", Path(inventaris.__file__).parent.parent.name
== "proyek")
for unit in buat_contoh():
    print(unit.ringkasan())
"""
hasil = subprocess.run([python, "-c", kode], capture_output=True, text=True)
print(hasil.stdout.strip())
```

```
Terpasang: inventaris 0.1.0
Kode dibaca dari folder proyek? True
Plot hutan A: 3 catatan, 2.0 pohon sehat
Kebun Kopi: 2 catatan, 555.5 kg
Sawah Hilir: 1 catatan, 1250.0 kg (2500 kg/ha)
Kolam A: 1 catatan, 432.4 kg biomassa
Kandang 1: 3 catatan, 1151.5 kg bobot
```

Jalan ceritanya: salinan proyek ditaruh di `_tmp/`, `uv venv` membuat kamar kosong, `uv pip install -e` memasang paket (uv mengunduh alat build setuptools, itulah sebabnya butuh internet, dan keluarannya disimpan dari satu kali jalan). Lalu **di dalam kamar itu** `inventaris` bisa diimpor dari mana saja dan hasilnya sama persis dengan Bab 16: plot hutan A, kebun kopi, sawah hilir, kolam, dan kandang.

Poetry: pesaing uv

poetry adalah pengelola proyek Python yang lebih dulu populer; ia juga memakai `pyproject.toml` dan membuat berkas kunci `poetry.lock`. Dari dokumentasi resmi poetry (dibaca 2026-10-10): `poetry new` membuat folder proyek, `poetry add` menambah dependensi dan memasangnya, `poetry`

`install` memasang semua dependensi, dan `poetry run` menjalankan perintah di environment proyek. Dependensi ditulis di `project.dependencies` (standar PEP 621); poetry versi lama memakai `tool.poetry.dependencies`.

```
# Bab 17 - Alur dasar poetry (hanya contoh; tidak dijalankan saat buku dibuat)
import subprocess

perintah = [
    ["poetry", "new", "inventaris-poetry"],   # membuat folder proyek baru
     ["poetry", "add", "requests"],               # menambah dependensi + mengunci
versinya (poetry.lock)
    ["poetry", "install"],                    # memasang semua dependensi proyek
   ["poetry", "run", "pytest"],           # menjalankan perintah di environment proyek
]
for p in perintah:
    print("$", " ".join(p))
    subprocess.run(p, check=True)
```

Kode ini **tidak dijalankan** saat buku dibuat karena poetry belum terpasang di komputer penulis (lihat hasil `06_cek_alat.py` di bawah), jadi keluarannya tidak ditampilkan. Perintahnya dicocokkan dengan dokumentasi resmi, tetapi belum diuji langsung.

Anaconda dan conda

Anaconda adalah cara lain memasang dan mengelola Python, populer di dunia data science. Ada tiga istilah yang sering tertukar:

- **conda**: alat baris perintah untuk membuat environment dan memasang paket. Dokumentasinya menyebut conda mengelola "paket, dependensi, dan environment untuk bahasa apa pun", jadi bukan hanya paket Python.
- **Anaconda Distribution**: paket pemasang yang membawa conda, Python, **Anaconda Navigator** (aplikasi berjendela untuk membuat environment dan membuka alat tanpa mengetik perintah), dan 600+ paket sains data siap pakai, sekitar 9,7 GB.
- **Miniconda**: versi ringkas: conda, Python, 130+ paket, tanpa Navigator, sekitar 900 MB.

(Angka ukuran dari halaman resmi Anaconda, dibaca 2026-10-10.)

Bedanya dengan python.org: pemasang python.org hanya memberi Python; selebihnya Anda urus dengan venv dan pip. Anaconda memberi Python plus perangkat pengelola environment dan banyak paket sekaligus. Paketnya diambil dari **channel** (saluran) conda, bukan dari PyPI.

Alurnya di terminal (dari dokumentasi resmi conda, dibaca 2026-10-10):

```
conda create --name inventaris python numpy   # buat environment berisi Python dan numpy
conda activate inventaris                       # masuk ke environment
conda install --name inventaris matplotlib      # pasang paket lain
conda info --envs                               # daftar environment
conda deactivate                                # keluar
```

```
# Catatan: "conda activate inventaris" diketik langsung di terminal, bukan lewat skrip.
# Bab 17 - Alur dasar conda (hanya contoh; tidak dijalankan saat buku dibuat)
import subprocess

perintah = [
      ["conda", "create", "--name", "inventaris", "python", "numpy"],    # membuat
environment + isinya
    ["conda", "install", "--name", "inventaris", "numpy"],        # memasang paket
    ["conda", "info", "--envs"],                                 # daftar environment
]
for p in perintah:
    print("$", " ".join(p))
    subprocess.run(p, check=True)
```

Kode ini **tidak dijalankan** saat buku dibuat karena conda belum terpasang di komputer penulis, jadi keluarannya tidak ditampilkan dan **seluruh bagian conda di bab ini belum diuji langsung**. (`conda activate` sengaja tidak ada dalam skrip: ia diketik di terminal.)

```
# Bab 17 - Memeriksa alat apa saja yang ada di komputer
import shutil

for alat in ["python", "pip", "uv", "git", "poetry", "conda"]:
    jalur = shutil.which(alat)
    print(f"{alat:7}", "ada" if jalur else "TIDAK ada")
```

```
python  ada
pip     ada
uv      ada
git     ada
poetry  TIDAK ada
conda   TIDAK ada
```

Keluaran ini diambil sekali di komputer penulis dan berbeda bila dijalankan di komputer Anda. Di sini `uv` dan `git` ada, `poetry` dan `conda` tidak.

conda vs pip: pip memasang dari PyPI, conda dari channel conda. Dokumentasi conda menyarankan: bila perlu keduanya di satu environment, **pasang sebanyak mungkin dengan conda dulu, baru pip** ("Use pip only after conda"). **conda-forge** adalah channel yang dijalankan komunitas, terpisah dari channel `defaults` milik Anaconda, Inc. Dokumentasi conda-forge menyarankan Miniforge sebagai cara termudah memulainya.

Jupyter: halaman resmi Anaconda menyebut Jupyter Notebook dapat dibuka dari Navigator. (Halaman itu tidak merinci apakah Jupyter terpasang bawaan; periksa setelah memasang.)

> ⚠ AWAS
>
> **Lisensi Anaconda.** Anaconda tidak gratis untuk semua orang. Menurut Ketentuan Layanan Anaconda (`https://www.anaconda.com/legal/terms/terms-of-service`, tanggal berlaku tertulis 15 Juli 2025, dibaca 2026-10-10), penggunaan platform **gratis** bila Anda: (1) individu untuk keperluan pribadi nonkomersial, (2) terkait lembaga akademik yang memenuhi syarat, (3) terkait organisasi nirlaba dan riset yang memenuhi syarat, atau (4) untuk organisasi **berorientasi laba dengan 200 karyawan/kontraktor atau kurang** (termasuk afiliasi). Organisasi berorientasi laba dengan **lebih dari 200** karyawan/kontraktor yang tidak termasuk kategori gratis harus berlangganan paket Business. Halaman harga (`https://www.anaconda.com/pricing`, dibaca 2026-10-10) menulis "200+" dan menyebut paket Free untuk yang belajar Python. Anaconda berhak meminta bukti kelayakan. Syarat ini bisa berubah, jadi **baca halaman resminya sendiri sebelum memakai Anaconda di tempat kerja**. Penulis tidak memeriksa apakah syarat itu berlaku bagi channel conda-forge; dokumen ketentuan tidak menyebutnya.

Kapan memilih yang mana? Rekomendasi penulis untuk pembaca buku ini: untuk proyek pengembangan umum (aplikasi, otomasi), **venv + pip atau uv** sudah cukup, ringan, dan tanpa urusan lisensi. Pertimbangkan Anaconda/conda bila Anda bekerja dengan pustaka sains yang sulit dipasang dengan pip, atau bila tim Anda sudah memakainya. Di tempat kerja, periksa dulu ketentuan lisensi di atas.

> 🔥 BAGIAN TEKNIS (boleh dilewati)
>
> Pada Windows, PowerShell bisa menolak `Activate.ps1` karena kebijakan eksekusi skrip. Dokumentasi Python menyarankan `Set-ExecutionPolicy -ExecutionPolicy RemoteSigned -Scope CurrentUser`. Ini mengubah pengaturan keamanan akun Anda sendiri, jadi lakukan hanya bila paham akibatnya; alternatifnya, panggil `.venv\Scripts\python.exe` langsung tanpa aktivasi.

Latihan

1. Di terminal, buat venv `latihan_venv` dengan `python -m venv latihan_venv`, aktifkan, lalu ketik `python -c "import sys; print(sys.prefix)"`. Bandingkan dengan hasil tanpa aktivasi.

2. Pasang `six` (paket kecil) di venv itu dengan `python -m pip install six`, lalu simpan `python -m pip freeze > requirements.txt` dan lihat isinya.
3. Ubah `03_baca_pyproject.py` agar mencetak juga daftar `dev`, satu paket per baris.
4. **Tantangan:** dengan `uv`, buat proyek baru (`uv init`), tambahkan `pytest` sebagai dependensi pengembangan (`uv add --dev pytest`; periksa `uv add --help`), lalu jalankan `uv run pytest --version`.

Ringkasan Bab

- **Environment** = kamar Python terpisah per proyek; **dependency** = paket luar yang dibutuhkan.
- **venv** membuat environment; aktivasi opsional; jangan dipindah atau dimasukkan ke git, cukup dibuat ulang.
- **pip** memasang dari PyPI; **`pip freeze`** menghasilkan `requirements.txt` berisi versi persis.
- **`pyproject.toml`** menyimpan identitas proyek, dependensi, dan pengaturan alat; dibaca dengan `tomllib`.
- **uv** menggabungkan venv, pip, dan berkas kunci (`uv.lock`) dengan cepat. **poetry** adalah pesaingnya (tidak diuji di sini).
- **Anaconda/conda** punya kegunaan dan lisensi sendiri; baca ketentuan resminya sebelum dipakai di tempat kerja. Bagian conda di buku ini belum diuji langsung.

Selanjutnya, di Bab 18: testing dasar dengan pytest, cara membuat kode yang memeriksa kode Anda sendiri.

Bab 18 — Testing Dasar: Memeriksa Kode Otomatis dengan pytest

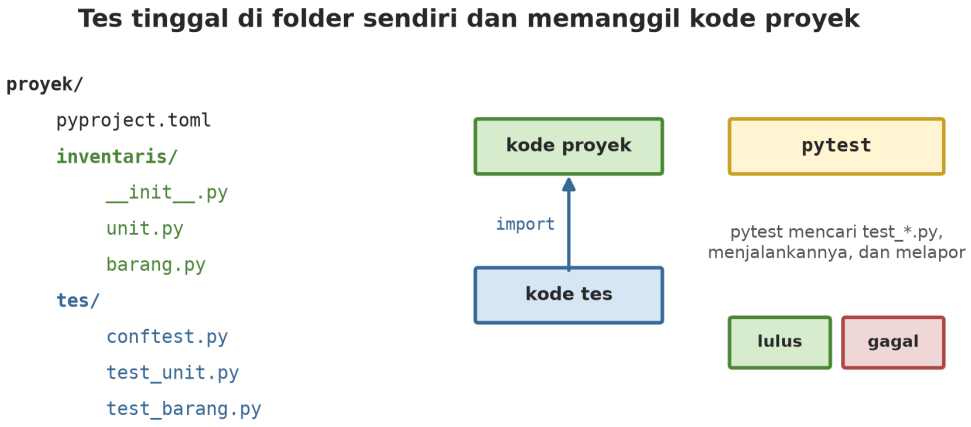
Seri I1 — Kode yang Rapi dan Terawat

Di bab ini, Anda akan:

- Memahami mengapa tes otomatis menyelamatkan Anda saat kode diubah
- Menulis pernyataan `assert` dan fungsi tes pertama
- Menjalankan **pytest**, membaca laporan lulus dan gagal
- Memakai **fixture** (bahan siap pakai), `parametrize`, dan `pytest.raises`
- Menata folder tes dan menguji paket `inventaris` dari proyek bersambung

Semua skrip bab ini ada di folder `kode/bab18/`. Di versi web, Anda juga bisa mengunduh semuanya sekaligus (zip).

Bayangkan Anda mengganti rumus biomassa di kelas `Kolam`. Apakah kandang, sawah, dan plot hutan masih menghasilkan angka yang benar? Memeriksa semuanya dengan mata tiap kali ada perubahan itu melelahkan dan pasti ada yang terlewat. Solusinya: tulis **tes otomatis**, yaitu kode kecil yang memanggil kode Anda lalu membandingkan hasilnya dengan jawaban yang diharapkan. Seperti timbangan ikan yang dikalibrasi tiap pagi: sekali disiapkan, ia memeriksa dalam sekejap.



Gambar 0.1: Tes tinggal di folder sendiri, memanggil kode proyek, dan pytest melaporkan lulus atau gagal.

assert: “pastikan ini benar”

Dasar semua tes di Python adalah **`assert`**. Bila syaratnya benar, program lanjut diam-diam. Bila salah, muncul `AssertionError` (error “pernyataan tidak terpenuhi”).

```
# Bab 18 - assert: "pastikan ini benar, kalau tidak, berhenti"
biomassa_kg = 800 * 540.5 / 1000

assert biomassa_kg > 0
print("assert pertama lolos: biomassa", biomassa_kg, "kg")

try:
    assert biomassa_kg == 400, "biomassa seharusnya 400 kg"
```

```
except AssertionError as e:
    print("assert kedua gagal:", e)

jumlah = 0
try:
    assert jumlah > 0
except AssertionError:
    print("assert tanpa pesan juga bisa gagal")
```

```
assert pertama lolos: biomassa 432.4 kg
assert kedua gagal: biomassa seharusnya 400 kg
assert tanpa pesan juga bisa gagal
```

Pernyataan pertama lolos tanpa suara selain `print` sesudahnya. Pada yang kedua, tanda koma menambahkan pesan yang muncul saat gagal. Skrip menangkapnya dengan `try/except` (Bab 12) supaya program tidak berhenti.

pytest dan tes pertama

pytest adalah pustaka tes yang paling banyak dipakai pemrogram Python. Ia sudah terpasang di environment buku ini; di komputer Anda, pasang dengan `pip install pytest` di dalam venv (Bab 17). Aturannya sederhana:

- Berkas tes bernama **`test_*.py`** (atau `*_test.py`).
- Fungsi tes bernama **`test_...`**.
- Di dalamnya, tulis `assert`.

(Sumber: dokumentasi resmi pytest, halaman “Good Integration Practices”, dibaca 2026-10-10.)

```
# Bab 18 - Tes pytest pertama: berkas test_*.py berisi fungsi test_*
from pathlib import Path

from bantu import FOLDER, jalankan_pytest

print(Path(FOLDER                                                              /
"contoh_sederhana" / "test_hitung.py").read_text(encoding="utf-8"))
print("--- hasil pytest ---")
teks, kode = jalankan_pytest(["-v"], FOLDER / "contoh_sederhana")
print(teks)
print("kode keluar:", kode)
```

```
def biomassa_kg(jumlah, bobot_g):
    return jumlah * bobot_g / 1000


def test_biomassa_sederhana():
    assert biomassa_kg(800, 500) == 400


def test_biomassa_nol():
    assert biomassa_kg(0, 540.5) == 0

--- hasil pytest ---
============================= test session starts ==============================
collecting ... collected 2 items

test_hitung.py::test_biomassa_sederhana PASSED                           [ 50%]
test_hitung.py::test_biomassa_nol PASSED                                 [100%]
```

```
============================== 2 passed ==============================
kode keluar: 0
```

Skrip ini menjalankan pytest lewat `subprocess` pada folder `contoh_sederhana/`. Pada keluaran, `test_hitung.py` memuat dua fungsi tes, keduanya `PASSED` (lulus), dan ringkasannya "2 passed". Kode keluar `0` artinya semuanya beres. pytest memberi kode keluar `1` bila ada tes gagal, sehingga bisa dipakai oleh skrip otomatis dan CI (Bab 20). Pada keluaran buku ini, waktu proses di ringkasan sengaja dibuang supaya hasilnya tetap sama setiap kali dibuat.

Saat tes gagal

Tes yang selalu lulus belum membuktikan apa-apa; Anda perlu tahu seperti apa laporannya bila gagal. Berkas `contoh_gagal/test_gagal.py` sengaja berisi harapan yang salah (400 kg, padahal hitungannya 432,4 kg).

```
# Bab 18 - Membaca laporan tes yang gagal
from bantu import FOLDER, jalankan_pytest

teks, kode = jalankan_pytest(["-q"], FOLDER / "contoh_gagal")
print(teks)
print("kode keluar:", kode)
```

```
F                                                                    [100%]
================================= FAILURES =================================
____________________________ test_biomassa_salah_harapan ____________________________

    def test_biomassa_salah_harapan():
        ikan = Ikan("lele", 800, 540.5, 7.2)
>       assert ikan.biomassa_kg() == 400.0
E       AssertionError: assert 432.4 == 400.0
E        +  where 432.4 = biomassa_kg()
E        +    where biomassa_kg = Ikan(jenis='lele', jumlah=800, bobot_rata_g=540.5,
ph_air=7.2).biomassa_kg

test_gagal.py:6: AssertionError
========================== short test summary info ==========================
FAILED test_gagal.py::test_biomassa_salah_harapan - AssertionError: assert 43...
1 failed
kode keluar: 1
```

Cara membacanya dari atas:

- Huruf **`F`** menandai satu tes gagal (tes lulus berupa titik).
- Tanda panah **`>`** menunjuk baris `assert` yang gagal.
- Baris **`E`** (*error*) menjelaskan: nilai sebenarnya 432,4, harapannya 400,0. pytest bahkan menguraikan dari mana 432,4 berasal (`biomassa_kg()` pada objek `Ikan` itu).
- Terakhir ada ringkasan `FAILED ...` dan kode keluar `1`.

> TIPS
>
> Saat tes gagal, jangan langsung mengubah tesnya agar lulus. Tanyakan dulu: *yang salah kodenya atau harapannya?* Di contoh ini yang salah adalah harapannya. Kadang yang salah justru kodenya, dan itulah gunanya tes.

Menata folder tes

Paket `inventaris` dari Bab 16 sekarang jadi folder `proyek/`. Tes ditaruh di folder `tes/` **di samping** paketnya, tidak dicampur. pytest menyarankan pemisahan seperti ini (dokumentasi resmi, 2026-10-10).

```
proyek/
  pyproject.toml
  inventaris/
    __init__.py   unit.py   barang.py   contoh.py   laporan.py
  tes/
    conftest.py
    test_barang.py   test_unit.py   test_laporan.py
```

Aturan pytest ditulis di `pyproject.toml` (Bab 17):

```
[tool.pytest.ini_options]
testpaths = ["tes"]
pythonpath = ["."]
```

`testpaths` memberi tahu pytest untuk mencari tes hanya di `tes/`. `pythonpath = ["."]` menambahkan folder proyek ke jalur impor, sehingga `import inventaris` berhasil tanpa memasangnya dulu.

```
# Bab 18 - Daftar tes yang ditemukan pytest di proyek (tanpa menjalankannya)
from bantu import FOLDER, jalankan_pytest

teks, kode = jalankan_pytest(["--collect-only", "-q"], FOLDER / "proyek")
print(teks)
```

```
tes/test_barang.py::test_biomassa_ikan
tes/test_barang.py::test_kategori_pohon[28.0-kecil]
tes/test_barang.py::test_kategori_pohon[32.5-sedang]
tes/test_barang.py::test_kategori_pohon[45.2-besar]
tes/test_barang.py::test_kategori_pohon[40.0-besar]
tes/test_barang.py::test_jumlah_ikan_negatif_ditolak
tes/test_laporan.py::test_simpan_ringkasan
tes/test_unit.py::test_luas_nol_ditolak
tes/test_unit.py::test_induk_tidak_bisa_menghitung
tes/test_unit.py::test_hasil_kolam
tes/test_unit.py::test_ringkasan_kolam
tes/test_unit.py::test_ringkasan_lima_unit
tes/test_unit.py::test_isi_tidak_bisa_diubah_dari_luar

13 tests collected
```

Ini mode `--collect-only`: pytest hanya **mengumpulkan** tes tanpa menjalankannya. Dari tiga berkas, ditemukan **13 tes**. Tes `test_kategori_pohon` muncul empat kali, dengan tanda `[28.0-kecil]` dan seterusnya. Sebabnya ada di bagian berikut.

Fixture, parametrize, dan raises

Tiga alat pytest yang paling sering dipakai, semuanya ada di proyek kita.

Fixture adalah "bahan siap pakai": fungsi bertanda `@pytest.fixture` yang dibuat sekali dan dipinjamkan ke tes lewat nama. Letakkan di `conftest.py` agar semua berkas tes bisa memakainya.

```
"""Fixture bersama: bahan siap pakai untuk semua berkas tes."""

import pytest

from inventaris import Ikan, Kolam
from inventaris.contoh import buat_contoh


@pytest.fixture
def unit_contoh():
    """Lima unit contoh, dibuat baru untuk setiap tes."""
    return buat_contoh()
```

```python
@pytest.fixture
def kolam_a():
    kolam = Kolam("A", 0.04)
    kolam.tambah(Ikan("lele", 800, 540.5, 7.2))
    return kolam
```

Tes cukup menulis nama fixture sebagai parameter, dan pytest mengisinya. Setiap tes mendapat objek **baru**, sehingga tes tidak saling mengganggu.

```python
import pytest

from inventaris import Ikan, Pohon


def test_biomassa_ikan():
    ikan = Ikan("lele", 800, 540.5, 7.2)
    assert ikan.biomassa_kg() == pytest.approx(432.4)


@pytest.mark.parametrize(
    ("diameter", "harapan"),
    [(28.0, "kecil"), (32.5, "sedang"), (45.2, "besar"), (40.0, "besar")],
)
def test_kategori_pohon(diameter, harapan):
    assert Pohon("Jati", diameter, 15.0).kategori() == harapan


def test_jumlah_ikan_negatif_ditolak():
    with pytest.raises(ValueError, match="negatif"):
        Ikan("lele", -1, 500.0, 7.0)
```

- **`@pytest.mark.parametrize`** menjalankan satu fungsi tes untuk beberapa pasang data. Empat pasangan diameter menjadi empat tes (itulah `[28.0-kecil]` tadi). Diameter 40,0 sengaja dimasukkan untuk menguji **batas**: kategori besar dimulai tepat di 40.
- **`pytest.approx(432.4)`** membandingkan bilangan desimal dengan toleransi kecil, supaya kesalahan pembulatan tidak menggagalkan tes (ingat Bab 6: bilangan desimal bisa sedikit meleset).
- **`pytest.raises(ValueError, match="negatif")`** menyatakan: "kode ini **harus** melempar error". Bila tidak melempar, tes gagal. Inilah cara menguji bahwa penjaga data bekerja.

```python
import pytest

from inventaris import Kolam, Unit


def test_luas_nol_ditolak():
    with pytest.raises(ValueError):
        Kolam("X", 0)


def test_induk_tidak_bisa_menghitung():
    with pytest.raises(NotImplementedError):
        Unit("Polos", 1.0).hasil()


def test_hasil_kolam(kolam_a):
    assert kolam_a.hasil() == pytest.approx(432.4)
```

```
def test_ringkasan_kolam(kolam_a):
    assert kolam_a.ringkasan() == "Kolam A: 1 catatan, 432.4 kg biomassa"


def test_ringkasan_lima_unit(unit_contoh):
    teks = [unit.ringkasan() for unit in unit_contoh]
    assert teks[0] == "Plot hutan A: 3 catatan, 2.0 pohon sehat"
    assert teks[2] == "Sawah Hilir: 1 catatan, 1250.0 kg (2500 kg/ha)"
    assert len(teks) == 5


def test_isi_tidak_bisa_diubah_dari_luar(kolam_a):
    assert isinstance(kolam_a.isi, tuple)
    with pytest.raises(AttributeError):
        kolam_a.isi.append("x")
```

Tes `test_ringkasan_lima_unit` memakai fixture `unit_contoh` dan membandingkan teks ringkasan persis. Ada pula tes `test_induk_tidak_bisa_menghitung` untuk perilaku polimorfisme Bab 16.

```
from inventaris.laporan import simpan_ringkasan


def test_simpan_ringkasan(tmp_path, unit_contoh):
    jalur = tmp_path / "ringkasan.txt"
    jumlah = simpan_ringkasan(unit_contoh, jalur)
    isi = jalur.read_text(encoding="utf-8").splitlines()
    assert jumlah == 5
    assert isi[3] == "Kolam A: 1 catatan, 432.4 kg biomassa"
```

Fixture bawaan **`tmp_path`** memberi folder sementara yang dibuang sesudah tes, sehingga tes yang menulis berkas tidak mengotori komputer Anda.

> **i** INGAT
>
> Tes yang baik: **kecil** (satu hal per tes), **mandiri** (tidak bergantung pada tes lain), dan **namanya menjelaskan harapan**. `test_luas_nol_ditolak` langsung memberi tahu apa yang rusak bila ia gagal.

Menjalankan sebagian tes

Pada proyek besar, Anda sering hanya ingin menjalankan tes yang berkaitan dengan bagian yang sedang diubah. Opsi **`-k`** memilih tes berdasarkan potongan nama, dan **`-v`** (*verbose*) menampilkan tiap tes.

```
# Bab 18 - Menjalankan sebagian tes: -k memilih berdasarkan nama
from bantu import FOLDER, jalankan_pytest

teks, kode = jalankan_pytest(["-v", "-k", "kolam"], FOLDER / "proyek")
print(teks)
```

```
============================= test session starts ==============================
testpaths: tes
collecting ... collected 13 items / 11 deselected / 2 selected

tes/test_unit.py::test_hasil_kolam PASSED                                [ 50%]
tes/test_unit.py::test_ringkasan_kolam PASSED                            [100%]

====================== 2 passed, 11 deselected ======================
```

Dari 13 tes yang terkumpul, 11 dikesampingkan (*deselected*) dan hanya 2 yang namanya memuat “kolam” dijalankan.

BAGIAN TEKNIS (boleh dilewati)

Tata letak yang disarankan pytest untuk proyek yang akan dipasang adalah `src/` layout: paket di `src/inventaris/` dan tes di `tes/`, lalu dipasang dengan `pip install -e .` ("editable": perubahan kode langsung terbaca tanpa memasang ulang). Buku ini memakai tata letak datar (paket langsung di `proyek/`) dengan `pythonpath = ["."]` karena lebih sederhana untuk pemula. Dokumentasi pytest juga menyarankan `--import-mode=importlib` untuk proyek baru; opsi itu tidak dipakai di sini.

Proyek Bersambung I1: tes untuk paket inventaris

Sekarang seluruh tes proyek dijalankan sekaligus. Lihat kembali `unit_contoh` dari Bab 16: kelima unit itu kini dijaga oleh tes.

```
# Bab 18 - Proyek bersambung I1: seluruh tes paket inventaris
from bantu import FOLDER, jalankan_pytest

teks, kode = jalankan_pytest(["-v"], FOLDER / "proyek")
print(teks)
print("kode keluar:", kode)
```

```
============================= test session starts ==============================
testpaths: tes
collecting ... collected 13 items

tes/test_barang.py::test_biomassa_ikan PASSED                            [  7%]
tes/test_barang.py::test_kategori_pohon[28.0-kecil] PASSED               [ 15%]
tes/test_barang.py::test_kategori_pohon[32.5-sedang] PASSED              [ 23%]
tes/test_barang.py::test_kategori_pohon[45.2-besar] PASSED               [ 30%]
tes/test_barang.py::test_kategori_pohon[40.0-besar] PASSED               [ 38%]
tes/test_barang.py::test_jumlah_ikan_negatif_ditolak PASSED              [ 46%]
tes/test_laporan.py::test_simpan_ringkasan PASSED                        [ 53%]
tes/test_unit.py::test_luas_nol_ditolak PASSED                           [ 61%]
tes/test_unit.py::test_induk_tidak_bisa_menghitung PASSED                [ 69%]
tes/test_unit.py::test_hasil_kolam PASSED                                [ 76%]
tes/test_unit.py::test_ringkasan_kolam PASSED                            [ 84%]
tes/test_unit.py::test_ringkasan_lima_unit PASSED                        [ 92%]
tes/test_unit.py::test_isi_tidak_bisa_diubah_dari_luar PASSED            [100%]

============================== 13 passed ===============================
kode keluar: 0
```

Semua **13 tes lulus**. Mulai sekarang, setiap kali Anda mengubah kelas `Unit` atau anak-anaknya, jalankan `python -m pytest`. Bila ada angka yang berubah tanpa sengaja, pytest memberi tahu persis di mana. Itulah arti kalimat "aman diubah" di penutup seri.

Latihan

1. Pada `contoh_sederhana/test_hitung.py`, tambahkan tes ketiga: `biomassa_kg(1250, 480)` harus `600`. Jalankan dengan `python -m pytest` di folder itu.
2. Ubah angka harapan pada `contoh_gagal/test_gagal.py` menjadi `pytest.approx(432.4)` lalu jalankan; pastikan tes lulus.
3. Tulis tes `test_hasil_kandang` untuk kelas `Kandang`: tambahkan tiga `Ternak` (350, 412,5, dan 389,0 kg) dan periksa hasilnya 1151,5.
4. **Tantangan:** tulis tes berparameter untuk `Pohon.kategori()` dengan lima diameter, termasuk batas 30,0. Lalu sengaja ubah `>= 30` menjadi `> 30` di `barang.py`, jalankan tes, lihat tes batas gagal, dan kembalikan kodenya.

Ringkasan Bab

- **Tes otomatis** memeriksa kode Anda tiap kali dijalankan, jadi perubahan tidak merusak diam-diam.
- **`assert`** adalah dasarnya; berkas `test_*.py` berisi fungsi `test_*` dibaca otomatis oleh **pytest**.
- Tes gagal ditandai `F`; baris `>` menunjuk assert, baris `E` memberi nilai sebenarnya dan harapannya. Kode keluar 1.
- **Fixture** (`conftest.py`), **`parametrize`**, **`pytest.approx`**, **`pytest.raises`**, dan **`tmp_path`** adalah alat sehari-hari.
- Taruh tes di folder `tes/` di samping paket; atur lewat `[tool.pytest.ini_options]` di `pyproject.toml`.
- Paket `inventaris` kini punya 13 tes yang lulus.

Selanjutnya, di Bab 19: kualitas kode, cara merapikan gaya penulisan otomatis dengan ruff, mencatat jalannya program dengan logging, dan mendokumentasikan proyek.

Bab 19 — Kualitas Kode: Rapi, Tercatat, dan Terdokumentasi

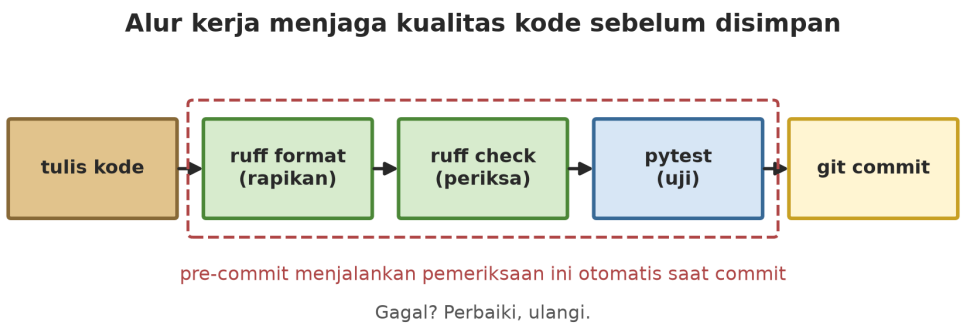
Seri I1 — Kode yang Rapi dan Terawat

Di bab ini, Anda akan:

- Mengenal **PEP 8**, panduan gaya penulisan Python
- Memakai **ruff** untuk memeriksa (*linter*) dan merapikan (*formatter*) kode secara otomatis
- Memasang **pre-commit** supaya pemeriksaan berjalan sendiri sebelum kode disimpan
- Mencatat jalannya program dengan **logging**, bukan `print`
- Mengenal cara debugging di VS Code dan Jupyter, serta dokumentasi proyek dengan MkDocs dan Sphinx

Semua skrip bab ini ada di folder `kode/bab19/`. Di versi web, Anda juga bisa mengunduh semuanya sekaligus (zip).

Kode lebih sering **dibaca** daripada ditulis: oleh rekan kerja, dan oleh Anda sendiri enam bulan lagi. Bayangkan buku catatan lapangan. Bila tiap petugas menulis dengan tata letak sendiri-sendiri, mencari satu angka jadi menyiksa. Kode yang seragam dan rapi mudah dibaca, dan kesalahan bodoh lebih cepat ketahuan. Bab 18 menjaga agar kode **benar**; bab ini menjaga agar kode **rapi, tercatat, dan bisa dipahami**.



Gambar 0.1: Alur menjaga kualitas sebelum kode disimpan: rapikan, periksa, uji, baru commit.

PEP 8: tata bahasa kode Python

PEP 8 adalah panduan gaya resmi Python (PEP = *Python Enhancement Proposal*, usulan peningkatan Python). Ringkasannya, dari dokumen resminya (`peps.python.org/pep-0008`, dibaca 2026-10-10):

Hal	Aturan PEP 8
Menjorok	4 spasi per tingkat
Panjang baris	maksimal 79 karakter (docstring dan komentar 72)
Nama fungsi dan variabel	huruf kecil dengan garis bawah: `hitung_biomassa`
Nama kelas	tiap kata berawalan huruf besar (*CapWords*): `KolamIkan`

Hal	Aturan PEP 8
Konstanta	huruf besar semua: `BATAS_PH`
Impor	di atas berkas, satu per baris; kelompokkan: pustaka standar, paket luar, kode sendiri
Spasi	satu spasi di sekitar `=`, `==`, `and`; dua baris kosong antar fungsi dan kelas

Hafalkan prinsipnya, bukan angkanya: **konsisten dan mudah dibaca**. Untungnya, Anda tidak perlu menghafal. Ada alat yang memeriksanya.

ruff: pemeriksa sekaligus perapi

ruff adalah alat Python (dibuat Astral, pembuat uv) yang menggabungkan dua tugas: **linter** (pemeriksa yang mencari kesalahan dan kebiasaan buruk) dan **formatter** (perapi yang menata tampilan). Ia sudah terpasang di environment buku ini; di komputer Anda: `python -m pip install ruff`. Mari beri makan ruff berkas yang sengaja dibuat berantakan:

```
# Bab 19 - ruff check: mencari masalah di kode (linter)
from bantu import FOLDER, ruff

contoh = FOLDER / "contoh"
print((contoh / "berantakan.py").read_text(encoding="utf-8"))
print("--- ruff check ---")
teks,   kode   =   ruff("check",   "--output-format",   "concise",   "berantakan.py",
folder=contoh)
print(teks)
print("kode keluar:", kode)
```

```
import os, sys
import json

def hitung_biomassa(jumlah,bobot_g,riwayat=[]):
    l = jumlah*bobot_g/1000
    tidak_dipakai = 5
    if jumlah == True:
        print( "jumlah aneh" )
    riwayat.append(l)
    print("Biomassa: %s kg" % l)
    return l

class kolam_ikan:
    def __init__(self,nama): self.nama=nama

--- ruff check ---
berantakan.py:1:1: E401 [*] Multiple imports on one line
berantakan.py:1:1: I001 [*] Import block is un-sorted or un-formatted
berantakan.py:1:8: F401 [*] `os` imported but unused
berantakan.py:1:12: F401 [*] `sys` imported but unused
berantakan.py:2:8: F401 [*] `json` imported but unused
berantakan.py:4:44: B006 Do not use mutable data structures for argument defaults
berantakan.py:5:5: E741 Ambiguous variable name: `l`
berantakan.py:6:5: F841 Local variable `tidak_dipakai` is assigned to but never used
berantakan.py:7:8: E712 Avoid equality comparisons to `True`; use `jumlah:` for truth
checks
berantakan.py:10:11: UP031 Use format specifiers instead of percent format
berantakan.py:13:7: N801 Class name `kolam_ikan` should use CapWords convention
Found 11 errors.
```

```
[*] 5 fixable with `--fix` (4 unsafe fixes available with `--fix --unsafe-fixes`).
kode keluar: 1
```

Setiap baris laporan berbentuk `berkas:baris:kolom: KODE pesan`. Beberapa yang muncul:

Kode	Arti
`E401`	beberapa impor dalam satu baris (`import os, sys`)
`I001`	urutan impor belum rapi
`F401`	`import` yang tidak pernah dipakai
`F841`	variabel diisi tetapi tidak pernah dipakai
`E712`	membandingkan dengan `True` memakai `==`
`E741`	nama yang membingungkan: huruf `l` mirip angka 1
`B006`	nilai bawaan parameter berupa list kosong (jebakan: list-nya **dipakai bersama** oleh semua pemanggilan)
`UP031`	format teks gaya lama dengan `%`; gaya baru f-string
`N801`	nama kelas `kolam_ikan` seharusnya `KolamIkan`

Tanda `[*]` berarti ruff **bisa memperbaikinya sendiri**. Kode keluar 1 menandakan ada masalah, jadi cocok dipakai di skrip otomatis.

> **INGAT**
>
> Aturan mana yang diperiksa ditentukan di `pyproject.toml`. Di proyek kita: `select = ["E", "F", "I", "UP", "B", "N"]` (kelompok aturan pycodestyle, pyflakes, impor, pembaruan sintaks, kesalahan umum, dan penamaan) dengan `line-length = 88`. Bawaan ruff memeriksa banyak aturan, dan himpunannya berubah antarversi, jadi menuliskannya sendiri membuat hasil stabil.

ruff format: merapikan tampilan

Sekarang bagian perapi. Opsi `--diff` memperlihatkan perubahan yang akan dibuat **tanpa mengubah berkas**:

```
# Bab 19 - ruff format --diff: melihat apa yang akan dirapikan, tanpa mengubah berkas
from bantu import FOLDER, ruff

teks, kode = ruff("format", "--diff", "berantakan.py", folder=FOLDER / "contoh")
print(teks)
print("kode keluar:", kode)
```

```
--- berantakan.py
+++ berantakan.py
@@ -1,14 +1,17 @@
 import os, sys
 import json

-def hitung_biomassa(jumlah,bobot_g,riwayat=[]):
-    l = jumlah*bobot_g/1000
+
+def hitung_biomassa(jumlah, bobot_g, riwayat=[]):
+    l = jumlah * bobot_g / 1000
```

```
     tidak_dipakai = 5
     if jumlah == True:
-        print( "jumlah aneh" )
+        print("jumlah aneh")
     riwayat.append(l)
     print("Biomassa: %s kg" % l)
     return l

+
 class kolam_ikan:
-    def __init__(self,nama): self.nama=nama
+    def __init__(self, nama):
+        self.nama = nama

1 file would be reformatted
kode keluar: 1
```

Tanda `-` adalah baris lama, `+` baris baru. Perubahannya murni tampilan: spasi setelah koma, dua baris kosong antar fungsi, isi `__init__` dipindah ke baris sendiri. Artinya program **tidak berubah**, hanya cara menulisnya. Kode keluar 1 di sini berarti "ada yang akan diubah".

ruff check –fix: perbaikan otomatis

```
# Bab 19 - ruff check --fix lalu ruff format, pada salinan berkas
import shutil

from bantu import FOLDER, ruff

kerja = FOLDER / "_tmp" / "perbaiki"
shutil.rmtree(kerja, ignore_errors=True)
shutil.copytree(FOLDER / "contoh", kerja)

teks, _ = ruff("check", "--fix", "--output-format", "concise", "berantakan.py",
folder=kerja)
print(teks)
print("--- ruff format ---")
teks, _ = ruff("format", "berantakan.py", folder=kerja)
print(teks)
print("--- berkas setelah dirapikan ---")
print((kerja / "berantakan.py").read_text(encoding="utf-8"))
```

```
berantakan.py:2:44: B006 Do not use mutable data structures for argument defaults
berantakan.py:3:5: E741 Ambiguous variable name: `l`
berantakan.py:4:5: F841 Local variable `tidak_dipakai` is assigned to but never used
berantakan.py:5:8: E712 Avoid equality comparisons to `True`; use `jumlah:` for truth
checks
berantakan.py:8:11: UP031 Use format specifiers instead of percent format
berantakan.py:11:7: N801 Class name `kolam_ikan` should use CapWords convention
Found 10 errors (4 fixed, 6 remaining).
4 unsafe fixes available with `--fix --unsafe-fixes`.
--- ruff format ---
1 file reformatted
--- berkas setelah dirapikan ---
def hitung_biomassa(jumlah, bobot_g, riwayat=[]):
    l = jumlah * bobot_g / 1000
    tidak_dipakai = 5
    if jumlah == True:
        print("jumlah aneh")
    riwayat.append(l)
```

```
    print("Biomassa: %s kg" % l)
    return l


class kolam_ikan:
    def __init__(self, nama):
        self.nama = nama
```

Perhatikan apa yang terjadi. `--fix` hanya menjalankan perbaikan **aman**: impor yang tak terpakai dibuang, dan urutan impor dirapikan (pada berkas hasil, baris impor di atas hilang). Enam masalah **tersisa**, dan ruff mencatat bahwa ada empat perbaikan "tidak aman" (*unsafe fixes*) yang hanya dijalankan bila Anda memintanya dengan `--unsafe-fixes`. Alasannya: ubah `riwayat=[]` atau `== True` akan **mengubah perilaku** program, dan keputusan itu ada di tangan Anda, bukan alat. Lalu `ruff format` merapikan tampilan.

Sisanya diperbaiki dengan tangan. Berikut hasil akhirnya, diperiksa ulang:

```
# Bab 19 - Sisa masalah diperbaiki dengan tangan (rapi.py), lalu diperiksa ulang
from bantu import FOLDER, ruff

contoh = FOLDER / "contoh"
print((contoh / "rapi.py").read_text(encoding="utf-8"))
print("--- ruff check ---")
print(ruff("check", "rapi.py", folder=contoh)[0])
print("--- ruff format --check ---")
print(ruff("format", "--check", "rapi.py", folder=contoh)[0])
```

```
def hitung_biomassa(jumlah, bobot_g, riwayat=None):
    """Menghitung biomassa ikan (kg) dan mencatatnya di riwayat."""
    if riwayat is None:
        riwayat = []
    biomassa_kg = jumlah * bobot_g / 1000
    if jumlah <= 0:
        print("jumlah tidak masuk akal")
    riwayat.append(biomassa_kg)
    print(f"Biomassa: {biomassa_kg} kg")
    return biomassa_kg


class KolamIkan:
    def __init__(self, nama):
        self.nama = nama

--- ruff check ---
All checks passed!
--- ruff format --check ---
1 file already formatted
```

Apa yang kita putuskan sendiri: `riwayat=None` (list dibuat baru di dalam fungsi), `l` menjadi `biomassa_kg` (nama yang bercerita), variabel tak terpakai dihapus, `== True` diganti pemeriksaan yang masuk akal (`jumlah <= 0`; maksud kode aslinya memang tidak jelas), `%` menjadi f-string, dan `kolam_ikan` menjadi `KolamIkan`. Semua pemeriksaan kini lulus ("All checks passed!").

> TIPS
>
> Biasakan dua perintah ini sebelum menyimpan pekerjaan: `ruff check --fix .` lalu `ruff format .`.

pre-commit: penjaga pintu otomatis

Anda pasti suatu saat lupa menjalankan ruff. **pre-commit** adalah alat yang memasang "penjaga" di git: setiap kali Anda `git commit` (menyimpan perubahan; dibahas di Bab 20), serangkaian pemeriksaan

berjalan otomatis. Bila ada yang gagal, commit **dibatalkan**. Pasang dengan `python -m pip install pre-commit`.

Pengaturannya di berkas `.pre-commit-config.yaml`. Pada dokumentasi resminya (`pre-commit.com`, dibaca 2026-10-10), kebanyakan contoh mengambil hook dari repo di internet (`repo: https://...` dengan `rev:`). Agar skrip kita tidak bergantung pada internet, di sini dipakai **hook lokal** (`repo: local`) yang menjalankan ruff yang sudah ada di komputer. Skrip berikut membuat repo git sementara di `_tmp/`, memasang penjaga, lalu mencoba dua commit.

```
# Bab 19 - pre-commit: pemeriksaan otomatis setiap kali git commit (di repo sementara)
import os
import shutil
import subprocess
import sys
from pathlib import Path

from bantu import FOLDER, hapus_folder

repo = FOLDER / "_tmp" / "repo_pre_commit"
hapus_folder(repo)
repo.mkdir(parents=True)
config_kosong = repo.parent / "gitconfig_kosong"
config_kosong.write_text("", encoding="utf-8")

# Git sementara: tidak membaca konfigurasi git pengguna; identitas lewat variabel
lingkungan
folder_python = str(Path(sys.executable).parent)
env = dict(
    os.environ,
    GIT_CONFIG_GLOBAL=str(config_kosong), GIT_CONFIG_NOSYSTEM="1",
    GIT_AUTHOR_NAME="Uji", GIT_AUTHOR_EMAIL="uji@contoh.invalid",
    GIT_COMMITTER_NAME="Uji", GIT_COMMITTER_EMAIL="uji@contoh.invalid",
     PATH=folder_python + os.pathsep + os.environ["PATH"],       # supaya perintah
'ruff' ditemukan
    PYTHONUTF8="1", NO_COLOR="1",
)


def jalan(*perintah):
    hasil = subprocess.run(perintah, cwd=repo, env=env, text=True, encoding="utf-8",
                           stdout=subprocess.PIPE, stderr=subprocess.STDOUT)
    return hasil.stdout.strip(), hasil.returncode


(repo / ".pre-commit-config.yaml").write_text("""\
repos:
  - repo: local
    hooks:
      - id: ruff-periksa
        name: ruff periksa
        entry: ruff check
        language: unsupported
        types: [python]
      - id: ruff-format
        name: ruff format
        entry: ruff format --check
        language: unsupported
        types: [python]
""", encoding="utf-8")
shutil.copy(FOLDER / "contoh" / "berantakan.py", repo / "berantakan.py")
shutil.copy(FOLDER / "contoh" / "pyproject.toml", repo / "pyproject.toml")
```

```
jalan("git", "init", "--quiet", "-b", "main")
print(jalan(sys.executable,  "-m",  "pre_commit",  "install")[0].replace(str(repo),
"<repo>"))

jalan("git", "add", ".")
teks, kode = jalan("git", "commit", "-m", "tambah berantakan.py")
print("--- commit pertama (kode berantakan) ---")
print("\n".join(b for b in teks.splitlines() if b.startswith(("ruff", "- hook id", "-
exit code"))))
print("commit berhasil?", kode == 0)

shutil.copy(FOLDER / "contoh" / "rapi.py", repo / "berantakan.py")
jalan("git", "add", ".")
teks, kode = jalan("git", "commit", "-m", "perbaiki berantakan.py")
print("--- commit kedua (kode sudah rapi) ---")
print("\n".join(b for b in teks.splitlines() if b.startswith("ruff")))
print("commit berhasil?", kode == 0)
print("--- riwayat ---")
print(jalan("git", "log", "--format=%s")[0])
```

```
pre-commit installed at .git\hooks\pre-commit
--- commit pertama (kode berantakan) ---
ruff periksa.....................................................Failed
- hook id: ruff-periksa
- exit code: 1
ruff format......................................................Failed
- hook id: ruff-format
- exit code: 1
commit berhasil? False
--- commit kedua (kode sudah rapi) ---
ruff periksa.....................................................Passed
ruff format......................................................Passed
commit berhasil? True
--- riwayat ---
perbaiki berantakan.py
```

Jalan ceritanya:

- `pre-commit install` menaruh skrip penjaga di `.git/hooks/pre-commit`.
- **Commit pertama** (kode berantakan): kedua hook menjawab `Failed`, dan commit **tidak terjadi** (`False`).
- Berkas diganti dengan versi rapi, lalu **commit kedua**: keduanya `Passed`, commit berhasil, dan riwayat git berisi satu entri.

Dokumentasi pre-commit mencatat bahwa nilai `language: system` kini hanya alias bagi `unsupported` (sejak versi 4.4.0) dan akan dihapus; konfigurasi baru sebaiknya memakai `unsupported`, seperti di skrip ini. Untuk pre-commit versi lebih lama, ganti ke `system`. Repo sementara ini memakai git yang dipisahkan dari konfigurasi git Anda (lihat Bab 20).

Logging: catatan harian program

`print()` cocok untuk percobaan, tetapi di program sungguhan Anda butuh yang lebih teratur: **logging**, modul bawaan untuk mencatat peristiwa dengan **tingkat kepentingan**. Menurut dokumentasi Python (HOWTO Logging, dibaca 2026-10-10):

Tingkat	Nilai	Pemakaian
`DEBUG`	10	detail teknis untuk mencari bug
`INFO`	20	penegasan bahwa semuanya berjalan wajar
`WARNING`	30	sesuatu yang tak terduga, program masih jalan

Tingkat	Nilai	Pemakaian
`ERROR`	40	ada fungsi yang gagal dikerjakan
`CRITICAL`	50	kesalahan serius

Pencatat hanya meneruskan pesan yang tingkatnya **sama atau lebih tinggi** dari tingkat yang berlaku. Tingkat bawaan pencatat induk (*root*) adalah `WARNING`, sehingga tanpa pengaturan `INFO` dan `DEBUG` tidak tampil.

```
# Bab 19 - logging: catatan harian program dengan tingkat kepentingan
import logging
import sys

logging.basicConfig(
    level=logging.INFO,
    format="%(levelname)-8s | %(name)s | %(message)s",
    stream=sys.stdout,
)
catat = logging.getLogger("kolam")

catat.debug("detail teknis (tidak tampil pada level INFO)")
catat.info("kolam A: 800 ikan ditebar")
catat.warning("pH 5.9 di bawah batas 6.5")
catat.error("sensor suhu tidak menjawab")

try:
    rata_rata = 540.5 / 0
except ZeroDivisionError:
    catat.exception("hitungan rata-rata gagal")      # sekaligus mencetak traceback

print("level yang berlaku:", logging.getLevelName(catat.getEffectiveLevel()))
```

```
INFO     | kolam | kolam A: 800 ikan ditebar
WARNING  | kolam | pH 5.9 di bawah batas 6.5
ERROR    | kolam | sensor suhu tidak menjawab
ERROR    | kolam | hitungan rata-rata gagal
Traceback (most recent call last):
  File "05_logging_dasar.py", line 18, in <module>
    rata_rata = 540.5 / 0
                ~~~~~~^~~
ZeroDivisionError: division by zero
level yang berlaku: INFO
```

- `basicConfig(level=..., format=..., stream=...)` mengatur tingkat dan bentuk catatan. Bentuk `%(levelname)-8s` artinya nama tingkat, rata kiri, selebar 8 huruf.
- Pesan `DEBUG` **tidak tampil** karena tingkat diatur `INFO`. Tiga lainnya tampil.
- `catat.exception(...)` dipanggil di dalam `except`: ia mencatat pesan pada tingkat `ERROR` **sekaligus traceback** error-nya. Inilah pengganti `print` yang paling berguna.
- `getEffectiveLevel()` menunjukkan tingkat yang berlaku: `INFO`.

Logging di paket inventaris

Dokumentasi Python menyarankan tiap modul memakai `logger = logging.getLogger(__name__)`. Nama pencatat lalu mengikuti hierarki paket (`inventaris.unit` adalah anak `inventaris`), dan pesan naik ke pencatat induknya. Dokumentasi juga menyarankan pustaka **tidak** memasang pengatur keluaran sendiri: itu urusan aplikasi pemakai. Itulah yang kita ikuti: paket `inventaris` hanya **mencatat** (`logger.debug` di `Unit.tambah`, `logger.info` di `simpan_ringkasan`), dan skrip memutuskan ke mana catatan pergi.

```
# Bab 19 - logging di paket inventaris: layar menampilkan INFO, berkas menyimpan DEBUG
import logging
import sys
from pathlib import Path

sys.path.insert(0, str(Path(__file__).parent / "proyek"))

from inventaris.contoh import buat_contoh  # noqa: E402
from inventaris.laporan import simpan_ringkasan  # noqa: E402

kerja = Path(__file__).parent / "_tmp"
kerja.mkdir(exist_ok=True)
berkas_log = kerja / "inventaris.log"
berkas_log.unlink(missing_ok=True)

format_log = "%(levelname)s %(name)s: %(message)s"
layar = logging.StreamHandler(sys.stdout)
layar.setLevel(logging.INFO)
berkas = logging.FileHandler(berkas_log, encoding="utf-8")
berkas.setLevel(logging.DEBUG)
for pencatat in (layar, berkas):
    pencatat.setFormatter(logging.Formatter(format_log))

akar = logging.getLogger("inventaris")          # induk dari inventaris.unit dan
inventaris.laporan
akar.setLevel(logging.DEBUG)
akar.addHandler(layar)
akar.addHandler(berkas)

unit = buat_contoh()
simpan_ringkasan(unit, kerja / "ringkasan.txt")
berkas.close()

isi = berkas_log.read_text(encoding="utf-8").splitlines()
print(f"Isi berkas log: {len(isi)} baris")
print("Baris pertama :", isi[0])
print("Baris terakhir:", isi[-1])
```

```
INFO inventaris.laporan: menulis 5 ringkasan ke ringkasan.txt
Isi berkas log: 11 baris
Baris pertama : DEBUG inventaris.unit: Plot hutan A menerima Pohon
Baris terakhir: INFO inventaris.laporan: menulis 5 ringkasan ke ringkasan.txt
```

Dua penampung (*handler*) dipasang pada pencatat `inventaris`: layar menerima tingkat `INFO` ke atas, dan berkas menerima semuanya sampai `DEBUG`. Hasilnya: layar hanya menampilkan satu baris (`menulis 5 ringkasan ...`), sedangkan berkas log berisi **11 baris**: 10 catatan `DEBUG` (tiap kali `tambah` dipanggil: 3 pohon + 2 + 1 + 1 + 3 = 10) ditambah 1 catatan `INFO`.

Debugging: mencari bug dengan teliti

Debugging adalah menjalankan program sambil berhenti di titik tertentu untuk melihat isi variabel. Cara paling dasar sudah ada di Python: fungsi bawaan `breakpoint()` menghentikan program dan membuka *debugger* `pdb` di terminal.

```
# Bab 19 - Tempat memasang titik henti (breakpoint) saat mencari bug
def biomassa_kg(jumlah, bobot_g):
    total_g = jumlah * bobot_g
    breakpoint()                    # program berhenti di sini; ketik "p total_g" lalu
"c" untuk lanjut
    return total_g / 1000
```

```
print(biomassa_kg(800, 540.5))
```

Kode ini **tidak dijalankan** saat buku dibuat karena `breakpoint()` menghentikan program dan menunggu perintah dari pengguna, sehingga tidak cocok untuk build otomatis. Penulis mencobanya sekali dengan masukan `p total_g` lalu `c`: program berhenti di baris `breakpoint()`, `p total_g` mencetak `432400.0` (gram), dan `c` (*continue*) melanjutkan hingga mencetak `432.4`.

Di VS Code, cara yang lebih nyaman (deskripsi dari dokumentasi resmi VS Code, dibaca 2026-10-10; **tidak dijalankan oleh penulis**):

- Pasang ekstensi Python (ekstensi *Python Debugger* ikut terpasang bersamanya).
- Klik di margin kiri editor untuk memasang **breakpoint** (titik henti). Breakpoint bisa diberi **syarat** atau hitungan, misalnya hanya berhenti setelah lima kali.
- Tekan **F5** (*Start Debugging*) atau pilih *Debug Python File*. Program berhenti di breakpoint, dan Anda bisa memeriksa variabel.
- Pengaturan lebih rinci disimpan di `.vscode/launch.json` (nama, `program` yang dijalankan, `args`, dan lain-lain).

Di notebook Jupyter dalam VS Code (juga dari dokumentasi resmi, tidak dijalankan): tersedia **Run by Line** untuk menjalankan sel satu pernyataan demi satu pernyataan, dan **Debug Cell** untuk debugger penuh dengan breakpoint. Keduanya membutuhkan ipykernel versi 6 ke atas.

> ⚠ AWAS
>
> Debugger menjawab pertanyaan "mengapa nilainya begini **di titik ini**?". Jangan memakainya sebagai pengganti tes. Tes (Bab 18) mencegah bug yang sama muncul lagi; debugger hanya membantu menemukannya sekali.

Dokumentasi proyek: MkDocs dan Sphinx

Kode tanpa petunjuk pemakaian cepat dilupakan. Dua alat populer mengubah berkas teks menjadi situs dokumentasi:

- **MkDocs** (`mkdocs.org`): menulis halaman dalam **Markdown**, lalu `mkdocs build` menghasilkan situs HTML statis di folder `site/`. `mkdocs serve` membuka pratinjau lokal. Satu-satunya pengaturan wajib di `mkdocs.yml` adalah `site_name`; `nav` mengatur urutan menu (dokumentasi resmi, dibaca 2026-10-10).
- **Sphinx** (`sphinx-doc.org`): alat yang lebih tua dan lebih kaya, memakai **reStructuredText**. `sphinx-quickstart` membuat kerangka (`conf.py`, `index.rst`), `make html` membangun situs, dan ekstensi **autodoc** menarik dokumentasi langsung dari docstring (dokumentasi resmi, dibaca 2026-10-10). Sphinx tidak dipasang atau dijalankan di bab ini.

Skrip berikut membangun situs MkDocs mini sungguhan:

```
# Bab 19 - MkDocs: halaman dokumentasi dari berkas Markdown (membangun situs sungguhan
di _tmp)
import shutil
import sys
from pathlib import Path

from bantu import FOLDER, jalankan

situs = FOLDER / "_tmp" / "situs_dokumentasi"
shutil.rmtree(situs, ignore_errors=True)
(situs / "docs").mkdir(parents=True)

(situs / "mkdocs.yml").write_text("""\
site_name: Dokumentasi Inventaris
nav:
  - Beranda: index.md
```

```
  - Cara memakai: cara-memakai.md
""", encoding="utf-8")
(situs / "docs" / "index.md").write_text(
     "# Inventaris\n\nPaket contoh untuk mencatat hutan, kebun, sawah, kolam, dan
kandang.\n",
    encoding="utf-8")
(situs / "docs" / "cara-memakai.md").write_text(
    "# Cara memakai\n\n```python\nfrom inventaris.contoh import buat_contoh\n```\n",
    encoding="utf-8")

teks, kode = jalankan([sys.executable, "-m", "mkdocs", "build", "--quiet"], situs)
print("kode keluar:", kode, "| pesan:", teks or "(tidak ada)")

hasil = sorted(p.relative_to(situs / "site").as_posix() for p in (situs /
"site").rglob("*.html"))
print("Halaman HTML yang dihasilkan:", hasil)
html = (situs / "site" / "index.html").read_text(encoding="utf-8")
print("Judul situs ada di halaman?", "Dokumentasi Inventaris" in html)
```

```
kode keluar: 0 | pesan: (tidak ada)
Halaman HTML yang dihasilkan: ['404.html', 'cara-memakai/index.html', 'index.html']
Judul situs ada di halaman? True
```

Kode keluar 0 dan tanpa pesan berarti pembangunan lancar. Tiga halaman HTML dihasilkan: `index.html` dan `cara-memakai/index.html` dari dua berkas Markdown kita, ditambah `404.html` yang dibuat MkDocs sendiri. Judul situs dari `mkdocs.yml` memang tertulis di halaman.

> **i** INGAT
>
> **Docstring** yang Anda tulis sejak Bab 10 adalah dokumentasi paling murah: ia ikut terbaca oleh `help()`, editor, dan alat seperti Sphinx autodoc. Menulis docstring satu kalimat di tiap fungsi dan kelas sudah banyak membantu.

Proyek Bersambung I1: merapikan inventaris

Sekarang semua alat dipakai pada paket `inventaris`, yang sudah memiliki logging dan 14 tes (satu tes baru memeriksa catatan log dengan fixture `caplog`). Skrip menyalin proyek ke `_tmp/`, mengecek, memperbaiki, dan mengujinya ulang.

```
# Bab 19 - Proyek bersambung I1: merapikan paket inventaris dengan ruff, lalu menguji
ulang
import re
import shutil
import sys

from bantu import FOLDER, jalankan, ruff

kerja = FOLDER / "_tmp" / "proyek_rapi"
shutil.rmtree(kerja, ignore_errors=True)
shutil.copytree(FOLDER                          /                          "proyek",
kerja, ignore=shutil.ignore_patterns("__pycache__"))


def periksa(judul):
    teks, _ = ruff("check", "--output-format", "concise", ".", folder=kerja)
    print(f"--- {judul} ---")
    print(teks)
```

```
periksa("ruff check, sebelum")

ruff("check", "--fix", "--quiet", ".", folder=kerja)      # perbaikan aman otomatis
(urutan impor)
print("--- ruff format ---")
print(ruff("format", ".", folder=kerja)[0])
periksa("ruff check, setelah perbaikan otomatis")

# Dua docstring terlalu panjang: ruff tidak mau menebak kalimat Anda, jadi dipersingkat
dengan tangan
perbaikan = {
    "inventaris/__init__.py": (
         "Paket inventaris: sistem informasi hutan, kebun, sawah, kolam, dan kandang
(data fiktif).",
        "Paket inventaris: data fiktif hutan, kebun, sawah, kolam, dan kandang.",
    ),
    "inventaris/barang.py": (
      "Besar bila diameter 40 cm ke atas, sedang bila 30 cm ke atas, selain itu kecil.",
        "Besar mulai 40 cm, sedang mulai 30 cm, selain itu kecil.",
    ),
}
for jalur, (lama, baru) in perbaikan.items():
    berkas = kerja / jalur
          berkas.write_text(berkas.read_text(encoding="utf-8").replace(lama,   baru),
encoding="utf-8")
periksa("ruff check, setelah perbaikan tangan")
print(ruff("format", "--check", ".", folder=kerja)[0])

teks, kode = jalankan([sys.executable, "-m", "pytest", "-q", "-p", "no:cacheprovider"],
kerja)
print("--- pytest ---")
print(re.sub(r" in \d+\.\d+s", "", teks.splitlines()[-1]))
```

```
--- ruff check, sebelum ---
inventaris\__init__.py:1:89: E501 Line too long (95 > 88)
inventaris\barang.py:14:89: E501 Line too long (93 > 88)
inventaris\contoh.py:3:1: I001 [*] Import block is un-sorted or un-formatted
inventaris\contoh.py:3:89: E501 Line too long (92 > 88)
inventaris\unit.py:22:89: E501 Line too long (92 > 88)
Found 5 errors.
[*] 1 fixable with `--fix`.
--- ruff format ---
3 files reformatted, 6 files left unchanged
--- ruff check, setelah perbaikan otomatis ---
inventaris\__init__.py:1:89: E501 Line too long (95 > 88)
inventaris\barang.py:14:89: E501 Line too long (93 > 88)
Found 2 errors.
--- ruff check, setelah perbaikan tangan ---
All checks passed!
9 files already formatted
--- pytest ---
14 passed
```

Telusuri:

- **Sebelum**: 5 temuan, hampir semuanya `E501` (baris lebih dari 88 karakter) dan satu `I001` (urutan impor).
- `--fix` dan `format` menyelesaikan tiga temuan (impor dirapikan, baris panjang dipecah per baris). Sisa **2** adalah docstring yang terlalu panjang: ruff tidak mau menebak kalimat Anda, jadi kita persingkat dengan tangan.
- Sesudahnya semua pemeriksaan lulus, 9 berkas rapi, dan **14 tes tetap lulus**: merapikan tidak merusak perilaku. Itulah gunanya tes di Bab 18.

Latihan

1. Jalankan `ruff check` pada skrip lama Anda (misalnya dari Bab 10). Berapa temuan? Pilih satu kode dan cari artinya dengan `python -m ruff rule F841`.
2. Ubah `05_logging_dasar.py` agar tingkatnya `DEBUG`. Pesan apa yang bertambah?
3. Tambahkan `logger.warning(...)` pada `Kolam` bila `ph_air` di luar 6,5 sampai 8,5, lalu amati lewat `06_logging_proyek.py`. (Batas ini hanya angka contoh.)
4. **Tantangan:** ubah `07_pre_commit.py` supaya berkas pertama hanya berisi `import os` dan satu fungsi rapi. Hook mana yang gagal, dan hook mana yang lolos? Jelaskan alasannya.

Ringkasan Bab

- **PEP 8** adalah panduan gaya resmi Python: konsisten dan mudah dibaca.
- **ruff check** memeriksa (kode `E`, `F`, `I`, `B`, ...), **`--fix`** hanya memperbaiki yang aman, **ruff format** menata tampilan tanpa mengubah perilaku.
- **pre-commit** menjalankan pemeriksaan otomatis tiap `git commit` dan membatalkan commit bila gagal.
- **logging** mencatat dengan tingkat DEBUG, INFO, WARNING, ERROR, CRITICAL; modul memakai `getLogger(__name__)`, aplikasi yang memilih tujuan catatan.
- **Debugging**: `breakpoint()` (pdb) atau fitur VS Code dan Jupyter; mencari bug, bukan pengganti tes.
- **MkDocs** (Markdown) dan **Sphinx** (reStructuredText) membuat situs dokumentasi dari berkas teks dan docstring.

Selanjutnya, di Bab 20: git, GitHub, dan alur kerja proyek: menyimpan riwayat, bercabang, bekerja bersama, dan menutup Seri I1.

Bab 20 — Git, GitHub, dan Alur Kerja Proyek: Mesin Waktu untuk Kode Anda

Seri I1 — Kode yang Rapi dan Terawat

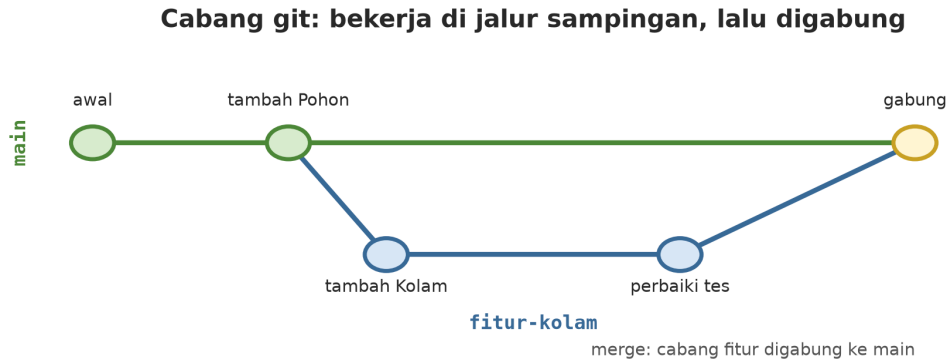
Di bab ini, Anda akan:

- Memahami **git**: menyimpan riwayat versi dengan **commit**, melihat perbedaan, dan membatalkan perubahan
- Bekerja di **branch** (cabang), menggabungkannya dengan **merge**, dan menyelesaikan **konflik**
- Menjaga rahasia keluar dari repo dengan **.gitignore**
- Mengenal **GitHub**: repo, issue, pull request, dan review, serta **GitHub Actions** dasar
- Menutup Seri I1: proyek inventaris menjadi kelas, bertes, rapi, dan tersimpan di git

Semua skrip bab ini ada di folder `kode/bab20/`. Di versi web, Anda juga bisa mengunduh semuanya sekaligus (zip).

Pernahkah Anda menyimpan `laporan_final.docx`, lalu `laporan_final2.docx`, lalu `laporan_final_BENAR.docx`? Itu cara manual melacak versi, dan cepat kacau. **Git** adalah alat yang melakukannya dengan rapi: ia mencatat setiap perubahan sebagai **riwayat**, sehingga Anda bisa melihat siapa mengubah apa dan kapan, serta kembali ke keadaan lama bila ada yang salah. Bayangkan buku besar kebun: setiap koreksi dicatat lengkap dengan tanggalnya, tidak pernah dihapus.

Aturan keselamatan bab ini. Semua perintah git di buku ini dijalankan di **repo sementara** di folder `_tmp/`, dengan pengaturan yang terpisah dari konfigurasi git Anda. Tidak ada repo GitHub yang dibuat, tidak ada push, dan tidak ada akun yang dipakai. Bagian GitHub dijelaskan saja.



Gambar 0.1: Cabang git: jalur sampingan lalu digabung ke jalur utama.

Konsep dasar git

Istilah	Artinya
repo (*repository*)	folder proyek yang riwayatnya dilacak git (ada folder tersembunyi `.git`)
commit	satu "foto" keadaan proyek, dengan pesan tentang apa yang berubah

Istilah	Artinya
staging (`git add`)	memilih perubahan mana yang akan masuk commit berikutnya
branch (cabang)	jalur riwayat tersendiri; cabang utama biasa bernama `main`
merge	menggabungkan cabang lain ke cabang yang sedang aktif
remote	salinan repo di tempat lain, misalnya GitHub

Alurnya: Anda **mengubah berkas**, memilihnya dengan **`git add`**, lalu **`git commit`** untuk menyimpannya ke riwayat. Perintah `git status` memberi tahu posisi Anda. (Git mencatat nama dan email pembuat commit. Di komputer Anda, atur sekali dengan `git config --global user.name "Nama Anda"` dan `git config --global user.email "email@anda"`. Skrip buku ini tidak menyentuh pengaturan itu: identitas ujinya dipasang lewat variabel lingkungan.)

```
# Bab 20 - git: repo baru, berkas pertama, commit pertama (di repo sementara)
from bantu import git, repo_baru, tulis

repo = repo_baru("repo_dasar")
tulis(repo, "panen.csv", "blok,hasil_kg\nA,245.5\nB,310.0\n")

print("Status awal   :", git("status", "--short", repo=repo)[0])
git("add", "panen.csv", repo=repo)
print("Setelah add   :", git("status", "--short", repo=repo)[0])
git("commit", "--quiet", "-m", "Catat panen blok A dan B", repo=repo)
print("Setelah commit:", repr(git("status", "--short", repo=repo)[0]))

print("--- git log ---")
print(git("log", "--format=%an: %s", repo=repo)[0])
```

```
Status awal   : ?? panen.csv
Setelah add   : A  panen.csv
Setelah commit: ''
--- git log ---
Uji: Catat panen blok A dan B
```

Telusuri tiga baris status: `??` berarti berkas **belum dilacak**, `A` berarti sudah **di-add** (menunggu commit), dan teks kosong sesudah commit berarti **tidak ada perubahan tertunda**. Perintah `git log --format=...` menampilkan riwayat: satu commit dengan pesannya.

Melihat perbedaan dan membatalkan

```
# Bab 20 - Mengubah berkas, melihat perbedaan (diff), dan membatalkan perubahan
from bantu import git, repo_baru, tulis

repo = repo_baru("repo_diff")
tulis(repo, "panen.csv", "blok,hasil_kg\nA,245.5\nB,310.0\n")
git("add", ".", repo=repo)
git("commit", "--quiet", "-m", "Catat panen blok A dan B", repo=repo)

tulis(repo, "panen.csv", "blok,hasil_kg\nA,245.5\nB,312.0\nC,198.5\n")
print("Status:", git("status", "--short", repo=repo)[0])

teks, _ = git("diff", repo=repo)
print("--- git diff ---")
print("\n".join(b for b in teks.splitlines() if not b.startswith("index ")))
```

```
git("commit", "--quiet", "-a", "-m", "Koreksi blok B dan tambah blok C", repo=repo)
print("--- riwayat ---")
print(git("log", "--format=%s", repo=repo)[0])

tulis(repo, "panen.csv", "salah ketik\n")
git("restore", "panen.csv", repo=repo)               # membuang perubahan yang belum
di-commit
print("--- setelah git restore ---")
print((repo / "panen.csv").read_text(encoding="utf-8"), end="")
```

```
Status:  M panen.csv
--- git diff ---
diff --git a/panen.csv b/panen.csv
--- a/panen.csv
+++ b/panen.csv
@@ -1,3 +1,4 @@
 blok,hasil_kg
 A,245.5
-B,310.0
+B,312.0
+C,198.5
--- riwayat ---
Koreksi blok B dan tambah blok C
Catat panen blok A dan B
--- setelah git restore ---
blok,hasil_kg
A,245.5
B,312.0
C,198.5
```

- `M` (spasi lalu M) berarti berkas **sudah dimodifikasi** tetapi belum di-add.
- **`git diff`** menampilkan beda: baris berawalan `-` dihapus, `+` ditambah. Di sini `B,310.0` berubah menjadi `B,312.0` dan baris blok C ditambahkan. (Baris teknis berawalan `index` dibuang dari tampilan.)
- `git commit -a` menyimpan semua berkas **yang sudah dilacak** yang berubah, tanpa `git add` terpisah.
- **`git restore berkas`** membuang perubahan yang belum di-commit. Hasil akhirnya: berkas kembali ke isi commit terakhir, jadi "salah ketik" hilang. Hati-hati: perubahan yang dibuang dengan cara ini **tidak bisa dikembalikan**.

Branch dan merge

Branch memungkinkan Anda mengerjakan fitur baru di jalur sampingan, tanpa mengganggu `main`. Kalau berhasil, jalur itu digabung. Kalau gagal, cukup dibuang.

```
# Bab 20 - Cabang (branch) dan merge: bekerja di jalur sampingan, lalu menggabungkan
from bantu import git, rapikan, repo_baru, tulis

repo = repo_baru("repo_cabang")
tulis(repo, "unit.txt", "plot hutan\nkebun\n")
git("add", ".", repo=repo)
git("commit", "--quiet", "-m", "Daftar unit awal", repo=repo)

git("switch", "--quiet", "-c", "fitur-kolam", repo=repo)
print("Cabang aktif:", git("branch", "--show-current", repo=repo)[0])
tulis(repo, "kolam.txt", "kolam A: 800 ikan\n")
git("add", ".", repo=repo)
git("commit", "--quiet", "-m", "Tambah data kolam", repo=repo)

git("switch", "--quiet", "main", repo=repo)
print("Di main, kolam.txt ada?", (repo / "kolam.txt").exists())
```

```
tulis(repo, "unit.txt", "plot hutan\nkebun\nkandang\n")
git("commit", "--quiet", "-a", "-m", "Tambah kandang", repo=repo)

teks, _ = git("merge", "--no-edit", "fitur-kolam", repo=repo)
print("--- git merge ---")
print(teks.splitlines()[0])
print("Di main, kolam.txt ada?", (repo / "kolam.txt").exists())
print("--- git log --graph ---")
print(rapikan(git("log", "--graph", "--format=%s", repo=repo)[0]))
```

```
Cabang aktif: fitur-kolam
Di main, kolam.txt ada? False
--- git merge ---
Merge made by the 'ort' strategy.
Di main, kolam.txt ada? True
--- git log --graph ---
*   Merge branch 'fitur-kolam'
|\
| * Tambah data kolam
* | Tambah kandang
|/
* Daftar unit awal
```

Perhatikan bagian yang menarik: sebelum merge, `kolam.txt` **tidak ada** di `main` (ia hidup di cabang `fitur-kolam`). Sesudah merge, ia ada. Diagram `git log --graph` menampilkan bentuknya: dua jalur (`*` di kiri dan `*` di kanan) yang berpisah dari commit "Daftar unit awal" lalu bertemu di commit merge. Perubahan di `main` ("Tambah kandang") dan di cabang ("Tambah data kolam") menyentuh berkas yang berbeda, jadi git menggabungkannya sendiri.

Konflik: bila dua orang mengubah baris yang sama

Bagaimana bila kedua jalur mengubah **baris yang sama**? Git tidak berani memilih; ia berhenti dan meminta manusia memutuskan. Itu disebut **konflik** dan bukan tanda Anda berbuat salah, melainkan hal biasa dalam kerja bersama.

```
# Bab 20 - Konflik merge: dua cabang mengubah baris yang sama
from bantu import git, rapikan, repo_baru, tulis

repo = repo_baru("repo_konflik")
tulis(repo, "kolam.txt", "kolam A\nbobot rata-rata: 500 g\n")
git("add", ".", repo=repo)
git("commit", "--quiet", "-m", "Data kolam awal", repo=repo)

git("switch", "--quiet", "-c", "koreksi-timbang", repo=repo)
tulis(repo, "kolam.txt", "kolam A\nbobot rata-rata: 540 g\n")
git("commit", "--quiet", "-a", "-m", "Bobot hasil timbang ulang: 540 g", repo=repo)

git("switch", "--quiet", "main", repo=repo)
tulis(repo, "kolam.txt", "kolam A\nbobot rata-rata: 520 g\n")
git("commit", "--quiet", "-a", "-m", "Bobot dari catatan lama: 520 g", repo=repo)

teks, kode = git("merge", "--no-edit", "koreksi-timbang", repo=repo)
print("--- git merge ---")
print(teks)
print("kode keluar:", kode)
print("--- isi kolam.txt saat konflik ---")
print((repo / "kolam.txt").read_text(encoding="utf-8"), end="")
print("--- status ---")
print(git("status", "--short", repo=repo)[0])
```

```
# Penyelesaian: manusia memutuskan angka mana yang benar, lalu menghapus penanda
tulis(repo, "kolam.txt", "kolam A\nbobot rata-rata: 540 g\n")
git("add", "kolam.txt", repo=repo)
git("commit", "--quiet", "--no-edit", repo=repo)
print("--- setelah diselesaikan ---")
print(rapikan(git("log", "--graph", "--format=%s", repo=repo)[0]))
```

```
--- git merge ---
Auto-merging kolam.txt
CONFLICT (content): Merge conflict in kolam.txt
Automatic merge failed; fix conflicts and then commit the result.
kode keluar: 1
--- isi kolam.txt saat konflik ---
kolam A
<<<<<<< HEAD
bobot rata-rata: 520 g
=======
bobot rata-rata: 540 g
>>>>>>> koreksi-timbang
--- status ---
UU kolam.txt
--- setelah diselesaikan ---
*   Merge branch 'koreksi-timbang'
|\
| * Bobot hasil timbang ulang: 540 g
* | Bobot dari catatan lama: 520 g
|/
* Data kolam awal
```

Cara membaca berkas yang berkonflik:

- `<<<<<<< HEAD` sampai `=======` adalah versi di cabang **Anda sekarang** (`main`: 520 g dari catatan lama).
- `=======` sampai `>>>>>>> koreksi-timbang` adalah versi dari cabang yang digabung (540 g dari timbang ulang).
- Status `UU` berarti kedua pihak mengubah berkas ini dan belum diselesaikan.

Untuk menyelesaikannya: **sunting berkas**, buang ketiga baris penanda, pilih (atau gabungkan) isi yang benar, lalu `git add` dan `git commit`. Skrip memilih 540 g karena itu hasil timbang ulang. Keputusan **angka mana yang benar** hanya bisa dibuat manusia yang tahu kenyataan di lapangan. Riwayat akhir memperlihatkan commit merge yang menyatukan keduanya.

.gitignore dan menjaga rahasia

Tidak semua berkas pantas masuk git: folder `.venv` (Bab 17), berkas sementara, hasil unduhan besar, dan terutama **rahasia** seperti kata sandi dan kunci API. Daftar yang diabaikan git ditulis di berkas **`.gitignore`**. Rahasia sebaiknya disimpan di berkas `.env` yang diabaikan, atau di variabel lingkungan, dan dibaca program lewat `os.environ`.

```
# Bab 20 - .gitignore dan menjaga rahasia keluar dari repo (nilai kunci di sini
hanya tiruan)
from bantu import git, repo_baru, tulis

repo = repo_baru("repo_rahasia")
tulis(repo, ".env", "API_KEY=[disamarkan]\n")
tulis(repo, "app.py", "import os\nkunci = os.environ['API_KEY']\n")
tulis(repo, "_tmp/hasil.txt", "berkas sementara\n")

print("--- tanpa .gitignore ---")
print(git("status", "--short", repo=repo)[0])
```

```
tulis(repo, ".gitignore", ".env\n_tmp/\n__pycache__/\n.venv/\n")
print("--- dengan .gitignore ---")
print(git("status", "--short", repo=repo)[0])
print("Mengapa .env diabaikan?", git("check-ignore", "-v", ".env", repo=repo)[0])

git("add", ".", repo=repo)
git("commit", "--quiet", "-m", "Awal proyek", repo=repo)
print("--- berkas yang dilacak git ---")
print(git("ls-files", repo=repo)[0])

print("=== Salah langkah: rahasia terlanjur di-commit ===")
repo2 = repo_baru("repo_terlanjur")
tulis(repo2, ".env", "API_KEY=[disamarkan]\n")
git("add", ".", repo=repo2)
git("commit", "--quiet", "-m", "oops", repo=repo2)
tulis(repo2, ".gitignore", ".env\n")
git("add", ".gitignore", repo=repo2)
git("commit", "--quiet", "-m", "tambah .gitignore", repo=repo2)
print("Dilacak setelah .gitignore ditambah:", git("ls-files", repo=repo2)[0].split())
print("Isi commit pertama masih memuat .env:", ".env" in git("show", "--stat", "--
format=", "HEAD~1", repo=repo2)[0])
git("rm", "--cached", "--quiet", ".env", repo=repo2)
git("commit", "--quiet", "-m", "berhenti melacak .env", repo=repo2)
print("Dilacak setelah git rm --cached:", git("ls-files", repo=repo2)[0].split())
print("Tetapi riwayat lama masih memuatnya:", ".env" in git("show", "--stat", "--
format=", "HEAD~2", repo=repo2)[0])
```

```
--- tanpa .gitignore ---
?? .env
?? _tmp/
?? app.py
--- dengan .gitignore ---
?? .gitignore
?? app.py
Mengapa .env diabaikan? .gitignore:1:.env   .env
--- berkas yang dilacak git ---
.gitignore
app.py
=== Salah langkah: rahasia terlanjur di-commit ===
Dilacak setelah .gitignore ditambah: ['.env', '.gitignore']
Isi commit pertama masih memuat .env: True
Dilacak setelah git rm --cached: ['.gitignore']
Tetapi riwayat lama masih memuatnya: True
```

Bagian pertama: tanpa `.gitignore`, git menawarkan `.env` dan `_tmp/` untuk dimasukkan (`??`). Dengan `.gitignore`, keduanya hilang dari daftar. `git check-ignore -v` menjelaskan **aturan mana** yang berlaku: baris 1 di `.gitignore`. Hanya `.gitignore` dan `app.py` yang dilacak.

Bagian kedua adalah **jebakan paling berbahaya**. Bila rahasia sudah **terlanjur di-commit**, menambahkannya ke `.gitignore` **tidak cukup**: berkas tetap dilacak (`['.env', '.gitignore']`). `git rm --cached` menghentikan pelacakan, tetapi **riwayat lama masih memuat isinya** (`True`). Siapa pun yang memegang salinan repo masih bisa membacanya.

⚠ AWAS

Rahasia yang pernah masuk commit harus dianggap **bocor**, terutama bila repo pernah dikirim ke GitHub. Langkah paling aman (saran penulis): **cabut dan ganti kuncinya** di penyedia layanan, baru bersihkan riwayat. Mencegah jauh lebih mudah daripada memperbaiki: buat `.gitignore` **sebelum** commit pertama. Dokumentasi GitHub juga menjelaskan *push protection*, fitur yang memblokir push bila terdeteksi rahasia; ketersediaannya bergantung pada pengaturan akun dan repo (dibaca 2026-10-10), jadi jangan menjadikannya satu-satunya pagar.

GitHub: git untuk bekerja bersama

GitHub adalah layanan situs yang menyimpan repo git di internet (git dan GitHub adalah dua hal berbeda: git alatnya, GitHub salah satu tempat penyimpanannya). Bagian ini hanya penjelasan; **tidak ada yang dijalankan**, karena membuat repo dan mengirim kode mengubah data di internet dan butuh akun.

- **Repo di GitHub** dihubungkan ke repo lokal sebagai **remote** (biasanya bernama `origin`). `git push` mengirim commit ke sana, `git pull` mengambil perubahan orang lain.
- **Issue** adalah catatan tugas, ide, atau bug dalam sebuah repo. Menurut dokumentasi GitHub (dibaca 2026-10-10), issue dapat diberi label dan penanggung jawab, dan kata kunci penutup di pull request dapat menutup issue otomatis saat digabung.
- **Pull request** (PR) adalah usulan menggabungkan perubahan ke proyek. Tab *Files changed* memperlihatkan beda yang diperiksa, tab *Conversation* memuat diskusi dan review, dan status merge menunjukkan hal yang masih menghalangi, seperti persetujuan yang belum ada.
- **Review**: rekan membaca perubahan Anda di PR, memberi komentar, lalu Anda memperbaikinya sebelum digabung.

```
# Bab 20 - Perintah untuk menghubungkan repo lokal ke GitHub (hanya contoh; NAMA-ANDA
diganti nama akun Anda)
import subprocess

perintah = [
    ["git", "remote", "add", "origin", "https://github.com/NAMA-ANDA/inventaris.git"],
    ["git", "push", "-u", "origin", "main"],             # kirim cabang main ke GitHub
    ["git", "switch", "-c", "fitur-kolam"],                # kerja di cabang baru
    ["git", "push", "-u", "origin", "fitur-kolam"],        # kirim cabang fitur
    ["gh", "pr", "create", "--fill"],                        # buka pull request (alat
GitHub CLI)
]
for p in perintah:
    print("$", " ".join(p))
    subprocess.run(p, check=True)
```

Kode ini **tidak dijalankan** saat buku dibuat karena membuat repo dan mengirim (push) kode ke GitHub butuh akun GitHub dan mengubah data di internet, jadi keluarannya tidak ditampilkan. Perintah `gh pr create` adalah bagian dari GitHub CLI; keberadaan perintahnya diperiksa lewat `gh pr create --help`, tetapi tidak dijalankan.

Alur kerja harian

Tim kecil pun bisa memakai kebiasaan ini:

1. **Ambil** versi terbaru (`git pull`).
2. **Buat cabang** untuk satu pekerjaan (`git switch -c nama-fitur`).
3. **Ubah kode**, lalu jalankan `ruff check --fix .`, `ruff format .`, dan `python -m pytest` (Bab 18 dan 19). pre-commit bisa mengerjakannya otomatis saat commit.
4. **Commit** dengan pesan yang menjelaskan *apa dan mengapa*, bukan sekadar "perbaikan".
5. **Push** cabang, buka **pull request**, minta **review**.
6. Setelah disetujui dan semua pemeriksaan hijau, **merge** ke `main`.

Kebiasaan “satu cabang, satu tujuan, commit kecil” membuat konflik jarang dan riwayat mudah dibaca.

GitHub Actions: pemeriksaan otomatis di server

Pemeriksaan di komputer sendiri bisa terlewat. **GitHub Actions** menjalankan pemeriksaan otomatis di server GitHub setiap ada push atau pull request. Alurnya ditulis di berkas YAML di folder `.github/workflows/` (dokumentasi GitHub, dibaca 2026-10-10). Berikut contoh untuk proyek kita:

```yaml
name: Tes

on:
  push:
    branches: [main]
  pull_request:

jobs:
  tes:
    runs-on: ubuntu-latest
    steps:
      - name: Ambil kode
        uses: actions/checkout@v6
      - name: Pasang Python
        uses: actions/setup-python@v7
        with:
          python-version: "3.12"
      - name: Pasang paket dan alat
        run: python -m pip install -e ".[dev]"
      - name: Periksa gaya kode
        run: |
          ruff check .
          ruff format --check .
      - name: Jalankan tes
        run: python -m pytest -q
```

- `on:` menentukan **pemicu**: setiap push ke `main` dan setiap pull request.
- `jobs:` berisi pekerjaan; `runs-on: ubuntu-latest` memilih mesin GitHub; `steps:` adalah langkah berurutan. `uses:` memakai aksi siap pakai, `run:` menjalankan perintah.
- Nomor versi aksi (`@v6`, `@v7`) diambil dari dokumentasi pada 2026-10-10 dan bisa berubah; periksa dokumentasinya saat Anda memakainya.

Alur kerja ini **tidak dijalankan** oleh GitHub saat buku dibuat (tidak ada repo GitHub), jadi **belum terbukti berjalan**. Yang diperiksa penulis hanya **sintaks YAML**-nya, dan hanya itu yang bisa dibuktikan di sini:

```python
# Bab 20 - Memeriksa sintaks berkas alur kerja GitHub Actions (hanya sintaks YAML; tidak dijalankan GitHub)
from pathlib import Path

import yaml

berkas = Path(__file__).parent / "contoh" / ".github" / "workflows" / "tes.yml"
alur = yaml.safe_load(berkas.read_text(encoding="utf-8"))

print("Nama alur kerja:", alur["name"])
print("Pemicu (on)   :", sorted(alur[True]))      # YAML membaca kata 'on' sebagai True
for nama_job, job in alur["jobs"].items():
    print(f"Job '{nama_job}' berjalan di {job['runs-on']} dengan {len(job['steps'])} langkah:")
    for langkah in job["steps"]:
        print("  -", langkah["name"])
```

```
Nama alur kerja: Tes
Pemicu (on)    : ['pull_request', 'push']
Job 'tes' berjalan di ubuntu-latest dengan 5 langkah:
  - Ambil kode
  - Pasang Python
  - Pasang paket dan alat
  - Periksa gaya kode
  - Jalankan tes
```

Skrip memakai pustaka PyYAML (`python -m pip install pyyaml`). Hasilnya menunjukkan berkas terbaca dengan benar: satu job berisi lima langkah. Catatan teknis: YAML membaca kata `on` sebagai `True`, itu sebabnya kuncinya ditulis `alur[True]` dalam skrip.

Proyek Bersambung I1: semua bergabung

Inilah puncak seri. Paket `inventaris` (kelas dari Bab 16, dipasang di Bab 17, dites di Bab 18, dirapikan di Bab 19) kini hidup di repo git dengan alur kerja yang sehat: pemeriksaan, cabang, merge.

```
# Bab 20 - Proyek bersambung I1: kelas + tes + ruff + git dalam satu alur kerja
import re
import shutil
import sys

from bantu import FOLDER, git, jalan, rapikan, repo_baru, tulis

repo = repo_baru("proyek_i1")
shutil.copytree(FOLDER / "proyek_akhir", repo, dirs_exist_ok=True,
                ignore=shutil.ignore_patterns("__pycache__"))
tulis(repo, ".gitignore", ".venv/\n_tmp/\n__pycache__/\n.pytest_cache/\n.ruff_cache/
\n.env\n")


def pemeriksaan():
     """Tiga pemeriksaan yang juga dijalankan pre-commit: ruff check, ruff format,
pytest."""
    ruff = [sys.executable, "-m", "ruff"]
    a = jalan([*ruff, "check", "--no-cache", "."], repo)[0].splitlines()[-1]
   b = jalan([*ruff, "format", "--no-cache", "--check", "."], repo)[0].splitlines()[-1]
   c = jalan([sys.executable, "-m", "pytest", "-q", "-p", "no:cacheprovider"], repo)[0]
    print("  ruff check :", a)
    print("  ruff format:", b)
    print("  pytest     :", re.sub(r" in \d+\.\d+s", "", c.splitlines()[-1]))


print("1. Pemeriksaan di main")
pemeriksaan()
git("add", ".", repo=repo)
git("commit", "--quiet", "-m", "Paket inventaris: kelas, tes, dan konfigurasi",
repo=repo)

print("2. Cabang baru: tes untuk Sawah")
git("switch", "--quiet", "-c", "tes-sawah", repo=repo)
tulis(repo, "tes/test_sawah.py", '''import pytest

from inventaris import Sawah, Tanaman


def test_produktivitas_sawah():
    sawah = Sawah("Hilir", 0.5)
    sawah.tambah(Tanaman("padi", "1", 110, 1250.0))
```

```
    assert sawah.produktivitas() == pytest.approx(2500)
''')
pemeriksaan()
git("add", ".", repo=repo)
git("commit", "--quiet", "-m", "Tambah tes produktivitas sawah", repo=repo)

print("3. Gabungkan ke main")
git("switch", "--quiet", "main", repo=repo)
git("merge", "--quiet", "--no-ff", "-m", "Gabung cabang tes-sawah", "tes-sawah",
repo=repo)
pemeriksaan()

print("--- riwayat ---")
print(rapikan(git("log", "--graph", "--format=%s", repo=repo)[0]))
print("Berkas dilacak git:", len(git("ls-files", repo=repo)[0].splitlines()))
print("Status akhir (kosong = bersih):", repr(git("status", "--short", repo=repo)[0]))
```

```
1. Pemeriksaan di main
  ruff check : All checks passed!
  ruff format: 9 files already formatted
  pytest     : 14 passed
2. Cabang baru: tes untuk Sawah
  ruff check : All checks passed!
  ruff format: 10 files already formatted
  pytest     : 15 passed
3. Gabungkan ke main
  ruff check : All checks passed!
  ruff format: 10 files already formatted
  pytest     : 15 passed
--- riwayat ---
*   Gabung cabang tes-sawah
|\
| * Tambah tes produktivitas sawah
|/
* Paket inventaris: kelas, tes, dan konfigurasi
Berkas dilacak git: 12
Status akhir (kosong = bersih): ''
```

Telusuri:

- Langkah 1: di `main`, ruff lolos, 9 berkas rapi, dan **14 tes lulus**. Commit pertama.
- Langkah 2: di cabang `tes-sawah`, satu berkas tes baru ditambahkan. Semua pemeriksaan dijalankan **sebelum** commit: 10 berkas rapi dan **15 tes lulus**.
- Langkah 3: merge ke `main` dengan `--no-ff` (selalu membuat commit merge supaya jalur cabang terlihat di riwayat), lalu pemeriksaan ulang: tetap 15 tes lulus.
- Git melacak **12 berkas**: 10 berkas paket, satu berkas tes baru, dan `.gitignore`. Status akhir kosong: tidak ada yang tertinggal.

Latihan

1. Jalankan `01_init_commit.py`, lalu buka folder `_tmp/repo_dasar` di terminal dan ketik `git log` serta `git status`. Bandingkan dengan keluaran buku.
2. Ubah `03_cabang_merge.py` supaya cabang kedua menambah berkas `kandang.txt`. Apakah merge masih berjalan tanpa konflik? Mengapa?
3. Tambahkan `.idea/` dan `*.log` ke `.gitignore` di `05_gitignore_rahasia.py`, buat berkas `debug.log`, dan periksa dengan `git check-ignore -v debug.log`.
4. **Tantangan:** ubah `04_konflik.py` sehingga konflik diselesaikan dengan **menghitung rata-rata** kedua angka (530 g), bukan memilih salah satu. Mengapa keputusan seperti ini tidak bisa diambil git sendiri?

Ringkasan Bab

- **git** mencatat riwayat: `git add` memilih, `git commit` menyimpan, `git status` dan `git diff` memeriksa, `git restore` membuang perubahan yang belum di-commit.
- **Branch** = jalur sampingan; **merge** menggabungkannya. **Konflik** muncul bila baris yang sama diubah; manusia yang memutuskan lalu commit.
- **.gitignore** menjauhkan berkas sementara dan rahasia. Rahasia yang sudah ter-commit harus dianggap bocor.
- **GitHub** menyimpan repo di internet; **issue**, **pull request**, dan **review** mengatur kerja bersama. Bagian ini hanya dijelaskan, tidak dijalankan.
- **GitHub Actions** menjalankan pemeriksaan otomatis dari berkas YAML di `.github/workflows/`; contohnya di sini baru diperiksa sintaksnya.
- Alur harian: cabang, ubah, ruff, tes, commit, pull request, review, merge.

Penutup Seri I1

Selamat! Anda menyelesaikan **Seri I1** dan melewati gerbang pertama Tingkat Intermediate. Kemampuan Anda kini sudah bergeser dari "menulis program yang jalan" menjadi "membangun proyek yang terawat":

- Mengubah data lapangan menjadi **kelas**: `Unit` dan anak-anaknya, dengan pewarisan, polimorfisme, dan `dataclass` (Bab 16).
- Memberi tiap proyek **environment** sendiri dengan venv, pip, uv, dan `pyproject.toml` (Bab 17).
- Menjaga kode dengan **tes pytest** (Bab 18) dan **ruff**, pre-commit, serta logging (Bab 19).
- Menyimpan riwayat dan bekerja bersama dengan **git** dan alur GitHub (Bab 20).

Proyek inventaris Anda kini rapi dan aman diubah. Tetapi seluruh datanya masih tiruan yang kita ketik sendiri. Di **Seri I2**, program Anda mulai **mengambil dan mengolah data dari luar**: internet, berkas besar, dan database.

Selanjutnya, di Bab 21: bekerja dengan API, cara program meminta data dari layanan di internet dan membaca balasannya.

Bab 21 — Bekerja dengan API: Meminta Data dari Komputer Lain

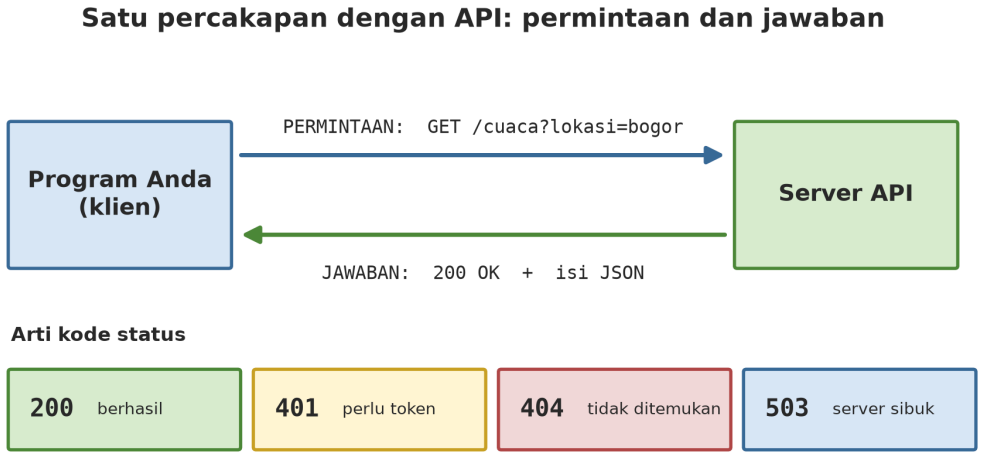
Seri I2 — Data dan Internet

Di bab ini, Anda akan:

- Memahami apa itu **API**, **HTTP**, dan **JSON** dengan analogi pelayan restoran
- Mengirim permintaan dengan `requests` dan `httpx`, lalu membaca jawabannya
- Memakai parameter dan **token** (kunci masuk) tanpa menulis rahasia di kode
- Menangani galat jaringan: kode status, **timeout**, dan coba ulang
- Mengambil data cuaca dan harga untuk Proyek Bersambung I2

Semua skrip bab ini ada di folder `kode/bab21/`. Di versi web, Anda juga bisa mengunduh semuanya sekaligus (zip).

Seri I1 membuat kode Anda rapi. Sekarang kode itu perlu **bahan**, dan banyak bahan ada di komputer lain: prakiraan cuaca di server badan cuaca, harga di server pasar. Bayangkan sebuah restoran. Anda (program) duduk di meja, **pelayan** (API) menerima pesanan, dan **dapur** (server) memasak. Anda tidak perlu masuk dapur; cukup memesan dengan cara yang sudah ditentukan, lalu menerima piring berisi jawabannya. **API** (*Application Programming Interface*) adalah "daftar menu dan aturan memesan" itu.



Gambar 0.1: Satu percakapan dengan API: Anda mengirim permintaan, server membalas dengan kode status dan isi JSON.

HTTP dan JSON dalam lima menit

Percakapan itu memakai bahasa bernama **HTTP**. Sebuah **permintaan** berisi: metode (`GET` = "tolong berikan"), alamat, dan kadang tambahan seperti parameter atau token. Sebuah **jawaban** berisi: **kode status** (angka tiga digit), *header* (keterangan), dan isi. Isi dari API biasanya berbentuk **JSON**, yaitu teks yang susunannya mirip dict dan list Python (Bab 9).

Kode	Artinya	Tindakan Anda
200	Berhasil	Baca isinya

Kode	Artinya	Tindakan Anda
400 / 404	Permintaan keliru / alamat tidak ada	Periksa alamat dan parameter; mengulang tidak membantu
401	Perlu token yang benar	Periksa token
500-an (mis. 503)	Server bermasalah atau sibuk	Boleh coba ulang nanti

Server tiruan: dapur milik kita sendiri

Agar contoh buku ini selalu menghasilkan keluaran yang sama dan tidak membebani server orang lain, kita membuat **server API tiruan** di komputer sendiri. File pembantunya `server_tiruan.py` (ada di folder bab, bukan skrip contoh) memakai `http.server` bawaan Python. Ia menyala di alamat lokal `127.0.0.1` pada port acak, melayani beberapa alamat dengan data **fiktif**, lalu dimatikan saat blok `with` selesai. Bagi `requests`, ia tampak seperti API sungguhan.

Permintaan pertama dengan requests

`requests` adalah pustaka pihak ketiga yang paling dikenal untuk HTTP (sudah ada di environment proyek ini; di komputer Anda: `pip install requests`).

```
# Bab 21 - Permintaan HTTP pertama dengan requests
import requests
from server_tiruan import ServerTiruan

with ServerTiruan() as server:          # menyalakan server tiruan di komputer sendiri
    jawaban = requests.get(f"{server.alamat}/cuaca?lokasi=bogor&hari=2", timeout=5)

    print("Kode status :", jawaban.status_code)
    print("Jenis isi   :", jawaban.headers["Content-Type"])
    print("Teks mentah :", jawaban.text)
# keluar dari blok `with`: server dimatikan lagi
```

```
Kode status : 200
Jenis isi   : application/json; charset=utf-8
Teks mentah : {"lokasi": "bogor", "satuan": {"suhu_maks": "C", "hujan_mm": "mm"},
"harian": {"tanggal": ["2026-01-01", "2026-01-02"], "suhu_maks": [30.1, 31.4],
"hujan_mm": [12.5, 0.0]}}
```

Kode `200` berarti berhasil. `Content-Type` memberi tahu bahwa isinya JSON. “Teks mentah” itu dikirim server sebagai satu baris panjang; berikutnya kita ubah menjadi data Python.

JSON dan parameter

```
# Bab 21 - JSON dan parameter (params)
import json
import requests
from server_tiruan import ServerTiruan

with ServerTiruan() as server:
   # params= menyusun bagian "?kunci=nilai" dengan benar (termasuk penyandian karakter)
      jawaban = requests.get(f"{server.alamat}/cuaca", params={"lokasi": "bandung",
"hari": 3}, timeout=5)
    print("Alamat lengkap:", jawaban.url.replace(server.alamat, "<server>"))

    data = jawaban.json()               # JSON -> dict/list Python
    print("Tipe data    :", type(data).__name__)
    print("Lokasi       :", data["lokasi"])
```

```
    print("Tanggal      :", data["harian"]["tanggal"])
    print("Hujan (mm)   :", data["harian"]["hujan_mm"])
    print("Total hujan  :", round(sum(data["harian"]["hujan_mm"]), 1), "mm")

    # arah sebaliknya: Python -> teks JSON
   print(json.dumps({"lokasi": "bandung", "hujan_total_mm": 19.2}, ensure_ascii=False,
indent=2))
```

```
Alamat lengkap: <server>/cuaca?lokasi=bandung&hari=3
Tipe data    : dict
Lokasi       : bandung
Tanggal      : ['2026-01-01', '2026-01-02', '2026-01-03']
Hujan (mm)   : [5.0, 0.0, 14.2]
Total hujan  : 19.2 mm
{
  "lokasi": "bandung",
  "hujan_total_mm": 19.2
}
```

- `params={...}` menyusun bagian `?lokasi=bandung&hari=3` dengan benar. Jangan merangkai alamat dengan tangan.
- `.json()` mengubah jawaban menjadi dict/list. Tiga hari hujan Bandung: 5,0 + 0,0 + 14,2 = **19,2 mm**.
- `json.dumps(...)` arah sebaliknya: dari Python menjadi teks JSON (`indent=2` merapikan).

httpx: saudara requests

`httpx` mirip `requests` tetapi punya **Client**: objek yang mengingat alamat dasar dan pengaturan sehingga tidak diulang di tiap panggilan (juga mendukung mode asinkron yang dibahas di seri lanjut).

```
# Bab 21 - httpx: saudara requests, dengan Client yang bisa dipakai ulang
import httpx
from server_tiruan import ServerTiruan

with ServerTiruan() as server:
   # Client menyimpan alamat dasar dan pengaturan, jadi tidak diulang di tiap panggilan
    with httpx.Client(base_url=server.alamat, timeout=5.0) as klien:
        for lokasi in ["bogor", "bandung", "malang"]:
            jawaban = klien.get("/cuaca", params={"lokasi": lokasi, "hari": 5})
            data = jawaban.json()
            hujan = data["harian"]["hujan_mm"]
         print(f"{lokasi:8} status {jawaban.status_code}  total hujan {sum(hujan):5.1f}
mm")
```

```
bogor    status 200  total hujan  49.2 mm
bandung  status 200  total hujan  51.5 mm
malang   status 200  total hujan  28.7 mm
```

Tiga lokasi, total hujan lima hari masing-masing 49,2; 51,5; dan 28,7 mm (data fiktif). Untuk pemula, pilih salah satu dulu; ilmunya hampir sama.

Autentikasi: menunjukkan “kartu anggota”

Banyak API hanya melayani pemegang **token** (semacam kartu anggota). Cara paling umum: mengirim token di *header* `Authorization: Bearer <token>`. Cara lain: **API key** di header atau parameter, atau *Basic auth* (nama pengguna dan kata sandi; di `requests` lewat `auth=(nama, sandi)`).

```
# Bab 21 - Autentikasi dasar: token di header Authorization
import os
import requests
```

```
from server_tiruan import ServerTiruan

with ServerTiruan() as server:
    alamat = f"{server.alamat}/rahasia"

    tanpa = requests.get(alamat, timeout=5)
    print("Tanpa token :", tanpa.status_code, tanpa.json())

    # Di dunia nyata, token dibaca dari variabel lingkungan, TIDAK ditulis di kode.
     # Di sini token acak milik server tiruan kita titipkan ke variabel lingkungan
sementara.
    os.environ["TOKEN_API_CONTOH"] = server.token
    token = os.environ["TOKEN_API_CONTOH"]

      dengan  =  requests.get(alamat,  headers={"Authorization":  f"Bearer  {token}"},
timeout=5)
    print("Dengan token:", dengan.status_code, dengan.json())
```

```
Tanpa token : 401 {'galat': 'token tidak ada atau salah'}
Dengan token: 200 {'pesan': 'Anda sudah masuk', 'petugas': 'fiktif'}
```

Tanpa token server menjawab `401`; dengan token menjawab `200`. Perhatikan cara token diperlakukan: dibaca dari **variabel lingkungan** (`os.environ`), tidak ditulis di kode. Di sini token acak dibuat server tiruan, jadi tidak ada rahasia sungguhan.

> ⚠ AWAS
>
> Jangan pernah menulis token, API key, atau kata sandi langsung di kode, apalagi bila kode itu akan diunggah ke GitHub (Bab 20). Simpan di variabel lingkungan atau berkas `.env` yang masuk `.gitignore`.

Ketika keadaan tidak mulus

Jaringan itu tidak pasti: alamat salah, server mati, koneksi putus. Program yang baik **mengantisipasinya**.

Kode status dan raise_for_status()

```
# Bab 21 - Kode status galat dan raise_for_status()
import requests
from server_tiruan import ServerTiruan

with ServerTiruan() as server:
     for alamat in ["/cuaca?lokasi=bogor&hari=1", "/cuaca?lokasi=atlantis", "/cuaca",
"/rahasia"]:
        jawaban = requests.get(server.alamat + alamat, timeout=5)
        print(f"{alamat:28} -> {jawaban.status_code}")

    # raise_for_status() mengubah kode 4xx/5xx menjadi pengecualian yang bisa ditangkap
    try:
                                requests.get(f"{server.alamat}/cuaca?lokasi=atlantis",
timeout=5).raise_for_status()
    except requests.HTTPError as galat:
                        print("Tertangkap   HTTPError:",   galat.response.status_code,
galat.response.json())
```

```
/cuaca?lokasi=bogor&hari=1   -> 200
/cuaca?lokasi=atlantis       -> 404
```

```
/cuaca                      -> 400
/rahasia                    -> 401
Tertangkap HTTPError: 404 {'galat': 'lokasi tidak dikenal'}
```

`requests` **tidak** menganggap `404` sebagai galat; itu hanya jawaban dengan kode berbeda. `raise_for_status()` mengubah kode 4xx/5xx menjadi pengecualian `HTTPError` yang bisa Anda `try`/`except` (Bab 12).

Timeout: jangan menunggu selamanya

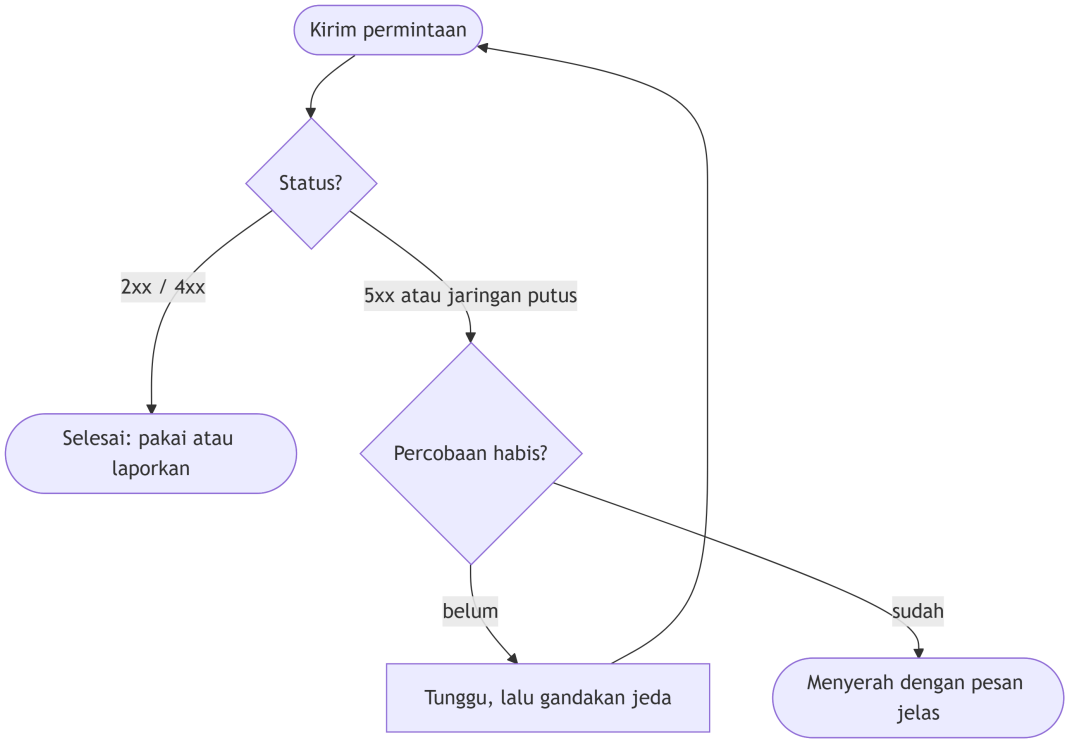
```python
# Bab 21 - Timeout: jangan menunggu selamanya
import requests
from server_tiruan import ServerTiruan

with ServerTiruan() as server:      # alamat /lambat sengaja membalas setelah 1,5 detik
    try:
        requests.get(f"{server.alamat}/lambat", timeout=0.3)
    except requests.Timeout:
        print("Server terlalu lama menjawab (> 0,3 detik): permintaan dibatalkan.")

    jawaban = requests.get(f"{server.alamat}/lambat", timeout=5)
    print("Dengan batas 5 detik:", jawaban.json()["pesan"])

# Tanpa timeout=, requests menunggu tanpa batas waktu (lihat dokumentasi resmi).
```

```
Server terlalu lama menjawab (> 0,3 detik): permintaan dibatalkan.
Dengan batas 5 detik: akhirnya selesai
```

> **i** INGAT
>
> Menurut dokumentasi resmi `requests`, bila `timeout=` tidak diberikan, permintaan **tidak punya batas waktu** dan program bisa menggantung. Biasakan selalu mengisinya. Di `httpx`, batas bawaannya 5 detik.

Coba ulang dengan jeda

Galat `5xx` sering sementara. Strateginya: coba lagi dengan jeda yang makin panjang (*exponential backoff*), tetapi hanya untuk galat yang memang layak diulang, dan berhenti setelah beberapa kali.



Alur coba ulang: hanya galat sementara (5xx, jaringan) yang diulang, dengan jeda berlipat.

```
# Bab 21 - Coba ulang dengan jeda yang makin panjang
import time
import requests
from server_tiruan import ServerTiruan


def ambil_dengan_ulang(alamat, maks_coba=4, jeda_awal=0.05):
    jeda = jeda_awal
    for percobaan in range(1, maks_coba + 1):
        try:
            jawaban = requests.get(alamat, timeout=5)
           if jawaban.status_code < 500:        # 2xx/4xx: tidak ada gunanya mengulang
                return jawaban
              print(f"  percobaan {percobaan}: status {jawaban.status_code}, tunggu
{jeda:.2f} detik")
        except (requests.ConnectionError, requests.Timeout):
         print(f"  percobaan {percobaan}: jaringan bermasalah, tunggu {jeda:.2f} detik")
        time.sleep(jeda)
         jeda = jeda * 2                             # jeda berlipat: 0,05 -> 0,10 -
> 0,20 ...
    raise RuntimeError("menyerah setelah beberapa kali mencoba")


with ServerTiruan() as server:      # /sibuk gagal dua kali (503), baru berhasil
    hasil = ambil_dengan_ulang(f"{server.alamat}/sibuk")
    print("Berhasil:", hasil.json())
```

```
  percobaan 1: status 503, tunggu 0.05 detik
  percobaan 2: status 503, tunggu 0.10 detik
Berhasil: {'pesan': 'berhasil', 'percobaan_ke': 3}
```

Server tiruan sengaja gagal dua kali (503), jadi percobaan ke-3 berhasil. Jeda berlipat dua tiap kali: 0,05 lalu 0,10 detik (di dunia nyata awali dengan 1 detik atau lebih agar tidak menghujani server).

Satu panggilan ke API nyata: Open-Meteo

Open-Meteo menyediakan prakiraan cuaca lewat API terbuka. Hasil pemeriksaan syaratnya pada **2026-10-10** (open-meteo.com/en/terms):

- Penggunaan gratis hanya untuk **non-komersial** dan **tanpa kunci**; batasnya kurang dari 10.000 panggilan per hari, 5.000 per jam, dan 600 per menit.
- Memakai API gratis berarti menerima lisensi **CC BY 4.0**, yaitu **wajib mencantumkan atribusi**. Itu sebabnya skrip mencetak "Data cuaca oleh Open-Meteo.com".
- Situs ini bukan aplikasi komersial; pembaca yang membuat produk berbayar harus membaca syarat itu sendiri.

```
# Bab 21 - Satu panggilan ke API nyata: Open-Meteo (tanpa kunci)
import requests

alamat = "https://api.open-meteo.com/v1/forecast"
parameter = {
    "latitude": -6.6,                  # sekitar Bogor
    "longitude": 106.8,
    "daily": "temperature_2m_max,precipitation_sum",
    "timezone": "Asia/Jakarta",
    "forecast_days": 3,
}
jawaban = requests.get(alamat, params=parameter, timeout=10)
jawaban.raise_for_status()
data = jawaban.json()

print("Status        :", jawaban.status_code)
print("Satuan        :", data["daily_units"])
for    tgl,    suhu,    hujan    in    zip(data["daily"]["time"],    data["daily"]
["temperature_2m_max"],
                          data["daily"]["precipitation_sum"]):
    print(f"{tgl}  suhu maks {suhu} C  hujan {hujan} mm")
print("Data cuaca oleh Open-Meteo.com (lisensi CC BY 4.0)")
```

```
Status        : 200
Satuan        : {'time': 'iso8601', 'temperature_2m_max': '°C', 'precipitation_sum':
'mm'}
2026-10-10  suhu maks 33.8 C  hujan 1.0 mm
2026-10-11  suhu maks 32.9 C  hujan 5.2 mm
2026-10-12  suhu maks 32.4 C  hujan 6.6 mm
Data cuaca oleh Open-Meteo.com (lisensi CC BY 4.0)
```

Kode ini **tidak dijalankan ulang** saat buku dibuat karena butuh internet dan hasilnya berubah setiap hari. Keluaran di atas **diambil sendiri pada 2026-10-10** lalu disimpan. Bila Anda menjalankannya, tanggal dan angkanya akan **berbeda**; itu wajar.

> TIPS
>
> Soal **harga komoditas**: pada 2026-10-10 saya tidak menemukan API harga komoditas pertanian yang terverifikasi gratis dan tanpa kunci. Daftar 71 sumber data API World Bank tidak memuat sumber harga komoditas, dan layanan harga komoditas yang muncul di pencarian umumnya meminta pendaftaran atau kunci (syaratnya tidak saya uji). Karena itu harga di buku ini berasal dari server tiruan (fiktif). Untuk proyek sungguhan, periksa sumber resmi seperti kementerian atau bursa komoditas, dan baca syarat penggunaannya.

Proyek Bersambung I2: mengambil data cuaca dan harga

Langkah 1 proyek Seri I2: mengambil cuaca tiga lokasi dan harga empat komoditas dari "API" kita, dengan fungsi kecil yang berhenti jelas bila ada galat, lalu menyimpannya sebagai JSON mentah di folder `_tmp`.

```
# Bab 21 - Proyek bersambung I2, langkah 1: mengambil data cuaca dan harga
import json
from pathlib import Path

import requests
from server_tiruan import ServerTiruan

tmp = Path(__file__).parent / "_tmp"
tmp.mkdir(exist_ok=True)


def ambil_json(alamat, **param):
    """Mengambil JSON; berhenti dengan pesan jelas bila ada galat."""
    jawaban = requests.get(alamat, params=param, timeout=5)
    jawaban.raise_for_status()
    return jawaban.json()


with ServerTiruan() as server:
    cuaca = {}
    for lokasi in ["bogor", "bandung", "malang"]:
        cuaca[lokasi] = ambil_json(f"{server.alamat}/cuaca", lokasi=lokasi)["harian"]
    harga = ambil_json(f"{server.alamat}/harga")["data"]

(tmp    /    "cuaca_mentah.json").write_text(json.dumps(cuaca,    ensure_ascii=False,
indent=1), encoding="utf-8")
(tmp    /    "harga_mentah.json").write_text(json.dumps(harga,    ensure_ascii=False,
indent=1), encoding="utf-8")

print("Lokasi cuaca  :", list(cuaca))
print("Hari per lokasi:", len(cuaca["bogor"]["tanggal"]))
print("Baris harga   :", len(harga), "| contoh:", harga[0])
print("Tersimpan di folder _tmp: cuaca_mentah.json, harga_mentah.json")
```

```
Lokasi cuaca  : ['bogor', 'bandung', 'malang']
Hari per lokasi: 5
Baris harga     :  20  |  contoh:  {'tanggal':  '2026-01-01',  'komoditas':  'padi',
'harga_per_kg': 6200}
Tersimpan di folder _tmp: cuaca_mentah.json, harga_mentah.json
```

Cuaca tiga lokasi, lima hari tiap lokasi, dan 20 baris harga (4 komoditas x 5 hari). Data mentah ini akan dibersihkan (Bab 22), digabung (Bab 23), disimpan (Bab 24), dan digambar (Bab 25). Agar tiap skrip berdiri sendiri, bab-bab berikutnya menyalin data yang sama langsung di kodenya.

Latihan

1. Ubah `02_json_dan_parameter.py` agar mengambil `malang` dan mencetak hari dengan hujan terbanyak.
2. Di `05_galat_status.py`, tambahkan alamat `/tidak-ada`. Kode status apa yang muncul?
3. Ubah `timeout` di `06_timeout.py` menjadi 2 detik. Apa yang berubah, dan mengapa?
4. **Tantangan:** tulis fungsi `ambil_aman(alamat)` yang memakai coba ulang untuk 5xx, tetapi **langsung berhenti** (tanpa mengulang) pada `404`, dan mengembalikan `None` bila akhirnya gagal.

Ringkasan Bab

- **API** = cara resmi meminta data dari program lain; **HTTP** = bahasanya; **JSON** = bentuk isinya.
- `requests.get(alamat, params=..., timeout=...)`; `.json()` mengubah jawaban menjadi data Python. `httpx.Client` sejenis dengan pengaturan bersama.
- Kode status: 2xx berhasil, 4xx salah di pihak kita, 5xx salah di server. `raise_for_status()` mengubah galat menjadi pengecualian.
- Selalu pakai **timeout**; coba ulang hanya galat sementara, dengan jeda berlipat.
- Token dibaca dari variabel lingkungan, **bukan** ditulis di kode.
- Open-Meteo: gratis non-komersial, tanpa kunci, wajib atribusi (CC BY 4.0). Harga komoditas gratis tanpa kunci: tidak ditemukan (belum terverifikasi).

Selanjutnya, di Bab 22: data yang datang tidak selalu rapi, misalnya catatan lapangan yang diketik manusia. Kita belajar **regex** untuk menggali dan membersihkan teks.

Bab 22 — Regex dan Pengolahan Teks: Menggali Data dari Tulisan Berantakan

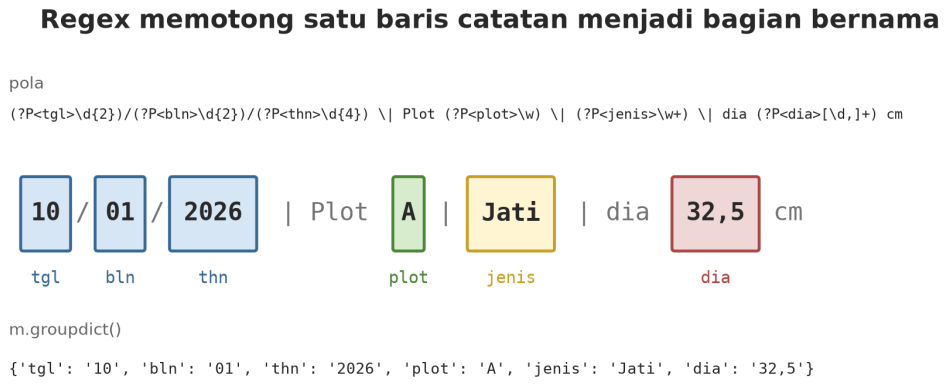
Seri I2 — Data dan Internet

Di bab ini, Anda akan:

- Memahami **regex** (*regular expression*): bahasa kecil untuk menggambarkan pola teks
- Mencari dan mengambil bagian teks dengan `re.search`, `re.findall`, dan **grup**
- Mengganti dan memisah teks dengan `re.sub` dan `re.split`
- Mengenali jebakan klasik: pengulang serakah dan lupa memeriksa `None`
- Membersihkan catatan lapangan tiruan untuk Proyek Bersambung I2

Semua skrip bab ini ada di folder `kode/bab22/`. Di versi web, Anda juga bisa mengunduh semuanya sekaligus (zip).

Data dari API (Bab 21) biasanya rapi. Data dari manusia tidak. Petugas lapangan mengetik "dia 32,5 cm", rekannya "diameter=32.5cm", yang lain "JATI" dengan huruf besar semua. Menyaring tulisan seperti itu dengan `if` dan `split` cepat berubah menjadi hutan belantara. **Regex** adalah ibarat **kartu cetakan pola**: Anda melubangi kartu sesuai bentuk yang dicari ("dua angka, garis miring, dua angka…"), lalu menempelkannya di atas teks; bagian yang pas dengan lubang itulah yang diambil.



Gambar 0.1: Satu pola dengan grup bernama memotong satu baris catatan menjadi bagian-bagian yang mudah dipakai.

Mencari: search, match, findall

Modul `re` sudah bawaan Python, tidak perlu dipasang.

```
# Bab 22 - Dasar modul re: search, match, fullmatch, findall
import re

teks = "Plot A: diameter 32,5 cm, tinggi 18 m"

print(re.search(r"diameter", teks))                # mencari di MANA SAJA
print(re.match(r"diameter", teks))                 # match: hanya dari AWAL teks
print(re.match(r"Plot", teks) is not None)
print(re.fullmatch(r"\d+", "2026"), re.fullmatch(r"\d+", "2026 "))
```

```
hasil = re.search(r"\d+,\d+", teks)                # angka desimal berkoma
print(hasil.group(), "ada di posisi", hasil.start(), "sampai", hasil.end())

print(re.findall(r"\d+", teks))                    # semua deretan angka
for cocok in re.finditer(r"[A-Z]", teks):        # setiap huruf besar beserta posisinya
    print(cocok.group(), cocok.start(), end=" | ")
print()
```

```
<re.Match object; span=(8, 16), match='diameter'>
None
True
<re.Match object; span=(0, 4), match='2026'> None
32,5 ada di posisi 17 sampai 21
['32', '5', '18']
P 0 | A 5 |
```

- `re.search` mencari **di mana saja**; hasilnya objek *Match* (ada posisi `span=(8, 16)`) atau `None` bila tidak ada.
- `re.match` hanya cocok bila pola ada di **awal** teks, makanya `diameter` menghasilkan `None`.
- `re.fullmatch` menuntut **seluruh** teks cocok: `"2026"` lolos, `"2026 "` (ada spasi) gagal.
- `findall` memberi daftar semua kecocokan: tiga deretan angka `['32', '5', '18']`. Perhatikan: `32,5` terpecah menjadi `32` dan `5`, karena pola `\d+` tidak tahu soal koma.
- Awalan `r"..."` (*raw string*) membuat garis miring terbalik dibaca apa adanya. Biasakan selalu memakainya untuk pola.

Bahan-bahan pola

Pola	Artinya
`\d`	satu angka; `\w` satu huruf/angka/garis bawah; `\s` satu spasi (atau tab)
`.`	satu karakter apa saja (kecuali baris baru)
`[A-C]`	satu karakter dari himpunan itu; `[^"]` = apa saja **kecuali** tanda kutip
`+` / `*` / `?`	satu atau lebih / nol atau lebih / nol atau satu
`{4}` / `{2,3}`	tepat 4 kali / 2 sampai 3 kali
`(...)`	grup; `(?:...)` grup tanpa nomor; `(?P<nama>...)` grup bernama
`|`	atau; `\b` batas kata

```
# Bab 22 - Pola umum: kelas karakter dan pengulang
import re

teks = "Plot A: 32,5 cm; Plot B: 45 cm; kode JT-0042; Plot C: 28,0 cm"

pola = {
    r"\d":           "satu angka",
    r"\d+":          "satu atau lebih angka",
    r"\d+,\d+":      "angka berkoma",
    r"\d+(?:,\d+)?": "angka, koma boleh ada boleh tidak",
    r"[A-C]":        "satu huruf antara A dan C",
    r"Plot \w":      "kata Plot, spasi, satu huruf/angka",
    r"[A-Z]{2}-\d{4}": "dua huruf besar, tanda minus, empat angka",
```

```
}
for p, arti in pola.items():
    print(f"{p:20} {arti:44} -> {re.findall(p, teks)}")
```

```
\d                   satu angka                                    -> ['3', '2', '5',
'4', '5', '0', '0', '4', '2', '2', '8', '0']
\d+                  satu atau lebih angka                        -> ['32', '5', '45',
'0042', '28', '0']
\d+,\d+              angka berkoma                                 -> ['32,5', '28,0']
\d+(?:,\d+)?         angka, koma boleh ada boleh tidak              -> ['32,5', '45',
'0042', '28,0']
[A-C]                satu huruf antara A dan C                     -> ['A', 'B', 'C']
Plot \w              kata Plot, spasi, satu huruf/angka            -> ['Plot A', 'Plot
B', 'Plot C']
[A-Z]{2}-\d{4}       dua huruf besar, tanda minus, empat angka     -> ['JT-0042']
```

Telusuri barisnya: `\d+(?:,\d+)?` menangkap `32,5` dan `45` sekaligus karena bagian koma dibuat *boleh ada, boleh tidak* (`?`). Pola terakhir menemukan kode `JT-0042`.

> TIPS
>
> Jangan menulis pola panjang sekaligus. Tulis satu bagian, uji dengan `findall` di teks contoh, tambah bagian berikutnya. Situs seperti regex101.com membantu melihat kecocokan secara visual.

Grup: mengambil bagian-bagian

Tanda kurung membuat **grup**, bagian pola yang hasilnya bisa diambil terpisah. Grup bernama (`?P<nama>`) jauh lebih mudah dibaca daripada nomor.

```
# Bab 22 - Grup: mengambil bagian-bagian yang cocok
import re

baris = "10/01/2026 | Plot A | Jati | dia 32,5 cm"

# kurung biasa = grup bernomor
m = re.search(r"(\d{2})/(\d{2})/(\d{4})", baris)
print(m.groups())                                    # ('10', '01', '2026')
print("Tahun:", m.group(3))

# (?P<nama>...) = grup bernama: lebih mudah dibaca
pola = r"(?P<tgl>\d{2})/(?P<bln>\d{2})/(?P<thn>\d{4}) \| Plot (?P<plot>\w) \| (?
P<jenis>\w+)"
m = re.match(pola, baris)
print(m.groupdict())
print(f"{m['thn']}-{m['bln']}-{m['tgl']}")           # menyusun tanggal ISO
```

```
('10', '01', '2026')
Tahun: 2026
{'tgl': '10', 'bln': '01', 'thn': '2026', 'plot': 'A', 'jenis': 'Jati'}
2026-01-10
```

`m.groups()` memberi tuple, `m.groupdict()` memberi dict, dan `m['thn']` mengambil satu grup. Tanggal `10/01/2026` kita susun ulang menjadi `2026-01-10` (format ISO, mudah diurutkan).

Mengganti dan memisah

```
# Bab 22 - Mengganti (sub) dan memisah (split)
import re

teks = "dia 32,5 cm; dia 45,2 cm; dia 28 cm"

print(re.sub(r"(\d),(\d)", r"\1.\2", teks))          # koma desimal -> titik; \1 \2 =
grup 1 dan 2
print(re.sub(r"\s+", " ", "Plot    A   Jati"))        # banyak spasi -> satu spasi
print(re.subn(r"cm", "CM", teks))                # subn juga memberi jumlah penggantian


def ke_meter(cocok):
    """Dipanggil untuk setiap kecocokan; mengembalikan teks penggantinya."""
    cm = float(cocok.group(1).replace(",", "."))
    return f"{cm / 100:.3f} m"


print(re.sub(r"(\d+(?:,\d+)?) cm", ke_meter, teks))
print(re.split(r"\s*[;,|]\s*", "A ; B,C |D"))         # pisah di salah satu pemisah
```

```
dia 32.5 cm; dia 45.2 cm; dia 28 cm
Plot A Jati
('dia 32,5 CM; dia 45,2 CM; dia 28 CM', 3)
dia 0.325 m; dia 0.452 m; dia 0.280 m
['A', 'B', 'C', 'D']
```

- `re.sub(pola, pengganti, teks)` mengganti semua kecocokan. `\1` dan `\2` di pengganti merujuk grup 1 dan 2, jadi `32,5` menjadi `32.5`.
- `re.subn` juga melaporkan jumlah penggantian: **3**.
- Pengganti boleh berupa **fungsi** yang dipanggil untuk tiap kecocokan: kita ubah sentimeter menjadi meter (`32,5 cm` menjadi `0.325 m`).
- `re.split` memotong di salah satu pemisah, termasuk bila ada spasi di sekitarnya.

compile, flags, dan VERBOSE

```
# Bab 22 - compile, flags (IGNORECASE) dan pola VERBOSE
import re

# Pola yang dipakai berulang: compile sekali, pakai berkali-kali
pola_jenis = re.compile(r"\b(jati|meranti|mahoni|sengon)\b", re.IGNORECASE)
print(pola_jenis.findall("JATI sehat, Meranti sakit, mahoni, pohon lain"))

# VERBOSE: spasi dan komentar di dalam pola diabaikan, jadi pola panjang bisa dijelaskan
pola_tanggal = re.compile(r"""
    (?P<tgl>\d{1,2})      # tanggal
    [/-]                  # pemisah: garis miring atau minus
    (?P<bln>\d{1,2})      # bulan
    [/-]
    (?P<thn>\d{4})        # tahun
""", re.VERBOSE)
for s in ["10/01/2026", "5-3-2026", "2026-01-10"]:
    m = pola_tanggal.fullmatch(s)
    print(f"{s:11} ->", m.groupdict() if m else "tidak cocok")
```

```
['JATI', 'Meranti', 'mahoni']
10/01/2026  -> {'tgl': '10', 'bln': '01', 'thn': '2026'}
```

```
5-3-2026    -> {'tgl': '5', 'bln': '3', 'thn': '2026'}
2026-01-10  -> tidak cocok
```

`re.compile` menyiapkan pola sekali untuk dipakai berulang. `re.IGNORECASE` mengabaikan besar-kecil huruf (makanya `JATI` ikut). `re.VERBOSE` membolehkan spasi dan komentar di dalam pola, sehingga pola panjang bisa dijelaskan baris demi baris. Perhatikan `2026-01-10` **tidak cocok** karena pola kita mengharapkan tahun di belakang.

Jebakan: serakah dan lupa None

```
# Bab 22 - Jebakan: pengulang yang serakah (greedy) dan yang malas (lazy)
import re

teks = 'catatan "sehat" lalu catatan "sakit" selesai'

print(re.findall(r'".*"', teks))        # serakah: menelan sebanyak mungkin
print(re.findall(r'".*?"', teks))       # malas (?): berhenti secepat mungkin
print(re.findall(r'"([^"]*)"', teks))   # cara lain: "apa pun kecuali tanda kutip"
```

```
['"sehat" lalu catatan "sakit"']
['"sehat"', '"sakit"']
['sehat', 'sakit']
```

Pola `".*"` **serakah** (*greedy*): ia menelan sebanyak mungkin, dari kutip pertama sampai kutip terakhir, jadi dua catatan menyatu. Tambahan `?` membuatnya **malas** (*lazy*): berhenti pada kutip terdekat. Cara ketiga, `[^"]*`, sering paling jelas.

> ⚠ AWAS
>
> `re.search` mengembalikan `None` bila tidak ada kecocokan. Memanggil `.group()` pada `None` memunculkan `AttributeError`. Selalu periksa dulu: `if hasil:`.

```
# Bab 22 - Dua galat klasik: pola rusak dan lupa memeriksa None
import re

try:
    re.compile(r"(\d+")                    # kurung buka tanpa pasangan
except re.error as galat:
    print("Pola rusak:", galat)

hasil = re.search(r"\d+", "tidak ada angka di sini")
print("Hasil pencarian:", hasil)            # None, bukan galat

try:
    print(hasil.group())
except AttributeError as galat:
    print("Galat:", galat)

# Cara aman: periksa dulu
if hasil:
    print(hasil.group())
else:
    print("Tidak ada angka; lanjutkan dengan aman.")
```

```
Pola rusak: missing ), unterminated subpattern at position 0
Hasil pencarian: None
Galat: 'NoneType' object has no attribute 'group'
Tidak ada angka; lanjutkan dengan aman.
```

Pola dengan kurung tanpa pasangan memunculkan `re.error` saat `compile`. Bunyi pesan galat bisa berbeda antarversi Python; yang penting Anda tahu galat itu ada di **pola**, bukan di teks.

Proyek Bersambung I2: membersihkan catatan lapangan

Langkah 2: lima catatan lapangan fiktif ditulis dengan gaya berbeda-beda, ditambah satu baris rusak. Satu pola VERBOSE menerima semua gaya, lalu kode merapikan: tanggal menjadi ISO, plot menjadi huruf besar, jenis dengan huruf awal besar, angka berkoma menjadi desimal Python, dan "sehat/sakit" menjadi `True`/`False`.

```
# Bab 22 - Proyek bersambung I2, langkah 2: membersihkan catatan lapangan dengan regex
import json
import re
from pathlib import Path

catatan = [
    "10/01/2026 | Plot A | Jati | dia 32,5 cm | tinggi 18 m | SEHAT",
    "10/01/2026 | plot a | meranti | diameter=45.2cm | tinggi: 25,5 m | sakit",
    "11-01-2026 | Plot B | Mahoni | dia 28 cm | tinggi 15 m | sehat",
    "11/01/2026 | Plot C | sengon | dia  19,8cm | tinggi 12 m | Sehat ",
    "12/01/2026 | Plot B | JATI | dia 36,1 cm | tinggi 20,5 m | sakit",
    "rusak: baris tanpa data yang jelas",
]

POLA = re.compile(r"""
    (?P<tgl>\d{1,2})[/-](?P<bln>\d{1,2})[/-](?P<thn>\d{4}) \s*\|\s*
    plot\s+(?P<plot>[a-z]) \s*\|\s*
    (?P<jenis>[a-z]+) \s*\|\s*
    (?:dia|diameter)\s*[=:]?\s*(?P<dia>\d+(?:[.,]\d+)?)\s*cm \s*\|\s*
    tinggi\s*[=:]?\s*(?P<tinggi>\d+(?:[.,]\d+)?)\s*m \s*\|\s*
    (?P<kondisi>sehat|sakit)
""", re.VERBOSE | re.IGNORECASE)


def angka(teks):
    return float(teks.replace(",", "."))


bersih, ditolak = [], []
for baris in catatan:
    m = POLA.fullmatch(baris.strip())
    if m is None:
        ditolak.append(baris)
        continue
    bersih.append({
        "tanggal": f"{m['thn']}-{int(m['bln']):02d}-{int(m['tgl']):02d}",
        "plot": m["plot"].upper(),
        "jenis": m["jenis"].capitalize(),
        "diameter_cm": angka(m["dia"]),
        "tinggi_m": angka(m["tinggi"]),
        "sehat": m["kondisi"].lower() == "sehat",
    })

for b in bersih:
    print(b)
print("Ditolak:", ditolak)

tmp = Path(__file__).parent / "_tmp"
tmp.mkdir(exist_ok=True)
(tmp   /   "catatan_bersih.json").write_text(json.dumps(bersih,   ensure_ascii=False,
```

```
indent=1), encoding="utf-8")
print(f"{len(bersih)} baris bersih, {len(ditolak)} ditolak")
```

```
{'tanggal': '2026-01-10', 'plot': 'A', 'jenis': 'Jati', 'diameter_cm': 32.5,
'tinggi_m': 18.0, 'sehat': True}
{'tanggal': '2026-01-10', 'plot': 'A', 'jenis': 'Meranti', 'diameter_cm': 45.2,
'tinggi_m': 25.5, 'sehat': False}
{'tanggal': '2026-01-11', 'plot': 'B', 'jenis': 'Mahoni', 'diameter_cm': 28.0,
'tinggi_m': 15.0, 'sehat': True}
{'tanggal': '2026-01-11', 'plot': 'C', 'jenis': 'Sengon', 'diameter_cm': 19.8,
'tinggi_m': 12.0, 'sehat': True}
{'tanggal': '2026-01-12', 'plot': 'B', 'jenis': 'Jati', 'diameter_cm': 36.1,
'tinggi_m': 20.5, 'sehat': False}
Ditolak: ['rusak: baris tanpa data yang jelas']
5 baris bersih, 1 ditolak
```

Dari enam baris, **5 bersih dan 1 ditolak**. Menolak dengan jujur lebih baik daripada menebak: baris yang ditolak dicatat agar manusia bisa memeriksanya. Hasil bersih disimpan di `_tmp/catatan_bersih.json`, siap digabung dengan data lain di bab berikutnya.

Latihan

1. Tulis pola yang menemukan semua kode seperti `JT-0042` di `"JT-0042, MR-0107, xx-12"`. (Hasilnya hanya dua.)
2. Dengan `re.sub`, ubah `"Plot  A,Plot B ,  Plot C"` menjadi `"Plot A, Plot B, Plot C"`.
3. Tambahkan satu baris catatan baru di `08_proyek_bersihkan.py` dengan gaya yang belum ada (mis. `tinggi=14 m` tanpa spasi). Apakah diterima? Ubah pola bila perlu.
4. **Tantangan:** perluas pola agar suhu seperti `suhu 27,5 C` di akhir baris menjadi grup opsional (boleh ada, boleh tidak).

Ringkasan Bab

- **Regex** = bahasa pola teks; modul `re` bawaan Python; tulis pola dengan `r"..."`.
- `search` (di mana saja), `match` (dari awal), `fullmatch` (seluruhnya), `findall` (semua).
- **Grup** `(...)` dan `(?P<nama>...)` mengambil bagian; `groupdict()` menghasilkan dict.
- `sub` mengganti (boleh dengan fungsi), `split` memisah, `compile` menyiapkan pola, `VERBOSE` membuat pola terbaca.
- Waspadai pengulang **serakah** dan selalu periksa hasil `None`.
- Data kotor dibersihkan dengan satu pola yang jelas; baris yang tidak cocok **ditolak dan dicatat**.

Selanjutnya, di Bab 23: data sudah bersih, tetapi puluhan ribu baris tak muat diolah dengan perulangan biasa. Kenalan dengan **numpy** dan **pandas**.

Bab 23 — Pengantar numpy dan pandas: Tabel Pintar untuk Data Lapangan

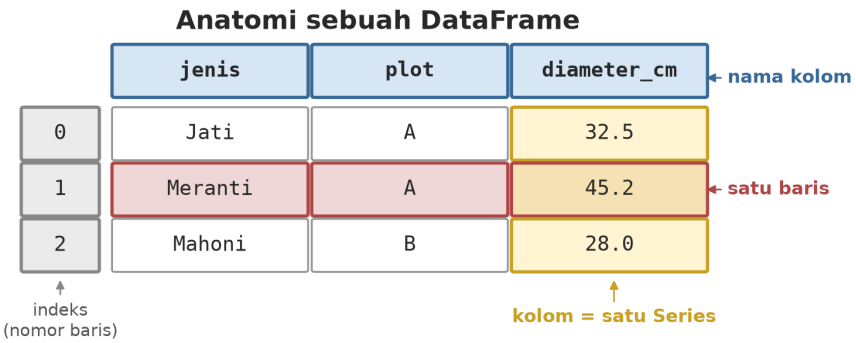
Seri I2 — Data dan Internet

Di bab ini, Anda akan:

- Memakai **array numpy**: satu rumus untuk ribuan angka sekaligus
- Membuat dan membaca **DataFrame** pandas, tabel dengan nama kolom
- Membaca dan menulis CSV, Excel, dan JSON
- Menyaring baris dan meringkas data dengan `groupby`
- Menggabungkan lima sektor, cuaca, dan harga: Proyek Bersambung I2

Semua skrip bab ini ada di folder `kode/bab23/`. Di versi web, Anda juga bisa mengunduh semuanya sekaligus (zip).

Bayangkan buku catatan lapangan dengan ribuan baris. Menghitung rata-rata diameter dengan `for` itu bisa, tetapi lambat dan panjang. **numpy** adalah ibarat **karung seragam**: semua isinya satu jenis (misalnya angka desimal), sehingga bisa ditimbang, dikali, dan dijumlahkan sekaligus. **pandas** adalah **lembar kerja berlabel** (mirip Excel) yang dibangun di atas numpy: setiap kolom punya nama, setiap baris punya nomor, dan Anda bisa menyaring serta meringkasnya dengan satu-dua baris kode. (Pemasangan: `pip install numpy pandas`; keduanya sudah ada di environment proyek ini.)



Gambar 0.1: Anatomi DataFrame: indeks di kiri, nama kolom di atas, satu kolom adalah satu Series, satu baris adalah satu catatan.

numpy: array

```
# Bab 23 - Array numpy: daftar angka yang cepat dan seragam
import numpy as np

diameter = np.array([32.5, 45.2, 28.0, 19.8, 36.1, 51.5])
print(diameter)
print("tipe:", diameter.dtype, "| bentuk:", diameter.shape, "| jumlah:", diameter.size)

print(np.arange(0, 10, 2))              # seperti range(), tetapi menghasilkan array
```

```
print(np.linspace(0, 1, 5))              # 5 angka rata antara 0 dan 1
print(np.zeros((2, 3)))                  # tabel 2 baris x 3 kolom berisi nol

tabel = np.array([[32.5, 18.0], [45.2, 25.5], [28.0, 15.0]])   # diameter, tinggi
print(tabel.shape, "->", tabel.shape[0], "baris,", tabel.shape[1], "kolom")
print(tabel[1, 0], "|", tabel[:, 1], "|", tabel[0])             # [baris, kolom]
```

```
[32.5 45.2 28.  19.8 36.1 51.5]
tipe: float64 | bentuk: (6,) | jumlah: 6
[0 2 4 6 8]
[0.   0.25 0.5  0.75 1.  ]
[[0. 0. 0.]
 [0. 0. 0.]]
(3, 2) -> 3 baris, 2 kolom
45.2 | [18.  25.5 15. ] | [32.5 18. ]
```

- `np.array([...])` membuat array. `dtype: float64` berarti semua isinya desimal 64-bit; `shape: (6,)` berarti satu dimensi, enam elemen.
- `np.arange` mirip `range` (Bab 8); `np.linspace(0, 1, 5)` membagi rata 0 sampai 1 menjadi 5 titik; `np.zeros((2, 3))` membuat tabel nol 2 baris x 3 kolom.
- Pada array dua dimensi, penulisan `[baris, kolom]`: `tabel[1, 0]` = 45,2 (baris ke-1, kolom ke-0), `tabel[:, 1]` = seluruh kolom ke-1 (tinggi).

Operasi sekaligus

```
# Bab 23 - Operasi sekaligus (vektor), irisan logika, dan sumbu
import numpy as np

diameter_cm = np.array([32.5, 45.2, 28.0, 19.8, 36.1, 51.5])

jari_m = diameter_cm / 2 / 100         # satu rumus untuk SEMUA elemen, tanpa perulangan
print("jari-jari (m):", jari_m)
luas_m2 = 3.14159 * jari_m ** 2
print("luas penampang (m2):", luas_m2.round(3))

besar = diameter_cm >= 30              # hasilnya array True/False
print("besar?   ", besar)
print("yang besar:", diameter_cm[besar])  # memilih lewat topeng True/False
print("rata-rata:", diameter_cm.mean().round(2), "| terbesar:", diameter_cm.max(), "|
simpangan baku:",
      diameter_cm.std().round(2))

tabel = np.array([[32.5, 18.0], [45.2, 25.5], [28.0, 15.0]])
print("rata-rata tiap kolom (axis=0):", tabel.mean(axis=0))
print("jumlah tiap baris  (axis=1):", tabel.sum(axis=1))
```

```
jari-jari (m): [0.1625 0.226  0.14   0.099  0.1805 0.2575]
luas penampang (m2): [0.083 0.16  0.062 0.031 0.102 0.208]
besar?    [ True  True False False  True  True]
yang besar: [32.5 45.2 36.1 51.5]
rata-rata: 35.52 | terbesar: 51.5 | simpangan baku: 10.51
rata-rata tiap kolom (axis=0): [35.23333333 19.5       ]
jumlah tiap baris  (axis=1): [50.5 70.7 43. ]
```

Tidak ada `for`: `diameter_cm / 2 / 100` dikerjakan pada keenam elemen sekaligus (**vektorisasi**). Perbandingan `diameter_cm >= 30` menghasilkan array `True/False`, dan array itu bisa dipakai sebagai **topeng** untuk memilih elemen (`diameter_cm[besar]`).

`axis` menentukan arah: `axis=0` meringkas **ke bawah** per kolom (rata-rata diameter 35,23 dan tinggi 19,5), `axis=1` meringkas **ke samping** per baris (50,5; 70,7; 43,0).

⚠ AWAS

Simpangan baku numpy (`std()`) bernilai **10,51**, tetapi `describe()` pandas di bagian DataFrame di bawah menunjukkan **11,51** untuk data yang sama. Bukan salah hitung: numpy membagi dengan *n* (populasi) secara bawaan, pandas membagi dengan *n-1* (sampel). Seri I3 menjelaskan alasannya; untuk sekarang, ingat bahwa dua pustaka bisa punya bawaan berbeda.

pandas: DataFrame

```
# Bab 23 - DataFrame: tabel dengan nama kolom
import pandas as pd

pohon = pd.DataFrame({
    "jenis": ["Jati", "Meranti", "Mahoni", "Sengon", "Jati", "Meranti"],
    "plot": ["A", "A", "B", "C", "B", "C"],
    "diameter_cm": [32.5, 45.2, 28.0, 19.8, 36.1, 51.5],
    "tinggi_m": [18.0, 25.5, 15.0, 12.0, 20.5, 28.0],
    "sehat": [True, False, True, True, False, True],
})
print(pohon)
print()
print("bentuk (baris, kolom):", pohon.shape)
print(pohon.dtypes)
print()
print(pohon["diameter_cm"].mean())        # satu kolom = Series
print(pohon.describe().round(2))          # ringkasan statistik kolom angka
```

```
     jenis plot  diameter_cm  tinggi_m  sehat
0     Jati    A         32.5      18.0   True
1  Meranti    A         45.2      25.5  False
2   Mahoni    B         28.0      15.0   True
3   Sengon    C         19.8      12.0   True
4     Jati    B         36.1      20.5  False
5  Meranti    C         51.5      28.0   True

bentuk (baris, kolom): (6, 5)
jenis              str
plot               str
diameter_cm    float64
tinggi_m       float64
sehat             bool
dtype: object

35.516666666666666
       diameter_cm  tinggi_m
count         6.00      6.00
mean         35.52     19.83
std          11.51      6.12
min          19.80     12.00
25%          29.12     15.75
50%          34.30     19.25
75%          42.92     24.25
max          51.50     28.00
```

Kolom-kolom di DataFrame boleh berbeda jenis: teks (`str`), desimal (`float64`), benar/salah (`bool`). Satu kolom, misalnya `pohon["diameter_cm"]`, disebut **Series**. `describe()` merangkum kolom angka: banyaknya, rata-rata (35,52), terkecil (19,8), kuartil, terbesar (51,5).

Membaca dan menulis file

```
# Bab 23 - Menulis dan membaca CSV, Excel, dan JSON
from pathlib import Path
import pandas as pd

tmp = Path(__file__).parent / "_tmp"
tmp.mkdir(exist_ok=True)

pohon = pd.DataFrame({
    "jenis": ["Jati", "Meranti", "Mahoni"],
    "diameter_cm": [32.5, 45.2, 28.0],
    "sehat": [True, False, True],
})

pohon.to_csv(tmp / "pohon.csv", index=False)            # index=False: jangan tulis
nomor baris
print((tmp / "pohon.csv").read_text(encoding="utf-8"))

dari_csv = pd.read_csv(tmp / "pohon.csv")
print(dari_csv.equals(pohon), "<- CSV kembali sama persis")

pohon.to_excel(tmp / "pohon.xlsx", index=False)       # memakai pustaka openpyxl
dari_excel = pd.read_excel(tmp / "pohon.xlsx")
print(dari_excel.equals(pohon), "<- Excel kembali sama persis")

pohon.to_json(tmp / "pohon.json", orient="records", force_ascii=False)
print((tmp / "pohon.json").read_text(encoding="utf-8"))
```

```
jenis,diameter_cm,sehat
Jati,32.5,True
Meranti,45.2,False
Mahoni,28.0,True

True <- CSV kembali sama persis
True <- Excel kembali sama persis
[{"jenis":"Jati","diameter_cm":32.5,"sehat":true},
{"jenis":"Meranti","diameter_cm":45.2,"sehat":false},
{"jenis":"Mahoni","diameter_cm":28.0,"sehat":true}]
```

`to_csv`/`read_csv`, `to_excel`/`read_excel` (butuh `openpyxl`), dan `to_json`/`read_json` membuat tukar-menukar data dengan program lain semudah satu baris. `index=False` mencegah nomor baris ikut tertulis sebagai kolom. Hasil `equals(...)` bernilai `True`: data yang dibaca kembali sama persis dengan aslinya.

Memilih dan menyaring

```
# Bab 23 - Memilih kolom/baris dan menyaring (filter)
import pandas as pd

pohon = pd.DataFrame({
    "jenis": ["Jati", "Meranti", "Mahoni", "Sengon", "Jati", "Meranti"],
    "plot": ["A", "A", "B", "C", "B", "C"],
    "diameter_cm": [32.5, 45.2, 28.0, 19.8, 36.1, 51.5],
    "sehat": [True, False, True, True, False, True],
})

print(pohon[["jenis", "diameter_cm"]].head(2))            # beberapa kolom
print(pohon.iloc[0])                                      # baris ke-0 menurut urutan
print(pohon.loc[2, "jenis"])                                     # baris berlabel 2,
```

```
kolom "jenis"

besar = pohon[pohon["diameter_cm"] >= 30]                 # saring dengan syarat
print(besar)

sakit_besar = pohon[(pohon["diameter_cm"] >= 30) & (~pohon["sehat"])]   # & = DAN, ~
= BUKAN
print(sakit_besar)
print(pohon.query("plot == 'A' or jenis == 'Jati'"))       # cara lain, mirip kalimat
print(pohon.sort_values("diameter_cm", ascending=False).head(3))
```

```
     jenis  diameter_cm
0     Jati         32.5
1  Meranti         45.2
jenis          Jati
plot              A
diameter_cm    32.5
sehat          True
Name: 0, dtype: object
Mahoni
     jenis plot  diameter_cm  sehat
0     Jati    A         32.5   True
1  Meranti    A         45.2  False
4     Jati    B         36.1  False
5  Meranti    C         51.5   True
     jenis plot  diameter_cm  sehat
1  Meranti    A         45.2  False
4     Jati    B         36.1  False
     jenis plot  diameter_cm  sehat
0     Jati    A         32.5   True
1  Meranti    A         45.2  False
4     Jati    B         36.1  False
     jenis plot  diameter_cm  sehat
5  Meranti    C         51.5   True
1  Meranti    A         45.2  False
4     Jati    B         36.1  False
```

- `df[["a", "b"]]` memilih kolom; `iloc` memilih menurut **urutan**, `loc` menurut **label**.
- `df[df["diameter_cm"] >= 30]` menyaring baris dengan syarat. Untuk menggabungkan syarat pakai `&` (dan), `|` (atau), `~` (bukan), **masing-masing syarat diberi kurung**.
- `query("plot == 'A' or jenis == 'Jati'")` adalah cara yang lebih mirip kalimat.
- `sort_values(..., ascending=False)` mengurutkan dari terbesar.

Agregasi: meringkas per kelompok

```
# Bab 23 - Agregasi: groupby, value_counts, pivot_table
import pandas as pd

pohon = pd.DataFrame({
    "jenis": ["Jati", "Meranti", "Mahoni", "Sengon", "Jati", "Meranti"],
    "plot": ["A", "A", "B", "C", "B", "C"],
    "diameter_cm": [32.5, 45.2, 28.0, 19.8, 36.1, 51.5],
    "tinggi_m": [18.0, 25.5, 15.0, 12.0, 20.5, 28.0],
})

print(pohon["jenis"].value_counts())                      # berapa pohon per jenis
print(pohon.groupby("jenis")["diameter_cm"].mean().round(2))
```

```
rekap = pohon.groupby("plot").agg(
    jumlah=("jenis", "count"),
    diameter_rata=("diameter_cm", "mean"),
    tinggi_maks=("tinggi_m", "max"),
)
print(rekap.round(2))

print(pohon.pivot_table(index="plot",      columns="jenis",      values="diameter_cm",
aggfunc="mean"))
```

```
jenis
Jati       2
Meranti    2
Mahoni     1
Sengon     1
Name: count, dtype: int64
jenis
Jati       34.30
Mahoni     28.00
Meranti    48.35
Sengon     19.80
Name: diameter_cm, dtype: float64
      jumlah  diameter_rata  tinggi_maks
plot
A          2          38.85         25.5
B          2          32.05         20.5
C          2          35.65         28.0
jenis  Jati  Mahoni  Meranti  Sengon
plot
A      32.5     NaN     45.2     NaN
B      36.1    28.0      NaN     NaN
C       NaN     NaN     51.5    19.8
```

`groupby` bekerja seperti memilah karung per jenis lalu menimbang tiap tumpukan: pola **pecah, hitung, gabung**. Rata-rata diameter Jati 34,30 (dari 32,5 dan 36,1), Meranti 48,35. `agg` memungkinkan beberapa ringkasan sekaligus dengan nama kolom hasil sendiri. `pivot_table` membuat tabel silang plot x jenis; sel yang tidak punya data diisi `NaN` (“bukan angka” = kosong).

Menggabung tabel dan data kosong

```
# Bab 23 - Menggabung tabel (merge, concat) dan data kosong (NaN)
import pandas as pd

panen  =  pd.DataFrame({"komoditas":  ["kopi",  "padi",  "lele"],  "hasil_kg":  [245.5,
3100.0, 432.4]})
harga = pd.DataFrame({"komoditas": ["kopi", "padi", "jagung"], "harga_per_kg": [58000,
6200, 5100]})

# how="left": semua baris panen dipertahankan; yang tak punya pasangan harga diisi NaN
gabung = panen.merge(harga, on="komoditas", how="left")
print(gabung)
print("Baris kosong harga:", gabung["harga_per_kg"].isna().sum())

gabung["pendapatan"] = gabung["hasil_kg"] * gabung["harga_per_kg"]
print(gabung)
print(gabung.fillna({"harga_per_kg": 0, "pendapatan": 0}))   # isi kosong dengan 0
print(gabung.dropna())                                       # buang baris yang
ada kosongnya
```

```
tambahan = pd.DataFrame({"komoditas": ["sapi"], "hasil_kg": [1151.5]})
print(pd.concat([panen, tambahan], ignore_index=True))       # menumpuk baris ke bawah
```

```
  komoditas  hasil_kg  harga_per_kg
0      kopi     245.5       58000.0
1      padi    3100.0        6200.0
2      lele     432.4           NaN
Baris kosong harga: 1
  komoditas  hasil_kg  harga_per_kg  pendapatan
0      kopi     245.5       58000.0  14239000.0
1      padi    3100.0        6200.0  19220000.0
2      lele     432.4           NaN         NaN
  komoditas  hasil_kg  harga_per_kg  pendapatan
0      kopi     245.5       58000.0  14239000.0
1      padi    3100.0        6200.0  19220000.0
2      lele     432.4           0.0         0.0
  komoditas  hasil_kg  harga_per_kg  pendapatan
0      kopi     245.5       58000.0  14239000.0
1      padi    3100.0        6200.0  19220000.0
  komoditas  hasil_kg
0      kopi     245.5
1      padi    3100.0
2      lele     432.4
3      sapi    1151.5
```

`merge` menyambung dua tabel lewat kolom bersama, seperti `JOIN` di basis data (Bab 24). Dengan `how="left"`, semua baris `panen` dipertahankan; lele tidak punya harga sehingga `NaN`, dan `pendapatan`-nya ikut kosong. Ada tiga cara menangani kosong: dibiarkan, `fillna` (isi nilai pengganti), atau `dropna` (buang barisnya). **Pilih sesuai makna data**: mengisi harga lele dengan 0 berarti "gratis", yang belum tentu benar.

> TIPS
>
> Setelah `merge`, selalu cek `isna().sum()`. Baris tak berpasangan hampir selalu tanda ada nama yang tidak cocok (misalnya `kopi` lawan `Kopi`) atau memang data yang belum ada.

Proyek Bersambung I2: menggabungkan lima sektor, cuaca, dan harga

Langkah 3: sepuluh catatan panen dari lima sektor (hutan, kebun, pertanian, perikanan, peternakan), cuaca Bogor, dan harga digabung menjadi satu tabel. (Data fiktif; disalin di skrip agar berdiri sendiri, bentuknya sama dengan hasil Bab 21.)

```
# Bab 23 - Proyek bersambung I2, langkah 3: menggabungkan lima sektor, cuaca, dan harga
import pandas as pd

# Data lima sektor (fiktif). Disalin di sini supaya skrip berdiri sendiri.
panen = pd.DataFrame(
    [("2026-01-01", "Hutan", "Plot A", "jati", 1200.0),
     ("2026-01-01", "Kebun", "Blok A", "kopi", 245.5),
     ("2026-01-02", "Kebun", "Blok B", "kakao", 198.5),
     ("2026-01-02", "Pertanian", "Sawah 1", "padi", 3100.0),
     ("2026-01-03", "Pertanian", "Sawah 2", "jagung", 1850.0),
     ("2026-01-03", "Perikanan", "Kolam A", "lele", 432.4),
     ("2026-01-04", "Peternakan", "Kandang 1", "sapi", 1151.5),
     ("2026-01-04", "Kebun", "Blok C", "kopi", 310.0),
     ("2026-01-05", "Pertanian", "Sawah 1", "padi", 2950.0),
     ("2026-01-05", "Kebun", "Blok D", "kakao", 276.0)],
```

```
    columns=["tanggal", "sektor", "unit", "komoditas", "hasil_kg"],
)
# Cuaca Bogor dan harga: bentuknya sama dengan data dari server tiruan di Bab 21 (fiktif)
cuaca = pd.DataFrame({
    "tanggal": ["2026-01-01", "2026-01-02", "2026-01-03", "2026-01-04", "2026-01-05"],
    "suhu_maks": [30.1, 31.4, 29.8, 28.9, 30.6],
    "hujan_mm": [12.5, 0.0, 8.2, 25.4, 3.1],
})
harga = pd.DataFrame(
    [(t, k, h) for k, daftar in {
        "padi": [6200, 6250, 6180, 6300, 6270],
        "jagung": [5100, 5150, 5200, 5120, 5180],
        "kopi": [58000, 58500, 59000, 57500, 58200],
        "kakao": [62000, 61500, 62500, 63000, 62800]}.items()
     for t, h in zip(cuaca["tanggal"], daftar)],
    columns=["tanggal", "komoditas", "harga_per_kg"],
)

gabung = (panen
          .merge(cuaca, on="tanggal", how="left")
          .merge(harga, on=["tanggal", "komoditas"], how="left"))
gabung["pendapatan_rp"] = gabung["hasil_kg"] * gabung["harga_per_kg"]

print(gabung.drop(columns=["tanggal"]).to_string(index=False))
print()
print("Baris tanpa harga (hutan, ikan, ternak):",
int(gabung["harga_per_kg"].isna().sum()))
per_sektor = gabung.groupby("sektor").agg(hasil_kg=("hasil_kg", "sum"),
pendapatan_rp=("pendapatan_rp", "sum"))
print(per_sektor.round(0))
```

```
   sektor     unit komoditas  hasil_kg  suhu_maks  hujan_mm  harga_per_kg  pendapatan_rp
    Hutan   Plot A      jati    1200.0       30.1      12.5           NaN            NaN
    Kebun   Blok A     kopi     245.5       30.1      12.5       58000.0     14239000.0
    Kebun   Blok B    kakao     198.5       31.4       0.0       61500.0     12207750.0
 Pertanian  Sawah 1     padi    3100.0       31.4       0.0        6250.0     19375000.0
 Pertanian  Sawah 2   jagung    1850.0       29.8       8.2        5200.0      9620000.0
 Perikanan  Kolam A     lele     432.4       29.8       8.2           NaN            NaN
Peternakan Kandang 1     sapi    1151.5       28.9      25.4           NaN            NaN
    Kebun   Blok C     kopi     310.0       28.9      25.4       57500.0     17825000.0
 Pertanian  Sawah 1     padi    2950.0       30.6       3.1        6270.0     18496500.0
    Kebun   Blok D    kakao     276.0       30.6       3.1       62800.0     17332800.0

Baris tanpa harga (hutan, ikan, ternak): 3
           hasil_kg  pendapatan_rp
sektor
Hutan         1200.0            0.0
Kebun         1030.0     61604550.0
Perikanan      432.0            0.0
Pertanian     7900.0     47491500.0
Peternakan    1152.0            0.0
```

Telusuri hasilnya:

- Tiap catatan panen menerima suhu dan hujan **hari yang sama** (`merge` pada `tanggal`) dan harga **komoditas dan hari yang sama** (`merge` pada dua kolom).
- **3 baris tanpa harga**: kayu jati, lele, sapi, karena API harga hanya memuat padi, jagung, kopi, kakao. `pendapatan_rp` untuk baris itu `NaN`.
- Ringkasan per sektor: Kebun 1030,0 kg dengan pendapatan Rp 61.604.550; Pertanian 7900,0 kg dengan Rp 47.491.500. Sektor Hutan, Perikanan, dan Peternakan tercatat **0** pendapatan hanya karena `sum`

mengabaikan `NaN`, **bukan** karena benar-benar tidak ada nilainya. Keterbatasan data seperti ini harus dikatakan terus terang saat melaporkan.

Latihan

1. Dari array `np.array([480, 500, 510, 520, 530])`, hitung bobot rata-rata dan pilih yang berbobot di atas 505 dengan topeng.
2. Buat DataFrame ikan dengan kolom `kolam`, `jenis`, `jumlah`, `bobot_rata_g`, lalu hitung biomassa tiap baris (`jumlah * bobot_rata_g / 1000`) sebagai kolom baru.
3. Di `06_agregasi.py`, tambahkan agregasi jumlah pohon dan diameter terkecil per jenis.
4. **Tantangan:** di `08_proyek_gabung.py`, hitung pendapatan rata-rata per sektor **hanya dari baris yang punya harga**, dan hitung berapa persen baris yang tidak punya harga.

Ringkasan Bab

- **numpy array**: isi satu jenis, operasi **sekaligus** tanpa `for`; `axis=0` per kolom, `axis=1` per baris.
- **DataFrame** = tabel berlabel; satu kolom = **Series**. `head`, `describe`, `dtypes`, `shape` untuk mengenalinya.
- Baca/tulis: `read_csv`, `read_excel`, `to_json`, dan kawan-kawan; `index=False` untuk menghindari kolom nomor.
- Saring dengan syarat (`&`, `|`, `~`, masing-masing berkurung) atau `query`; urutkan dengan `sort_values`.
- `groupby` + `agg` meringkas per kelompok; `merge` menyambung tabel; waspadai `NaN` setelah penggabungan.
- Pustaka bisa berbeda bawaan (simpangan baku numpy vs pandas).

Selanjutnya, di Bab 24: data gabungan itu hilang begitu program ditutup. Kita simpan di **basis data**: SQLite, SQLAlchemy, dan PostgreSQL.

Bab 24 — Database: Lemari Arsip yang Tidak Pernah Lupa

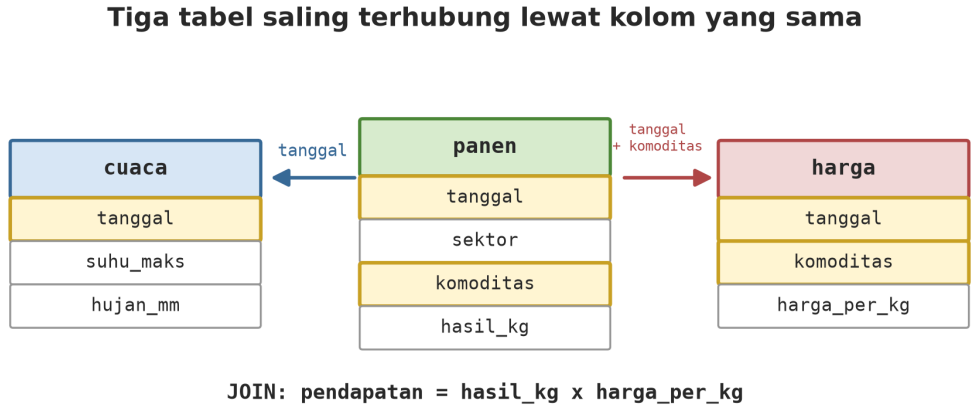
Seri I2 — Data dan Internet

Di bab ini, Anda akan:

- Memahami apa itu **basis data (database)**, tabel, baris, dan **SQL**
- Memakai **SQLite** lewat modul `sqlite3` bawaan Python
- Menulis `INSERT`, `SELECT`, `UPDATE`, `DELETE`, `JOIN`, dan `GROUP BY`, serta menghindari **SQL injection**
- Mengenal **SQLAlchemy** (Core dan ORM) dan cara menyambung ke **PostgreSQL**
- Menyimpan data gabungan Proyek I2 ke basis data

Semua skrip bab ini ada di folder `kode/bab24/`. Di versi web, Anda juga bisa mengunduh semuanya sekaligus (zip).

File CSV (Bab 23) itu seperti tumpukan kertas catatan: mudah dibuat, tetapi sulit dicari, dan bisa kacau bila dua orang menulis bersamaan. **Basis data** adalah **lemari arsip berlaci**: tiap laci (**tabel**) punya kolom yang tetap, tiap kertas (**baris**) mengikuti aturan, dan petugas arsip (**SQL**, *Structured Query Language*) menjawab pertanyaan seperti "berapa total panen kebun?" tanpa Anda membongkar semua kertas. Ia juga menjaga agar penyimpanan **aman**: perubahan yang setengah jalan bisa dibatalkan.



Gambar 0.1: Tiga tabel (panen, cuaca, harga) terhubung lewat kolom yang sama. Itulah inti basis data relasional.

SQLite: basis data dalam satu berkas

SQLite adalah mesin basis data yang hidup sebagai **satu berkas** dan sudah dibawa Python (modul `sqlite3`), jadi tanpa pemasangan dan tanpa server. Cocok untuk belajar, aplikasi kecil, dan data yang bisa dibawa-bawa.

```
# Bab 24 - SQLite dengan modul sqlite3 (bawaan Python)
import sqlite3
from pathlib import Path

tmp = Path(__file__).parent / "_tmp"
tmp.mkdir(exist_ok=True)
```

```
berkas = tmp / "lapangan.db"
berkas.unlink(missing_ok=True)                  # mulai dari basis data kosong agar
hasilnya sama

koneksi = sqlite3.connect(berkas)              # berkas dibuat bila belum ada
koneksi.execute("""
    CREATE TABLE pohon (
        id INTEGER PRIMARY KEY,
        jenis TEXT NOT NULL,
        plot TEXT NOT NULL,
        diameter_cm REAL
    )
""")
data = [("Jati", "A", 32.5), ("Meranti", "A", 45.2), ("Mahoni", "B", 28.0), ("Jati",
"B", 36.1)]
koneksi.executemany("INSERT INTO pohon (jenis, plot, diameter_cm) VALUES (?, ?, ?)",
data)
koneksi.commit()                                        # tanpa commit, perubahan tidak
disimpan permanen

for baris in koneksi.execute("SELECT id, jenis, plot, diameter_cm FROM pohon ORDER BY
diameter_cm DESC"):
    print(baris)

jumlah = koneksi.execute("SELECT COUNT(*) FROM pohon").fetchone()[0]
print("Jumlah pohon:", jumlah)
koneksi.close()
```

```
(2, 'Meranti', 'A', 45.2)
(4, 'Jati', 'B', 36.1)
(1, 'Jati', 'A', 32.5)
(3, 'Mahoni', 'B', 28.0)
Jumlah pohon: 4
```

- `CREATE TABLE` mendefinisikan kolom dan jenisnya; `PRIMARY KEY` = nomor unik tiap baris; `NOT NULL` = wajib diisi.
- `?` adalah **tempat nilai** (placeholder); nilainya diberikan terpisah. `executemany` memasukkan banyak baris sekaligus.
- **`commit()`** menyimpan perubahan secara permanen. Tanpa commit, perubahan bisa hilang.
- `ORDER BY diameter_cm DESC` mengurutkan dari besar ke kecil: Meranti 45,2 paling atas.

Jangan menempel teks pengguna ke SQL

```
# Bab 24 - Parameter (?) melawan SQL injection
import sqlite3

koneksi = sqlite3.connect(":memory:")          # basis data sementara di memori
koneksi.execute("CREATE TABLE petugas (nama TEXT, wilayah TEXT)")
koneksi.executemany("INSERT INTO petugas VALUES (?, ?)", [("Sari", "Plot A"), ("Budi",
"Plot B")])

masukan = "x' OR '1'='1"                        # ketikan jahil seorang pengguna

# CARA SALAH: menempelkan teks pengguna langsung ke SQL (f-string)
sql_salah = f"SELECT nama FROM petugas WHERE nama = '{masukan}'"
print("SQL yang terbentuk:", sql_salah)
print("Hasil cara salah  :", koneksi.execute(sql_salah).fetchall())     # bocor: semua
baris keluar!
```

```
# CARA BENAR: tanda tanya + nilai terpisah; teks diperlakukan sebagai DATA, bukan perintah
print("Hasil cara benar  :", koneksi.execute("SELECT nama FROM petugas WHERE nama = ?", (masukan,)).fetchall())
```

```
SQL yang terbentuk: SELECT nama FROM petugas WHERE nama = 'x' OR '1'='1'
Hasil cara salah  : [('Sari',), ('Budi',)]
Hasil cara benar  : []
```

> ⚠ AWAS
>
> Pada cara salah, ketikan jahil `x' OR '1'='1` ikut menjadi **bagian perintah SQL** dan membuat syarat selalu benar, sehingga **semua baris bocor**. Ini disebut **SQL injection**, salah satu serangan paling umum di dunia. Cara benar: selalu pakai tanda `?` (atau `:nama`) dan berikan nilainya terpisah; teks diperlakukan sebagai **data**, tidak pernah sebagai perintah. Hasilnya: tidak ada baris yang cocok.

Mengubah, menghapus, dan transaksi

```
# Bab 24 - UPDATE, DELETE, dan transaksi (commit / rollback)
import sqlite3

koneksi = sqlite3.connect(":memory:")
koneksi.execute("CREATE TABLE kolam (nama TEXT PRIMARY KEY, ikan INTEGER CHECK (ikan >= 0))")
koneksi.executemany("INSERT INTO kolam VALUES (?, ?)", [("A", 800), ("B", 650), ("C", 400)])
koneksi.commit()

koneksi.execute("UPDATE kolam SET ikan = ikan - 50 WHERE nama = 'A'")       # 50 ikan dipanen
koneksi.execute("DELETE FROM kolam WHERE nama = 'C'")                       # kolam C dibongkar
print(koneksi.execute("SELECT * FROM kolam ORDER BY nama").fetchall())
koneksi.commit()                                    # simpan dulu: titik aman sebelum percobaan berikut

# Transaksi: semua langkah berhasil bersama, atau batal bersama
try:
    with koneksi:                                    # commit otomatis; rollback bila ada galat
        koneksi.execute("UPDATE kolam SET ikan = ikan - 100 WHERE nama = 'B'")
        koneksi.execute("UPDATE kolam SET ikan = ikan - 900 WHERE nama = 'A'")   # melanggar CHECK
except sqlite3.IntegrityError as galat:
    print("Dibatalkan:", galat)

print("Setelah rollback:", koneksi.execute("SELECT * FROM kolam ORDER BY nama").fetchall())
```

```
[('A', 750), ('B', 650)]
Dibatalkan: CHECK constraint failed: ikan >= 0
Setelah rollback: [('A', 750), ('B', 650)]
```

Setelah panen 50 ikan, kolam A tinggal 750; kolam C dibongkar. Lalu datang **transaksi**: dua perubahan yang harus berhasil **bersama**. Perubahan kedua (kurangi 900 ikan dari A) melanggar aturan `CHECK (ikan >= 0)`, sehingga SQLite menolak dengan `IntegrityError`, dan blok `with koneksi:` **membatalkan** kedua

perubahan (*rollback*): kolam B tetap 650, bukan 550. Itulah gunanya transaksi: tidak ada keadaan setengah jadi.

> **INGAT**
>
> Basis data bisa menjaga aturan data untuk Anda lewat **batasan** (`NOT NULL`, `UNIQUE`, `CHECK`, `PRIMARY KEY`, `FOREIGN KEY`). Biarkan basis data menolak data buruk, jangan hanya mengandalkan kode Python.

Dua tabel: JOIN dan GROUP BY

```
# Bab 24 - Dua tabel: JOIN dan GROUP BY
import sqlite3

koneksi = sqlite3.connect(":memory:")
koneksi.executescript("""
    CREATE TABLE sektor (id INTEGER PRIMARY KEY, nama TEXT);
   CREATE TABLE panen (id INTEGER PRIMARY KEY, sektor_id INTEGER REFERENCES sektor(id),
                       komoditas TEXT, hasil_kg REAL);
    INSERT INTO sektor VALUES (1, 'Kebun'), (2, 'Pertanian'), (3, 'Perikanan');
    INSERT INTO panen (sektor_id, komoditas, hasil_kg) VALUES
        (1, 'kopi', 245.5), (1, 'kakao', 198.5), (1, 'kopi', 310.0),
        (2, 'padi', 3100.0), (2, 'jagung', 1850.0), (3, 'lele', 432.4);
""")

sql = """
    SELECT sektor.nama AS sektor, COUNT(*) AS jumlah_panen, ROUND(SUM(panen.hasil_kg),
1) AS total_kg
    FROM panen
    JOIN sektor ON sektor.id = panen.sektor_id
    GROUP BY sektor.nama
    HAVING SUM(panen.hasil_kg) > 500
    ORDER BY total_kg DESC
"""
for baris in koneksi.execute(sql):
    print(baris)
```

```
('Pertanian', 2, 4950.0)
('Kebun', 3, 754.0)
```

`JOIN` menyambung baris dua tabel yang `sektor_id`-nya sama dengan `id` sektor, sehingga nama sektor ikut tampil. `GROUP BY` mengelompokkan dan `SUM`/`COUNT` meringkas, persis seperti `groupby` pandas di Bab 23. `HAVING` menyaring **setelah** pengelompokan (hanya sektor dengan total di atas 500 kg), sedangkan `WHERE` menyaring **sebelumnya**. Pertanian: 2 panen, 4950,0 kg (3100 + 1850); Kebun: 3 panen, 754,0 kg; Perikanan dibuang oleh `HAVING` karena hanya 432,4 kg.

pandas dan SQL

```
# Bab 24 - pandas dan SQL saling bertukar data
import sqlite3
import pandas as pd

koneksi = sqlite3.connect(":memory:")

kolam = pd.DataFrame({"kolam": ["A", "B", "C"], "jenis": ["lele", "nila", "lele"],
                      "ikan": [800, 650, 400], "bobot_g": [540.5, 320.0, 480.0]})
kolam.to_sql("kolam", koneksi, index=False)              # DataFrame -> tabel SQL
```

```
hasil = pd.read_sql_query(
    "SELECT jenis, SUM(ikan) AS total_ikan, ROUND(AVG(bobot_g), 1) AS bobot_rata_g FROM
kolam GROUP BY jenis",
    koneksi,
)                                                    # hasil SQL -> DataFrame
print(hasil)

berat = pd.read_sql_query("SELECT * FROM kolam WHERE bobot_g > ?", koneksi,
params=(450,))
print(berat)
```

```
  jenis  total_ikan  bobot_rata_g
0  lele        1200         510.3
1  nila         650         320.0
  kolam jenis  ikan  bobot_g
0     A  lele   800    540.5
1     C  lele   400    480.0
```

`to_sql` mengubah DataFrame menjadi tabel; `read_sql_query` mengubah hasil `SELECT` menjadi DataFrame. Dua alat itu bersahabat: SQL bagus untuk memilah data besar di basis data, pandas bagus untuk menganalisis hasilnya.

SQLAlchemy: satu cara untuk banyak basis data

Menulis SQL langsung terikat pada satu merek basis data. **SQLAlchemy** adalah pustaka (`pip install sqlalchemy`) yang menjadi penerjemah: kode yang sama bisa berjalan di SQLite maupun PostgreSQL, cukup mengganti **alamat sambungan** (URL) pada `create_engine`. Ia punya dua lapisan.

Core: Anda tetap menulis SQL, tetapi dengan parameter bernama `:nama` yang aman dan pengelolaan sambungan yang rapi (`engine.begin()` = commit otomatis di akhir blok):

```
# Bab 24 - SQLAlchemy: engine dan text() (lapisan "Core")
from pathlib import Path
from sqlalchemy import create_engine, text

tmp = Path(__file__).parent / "_tmp"
tmp.mkdir(exist_ok=True)
berkas = tmp / "sqlalchemy_core.db"
berkas.unlink(missing_ok=True)

engine = create_engine(f"sqlite:///{berkas.as_posix()}")                # URL:
jenis_basis_data:///jalur

with engine.begin() as koneksi:                                    # begin(): commit
otomatis di akhir blok
    koneksi.execute(text("CREATE TABLE ternak (kandang TEXT, jenis TEXT, bobot_kg
REAL)"))
    koneksi.execute(
        text("INSERT INTO ternak VALUES (:kandang, :jenis, :bobot)"),     # :nama =
parameter bernama
        [{"kandang": "K1", "jenis": "sapi", "bobot": 350.0},
         {"kandang": "K1", "jenis": "sapi", "bobot": 412.5},
         {"kandang": "K2", "jenis": "sapi", "bobot": 389.0}],
    )

with engine.connect() as koneksi:
    hasil = koneksi.execute(text("SELECT kandang, AVG(bobot_kg) AS rata FROM ternak
GROUP BY kandang"))
    for baris in hasil:
        print(baris.kandang, round(baris.rata, 2))
```

```
print("dialek:", engine.dialect.name)
engine.dispose()                                          # lepaskan berkas
```

```
K1 381.25
K2 389.0
dialek: sqlite
```

ORM (*Object-Relational Mapper*): tabel ditulis sebagai **kelas Python** (ingat OOP, Bab 16), baris menjadi **objek**, dan hubungan antartabel menjadi atribut:

```
# Bab 24 - SQLAlchemy ORM: tabel sebagai kelas Python
from pathlib import Path
from sqlalchemy import ForeignKey, create_engine, func, select
from sqlalchemy.orm import DeclarativeBase, Mapped, Session, mapped_column,
relationship

tmp = Path(__file__).parent / "_tmp"
tmp.mkdir(exist_ok=True)
berkas = tmp / "sqlalchemy_orm.db"
berkas.unlink(missing_ok=True)


class Dasar(DeclarativeBase):
    pass


class Sektor(Dasar):
    __tablename__ = "sektor"
    id: Mapped[int] = mapped_column(primary_key=True)
    nama: Mapped[str]
    panen: Mapped[list["Panen"]] = relationship(back_populates="sektor")


class Panen(Dasar):
    __tablename__ = "panen"
    id: Mapped[int] = mapped_column(primary_key=True)
    komoditas: Mapped[str]
    hasil_kg: Mapped[float]
    sektor_id: Mapped[int] = mapped_column(ForeignKey("sektor.id"))
    sektor: Mapped[Sektor] = relationship(back_populates="panen")


engine = create_engine(f"sqlite:///{berkas.as_posix()}")
Dasar.metadata.create_all(engine)                 # membuat tabel dari definisi kelas

with Session(engine) as sesi:
    kebun = Sektor(nama="Kebun", panen=[Panen(komoditas="kopi", hasil_kg=245.5),
                                        Panen(komoditas="kakao", hasil_kg=198.5)])
    sawah = Sektor(nama="Pertanian", panen=[Panen(komoditas="padi", hasil_kg=3100.0)])
    sesi.add_all([kebun, sawah])
    sesi.commit()

with Session(engine) as sesi:
    for sektor in sesi.scalars(select(Sektor).order_by(Sektor.nama)):
          print(sektor.nama, "->", [p.komoditas for p in sektor.panen])          #
relasi otomatis

    terbesar = sesi.scalars(select(Panen).order_by(Panen.hasil_kg.desc())).first()
       print("Panen terbesar:", terbesar.komoditas, terbesar.hasil_kg, "kg dari",
terbesar.sektor.nama)
```

```
    print("Total kg:", sesi.scalar(select(func.sum(Panen.hasil_kg))))
engine.dispose()
```

```
Kebun -> ['kopi', 'kakao']
Pertanian -> ['padi']
Panen terbesar: padi 3100.0 kg dari Pertanian
Total kg: 3544.0
```

`Sektor.panen` berisi daftar objek `Panen` tanpa Anda menulis `JOIN`; sebaliknya `terbesar.sektor.nama` menelusuri ke atas. Total 3544,0 kg = 245,5 + 198,5 + 3100,0.

PostgreSQL: basis data tingkat server

PostgreSQL adalah basis data berupa **server** yang dipakai banyak orang bersamaan, dengan fitur yang jauh lebih lengkap (pengguna dan hak akses, kapasitas besar, ekstensi seperti PostGIS untuk peta di Seri A2). Dengan SQLAlchemy, perbedaan terbesar hanyalah **alamat sambungannya**, berbentuk `postgresql+psycopg://PENGGUNA:KATA_SANDI@SERVER:5432/NAMA_BASIS_DATA`, yang memakai pustaka `psycopg` versi 3 (`pip install "psycopg[binary]"`).

```
# Bab 24 - Menyambung ke PostgreSQL (server sementara yang dinyalakan alat pembangun
buku)
import os
from sqlalchemy import create_engine, text

# Alamat sambungan dibaca dari variabel lingkungan, jadi kata sandi tidak tertulis
di kode.
# Bentuk alamat di server milik Anda:
#   postgresql://PENGGUNA:KATA_SANDI@SERVER:5432/NAMA_BASIS_DATA
# Contoh nilai variabel (kata sandi disamarkan):
#   DATABASE_URL = postgresql://petugas:[disamarkan]@localhost:5432/hutan_kebun
alamat = os.environ["DATABASE_URL"]

# SQLAlchemy memilih driver dari awalan alamat; "+psycopg" meminta psycopg versi 3.
alamat = alamat.replace("postgresql://", "postgresql+psycopg://", 1)
engine = create_engine(alamat)

with engine.begin() as koneksi:
    koneksi.execute(text("""
        CREATE TABLE IF NOT EXISTS panen (
            id SERIAL PRIMARY KEY,
            komoditas TEXT NOT NULL,
            hasil_kg NUMERIC(10, 1)
        )
    """))
    koneksi.execute(text("INSERT INTO panen (komoditas, hasil_kg) VALUES (:k, :h)"),
{"k": "kopi", "h": 245.5})

with engine.connect() as koneksi:
    print(koneksi.execute(text("SELECT komoditas, hasil_kg FROM panen")).fetchall())
```

```
[('kopi', Decimal('245.5'))]
```

Kode ini **dijalankan sungguhan** saat buku dibuat, pada PostgreSQL 14.5 **sementara** yang dinyalakan alat pembangun buku: server itu hanya hidup selama buku dibuat, hanya bisa dihubungi dari komputer penulis, tanpa kata sandi, lalu dihapus. Server milik pembaca atau kantor biasanya memakai nama pengguna dan kata sandi; bagian itu **tidak diuji** di sini. Perhatikan dua hal pada keluarannya. Pertama, kolom `NUMERIC` kembali sebagai `Decimal('245.5')`, bukan `float` seperti di SQLite. Kedua, baris `alamat.replace(...)` menambahkan `+psycopg` pada awalan alamat supaya SQLAlchemy memakai driver psycopg 3.

Untuk mencobanya di komputer Anda, atur dulu variabel lingkungannya (contoh di PowerShell, kata sandi disamarkan): `$env:DATABASE_URL = "postgresql://petugas:[disamarkan]@localhost:5432/hutan_kebun"`. Perhatikan dua kebiasaan aman: alamat dibaca dari variabel lingkungan `DATABASE_URL` (kata sandi tidak ada di kode), dan contoh nilainya ditulis `[disamarkan]`.

BAGIAN TEKNIS (boleh dilewati)

Beda SQLite dan PostgreSQL yang akan Anda temui: PostgreSQL ketat pada jenis kolom (kolom `NUMERIC` kembali sebagai `Decimal`, bukan `float`; ini **terbukti** dari keluaran di atas), memakai `SERIAL` atau `GENERATED ... AS IDENTITY` untuk nomor otomatis (`SERIAL` dipakai dan berjalan di contoh), dan sebuah tabel yang dibuat dalam transaksi baru terlihat orang lain setelah commit. SQLite lebih longgar soal jenis kolom. Itulah alasan menguji di mesin yang akan dipakai sungguhan. (Butir tentang `GENERATED ... AS IDENTITY` dan visibilitas transaksi adalah pengetahuan umum yang **belum diuji** di buku ini.)

Proyek Bersambung I2: menyimpan data gabungan

Langkah 4: data fiktif disimpan sebagai **tiga tabel terpisah** (panen, cuaca, harga) di berkas SQLite, seperti basis data sungguhan, lalu satu pertanyaan SQL dengan dua `LEFT JOIN` menghitung total hasil, pendapatan, dan hujan rata-rata per sektor.

```
# Bab 24 - Proyek bersambung I2, langkah 4: menyimpan data gabungan ke basis data
from pathlib import Path
import pandas as pd
from sqlalchemy import create_engine, text

tmp = Path(__file__).parent / "_tmp"
tmp.mkdir(exist_ok=True)
berkas = tmp / "proyek_i2.db"
berkas.unlink(missing_ok=True)

# Data (fiktif) disalin supaya skrip berdiri sendiri; bentuknya sama dengan Bab 21 dan 23.
panen = pd.DataFrame(
    [("2026-01-01", "Hutan", "Plot A", "jati", 1200.0), ("2026-01-01", "Kebun", "Blok A", "kopi", 245.5),
     ("2026-01-02", "Kebun", "Blok B", "kakao", 198.5), ("2026-01-02", "Pertanian", "Sawah 1", "padi", 3100.0),
     ("2026-01-03", "Pertanian", "Sawah 2", "jagung", 1850.0), ("2026-01-03", "Perikanan", "Kolam A", "lele", 432.4),
     ("2026-01-04", "Peternakan", "Kandang 1", "sapi", 1151.5), ("2026-01-04", "Kebun", "Blok C", "kopi", 310.0),
     ("2026-01-05", "Pertanian", "Sawah 1", "padi", 2950.0), ("2026-01-05", "Kebun", "Blok D", "kakao", 276.0)],
    columns=["tanggal", "sektor", "unit", "komoditas", "hasil_kg"])
tanggal = ["2026-01-01", "2026-01-02", "2026-01-03", "2026-01-04", "2026-01-05"]
cuaca = pd.DataFrame({"tanggal": tanggal, "suhu_maks": [30.1, 31.4, 29.8, 28.9, 30.6],
                      "hujan_mm": [12.5, 0.0, 8.2, 25.4, 3.1]})
daftar_harga = {"padi": [6200, 6250, 6180, 6300, 6270], "jagung": [5100, 5150, 5200, 5120, 5180],
                "kopi": [58000, 58500, 59000, 57500, 58200], "kakao": [62000, 61500, 62500, 63000, 62800]}
harga = pd.DataFrame([(t, k, h) for k, v in daftar_harga.items() for t, h in zip(tanggal, v)],
                     columns=["tanggal", "komoditas", "harga_per_kg"])

engine = create_engine(f"sqlite:///{berkas.as_posix()}")
panen.to_sql("panen", engine, index=False)        # tiga tabel terpisah, seperti basis data sungguhan
```

```
cuaca.to_sql("cuaca", engine, index=False)
harga.to_sql("harga", engine, index=False)

sql = text("""
    SELECT p.sektor,
           ROUND(SUM(p.hasil_kg), 1) AS hasil_kg,
           ROUND(SUM(p.hasil_kg * h.harga_per_kg), 0) AS pendapatan_rp,
           ROUND(AVG(c.hujan_mm), 1) AS hujan_rata_mm
    FROM panen p
    LEFT JOIN harga h ON h.tanggal = p.tanggal AND h.komoditas = p.komoditas
    LEFT JOIN cuaca c ON c.tanggal = p.tanggal
    GROUP BY p.sektor
    ORDER BY p.sektor
""")
with engine.connect() as koneksi:
    for tabel in ["panen", "cuaca", "harga"]:
        n = koneksi.execute(text(f"SELECT COUNT(*) FROM {tabel}")).scalar()
        print(f"tabel {tabel:6}: {n} baris")
    print(pd.read_sql_query(sql, koneksi).to_string(index=False))
engine.dispose()
```

```
tabel panen : 10 baris
tabel cuaca : 5 baris
tabel harga : 20 baris
    sektor  hasil_kg  pendapatan_rp  hujan_rata_mm
     Hutan    1200.0            NaN           12.5
     Kebun    1030.0     61604550.0           10.3
 Perikanan     432.4            NaN            8.2
 Pertanian    7900.0     47491500.0            3.8
Peternakan    1151.5            NaN           25.4
```

Hasilnya cocok dengan pandas di Bab 23: Kebun 1030,0 kg dan Rp 61.604.550; Pertanian 7900,0 kg dan Rp 47.491.500. Sektor tanpa harga menunjukkan `NaN` pada pendapatan karena `SUM` atas nilai kosong tetap kosong. Beda dengan pandas yang menjumlahkan kosong menjadi 0. **Hujan rata-rata per sektor** adalah rata-rata hujan pada hari-hari panen sektor itu (bukan hujan sebulan), jadi tafsirkan dengan hati-hati.

Latihan

1. Di `01_sqlite_dasar.py`, tambahkan satu pohon baru, lalu cetak hanya pohon dengan `plot = 'B'` (pakai `WHERE plot = ?`).
2. Di `03_ubah_hapus_transaksi.py`, ubah pengurangan A menjadi 100 sehingga tidak melanggar aturan. Apa yang terjadi pada seluruh transaksi?
3. Di `04_join_agregasi.py`, hapus `HAVING` dan tambahkan `ORDER BY sektor.nama`. Sektor apa saja yang tampil?
4. **Tantangan:** di `07_sqlalchemy_orm.py`, tambahkan sektor "Perikanan" dengan satu panen "lele" 432,4 kg, lalu cetak total hasil per sektor dengan `func.sum` dan `group_by`.

Ringkasan Bab

- **Basis data** = lemari arsip berlaci: tabel, baris, kolom; **SQL** = bahasa untuk bertanya dan mengubahnya.
- **SQLite** = satu berkas, bawaan Python (`sqlite3`); simpan perubahan dengan `commit()`.
- Selalu pakai parameter (`?`, `:nama`), **jangan** menempel teks pengguna ke SQL (SQL injection).
- **Transaksi** membuat sekelompok perubahan berhasil bersama atau batal bersama; batasan (`CHECK`, `NOT NULL`, ...) menjaga mutu data.
- `JOIN` menyambung tabel; `GROUP BY` + `SUM`/`COUNT` meringkas; `WHERE` menyaring sebelum, `HAVING` sesudah pengelompokan.
- **SQLAlchemy** (Core dan ORM) menyatukan akses ke banyak basis data; **PostgreSQL** cukup mengganti alamat sambungan, dan bagian itu dijalankan di buku ini pada server sementara (server milik Anda tetap butuh pengguna dan kata sandi).

Selanjutnya, di Bab 25: data tersimpan rapi. Saatnya menganalisisnya: pandas lanjut, numpy, matplotlib, scikit-learn, dan Jupyter Notebook, lalu menutup Seri I2.

Bab 25 — Data Science dan Analisis: Dari Tabel Menjadi Gambar dan Dugaan

Seri I2 — Data dan Internet

Di bab ini, Anda akan:

- Memakai pandas lanjut: kolom turunan, kategori, dan data berurutan waktu
- Memakai numpy untuk statistik, `np.where`, dan **broadcasting**
- Menggambar grafik dengan **matplotlib**
- Membuat model pertama dengan **scikit-learn**: regresi dan klasifikasi, serta menguji dengan data yang belum dilihat
- Mengenal **Jupyter Notebook** dan **Anaconda**, lalu menutup Proyek Bersambung I2

Semua skrip bab ini ada di folder `kode/bab25/`. Di versi web, Anda juga bisa mengunduh semuanya sekaligus (zip).

Data sudah masuk (Bab 21), bersih (22), tertata (23), dan tersimpan (24). Sekarang ada pertanyaan khas ilmuwan data: *"Apa yang bisa kita lihat dan duga dari data ini?"* Bayangkan Anda mandor kebun yang menyimpan catatan hujan dan panen kopi dua bulan. Mandor yang baik melakukan empat hal: **memeriksa** (ringkasan angka), **menggambar** (grafik), **menduga** (model), dan **mengujinya** (apakah dugaan itu masih benar untuk data baru?). Bab ini mencoba keempatnya. Semua data di sini **fiktif**, dibuat dengan bilangan acak berbenih tetap (`default_rng(42)`) agar hasilnya selalu sama; karena itu, rumus pembuatnya diketahui dan kita bisa memeriksa apakah model menemukannya kembali.

pandas lanjut: kolom turunan dan kategori

```
# Bab 25 - pandas lanjut: kolom baru, kategori, dan ringkasan
import numpy as np
import pandas as pd

rng = np.random.default_rng(42)                       # benih tetap: hasil selalu sama
n = 60
data = pd.DataFrame({
    "tanggal": pd.date_range("2026-01-01", periods=n),
    "hujan_mm": rng.gamma(shape=1.2, scale=8.0, size=n).round(1),
})
data["hasil_kopi_kg"] = (200 + 1.5 * data["hujan_mm"].clip(upper=30) + rng.normal(0,
8, n)).round(1)

# pd.cut: mengubah angka menjadi kategori berdasarkan batas
data["kategori"] = pd.cut(data["hujan_mm"], bins=[-0.1, 5, 15, 1000], labels=["kering",
"sedang", "lebat"])
print(data.head(5))
print(data["kategori"].value_counts())

ringkas   =   data.groupby("kategori",   observed=True)["hasil_kopi_kg"].agg(["count",
"mean", "max"]).round(1)
print(ringkas)

data["di_atas_rata"] = data["hasil_kopi_kg"] > data["hasil_kopi_kg"].mean()   # kolom
benar/salah
```

```
print("Hari di atas rata-rata:", int(data["di_atas_rata"].sum()), "dari", n)
print(data.nlargest(3, "hasil_kopi_kg")[["tanggal", "hujan_mm", "hasil_kopi_kg"]])
```

```
     tanggal  hujan_mm  hasil_kopi_kg kategori
0 2026-01-01       9.5          220.9   sedang
1 2026-01-02      14.2          224.2   sedang
2 2026-01-03       7.9          223.6   sedang
3 2026-01-04       6.8          200.7   sedang
4 2026-01-05      15.8          218.6    lebat
kategori
sedang    29
kering    24
lebat      7
Name: count, dtype: int64
          count   mean    max
kategori
kering       24  201.5  221.7
sedang       29  216.5  242.2
lebat         7  227.7  242.3
Hari di atas rata-rata: 30 dari 60
      tanggal  hujan_mm  hasil_kopi_kg
57 2026-02-27      27.0          242.3
15 2026-01-16      12.6          242.2
38 2026-02-08      12.7          236.1
```

- Kolom baru dibuat dengan menugaskan hasil rumus: `data["hasil_kopi_kg"] = ...`.
- `pd.cut` mengubah angka menjadi kategori lewat batas: hujan 0 sampai 5 mm = kering, sampai 15 = sedang, di atasnya = lebat. Dari 60 hari: 24 kering, 29 sedang, 7 lebat.
- `groupby(...).agg(...)` memberi hasil kopi rata-rata per kategori: **201,5 kg** (kering), **216,5 kg** (sedang), dan **227,7 kg** (lebat). Pola naik terlihat jelas.
- `nlargest(3, ...)` mengambil tiga hari panen terbanyak; `.sum()` pada kolom `True/False` menghitung jumlah `True` (30 hari di atas rata-rata).

Data berurutan waktu

```
# Bab 25 - Data berurutan waktu: indeks tanggal, resample, rolling
import numpy as np
import pandas as pd

rng = np.random.default_rng(42)
hujan = pd.Series(rng.gamma(1.2, 8.0, 28).round(1), index=pd.date_range("2026-01-01",
periods=28), name="hujan_mm")

print(hujan.loc["2026-01-03":"2026-01-05"])           # memilih lewat rentang tanggal
mingguan = hujan.resample("W").sum().round(1)            # jumlah per minggu (minggu
berakhir hari Minggu)
print(mingguan)

rata_geser = hujan.rolling(window=7).mean().round(2)    # rata-rata 7 hari terakhir
print(rata_geser.iloc[[5, 6, 7, 27]])                         # 6 hari pertama masih
kosong (NaN)
print("Selisih        dari        kemarin,        3        hari        pertama:",
hujan.diff().iloc[1:4].round(1).tolist())
```

```
2026-01-03     7.9
2026-01-04     6.8
2026-01-05    15.8
Freq: D, Name: hujan_mm, dtype: float64
```

```
2026-01-04    38.4
2026-01-11    86.2
2026-01-18    51.3
2026-01-25    64.7
2026-02-01     9.9
Freq: W-SUN, Name: hujan_mm, dtype: float64
2026-01-06      NaN
2026-01-07    10.37
2026-01-08    10.46
2026-01-28     8.00
Name: hujan_mm, dtype: float64
Selisih dari kemarin, 3 hari pertama: [4.7, -6.3, -1.1]
```

Bila indeks berupa tanggal, Anda bisa **memotong menurut rentang tanggal** (`loc["2026-01-03":"2026-01-05"]`), **meringkas per periode** (`resample("W")` = per minggu), dan menghitung **rata-rata bergeser** (`rolling(7).mean()`). Enam hari pertama kosong (`NaN`) karena belum ada tujuh hari yang bisa dirata-ratakan; hari ke-7 (7 Januari) bernilai 10,37. Perhatikan minggu pertama (4 hari) dan minggu terakhir (berakhir 1 Februari, hanya 3 hari data) tidak penuh; jumlah minggu terakhir kecil (9,9 mm) karena itu. Periode yang tidak lengkap bisa menyesatkan.

numpy lanjut

```
# Bab 25 - numpy lanjut: statistik, np.where, dan broadcasting
import numpy as np

rng = np.random.default_rng(42)
hujan = rng.gamma(1.2, 8.0, 30).round(1)
suhu = (28 + 0.05 * hujan + rng.normal(0, 1, 30)).round(1)

print("persentil 25/50/75:", np.percentile(hujan, [25, 50, 75]).round(2))
print("hari paling basah: hari ke-", int(hujan.argmax()) + 1, "dengan", hujan.max(),
"mm")
print("korelasi hujan-suhu:", np.corrcoef(hujan, suhu)[0, 1].round(3))

label = np.where(hujan > 10, "basah", "kering")        # if-else untuk satu array penuh
print("jumlah hari basah:", int((label == "basah").sum()), "dari", label.size)

kumulatif = np.cumsum(hujan)                           # jumlah berjalan
print("hujan kumulatif hari ke-30:", kumulatif[-1].round(1))

# Broadcasting: array 2 dimensi dikurangi array 1 dimensi (per kolom), tanpa perulangan
tabel = np.column_stack([hujan, suhu])
z = (tabel - tabel.mean(axis=0)) / tabel.std(axis=0)    # standarisasi: rata-rata 0,
sebaran 1
print("rata-rata  setelah  standarisasi:",  z.mean(axis=0).round(6),  "|  sebaran:",
z.std(axis=0).round(6))
```

```
persentil 25/50/75: [ 4.05  7.85 12.38]
hari paling basah: hari ke- 11 dengan 20.6 mm
korelasi hujan-suhu: 0.233
jumlah hari basah: 12 dari 30
hujan kumulatif hari ke-30: 253.2
rata-rata setelah standarisasi: [-0. -0.] | sebaran: [1. 1.]
```

- `np.percentile` memberi nilai batas: separuh hari punya hujan di bawah 7,85 mm (median).
- `argmax()` memberi **posisi** nilai terbesar (hari ke-11, 20,6 mm); karena posisi mulai dari 0, kita tambah 1 untuk nomor hari.
- `np.corrcoef` menghitung **korelasi** (0,233: lemah). Korelasi hanya mengukur seberapa searah dua angka; ia **bukan** bukti sebab-akibat. Seri I3 membahasnya dalam.

- `np.where(syarat, a, b)` adalah `if-else` untuk satu array penuh. 12 dari 30 hari "basah".
- **Broadcasting**: `tabel - tabel.mean(axis=0)` mengurangkan rata-rata tiap kolom dari kolomnya sendiri tanpa perulangan. Setelah standarisasi, rata-rata menjadi 0 dan sebaran 1 (nilai `-0.` hanyalah nol negatif yang sangat kecil dari pembulatan).

matplotlib: menggambar

matplotlib (`pip install matplotlib`) adalah pustaka grafik paling dasar di Python. Dua gagasan: **figure** (kanvas) dan **axes** (satu bidang gambar di kanvas itu).

```
# Bab 25 - matplotlib: grafik garis dan batang
from pathlib import Path
import matplotlib
matplotlib.use("Agg")                                          # menggambar ke berkas,
tanpa jendela
import matplotlib.pyplot as plt
import numpy as np

tmp = Path(__file__).parent / "_tmp"
tmp.mkdir(exist_ok=True)

rng = np.random.default_rng(42)
hari = np.arange(1, 31)
hujan = rng.gamma(1.2, 8.0, 30).round(1)

fig, (atas, bawah) = plt.subplots(2, 1, figsize=(8, 6), sharex=True)
atas.plot(hari, hujan, marker="o", color="tab:blue")
atas.set_ylabel("Hujan (mm)")
atas.set_title("Hujan harian, Januari 2026 (data fiktif)")
bawah.bar(hari, np.cumsum(hujan), color="tab:green")
bawah.set_ylabel("Hujan kumulatif (mm)")
bawah.set_xlabel("Hari ke-")
fig.tight_layout()
fig.savefig(tmp / "hujan.png", dpi=100)
plt.close(fig)

print("Gambar tersimpan:", (tmp / "hujan.png").exists())
print("Jumlah sumbu (axes):", len(fig.axes))
```

```
Gambar tersimpan: True
Jumlah sumbu (axes): 2
```

`matplotlib.use("Agg")` membuat matplotlib menggambar ke berkas tanpa membuka jendela (cocok untuk skrip otomatis). Grafik tersimpan di `_tmp/hujan.png`; skrip hanya mencetak bahwa berkas ada dan bahwa figure punya 2 axes. Bentuk umum: `plot` (garis), `bar` (batang), `scatter` (titik), lalu beri `title`, `xlabel`, `ylabel`, dan `savefig`.

> TIPS
>
> Setiap grafik harus menjawab satu pertanyaan, dan sumbu serta judulnya harus menyebut **satuan**. Grafik tanpa satuan membuat pembaca menebak. Beri pula catatan "data fiktif" bila memang fiktif.

```
# Bab 25 - Diagram sebar dan garis kecenderungan
from pathlib import Path
import matplotlib
matplotlib.use("Agg")
import matplotlib.pyplot as plt
import numpy as np

tmp = Path(__file__).parent / "_tmp"
```

```
tmp.mkdir(exist_ok=True)

rng = np.random.default_rng(42)
hujan = rng.uniform(0, 30, 40)
hasil = 200 + 1.5 * hujan + rng.normal(0, 8, 40)       # hubungan sebenarnya: +1,5 kg
per mm (kita tahu karena kita yang membuatnya)

kemiringan, titik_potong = np.polyfit(hujan, hasil, deg=1)   # garis lurus terbaik
print(f"garis: hasil = {titik_potong:.1f} + {kemiringan:.2f} x hujan")

plt.figure(figsize=(6, 4))
plt.scatter(hujan, hasil, alpha=0.7)
garis_x = np.array([0, 30])
plt.plot(garis_x, titik_potong + kemiringan * garis_x, color="tab:red")
plt.xlabel("Hujan (mm)")
plt.ylabel("Hasil kopi (kg)")
plt.title("Hujan dan hasil kopi (data fiktif)")
plt.tight_layout()
plt.savefig(tmp / "sebar.png", dpi=100)
plt.close()
print("Gambar tersimpan:", (tmp / "sebar.png").exists())
```

```
garis: hasil = 202.6 + 1.34 x hujan
Gambar tersimpan: True
```

Diagram sebar (*scatter*) memperlihatkan hubungan dua angka. `np.polyfit(..., deg=1)` mencari garis lurus terbaik: **hasil = 202,6 + 1,34 x hujan**. Padahal aturan pembuat data sebenarnya 200 + 1,5 x hujan. Selisihnya wajar karena hanya 40 titik yang diberi derau (*noise*); makin banyak data, makin dekat. Ini pelajaran penting: **hasil dari sampel kecil hanya perkiraan**.

scikit-learn: model pertama

scikit-learn (`pip install scikit-learn`; di kode ditulis `sklearn`) adalah pustaka pembelajaran mesin (*machine learning*) klasik. Prosesnya selalu mirip: **bagi data** menjadi data latih dan data uji, **latih** model (`fit`), lalu **uji** pada data yang belum pernah dilihat (`predict`).

Regresi (memprediksi angka):

```
# Bab 25 - scikit-learn: model regresi pertama (memprediksi angka)
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, r2_score
from sklearn.model_selection import train_test_split

rng = np.random.default_rng(42)
hujan = rng.uniform(0, 30, 80)
hasil = 200 + 1.5 * hujan + rng.normal(0, 8, 80)

X = hujan.reshape(-1, 1)                                  # scikit-learn minta fitur
berbentuk tabel (baris x kolom)
y = hasil
X_latih,   X_uji,   y_latih,   y_uji   =   train_test_split(X,   y,   test_size=0.25,
random_state=42)
print("data latih:", len(X_latih), "| data uji:", len(X_uji))

model = LinearRegression()
model.fit(X_latih, y_latih)                               # "belajar" dari data latih
print(f"model: hasil = {model.intercept_:.1f} + {model.coef_[0]:.2f} x hujan")

prediksi = model.predict(X_uji)                           # ujian: data yang belum
pernah dilihat
```

```
print("R2 pada data uji:", round(r2_score(y_uji, prediksi), 3))
print("Rata-rata selisih mutlak (kg):", round(mean_absolute_error(y_uji, prediksi),
2))
print("Prediksi untuk hujan 10 mm dan 25 mm:", model.predict([[10], [25]]).round(1))
```

```
data latih: 60 | data uji: 20
model: hasil = 196.6 + 1.63 x hujan
R2 pada data uji: 0.709
Rata-rata selisih mutlak (kg): 7.42
Prediksi untuk hujan 10 mm dan 25 mm: [212.9 237.4]
```

Dari 80 data, 60 dipakai melatih dan 20 untuk ujian. Model menemukan **hasil = 196,6 + 1,63 x hujan**. **R2** pada data uji 0,709 artinya model menjelaskan kira-kira 71% variasi hasil; rata-rata meleset sekitar 7,42 kg. Prediksi untuk hujan 10 mm dan 25 mm: 212,9 dan 237,4 kg.

> **i** INGAT
>
> Selalu uji model dengan **data yang tidak ikut dipakai melatih**. Model yang hanya diuji dengan data latihnya itu seperti murid yang diuji dengan soal yang sudah ia hafal jawabannya.

Klasifikasi (memprediksi kategori), yaitu pohon sehat atau sakit:

```
# Bab 25 - scikit-learn: klasifikasi (memprediksi kategori) pohon sehat atau sakit
import numpy as np
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier

rng = np.random.default_rng(42)
n = 120
diameter = rng.uniform(15, 55, n)
tinggi = diameter * 0.55 + rng.normal(0, 2, n)
# aturan fiktif untuk membuat data: pohon langsing (tinggi besar, diameter kecil) lebih
sering sakit
peluang_sakit = 1 / (1 + np.exp(-(tinggi / diameter - 0.62) * 25))
sakit = rng.uniform(size=n) < peluang_sakit

X = np.column_stack([diameter, tinggi])
X_latih, X_uji, y_latih, y_uji = train_test_split(X, sakit, test_size=0.3,
random_state=42, stratify=sakit)

pohon = DecisionTreeClassifier(max_depth=3, random_state=42).fit(X_latih, y_latih)
print("Akurasi data latih:", round(accuracy_score(y_latih, pohon.predict(X_latih)),
3))
print("Akurasi data uji  :", round(accuracy_score(y_uji, pohon.predict(X_uji)), 3))
print("Pembanding (selalu menebak 'tidak sakit'):", round(float((~y_uji).mean()), 3))
print("Pentingnya fitur (diameter, tinggi):", pohon.feature_importances_.round(2))
print("Prediksi pohon 20 cm/16 m dan 40 cm/22 m:", pohon.predict([[20, 16], [40, 22]]))
```

```
Akurasi data latih: 0.893
Akurasi data uji  : 0.694
Pembanding (selalu menebak 'tidak sakit'): 0.778
Pentingnya fitur (diameter, tinggi): [0.42 0.58]
Prediksi pohon 20 cm/16 m dan 40 cm/22 m: [False  True]
```

Hasilnya mengajarkan sesuatu yang jujur. Akurasi data latih 0,893, tetapi data uji hanya 0,694, **lebih rendah** daripada tebakan bodoh "selalu bilang tidak sakit" (0,778). Model **menghafal** data latih (*overfitting*) dan belum berguna pada data baru. Dengan hanya 120 pohon dan model sebebas pohon keputusan, ini wajar. Cara memperbaikinya (data lebih banyak, model lebih sederhana, validasi silang, ukuran lain selain akurasi) adalah bahan Seri I3 dan Seri A3. Yang penting sekarang: **selalu bandingkan model dengan pembanding sederhana**.

Jupyter Notebook dan Anaconda

Jupyter Notebook adalah "buku catatan" interaktif: sel teks (Markdown) dan sel kode berselang-seling, hasil (angka, tabel, grafik) tampil tepat di bawah sel kodenya. Sangat populer untuk eksplorasi data karena Anda bisa mencoba, melihat, dan menjelaskan di satu tempat. Berkasnya berekstensi `.ipynb`, sebenarnya JSON biasa.

Anaconda adalah paket "semua dalam satu" yang membawa Python, pengelola paket `conda`, Jupyter, dan banyak pustaka ilmu data (pemasangan dan perbandingannya dengan `venv` dibahas di Bab 17). Di komputer penulis, perintah `conda` **tidak ditemukan**, jadi Anaconda **tidak dijalankan** di buku ini. Notebook di bab ini dijalankan dengan Jupyter yang terpasang di environment proyek (`pip`), yang cara kerjanya sama. Anda bisa memakai keduanya: Anaconda Navigator bila ingin antarmuka klik-klik, atau `pip install jupyterlab` lalu `jupyter lab`.

Skrip berikut membuat notebook lewat kode (`nbformat`), menjalankan semua selnya (`nbconvert`), lalu mencetak keluarannya, seperti menekan "Run All".

```
# Bab 25 - Membuat dan menjalankan Jupyter Notebook lewat kode
import os
from pathlib import Path
import nbformat
from nbconvert.preprocessors import ExecutePreprocessor

tmp = Path(__file__).parent / "_tmp"
tmp.mkdir(exist_ok=True)

buku = nbformat.v4.new_notebook()
buku.cells = [
     nbformat.v4.new_markdown_cell("# Catatan hujan\nNotebook kecil: teks dan kode
berselang-seling."),
    nbformat.v4.new_code_cell("import pandas as pd\nhujan = pd.Series([12.5, 0.0, 8.2,
25.4, 3.1])\nhujan.sum()"),
                                        nbformat.v4.new_code_cell("print('Rata-rata:',
hujan.mean())\nhujan.describe().round(2)"),
]
nbformat.write(buku, tmp / "catatan_hujan.ipynb")

# Menjalankan semua sel, seperti menekan "Run All" di Jupyter
# (Peringatan teknis kernel di layar kita sembunyikan sementara: isinya soal sambungan
lokal, bukan galat.)
layar_galat = os.dup(2)
os.dup2(os.open(os.devnull, os.O_WRONLY), 2)
try:
           ExecutePreprocessor(timeout=120,   kernel_name="python3").preprocess(buku,
{"metadata": {"path": str(tmp)}})
finally:
    os.dup2(layar_galat, 2)

for nomor, sel in enumerate(buku.cells, start=1):
    if sel.cell_type != "code":
        continue
    print(f"--- sel {nomor} ---")
    for keluar in sel.outputs:
        if keluar.output_type == "stream":
            print(keluar.text.rstrip())
        elif keluar.output_type == "execute_result":
            print(keluar.data["text/plain"])
```

```
--- sel 2 ---
np.float64(49.199999999999996)
--- sel 3 ---
Rata-rata: 9.84
```

```
count     5.00
mean      9.84
std       9.93
min       0.00
25%       3.10
50%       8.20
75%      12.50
max      25.40
dtype: float64
```

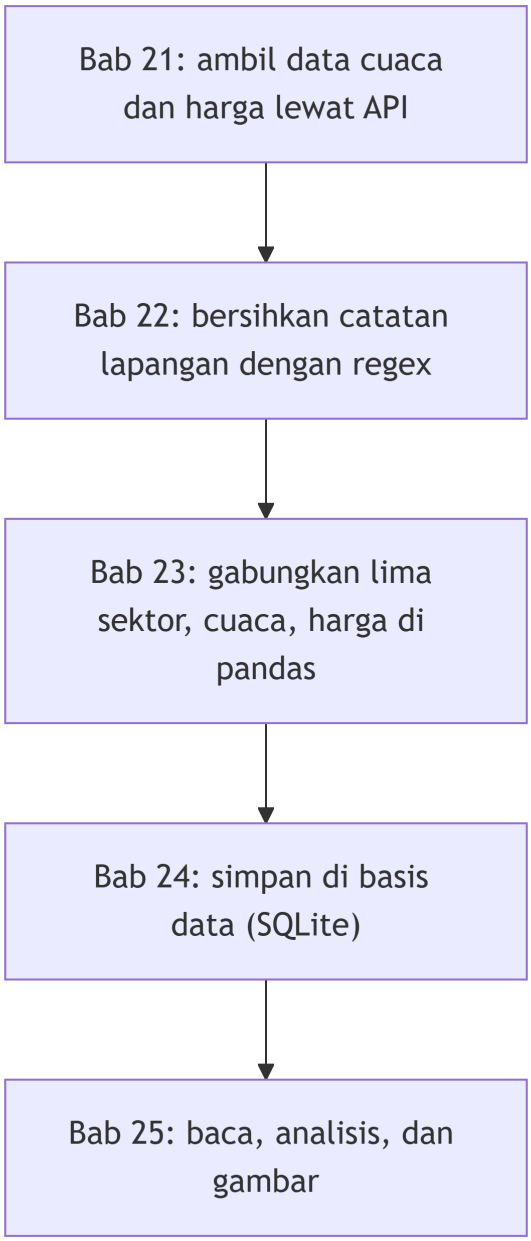
Sel 2 menjumlahkan hujan: `np.float64(49.199999999999996)`. Itulah isi nyata hasil sel; angka desimal komputer memang tidak selalu tepat (Bab 5), dan numpy 2 menampilkan jenisnya. Sel 3 mencetak rata-rata 9,84 lalu hasil `describe()` (hasil ekspresi terakhir sel otomatis ditampilkan, tanpa `print`). Berkas notebook yang dibuat ada di `_tmp/catatan_hujan.ipynb`; bukalah dengan JupyterLab untuk melihat tampilannya.

BAGIAN TEKNIS (boleh dilewati)

Notebook menyimpan **keadaan** kernel (urutan eksekusi sel). Sel yang dijalankan tidak berurutan bisa membuat hasil yang tidak bisa diulang dari nol. Kebiasaan baik: sebelum membagikan notebook, pilih "Restart kernel and run all" dan pastikan semua sel tetap berjalan. Untuk laporan yang bisa direproduksi, Bab 29 membahas Quarto dan cara lain.

Proyek Bersambung I2: dari basis data ke grafik

Penutup Proyek I2. Skrip menyimpan data fiktif ke SQLite (ringkasan Bab 21-24), **membacanya kembali** dengan satu pertanyaan SQL, menganalisis dengan pandas, dan menggambar dua grafik.



Proyek bersambung I2 dari awal sampai akhir: lima bab, satu alur data.

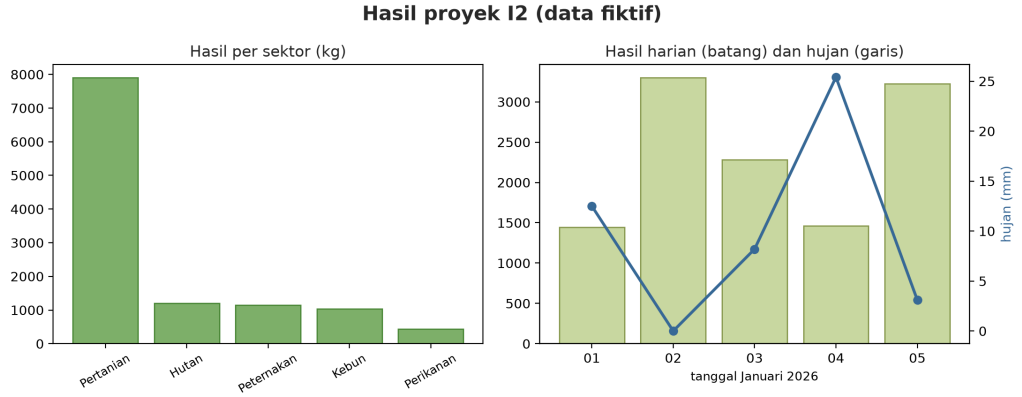
```
# Bab 25 - Proyek bersambung I2, langkah akhir: dari basis data ke grafik
from pathlib import Path
import matplotlib
matplotlib.use("Agg")
import matplotlib.pyplot as plt
import pandas as pd
from sqlalchemy import create_engine

tmp = Path(__file__).parent / "_tmp"
tmp.mkdir(exist_ok=True)
berkas = tmp / "penutup_i2.db"
berkas.unlink(missing_ok=True)
```

```
engine = create_engine(f"sqlite:///{berkas.as_posix()}")

# Langkah 1-4 (Bab 21-24) diringkas: data fiktif disimpan ke basis data
panen = pd.DataFrame(
    [("2026-01-01", "Hutan", "jati", 1200.0), ("2026-01-01", "Kebun", "kopi", 245.5),
       ("2026-01-02", "Kebun", "kakao", 198.5), ("2026-01-02", "Pertanian", "padi",
3100.0),
    ("2026-01-03", "Pertanian", "jagung", 1850.0), ("2026-01-03", "Perikanan", "lele",
432.4),
       ("2026-01-04", "Peternakan", "sapi", 1151.5), ("2026-01-04", "Kebun", "kopi",
310.0),
       ("2026-01-05", "Pertanian", "padi", 2950.0), ("2026-01-05", "Kebun", "kakao",
276.0)],
    columns=["tanggal", "sektor", "komoditas", "hasil_kg"])
tanggal = ["2026-01-01", "2026-01-02", "2026-01-03", "2026-01-04", "2026-01-05"]
cuaca = pd.DataFrame({"tanggal": tanggal, "hujan_mm": [12.5, 0.0, 8.2, 25.4, 3.1]})
daftar = {"padi": [6200, 6250, 6180, 6300, 6270], "jagung": [5100, 5150, 5200, 5120,
5180],
          "kopi": [58000, 58500, 59000, 57500, 58200], "kakao": [62000, 61500, 62500,
63000, 62800]}
harga = pd.DataFrame([(t, k, h) for k, v in daftar.items() for t, h in zip(tanggal, v)],
                     columns=["tanggal", "komoditas", "harga_per_kg"])
panen.to_sql("panen", engine, index=False)
cuaca.to_sql("cuaca", engine, index=False)
harga.to_sql("harga", engine, index=False)

# Langkah 5 (bab ini): baca kembali dari basis data, gabungkan, analisis, gambar
df = pd.read_sql_query("""
     SELECT p.tanggal, p.sektor, p.hasil_kg, c.hujan_mm, p.hasil_kg * h.harga_per_kg
AS pendapatan_rp
    FROM panen p
    LEFT JOIN cuaca c ON c.tanggal = p.tanggal
    LEFT JOIN harga h ON h.tanggal = p.tanggal AND h.komoditas = p.komoditas
""", engine)
engine.dispose()

per_sektor = df.groupby("sektor")["hasil_kg"].sum().sort_values(ascending=False)
per_hari       =       df.groupby("tanggal").agg(hasil_kg=("hasil_kg",       "sum"),
hujan_mm=("hujan_mm", "first"))
print(per_sektor)
print(per_hari)
print("Korelasi hujan dan total hasil harian (hanya 5 hari, jangan dipercaya):",
      round(per_hari["hujan_mm"].corr(per_hari["hasil_kg"]), 2))
print("Pendapatan tercatat (Rp):", f"{df['pendapatan_rp'].sum():,.0f}")

fig, (kiri, kanan) = plt.subplots(1, 2, figsize=(10, 4))
per_sektor.plot.bar(ax=kiri, color="tab:green", rot=30)
kiri.set_title("Hasil per sektor (kg)")
kanan.bar(per_hari.index.str[-2:], per_hari["hasil_kg"], color="tab:olive")
kanan2 = kanan.twinx()
kanan2.plot(per_hari.index.str[-2:],     per_hari["hujan_mm"],     color="tab:blue",
marker="o")
kanan.set_title("Hasil harian (batang) dan hujan (garis)")
fig.tight_layout()
fig.savefig(tmp / "proyek_i2.png", dpi=100)
print("Grafik tersimpan:", (tmp / "proyek_i2.png").exists())
```

```
sektor
Pertanian     7900.0
Hutan         1200.0
```

```
Peternakan    1151.5
Kebun         1030.0
Perikanan      432.4
Name: hasil_kg, dtype: float64
            hasil_kg  hujan_mm
tanggal
2026-01-01    1445.5      12.5
2026-01-02    3298.5       0.0
2026-01-03    2282.4       8.2
2026-01-04    1461.5      25.4
2026-01-05    3226.0       3.1
Korelasi hujan dan total hasil harian (hanya 5 hari, jangan dipercaya): -0.87
Pendapatan tercatat (Rp): 109,096,050
Grafik tersimpan: True
```

Hasilnya, Pertanian terbesar (7900,0 kg), disusul Hutan (1200,0), Peternakan (1151,5), Kebun (1030,0), dan Perikanan (432,4). Pendapatan tercatat Rp 109.096.050 adalah jumlah Kebun dan Pertanian saja (61.604.550 + 47.491.500), karena hanya itu yang punya harga. Korelasi hujan dan total hasil harian **-0,87** terlihat kuat, tetapi **hanya dari 5 hari**; itu kebetulan yang menarik, **bukan** bukti bahwa hujan menurunkan panen. Mengukur seberapa yakin kita pada angka seperti itulah pekerjaan statistika (Seri I3).



Gambar 0.1: Hasil Proyek I2: hasil per sektor (kiri) dan hasil harian dengan hujan (kanan). Data fiktif.

Latihan

1. Di `01_pandas_lanjut.py`, ubah batas kategori hujan menjadi 0-10 (kering), 10-20 (sedang), lebih dari 20 (lebat). Bagaimana jumlah hari tiap kategori berubah?
2. Di `02_deret_waktu.py`, ganti `resample("W")` dengan `resample("D")` lalu `rolling(window=3)`. Apa bedanya dengan rolling 7 hari?
3. Di `06_sklearn_regresi.py`, ubah `test_size` menjadi 0,5. Apakah R2 berubah? Mengapa?
4. **Tantangan:** di `07_sklearn_klasifikasi.py`, ubah `max_depth` menjadi 1, lalu 6. Bandingkan akurasi latih dan uji. Model mana paling jujur?

Ringkasan Bab

- pandas lanjut: kolom turunan, `pd.cut` untuk kategori, `groupby().agg()`, indeks tanggal dengan `resample` dan `rolling`.
- numpy: `percentile`, `corrcoef`, `where`, `cumsum`, dan **broadcasting** tanpa perulangan.
- matplotlib: figure dan axes; `plot`, `bar`, `scatter`; beri judul, sumbu, dan **satuan**.
- scikit-learn: bagi data latih/uji, `fit`, `predict`, nilai R2 atau akurasi; **bandingkan dengan pembanding sederhana**.
- Hasil dari sampel kecil hanya perkiraan; korelasi bukan sebab-akibat.
- **Jupyter Notebook** menggabungkan teks, kode, dan hasil; **Anaconda** membawa Jupyter dan pustaka ilmu data (tidak dijalankan di sini).

Penutup Seri I2

Selamat! Anda sudah menempuh perjalanan data lengkap: **mengambil** data lewat API, **membersihkannya** dengan regex, **mengolahnya** di numpy dan pandas, **menyimpannya** di basis data, dan **menggambar** serta **menduganya** dengan matplotlib dan scikit-learn. Kelima sektor, cuaca, dan harga kini hidup dalam satu alur kerja yang bisa Anda ulang.

Tetapi data yang sudah rapi segera memunculkan pertanyaan yang lebih sulit: **apa artinya?** Apakah beda rata-rata dua plot itu nyata atau kebetulan? Seberapa yakin kita bahwa hujan memengaruhi panen? Seri I3 menjawabnya dengan **statistik** dan pemodelan.

Selanjutnya, di Bab 26: statistika dengan Python, dari rata-rata dan sebaran sampai uji hipotesis dan cara menafsirkan p-value dengan benar.

Bab 26 — Statistika dengan Python: Dari Angka Menjadi Keputusan

Seri I3 — Statistik dan Pemodelan

Di bab ini, Anda akan:

- Merangkum data dengan statistika deskriptif (rata-rata, median, sebaran)
- Menghitung peluang dengan distribusi normal, binomial, dan Poisson
- Membuat selang kepercayaan dari sampel
- Menjalankan uji hipotesis: uji t, chi-square, ANOVA, dan cadangan nonparametriknya
- Menafsirkan p-value dengan benar, membaca ukuran efek, dan membedakan korelasi dari sebab-akibat

Semua skrip bab ini ada di folder `kode/bab26/`. Di versi web, Anda juga bisa mengunduh semuanya sekaligus (zip).

Di Seri I2, Anda sudah mengambil, membersihkan, dan menyimpan data. Data yang rapi belum menjawab pertanyaan yang sebenarnya: *"Apakah pohon di plot B memang tumbuh lebih cepat, atau itu kebetulan saja?"* Menjawabnya perlu **statistika**, yaitu cara berpikir yang jujur tentang angka yang bergoyang. Bayangkan Anda mencicipi satu sendok sup untuk menilai sepanci: Anda tidak perlu menghabiskan panci, asalkan sendoknya diambil setelah diaduk. Itulah inti bab ini: menilai yang besar dari contoh kecil, sambil tahu seberapa yakin kita boleh merasa.

> **i** INGAT
>
> Semua data di bab ini **tiruan (fiktif)** buatan skrip dengan benih tetap (`default_rng(42)`), jadi angkanya sama setiap dijalankan. Angka-angka itu bukan hasil penelitian sungguhan dan tidak boleh dipakai sebagai anjuran budidaya.

Pustaka yang dipakai sudah dikenalkan di Seri I2 (`numpy`, `pandas`, `matplotlib`) ditambah tiga yang baru: **scipy** (`scipy.stats`: distribusi dan uji klasik), **statsmodels** (model dan tabel ANOVA bergaya laporan), dan **pingouin** (uji statistik dengan tabel hasil yang ringkas, termasuk ukuran efek). Untuk gambar, kita pakai **seaborn**.

Statistika deskriptif: merangkum tanpa menipu

Anda mengukur pertumbuhan diameter 30 pohon di plot A (cm per tahun). Memandangi 30 angka tidak membantu, jadi kita ringkas.

```
# Bab 26 - Statistika deskriptif: merangkum data pohon
import numpy as np
import pandas as pd

rng = np.random.default_rng(42)
# Data tiruan (fiktif): pertumbuhan diameter (cm per tahun) 30 pohon di plot A
pertumbuhan = rng.normal(loc=1.8, scale=0.6, size=30).round(2)
pertumbuhan[0] = 4.9  # satu pohon luar biasa cepat (pencilan)

s = pd.Series(pertumbuhan, name="pertumbuhan_cm")
print("Jumlah data      :", s.count())
print("Rata-rata        :", round(s.mean(), 3))
print("Median           :", round(s.median(), 3))
print("Simpangan baku   :", round(s.std(), 3))
```

```
print("Kuartil 1 dan 3  :", s.quantile(0.25).round(3), s.quantile(0.75).round(3))
print("Min dan maks     :", s.min(), s.max())
print("Kemencengan      :", round(s.skew(), 3))

tanpa = s.drop(0)
print("Rata-rata tanpa pencilan:", round(tanpa.mean(), 3))
print("Median tanpa pencilan   :", round(tanpa.median(), 3))
print()
print(s.describe().round(3))
```

```
Jumlah data      : 30
Rata-rata        : 1.908
Median           : 1.86
Simpangan baku   : 0.732
Kuartil 1 dan 3  : 1.552 2.218
Min dan maks     : 0.63 4.9
Kemencengan      : 2.198
Rata-rata tanpa pencilan: 1.804
Median tanpa pencilan   : 1.84

count    30.000
mean      1.908
std       0.732
min       0.630
25%       1.552
50%       1.860
75%       2.218
max       4.900
Name: pertumbuhan_cm, dtype: float64
```

- **Rata-rata** (*mean*) = jumlah dibagi banyaknya. **Median** = nilai di tengah bila diurutkan. Keduanya "angka tengah", tetapi berperilaku berbeda.
- **Simpangan baku** (*standard deviation*) = ukuran kasar seberapa jauh data biasanya menyebar dari rata-ratanya. **Kuartil** membagi data urut menjadi empat bagian sama banyak.
- Satu pohon "ajaib" bernilai 4,9 menarik rata-rata ke 1,908, sedangkan tanpa pohon itu rata-rata 1,804. **Median hampir tidak bergeser** (1,86 menjadi 1,84). Karena itu bila data punya pencilan (*outlier*) atau miring (kemencengan 2,198, jauh dari 0), median sering lebih jujur daripada rata-rata.
- `describe()` memberi ringkasan lima angka sekaligus.

> TIPS
>
> Selalu **gambar dulu** sebelum menghitung. Satu histogram atau *boxplot* menunjukkan pencilan dan kemiringan yang tidak tampak dalam deretan angka. Bagian seaborn di bawah memberi caranya.

Peluang dan distribusi

Distribusi adalah "peta" seberapa mungkin tiap nilai muncul. `scipy.stats` menyediakan banyak distribusi dengan metode yang seragam: `cdf` (peluang di bawah suatu nilai), `sf` (peluang di atasnya), `pmf` (peluang tepat satu nilai, untuk data cacah), dan `ppf` (kebalikan `cdf`: nilai untuk peluang tertentu).

```
# Bab 26 - Peluang dan distribusi dengan scipy.stats
from scipy import stats

# Normal: diameter pohon rata-rata 30 cm, simpangan baku 5 cm
d = stats.norm(loc=30, scale=5)
print("P(diameter < 25)        =", round(d.cdf(25), 4))
print("P(diameter > 40)        =", round(d.sf(40), 4))
print("P(25 < diameter < 35)   =", round(d.cdf(35) - d.cdf(25), 4))
```

```
print("Diameter persentil ke-90 =", round(d.ppf(0.90), 2))

# Binomial: 20 biji disemai, peluang tumbuh 0,8 per biji
b = stats.binom(n=20, p=0.8)
print()
print("Rata-rata biji tumbuh   =", b.mean())
print("P(tepat 16 tumbuh)      =", round(b.pmf(16), 4))
print("P(paling sedikit 18)    =", round(b.sf(17), 4))

# Poisson: rata-rata 3 pohon tumbang per hektar per musim
p = stats.poisson(mu=3)
print()
print("P(tidak ada yang tumbang) =", round(p.pmf(0), 4))
print("P(lebih dari 5 tumbang)   =", round(p.sf(5), 4))
```

```
P(diameter < 25)        = 0.1587
P(diameter > 40)        = 0.0228
P(25 < diameter < 35)   = 0.6827
Diameter persentil ke-90 = 36.41

Rata-rata biji tumbuh   = 16.0
P(tepat 16 tumbuh)      = 0.2182
P(paling sedikit 18)    = 0.2061

P(tidak ada yang tumbang) = 0.0498
P(lebih dari 5 tumbang)   = 0.0839
```

- **Normal** (lonceng): data ukuran sering mendekati ini. Diameter pohon rata-rata 30 cm dengan simpangan baku 5 cm: sekitar 68% pohon (0,6827) ada dalam satu simpangan baku dari rata-rata (25 sampai 35 cm), dan hanya 2,28% yang lebih dari 40 cm.
- **Binomial**: jumlah "berhasil" dari beberapa percobaan ya/tidak. 20 biji dengan peluang tumbuh 0,8 rata-rata menghasilkan 16 bibit; peluang tepat 16 hanya 0,2182, bukan 1. Peluang minimal 18 tumbuh sekitar 21%.
- **Poisson**: banyaknya kejadian langka dalam satu wilayah atau waktu (pohon tumbang per hektar). Dengan rata-rata 3, peluang tidak ada yang tumbang hanya 4,98%.

Sampling dan selang kepercayaan

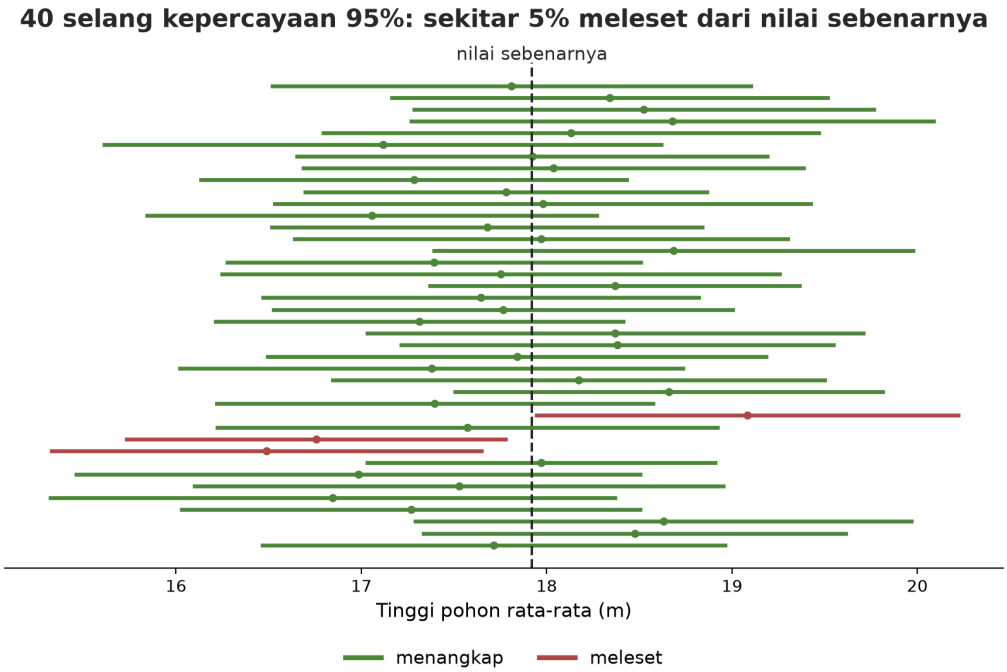
Kita hampir tidak pernah mengukur **semua** pohon (populasi). Kita mengambil **sampel** lalu menaksir. Taksiran dari sampel pasti agak meleset, dan **galat baku** (*standard error*) mengukur seberapa besar meleset itu: simpangan baku dibagi akar jumlah data. **Selang kepercayaan** 95% adalah rentang di sekitar taksiran yang dibuat dengan resep yang, bila diulang berkali-kali, menangkap nilai sebenarnya sekitar 95% dari waktu.

```
# Bab 26 - Sampling dan selang kepercayaan
import numpy as np
from scipy import stats

rng = np.random.default_rng(42)
# "Populasi" fiktif: tinggi 5000 pohon (m)
populasi = rng.normal(loc=18.0, scale=4.0, size=5000)
print("Rata-rata populasi (nilai sebenarnya):", round(populasi.mean(), 3))

# Ambil satu sampel acak 40 pohon
sampel = rng.choice(populasi, size=40, replace=False)
rata = sampel.mean()
galat_baku = stats.sem(sampel)  # simpangan baku / akar n
print("Rata-rata sampel      :", round(rata, 3))
print("Galat baku            :", round(galat_baku, 3))
```

```
# Selang kepercayaan 95% memakai distribusi t
bawah, atas = stats.t.interval(0.95, df=len(sampel) - 1, loc=rata, scale=galat_baku)
print("Selang 95%            :", round(bawah, 2), "sampai", round(atas, 2))

# Ulangi 1000 kali: berapa kali selang menangkap nilai sebenarnya?
tangkap = 0
for _ in range(1000):
    s = rng.choice(populasi, size=40, replace=False)
    lo, hi = stats.t.interval(0.95, df=39, loc=s.mean(), scale=stats.sem(s))
    tangkap += lo <= populasi.mean() <= hi
print("Selang yang menangkap nilai sebenarnya:", tangkap, "dari 1000")
```

```
Rata-rata populasi (nilai sebenarnya): 17.92
Rata-rata sampel      : 17.714
Galat baku            : 0.617
Selang 95%            : 16.47 sampai 18.96
Selang yang menangkap nilai sebenarnya: 961 dari 1000
```

Nilai sebenarnya (rata-rata populasi) 17,92 m. Sampel kita menaksir 17,714 dengan selang 16,47 sampai 18,96, yang memang menangkap 17,92. Lalu kita ulangi 1000 kali: 961 selang menangkap nilai sebenarnya, dekat dengan janji 95%.



Gambar 0.1: Setiap garis adalah satu selang 95% dari satu sampel. Sekitar 5% (merah) meleset dari nilai sebenarnya; kita tidak tahu yang mana saat bekerja di lapangan.

> ⚠ AWAS
>
> Selang 95% **bukan** berarti "ada peluang 95% nilai sebenarnya berada di dalam selang ini". Nilai sebenarnya tetap (tidak acak); yang acak adalah selangnya. Yang benar: *resep* ini menangkap nilai sebenarnya di 95% dari sampel yang mungkin kita ambil. Lebih banyak data membuat selang lebih sempit.

Uji hipotesis: dua kelompok

Sekarang pertanyaan awal bab ini. Plot A dan B dibandingkan dengan **uji hipotesis**:

1. **Hipotesis nol** (H0): tidak ada beda sebenarnya; selisih yang tampak hanya kebetulan.
2. Hitung seberapa langka data seperti ini bila H0 benar. Ukurannya **p-value**.
3. Bila p-value kecil (kebiasaan: di bawah 0,05, disebut **taraf signifikansi**), kita menolak H0.

```
# Bab 26 - Uji t: apakah pertumbuhan plot A beda dengan plot B?
import numpy as np
import pingouin as pg
from scipy import stats

rng = np.random.default_rng(42)
plot_a = rng.normal(1.8, 0.6, 30).round(2)   # cm per tahun (fiktif)
plot_b = rng.normal(2.2, 0.7, 30).round(2)

print("Rata-rata A:", plot_a.mean().round(3), " Rata-rata B:", plot_b.mean().round(3))

# Uji t Welch (tidak mengandaikan ragam sama)
hasil = stats.ttest_ind(plot_a, plot_b, equal_var=False)
print("t =", round(hasil.statistic, 3), " p =", round(hasil.pvalue, 4))

# pingouin memberi tabel lengkap, termasuk ukuran efek (Cohen d)
tabel = pg.ttest(plot_a, plot_b, correction=True)
print(tabel[["T", "dof", "p_val", "cohen_d", "CI95"]].round(4).to_string())

# Uji berpasangan: 12 pohon yang sama diukur sebelum dan sesudah pemupukan
sebelum = rng.normal(1.5, 0.4, 12)
sesudah = sebelum + rng.normal(0.3, 0.25, 12)
pasangan = stats.ttest_rel(sesudah, sebelum)
print("Berpasangan: t =", round(pasangan.statistic, 3), " p =", format(pasangan.pvalue,
".1e"))
```

```
Rata-rata A: 1.81  Rata-rata B: 2.28
t = -3.512  p = 0.0009
             T     dof   p_val  cohen_d           CI95
T_test -3.5121  56.012  0.0009   0.9068  [-0.74, -0.2]
Berpasangan: t = 7.457  p = 1.3e-05
```

- Rata-rata A 1,81 dan B 2,28. **Uji t Welch** menghasilkan t = −3,512 dan p = 0,0009. Tanda minus hanya karena A dikurangi B. Welch tidak mengandaikan ragam kedua plot sama, jadi lebih aman sebagai pilihan awal.
- `pingouin.ttest` memberi tabel yang sama ditambah **Cohen d** = 0,9068 dan selang kepercayaan untuk **selisih** rata-rata (-0,74 sampai −0,2). Selang itu tidak memuat 0, sejalan dengan p kecil.
- **Uji berpasangan** (`ttest_rel`) dipakai bila pohon yang sama diukur dua kali (sebelum dan sesudah pemupukan). Ia lebih peka karena membandingkan tiap pohon dengan dirinya sendiri.

> **i** INGAT
>
> **p-value bukan peluang bahwa hipotesis Anda benar.** p = 0,0009 berarti: *bila* tidak ada beda sebenarnya, data sejauh ini hanya muncul sekitar 9 kali dalam 10.000 pengambilan. Ia tidak mengatakan seberapa **besar** bedanya atau seberapa **penting** bedanya. Itu tugas ukuran efek.

Ukuran efek menjawab "seberapa besar". Cohen d adalah selisih rata-rata dibagi simpangan baku gabungan. Pedoman kasar Cohen: 0,2 kecil, 0,5 sedang, 0,8 besar. Nilai 0,9 di atas tergolong besar.

Chi-square: dua kategori

Uji t untuk angka; untuk **hitungan per kategori** dipakai **chi-square**. Apakah jenis pohon berhubungan dengan sehat/sakit?

```
# Bab 26 - Chi-square: apakah jenis pohon berhubungan dengan kesehatan?
import pandas as pd
from scipy import stats

# Tabel kontingensi fiktif (jumlah pohon)
tabel = pd.DataFrame(
    {"sehat": [42, 35, 28], "sakit": [8, 15, 22]},
    index=["Jati", "Meranti", "Mahoni"],
)
print(tabel)

chi2, p, dof, harapan = stats.chi2_contingency(tabel)
print()
print("chi2 =", round(chi2, 3), " derajat bebas =", dof, " p =", round(p, 4))
print("Nilai harapan jika tidak berhubungan:")
print(pd.DataFrame(harapan, index=tabel.index, columns=tabel.columns).round(1))

# Ukuran efek: Cramer's V
n = tabel.values.sum()
v = (chi2 / (n * (min(tabel.shape) - 1))) ** 0.5
print("Cramer V =", round(v, 3))
```

```
         sehat  sakit
Jati        42      8
Meranti     35     15
Mahoni      28     22

chi2 = 9.333  derajat bebas = 2  p = 0.0094
Nilai harapan jika tidak berhubungan:
         sehat  sakit
Jati      35.0   15.0
Meranti   35.0   15.0
Mahoni    35.0   15.0
Cramer V = 0.249
```

Bila jenis dan kesehatan tidak berhubungan, tiap jenis (masing-masing 50 pohon) diharapkan 35 sehat dan 15 sakit. Yang teramati menyimpang: Jati lebih sehat (42), Mahoni lebih sakit (22). Chi-square = 9,333 dengan 2 derajat bebas (*degrees of freedom*) memberi p = 0,0094. Ukuran efeknya **Cramer V** = 0,249, hubungan yang nyata tetapi bukan yang sangat kuat.

ANOVA: tiga kelompok atau lebih

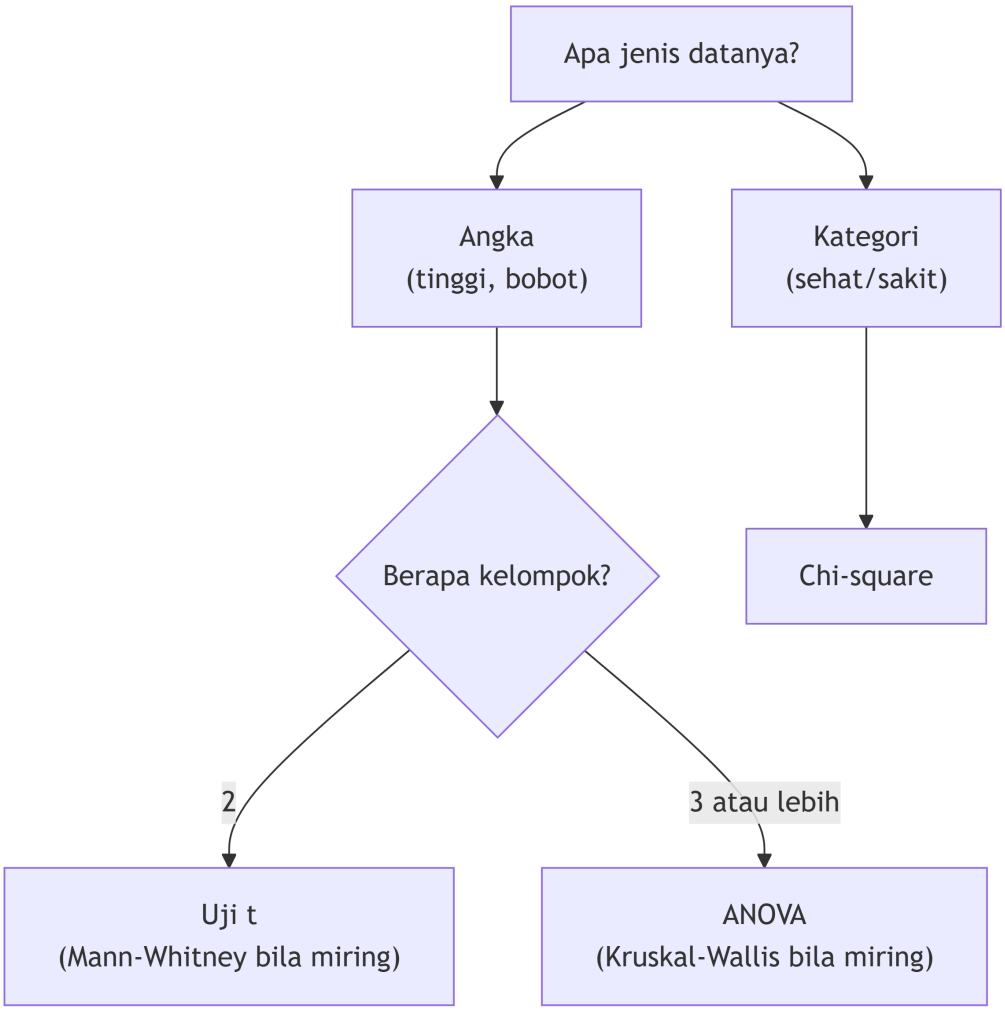
Membandingkan tiga varietas dengan tiga uji t berpasangan menaikkan risiko salah (lihat jebakan di bawah). **ANOVA** menguji semua kelompok sekaligus: "adakah paling tidak satu yang beda?" Ia mengandaikan data tiap kelompok mendekati normal dan ragamnya mirip, jadi diperiksa dulu dengan uji Shapiro dan Levene.

```
# Bab 26 - ANOVA untuk tiga varietas, dan uji nonparametrik sebagai cadangan
import numpy as np
import pandas as pd
import pingouin as pg
from scipy import stats

rng = np.random.default_rng(42)
data = pd.DataFrame({
```

```
    "varietas": np.repeat(["V1", "V2", "V3"], 25),
    "tinggi_cm": np.concatenate([
        rng.normal(120, 12, 25), rng.normal(128, 12, 25), rng.normal(137, 12, 25)
    ]).round(1),
})
print(data.groupby("varietas")["tinggi_cm"].agg(["mean", "std"]).round(2))

kelompok = [g["tinggi_cm"].values for _, g in data.groupby("varietas")]
print()
print("Shapiro (normal?) p per kelompok:", [round(float(stats.shapiro(k).pvalue), 3)
for k in kelompok])
print("Levene (ragam sama?) p          :", round(stats.levene(*kelompok).pvalue, 3))

anova = pg.anova(data=data, dv="tinggi_cm", between="varietas", detailed=False)
print()
print("F =", round(anova.loc[0, "F"], 2), " p =", format(anova.loc[0, "p_unc"], ".2e"),
" eta-kuadrat parsial =", round(anova.loc[0, "np2"], 3))

print()
print(pg.pairwise_tukey(data=data,  dv="tinggi_cm",  between="varietas")[["A",  "B",
"diff", "p_tukey"]].round(4).to_string())

# Cadangan nonparametrik (tidak mengandaikan sebaran normal)
print()
print("Kruskal-Wallis p =", format(stats.kruskal(*kelompok).pvalue, ".2e"))
print("Mann-Whitney   V1   vs   V3   p   =",   round(stats.mannwhitneyu(kelompok[0],
kelompok[2]).pvalue, 5))
```

```
            mean   std
varietas
V1        119.58  9.97
V2        130.61  8.32
V3        135.45  9.94

Shapiro (normal?) p per kelompok: [0.572, 0.168, 0.851]
Levene (ragam sama?) p          : 0.433

F = 18.55  p = 3.17e-07  eta-kuadrat parsial = 0.34

    A   B    diff  p_tukey
0  V1  V2 -11.036   0.0003
1  V1  V3 -15.872   0.0000
2  V2  V3  -4.836   0.1734

Kruskal-Wallis p = 4.19e-06
Mann-Whitney V1 vs V3 p = 1e-05
```

- Rata-rata tinggi V1, V2, V3: 119,58, 130,61, 135,45 cm. Uji Shapiro (p semua di atas 0,05) dan Levene (p = 0,433) tidak menunjukkan pelanggaran asumsi.
- ANOVA: F = 18,55, p = 3,17e-07 (artinya 0,000000317). Eta-kuadrat parsial 0,34: sekitar 34% keragaman tinggi berkaitan dengan varietas.
- ANOVA hanya bilang "ada yang beda". **Uji lanjutan Tukey** menunjukkan yang mana: V1-V2 (p = 0,0003) dan V1-V3 (0,0000, artinya di bawah 0,00005) beda, sedangkan V2-V3 (0,1734) belum terbukti beda.
- **Uji nonparametrik** tidak mengandaikan sebaran normal; ia bekerja pada peringkat data. Kruskal-Wallis menggantikan ANOVA, Mann-Whitney menggantikan uji t. Di sini keduanya sepakat dengan ANOVA. Pakai ini bila data sangat miring, berupa peringkat, atau sampelnya sangat kecil.



Memilih uji hipotesis: mulai dari jenis data, lalu jumlah kelompok.

Dua jebakan p-value

```
# Bab 26 - Dua jebakan p-value: banyak uji dan sampel raksasa
import numpy as np
from scipy import stats

rng = np.random.default_rng(42)

# Jebakan 1: 20 uji pada data dari populasi yang sama (tidak ada beda sebenarnya)
signifikan = 0
for i in range(20):
    a = rng.normal(0, 1, 30)
    b = rng.normal(0, 1, 30)
    p = stats.ttest_ind(a, b).pvalue
    if p < 0.05:
        signifikan += 1
       print(f"  uji ke-{i + 1}: p = {p:.4f}  <- 'signifikan' padahal tidak ada beda")
print("Uji 'signifikan' dari 20:", signifikan)
print("Setelah koreksi Bonferroni, ambang menjadi", 0.05 / 20)
```

```
# Jebakan 2: beda sangat kecil, sampel sangat besar
a = rng.normal(100.0, 15, 200_000)
b = rng.normal(100.3, 15, 200_000)
hasil = stats.ttest_ind(a, b)
d = (b.mean() - a.mean()) / 15
print()
print("Beda rata-rata:", round(b.mean() - a.mean(), 3), " p =", format(hasil.pvalue,
".1e"))
print("Ukuran efek (d) :", round(d, 3), "-> sangat kecil, walau p sangat kecil")
```

```
  uji ke-15: p = 0.0327  <- 'signifikan' padahal tidak ada beda
Uji 'signifikan' dari 20: 1
Setelah koreksi Bonferroni, ambang menjadi 0.0025

Beda rata-rata: 0.323  p = 9.7e-12
Ukuran efek (d) : 0.022 -> sangat kecil, walau p sangat kecil
```

- **Jebakan 1, terlalu banyak uji.** Kita menjalankan 20 uji pada data yang sebenarnya dari populasi yang sama. Hasilnya satu uji (ke-15, p = 0,0327) tampak "signifikan". Dengan ambang 0,05, kira-kira satu dari 20 uji tak berbeda *pasti* tampak signifikan. Bila Anda menguji banyak hal lalu hanya melaporkan yang "signifikan", Anda sedang memanen kebetulan. Obatnya: tetapkan hipotesis lebih dulu, dan pakai **koreksi** seperti Bonferroni (ambang dibagi jumlah uji, di sini 0,0025).
- **Jebakan 2, sampel raksasa.** Dengan 200.000 data per kelompok, selisih 0,323 poin pada skala simpangan baku 15 (d = 0,022) menghasilkan p = 9,7e-12. Signifikan secara statistik, tetapi nyaris tidak berarti di dunia nyata. **Signifikan bukan berarti penting.** Selalu laporkan ukuran efek dan selang kepercayaan di samping p-value.

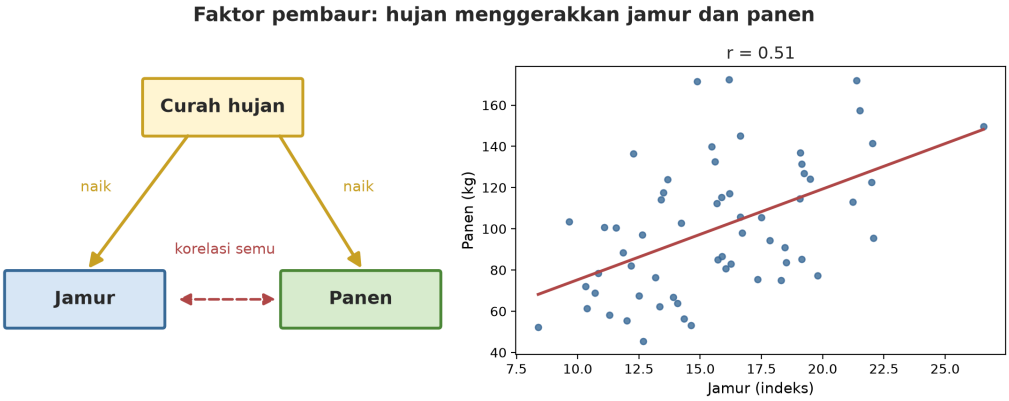
Korelasi bukan sebab-akibat

Korelasi mengukur seberapa kompak dua angka naik-turun bersama (dari −1 sampai 1). Pearson untuk hubungan garis lurus; Spearman memakai peringkat sehingga tahan pencilan.

```
# Bab 26 - Korelasi bukan sebab-akibat: faktor tersembunyi
import numpy as np
import pandas as pd
from scipy import stats
import pingouin as pg

rng = np.random.default_rng(42)
n = 60
hujan = rng.normal(200, 50, n)                    # curah hujan (mm), faktor tersembunyi
jamur = 0.08 * hujan + rng.normal(0, 3, n)              # jamur tumbuh lebih banyak
saat hujan
panen = 0.5 * hujan + rng.normal(0, 25, n)           # panen juga naik saat hujan
df = pd.DataFrame({"hujan_mm": hujan, "jamur": jamur, "panen_kg": panen})

r, p = stats.pearsonr(df["jamur"], df["panen_kg"])
print("Korelasi jamur vs panen (Pearson):", round(r, 3), " p =", format(p, ".1e"))
rho, p2 = stats.spearmanr(df["jamur"], df["panen_kg"])
print("Korelasi peringkat (Spearman)     :", round(rho, 3))

# Korelasi parsial: buang pengaruh hujan
parsial = pg.partial_corr(data=df, x="jamur", y="panen_kg", covar="hujan_mm")
print()
print("Korelasi jamur vs panen setelah hujan dikendalikan:")
print(parsial[["n", "r", "p_val"]].round(3).to_string())
```

```
Korelasi jamur vs panen (Pearson): 0.513  p = 2.8e-05
Korelasi peringkat (Spearman)     : 0.504

Korelasi jamur vs panen setelah hujan dikendalikan:
          n      r  p_val
pearson  60  0.057  0.667
```

Jamur dan panen berkorelasi 0,513 (p = 2,8e-05): hubungan tampak kuat. Apakah memberantas jamur menurunkan panen? Tidak. Dalam data ini, **hujan** menaikkan keduanya (jamur menyukai lembap, tanaman menyukai air). Setelah pengaruh hujan dikendalikan lewat **korelasi parsial**, hubungan jamur-panen tinggal 0,057 dengan p = 0,667, alias hilang. Hujan di sini disebut **faktor pembaur** (*confounder*).



Gambar 0.2: Faktor pembaur: hujan menggerakkan jamur dan panen sehingga keduanya tampak berkorelasi.

> ⚠ AWAS
>
> Sebab-akibat hanya bisa disimpulkan dari **percobaan terkontrol** (misalnya plot diberi perlakuan secara acak) atau dari analisis yang sangat hati-hati. Data pengamatan biasa hanya menunjukkan hubungan. Waspadai kalimat "X menyebabkan Y" dari sekadar korelasi.

Visualisasi statistik dengan seaborn

```python
# Bab 26 - Visualisasi statistik dengan seaborn
from pathlib import Path

import matplotlib
matplotlib.use("Agg")
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns

rng = np.random.default_rng(42)
data = pd.DataFrame({
    "plot": np.repeat(["A", "B", "C"], 30),
    "pertumbuhan_cm": np.concatenate([
        rng.normal(1.8, 0.6, 30), rng.normal(2.3, 0.7, 30), rng.normal(2.0, 0.5, 30)
    ]),
})

keluar = Path(__file__).parent / "_tmp"
```

```
keluar.mkdir(exist_ok=True)

fig, sumbu = plt.subplots(1, 2, figsize=(9, 3.6))
sns.boxplot(data=data, x="plot", y="pertumbuhan_cm", ax=sumbu[0])
sns.stripplot(data=data,   x="plot",   y="pertumbuhan_cm",   color="black",   size=3,
ax=sumbu[0])
sns.histplot(data=data, x="pertumbuhan_cm", hue="plot", kde=True, ax=sumbu[1])
fig.tight_layout()
fig.savefig(keluar / "seaborn_plot.png", dpi=120)
print("Gambar tersimpan:", (keluar / "seaborn_plot.png").name)
print("Ukuran per plot:", data.groupby("plot").size().to_dict())
```

```
Gambar tersimpan: seaborn_plot.png
Ukuran per plot: {'A': 30, 'B': 30, 'C': 30}
```

Skrip menyimpan gambar ke folder `_tmp/` (tidak ditampilkan di buku). Buka dan lihat: *boxplot* menunjukkan median, kuartil, dan pencilan; titik-titik (`stripplot`) menunjukkan data mentahnya; histogram dengan garis `kde` menunjukkan bentuk sebaran. Menggabungkan ketiganya membuat Anda jarang tertipu oleh satu angka rata-rata.

Proyek Bersambung I3: uji beda pertumbuhan antar-plot dan antar-varietas

Tugas pertama proyek I3: pertumbuhan diameter pohon (cm per tahun) di tiga plot (A, B, C) dan dua varietas (V1, V2), 20 pohon per kombinasi. Semua angka fiktif dengan beda yang kita tanam sendiri (B lebih tinggi 0,5; V2 lebih tinggi 0,3), sehingga kita bisa memeriksa apakah uji menemukannya.

```
# Bab 26 - Proyek bersambung I3: uji beda pertumbuhan antar-plot dan antar-varietas
import numpy as np
import pandas as pd
import pingouin as pg
import statsmodels.api as sm
import statsmodels.formula.api as smf

rng = np.random.default_rng(42)
efek_plot = {"A": 0.0, "B": 0.5, "C": 0.2}      # fiktif
efek_varietas = {"V1": 0.0, "V2": 0.3}          # fiktif
baris = []
for p, ep in efek_plot.items():
    for v, ev in efek_varietas.items():
        for g in rng.normal(1.8 + ep + ev, 0.6, 20):
            baris.append({"plot": p, "varietas": v, "pertumbuhan_cm": round(g, 2)})
pohon = pd.DataFrame(baris)

print(pohon.pivot_table(index="plot",   columns="varietas",   values="pertumbuhan_cm",
aggfunc="mean").round(2))

model = smf.ols("pertumbuhan_cm ~ C(plot) + C(varietas)", data=pohon).fit()
anova = sm.stats.anova_lm(model, typ=2)
print()
print(anova.round(4))

print()
print("Beda antar-plot (Tukey):")
print(pg.pairwise_tukey(data=pohon,  dv="pertumbuhan_cm",  between="plot")[["A",  "B",
"diff", "p_tukey", "hedges"]].round(3).to_string())

v1 = pohon.query("varietas == 'V1'")["pertumbuhan_cm"]
v2 = pohon.query("varietas == 'V2'")["pertumbuhan_cm"]
```

```
uji = pg.ttest(v1, v2)
print()
print("V1 vs V2: p =", uji["p_val"].round(4).iloc[0], " Cohen d =",
uji["cohen_d"].round(2).iloc[0], " CI95 beda =", uji["CI95"].iloc[0])
```

```
varietas    V1    V2
plot
A         1.78  2.17
B         2.37  2.54
C         1.79  2.23

               sum_sq     df         F  PR(>F)
C(plot)        5.8334    2.0  13.1621  0.0000
C(varietas)    3.3534    1.0  15.1327  0.0002
Residual      25.7053  116.0      NaN     NaN

Beda antar-plot (Tukey):
   A  B   diff  p_tukey  hedges
0  A  B -0.485    0.000  -0.984
1  A  C -0.037    0.942  -0.069
2  B  C  0.448    0.000   0.922

V1 vs V2: p = 0.0006  Cohen d = 0.65  CI95 beda = [-0.52 -0.15]
```

- Tabel pertama menunjukkan rata-rata tiap kombinasi. Plot B paling tinggi di kedua varietas, V2 selalu di atas V1.
- `ols("pertumbuhan_cm ~ C(plot) + C(varietas)")` membuat model dengan dua faktor sekaligus; `anova_lm` merangkumnya. Plot: F = 13,16, varietas: F = 15,13, keduanya p di bawah 0,001 (tampil 0,0000 dan 0,0002 karena dibulatkan).
- Tukey antar-plot: A-B beda (selisih −0,485; ukuran efek Hedges −0,984), B-C beda, tetapi **A-C tidak** (p = 0,942, selisih −0,037). Uji menemukan beda yang kita tanam, dan tidak mengada-ada untuk pasangan yang memang mirip.
- V1 vs V2: p = 0,0006, Cohen d = 0,65 (sedang), selisih rata-rata ada di rentang −0,52 sampai −0,15 cm per tahun.

Proyek ini dilanjutkan di Bab 27 dengan regresi, dan seluruh hasilnya masuk ke laporan Quarto di Bab 29.

Latihan

1. Ubah `01_deskriptif.py`: ganti pencilan 4,9 menjadi 9,9. Mana yang bergeser lebih banyak, rata-rata atau median? Mengapa?
2. Dengan distribusi normal di `02_distribusi.py`, hitung peluang diameter pohon antara 20 dan 30 cm, dan diameter yang melampaui 95% pohon.
3. Di `03_sampling_ci.py`, ubah ukuran sampel dari 40 menjadi 160. Bagaimana lebar selang 95% berubah? (Petunjuk: galat baku turun kira-kira setengahnya.)
4. **Tantangan:** buat data untuk dua plot dengan rata-rata sama persis, ulang uji t 1000 kali dengan benih berbeda, dan hitung berapa persen uji menghasilkan p di bawah 0,05. Apakah mendekati 5%? Apa artinya?

Ringkasan Bab

- **Deskriptif**: rata-rata, median, simpangan baku, kuartil. Pencilan mengganggu rata-rata, tidak median.
- **Distribusi** di `scipy.stats`: `cdf`, `sf`, `pmf`, `ppf`; normal untuk ukuran, binomial untuk berhasil/gagal, Poisson untuk kejadian langka.
- **Selang kepercayaan** 95% = resep yang menangkap nilai sebenarnya di sekitar 95% sampel; bukan peluang untuk satu selang.
- **Uji hipotesis**: uji t (dua kelompok), chi-square (kategori), ANOVA (tiga atau lebih, dilanjutkan Tukey), dan cadangan nonparametrik.

- **p-value** bukan peluang hipotesis benar, tidak mengukur besar efek. Waspada banyak uji dan sampel raksasa; selalu laporkan **ukuran efek**.
- **Korelasi bukan sebab-akibat**: cari faktor pembaur dan kendalikan.

Selanjutnya, di Bab 27: regresi, cara membuat model yang menjelaskan dan meramalkan: pohon, ikan, ternak, panen, dan hujan.

Bab 27 — Regresi dan Pemodelan Kuantitatif: Membuat Model dari Data

Seri I3 — Statistik dan Pemodelan

Di bab ini, Anda akan:

- Membuat model regresi linear sederhana dan berganda dengan statsmodels, lalu memeriksa asumsinya
- Memakai regresi logistik (ya/tidak), GLM Poisson (cacah), dan regularisasi Ridge/Lasso
- Mencocokkan kurva non-linear, model efek campuran, dan peramalan deret waktu (ARIMA)
- Mengenal regresi spasial (GWR) dan tahu apa yang diuji dan tidak diuji di buku ini
- Menyimulasikan ketidakpastian (Monte Carlo), mengoptimalkan, dan menganalisis sensitivitas model sistem dinamis

Semua skrip bab ini ada di folder `kode/bab27/`. Di versi web, Anda juga bisa mengunduh semuanya sekaligus (zip).

Bab 26 menjawab "apakah dua kelompok berbeda?". Bab ini menjawab pertanyaan yang lebih berguna: *"Seberapa besar pengaruh X terhadap Y, dan apa ramalan Y bila X berubah?"* Itulah **regresi**: menarik garis (atau lengkung) terbaik lewat sekumpulan titik, lalu memakainya untuk menjelaskan dan meramal. Bayangkan Anda menimbang sapi tiap bulan dan mencatat pakannya; regresi menyimpulkan "tiap tambahan 1 kg pakan harian rata-rata menambah sekian kg bobot", lengkap dengan seberapa yakin Anda terhadap angka itu.

i INGAT

Semua data di bab ini **tiruan (fiktif)** buatan skrip dengan benih tetap, dan sengaja dibuat dari aturan yang kita ketahui supaya kita bisa memeriksa apakah model menemukannya kembali. Jangan pakai angkanya sebagai anjuran budidaya, harga, atau pakan.

Regresi linear sederhana: persamaan alometrik pohon

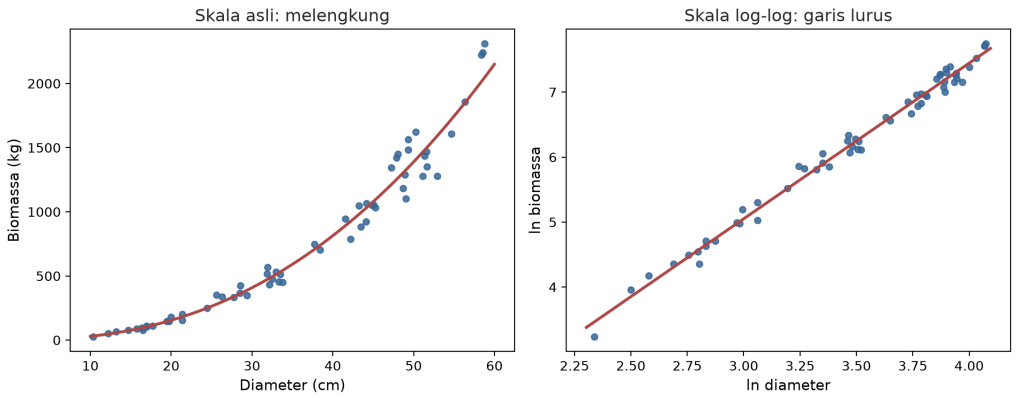
Menimbang biomassa pohon berarti menebangnya. Para rimbawan lebih suka mengukur **diameter** (mudah) lalu meramal biomassa lewat **persamaan alometrik**. Hubungannya melengkung (pohon dua kali lebih tebal jauh lebih dari dua kali lebih berat), tetapi bila kedua sisi di-*log*, hubungannya menjadi garis lurus: ln(biomassa) = a + b × ln(diameter).

```
# Bab 27 - Regresi linear sederhana: persamaan alometrik pohon
import numpy as np
import pandas as pd
import statsmodels.formula.api as smf

rng = np.random.default_rng(42)
n = 60
diameter = rng.uniform(10, 60, n)                                  # cm
biomassa = 0.12 * diameter ** 2.4 * np.exp(rng.normal(0, 0.15, n))  # kg (fiktif)
pohon = pd.DataFrame({"diameter_cm": diameter, "biomassa_kg": biomassa})
pohon["ln_d"] = np.log(pohon["diameter_cm"])
pohon["ln_b"] = np.log(pohon["biomassa_kg"])

# Hubungan asli melengkung; setelah dilog menjadi garis lurus
model = smf.ols("ln_b ~ ln_d", data=pohon).fit()
```

```
print(model.params.round(3))
print("R kuadrat       :", round(model.rsquared, 3))
print("Selang 95% ln_d :", model.conf_int().loc["ln_d"].round(3).tolist())
print("p-value ln_d    :", format(model.pvalues["ln_d"], ".1e"))

# Meramal biomassa pohon berdiameter 40 cm (kembalikan dari skala log)
a, b = model.params["Intercept"], model.params["ln_d"]
koreksi = np.exp(model.mse_resid / 2)        # koreksi bias hasil log
print()
print("Ramalan tanpa koreksi :", round(np.exp(a) * 40 ** b, 1), "kg")
print("Ramalan dengan koreksi:", round(np.exp(a) * 40 ** b * koreksi, 1), "kg")

# Bandingkan dengan garis lurus langsung pada skala asli
lurus = smf.ols("biomassa_kg ~ diameter_cm", data=pohon).fit()
print("R kuadrat garis lurus langsung:", round(lurus.rsquared, 3))
print("Ramalan garis lurus di 5 cm   :",
round(lurus.predict(pd.DataFrame({"diameter_cm": [5]}))[0], 1), "kg (ngawur: bisa
negatif)")
```

```
Intercept   -2.140
ln_d         2.397
dtype: float64
R kuadrat       : 0.99
Selang 95% ln_d : [2.333, 2.461]
p-value ln_d    : 1.9e-59

Ramalan tanpa koreksi : 813.5 kg
Ramalan dengan koreksi: 818.8 kg
R kuadrat garis lurus langsung: 0.908
Ramalan garis lurus di 5 cm   : -501.4 kg (ngawur: bisa negatif)
```

- Rumus `"ln_b ~ ln_d"` dibaca “ln_b dijelaskan oleh ln_d”. `smf.ols(...).fit()` mencari garis yang meminimalkan jumlah kuadrat selisih (metode **kuadrat terkecil**, *ordinary least squares*, OLS).
- Kemiringan b = 2,397 dengan selang 95% 2,333 sampai 2,461. Kita menanam 2,4 di data, dan model menemukannya kembali. Bila diameter naik 1%, biomassa naik sekitar 2,4%.
- **R kuadrat** = 0,99: sekitar 99% keragaman ln(biomassa) dijelaskan oleh ln(diameter). R kuadrat tinggi bukan jaminan model benar; ia hanya menunjukkan kecocokan dengan data ini.
- Meramal kembali ke kilogram perlu `np.exp`, dan karena rata-rata log tidak sama dengan log rata-rata, ada **koreksi bias** kecil: 813,5 kg tanpa koreksi, 818,8 kg dengan koreksi (faktor `exp(sisaan / 2)`).
- Garis lurus langsung pada skala asli hanya mencapai R kuadrat 0,908 dan meramal −501,4 kg untuk pohon 5 cm: **massa negatif**. Pilihan bentuk model itu penting.



Gambar 0.1: Persamaan alometrik: melengkung di skala asli (kiri), lurus setelah di-log (kanan).

Regresi berganda dan diagnostik

Bobot ikan dipengaruhi banyak hal sekaligus. **Regresi berganda** menaruh beberapa penjelas dalam satu model; tiap koefisien berarti "pengaruh variabel itu bila yang lain **ditahan tetap**".

```
# Bab 27 - Regresi berganda dan diagnostik: pertumbuhan ikan terhadap pakan
import numpy as np
import pandas as pd
import statsmodels.api as sm
import statsmodels.formula.api as smf
from scipy import stats
from statsmodels.stats.diagnostic import het_breuschpagan
from statsmodels.stats.outliers_influence import variance_inflation_factor

rng = np.random.default_rng(42)
n = 80
pakan = rng.uniform(20, 60, n)                          # gram pakan per ikan per hari
suhu = rng.normal(28, 1.5, n)                           # derajat Celsius
padat = rng.uniform(10, 40, n)                          # ekor per m3
bobot = 50 + 6.0 * pakan + 12 * (suhu - 28) - 2.5 * padat + rng.normal(0, 25, n)
ikan = pd.DataFrame({"bobot_g": bobot, "pakan_g": pakan, "suhu_c": suhu, "padat":
padat})

model = smf.ols("bobot_g ~ pakan_g + suhu_c + padat", data=ikan).fit()
print(model.summary().tables[1])
print("R kuadrat:", round(model.rsquared, 3), " R kuadrat disesuaikan:",
round(model.rsquared_adj, 3))

# Diagnostik 1: apakah sisaan (residual) berdistribusi normal?
print()
print("Shapiro sisaan p      :", round(stats.shapiro(model.resid).pvalue, 3))
# Diagnostik 2: apakah ragam sisaan sama di semua tingkat? (homoskedastisitas)
print("Breusch-Pagan p       :", round(het_breuschpagan(model.resid, model.model.exog)
[1], 3))

# Diagnostik 3: multikolinearitas (VIF). Tambahkan pakan "kembar" yang hampir sama
ikan["pakan_pelet_g"] = 0.9 * ikan["pakan_g"] + rng.normal(0, 1.5, n)
X = sm.add_constant(ikan[["pakan_g", "suhu_c", "padat", "pakan_pelet_g"]])
print()
for i, nama in enumerate(X.columns):
    if nama != "const":
        print(f"VIF {nama:14}: {variance_inflation_factor(X.values, i):6.1f}")
```

```
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept   -336.0406     61.612     -5.454      0.000    -458.751    -213.330
pakan_g        5.9493      0.282     21.133      0.000       5.389       6.510
suhu_c        13.8181      2.183      6.330      0.000       9.470      18.166
padat         -2.4164      0.325     -7.436      0.000      -3.064      -1.769
==============================================================================
R kuadrat: 0.888  R kuadrat disesuaikan: 0.883

Shapiro sisaan p      : 0.804
Breusch-Pagan p       : 0.746

VIF pakan_g       :   43.4
VIF suhu_c        :    1.0
VIF padat         :    1.0
VIF pakan_pelet_g :   43.4
```

- Tabel koefisien: tiap gram pakan menambah bobot 5,95 g; tiap derajat suhu menambah 13,8 g; tiap tambahan ekor per m3 mengurangi 2,42 g. Nilai yang kita tanam: 6, 12, dan −2,5. Selang kepercayaan (dua kolom terakhir) dan p-value (`P>|t|`) tersedia untuk tiap baris. `0.000` berarti di bawah 0,0005.
- **R kuadrat disesuaikan** (0,883) menghukum penambahan variabel yang tidak berguna; pakai ini untuk membandingkan model dengan jumlah variabel berbeda.
- **Diagnostik** memeriksa apakah asumsi regresi masuk akal. (1) **Sisaan** (residual, selisih data dan model) sebaiknya mendekati normal: uji Shapiro p = 0,804, tidak ada tanda pelanggaran. (2) Ragam sisaan sebaiknya sama di semua tingkat (**homoskedastisitas**): uji Breusch-Pagan p = 0,746, aman. (3) Penjelas sebaiknya tidak terlalu kembar satu sama lain (**multikolinearitas**).
- **VIF** (*variance inflation factor*) mengukur itu. Kita tambah `pakan_pelet_g` yang hampir kembar dengan `pakan_g`, dan VIF keduanya melonjak ke 43,4 (aturan kasar: di atas 5 sampai 10 mengkhawatirkan), sedangkan suhu dan padat tetap 1,0. Akibatnya koefisien tiap kembaran menjadi tidak stabil. Obatnya: buang salah satu, gabungkan, atau pakai regularisasi (bagian berikut).

> TIPS
>
> Selalu **gambar sisaan** terhadap nilai ramalan. Pola lengkung berarti bentuk model salah; pola corong (melebar) berarti ragam tidak sama. Diagnostik bukan formalitas: model yang asumsinya patah memberi selang kepercayaan yang menyesatkan.

Regresi logistik: hasilnya ya atau tidak

Bila yang diramal hanya dua kemungkinan (pohon hidup atau mati, ternak sakit atau sehat), garis lurus akan meramal "peluang" di bawah 0 atau di atas 1. **Regresi logistik** meramal **peluang** (selalu antara 0 dan 1) lewat kurva S.

```
# Bab 27 - Regresi logistik: peluang pohon bertahan hidup
import numpy as np
import pandas as pd
import statsmodels.formula.api as smf

rng = np.random.default_rng(42)
n = 200
diameter = rng.uniform(5, 50, n)
plot = rng.choice(["A", "B", "C"], n)
logit = -2.5 + 0.12 * diameter + np.where(plot == "C", -0.8, 0)
hidup = rng.random(n) < 1 / (1 + np.exp(-logit))
pohon = pd.DataFrame({"hidup": hidup.astype(int), "diameter_cm": diameter, "plot":
plot})

model = smf.logit("hidup ~ diameter_cm + C(plot)", data=pohon).fit(disp=False)
print(model.params.round(3))

# Koefisien ada dalam skala log-odds; ubah jadi rasio odds (odds ratio)
print()
print("Rasio odds (OR):")
print(np.exp(model.params).round(3))
print("Tiap tambahan 1 cm diameter mengalikan odds bertahan dengan",
round(np.exp(model.params["diameter_cm"]), 3))

# Peluang untuk pohon baru
baru = pd.DataFrame({"diameter_cm": [10, 30], "plot": ["A", "A"]})
print()
print("Peluang bertahan (diameter 10 dan 30 cm, plot A):",
model.predict(baru).round(3).tolist())

# Ketepatan dengan ambang 0,5
prediksi = (model.predict(pohon) >= 0.5).astype(int)
```

```
print("Ketepatan:", round((prediksi == pohon["hidup"]).mean(), 3))
print("Proporsi hidup sebenarnya:", round(pohon["hidup"].mean(), 3))
```

```
Intercept      -2.385
C(plot)[T.B]   -0.084
C(plot)[T.C]   -0.487
diameter_cm     0.107
dtype: float64

Rasio odds (OR):
Intercept       0.092
C(plot)[T.B]    0.919
C(plot)[T.C]    0.614
diameter_cm     1.113
dtype: float64
Tiap tambahan 1 cm diameter mengalikan odds bertahan dengan 1.113

Peluang bertahan (diameter 10 dan 30 cm, plot A): [0.211, 0.693]
Ketepatan: 0.77
Proporsi hidup sebenarnya: 0.555
```

- Koefisien ada di skala **log-odds**, sulit dibaca langsung. `np.exp(koefisien)` mengubahnya menjadi **rasio odds** (OR): tiap tambahan 1 cm diameter mengalikan odds bertahan hidup dengan 1,113 (naik sekitar 11%). Plot C punya OR 0,614: odds bertahan lebih rendah, sesuai yang kita tanam.
- **Odds** = peluang / (1 - peluang). Peluang 0,5 sama dengan odds 1. Odds dan peluang bukan hal yang sama, jadi jangan menyebut "11% lebih mungkin" untuk peluang.
- `predict` memberi peluang: pohon 10 cm di plot A sekitar 0,211 dan 30 cm sekitar 0,693.
- Dengan ambang 0,5, ketepatan 0,77, lebih baik dari menebak "semua hidup" (proporsi hidup sebenarnya 0,555). Untuk kasus tak seimbang (penyakit langka), ketepatan saja menyesatkan; periksa juga presisi dan *recall* di Seri A3.

GLM: data cacah dengan Poisson

Jumlah buah per pohon, jumlah telur, atau jumlah pohon tumbang adalah bilangan bulat tak negatif. **GLM** (*generalized linear model*) memperluas regresi ke keluarga sebaran lain. Untuk cacahan, keluarga **Poisson** dengan tautan log dipakai.

```
# Bab 27 - GLM Poisson untuk data cacah: jumlah buah per pohon
import numpy as np
import pandas as pd
import statsmodels.api as sm
import statsmodels.formula.api as smf

rng = np.random.default_rng(42)
n = 150
diameter = rng.uniform(10, 50, n)
mu = np.exp(0.5 + 0.06 * diameter)
rata = mu * rng.gamma(shape=4, scale=0.25, size=n)   # keragaman ekstra antarpohon
buah = rng.poisson(rata)
data = pd.DataFrame({"buah": buah, "diameter_cm": diameter})

poisson = smf.glm("buah ~ diameter_cm", data=data, family=sm.families.Poisson()).fit()
print(poisson.params.round(4))
print("Rasio laju (IRR) per cm:", round(np.exp(poisson.params["diameter_cm"]), 4))

# Periksa overdispersion: bagi chi-kuadrat Pearson dengan derajat bebas; idealnya
sekitar 1
dispersi = poisson.pearson_chi2 / poisson.df_resid
print("Dispersi:",  round(dispersi,  2),  "(jauh  di  atas  1  berarti  Poisson  terlalu
```

```
optimis)")

# Alternatif: binomial negatif, yang menaksir sendiri keragaman ekstra (alpha)
nb = smf.negativebinomial("buah ~ diameter_cm", data=data).fit(disp=False)
print()
print("Binomial negatif, alpha  :", round(nb.params["alpha"], 3))
print("SE diameter Poisson      :", round(poisson.bse["diameter_cm"], 5))
print("SE diameter binom. negatif:", round(nb.bse["diameter_cm"], 5))
```

```
Intercept      0.6400
diameter_cm    0.0577
dtype: float64
Rasio laju (IRR) per cm: 1.0594
Dispersi: 4.64 (jauh di atas 1 berarti Poisson terlalu optimis)

Binomial negatif, alpha  : 0.267
SE diameter Poisson      : 0.00234
SE diameter binom. negatif: 0.00477
```

- `exp(0,0577)` = 1,0594: tiap tambahan 1 cm diameter menaikkan jumlah buah rata-rata sekitar 5,9% (**rasio laju**, IRR).
- Poisson mengandaikan **ragam = rata-rata**. Data nyata sering lebih beragam (**overdispersion**). Ukurannya: chi-kuadrat Pearson dibagi derajat bebas. Di sini **4,64**, jauh di atas 1. Akibatnya galat baku Poisson terlalu kecil dan kesimpulan terlalu percaya diri.
- **Binomial negatif** menaksir keragaman ekstra sendiri (`alpha` = 0,267). Galat baku diameter menjadi 0,00477, hampir dua kali lipat 0,00234 dari Poisson. Itulah harga kejujuran.

Ridge dan Lasso: bila penjelas banyak dan saling mirip

Dengan banyak penjelas yang saling berkaitan, OLS menaksir koefisien yang liar (positif di satu tempat, negatif di tempat lain) dan **overfit**: bagus pada data latih, jelek pada data baru. **Regularisasi** menambahkan "denda" untuk koefisien besar. **Ridge** menyusutkan semua koefisien; **Lasso** menyusutkan dan bisa menolkan sebagian, sehingga sekaligus memilih variabel.

```
# Bab 27 - Regularisasi (Ridge dan Lasso): bobot sapi dari banyak ukuran badan
import numpy as np
from sklearn.linear_model import LassoCV, LinearRegression, RidgeCV
from sklearn.model_selection import train_test_split
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler

rng = np.random.default_rng(42)
n, p = 90, 10
dasar = rng.normal(size=(n, 1))
# 10 pengukuran yang saling berkaitan erat; hanya 3 yang benar-benar berpengaruh
X = dasar + rng.normal(0, 0.5, size=(n, p))
koef_sebenarnya = np.array([30, 20, 10, 0, 0, 0, 0, 0, 0, 0])
bobot = 350 + X @ koef_sebenarnya / 3 + rng.normal(0, 15, n)   # kg (fiktif)

X_latih, X_uji, y_latih, y_uji = train_test_split(X, bobot, test_size=0.3,
random_state=42)

model = {
    "OLS biasa": make_pipeline(StandardScaler(), LinearRegression()),
    "Ridge": make_pipeline(StandardScaler(), RidgeCV(alphas=np.logspace(-2, 3, 30))),
    "Lasso": make_pipeline(StandardScaler(), LassoCV(cv=5, random_state=42)),
}
for nama, m in model.items():
    m.fit(X_latih, y_latih)
```

```
    koef = m[-1].coef_
       print(f"{nama:10}  R2  uji  =  {m.score(X_uji,  y_uji):.3f}     koefisien  nol:
{int((np.abs(koef) < 1e-9).sum())} dari {p}")
    print("           ", np.round(koef, 1) + 0.0)
```

```
OLS biasa  R2 uji = 0.680   koefisien nol: 0 dari 10
           [12.9  0.5 -4.8  6.5 -1.8 -0.4  3.6  2.   2.5 -0.4]
Ridge      R2 uji = 0.754   koefisien nol: 0 dari 10
           [ 8.5  2.2 -2.   4.3 -1.1  0.6  2.5  2.1  2.5  0.6]
Lasso      R2 uji = 0.724   koefisien nol: 5 dari 10
           [12.2  0.   0.   3.2  0.   0.   1.5  0.7  1.9  0. ]
```

Kita memisahkan 30% data sebagai **data uji** yang tidak dilihat model saat belajar. R kuadrat pada data uji: OLS 0,680, Lasso 0,724, Ridge 0,754. Koefisien OLS acak-acakan (12,9, 0,5, −4,8, ...), Ridge lebih tenang, dan Lasso menolkan 5 dari 10. Skala penjelas diseragamkan lebih dulu (`StandardScaler`) karena denda bergantung pada skala. Besar denda dipilih otomatis oleh validasi silang (`RidgeCV`, `LassoCV`).

> ⚠ AWAS
>
> Skor di **data latih** selalu terlalu bagus. Nilai model hanya jujur bila diuji pada data yang tidak dipakai menyusunnya. Prinsip ini dipakai terus di Seri A3.

Regresi non-linear: kurva pertumbuhan

Ikan tidak tumbuh lurus selamanya; ia mendekati panjang maksimum. Kurva **von Bertalanffy** menggambarkannya: L(t) = Lmaks × (1 - e^(-k (t - t0))). Parameternya bukan koefisien linear, jadi kita pakai `curve_fit` yang mencari nilai terbaik secara bertahap, dimulai dari tebakan awal `p0`.

```
# Bab 27 - Regresi non-linear: kurva pertumbuhan ikan (von Bertalanffy)
import numpy as np
from scipy.optimize import curve_fit


def panjang(umur, l_maks, k, t0):
    """Panjang ikan (cm) pada umur tertentu (bulan)."""
    return l_maks * (1 - np.exp(-k * (umur - t0)))


rng = np.random.default_rng(42)
umur = np.linspace(1, 24, 30)
nyata = panjang(umur, l_maks=45, k=0.15, t0=-0.5)
ukur = nyata + rng.normal(0, 1.2, umur.size)       # panjang terukur dengan galat

taksiran, kovarians = curve_fit(panjang, umur, ukur, p0=[40, 0.1, 0])
galat = np.sqrt(np.diag(kovarians))
for nama, nilai, se in zip(["l_maks", "k", "t0"], taksiran, galat):
    print(f"{nama:7} = {nilai:7.3f}  (galat baku {se:.3f})")

sisa = ukur - panjang(umur, *taksiran)
print("Akar rata-rata kuadrat sisaan:", round(float(np.sqrt((sisa ** 2).mean())), 3),
"cm")
print("Panjang ramalan umur 12 bulan:", round(float(panjang(12, *taksiran)), 1), "cm")
print("Panjang ramalan umur 36 bulan:", round(float(panjang(36, *taksiran)), 1), "cm
(mendekati l_maks)")
```

```
l_maks  =  45.405  (galat baku 0.521)
k       =   0.146  (galat baku 0.007)
t0      =  -0.484  (galat baku 0.165)
```

```
Akar rata-rata kuadrat sisaan: 0.899 cm
Panjang ramalan umur 12 bulan: 38.1 cm
Panjang ramalan umur 36 bulan: 45.2 cm (mendekati l_maks)
```

Taksiran: Lmaks = 45,405 cm (galat baku 0,521), k = 0,146 per bulan (0,007), t0 = −0,484. Nilai yang kita tanam: 45, 0,15, dan −0,5, jadi ketiganya ditemukan kembali dalam kisaran galat baku. Ramalan umur 36 bulan (45,2 cm) mendekati Lmaks: kurva melandai. Galat baku berasal dari matriks kovarians yang dikembalikan `curve_fit`.

> ⚠ AWAS
>
> Kurva non-linear bisa sangat peka pada tebakan awal `p0` dan bisa gagal konvergen. Gambar kurva di atas datanya sebelum percaya. Jangan meramal jauh di luar rentang data (ekstrapolasi): model hanya tahu apa yang pernah dilihatnya.

Model efek campuran: data berkelompok

Sapi di kandang yang sama tidak independen: mereka berbagi pakan, kebersihan, dan pengurus. Memperlakukan semuanya seolah independen (OLS biasa) menipu galat baku. **Model efek campuran** (*mixed-effects*) menambahkan **efek acak** per kelompok (kandang) di samping **efek tetap** yang ingin diketahui (pakan).

```
# Bab 27 - Model efek campuran (mixed-effects): sapi di banyak kandang
import numpy as np
import pandas as pd
import statsmodels.formula.api as smf

rng = np.random.default_rng(42)
baris = []
for kandang in range(1, 9):
    efek_kandang = rng.normal(0, 12)            # tiap kandang punya "nasib" sendiri
    for _ in range(10):
        pakan = rng.uniform(8, 14)              # kg pakan per hari
        tambah = 20 + 6 * pakan + efek_kandang + rng.normal(0, 8)
        baris.append({"kandang": f"K{kandang}", "pakan_kg": pakan, "tambah_bobot_kg":
tambah})
sapi = pd.DataFrame(baris)

biasa = smf.ols("tambah_bobot_kg ~ pakan_kg", data=sapi).fit()
campuran     =     smf.mixedlm("tambah_bobot_kg     ~     pakan_kg",     data=sapi,
groups=sapi["kandang"]).fit()

print("OLS biasa     : koef pakan =", round(biasa.params["pakan_kg"], 3), " galat baku
=", round(biasa.bse["pakan_kg"], 3))
print("Efek campuran : koef pakan =", round(campuran.params["pakan_kg"], 3), " galat
baku =", round(campuran.bse["pakan_kg"], 3))

ragam_kandang = float(campuran.cov_re.iloc[0, 0])
ragam_sisa = float(campuran.scale)
print("Ragam antar-kandang :", round(ragam_kandang, 1))
print("Ragam sisa          :", round(ragam_sisa, 1))
print("ICC (porsi keragaman karena kandang):", round(ragam_kandang / (ragam_kandang +
ragam_sisa), 3))
print()
print("Efek acak per kandang (selisih dari rata-rata):")
print(pd.Series({k:     round(float(v.iloc[0]),     1)     for     k,     v     in
campuran.random_effects.items()}).to_string())
```

```
OLS biasa     : koef pakan = 5.131  galat baku = 0.772
Efek campuran : koef pakan = 5.635  galat baku = 0.476
Ragam antar-kandang : 103.0
Ragam sisa          : 47.9
ICC (porsi keragaman karena kandang): 0.683

Efek acak per kandang (selisih dari rata-rata):
K1     6.5
K2    -5.7
K3    -8.0
K4     7.5
K5   -14.9
K6    -2.6
K7    16.3
K8     1.0
```

- OLS biasa menaksir pengaruh pakan 5,131 (galat baku 0,772); model campuran 5,635 (0,476). Nilai yang kita tanam 6. Model campuran memisahkan "nasib kandang" dari kebisingan, sehingga lebih tepat.
- **Ragam antar-kandang** 103,0 dan ragam sisa 47,9. **ICC** (*intraclass correlation*) = 103,0 / (103,0 + 47,9) = 0,683: sekitar 68% keragaman pertambahan bobot berasal dari perbedaan kandang. Ini besar: mengabaikan kandang adalah kesalahan serius.
- Efek acak per kandang menunjukkan K7 (+16,3) paling beruntung dan K5 (-14,9) paling merugi, relatif terhadap rata-rata.

Model campuran adalah alat standar untuk plot bersarang di hutan, kolam dalam tambak, atau pengukuran berulang pada pohon yang sama.

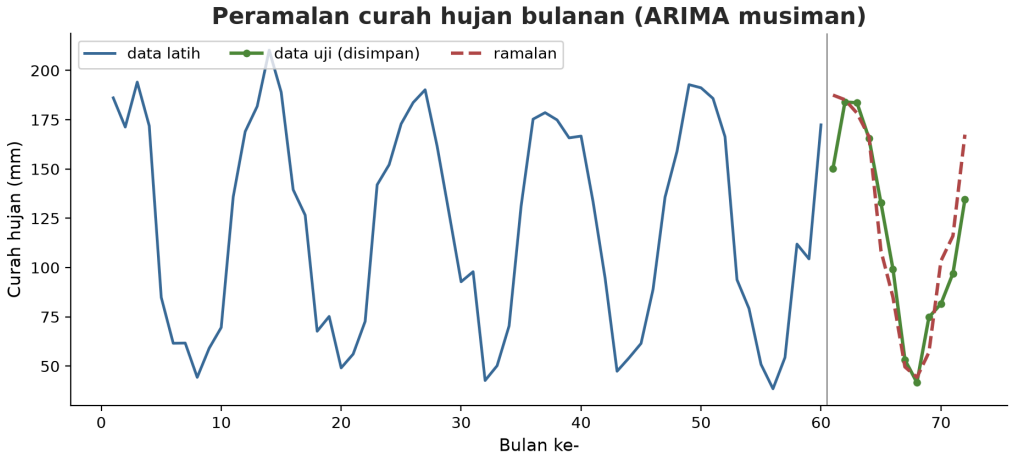
Deret waktu dan peramalan dengan ARIMA

Data yang diukur berurutan dalam waktu (curah hujan bulanan) punya ingatan: nilai bulan ini mirip bulan lalu, dan ada pola musiman. **ARIMA** memodelkan itu lewat tiga bagian: AR (nilai lalu menjelaskan nilai sekarang), I (selisih untuk meratakan tren), MA (galat lalu ikut berpengaruh). `seasonal_order=(P, D, Q, 12)` menambahkan pola berulang tiap 12 bulan.

```
# Bab 27 - Deret waktu dan peramalan dengan ARIMA: curah hujan bulanan
import numpy as np
import pandas as pd
from statsmodels.tsa.arima.model import ARIMA

rng = np.random.default_rng(42)
bulan = pd.period_range("2019-01", periods=72, freq="M")
musim = 120 + 70 * np.cos(2 * np.pi * (np.arange(72) - 1) / 12)       # pola tahunan
hujan = pd.Series(musim + rng.normal(0, 18, 72), index=bulan).clip(lower=0).round(1)
# mm (fiktif)

latih, uji = hujan[:60], hujan[60:]              # simpan 12 bulan terakhir untuk
menguji ramalan

model = ARIMA(latih, order=(1, 0, 0), seasonal_order=(1, 1, 0, 12)).fit()
ramalan = model.forecast(steps=12)

mae = (uji - ramalan).abs().mean()
# Pembanding sederhana: "bulan ini sama dengan bulan yang sama tahun lalu"
naif = pd.Series(latih.values[-12:], index=uji.index)
mae_naif = (uji - naif).abs().mean()
print("MAE ramalan ARIMA musiman  :", round(mae, 1), "mm")
print("MAE pembanding naif musiman:", round(mae_naif, 1), "mm")
print()
print(pd.DataFrame({"sebenarnya": uji, "ramalan": ramalan.round(1)}).head(6))
```

```
MAE ramalan ARIMA musiman  : 15.0 mm
MAE pembanding naif musiman: 17.8 mm

         sebenarnya  ramalan
2024-01       150.3    187.5
2024-02       184.0    185.1
2024-03       183.6    178.4
2024-04       165.6    166.5
2024-05       132.8    108.5
2024-06        99.3     85.0
```

- Kita **menyimpan 12 bulan terakhir** sebagai ujian dan meramalnya dari 60 bulan sebelumnya. Itu cara jujur menguji peramalan.
- Pembanding sederhana (“bulan ini = bulan sama tahun lalu”) rata-rata meleset 17,8 mm; ARIMA musiman 15,0 mm. Model yang rumit harus selalu dibandingkan dengan pembanding sederhana; bila tidak menang jauh, pakai yang sederhana.
- Tabel menunjukkan enam bulan pertama: ramalan dekat pada musim, tetapi tidak menangkap kebisingan acak tiap bulan (Januari: 187,5 vs 150,3 sebenarnya). Peramalan memberi pola, bukan kepastian.



Gambar 0.2: Peramalan curah hujan bulanan: data latih, data uji, dan ramalan ARIMA musiman.

Regresi spasial: GWR

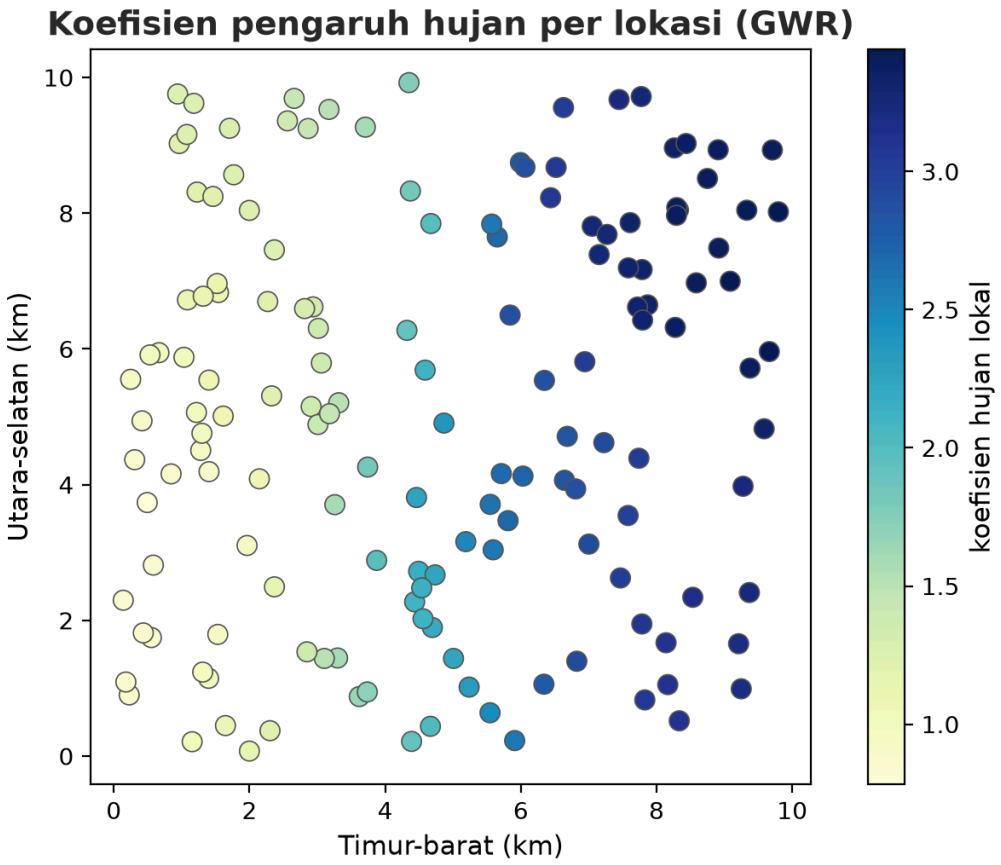
Regresi biasa memberi **satu** koefisien untuk seluruh wilayah. Padahal pengaruh hujan terhadap hasil panen bisa berbeda di lereng barat dan timur. **GWR** (*geographically weighted regression*) menjalankan banyak regresi lokal, tiap lokasi dengan koefisien sendiri, menimbang tetangga dekat lebih berat. Pustaka `mgwr` mengerjakannya.

```
# Bab 27 - Regresi spasial: GWR (geographically weighted regression) dengan mgwr
import numpy as np
import statsmodels.api as sm
from mgwr.gwr import GWR
from mgwr.sel_bw import Sel_BW

rng = np.random.default_rng(42)
n = 150
# Koordinat 150 petak kebun di area 10 x 10 km (fiktif)
koordinat = rng.uniform(0, 10, size=(n, 2))
hujan = rng.normal(0, 1, size=(n, 1))                  # curah hujan terstandar
# Pengaruh hujan menguat dari barat (x = 0) ke timur (x = 10)
pengaruh = 0.5 + 0.35 * koordinat[:, [0]]
```

```
panen = 5 + pengaruh * hujan + rng.normal(0, 0.4, size=(n, 1))   # ton/ha (fiktif)

# Model global: satu koefisien untuk seluruh area
global_ols = sm.OLS(panen, sm.add_constant(hujan)).fit()
print("OLS global: koefisien hujan =", round(float(global_ols.params[1]), 3), " R2 =",
round(float(global_ols.rsquared), 3))

# GWR: koefisien berbeda di tiap lokasi
lebar = Sel_BW(koordinat, panen, hujan).search()
hasil = GWR(koordinat, panen, hujan, lebar).fit()
lokal = hasil.params[:, 1]
print("Lebar jendela terpilih (jumlah tetangga):", int(lebar))
print("GWR: R2 =", round(float(hasil.R2), 3))
print("Koefisien hujan lokal: min =", round(float(lokal.min()), 2), " maks =",
round(float(lokal.max()), 2))

barat = koordinat[:, 0] < 3
timur = koordinat[:, 0] > 7
print("Rata-rata koefisien di barat (x < 3):", round(float(lokal[barat].mean()), 2))
print("Rata-rata koefisien di timur (x > 7):", round(float(lokal[timur].mean()), 2))
print("Pengaruh sebenarnya di barat/timur :", round(float(pengaruh[barat].mean()), 2),
"/", round(float(pengaruh[timur].mean()), 2))
```

```
OLS global: koefisien hujan = 2.176  R2 = 0.788
Lebar jendela terpilih (jumlah tetangga): 46
GWR: R2 = 0.97
Koefisien hujan lokal: min = 0.78  maks = 3.44
Rata-rata koefisien di barat (x < 3): 1.09
Rata-rata koefisien di timur (x > 7): 3.26
Pengaruh sebenarnya di barat/timur : 1.01 / 3.41
```

- Skrip ini memakai `mgwr`, pustaka tambahan di luar paket dasar bab ini; pasang dengan `pip install mgwr` (ia ikut membawa `libpysal`, `spreg`, dan `spglm`). Di komputer penulis, pustaka itu tersedia sebagai paket jadi untuk Python 3.14 dan skrip ini dijalankan sungguhan.
- OLS global menaksir satu koefisien hujan 2,176 (R kuadrat 0,788). GWR menemukan koefisien lokal dari 0,78 sampai 3,44: lemah di barat (rata-rata 1,09) dan kuat di timur (3,26), mendekati nilai yang kita tanam (1,01 dan 3,41). Satu angka global menyembunyikan pola ini.
- `Sel_BW` memilih **lebar jendela** (di sini 46 tetangga terdekat) lewat pencarian otomatis. Jendela sempit = sangat lokal tetapi berisik; lebar = mulus tetapi kurang peka.
- Hati-hati: R kuadrat GWR 0,97 dihitung pada data yang sama dengan yang dipakai menaksir, dan GWR jauh lebih fleksibel dari OLS, jadi angka itu tidak boleh dibandingkan begitu saja dengan 0,788. Gunakan kriteria seperti AICc atau data uji. Koordinat di sini fiktif; data peta sungguhan dan proyeksinya dibahas di seri geospasial.



Gambar 0.3: Koefisien hujan lokal hasil GWR pada 150 petak kebun fiktif: makin ke timur makin kuat.

Simulasi Monte Carlo: menghitung ketidakpastian

Hasil, harga, dan biaya tidak pasti. Alih-alih satu angka laba, kita **mengundi** nilai masukan ratusan ribu kali dari sebaran yang masuk akal, menghitung laba tiap kali, dan melihat sebaran labanya. Ini **Monte Carlo** (dinamai kasino, karena "mengundi").

```
# Bab 27 - Simulasi Monte Carlo: ketidakpastian pendapatan panen jagung
import numpy as np

rng = np.random.default_rng(42)
N = 100_000
luas_ha = 2.0

# Tiap masukan tidak pasti, jadi kita undi dari sebaran yang masuk akal (angka contoh
fiktif)
hasil_ton_ha = rng.triangular(left=3.0, mode=5.0, right=6.5, size=N)   # hasil panen
harga_rp_kg = rng.normal(loc=5000, scale=700, size=N)                  # harga jual
biaya_rp = rng.normal(loc=40_000_000, scale=3_000_000, size=N)         # biaya total

pendapatan = hasil_ton_ha * 1000 * luas_ha * harga_rp_kg
laba = pendapatan - biaya_rp

print("Rata-rata laba (juta Rp):", round(laba.mean() / 1e6, 2))
print("Simpangan baku (juta Rp):", round(laba.std() / 1e6, 2))
print("Selang 90% laba (juta Rp):", np.percentile(laba, [5, 95]).round(-4) / 1e6)
print("Peluang rugi             :", round(float((laba < 0).mean()), 3))
```

```
# Berapa banyak undian yang cukup? Bandingkan beberapa ukuran
for n in (100, 1000, 100_000):
     print(f"  taksiran peluang rugi dari {n:>7} undian:", round(float((laba[:n] <
0).mean()), 3))
```

```
Rata-rata laba (juta Rp): 8.36
Simpangan baku (juta Rp): 10.34
Selang 90% laba (juta Rp): [-8.03 25.96]
Peluang rugi             : 0.216
  taksiran peluang rugi dari     100 undian: 0.21
  taksiran peluang rugi dari    1000 undian: 0.224
  taksiran peluang rugi dari  100000 undian: 0.216
```

Rata-rata laba 8,36 juta rupiah, tetapi simpangan bakunya 10,34 juta: ketidakpastiannya besar. Selang 90% laba: dari −8,03 sampai 25,96 juta. **Peluang rugi** 21,6%. Satu angka rata-rata menyembunyikan risiko ini. Tiga baris terakhir menunjukkan taksiran peluang rugi stabil di sekitar 0,2 setelah ribuan undian; 100 undian terlalu sedikit untuk angka yang rapi. Semua angka (hasil, harga, biaya) adalah contoh fiktif, bukan harga atau anjuran sebenarnya.

Optimisasi: mencari pilihan terbaik

`scipy.optimize` menyelesaikan dua jenis soal umum. **Pemrograman linear** (`linprog`): minimalkan biaya dengan syarat berupa persamaan dan pertidaksamaan lurus. **Optimisasi satu variabel** (`minimize_scalar`): cari titik tertinggi atau terendah suatu fungsi.

```
# Bab 27 - Optimisasi dengan scipy.optimize: ramuan pakan termurah
from scipy.optimize import linprog, minimize_scalar

# Dua bahan: jagung (Rp 5.000/kg, protein 8%) dan bungkil kedelai (Rp 9.000/kg,
protein 44%)
# Kebutuhan: 100 kg ramuan dengan protein paling sedikit 18%. Angka contoh fiktif.
biaya = [5000, 9000]
# linprog meminta bentuk "kurang dari": -(protein) <= -(kebutuhan)
batas_protein_a = [[-0.08, -0.44]]
batas_protein_b = [-0.18 * 100]
total_a = [[1, 1]]
total_b = [100]

hasil = linprog(biaya, A_ub=batas_protein_a, b_ub=batas_protein_b,
                A_eq=total_a, b_eq=total_b, bounds=[(0, None), (0, None)])
print("Berhasil?   :", hasil.success)
print("Jagung (kg) :", round(hasil.x[0], 2))
print("Kedelai (kg):", round(hasil.x[1], 2))
print("Biaya (Rp)  :", round(hasil.fun))

# Optimisasi satu variabel: padat tebar ikan yang memaksimalkan laba (model fiktif)
def laba_negatif(padat):
    bobot_akhir = 900 * (1 - padat / 120)        # bobot makin kecil bila terlalu padat
     return -(padat * bobot_akhir / 1000 * 25000 - padat * 800)

cari = minimize_scalar(laba_negatif, bounds=(1, 100), method="bounded")
print()
print("Padat tebar terbaik (ekor/m2):", round(cari.x, 1))
print("Laba maksimum (Rp/m2)        :", round(-cari.fun))
```

```
Berhasil?   : True
Jagung (kg) : 72.22
Kedelai (kg): 27.78
Biaya (Rp)  : 611111
```

```
Padat tebar terbaik (ekor/m2): 57.9
Laba maksimum (Rp/m2)        : 627853
```

- Ramuan pakan 100 kg dengan protein paling sedikit 18% memakai 72,22 kg jagung dan 27,78 kg bungkil kedelai, biaya Rp 611.111. Bungkil kedelai lebih mahal tetapi kaya protein; mesin menemukan campuran termurah yang memenuhi syarat. `linprog` hanya menerima bentuk "kurang dari atau sama dengan", sehingga syarat "protein minimal" ditulis dengan tanda minus.
- Soal kedua: padat tebar yang memaksimalkan laba adalah 57,9 ekor/m2. Terlalu jarang memboroskan kolam; terlalu padat mengerdilkan ikan. Optimum ada di tengah. Modelnya sengaja sederhana dan fiktif.

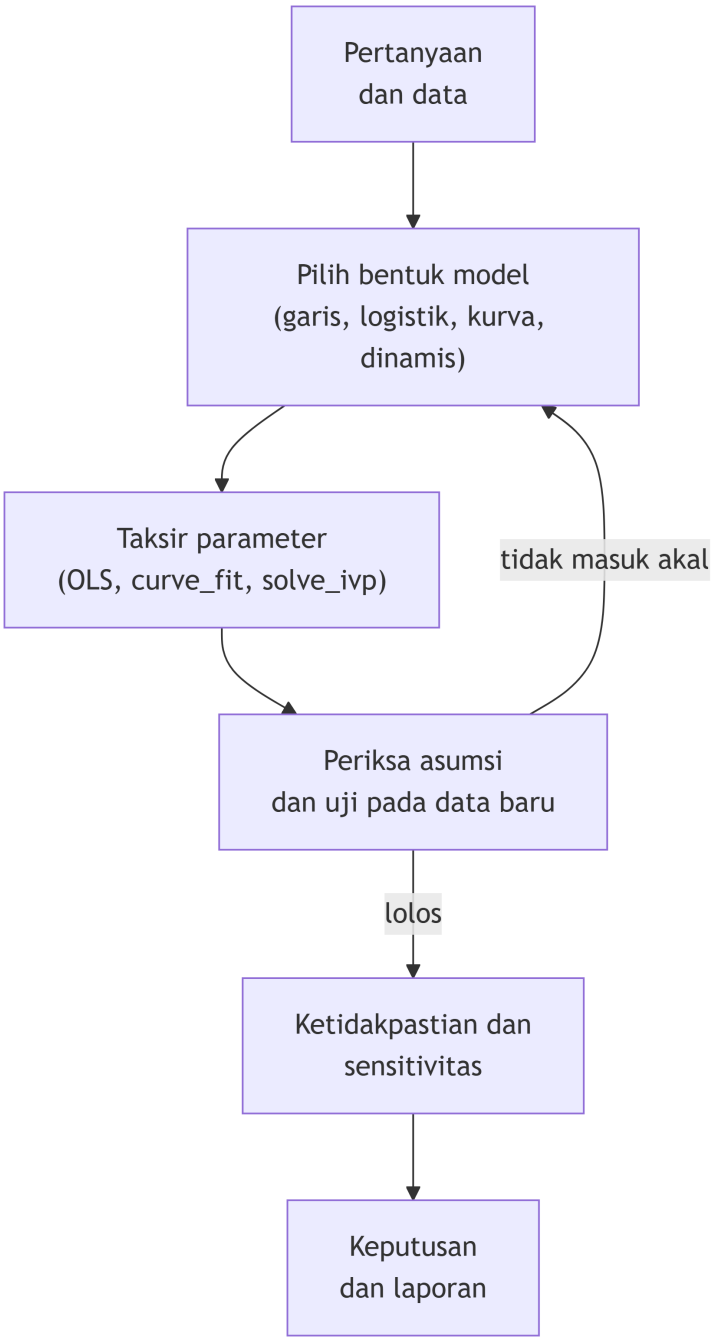
Model sistem dinamis dan analisis sensitivitas

Sistem yang berubah terhadap waktu (stok ikan, tegakan hutan) sering ditulis sebagai persamaan **laju perubahan**. Model **logistik** menyatakan stok tumbuh sebesar r × stok × (1 - stok / K) (K = daya dukung), dikurangi panen. `solve_ivp` menghitung perjalanannya langkah demi langkah.

```
# Bab 27 - Sistem dinamis dan analisis sensitivitas: stok ikan di waduk kecil
import numpy as np
from scipy.integrate import solve_ivp


def stok_akhir(r=0.8, K=1000.0, panen=100.0, awal=300.0, tahun=10):
    """Populasi logistik dengan panen tetap per tahun (ton, angka fiktif)."""
    def laju(t, y):
        return [r * y[0] * (1 - y[0] / K) - panen]
    punah = lambda t, y: y[0]       # peristiwa: stok menyentuh nol
    punah.terminal = True
    sol = solve_ivp(laju, (0, tahun), [awal], events=punah, t_eval=[tahun])
    return 0.0 if sol.status == 1 else float(sol.y[0, -1])


dasar = stok_akhir()
print("Stok setelah 10 tahun (kasus dasar):", round(dasar, 1), "ton")

# Analisis sensitivitas satu-per-satu: ubah tiap parameter +-20%
print()
print(f"{'parameter':10} {'-20%':>8} {'+20%':>8} {'rentang':>9}")
for nama in ("r", "K", "panen", "awal"):
    nilai = dict(r=0.8, K=1000.0, panen=100.0, awal=300.0)
    bawah = stok_akhir(**{**nilai, nama: nilai[nama] * 0.8})
    atas = stok_akhir(**{**nilai, nama: nilai[nama] * 1.2})
    print(f"{nama:10} {bawah:8.1f} {atas:8.1f} {abs(atas - bawah):9.1f}")

# Titik kritis: panen berapa yang masih lestari?
for panen in (100, 150, 190, 210):
    print(f"panen {panen} ton/tahun -> stok akhir {stok_akhir(panen=panen):7.1f} ton")
```

```
Stok setelah 10 tahun (kasus dasar): 844.8 ton

parameter      -20%     +20%   rentang
r             753.3    880.2     126.9
K             636.5   1048.6     412.1
panen         882.4    798.9      83.5
awal          837.7    847.9      10.2
panen 100 ton/tahun -> stok akhir   844.8 ton
panen 150 ton/tahun -> stok akhir   679.6 ton
panen 190 ton/tahun -> stok akhir     0.0 ton
panen 210 ton/tahun -> stok akhir     0.0 ton
```

- Kasus dasar: stok 300 ton naik menjadi 844,8 ton setelah 10 tahun dengan panen 100 ton per tahun.
- **Analisis sensitivitas** mengubah satu parameter sebesar ±20% sambil menahan yang lain, lalu melihat seberapa jauh hasil bergeser. Daya dukung K paling berpengaruh (rentang 412,1 ton), laju r sedang (126,9), panen kecil (83,5), dan stok awal hampir tidak (10,2) karena stok akan menuju keseimbangan. Hasil ini menunjukkan data mana yang layak diukur teliti: **K**.
- **Titik kritis**: panen 150 ton/tahun masih aman (stok 679,6), tetapi 190 ton/tahun **menghancurkan stok** (0,0), padahal angka itu di bawah batas teoretis rK/4 = 200 ton. Sebabnya, dengan panen 190 persamaan keseimbangan stok memiliki dua titik: sekitar 388 ton (tak stabil) dan 612 ton (stabil). Stok awal 300 berada di bawah titik tak stabil, jadi stok meluncur turun sampai habis. Stok ikan nyata bisa runtuh mendadak. Itulah alasan model sistem berguna.



Alur pemodelan kuantitatif: dari pertanyaan ke keputusan, dengan putaran kembali bila hasil tidak masuk akal.

Proyek Bersambung I3: model untuk sistem informasi kita

Kita menyatukan empat model proyek: alometrik pohon, bobot ikan terhadap pakan, bobot sapi terhadap pakan, dan hasil panen terhadap curah hujan. Fungsi `ringkas()` merangkum tiap model menjadi satu baris.

```
# Bab 27 - Proyek bersambung I3: empat model untuk sistem informasi kita
import numpy as np
import pandas as pd
import statsmodels.formula.api as smf

rng = np.random.default_rng(42)


def ringkas(nama, model, peubah):
    """Satu baris ringkasan: koefisien, selang 95%, R kuadrat."""
    b = model.params[peubah]
    lo, hi = model.conf_int().loc[peubah]
    return {"model": nama, "peubah": peubah, "koef": round(b, 3),
            "bawah95": round(lo, 3), "atas95": round(hi, 3), "R2": round(model.rsquared,
3)}


# 1. Alometrik pohon: ln(biomassa) ~ ln(diameter)
d = rng.uniform(10, 60, 60)
pohon = pd.DataFrame({"ln_d": np.log(d), "ln_b": np.log(0.12 * d ** 2.4 *
np.exp(rng.normal(0, 0.15, 60)))})
m1 = smf.ols("ln_b ~ ln_d", data=pohon).fit()

# 2. Ikan: pertambahan bobot (g) ~ pakan (g/hari)
pakan_ikan = rng.uniform(20, 60, 50)
ikan = pd.DataFrame({"pakan": pakan_ikan, "tambah_g": 15 + 4.2 * pakan_ikan +
rng.normal(0, 20, 50)})
m2 = smf.ols("tambah_g ~ pakan", data=ikan).fit()

# 3. Sapi: pertambahan bobot (kg) ~ pakan (kg/hari)
pakan_sapi = rng.uniform(8, 14, 40)
sapi = pd.DataFrame({"pakan": pakan_sapi, "tambah_kg": 10 + 5.5 * pakan_sapi +
rng.normal(0, 7, 40)})
m3 = smf.ols("tambah_kg ~ pakan", data=sapi).fit()

# 4. Panen vs curah hujan: kurva berpunuk (terlalu kering atau terlalu basah sama-
sama merugikan)
hujan = rng.uniform(600, 2400, 70)                      # mm per musim
panen = 0.5 + 0.0060 * hujan - 0.0000018 * hujan ** 2 + rng.normal(0, 0.3, 70)   #
ton/ha
kebun = pd.DataFrame({"hujan": hujan, "panen": panen})
m4 = smf.ols("panen ~ hujan + I(hujan ** 2)", data=kebun).fit()

tabel = pd.DataFrame([
    ringkas("Alometrik pohon (skala log)", m1, "ln_d"),
    ringkas("Ikan: bobot ~ pakan", m2, "pakan"),
    ringkas("Sapi: bobot ~ pakan", m3, "pakan"),
])
print(tabel.to_string(index=False))

b1, b2 = m4.params["hujan"], m4.params["I(hujan ** 2)"]
optimum = -b1 / (2 * b2)
```

```
print()
print("Panen ~ hujan + hujan^2: R2 =", round(m4.rsquared, 3), " koef kuadrat p =",
format(m4.pvalues["I(hujan ** 2)"], ".1e"))
print("Curah hujan optimum (puncak kurva):", round(optimum), "mm per musim")

# Ketidakpastian titik puncak lewat bootstrap (ulang-sampel dengan pengembalian)
puncak = []
for _ in range(500):
                          s      =      kebun.sample(len(kebun),      replace=True,
random_state=int(rng.integers(1_000_000)))
    mb = smf.ols("panen ~ hujan + I(hujan ** 2)", data=s).fit()
    puncak.append(-mb.params["hujan"] / (2 * mb.params["I(hujan ** 2)"]))
print("Selang   95%   bootstrap   untuk   puncak:",   np.percentile(puncak,   [2.5,
97.5]).round().tolist(), "mm")
```

```
                    model peubah  koef  bawah95  atas95    R2
Alometrik pohon (skala log)   ln_d 2.397    2.333   2.461 0.990
        Ikan: bobot ~ pakan  pakan 4.210    3.753   4.667 0.877
        Sapi: bobot ~ pakan  pakan 5.917    4.788   7.045 0.748

Panen ~ hujan + hujan^2: R2 = 0.608  koef kuadrat p = 1.6e-10
Curah hujan optimum (puncak kurva): 1711 mm per musim
Selang 95% bootstrap untuk puncak: [1653.0, 1800.0] mm
```

- Tabel: alometrik b = 2,397 (R kuadrat 0,990), ikan 4,210 g per gram pakan (0,877), sapi 5,917 kg per kg pakan (0,748). Selang kepercayaan memuat nilai yang kita tanam (2,4; 4,2; 5,5). Selang sapi paling lebar (4,788 sampai 7,045) karena datanya paling sedikit (40) dan paling berisik.
- Panen versus hujan berbentuk **lengkung berpunuk**: terlalu kering atau terlalu basah sama-sama merugikan. Model `panen ~ hujan + I(hujan ** 2)` tetap **linear dalam koefisien**, jadi OLS biasa cukup. Suku kuadrat sangat signifikan (p = 1,6e-10), R kuadrat 0,608.
- Puncak kurva, -b1 / (2 × b2), ada di sekitar **1711 mm** per musim (nilai yang kita tanam: 1667). Bootstrap (500 pengulangan sampel dengan pengembalian) memberi selang 95% untuk puncak: 1653 sampai 1800 mm. Memberi selang ketidakpastian pada angka "optimum" jauh lebih jujur daripada satu angka tunggal.

Semua model ini masuk ke laporan Quarto di Bab 29.

Latihan

1. Di `01_ols_alometrik.py`, ramal biomassa pohon 25 cm dan 55 cm (dengan koreksi bias). Mengapa meramal pohon 150 cm tidak aman?
2. Di `03_logistik.py`, ubah koefisien diameter di data dari 0,12 menjadi 0,02. Bagaimana ketepatan dan rasio odds berubah?
3. Di `12_sistem_dinamis.py`, cari (dengan mencoba beberapa nilai) panen terbesar yang membuat stok masih bertahan setelah 10 tahun. Bandingkan dengan 200 ton.
4. **Tantangan:** buat data dengan satu penjelas penting dan lima penjelas sampah, lalu bandingkan R kuadrat uji OLS dan Lasso. Kapan Lasso menang?

Ringkasan Bab

- **OLS**: `smf.ols("y ~ x", data=...)`; baca koefisien, selang 95%, R kuadrat. Log-log mengubah hubungan pangkat menjadi garis lurus.
- **Diagnostik**: sisaan normal (Shapiro), ragam sama (Breusch-Pagan), tidak kembar (VIF).
- **Logistik** untuk ya/tidak (rasio odds), **GLM Poisson** untuk cacah (periksa overdispersion), **Ridge/Lasso** untuk penjelas banyak (uji pada data uji).
- **Non-linear** (`curve_fit`), **efek campuran** (`mixedlm`) untuk data berkelompok, **ARIMA** untuk deret waktu (uji dengan data simpanan), **GWR** untuk koefisien yang berubah menurut lokasi.
- **Monte Carlo** menghitung ketidakpastian; **optimisasi** mencari pilihan terbaik; **sistem dinamis** dan **sensitivitas** menunjukkan titik kritis dan variabel yang paling berpengaruh.

- Model itu alat, bukan kebenaran: periksa asumsi, uji pada data baru, dan laporkan ketidakpastiannya.

Selanjutnya, di Bab 28: metode kualitatif dan campuran: bagaimana menganalisis wawancara dengan petani, nelayan, dan peternak, dan menggabungkannya dengan angka.

Bab 28 — Metode Kualitatif dan Campuran: Mendengar Cerita, Lalu Menghitungnya

Seri I3 — Statistik dan Pemodelan

Di bab ini, Anda akan:

- Memahami kapan memakai metode kualitatif, kuantitatif, atau campuran
- Mengode wawancara menjadi tema dengan buku kode, lalu menghitung kemunculannya
- Memakai alat teks dasar: frekuensi kata, KWIC, TF-IDF, topic modeling, dan sentimen berbasis kamus
- Memahami embedding secara konsep dan mengubah data kualitatif menjadi angka
- Menilai validitas, reliabilitas, dan etika penelitian, lalu menggabungkan cerita dan angka dalam mixed methods

Semua skrip bab ini ada di folder `kode/bab28/`. Di versi web, Anda juga bisa mengunduh semuanya sekaligus (zip).

Bab 26 dan 27 berurusan dengan angka. Tetapi bila Anda ingin tahu **mengapa** petani enggan menanam pohon di lahannya, atau **bagaimana** nelayan menghadapi musim paceklik, angka saja tidak cukup. Anda perlu **mendengar cerita**. Bayangkan seorang dokter hewan: angka suhu sapi memberi tahu bahwa sapi demam, tetapi cerita peternak ("kemarin sapi saya menolak makan, dan tetangga saya kehilangan dua ekor") memberi tahu mengapa dan apa yang harus dicari. Metode **kualitatif** menganalisis kata, makna, dan cerita; metode **campuran** (*mixed methods*) menggabungkan keduanya.

⚠ AWAS

Catatan etika dan data tiruan. Semua responden, kutipan wawancara, dan angka survei di bab ini **fiktif**, ditulis oleh penulis buku untuk latihan, dan tidak mewakili orang atau desa mana pun. Pada penelitian sungguhan Anda wajib: (1) meminta **persetujuan** responden setelah menjelaskan tujuan dan risikonya (*informed consent*, idealnya tertulis); (2) **menyamarkan identitas** (anonimisasi) pada data dan laporan; (3) menyimpan rekaman dan transkrip dengan aman; (4) menjelaskan bahwa responden boleh berhenti kapan saja. Untuk komunitas adat dan lokal, ada pula prinsip persetujuan komunitas yang diminta lebih dulu tanpa paksaan, yang perlu dipelajari dari pedoman lembaga etik Anda. Cek juga aturan lembaga atau universitas tentang izin penelitian.

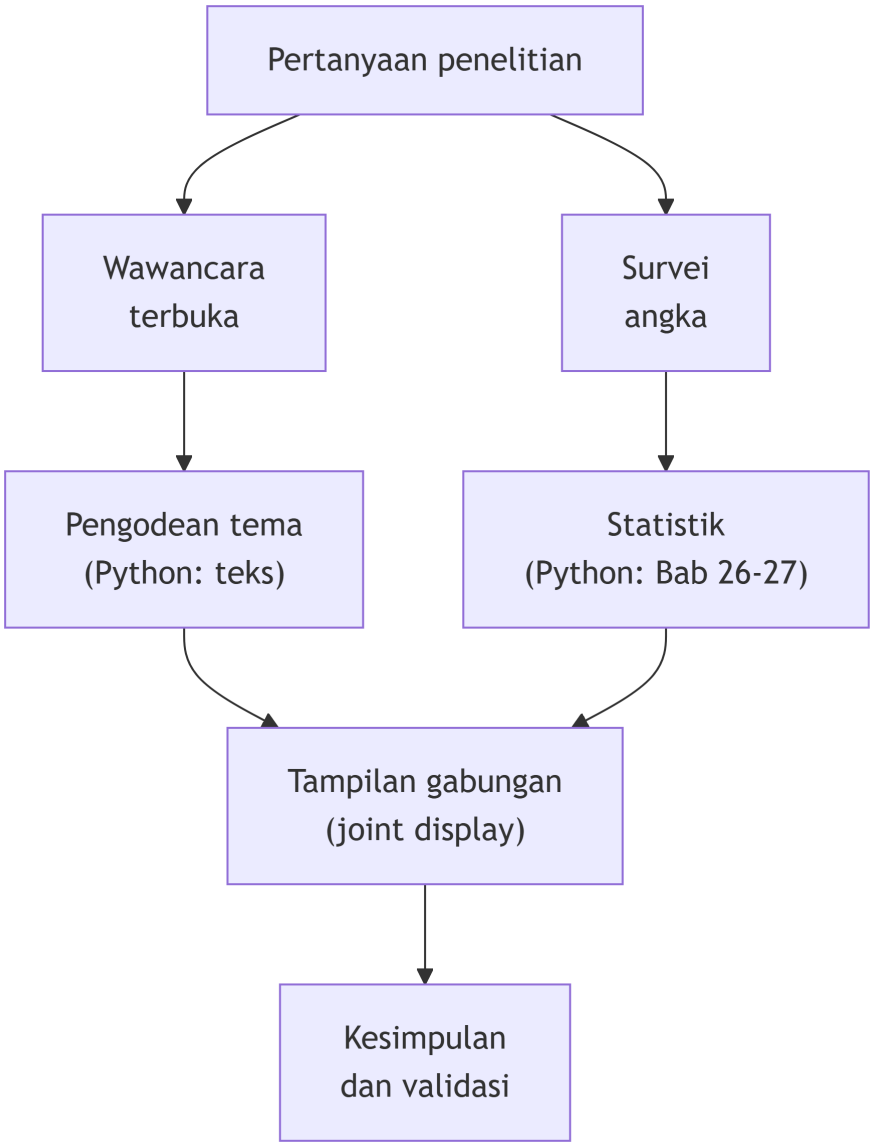
Kualitatif, kuantitatif, atau campuran?

	Kuantitatif	Kualitatif
Pertanyaan khas	Berapa banyak? Seberapa besar? Apakah beda?	Mengapa? Bagaimana? Apa maknanya?
Data	Angka (bobot, hasil panen, skor 1-5)	Teks, cerita, pengamatan, dokumen
Sampel	Besar, acak	Kecil, dipilih sesuai tujuan
Alat utama	Uji hipotesis, regresi (Bab 26-27)	Pengodean tema, analisis isi
Kekuatan	Dapat digeneralisasi, terukur	Kaya konteks, menemukan hal tak terduga

	Kuantitatif	Kualitatif
Kelemahan	Dangkal pada “mengapa”	Sulit digeneralisasi, rawan subjektivitas

Aturan praktis: **kuantitatif untuk “berapa”, kualitatif untuk “mengapa”**, dan keduanya bila keputusan yang dibuat penting. Ada tiga **desain campuran** umum:

- **Sekuensial menjelaskan**: survei angka dulu, lalu wawancara untuk menjelaskan temuan aneh.
- **Sekuensial menjajaki**: wawancara dulu untuk menemukan topik, lalu survei untuk mengukur seberapa umum.
- **Konvergen** (paralel): keduanya dikumpulkan bersamaan lalu dibandingkan. Proyek kita memakai ini.



Desain campuran konvergen: dua jalur data dikumpulkan bersamaan lalu digabung untuk satu kesimpulan.

Data wawancara tiruan kita

Kita memakai 24 wawancara singkat fiktif: 8 petani, 8 nelayan, 8 peternak. Pertanyaan terbukanya: *"Apa perubahan terbesar yang Anda rasakan dalam lima tahun terakhir, dan bagaimana Anda menghadapinya?"* Tiap responden juga mengisi skor **kepuasan** (1 sampai 5) dan **penghasilan** (juta rupiah per bulan), sehingga kita punya cerita dan angka untuk orang yang sama. Datanya ada di `kode/bab28/data/wawancara.csv`.

```
# Bab 28 - Membaca data wawancara tiruan (fiktif)
from pathlib import Path

import pandas as pd

# CATATAN ETIKA: seluruh responden dan kutipan di bab ini FIKTIF, ditulis penulis buku.
# Pada penelitian sungguhan Anda wajib meminta persetujuan tertulis (informed consent),
# menyamarkan identitas (anonimisasi), dan menyimpan rekaman dengan aman.
data = pd.read_csv(Path(__file__).parent / "data" / "wawancara.csv")

print("Jumlah responden:", len(data))
print(data["peran"].value_counts().to_string())
print()
print("Kepuasan rata-rata (skala 1-5) per peran:")
print(data.groupby("peran")["kepuasan"].mean().round(2).to_string())
print()
data["jumlah_kata"] = data["jawaban"].str.split().str.len()
print("Panjang   jawaban   (kata):   min",   data["jumlah_kata"].min(),   "   maks",
data["jumlah_kata"].max(),
      " rata-rata", round(data["jumlah_kata"].mean(), 1))
print()
for _, baris in data.head(3).iterrows():
              print(f"[{baris['kode']},   {baris['peran']},   {baris['umur']}   th]
{baris['jawaban']}")
```

```
Jumlah responden: 24
peran
petani      8
nelayan     8
peternak    8

Kepuasan rata-rata (skala 1-5) per peran:
peran
nelayan     2.50
petani      2.75
peternak    3.00

Panjang jawaban (kata): min 18  maks 24  rata-rata 20.4

[R01, petani, 45 th] Hujan sekarang tidak menentu. Tahun lalu sawah saya kekeringan
padahal sudah tanam, dan panen turun hampir separuh. Saya khawatir tahun ini terulang.
[R02, petani, 52 th] Pupuk makin mahal, tetapi harga gabah jatuh waktu panen raya.
Tengkulak datang menawar murah dan kami terpaksa menjual karena butuh uang.
[R03, petani, 38 th] Kelompok tani kami membantu. Kami membeli pupuk bersama sehingga
lebih murah, dan penyuluh datang tiap bulan. Hasil panen tahun ini bagus dan saya senang.
```

Tiap kelompok 8 orang. Kepuasan rata-rata nelayan 2,50, petani 2,75, peternak 3,00. Jawaban pendek, 18 sampai 24 kata (wawancara sungguhan jauh lebih panjang). Tiga kutipan pertama menunjukkan ragam isinya: kekeringan, harga murah dari tengkulak, dan kelompok tani yang membantu.

Pengodean tema

Pengodean (*coding*) adalah memberi label pada potongan teks yang membicarakan hal yang sama. Label itu disebut **kode** atau **tema**. Daftar tema beserta definisinya disebut **buku kode** (*codebook*). Ada dua cara menyusunnya:

- **Induktif**: tema muncul dari data; Anda membaca dulu, lalu menamai pola yang berulang.
- **Deduktif**: tema berasal dari teori atau pertanyaan penelitian, lalu dicari di data.

Pada praktiknya peneliti memakai campuran: mulai induktif dari sebagian transkrip, menyusun buku kode, lalu menerapkannya deduktif ke seluruh data. Python membantu langkah terakhir. Cara paling sederhana: tiap tema punya daftar **kata kunci**.

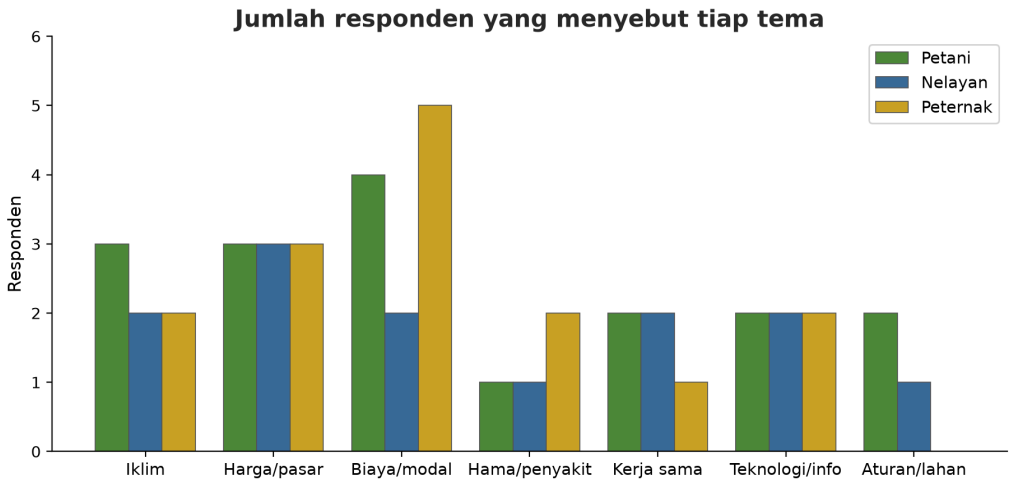
```
# Bab 28 - Pengodean tema berbasis buku kode (codebook) sederhana
import re
from pathlib import Path

import pandas as pd

data = pd.read_csv(Path(__file__).parent / "data" / "wawancara.csv")

# Buku kode: tema -> kata kunci (awalan kata). Disusun peneliti setelah membaca
sebagian data.
buku_kode = {
   "iklim": ["hujan", "kemarau", "cuaca", "banjir", "kering", "gelombang", "paceklik"],
    "harga_pasar": ["harga", "tengkulak", "pengepul", "jual", "margin"],
     "biaya_modal": ["modal", "pinjam", "bunga", "mahal", "biaya", "solar", "pupuk",
"pakan"],
    "hama_penyakit": ["hama", "wereng", "penyakit", "vaksin", "mati", "bom ikan"],
      "kerja_sama": ["kelompok", "koperasi", "gotong royong", "bersama", "berbagi",
"kerja sama"],
       "teknologi_info": ["aplikasi", "ponsel", "video", "informasi", "pelatihan",
"penyuluh"],
     "aturan_lahan": ["izin", "kawasan", "digusur", "aturan", "zona", "sertifikat",
"petugas"],
}


def beri_kode(teks):
    """Daftar tema yang kata kuncinya muncul di teks (huruf kecil, awal kata)."""
    teks = teks.lower()
    return [tema for tema, kunci in buku_kode.items()
            if any(re.search(r"\b" + re.escape(k), teks) for k in kunci)]


data["tema"] = data["jawaban"].apply(beri_kode)
for tema in buku_kode:
    data[tema] = data["tema"].apply(lambda daftar: tema in daftar)

print("Jumlah responden yang menyebut tiap tema (dari", len(data), "responden):")
print(data[list(buku_kode)].sum().sort_values(ascending=False).to_string())
print()
print("Rata-rata tema per responden:", round(data["tema"].str.len().mean(), 2))
print("Responden tanpa tema satu pun:", int((data["tema"].str.len() == 0).sum()))
print()
for kode in ("R01", "R03", "R09"):
    baris = data[data["kode"] == kode].iloc[0]
    print(kode, "->", baris["tema"])
```

```
Jumlah responden yang menyebut tiap tema (dari 24 responden):
biaya_modal       11
```

```
harga_pasar        9
iklim              7
teknologi_info     6
kerja_sama         5
hama_penyakit      4
aturan_lahan       3

Rata-rata tema per responden: 1.88
Responden tanpa tema satu pun: 0

R01 -> ['iklim']
R03 -> ['biaya_modal', 'kerja_sama', 'teknologi_info']
R09 -> ['iklim']
```

- `re.search(r"\b" + kata, teks)` mencari kata kunci yang **diawali** pada batas kata. `\b` berarti "batas kata", dipelajari di Bab 22.
- Tema terbanyak adalah `biaya_modal` (11 dari 24 responden), lalu `harga_pasar` (9) dan `iklim` (7). Tema paling jarang: `aturan_lahan` (3).
- Rata-rata 1,88 tema per responden: satu jawaban sering membicarakan beberapa hal, jadi jumlah tema lebih besar dari jumlah responden. R03 mendapat tiga tema: `biaya_modal`, `kerja_sama`, dan `teknologi_info`.

> ⚠ AWAS
>
> Kode berbasis kata kunci cepat dan konsisten tetapi **kasar**. Kata `kering` tidak menemukan "kekeringan" (awalannya beda), dan kata `mahal` selalu dihitung sebagai biaya padahal bisa tentang hal lain. Karena itu hasil mesin perlu **dibaca ulang manusia**. Anggap kode otomatis sebagai asisten yang cepat tetapi sembrono, bukan hakim.



Gambar 0.1: Jumlah responden yang menyebut tiap tema, per kelompok peran. Biaya dan modal disebut banyak kelompok; aturan lahan lebih khas bagi petani.

Analisis isi: frekuensi kata dan KWIC

Analisis isi menghitung dan mengelompokkan unsur teks secara sistematis. Dua alat dasar: **frekuensi kata** (setelah membuang kata umum seperti "yang", "dan", yang disebut *stopword*) dan **KWIC** (*keyword in context*), yaitu menampilkan sebuah kata kunci dengan beberapa kata di kiri dan kanannya agar Anda melihat bagaimana kata itu dipakai.

```
# Bab 28 - Analisis isi: frekuensi kata dan KWIC (kata dalam konteks)
import re
from collections import Counter
from pathlib import Path

import pandas as pd

data = pd.read_csv(Path(__file__).parent / "data" / "wawancara.csv")

KATA_UMUM = set("""yang dan di ke dari untuk dengan pada itu ini saya kami kita tidak
tetapi karena
sudah masih juga atau jadi tahun lalu sekarang makin sangat bisa ada akan kalau saat
waktu lebih
hanya agar oleh dalam sering banyak tiap satu dua sehingga padahal serta mereka mau
sama harus""".split())


def kata_kata(teks):
    return [k for k in re.findall(r"[a-z]+", teks.lower()) if k not in KATA_UMUM and
len(k) > 2]


semua = Counter()
for jawaban in data["jawaban"]:
    semua.update(kata_kata(jawaban))

print("15 kata paling sering (setelah membuang kata umum):")
for kata, n in semua.most_common(15):
    print(f"  {kata:12} {n}")

# KWIC: tampilkan kata kunci beserta 4 kata di kiri dan kanan
print()
print("Konteks kata 'harga':")
for kode, jawaban in zip(data["kode"], data["jawaban"]):
    token = re.findall(r"[A-Za-z]+", jawaban)
    for i, t in enumerate(token):
        if t.lower() == "harga":
            kiri = " ".join(token[max(0, i - 4):i])
            kanan = " ".join(token[i + 1:i + 5])
            print(f"  {kode}: ...{kiri} [{t}] {kanan}...")
            break
```

```
15 kata paling sering (setelah membuang kata umum):
  harga        9
  ikan         8
  pakan        6
  datang       5
  membantu     5
  rugi         5
  naik         5
  turun        4
  mahal        4
  bagus        4
  musim        4
  sulit        4
  biaya        4
  panen        3
  khawatir     3

Konteks kata 'harga':
```

```
  R02: ...Pupuk makin mahal tetapi [harga] gabah jatuh waktu panen...
 R05: ...melihat prakiraan cuaca dan [harga] Informasinya membantu menentukan waktu...
  R11: ...kami menjual ikan dengan [harga] stabil dan menyediakan es...
  R12: ...Pengepul menentukan [harga] dan sering menekan kami...
  R13: ...ikan lebih segar dan [harga] lebih baik...
  R21: ...Bobot sapi naik dan [harga] jual bagus Kerja sama...
  R22: ... [Harga] ayam jatuh waktu banyak...
```

Kata paling sering: `harga` (9), `ikan` (8), `pakan` (6). Frekuensi memberi **petunjuk**, bukan jawaban: "harga" muncul banyak, tetapi KWIC memperlihatkan konteksnya sangat berbeda. R02: harga gabah jatuh (buruk); R11: harga stabil lewat koperasi (baik); R22: harga ayam jatuh sementara pakan tetap mahal. Kata yang sama, makna berlawanan. Itulah mengapa analisis isi tidak boleh berhenti di hitungan.

Pada **analisis dokumen kebijakan** (misalnya peraturan pengelolaan hutan), alat yang sama dipakai: hitung kata kunci seperti "masyarakat", "izin", dan "sanksi", lalu bandingkan dengan KWIC untuk melihat siapa yang disebut sebagai subjek dan siapa sebagai objek. **Analisis wacana** melangkah lebih jauh: ia bertanya bagaimana pilihan kata membentuk gambaran tentang siapa yang berwenang dan siapa yang bersalah. Bagian ini lebih banyak membaca cermat daripada menghitung, dan Python hanya membantu menemukan bagian yang layak dibaca.

TF-IDF: kata yang khas

Kata yang sering belum tentu **khas**. "Ikan" sering muncul pada nelayan, tetapi apa yang membedakan nelayan dari petani? **TF-IDF** (*term frequency-inverse document frequency*) memberi skor tinggi pada kata yang **sering di dokumen ini tetapi jarang di dokumen lain**. Di sini, tiap kelompok peran digabung menjadi satu "dokumen".

```
# Bab 28 - TF-IDF: kata yang khas untuk tiap kelompok responden
from pathlib import Path

import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer

data = pd.read_csv(Path(__file__).parent / "data" / "wawancara.csv")

KATA_UMUM = """yang dan di ke dari untuk dengan pada itu ini saya kami kita tidak
tetapi karena
sudah masih juga atau jadi tahun lalu sekarang makin sangat bisa ada akan kalau saat
waktu lebih
hanya agar oleh dalam sering banyak tiap satu dua sehingga padahal serta mereka mau
sama harus""".split()

# Gabungkan semua jawaban per peran menjadi satu "dokumen"
per_peran = data.groupby("peran")["jawaban"].apply(" ".join)

vektor     =     TfidfVectorizer(stop_words=KATA_UMUM,     token_pattern=r"[a-z]{3,}",
lowercase=True)
matriks = vektor.fit_transform(per_peran)
kata = vektor.get_feature_names_out()

print("Ukuran matriks (peran x kata):", matriks.shape)
for i, peran in enumerate(per_peran.index):
    skor = pd.Series(matriks[i].toarray()[0], index=kata).sort_values(ascending=False)
    print(f"{peran:9}", ", ".join(f"{k} ({s:.2f})" for k, s in skor.head(5).items()))
```

```
Ukuran matriks (peran x kata): (3, 217)
nelayan   ikan (0.63), harga (0.19), karang (0.16), tangkapan (0.16), penghasilan (0.16)
petani    panen (0.29), pupuk (0.19), murah (0.19), tanam (0.19), tengkulak (0.19)
peternak  pakan (0.48), membuat (0.24), ternak (0.24), naik (0.18), datang (0.18)
```

Matriks berukuran 3 peran x 217 kata. Kata khas nelayan: `ikan`, `karang`, `tangkapan`. Petani: `panen`, `pupuk`, `tengkulak`, `tanam`. Peternak: `pakan`, `ternak`. Skor TF-IDF tinggi artinya "ciri pembeda kelompok ini". Ini cara cepat menyusun gambaran awal sebelum membaca mendalam.

Topic modeling: topik dari tumpukan teks

Bila teksnya ribuan halaman, tidak mungkin membaca semuanya. **Topic modeling** mencari **topik** yang tersembunyi: kelompok kata yang cenderung muncul bersama. Kita pakai **NMF** (*non-negative matrix factorization*) dari scikit-learn pada matriks TF-IDF. Alternatif yang populer adalah `LatentDirichletAllocation` (LDA) di pustaka yang sama; keduanya dipakai serupa. Jumlah topik (`n_components`) kita tentukan sendiri.

```
# Bab 28 - Topic modeling dengan NMF (scikit-learn)
from pathlib import Path

import numpy as np
import pandas as pd
from sklearn.decomposition import NMF
from sklearn.feature_extraction.text import TfidfVectorizer

data = pd.read_csv(Path(__file__).parent / "data" / "wawancara.csv")

KATA_UMUM = """yang dan di ke dari untuk dengan pada itu ini saya kami kita tidak
tetapi karena
sudah masih juga atau jadi tahun lalu sekarang makin sangat bisa ada akan kalau saat
waktu lebih
hanya agar oleh dalam sering banyak tiap satu dua sehingga padahal serta mereka mau
sama harus""".split()

vektor = TfidfVectorizer(stop_words=KATA_UMUM, token_pattern=r"[a-z]{3,}", min_df=2)
X = vektor.fit_transform(data["jawaban"])
kata = vektor.get_feature_names_out()
print("Kata yang dipakai (muncul di >= 2 jawaban):", len(kata))

model = NMF(n_components=4, init="nndsvd", random_state=42, max_iter=500)
bobot_dok = model.fit_transform(X)          # responden x topik

for t, komponen in enumerate(model.components_):
    atas = np.argsort(komponen)[::-1][:6]
    print(f"Topik {t + 1}:", ", ".join(kata[atas]))

data["topik"] = bobot_dok.argmax(axis=1) + 1
print()
print("Jumlah responden dominan di tiap topik:")
print(data["topik"].value_counts().sort_index().to_string())
print()
print(pd.crosstab(data["topik"], data["peran"]).to_string())
```

```
Kata yang dipakai (muncul di >= 2 jawaban): 55
Topik 1: harga, jatuh, turun, hampir, menjual, khawatir
Topik 2: musim, sulit, biaya, panjang, pinjaman, membuat
Topik 3: jelas, takut, butuh, hewan, besar, menyerang
Topik 4: ikan, bagus, belajar, naik, bersama, membantu

Jumlah responden dominan di tiap topik:
topik
1    6
2    6
3    4
4    8
```

```
peran  nelayan  petani  peternak
topik
1            2       3         1
2            1       3         2
3            1       1         2
4            4       1         3
```

Kata yang dipakai hanya yang muncul di sedikitnya 2 jawaban (55 kata). Empat topik yang ditemukan, dengan kata teratasnya:

- Topik 1: *harga, jatuh, turun, menjual*: soal **harga yang anjlok**.
- Topik 2: *musim, sulit, biaya, pinjaman*: soal **musim sulit dan biaya**.
- Topik 3: *takut, hewan, menyerang, jelas*: soal **ancaman dan ketidakpastian**.
- Topik 4: *ikan, bagus, belajar, bersama, membantu*: soal **dukungan dan keberhasilan bersama**.

Nama topik adalah tafsiran **kita**, bukan keluaran mesin. Dengan hanya 24 jawaban pendek, topiknya kasar dan akan berubah bila Anda mengganti jumlah topik atau benih acak. Topic modeling cocok untuk **menjelajah** tumpukan teks besar, bukan untuk membuktikan sesuatu. Tabel silang di bawah menunjukkan topik 4 (dukungan) paling banyak dibicarakan oleh nelayan (4 orang) dan peternak (3).

Sentimen berbasis kamus

Analisis sentimen menilai apakah teks bernada positif atau negatif. Cara paling sederhana memakai **kamus**: daftar kata positif dan negatif, lalu skor = jumlah positif dikurangi jumlah negatif. Kita bandingkan dengan skor kepuasan yang diisi responden sendiri untuk menguji apakah kamus kita masuk akal.

```
# Bab 28 - Analisis sentimen berbasis kamus sederhana
from pathlib import Path
import re

import pandas as pd
from scipy import stats

data = pd.read_csv(Path(__file__).parent / "data" / "wawancara.csv")

POSITIF = {"membantu", "bagus", "senang", "puas", "membaik", "naik", "lancar",
"stabil", "segar",
           "sehat", "baik", "melimpah", "murah", "menolong", "unggul"}
NEGATIF = {"khawatir", "rugi", "sulit", "susah", "mahal", "turun", "jatuh", "takut",
"mati",
        "rusak", "hilang", "kurang", "tipis", "lambat", "membingungkan", "kekeringan",
"kurus", "menekan"}


def skor_sentimen(teks):
    kata = re.findall(r"[a-z]+", teks.lower())
    return sum(k in POSITIF for k in kata) - sum(k in NEGATIF for k in kata)


data["sentimen"] = data["jawaban"].apply(skor_sentimen)
data["label"] = pd.cut(data["sentimen"], bins=[-99, -1, 0, 99], labels=["negatif",
"netral", "positif"])

print(data["label"].value_counts().reindex(["negatif",
"netral", "positif"]).to_string())
print()
print("Rata-rata kepuasan per label sentimen:")
print(data.groupby("label", observed=True)["kepuasan"].mean().round(2).to_string())
```

```
rho, p = stats.spearmanr(data["sentimen"], data["kepuasan"])
print()
print("Korelasi sentimen kamus vs skor kepuasan (Spearman):", round(rho, 3), " p =",
format(p, ".1e"))

# Kelemahan kamus: cari responden yang labelnya bertentangan dengan skor kepuasannya
salah = data[((data["label"] == "positif") & (data["kepuasan"] <= 3))
             | ((data["label"] == "negatif") & (data["kepuasan"] >= 4))]
print()
print("Label bertentangan dengan kepuasan:", len(salah), "dari", len(data))

# Di data tiruan yang rapi ini kamus nyaris sempurna. Di dunia nyata ia gampang tertipu:
for kalimat in ("Hasil panen tidak bagus", "Pakan naik terus", "Harga naik, saya
senang"):
    print(f"  '{kalimat}' -> skor {skor_sentimen(kalimat)}")
```

```
label
negatif    17
netral      0
positif     7

Rata-rata kepuasan per label sentimen:
label
negatif    2.06
positif    4.43

Korelasi sentimen kamus vs skor kepuasan (Spearman): 0.802  p = 2.5e-06

Label bertentangan dengan kepuasan: 0 dari 24
  'Hasil panen tidak bagus' -> skor 1
  'Pakan naik terus' -> skor 1
  'Harga naik, saya senang' -> skor 2
```

- 17 jawaban bernada negatif dan 7 positif. Rata-rata kepuasan: 2,06 untuk yang negatif dan 4,43 untuk yang positif. Korelasi Spearman antara skor kamus dan kepuasan 0,802 (p = 2,5e-06): kamus kita cocok dengan penilaian responden. Perhatikan bahwa ini **validasi**: kita menguji alat dengan data pembanding.
- Pada data tiruan yang rapi ini tidak ada label yang bertentangan dengan kepuasan. Di dunia nyata kamus mudah tertipu. Tiga kalimat uji menunjukkan kelemahannya: "Hasil panen **tidak bagus**" bernilai +1 (kamus tidak mengerti negasi), "Pakan **naik** terus" bernilai +1 (naik itu baik untuk penghasilan, buruk untuk biaya), dan sarkasme tak terdeteksi.

> TIPS
>
> Untuk bahasa Indonesia, kamus sentimen siap pakai terbatas, dan kata bergantung pada bidang (di pertanian, "naik" bisa baik atau buruk). Susun kamus khusus bidang Anda, uji pada sampel yang dinilai manusia, dan laporkan kecocokannya.

Embedding: kata menjadi titik di peta makna

Komputer tidak mengerti kata, hanya angka. **Embedding** adalah cara mengubah teks menjadi deretan angka (**vektor**) sehingga teks yang mirip maknanya punya vektor yang berdekatan. Bayangkan peta: kota yang berdekatan secara geografis dekat di peta. Embedding membuat "peta makna": "banjir" dan "hujan lebat" berada berdekatan, "vaksin" jauh dari "solar".

Model bahasa besar menghasilkan embedding berkualitas tinggi, tetapi model itu besar dan **tidak dipasang di buku ini**; ia dibahas di Seri A4. Di sini kita membuat **embedding mini** dengan cara klasik: TF-IDF lalu dipadatkan oleh `TruncatedSVD` (disebut juga *latent semantic analysis*, LSA). Prinsipnya sama: teks menjadi vektor pendek, lalu kemiripan diukur dengan **kemiripan kosinus** (1 = arah sama, 0 = tak berkaitan).

```
# Bab 28 - Konsep embedding: teks menjadi vektor, jarak = kemiripan makna
from pathlib import Path

import numpy as np
import pandas as pd
from sklearn.decomposition import TruncatedSVD
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity

data = pd.read_csv(Path(__file__).parent / "data" / "wawancara.csv")

# Langkah 1: teks -> vektor TF-IDF (satu angka per kata, mayoritas nol)
tfidf = TfidfVectorizer(token_pattern=r"[a-z]{3,}")
X = tfidf.fit_transform(data["jawaban"])
print("Vektor TF-IDF: tiap jawaban punya", X.shape[1], "angka")

# Langkah 2: padatkan jadi 6 angka (LSA). Ini embedding mini, bukan model bahasa besar.
svd = TruncatedSVD(n_components=6, random_state=42)
tertanam = svd.fit_transform(X)
print("Setelah dipadatkan: tiap jawaban punya", tertanam.shape[1], "angka")

# Langkah 3: kemiripan kosinus antar jawaban; cari tetangga terdekat R01
kemiripan = cosine_similarity(tertanam)
np.fill_diagonal(kemiripan, -1)
print()
for kode in ("R01", "R17"):
    i = data.index[data["kode"] == kode][0]
    terdekat = np.argsort(kemiripan[i])[::-1][:2]
    print(f"Jawaban termirip dengan {kode}:")
    for j in terdekat:
      print(f"  {data.loc[j, 'kode']} (kemiripan {kemiripan[i, j]:.2f}): {data.loc[j,
'jawaban'][:55]}...")
```

```
Vektor TF-IDF: tiap jawaban punya 250 angka
Setelah dipadatkan: tiap jawaban punya 6 angka

Jawaban termirip dengan R01:
   R24 (kemiripan 0.66): Kandang saya rusak karena banjir. Modal untuk memperbai...
   R06 (kemiripan 0.65): Lahan saya dekat kawasan hutan dan izinnya tidak jelas....
Jawaban termirip dengan R17:
   R10 (kemiripan 0.89): Solar mahal dan hasil tangkapan menurun. Ikan makin jau...
   R23 (kemiripan 0.76): Saya belajar dari video di ponsel tentang membuat pakan...
```

Tiap jawaban awalnya 250 angka (satu per kata), lalu dipadatkan menjadi 6 angka. Jawaban termirip dengan R17 (pakan ternak mahal) adalah R10 (solar mahal dan hasil tangkapan menurun, kemiripan 0,89) dan R23 (biaya pakan, 0,76): pola "biaya input" ditemukan tanpa kita menyebut kata kuncinya. Untuk R01 (kekeringan), tetangganya R24 (banjir) dan R06 (izin lahan); yang kedua kurang masuk akal, tanda bahwa embedding mini dari 24 jawaban masih kasar. Embedding dari model besar jauh lebih peka terhadap makna.

Mengubah data kualitatif menjadi kuantitatif

Setelah tema dikode, tiap responden punya vektor 0/1: menyebut tema itu atau tidak. Itu **data kuantitatif**, dan semua alat Bab 26 berlaku.

```
# Bab 28 - Mengubah data kualitatif menjadi kuantitatif: tema menjadi tabel angka
import re
from pathlib import Path

import pandas as pd
```

```
from scipy import stats

data = pd.read_csv(Path(__file__).parent / "data" / "wawancara.csv")

buku_kode = {
   "iklim": ["hujan", "kemarau", "cuaca", "banjir", "kering", "gelombang", "paceklik"],
    "harga_pasar": ["harga", "tengkulak", "pengepul", "jual", "margin"],
     "biaya_modal": ["modal", "pinjam", "bunga", "mahal", "biaya", "solar", "pupuk",
"pakan"],
    "hama_penyakit": ["hama", "wereng", "penyakit", "vaksin", "mati", "bom ikan"],
     "kerja_sama": ["kelompok", "koperasi", "gotong royong", "bersama", "berbagi",
"kerja sama"],
      "teknologi_info": ["aplikasi", "ponsel", "video", "informasi", "pelatihan",
"penyuluh"],
     "aturan_lahan": ["izin", "kawasan", "digusur", "aturan", "zona", "sertifikat",
"petugas"],
}
for tema, kunci in buku_kode.items():
    data[tema] = data["jawaban"].str.lower().apply(
        lambda t: int(any(re.search(r"\b" + re.escape(k), t) for k in kunci)))

# 1. Tabel tema x peran (jumlah responden yang menyebut)
print("Jumlah penyebut tema per peran:")
print(data.groupby("peran")[list(buku_kode)].sum().T.to_string())

# 2. Apakah responden yang menyebut kerja sama lebih puas?
ya = data[data["kerja_sama"] == 1]["kepuasan"]
tidak = data[data["kerja_sama"] == 0]["kepuasan"]
print()
print("Kepuasan rata-rata, menyebut kerja sama:", round(ya.mean(), 2), "(n =", len(ya),
")")
print("Kepuasan rata-rata, tidak menyebut      :", round(tidak.mean(), 2), "(n =",
len(tidak), ")")
print("Mann-Whitney p =", round(stats.mannwhitneyu(ya, tidak).pvalue, 4))

# 3. Tabel 2x2 dan uji Fisher (cocok untuk sampel kecil)
tinggi = (data["kepuasan"] >= 4).astype(int)
tabel = pd.crosstab(data["kerja_sama"], tinggi)
print()
print(tabel.to_string())
print("Fisher exact p =", round(stats.fisher_exact(tabel)[1], 4))
```

```
Jumlah penyebut tema per peran:
peran           nelayan  petani  peternak
iklim                 2       3         2
harga_pasar           3       3         3
biaya_modal           2       4         5
hama_penyakit         1       1         2
kerja_sama            2       2         1
teknologi_info        2       2         2
aturan_lahan          1       2         0

Kepuasan rata-rata, menyebut kerja sama: 3.8 (n = 5 )
Kepuasan rata-rata, tidak menyebut     : 2.47 (n = 19 )
Mann-Whitney p = 0.0393

kepuasan     0  1
kerja_sama
0           15  4
```

```
1           2  3
Fisher exact p = 0.1265
```

- Tabel tema x peran menunjukkan misalnya `biaya_modal` disebut 5 dari 8 peternak, 4 petani, dan 2 nelayan.
- Responden yang menyebut kerja sama (n = 5) rata-rata puas 3,8 versus 2,47 untuk yang tidak menyebut (n = 19). Uji Mann-Whitney: p = 0,0393.
- Tetapi ketika kepuasan dibuat dua kelompok (4 atau lebih) dan diuji dengan **Fisher exact** (cocok untuk tabel kecil), p = 0,1265: tidak signifikan. Dengan sampel sekecil ini, kesimpulan bergantung pada cara menguji, dan keduanya wajib dilaporkan dengan jujur. Jangan memilih uji yang memberi p kecil.

Validitas dan reliabilitas

- **Validitas**: apakah kita mengukur apa yang kita maksud? Dalam kualitatif, caranya antara lain **triangulasi** (memeriksa dengan sumber atau metode lain, misalnya wawancara dibandingkan dengan survei), **member checking** (mengonfirmasi tafsiran kepada responden), dan **jejak audit** (mencatat keputusan pengodean agar orang lain bisa menelusuri).
- **Reliabilitas**: apakah hasilnya konsisten? Untuk pengodean, dua orang mengode data yang sama lalu kesepakatannya dihitung dengan **Cohen kappa**, yang membuang kesepakatan karena kebetulan. Untuk skala kuesioner (beberapa butir pertanyaan tentang hal yang sama), konsistensi diukur dengan **alfa Cronbach**.

```python
# Bab 28 - Reliabilitas: kesepakatan dua pengode (Cohen kappa) dan konsistensi skala (alfa Cronbach)
import re
from pathlib import Path

import numpy as np
import pandas as pd
import pingouin as pg
from sklearn.metrics import cohen_kappa_score

data = pd.read_csv(Path(__file__).parent / "data" / "wawancara.csv")
rng = np.random.default_rng(42)

kunci_kerja_sama = ["kelompok", "koperasi", "gotong royong", "bersama", "berbagi", "kerja sama"]
pengode_a = data["jawaban"].str.lower().apply(
    lambda t: int(any(re.search(r"\b" + k, t) for k in kunci_kerja_sama)))

# Pengode B (SIMULASI): sama dengan A, tetapi 3 dari 24 keputusan berbeda
pengode_b = pengode_a.copy()
beda = rng.choice(len(data), size=3, replace=False)
pengode_b.iloc[beda] = 1 - pengode_b.iloc[beda]

kesepakatan = (pengode_a == pengode_b).mean()
print("Kesepakatan mentah    :", round(kesepakatan, 3))
print("Cohen kappa           :", round(cohen_kappa_score(pengode_a, pengode_b), 3))
print("Responden yang berbeda:", list(data["kode"].iloc[sorted(beda)]))

# Alfa Cronbach: apakah 4 butir pertanyaan (skala 1-5) mengukur hal yang sama?
sifat = rng.normal(0, 1, len(data))          # "kepuasan sebenarnya" yang tak terlihat
butir = pd.DataFrame({
    f"butir{i}": np.clip(np.round(3 + sifat + rng.normal(0, 0.6, len(data))), 1, 5)
    for i in range(1, 5)
})
alfa, selang = pg.cronbach_alpha(data=butir)
print()
```

```
print("Alfa Cronbach 4 butir :", round(alfa, 3), " selang 95%:", [round(float(x), 2)
for x in selang])
```

```
Kesepakatan mentah    : 0.875
Cohen kappa           : 0.69
Responden yang berbeda: ['R02', 'R16', 'R18']

Alfa Cronbach 4 butir : 0.896  selang 95%: [0.81, 0.95]
```

Pengode B di skrip ini **simulasi**: sama dengan A kecuali 3 dari 24 keputusan sengaja dibalik (R02, R16, R18). Kesepakatan mentah 0,875 tampak tinggi, tetapi kappa hanya 0,69 karena tema kerja sama jarang muncul, sehingga banyak "sepakat" terjadi karena kebetulan. Pedoman kasar yang umum dipakai: kappa sekitar 0,6 sampai 0,8 dianggap cukup baik, di atas 0,8 sangat baik. Alfa Cronbach untuk 4 butir simulasi 0,896 dengan selang 95% 0,81 sampai 0,95; nilai di atas 0,7 umumnya dianggap memadai untuk skala penelitian.

Proyek Bersambung I3: analisis wawancara dengan mixed methods

Kita memakai desain konvergen: tema wawancara (kualitatif) dijajarkan dengan kepuasan dan penghasilan (kuantitatif) dalam satu **tampilan gabungan** (*joint display*), lalu diuji dengan regresi (Bab 27).

```
# Bab 28 - Proyek bersambung I3: mixed methods (menggabungkan tema wawancara dengan
angka survei)
import re
from pathlib import Path

import pandas as pd
import statsmodels.formula.api as smf

data = pd.read_csv(Path(__file__).parent / "data" / "wawancara.csv")

buku_kode = {
   "iklim": ["hujan", "kemarau", "cuaca", "banjir", "kering", "gelombang", "paceklik"],
    "harga_pasar": ["harga", "tengkulak", "pengepul", "jual", "margin"],
     "biaya_modal": ["modal", "pinjam", "bunga", "mahal", "biaya", "solar", "pupuk",
"pakan"],
    "hama_penyakit": ["hama", "wereng", "penyakit", "vaksin", "mati", "bom ikan"],
     "kerja_sama": ["kelompok", "koperasi", "gotong royong", "bersama", "berbagi",
"kerja sama"],
      "teknologi_info": ["aplikasi", "ponsel", "video", "informasi", "pelatihan",
"penyuluh"],
     "aturan_lahan": ["izin", "kawasan", "digusur", "aturan", "zona", "sertifikat",
"petugas"],
}
for tema, kunci in buku_kode.items():
    data[tema] = data["jawaban"].str.lower().apply(
        lambda t: int(any(re.search(r"\b" + re.escape(k), t) for k in kunci)))

# Tampilan gabungan (joint display): temuan kualitatif di samping angka kuantitatif
baris = []
for tema in buku_kode:
    ada = data[data[tema] == 1]
    baris.append({"tema": tema, "n": len(ada),
                  "kepuasan": round(ada["kepuasan"].mean(), 2),
                  "penghasilan_juta": round(ada["penghasilan_juta"].mean(), 2)})
print(pd.DataFrame(baris).to_string(index=False))
print("Semua responden: kepuasan", round(data["kepuasan"].mean(), 2),
```

```
      " penghasilan", round(data["penghasilan_juta"].mean(), 2))

# Model: apakah tema "dukungan" (kerja sama, teknologi) berkaitan dengan penghasilan?
model = smf.ols("penghasilan_juta ~ kerja_sama + teknologi_info + hama_penyakit +
iklim", data=data).fit()
print()
print(model.params.round(2).to_string())
print("Selang 95% kerja_sama :", model.conf_int().loc["kerja_sama"].round(2).tolist())
print("R kuadrat:", round(model.rsquared, 3), " n =", int(model.nobs))
```

```
          tema  n  kepuasan  penghasilan_juta
         iklim  7      2.43              2.07
   harga_pasar  9      3.00              2.53
   biaya_modal 11      2.45              2.06
 hama_penyakit  4      2.50              2.20
    kerja_sama  5      3.80              2.98
teknologi_info  6      4.17              3.03
  aturan_lahan  3      2.33              1.87
Semua responden: kepuasan 2.75  penghasilan 2.3

Intercept         1.94
kerja_sama        0.91
teknologi_info    1.00
hama_penyakit    -0.22
iklim            -0.15
Selang 95% kerja_sama : [0.38, 1.44]
R kuadrat: 0.623  n = 24
```

- Tampilan gabungan: responden yang menyebut `teknologi_info` (n = 6) punya kepuasan rata-rata 4,17 dan penghasilan 3,03 juta; yang menyebut `kerja_sama` (n = 5) 3,80 dan 2,98 juta. Sebaliknya yang menyebut `aturan_lahan` (n = 3) hanya 2,33 dan 1,87 juta; rata-rata seluruh responden 2,75 dan 2,3 juta.
- Regresi: menyebut kerja sama berkaitan dengan penghasilan **0,91 juta lebih tinggi** (selang 95% 0,38 sampai 1,44), dan menyebut teknologi dan informasi 1,00 juta lebih tinggi, setelah mengendalikan tema hama dan iklim. R kuadrat 0,623 dengan n = 24.
- Kesimpulan yang boleh ditarik: cerita dan angka **sejalan**. Orang yang bercerita tentang dukungan kelompok dan informasi cenderung lebih sejahtera. Yang **tidak** boleh ditarik: bahwa kelompok atau aplikasi *menyebabkan* penghasilan naik. Mungkin petani yang sudah berpenghasilan baik lebih mudah bergabung ke kelompok (sebab-akibat terbalik), atau ada faktor ketiga (Bab 26). Dan 24 responden fiktif tidak mewakili populasi mana pun. Wawancara lanjutan (“mengapa kelompok membantu?”) adalah langkah berikutnya pada desain sekuensial.

Hasil ini masuk ke laporan Quarto di Bab 29.

Latihan

1. Tambahkan tema baru `kesehatan_keluarga` ke buku kode dengan kata kunci pilihan Anda. Apakah ada responden yang menyebutnya? Apa artinya untuk buku kode?
2. Di `03_frekuensi_kata.py`, ganti KWIC dari “harga” ke “musim” atau “modal”. Apa yang Anda pelajari dari konteksnya?
3. Di `05_topik_nmf.py`, ubah `n_components` menjadi 3 dan 6. Seberapa stabil topiknya? Apa yang Anda simpulkan tentang memilih jumlah topik?
4. **Tantangan:** tulis 8 jawaban wawancara fiktif baru dengan kata-kata Anda sendiri (dengan catatan etika), tambahkan ke CSV, dan periksa berapa banyak yang dikode benar oleh buku kode. Perbaiki buku kodenya dan hitung kappa antara kode mesin dan kode Anda.

Ringkasan Bab

- **Kuantitatif** menjawab “berapa”, **kualitatif** menjawab “mengapa”, dan **campuran** menggabungkan keduanya (sekuensial atau konvergen).

- **Buku kode** berisi tema dan kata kunci; kode otomatis cepat tetapi kasar dan perlu dibaca ulang manusia.
- **Frekuensi kata**, **KWIC**, **TF-IDF**, **topic modeling** (NMF/LDA), dan **sentimen kamus** adalah alat penjelajah, bukan pembuktian. Validasi alat dengan data pembanding.
- **Embedding** mengubah teks menjadi vektor sehingga makna yang mirip berdekatan; model besar dibahas di Seri A4.
- Tema yang dikode menjadi data kuantitatif: uji dengan Mann-Whitney, Fisher, atau regresi, dan laporkan jujur bila hasilnya bergantung pada uji.
- **Validitas** (triangulasi, member checking), **reliabilitas** (Cohen kappa, alfa Cronbach), dan **etika** (persetujuan, anonimisasi) sama pentingnya dengan kode Python.

Selanjutnya, di Bab 29: riset yang dapat direproduksi: bagaimana membungkus semua analisis ini dalam laporan Quarto yang bisa dijalankan ulang oleh siapa pun.

Bab 29 — Riset yang Dapat Direproduksi: Analisis yang Bisa Diulang Siapa Saja

Seri I3 — Statistik dan Pemodelan

Di bab ini, Anda akan:

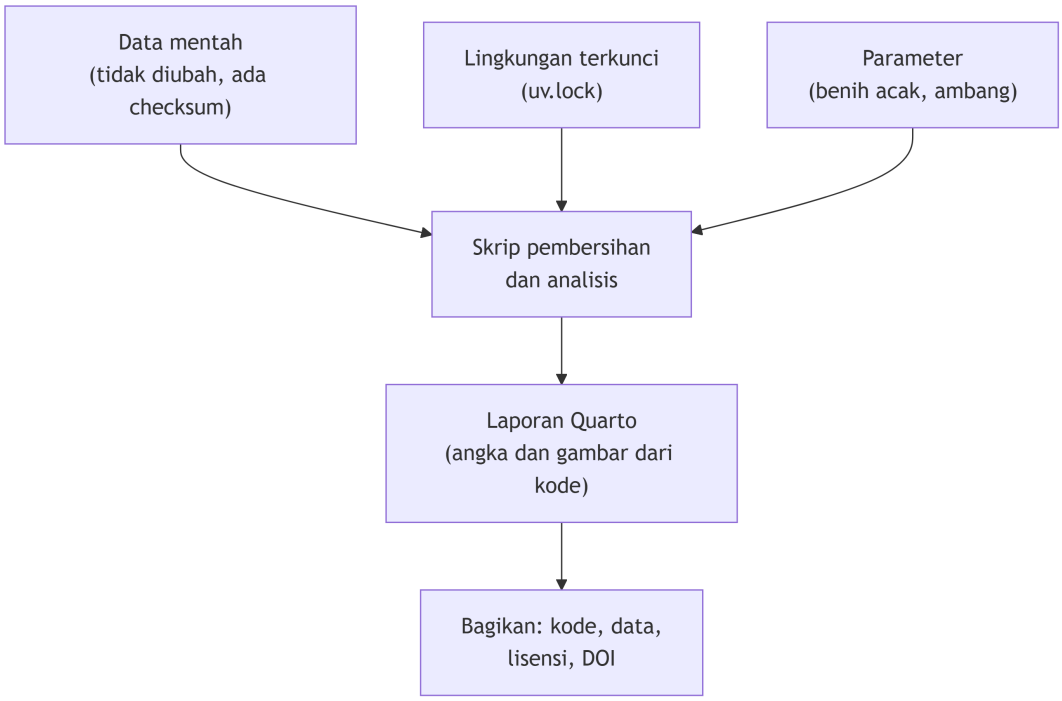
- Memahami beda mengulang (*replikasi*) dan mereproduksi, dan mengapa keduanya penting
- Menulis notebook Jupyter yang rapi dan bisa dijalankan dari atas ke bawah
- Membuat laporan Quarto berparameter yang dirender dari kode, dan mengenal Jupyter Book
- Mengunci lingkungan dengan `uv lock`, serta mengenal Snakemake dan DVC untuk alur dan versi data
- Membagikan data dan kode dengan benar: README, lisensi, CITATION.cff, checksum, dan DOI

Semua skrip bab ini ada di folder `kode/bab29/`. Di versi web, Anda juga bisa mengunduh semuanya sekaligus (zip).

Bayangkan Anda memberi resep rendang kepada tetangga, tetapi lupa menulis bahwa santannya harus kental dan apinya kecil. Hasilnya berbeda, dan Anda berdua bingung. Analisis data sering bernasib sama: setengah tahun kemudian, Anda sendiri tidak bisa mengulang angka di laporan, karena data sudah berubah, paket sudah diperbarui, atau lupa langkahnya. **Riset yang dapat direproduksi** artinya orang lain (dan Anda di masa depan) bisa mengambil **data dan kode yang sama**, menjalankannya, dan mendapat **angka yang sama**. Itu syarat minimal agar hasil analisis bisa dipercaya.

i INGAT

Dua istilah yang sering tertukar. **Reproduksi** (*reproducibility*): data dan kode yang sama menghasilkan angka yang sama. **Replikasi** (*replicability*): penelitian baru dengan data baru menghasilkan kesimpulan yang sama. Bab ini tentang yang pertama, dan itulah yang bisa kita jamin lewat teknik di bawah.



Alur riset yang dapat direproduksi: tiap kotak ditulis sebagai kode atau berkas, bukan klik manual. Seluruh bab ini berpegang pada satu prinsip: **semua langkah tertulis sebagai kode atau berkas, tidak ada langkah "saya klik sana-sini"**. Catatan penting: semua data di bab ini fiktif.

Notebook Jupyter yang rapi

Jupyter notebook menggabungkan teks, kode, dan hasil dalam satu berkas `.ipynb`. Sangat enak untuk menjelajah, tetapi mudah rusak: sel dijalankan tidak berurutan, variabel "hantu" tersisa di memori, dan hasilnya tidak bisa diulang. Aturan agar notebook rapi:

1. **Urutan dari atas ke bawah.** Notebook harus bisa dijalankan dengan "Restart kernel and run all" tanpa error.
2. **Pengaturan di satu sel di awal** (benih acak, jalur berkas, ambang), supaya mudah diubah.
3. **Impor di awal**, dan jangan bergantung pada sel yang sudah terhapus.
4. **Benih acak tetap** bila memakai bilangan acak.
5. **Teks penjelas** tentang apa dan mengapa, bukan hanya kode.
6. Bila isinya sudah matang, **pindahkan fungsi ke berkas `.py`** dan impor dari notebook.

Skrip berikut membuat notebook yang mengikuti aturan itu lewat kode (pustaka `nbformat`), lalu menjalankannya dari atas ke bawah di kernel baru (pustaka `nbclient`), seperti "Restart and run all".

```
# Bab 29 - Notebook Jupyter yang rapi: dibuat lewat kode, dijalankan dari atas ke bawah
import os
from pathlib import Path

import nbformat
from nbclient import NotebookClient

keluar = Path(__file__).parent / "_tmp"
keluar.mkdir(exist_ok=True)

nb = nbformat.v4.new_notebook()
nb.cells = [
    nbformat.v4.new_markdown_cell("# Uji beda tinggi pohon antar-plot\nData tiruan.
Benih tetap supaya hasilnya sama."),
```

```
    nbformat.v4.new_code_cell("# 1. Pengaturan: semua angka yang bisa diubah ada di
sini\nBENIH = 42\nN = 30"),
     nbformat.v4.new_code_cell("# 2. Impor\nimport numpy as np\nfrom scipy import
stats"),
    nbformat.v4.new_code_cell(
        "# 3. Data\nrng = np.random.default_rng(BENIH)\n"
        "plot_a = rng.normal(15, 2, N)\nplot_b = rng.normal(17, 2, N)"),
    nbformat.v4.new_code_cell(
        "# 4. Analisis\nhasil = stats.ttest_ind(plot_a, plot_b, equal_var=False)\n"
             "print(f'rata-rata A {plot_a.mean():.2f}, B {plot_b.mean():.2f}, p =
{hasil.pvalue:.1e}')"),
]
nb.metadata["kernelspec"] = {"name": "python3", "display_name": "Python 3", "language":
"python"}

# "Restart and run all": jalankan semua sel berurutan di kernel baru
# (Peringatan teknis dari kernel di Windows kita alihkan ke "kotak sampah" supaya
keluaran bersih.)
simpan_stderr = os.dup(2)
os.dup2(os.open(os.devnull, os.O_WRONLY), 2)
try:
     NotebookClient(nb, timeout=120, kernel_name="python3", resources={"metadata":
{"path": str(keluar)}}).execute()
finally:
    os.dup2(simpan_stderr, 2)
nbformat.validate(nb)
nbformat.write(nb, keluar / "uji_beda.ipynb")

print("Jumlah sel:", len(nb.cells), "(kode:", sum(c.cell_type == "code" for c in
nb.cells), ")")
print("Nomor eksekusi berurutan:", [c.execution_count for c in nb.cells if c.cell_type
== "code"])
for c in nb.cells:
    if c.cell_type == "code" and c.outputs:
        print("Keluaran sel:", c.outputs[0].text.strip())
```

```
Jumlah sel: 5 (kode: 4 )
Nomor eksekusi berurutan: [1, 2, 3, 4]
Keluaran sel: rata-rata A 15.03, B 17.23, p = 1.4e-06
```

Notebook punya 5 sel (4 kode, 1 teks). Nomor eksekusi 1, 2, 3, 4 berurutan: tanda notebook bersih. Bila nomornya melompat (misalnya 1, 5, 2), berarti sel pernah dijalankan acak, dan hasilnya patut dicurigai. Berkas hasilnya disimpan di folder `_tmp/`.

Quarto: laporan yang dibuat dari kode

Quarto adalah perangkat penerbitan ilmiah: satu berkas teks `.qmd` (Markdown ditambah sel kode) dirender menjadi HTML, PDF, atau Word. Kodenya dijalankan **saat merender**, sehingga angka dan gambar di laporan selalu berasal dari kode terbaru, tidak pernah disalin-tempel. Buku yang sedang Anda baca dibuat dengan cara yang sama.

Laporan kecil proyek kita ada di `kode/bab29/laporan/laporan.qmd`. Bagian pentingnya:

- **Kepala YAML** (di antara dua baris `---`) berisi judul dan format, misalnya `embed-resources: true` agar HTML menjadi satu berkas mandiri.
- Sel bertanda `#| tags: [parameters]` menyimpan nilai bawaan (`benih = 42`). Nilainya bisa diganti dari baris perintah tanpa mengubah berkas: `quarto render laporan.qmd -P benih:7`.
- **Ekspresi sebaris** `` `{python} benih` `` menaruh nilai variabel langsung di kalimat.
- Sel dengan `#| label: fig-...` dan `#| fig-cap: "..."` menghasilkan gambar bernomor dan bersubjudul.

```python
# Bab 29 - Merender laporan Quarto dari kode (laporan/laporan.qmd)
import os
import re
import shutil
import subprocess
import sys
from pathlib import Path

folder = Path(__file__).parent
kerja = folder / "_tmp" / "laporan"
if kerja.exists():
    shutil.rmtree(kerja)
kerja.mkdir(parents=True)
shutil.copy(folder / "laporan" / "laporan.qmd", kerja)

# Quarto menjalankan sel Python dengan penafsir yang kita tunjuk lewat QUARTO_PYTHON
lingkungan = dict(os.environ, QUARTO_PYTHON=sys.executable)
quarto = shutil.which("quarto") or "quarto"

for benih in (42, 7):
    hasil = subprocess.run(
        [quarto, "render", "laporan.qmd", "-P", f"benih:{benih}"],
        cwd=kerja, env=lingkungan, capture_output=True, text=True,
    )
    html = (kerja / "laporan.html").read_text(encoding="utf-8")
    ada = "Pada benih" in html
      print(f"benih {benih}: kode keluar {hasil.returncode}, HTML dibuat {(kerja /
'laporan.html').exists()}, "
          f"parameter terbaca di laporan: {ada}")
    teks = re.sub(r"<[^>]+>", "", html)
    print("   hasil di laporan:", re.search(r"Rata-rata A [0-9].*", teks).group(0))
```

```
benih 42: kode keluar 0, HTML dibuat True, parameter terbaca di laporan: True
   hasil di laporan: Rata-rata A 1.81, B 2.28; p = 0.0009
benih 7: kode keluar 0, HTML dibuat True, parameter terbaca di laporan: True
   hasil di laporan: Rata-rata A 1.55, B 2.20; p = 0.0001
```

Kode ini **tidak dijalankan** saat buku dibuat karena butuh Quarto dan Jupyter terpasang, jadi keluarannya di atas diambil sekali dari komputer penulis. Skrip merender laporan dua kali, dengan benih 42 dan 7. Hasilnya: dengan benih 42 rata-rata plot A 1,81 dan B 2,28 (p = 0,0009), persis sama dengan Bab 26; dengan benih 7 angkanya berubah (1,55 dan 2,20) **tanpa** menyentuh isi laporan. Itulah kekuatan laporan berparameter.

> TIPS
>
> Beberapa temuan nyata saat menyiapkan bab ini: (1) bendera `--output-dir` pada Quarto hanya berlaku untuk **proyek**, bukan berkas tunggal, jadi skrip menyalin laporan ke folder kerja lalu merender di sana; (2) parameter lewat `-P` pada notebook Jupyter membutuhkan pustaka `papermill` (Quarto akan memberi tahu bila belum terpasang); (3) Quarto memilih penafsir Python lewat variabel lingkungan `QUARTO_PYTHON`, dan skrip mengisinya dengan `sys.executable`.

Jupyter Book adalah perangkat serupa untuk membuat buku atau situs dari notebook. Menurut dokumentasi resminya, Jupyter Book 2 dibangun di atas MyST Document Engine; perintahnya `jupyter book` (dan `myst` untuk MyST langsung). Jupyter Book tidak dijalankan di buku ini; buka dokumentasi di jupyterbook.org bila Anda ingin menerbitkan seri notebook sebagai situs.

Lingkungan terkunci

Kode yang sama bisa menghasilkan angka berbeda bila versi `numpy` atau `scipy` berubah. Solusinya **mengunci** versi tepat semua paket. Dengan `uv` (dikenalkan di Seri I1), `pyproject.toml` menyatakan

paket yang Anda butuhkan, dan `uv lock` memilih versi persis semua paket (termasuk paket yang ikut terbawa) dan menuliskannya ke `uv.lock`. Orang lain cukup `uv sync` (instal sesuai kunci), atau `uv run` yang mengunci dan menyinkronkan otomatis. Opsi `--locked` memeriksa apakah kunci sesuai `pyproject.toml` dan gagal bila usang; `--frozen` memakai kunci apa adanya tanpa memeriksa. Alternatif yang lebih sederhana tetapi kurang lengkap adalah `pip freeze > requirements.txt`.

```
# Bab 29 - Lingkungan terkunci dengan uv: pyproject.toml menjadi uv.lock
import shutil
import subprocess
import tomllib
from pathlib import Path

kerja = Path(__file__).parent / "_tmp" / "proyek_kunci"
if kerja.exists():
    shutil.rmtree(kerja)
kerja.mkdir(parents=True)

(kerja / "pyproject.toml").write_text('''[project]
name = "analisis-hutan"
version = "0.1.0"
requires-python = ">=3.12"
dependencies = ["numpy", "scipy"]
''', encoding="utf-8")

# uv lock: pilih versi semua paket (termasuk paket bawaan) dan tulis ke uv.lock
hasil = subprocess.run(["uv", "lock"], cwd=kerja, capture_output=True, text=True)
print("uv lock, kode keluar:", hasil.returncode)

kunci = tomllib.loads((kerja / "uv.lock").read_text(encoding="utf-8"))
paket = {p["name"]: p["version"] for p in kunci["package"]}
print("Jumlah paket di uv.lock:", len(paket))
for nama in sorted(paket):
    print(f"  {nama} == {paket[nama]}")

# uv lock --check: apakah uv.lock masih sesuai dengan pyproject.toml?
cek = subprocess.run(["uv", "lock", "--check"], cwd=kerja, capture_output=True,
text=True)
print("uv lock --check, kode keluar:", cek.returncode, "(0 = masih sesuai)")

# Tambah dependensi di pyproject.toml tanpa mengunci ulang: kunci jadi usang
teks = (kerja / "pyproject.toml").read_text(encoding="utf-8").replace('"scipy"',
'"scipy", "pandas"')
(kerja / "pyproject.toml").write_text(teks, encoding="utf-8")
cek2 = subprocess.run(["uv", "lock", "--check"], cwd=kerja, capture_output=True,
text=True)
print("Setelah menambah pandas, uv lock --check, kode keluar:", cek2.returncode,
"(bukan 0 = kunci usang)")
```

```
uv lock, kode keluar: 0
Jumlah paket di uv.lock: 3
  analisis-hutan == 0.1.0
  numpy == 2.5.3
  scipy == 1.18.1
uv lock --check, kode keluar: 0 (0 = masih sesuai)
Setelah menambah pandas, uv lock --check, kode keluar: 1 (bukan 0 = kunci usang)
```

Kode ini **tidak dijalankan** saat buku dibuat karena butuh internet dan versi paket berubah seiring waktu, jadi keluarannya diambil sekali. Hasilnya: `uv lock` sukses, dan `uv.lock` berisi 3 paket (proyek itu sendiri, `numpy` 2.5.3, dan `scipy` 1.18.1 pada hari dijalankan). `uv lock --check` bernilai 0 (kunci sesuai). Setelah kita menambah `pandas` ke `pyproject.toml` tanpa mengunci ulang, `--check` bernilai 1: kunci **usang**. Aturan praktis: simpan `uv.lock` bersama kode di git, dan kunci ulang tiap kali dependensi berubah.

⚠ AWAS

Kunci hanya menjamin versi **paket Python**. Ia tidak mengunci sistem operasi, versi Python itu sendiri (selain syarat minimum), atau pustaka sistem seperti GDAL. Untuk riset yang sangat sensitif, catat juga versi Python dan sistem Anda di README.

Snakemake: alur analisis yang tahu apa yang perlu diulang

Analisis nyata punya banyak langkah: ambil data, bersihkan, uji, gambar, laporan. Menjalankan semuanya dengan tangan itu rawan salah, dan mengulang semua langkah saat hanya satu berubah itu boros. **Snakemake** menyatakan alur sebagai **aturan** (*rule*): tiap aturan tahu masukan dan keluarannya, dan Snakemake menyusun urutan serta hanya menjalankan yang perlu.

```
# Bab 29 - Alur analisis dengan Snakemake: aturan, ketergantungan, dan jalankan ulang seperlunya
import os
import shutil
import subprocess
from pathlib import Path

kerja = Path(__file__).parent / "_tmp" / "alur"
if kerja.exists():
    shutil.rmtree(kerja)
kerja.mkdir(parents=True)

(kerja / "Snakefile").write_text(r'''rule semua:
    input: "hasil/ringkasan.txt"

rule buat_data:
    output: "data/tinggi.csv"
    run:
        import random
        random.seed(42)
        with open(output[0], "w") as f:
            f.write("plot,tinggi_m\n")
            for p, rata in (("A", 15.0), ("B", 18.0)):
                for _ in range(10):
                    f.write(f"{p},{random.gauss(rata, 2):.2f}\n")

rule ringkas:
    input: "data/tinggi.csv"
    output: "hasil/ringkasan.txt"
    run:
        import csv, statistics
        nilai = {}
        for baris in csv.DictReader(open(input[0])):
            nilai.setdefault(baris["plot"], []).append(float(baris["tinggi_m"]))
        with open(output[0], "w") as f:
            for p, v in sorted(nilai.items()):
             f.write(f"plot {p}: rata-rata {statistics.mean(v):.2f} m (n={len(v)})\n")
''', encoding="utf-8")


def jalankan(*argumen):
     """Jalankan snakemake lalu tampilkan hanya baris yang penting (tanpa jam dan jalur komputer)."""
    hasil = subprocess.run(["snakemake", "--cores", "1", *argumen], cwd=kerja,
                           capture_output=True, text=True, env=dict(os.environ, PYTHONUTF8="1"))
```

```
    for baris in (hasil.stdout + hasil.stderr).splitlines():
        if baris.startswith(("localrule", "rule", "    reason", "Nothing to be done",
"Job stats")) \
             or baris.strip().startswith(("buat_data", "ringkas", "semua", "total")):
            print(baris)


print("=== Jalankan pertama ===")
jalankan()
print((kerja / "hasil" / "ringkasan.txt").read_text(encoding="utf-8").rstrip())

print("=== Jalankan kedua (tidak ada yang berubah) ===")
jalankan()

print("=== Satu baris data ditambah, lalu simulasi (dry-run, -n) ===")
with open(kerja / "data" / "tinggi.csv", "a", encoding="utf-8") as f:
    f.write("A,15.00" + chr(10))
jalankan("-n")
```

```
=== Jalankan pertama ===
Job stats:
buat_data        1
ringkas          1
semua            1
total            3
localrule buat_data:
    reason: Missing output files: data/tinggi.csv
localrule ringkas:
    reason: Missing output files: hasil/ringkasan.txt; Input files updated by another
job: data/tinggi.csv
localrule semua:
    reason: Input files updated by another job: hasil/ringkasan.txt
plot A: rata-rata 14.72 m (n=10)
plot B: rata-rata 18.38 m (n=10)
=== Jalankan kedua (tidak ada yang berubah) ===
Nothing to be done (all requested files are present and up to date).
=== Satu baris data ditambah, lalu simulasi (dry-run, -n) ===
Job stats:
ringkas        1
semua          1
total          2
rule ringkas:
    reason: Updated input files: data/tinggi.csv
rule semua:
    reason: Input files updated by another job: hasil/ringkasan.txt
Job stats:
ringkas        1
semua          1
total          2
        semua
        ringkas
```

Kode ini **tidak dijalankan** saat buku dibuat karena Snakemake belum tersedia sebagai paket jadi untuk Python 3.14. Penulis menjalankannya di lingkungan sementara Python 3.12 (Snakemake 9.27.0), dan keluarannya di atas diambil sekali. Skrip menulis `Snakefile` berisi tiga aturan (`buat_data`, `ringkas`, `semua`) lalu:

1. **Jalan pertama**: semua 3 pekerjaan dijalankan, dengan alasan masing-masing (“Missing output files”). Hasil ringkasan: rata-rata plot A 14,72 m, plot B 18,38 m.
2. **Jalan kedua**: “Nothing to be done”. Tidak ada yang berubah, jadi tidak ada yang diulang.

3. **Data diubah** (satu baris ditambah) lalu dijalankan sebagai simulasi (`-n`, *dry-run*): hanya `ringkas` dan `semua` yang akan diulang, alasannya "Updated input files". `buat_data` tidak diulang. Pada proyek besar, ini menghemat jam komputasi.

Snakemake juga mendukung pencatatan log, lingkungan per aturan, dan menjalankan di klaster; lihat dokumentasi resminya untuk itu (tidak dicoba di sini).

DVC: versi untuk data

Git hebat untuk kode, tetapi kurang cocok untuk berkas data besar. **DVC** (*Data Version Control*) menyimpan data di tempat terpisah ("cache" atau penyimpanan jarak jauh), dan di Git hanya menyimpan **berkas penunjuk** kecil berisi sidik jari (md5) data. Mengganti versi kode otomatis menunjuk ke versi data yang cocok.

```
# Bab 29 - Versi data dengan DVC: Git menyimpan penunjuk kecil, DVC menyimpan datanya
import os
import shutil
import subprocess
from pathlib import Path

kerja = Path(__file__).parent / "_tmp" / "demo_dvc"
if kerja.exists():
    shutil.rmtree(kerja, onexc=lambda fungsi, jalur, galat: (os.chmod(jalur, 0o700),
fungsi(jalur)))   # cache DVC berstatus baca-saja
(kerja / "data").mkdir(parents=True)
env = dict(os.environ, DVC_NO_ANALYTICS="1")      # matikan analitik pemakaian DVC
NL = chr(10)


def dvc(*argumen):
   hasil = subprocess.run(["dvc", *argumen], cwd=kerja, capture_output=True, text=True,
env=env)
    return (hasil.stdout + hasil.stderr).strip()


# --no-scm: demo tanpa Git. Di proyek sungguhan, jalankan di dalam repo Git.
print(dvc("init", "--no-scm").splitlines()[0])

data = kerja / "data" / "ukur.csv"
data.write_text("plot,tinggi_m" + NL + "A,15.2" + NL + "B,18.1" + NL, encoding="utf-8",
newline=NL)
dvc("add", "data/ukur.csv")
print("--- isi data/ukur.csv.dvc (penunjuk kecil yang di-commit ke Git) ---")
print((kerja / "data" / "ukur.csv.dvc").read_text(encoding="utf-8").strip())
print("--- dvc status ---")
print(dvc("status"))

data.write_text(data.read_text(encoding="utf-8") + "C,16.0" + NL, encoding="utf-8",
newline=NL)
print("--- setelah data diubah, dvc status ---")
print(dvc("status").replace(chr(92), "/"))

dvc("add", "data/ukur.csv")
print("--- penunjuk baru ---")
print((kerja / "data" / "ukur.csv.dvc").read_text(encoding="utf-8").strip())
```

```
Initialized DVC repository.
--- isi data/ukur.csv.dvc (penunjuk kecil yang di-commit ke Git) ---
outs:
- md5: f273181e42e11399cec54d22dec892b8
  size: 28
```

```
  hash: md5
  path: ukur.csv
--- dvc status ---
Data and pipelines are up to date.
--- setelah data diubah, dvc status ---
data/ukur.csv.dvc:
    changed outs:
        modified:           data/ukur.csv
--- penunjuk baru ---
outs:
- md5: 822fd4be8cc45f889fdc709553c433cb
  size: 35
  hash: md5
  path: ukur.csv
```

Kode ini **tidak dijalankan** saat buku dibuat karena DVC belum tersedia sebagai paket jadi untuk Python 3.14. Penulis menjalankannya di lingkungan sementara Python 3.12 (DVC 3.67.1) dan keluarannya diambil sekali. Perhatikan alurnya: `dvc add` membuat `ukur.csv.dvc` berisi md5 dan ukuran (28 byte); `dvc status` melaporkan "up to date"; setelah data berubah, status melaporkan `modified`; `dvc add` ulang membuat penunjuk baru (md5 berbeda, ukuran 35 byte). Demo memakai `--no-scm` karena tanpa Git; di proyek sungguhan, jalankan di dalam repo Git lalu `git add` berkas `.dvc`. Skrip juga mematikan analitik pemakaian DVC lewat variabel `DVC_NO_ANALYTICS`, sesuai dokumentasi resminya, supaya tidak ada data terkirim keluar.

Berbagi data dan kode

Riset yang dapat direproduksi baru berguna bila orang lain **menerima paketnya**. Isi paket yang baik:

- **README**: apa isinya, cara menjalankan, dan lisensinya.
- **Lisensi**: tanpa lisensi, orang lain secara hukum tidak boleh memakai karya Anda. Pilihan umum: **MIT** atau **Apache-2.0** untuk kode, **CC BY 4.0** (boleh dipakai dengan menyebut pencipta) untuk data dan teks. Ini ringkasan umum; baca teks lisensinya dan aturan lembaga Anda.
- **Metadata pengutipan**: berkas `CITATION.cff` (Citation File Format) memuat versi, pengarang, tanggal rilis, dan DOI.
- **Checksum**: sidik jari SHA-256 tiap berkas supaya penerima bisa memastikan data tidak berubah.
- **Data sensitif**: jangan bagikan data pribadi (transkrip wawancara asli, koordinat rumah). Anonimkan atau bagikan hanya yang diizinkan (Bab 28).

```
# Bab 29 - Menyiapkan paket untuk dibagikan: README, lisensi, CITATION.cff, dan checksum
import hashlib
import json
import shutil
from pathlib import Path

paket = Path(__file__).parent / "_tmp" / "paket_berbagi"
if paket.exists():
    shutil.rmtree(paket)
(paket / "data").mkdir(parents=True)

# Data tiruan kecil
(paket / "data" / "tinggi.csv").write_text("plot,tinggi_m\nA,15.2\nB,18.1\n",
encoding="utf-8", newline=chr(10))

# README: apa isi paket dan cara menjalankannya
(paket / "README.md").write_text("""# Analisis tinggi pohon (data tiruan)

Isi: `data/tinggi.csv` (data fiktif), `analisis.py`.
Cara menjalankan: `uv run analisis.py`.
Lisensi: kode MIT, data CC BY 4.0.
""", encoding="utf-8", newline=chr(10))
```

```
# CITATION.cff: cara mengutip. Daftar pengarang dan DOI di bawah hanyalah contoh fiktif.
(paket / "CITATION.cff").write_text("""cff-version: 1.2.0
message: "Jika Anda memakai paket ini, mohon kutip seperti berikut."
title: "Analisis tinggi pohon (data tiruan)"
version: 0.1.0
date-released: 2026-10-10
authors:
  - family-names: Contoh
    given-names: Penulis
identifiers:
  - type: doi
    value: 10.5281/zenodo.0000000
""", encoding="utf-8", newline=chr(10))

# Checksum SHA-256: sidik jari berkas supaya orang bisa memastikan datanya tidak berubah
checksum = {}
for berkas in sorted(paket.rglob("*")):
    if berkas.is_file() and berkas.name != "checksum.json":
                                    checksum[berkas.relative_to(paket).as_posix()]    =
hashlib.sha256(berkas.read_bytes()).hexdigest()
(paket / "checksum.json").write_text(json.dumps(checksum, indent=2), encoding="utf-8",
newline=chr(10))

print("Isi paket:", sorted(p.relative_to(paket).as_posix() for p in paket.rglob("*")
if p.is_file()))
for nama, nilai in checksum.items():
    print(f"  {nama:16} sha256 {nilai[:12]}...")

# Verifikasi: ubah data satu karakter, checksum harus berbeda
data = paket / "data" / "tinggi.csv"
asli = hashlib.sha256(data.read_bytes()).hexdigest()
data.write_text(data.read_text(encoding="utf-8").replace("15.2",             "15.3"),
encoding="utf-8", newline=chr(10))
baru = hashlib.sha256(data.read_bytes()).hexdigest()
print("Data berubah, checksum sama?", asli == baru)
```

```
Isi paket: ['CITATION.cff', 'README.md', 'checksum.json', 'data/tinggi.csv']
  CITATION.cff     sha256 2d6ff41e2b08...
  data/tinggi.csv  sha256 2d3d0a182611...
  README.md        sha256 e84f1bdecd00...
Data berubah, checksum sama? False
```

Skrip menyusun paket kecil (README, `CITATION.cff`, data, `checksum.json`) di folder `_tmp/`. Sidik jari tiap berkas ditampilkan 12 karakter pertama. Ketika satu karakter data diubah (15,2 menjadi 15,3), checksum berubah, dan pertanyaan "checksum sama?" dijawab `False`. Itu cara orang memastikan berkas yang mereka unduh sama dengan yang Anda maksud. Daftar pengarang dan DOI dalam `CITATION.cff` contoh ini **fiktif**.

DOI (*digital object identifier*) adalah alamat tetap untuk sebuah karya, sehingga tautan tidak mati bila situs pindah. **Zenodo** adalah repositori terbuka yang memberi DOI pada data dan kode yang diunggah. Menurut dokumentasinya, tiap versi baru adalah catatan terpisah dengan metadata, berkas, dan pengenal tetapnya sendiri, tertaut dengan versi lain; dengan begitu pengutip bisa menunjuk versi persis. Zenodo juga punya integrasi dengan GitHub untuk mengarsipkan rilis (detailnya ada di dokumentasi mereka). **Bab ini tidak mengunggah apa pun ke Zenodo**: DOI dalam contoh adalah angka nol, bukan DOI sungguhan.

Proyek Bersambung I3: analisis yang bisa diulang

Puncak proyek I3: seluruh analisis dibungkus **satu fungsi** yang masukannya hanya benih acak. Bila bekerja benar, dua kali jalan dengan benih sama menghasilkan hasil **identik**, dan itu bisa diperiksa dengan sidik jari.

```
# Bab 29 - Proyek bersambung I3: analisis utuh sebagai fungsi, hasilnya identik
bila diulang
import hashlib
import json

import numpy as np
import pandas as pd
import statsmodels.formula.api as smf
from scipy import stats


def analisis(benih):
    """Seluruh analisis proyek I3 (data tiruan). Satu masukan: benih acak."""
    rng = np.random.default_rng(benih)
    plot_a, plot_b = rng.normal(1.8, 0.6, 30), rng.normal(2.2, 0.7, 30)
    uji = stats.ttest_ind(plot_a, plot_b, equal_var=False)
    d = rng.uniform(10, 60, 60)
    b = 0.12 * d ** 2.4 * np.exp(rng.normal(0, 0.15, 60))
      model = smf.ols("ln_b ~ ln_d", data=pd.DataFrame({"ln_d": np.log(d), "ln_b":
np.log(b)})).fit()
        return {"p_uji_t": round(float(uji.pvalue), 6), "kemiringan_alometrik":
round(float(model.params["ln_d"]), 6)}


def sidik_jari(hasil):
   return hashlib.sha256(json.dumps(hasil, sort_keys=True).encode()).hexdigest()[:12]


h1, h2, h3 = analisis(42), analisis(42), analisis(7)
print("Benih 42, lari pertama :", h1, sidik_jari(h1))
print("Benih 42, lari kedua   :", h2, sidik_jari(h2))
print("Benih 7                :", h3, sidik_jari(h3))
print("Dua lari benih 42 identik?", h1 == h2)
print("Benih berbeda -> hasil berbeda?", h1 != h3)

# Tanpa benih tetap, hasil berubah tiap dijalankan: laporan tidak bisa direproduksi
bebas = [round(float(np.random.default_rng().normal()), 3) for _ in range(2)]
print("Tanpa benih, dua undian berbeda?", bebas[0] != bebas[1])
```

```
Benih 42, lari pertama : {'p_uji_t': 0.000871, 'kemiringan_alometrik': 2.406733}
4896ee7e5c71
Benih 42, lari kedua   : {'p_uji_t': 0.000871, 'kemiringan_alometrik': 2.406733}
4896ee7e5c71
Benih 7          : {'p_uji_t': 5.6e-05, 'kemiringan_alometrik': 2.409982} 89da579235e4
Dua lari benih 42 identik? True
Benih berbeda -> hasil berbeda? True
Tanpa benih, dua undian berbeda? True
```

Dua jalan dengan benih 42 menghasilkan hasil dan sidik jari sama (`4896ee7e5c71`); benih 7 menghasilkan hasil berbeda (`89da579235e4`). Pada baris terakhir, tanpa benih tetap dua undian berbeda: laporan seperti itu mustahil direproduksi. Laporan Quarto di `laporan/laporan.qmd` memakai prinsip yang sama untuk dua bagian proyek (uji beda dan regresi alometrik). Untuk melengkapi seluruh proyek I3, Anda tinggal menambah bagian dari Bab 27 (ikan, ternak, hasil panen terhadap hujan) dan Bab 28 (analisis wawancara) ke laporan itu: tiap bagian cukup satu sel kode yang memanggil fungsi analisis.

Daftar periksa ringkas untuk proyek reproduksi Anda:

1. Data mentah tidak pernah diedit; hasil olahan selalu dibuat oleh kode.
2. Benih acak tetap, dan ditaruh di parameter.
3. `uv.lock` ada dan terbaru.
4. Satu perintah merender seluruh laporan dari nol.
5. README, lisensi, `CITATION.cff`, dan checksum disertakan.
6. Seseorang lain (atau Anda di komputer lain) sudah mencoba menjalankannya.

Latihan

1. Di `laporan/laporan.qmd`, tambahkan parameter `ukuran_efek` yang dipakai pada rata-rata plot B (misalnya `2.2` menjadi `1.8 + ukuran_efek`), lalu render dengan `-P ukuran_efek:0.1` dan `-P ukuran_efek:0.8`. Apa yang terjadi pada p-value?
2. Ubah `06_berbagi_data_kode.py` agar checksum memakai algoritma lain dari `hashlib` (misalnya `sha512`). Apa bedanya pada panjang sidik jari?
3. Tambahkan satu aturan baru ke Snakefile pada `04_snakemake.py`, misalnya `grafik` yang bergantung pada `data/tinggi.csv`. Jalankan dan lihat berapa pekerjaan dalam "Job stats".
4. **Tantangan:** buat satu perintah (skrip Python) yang merender laporan, menghitung checksum hasilnya, dan membandingkannya dengan render sebelumnya. Apakah dua render dengan parameter sama menghasilkan HTML yang sama bit demi bit? Mengapa mungkin tidak?

Ringkasan Bab

- **Reproduksi** = data dan kode sama, angka sama. **Replikasi** = data baru, kesimpulan sama.
- **Notebook** rapi: urut dari atas, pengaturan di awal, benih tetap, "restart and run all" tanpa error.
- **Quarto** merender laporan dari kode; **parameter** (`-P`) mengubah hasil tanpa mengubah berkas; **Jupyter Book 2** dibangun di atas MyST.
- **`uv lock`** mengunci versi paket di `uv.lock`; `--check` mengungkap kunci yang usang.
- **Snakemake** menjalankan hanya langkah yang perlu; **DVC** menyimpan penunjuk data di Git. Keduanya belum ada sebagai paket jadi untuk Python 3.14 dan dijalankan di Python 3.12.
- **Berbagi**: README, lisensi, `CITATION.cff`, checksum, DOI (Zenodo); jangan membagikan data pribadi.

Penutup Seri I3

Selamat! Anda menyelesaikan Seri I3. Anda sekarang bisa merangkum data dan membuat selang kepercayaan, menguji beda kelompok dan membaca p-value dengan benar, membangun model regresi dan memeriksanya, meramal deret waktu, menyimulasikan ketidakpastian, menganalisis wawancara petani, nelayan, dan peternak, serta membungkus semuanya dalam laporan yang bisa diulang siapa saja.

Tetapi analisis kita makin berat dan berulang: fungsi yang sama dipanggil ratusan kali, berkas data makin besar, dan kode makin sering perlu dibungkus agar tak diulang-ulang. Seri I4 memperkuat kode dengan fitur bahasa tingkat lanjut: iterator dan generator, decorator, context manager, type hint, validasi dengan Pydantic, dan async.

Selanjutnya, di Bab 30: iterator dan generator, cara memproses berkas data besar sepotong demi sepotong tanpa menghabiskan memori.

Bab 30 — Iterator dan Generator: Membaca Satu per Satu, Tanpa Menumpuk

Seri I4 — Bahasa Python Tingkat Lanjut

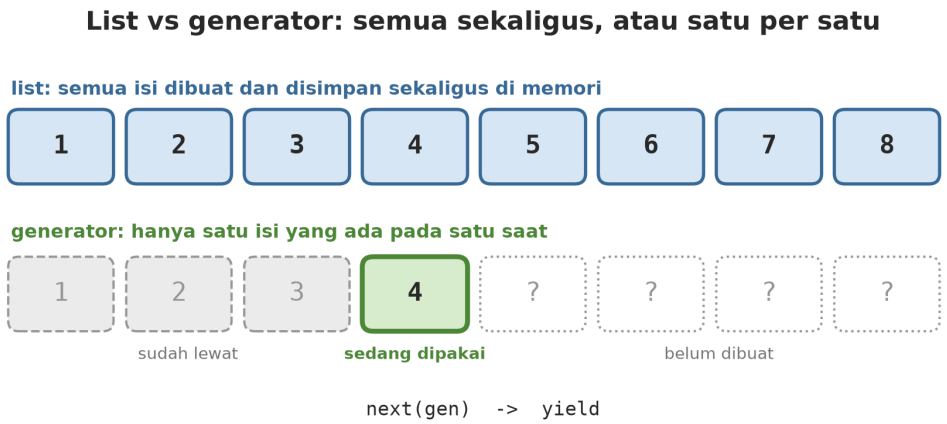
Di bab ini, Anda akan:

- Memahami apa yang sebenarnya dilakukan perulangan `for` (protokol iterasi)
- Membuat iterator sendiri dan **generator** dengan `yield`
- Memakai ekspresi generator dan `itertools` untuk data "malas" (*lazy*)
- Membuktikan bahwa generator menghemat memori
- Membaca berkas data besar baris demi baris: proyek Seri I4 dimulai

Semua skrip bab ini ada di folder `kode/bab30/`. Di versi web, Anda juga bisa mengunduh semuanya sekaligus (zip).

Seri I1 sampai I3 membuat kode Anda rapi dan hasilnya bisa dipertanggungjawabkan. Di Seri I4, kita memperkuat **bahasa Python itu sendiri**: fitur-fitur yang membuat kode lebih hemat, lebih ringkas, dan lebih tahan banting. Kita mulai dari yang paling sering dipakai diam-diam: perulangan.

Bayangkan seorang mandor yang mencatat panen. Cara pertama: ia menunggu **semua** karung selesai diangkut, menumpuknya di gudang, baru mulai mencatat. Cara kedua: ia mencatat **satu karung setiap kali ada yang datang**, lalu karung itu dilepas. Cara pertama butuh gudang besar. Cara kedua hanya butuh meja kecil. Bab ini tentang cara kedua.



Gambar 0.1: List menyimpan semua isi sekaligus; generator hanya menyimpan satu isi pada satu saat.

Apa yang dikerjakan `for`: protokol iterasi

Perulangan `for` bekerja pada apa pun yang bisa **diiterasi** (*iterable*): list, tuple, dict, string, berkas. Di baliknya ada dua langkah: Python meminta **iterator** (semacam penanda bacaan) lewat `iter()`, lalu terus meminta isi berikutnya lewat `next()` sampai muncul sinyal `StopIteration`.

```
# Bab 30 - Apa yang sebenarnya dikerjakan perulangan for
kolam = ["A", "B", "C"]

pengulang = iter(kolam)       # minta "penunjuk bacaan" dari list
```

```
print(type(pengulang).__name__)
print(next(pengulang))
print(next(pengulang))
print(next(pengulang))

try:
    next(pengulang)           # sudah habis
except StopIteration:
    print("StopIteration: tidak ada isi lagi")

# for melakukan hal yang sama, dan menangkap StopIteration untuk Anda
for nama in kolam:
    print("Kolam", nama)
```

```
list_iterator
A
B
C
StopIteration: tidak ada isi lagi
Kolam A
Kolam B
Kolam C
```

- `iter(kolam)` memberi objek bertipe `list_iterator`, penanda bacaan atas list itu.
- Tiga panggilan `next()` memberi `A`, `B`, `C`. Panggilan keempat melempar `StopIteration`, artinya "habis".
- Perulangan `for` melakukan persis itu, dan menangkap `StopIteration` untuk Anda, sehingga Anda tidak pernah melihatnya.

> **i** INGAT
>
> **Iterable** = benda yang bisa diminta iterator-nya (`iter()`). **Iterator** = benda yang bisa diminta isi berikutnya (`next()`). Iterator sekali jalan: setelah habis, ia tidak mundur.

Iterator buatan sendiri dengan kelas

Agar objek buatan Anda bisa dipakai di `for`, kelasnya cukup punya dua metode: `__iter__` (mengembalikan iterator) dan `__next__` (memberi isi berikutnya, atau melempar `StopIteration`).

```
# Bab 30 - Membuat iterator sendiri dengan kelas
class HitungMundur:
    """Menghitung mundur hari sampai panen."""

    def __init__(self, hari):
        self.hari = hari

    def __iter__(self):
        return self

    def __next__(self):
        if self.hari <= 0:
            raise StopIteration
        self.hari -= 1
        return self.hari + 1


for sisa in HitungMundur(3):
    print("Sisa", sisa, "hari")
```

```
hitung = HitungMundur(2)
print(list(hitung))
print(list(hitung))   # sudah habis: kosong
```

```
Sisa 3 hari
Sisa 2 hari
Sisa 1 hari
[2, 1]
[]
```

Perhatikan bagian akhirnya: `list(hitung)` pertama menghasilkan `[2, 1]`, tetapi yang kedua `[]`. Iterator-nya sudah habis dibaca. Ini sifat dasar iterator, dan akan muncul lagi sebagai jebakan di bawah.

Generator: cara pendek membuat iterator

Menulis kelas hanya untuk menghitung mundur itu repot. Python punya jalan pintas: fungsi biasa yang memakai **`yield`** menjadi **fungsi generator**. `yield` berarti "berikan nilai ini, lalu *jeda* di sini sampai diminta lagi".

```
# Bab 30 - Fungsi generator dengan yield
def hitung_mundur(hari):
    print("  (mulai)")
    while hari > 0:
        yield hari
        print("  (lanjut setelah yield)")
        hari -= 1
    print("  (selesai)")


gen = hitung_mundur(2)
print("Generator dibuat, belum ada yang jalan")
print("nilai pertama:", next(gen))
print("nilai kedua:", next(gen))
try:
    next(gen)
except StopIteration:
    print("StopIteration")
```

```
Generator dibuat, belum ada yang jalan
  (mulai)
nilai pertama: 2
  (lanjut setelah yield)
nilai kedua: 1
  (lanjut setelah yield)
  (selesai)
StopIteration
```

Telusuri urutannya, karena di sinilah letak sihirnya:

1. `hitung_mundur(2)` **belum menjalankan apa-apa**. Hanya terbentuk objek generator. Itu sebabnya "(mulai)" belum tercetak saat "Generator dibuat".
2. `next(gen)` pertama menjalankan fungsi dari awal sampai `yield 2`, lalu berhenti. Tercetak "(mulai)" lalu nilainya, 2.
3. `next(gen)` kedua melanjutkan **tepat setelah** `yield` tadi: "(lanjut setelah yield)", `hari` turun menjadi 1, lalu `yield 1`.
4. `next(gen)` ketiga melanjutkan, `while` selesai, tercetak "(selesai)", dan fungsi berakhir, sehingga muncul `StopIteration`.

Generator adalah fungsi yang **mengingat posisinya**. Semua variabel lokalnya tetap hidup di antara dua `yield`.

Ekspresi generator dan bukti hemat memori

Seperti *list comprehension* (Bab 9), ada versi generatornya: ganti kurung siku dengan kurung biasa. Hasilnya bukan list, melainkan generator yang menghitung isi **saat diminta**.

```
# Bab 30 - Ekspresi generator dan penghematan memori
import sys
import tracemalloc

kuadrat_list = [n * n for n in range(1_000_000)]   # kurung siku: semua dibuat sekarang
kuadrat_gen = (n * n for n in range(1_000_000))    # kurung biasa: dibuat satu per satu

print("jumlah list     :", sum(kuadrat_list))
print("jumlah generator:", sum(kuadrat_gen))
del kuadrat_list

# Ukuran objek generator tetap kecil, berapa pun banyaknya isi
kecil = (n for n in range(10))
besar = (n for n in range(10_000_000))
print("ukuran sama untuk 10 dan 10 juta isi:", sys.getsizeof(kecil) ==
sys.getsizeof(besar))


def puncak_memori(pembuat):
    tracemalloc.start()
    hasil = sum(pembuat())
    _, puncak = tracemalloc.get_traced_memory()
    tracemalloc.stop()
    return hasil, puncak


_, puncak_list = puncak_memori(lambda: [n for n in range(500_000)])
_, puncak_gen = puncak_memori(lambda: (n for n in range(500_000)))
print("list memakai memori puncak lebih dari 100x generator:", puncak_list > 100
* puncak_gen)
```

```
jumlah list     : 333332833333500000
jumlah generator: 333332833333500000
ukuran sama untuk 10 dan 10 juta isi: True
list memakai memori puncak lebih dari 100x generator: True
```

- Kedua cara menghasilkan jumlah yang sama, `333332833333500000`: hasilnya tidak berubah, hanya cara kerjanya.
- Objek generator berukuran sama (`sys.getsizeof` sama) baik untuk 10 isi maupun 10 juta isi. Ia tidak menyimpan isinya, hanya "resep" cara membuatnya.
- `tracemalloc` (alat bawaan untuk mencatat pemakaian memori) menunjukkan bahwa memori puncak list lebih dari 100 kali memori puncak generator untuk 500.000 angka.

Perhatikan bahwa program ini **tidak mencetak angka byte**, hanya perbandingan (`True`). Ukuran persisnya berubah antar versi Python dan antar komputer. Perbandingan "jauh lebih besar" tetap benar di mana pun.

> TIPS
>
> Kalau Anda hanya perlu **melewati** data sekali (menjumlahkan, mencari yang terbesar, menyaring), jangan buat list dulu. `sum(x * x for x in data)` lebih hemat dan tidak butuh kurung siku.

Generator lebih jauh: `itertools`, `yield from`, dan jebakan

```
# Bab 30 - itertools, yield from, dan jebakan generator habis
from itertools import islice


def bilangan_tak_hingga():
    n = 1
    while True:          # tidak pernah berhenti, tetapi aman karena malas
        yield n
        n += 1


print(list(islice(bilangan_tak_hingga(), 5)))   # ambil 5 saja


def semua_kolam():
    yield from ["A1", "A2"]
    yield from ["B1"]


print(list(semua_kolam()))

# Jebakan: generator hanya bisa dibaca SEKALI
panen = (kg for kg in [245.5, 310.0, 198.5])
print("pertama:", sum(panen))
print("kedua  :", sum(panen))   # kosong, bukan error
```

```
[1, 2, 3, 4, 5]
['A1', 'A2', 'B1']
pertama: 754.0
kedua  : 0
```

- Generator `bilangan_tak_hingga()` tidak pernah berhenti, tetapi aman: ia hanya bekerja bila diminta. `itertools.islice(..., 5)` mengambil lima isi pertama lalu berhenti meminta.
- `yield from` meneruskan semua isi dari iterable lain, jadi `semua_kolam()` menggabungkan dua daftar.
- **Jebakan**: `sum(panen)` pertama memberi 754,0, tetapi yang kedua memberi `0`. Generator hanya bisa dibaca **sekali**. Tidak ada galat, hanya hasil kosong, jadi bug ini licin. Bila data perlu dibaca dua kali, buat generator baru atau simpan ke list.

Merangkai generator menjadi pipa

Kekuatan terbesar generator: beberapa generator bisa disambung seperti pipa air. Setiap tahap mengerjakan satu tugas kecil, dan data mengalir satu isi per satu isi.

```
# Bab 30 - Merangkai generator menjadi pipa
catatan = ["kolam,A,540.5", "kolam,B,rusak", "kolam,C,498.0", "", "kolam,D,610.0"]


def tanpa_kosong(baris):
    for b in baris:
        if b.strip():
            yield b


def pecah(baris):
    for b in baris:
        yield b.split(",")
```

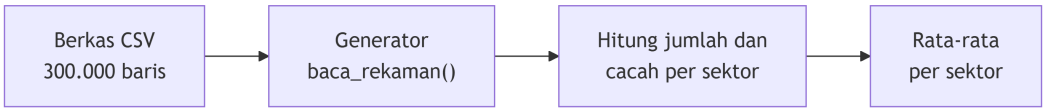
```
def hanya_angka(rekaman):
    for sektor, unit, nilai in rekaman:
        try:
            yield unit, float(nilai)
        except ValueError:
            print("  dilewati:", unit, nilai)


pipa = hanya_angka(pecah(tanpa_kosong(catatan)))
print("pipa dibuat, belum ada yang diproses")
for unit, bobot in pipa:
    print(unit, bobot)
```

```
pipa dibuat, belum ada yang diproses
A 540.5
  dilewati: B rusak
C 498.0
D 610.0
```

Saat `pipa` dibuat, belum ada yang diproses (itu sebabnya "pipa dibuat, belum ada yang diproses" tampil lebih dulu). Pekerjaan baru terjadi di `for`. Baris kosong disaring oleh `tanpa_kosong`, baris `kolam,B,rusak` dilewati oleh `hanya_angka` (dengan pesan "dilewati"), dan sisanya mengalir sampai ke ujung. Tidak ada list perantara.

Proyek Bersambung I4: membaca berkas data besar

Proyek Seri I4 menguatkan kode lapangan Anda, bab demi bab. Langkah pertama: membaca berkas catatan lapangan yang besar. Skrip di bawah membuat berkas tiruan (angka contoh, fiktif) lima sektor (hutan, kebun, sawah, kolam, kandang) berisi 300.000 baris, besarnya lebih dari 5 MB, lalu membacanya dengan generator.



Alur proyek: berkas besar dibaca satu baris per satu, dirangkum, tanpa menyimpan semuanya.

```
# Bab 30 - Proyek bersambung I4: membaca berkas besar dengan generator
import random
import tracemalloc
from pathlib import Path

folder = Path(__file__).parent / "_tmp"
folder.mkdir(exist_ok=True)
berkas = folder / "lapangan_besar.csv"

# 1) Buat berkas tiruan (angka contoh, fiktif)
rentang = {"hutan": (20, 60), "kebun": (150, 350), "sawah": (3000, 7000),
           "kolam": (400, 700), "kandang": (300, 450)}
acak = random.Random(42)
with open(berkas, "w", encoding="utf-8") as f:
    f.write("sektor,unit,hari,nilai\n")
    for i in range(300_000):
        sektor = list(rentang)[i % 5]
        bawah, atas = rentang[sektor]
               f.write(f"{sektor},{sektor[0].upper()}{i % 7 + 1},{i // 5 + 1},
```

```
{acak.uniform(bawah, atas):.1f}\n")
print("ukuran berkas lebih dari 5 MB:", berkas.stat().st_size > 5_000_000)


# 2) Generator membaca baris demi baris
def baca_rekaman(jalur):
    with open(jalur, encoding="utf-8") as f:
        next(f)                      # lewati judul kolom
        for baris in f:
            sektor, unit, hari, nilai = baris.strip().split(",")
            yield sektor, unit, int(hari), float(nilai)


def rata_rata_per_sektor(jalur):
    jumlah, cacah = {}, {}
    for sektor, _, _, nilai in baca_rekaman(jalur):
        jumlah[sektor] = jumlah.get(sektor, 0) + nilai
        cacah[sektor] = cacah.get(sektor, 0) + 1
    return {s: jumlah[s] / cacah[s] for s in sorted(jumlah)}, sum(cacah.values())


tracemalloc.start()
rata, banyak = rata_rata_per_sektor(berkas)
_, puncak_gen = tracemalloc.get_traced_memory()
tracemalloc.stop()
print("rekaman dibaca:", banyak)
for sektor, nilai in rata.items():
    print(f"{sektor:8} rata-rata {nilai:8.1f}")

# 3) Pembanding: baca semua ke list dulu
tracemalloc.start()
semua = list(baca_rekaman(berkas))
_, puncak_list = tracemalloc.get_traced_memory()
tracemalloc.stop()
print("jumlah rekaman di list:", len(semua))
print("cara list memakai memori puncak lebih dari 20x generator:", puncak_list > 20
* puncak_gen)
```

```
ukuran berkas lebih dari 5 MB: True
rekaman dibaca: 300000
hutan    rata-rata     40.0
kandang  rata-rata    375.2
kebun    rata-rata    250.1
kolam    rata-rata    550.2
sawah    rata-rata   4996.2
jumlah rekaman di list: 300000
cara list memakai memori puncak lebih dari 20x generator: True
```

Telusuri hasilnya:

- Berkas dibuat dengan benih acak tetap (`random.Random(42)`), sehingga angkanya sama setiap dijalankan. Berkasnya ditaruh di `_tmp/`, bukan di folder skrip.
- Tiap sektor mendapat 60.000 baris (300.000 dibagi 5). Rata-rata per sektor mengikuti rentang yang kita tetapkan: sawah sekitar 5.000, kolam sekitar 550, dan seterusnya.
- Generator `baca_rekaman` memakai `with open(...)` (Bab 13): berkas ditutup otomatis. Perulangan `for baris in f` sendiri sudah membaca berkas **baris demi baris**, tanpa memuat semuanya.
- Pembanding: `list(baca_rekaman(berkas))` menyimpan semua 300.000 rekaman, dan memori puncaknya lebih dari 20 kali lipat. Karena hanya perbandingan yang dicetak, keluarannya stabil.

BAGIAN TEKNIS (boleh dilewati)

`sys.getsizeof` hanya mengukur objek itu sendiri, **bukan** isi yang dirujuknya. Untuk list, ukurannya ikut bertambah karena list memang menyimpan penunjuk ke setiap isi. Untuk generator, ukurannya tetap karena ia hanya menyimpan keadaan terakhirnya (posisi di fungsi dan variabel lokal). Karena itu bab ini memakai `tracemalloc` untuk membandingkan pemakaian memori sungguhan, dan hanya mencetak perbandingannya.

Latihan

1. Tulis fungsi generator `genap_sampai(n)` yang memberikan bilangan genap dari 2 sampai `n`. Cetak `list(genap_sampai(10))`.
2. Ubah ekspresi generator pada `04_ekspresi_generator.py` supaya menjumlahkan hanya kuadrat bilangan ganjil. Apakah jumlahnya berubah? Mengapa?
3. Tambahkan satu tahap pada pipa di `06_pipa_generator.py`: generator `di_atas(rekaman, batas)` yang hanya meloloskan bobot di atas 500. Sambungkan di ujung pipa.
4. **Tantangan:** ubah `07_proyek_baca_besar.py` supaya juga menghitung nilai **terbesar** tiap sektor dalam sekali baca, tanpa menyimpan seluruh rekaman.

Ringkasan Bab

- `for` memakai **protokol iterasi**: `iter()` untuk meminta iterator, `next()` untuk isi berikutnya, `StopIteration` bila habis.
- **Generator** dibuat dengan `yield`; fungsinya **berjeda** dan melanjutkan dari posisi terakhir.
- **Ekspresi generator** `(x for x in ...)` menghitung saat diminta, ukurannya tetap kecil, dan menghemat memori.
- Generator hanya bisa dibaca **sekali**; hasil kosong pada pembacaan kedua adalah jebakan klasik.
- Beberapa generator bisa **dirangkai menjadi pipa** tanpa list perantara.
- Proyek I4: berkas besar dibaca dengan generator, 300.000 rekaman dirangkum tanpa menumpuk di memori.

Selanjutnya, di Bab 31: decorator dan closure, cara "membungkus" fungsi dengan tambahan (log, coba-ulang) tanpa mengubah isinya.

Bab 31 — Decorator dan Closure: Membungkus Fungsi dengan Tambahan

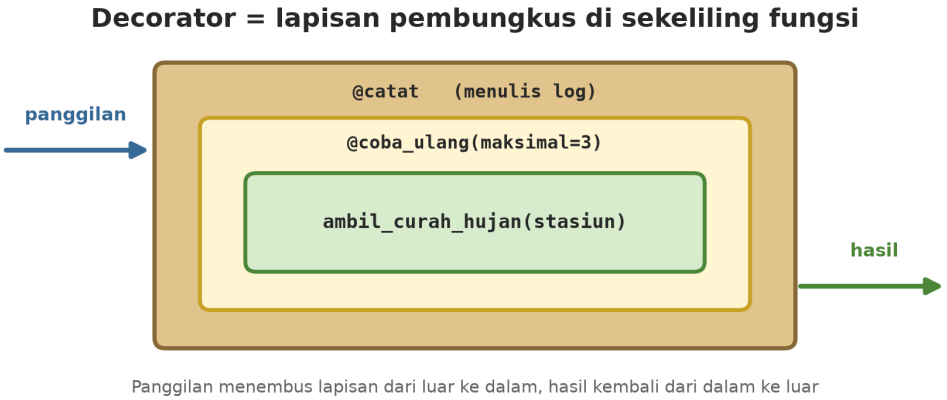
Seri I4 — Bahasa Python Tingkat Lanjut

Di bab ini, Anda akan:

- Memahami bahwa fungsi adalah **objek** yang bisa disimpan dan dikirim
- Membuat **closure**: fungsi yang mengingat variabel di sekitarnya
- Menulis **decorator**, termasuk yang memakai argumen dan `functools.wraps`
- Mengenali jebakan umum dan decorator siap pakai (`functools.cache`)
- Menambah log dan coba-ulang pada fungsi proyek tanpa mengubah isinya

Semua skrip bab ini ada di folder `kode/bab31/`. Di versi web, Anda juga bisa mengunduh semuanya sekaligus (zip).

Di Bab 30, proyek Anda belajar membaca berkas besar. Sekarang bayangkan Anda punya sepuluh fungsi yang mengambil data dari berbagai tempat, dan Anda ingin: (1) mencatat di log setiap kali fungsi dimulai dan selesai, (2) mengulang otomatis bila jaringan putus. Menyalin kode log dan ulang ke sepuluh fungsi itu membosankan dan rawan salah. Lebih baik "membungkus" setiap fungsi dengan lapisan tambahan, seperti membungkus kado: isinya sama, tetapi ada lapisan di luarnya. Itulah **decorator**.



Gambar 0.1: Decorator = lapisan pembungkus di sekeliling fungsi: panggilan menembus lapisan, hasil kembali keluar.

Fungsi adalah objek

Pondasi decorator: di Python, fungsi hanyalah salah satu jenis objek, seperti angka atau teks. Ia bisa diberi nama lain, disimpan di dict, dan dikirim ke fungsi lain.

```
# Bab 31 - Fungsi adalah objek
def biomassa_kg(jumlah, bobot_g):
    return jumlah * bobot_g / 1000


alat = biomassa_kg            # menyalin "nama", bukan memanggil
print(alat(800, 540.5))
```

```
print(alat is biomassa_kg)
print(type(biomassa_kg).__name__)

# Fungsi boleh disimpan di dict dan dipilih lewat nama
rumus = {
    "biomassa": biomassa_kg,
    "luas": lambda panjang, lebar: panjang * lebar,
}
print(rumus["luas"](20, 15))


# Fungsi boleh dikirim ke fungsi lain
def terapkan(fungsi, *argumen):
    return fungsi(*argumen)


print(terapkan(biomassa_kg, 1000, 500))
```

```
432.4
True
function
300
500.0
```

- `alat = biomassa_kg` (tanpa tanda kurung) **tidak memanggil** fungsi; ia hanya menambah satu nama untuk objek yang sama. Buktinya `alat is biomassa_kg` bernilai `True`.
- Tipe objeknya `function`.
- Di dict `rumus`, fungsi dipilih lewat nama: `rumus["luas"](20, 15)` menghasilkan 300.
- `terapkan(biomassa_kg, 1000, 500)` mengirim fungsi sebagai bahan. `*argumen` mengumpulkan sisa argumen, dan `fungsi(*argumen)` membukanya kembali. Hasilnya 500,0.

Closure: fungsi yang mengingat

Fungsi boleh dibuat **di dalam** fungsi lain. Fungsi dalam itu bisa memakai variabel milik fungsi luar, bahkan setelah fungsi luar selesai. Gabungan "fungsi + variabel yang diingatnya" disebut **closure**.

```
# Bab 31 - Closure: fungsi yang mengingat
def buat_penghitung(awal=0):
    total = awal

    def tambah(jumlah):
        nonlocal total          # tanpa ini, total dianggap variabel baru
        total += jumlah
        return total

    return tambah


panen_kebun = buat_penghitung()
panen_kolam = buat_penghitung(100)
print(panen_kebun(10))
print(panen_kebun(5))
print(panen_kolam(1))           # penghitung lain, ingatannya terpisah

isi = panen_kebun.__closure__[0].cell_contents
print("isi ingatan panen_kebun:", isi)
```

```
10
15
```

```
101
isi ingatan panen_kebun: 15
```

- `buat_penghitung()` mengembalikan fungsi `tambah`. Fungsi itu "membawa" variabel `total`.
- `panen_kebun` dan `panen_kolam` punya ingatan **terpisah**: 10, lalu 15 untuk kebun, sementara kolam mulai dari 100 dan menjadi 101.
- **`nonlocal total`** wajib bila fungsi dalam ingin **mengubah** variabel luar. Tanpa itu, `total += jumlah` dianggap membuat variabel lokal baru dan memicu galat (ingat bahasan scope di Bab 10).
- `__closure__[0].cell_contents` memperlihatkan "ingatan" itu secara langsung: 15.

Decorator pertama

Decorator adalah fungsi yang **menerima fungsi** dan **mengembalikan fungsi baru** (pembungkus). Tanda `@` hanyalah singkatan.

```
# Bab 31 - Decorator pertama (belum sempurna)
def catat(fungsi):
    def pembungkus(*args, **kwargs):
        print(f"memanggil {fungsi.__name__}")
        hasil = fungsi(*args, **kwargs)
        print(f"selesai, hasil = {hasil}")
        return hasil
    return pembungkus


@catat
def biomassa_kg(jumlah, bobot_g):
    """Biomassa kolam dalam kg."""
    return jumlah * bobot_g / 1000


biomassa_kg(800, 540.5)

# @catat sama saja dengan: biomassa_kg = catat(biomassa_kg)
print(biomassa_kg.__name__)     # identitas aslinya hilang
print(biomassa_kg.__doc__)
```

```
memanggil biomassa_kg
selesai, hasil = 432.4
pembungkus
None
```

- `@catat` di atas `def biomassa_kg` artinya sama dengan `biomassa_kg = catat(biomassa_kg)`.
- Setiap pemanggilan sekarang melewati `pembungkus`: mencetak "memanggil", menjalankan fungsi asli, mencetak hasil, lalu mengembalikannya.
- `*args, **kwargs` membuat pembungkus menerima **argumen apa pun** lalu meneruskannya apa adanya.
- Ada cacat: `biomassa_kg.__name__` menjadi `pembungkus` dan docstring-nya hilang (`None`). Fungsi kehilangan identitas.

`functools.wraps`: menjaga identitas

Obatnya satu baris: `@wraps(fungsi)` pada pembungkus.

```
# Bab 31 - functools.wraps menjaga identitas fungsi
from functools import wraps


def catat(fungsi):
```

```
    @wraps(fungsi)
    def pembungkus(*args, **kwargs):
        print(f"memanggil {fungsi.__name__}")
        return fungsi(*args, **kwargs)
    return pembungkus


@catat
def biomassa_kg(jumlah, bobot_g):
    """Biomassa kolam dalam kg."""
    return jumlah * bobot_g / 1000


print(biomassa_kg(800, 540.5))
print(biomassa_kg.__name__)
print(biomassa_kg.__doc__)
print(biomassa_kg.__wrapped__ is not None)
```

```
memanggil biomassa_kg
432.4
biomassa_kg
Biomassa kolam dalam kg.
True
```

Sekarang `__name__` dan `__doc__` kembali ke aslinya, dan `__wrapped__` menyimpan rujukan ke fungsi sebelum dibungkus. Biasakan: **setiap decorator selalu memakai `@wraps`**. Ini penting untuk pesan galat, dokumentasi, dan alat seperti pytest.

Decorator yang punya argumen

Bila decorator perlu pengaturan (misalnya "ulangi 3 kali"), dibutuhkan satu lapisan lagi: fungsi luar menerima pengaturan dan mengembalikan decorator yang sesungguhnya.

```
# Bab 31 - Decorator yang punya argumen
from functools import wraps


def ulangi(kali):
    """Pabrik decorator: mengulang pemanggilan sebanyak `kali`."""
    def decorator(fungsi):
        @wraps(fungsi)
        def pembungkus(*args, **kwargs):
            hasil = None
            for i in range(kali):
                hasil = fungsi(*args, **kwargs)
            return hasil
        return pembungkus
    return decorator


@ulangi(3)
def sapa(nama):
    print("Halo,", nama)


sapa("Pak Tani")


# Menumpuk decorator: yang paling dekat dengan def dipasang lebih dulu
def tebal(f):
```

```
    return lambda: "<b>" + f() + "</b>"


def miring(f):
    return lambda: "<i>" + f() + "</i>"


@tebal
@miring
def judul():
    return "Panen"


print(judul())
```

```
Halo, Pak Tani
Halo, Pak Tani
Halo, Pak Tani
<b><i>Panen</i></b>
```

Tiga tingkat: `ulangi(3)` → mengembalikan `decorator` → yang menerima `sapa` dan mengembalikan `pembungkus`. `@ulangi(3)` menjalankan tingkat pertama langsung, hasilnya dipasang sebagai decorator.

Bagian bawah menunjukkan **decorator bertumpuk**: yang paling dekat dengan `def` dipasang lebih dulu. `@miring` membungkus `judul` lebih dulu, lalu `@tebal` membungkus hasilnya, jadi tag `<b>` berada di luar dan `<i>` di dalam: `<b><i>Panen</i></b>`.

Decorator siap pakai dan jebakan

Pustaka standar menyediakan banyak decorator. Salah satu yang paling berguna adalah `functools.cache`, yang mengingat hasil agar pemanggilan dengan argumen sama tidak dihitung ulang.

```
# Bab 31 - Decorator siap pakai: functools.cache
from functools import cache

panggilan = 0


@cache
def harga_pakan_per_kg(kode):
    global panggilan
    panggilan += 1             # menghitung berapa kali isi fungsi benar-benar jalan
    return {"P1": 8500, "P2": 9200}[kode]


print(harga_pakan_per_kg("P1"))
print(harga_pakan_per_kg("P1"))   # diambil dari ingatan
print(harga_pakan_per_kg("P2"))
print("isi fungsi berjalan:", panggilan, "kali")
```

```
8500
8500
9200
isi fungsi berjalan: 2 kali
```

Tiga kali dipanggil, tetapi isi fungsi hanya berjalan **2 kali**: pemanggilan kedua untuk `"P1"` diambil dari ingatan. Cocok untuk fungsi yang lambat dan hasilnya selalu sama untuk masukan yang sama. Jangan dipakai untuk fungsi yang hasilnya berubah (misalnya membaca harga hari ini).

Sekarang jebakan paling umum: **lupa tanda kurung** pada decorator berargumen.

```
# Bab 31 - Jebakan: decorator berargumen dipasang tanpa tanda kurung
def ulangi(kali):
    def decorator(fungsi):
        def pembungkus(*args, **kwargs):
            for _ in range(kali):
                fungsi(*args, **kwargs)
        return pembungkus
    return decorator


@ulangi            # seharusnya @ulangi(3)
def sapa():
    print("Halo")


sapa()
```

```
Traceback (most recent call last):
  File "07_lupa_kurung.py", line 16, in <module>
    sapa()
    ~~~~^^
TypeError: ulangi.<locals>.decorator() missing 1 required positional argument: 'fungsi'
```

`@ulangi` tanpa `(3)` berarti `sapa = ulangi(sapa)`. Argumen `kali` jadi berisi fungsi `sapa`, dan yang dikembalikan adalah `decorator`, bukan pembungkus. Saat `sapa()` dipanggil, sebenarnya `decorator()` yang jalan, dan ia kekurangan argumen `fungsi`. Pesan galatnya (`ulangi.<locals>.decorator() missing 1 required positional argument: 'fungsi'`) adalah petunjuk khasnya.

> ⚠ AWAS
>
> Decorator berargumen **selalu** ditulis dengan kurung, walau argumennya kosong atau bawaan. Bila galatnya berbunyi `<locals>.decorator() missing ... 'fungsi'`, hampir pasti Anda lupa kurung.

Proyek Bersambung I4: log dan coba-ulang

Sekarang gabungkan: dua decorator, `catat` (log) dan `coba_ulang` (mengulang bila `ConnectionError`), dipasang pada fungsi pengambil curah hujan. Jaringan tiruan: gagal dua kali, baru berhasil. Log memakai modul `logging` (Bab 19) dengan format tanpa jam, agar keluaran stabil. Jeda antar-percobaan sengaja kecil (0,01 detik).

```
# Bab 31 - Proyek bersambung I4: decorator untuk log dan coba-ulang
import logging
import sys
import time
from functools import wraps

logging.basicConfig(stream=sys.stdout,    level=logging.INFO,    format="%(levelname)s
%(message)s")
log = logging.getLogger("lapangan")


def catat(fungsi):
    @wraps(fungsi)
    def pembungkus(*args, **kwargs):
        log.info("mulai %s", fungsi.__name__)
        hasil = fungsi(*args, **kwargs)
        log.info("selesai %s", fungsi.__name__)
        return hasil
    return pembungkus
```

```
def coba_ulang(maksimal=3, jeda=0.01, galat=(ConnectionError,)):
    def decorator(fungsi):
        @wraps(fungsi)
        def pembungkus(*args, **kwargs):
            for percobaan in range(1, maksimal + 1):
                try:
                    return fungsi(*args, **kwargs)
                except galat as e:
                 log.warning("percobaan %d dari %d gagal: %s", percobaan, maksimal, e)
                    if percobaan == maksimal:
                        raise
                    time.sleep(jeda)
        return pembungkus
    return decorator


gangguan = {"sisa": 2}          # jaringan tiruan: gagal dua kali, lalu normal


@catat
@coba_ulang(maksimal=3)
def ambil_curah_hujan(stasiun):
    if gangguan["sisa"] > 0:
        gangguan["sisa"] -= 1
        raise ConnectionError("sambungan terputus")
    return {"stasiun": stasiun, "mm": 12.5}


print(ambil_curah_hujan("S1"))

gangguan["sisa"] = 5            # sekarang gangguan lebih panjang dari batas coba
try:
    ambil_curah_hujan("S2")
except ConnectionError as e:
    print("menyerah:", e)
```

```
INFO mulai ambil_curah_hujan
WARNING percobaan 1 dari 3 gagal: sambungan terputus
WARNING percobaan 2 dari 3 gagal: sambungan terputus
INFO selesai ambil_curah_hujan
{'stasiun': 'S1', 'mm': 12.5}
INFO mulai ambil_curah_hujan
WARNING percobaan 1 dari 3 gagal: sambungan terputus
WARNING percobaan 2 dari 3 gagal: sambungan terputus
WARNING percobaan 3 dari 3 gagal: sambungan terputus
menyerah: sambungan terputus
```

Telusuri:

- Pemanggilan pertama: `mulai` dicatat, percobaan 1 dan 2 gagal (peringatan `WARNING`), percobaan 3 berhasil, lalu `selesai` dicatat. Hasilnya `{'stasiun': 'S1', 'mm': 12.5}`.
- Urutan decorator penting: `@catat` di luar, `@coba_ulang` di dalam. Maka log `mulai` dan `selesai` hanya muncul **sekali** per pemanggilan, walau terjadi tiga percobaan di dalamnya.
- Pemanggilan kedua: gangguan lebih panjang daripada tiga percobaan, sehingga galat dilempar ulang (`raise`). Perhatikan: tidak ada `selesai`, karena fungsi tidak selesai dengan baik. Program menangkapnya di luar dengan `try/except` dan mencetak “menyerah”.

TIPS

Coba-ulang hanya cocok untuk galat **sementara** (jaringan putus, server sibuk). Jangan mengulang galat yang pasti berulang (data salah): ia hanya membuang waktu. Itulah sebabnya `coba_ulang` hanya menangkap jenis galat yang disebutkan.

Latihan

1. Tulis closure `buat_pengali(faktor)` yang mengembalikan fungsi mengalikan angka dengan `faktor`. Buat `kali_tiga` dan `kali_sepuluh`, lalu coba keduanya.
2. Tulis decorator `hitung_panggilan` yang menyimpan jumlah panggilan di atribut fungsi (`pembungkus.jumlah`). Pasang pada satu fungsi, panggil empat kali, dan cetak `jumlah`.
3. Ubah `05_decorator_argumen.py`: tambahkan `@ulangi(2)` pada fungsi lain yang mencetak nama kolam, dan pastikan `__name__`-nya tetap benar.
4. **Tantangan:** tulis decorator `lebih_dari(batas)` yang melempar `ValueError` bila hasil fungsi di bawah `batas`. Pasang pada fungsi biomassa, dan uji dengan satu masukan yang lolos dan satu yang gagal.

Ringkasan Bab

- Fungsi adalah **objek**: bisa diberi nama lain, disimpan, dan dikirim ke fungsi lain.
- **Closure** = fungsi dalam yang mengingat variabel fungsi luar; ubah dengan `nonlocal`.
- **Decorator** = fungsi yang menerima fungsi dan mengembalikan pembungkusnya; `@nama` adalah singkatan.
- Selalu pakai **`@wraps`** agar nama dan docstring fungsi asli terjaga.
- Decorator **berargumen** butuh tiga tingkat dan **wajib berkurung** saat dipasang.
- Proyek I4: fungsi pengambil data kini punya log dan coba-ulang tanpa mengubah isinya.

Selanjutnya, di Bab 32: context manager, cara memastikan sumber daya (berkas, koneksi) selalu ditutup rapi.

Bab 32 — Context Manager: Membuka, Memakai, dan Pasti Menutup

Seri I4 — Bahasa Python Tingkat Lanjut

Di bab ini, Anda akan:

- Memahami mengapa `with` lebih aman daripada `open()` lalu `close()`
- Membuat context manager dengan kelas (`__enter__` dan `__exit__`)
- Membuat context manager lebih singkat dengan `@contextmanager`
- Memakai alat `contextlib` (`suppress`, `redirect_stdout`, `ExitStack`)
- Menulis laporan dengan aman: berkas lama tidak rusak bila penulisan gagal

Semua skrip bab ini ada di folder `kode/bab32/`. Di versi web, Anda juga bisa mengunduh semuanya sekaligus (zip).

Pernahkah Anda lupa mematikan pompa kolam setelah selesai mengisi air? Pompa terus menyala, listrik terbuang, atau kolam meluap. Program punya masalah serupa: berkas yang dibuka dan lupa ditutup, koneksi yang dibiarkan menggantung. Lebih parah lagi bila di tengah pekerjaan terjadi galat: baris `close()` di ujung tidak pernah tercapai.

Python punya solusi: pernyataan **`with`**. Ia berjanji bahwa apa pun yang terjadi di dalam blok, urusan "tutup dan bersihkan" **pasti** dikerjakan.

Gambar 0.1: with: pintu dibuka, pekerjaan dilakukan, lalu pintu dipastikan ditutup, baik berhasil maupun gagal.

`with` pada berkas

Anda sudah memakai `with open(...)` sejak Bab 13. Bandingkan dengan cara lama:

```
# Bab 32 - with menutup berkas otomatis
from pathlib import Path

folder = Path(__file__).parent / "_tmp"
folder.mkdir(exist_ok=True)
jalur = folder / "catatan.txt"

# Cara lama: harus ingat close()
```

```
f = open(jalur, "w", encoding="utf-8")
f.write("panen kebun A: 245.5 kg\n")
print("sebelum close, tertutup?", f.closed)
f.close()
print("sesudah close, tertutup?", f.closed)

# Cara with: tertutup sendiri, bahkan bila terjadi galat
with open(jalur, encoding="utf-8") as f:
    isi = f.read()
    print("di dalam with, tertutup?", f.closed)
print("di luar with, tertutup?", f.closed)
print(isi.strip())
```

```
sebelum close, tertutup? False
sesudah close, tertutup? True
di dalam with, tertutup? False
di luar with, tertutup? True
panen kebun A: 245.5 kg
```

- Cara lama: `f.closed` bernilai `False` sampai Anda sendiri memanggil `f.close()`. Kalau ada galat sebelum `close()`, berkasnya tetap terbuka.
- Cara `with`: di dalam blok, berkas terbuka (`False`). Begitu blok selesai, ia tertutup (`True`), tanpa Anda menulis `close()`.
- Perhatikan bahwa variabel `f` masih ada setelah blok, tetapi berkasnya sudah tertutup.

Di balik layar: `__enter__` dan `__exit__`

`with` bisa dipakai pada objek apa pun yang punya dua metode khusus: **`__enter__`** (dijalankan saat masuk blok; nilainya masuk ke variabel setelah `as`) dan **`__exit__`** (dijalankan saat keluar blok, **selalu**, termasuk bila terjadi galat). Objek seperti itu disebut **context manager**.

```
# Bab 32 - Context manager dengan kelas: __enter__ dan __exit__
class Pintu:
    def __init__(self, nama):
        self.nama = nama

    def __enter__(self):
        print(f"buka pintu {self.nama}")
        return self                     # nilai setelah kata "as"

    def __exit__(self, jenis_galat, galat, jejak):
        print(f"tutup pintu {self.nama}")
        return False                    # False: galat (bila ada) tetap diteruskan


with Pintu("kandang") as p:
    print("bekerja di dalam", p.nama)

print("--- sekarang dengan galat ---")
try:
    with Pintu("kolam"):
        print("bekerja...")
        raise ValueError("pompa mati")
except ValueError as e:
    print("galat ditangkap di luar:", e)
```

```
buka pintu kandang
bekerja di dalam kandang
tutup pintu kandang
```

```
--- sekarang dengan galat ---
buka pintu kolam
bekerja...
tutup pintu kolam
galat ditangkap di luar: pompa mati
```

- Pada percobaan pertama urutannya: buka, bekerja, tutup.
- Pada percobaan kedua, galat `ValueError("pompa mati")` terjadi di dalam blok. Pintu **tetap ditutup** ("tutup pintu kolam" tercetak) sebelum galatnya muncul di luar dan ditangkap `except`.
- `__exit__` menerima tiga informasi galat (jenis, isi, jejak; nilainya `None` bila tidak ada galat). Mengembalikan `False` berarti "galatnya silakan diteruskan".

Cara singkat: `@contextmanager`

Menulis kelas dengan dua metode itu panjang. `contextlib.contextmanager` mengubah **fungsi generator** (Bab 30) menjadi context manager: kode sebelum `yield` menjadi `__enter__`, kode sesudahnya menjadi `__exit__`.

```
# Bab 32 - @contextmanager: context manager dari fungsi generator
from contextlib import contextmanager


@contextmanager
def sesi_kerja(nama):
    print(f"[{nama}] mulai")
    try:
        yield nama.upper()              # isi blok with dijalankan di sini
    finally:
        print(f"[{nama}] selesai dan dibersihkan")


with sesi_kerja("pemupukan") as kode:
    print("bekerja dengan kode", kode)

try:
    with sesi_kerja("pemanenan"):
        print("bekerja...")
        1 / 0
except ZeroDivisionError:
    print("galat tertangkap, tetapi pembersihan tetap jalan")
```

```
[pemupukan] mulai
bekerja dengan kode PEMUPUKAN
[pemupukan] selesai dan dibersihkan
[pemanenan] mulai
bekerja...
[pemanenan] selesai dan dibersihkan
galat tertangkap, tetapi pembersihan tetap jalan
```

- `yield nama.upper()` melepaskan kendali ke blok `with` dan memberikan nilai `PEMUPUKAN` ke variabel `kode`.
- `try ... finally` di sekeliling `yield` menjamin pembersihan. Pada percobaan kedua terjadi `1 / 0`, tetapi "selesai dan dibersihkan" tetap tercetak, lalu galatnya tertangkap di luar.

> TIPS
>
> Selalu bungkus `yield` di `@contextmanager` dengan `try ... finally` bila ada pembersihan. Tanpa `finally`, bila blok `with` terjadi galat, baris pembersihan sesudah `yield` **tidak dijalankan**.

`__exit__` boleh menelan galat

Bila `__exit__` mengembalikan `True`, galatnya dianggap **sudah ditangani** dan program lanjut seolah tidak ada apa-apa. Ini kuat tetapi berbahaya.

```
# Bab 32 - __exit__ boleh menelan galat (dengan hati-hati)
from contextlib import suppress


class AbaikanKosong:
    def __enter__(self):
        return self

    def __exit__(self, jenis_galat, galat, jejak):
        if jenis_galat is ZeroDivisionError:
            print("dibagi nol diabaikan")
            return True                 # True: galat dianggap sudah ditangani
        return False


with AbaikanKosong():
    print(10 / 0)
print("program lanjut")

# Pustaka standar sudah menyediakan yang serupa
with suppress(FileNotFoundError):
    open("tidak_ada.txt", encoding="utf-8")
print("lanjut lagi")
```

```
dibagi nol diabaikan
program lanjut
lanjut lagi
```

Pada blok pertama, `10 / 0` melempar `ZeroDivisionError`, tetapi `__exit__` mengembalikan `True`, jadi "program lanjut" tetap tampil. Pustaka standar menyediakan versi yang lebih rapi: `contextlib.suppress(FileNotFoundError)`, yang menelan **hanya** jenis galat yang Anda sebut.

> ⚠ AWAS
>
> Menelan galat bisa menyembunyikan bug. Telan hanya jenis galat yang Anda **harapkan** dan memang tidak berbahaya (misalnya berkas yang memang boleh tidak ada). Jangan menelan semua galat.

Alat siap pakai di `contextlib`

```
# Bab 32 - Alat siap pakai di contextlib dan modul lain
import io
from contextlib import ExitStack, redirect_stdout
from pathlib import Path

# redirect_stdout: tangkap apa yang dicetak
tangkap = io.StringIO()
with redirect_stdout(tangkap):
    print("ini tidak tampil di layar")
print("tertangkap:", repr(tangkap.getvalue()))

# ExitStack: jumlah berkas baru diketahui saat program jalan
folder = Path(__file__).parent / "_tmp"
folder.mkdir(exist_ok=True)
```

```
nama = ["kolam_a.txt", "kolam_b.txt", "kolam_c.txt"]
with ExitStack() as tumpukan:
     berkas = [tumpukan.enter_context(open(folder / n, "w", encoding="utf-8")) for n
in nama]
    for i, f in enumerate(berkas):
        f.write(f"data kolam nomor {i + 1}\n")
    print("terbuka bersamaan:", len(berkas))
print("semua tertutup:", all(f.closed for f in berkas))
```

```
tertangkap: 'ini tidak tampil di layar\n'
terbuka bersamaan: 3
semua tertutup: True
```

- `redirect_stdout` mengalihkan apa yang dicetak `print` ke penampung `io.StringIO`. Layar tidak menampilkan apa-apa, tetapi isinya bisa dibaca lagi (`'ini tidak tampil di layar\n'`). Berguna untuk menguji fungsi yang mencetak.
- `ExitStack` dipakai bila jumlah sumber daya baru diketahui saat program berjalan. Di sini tiga berkas dibuka lewat `enter_context`, dan ketiganya ditutup bersama saat blok selesai.

Jebakan: `with` pada objek yang bukan context manager

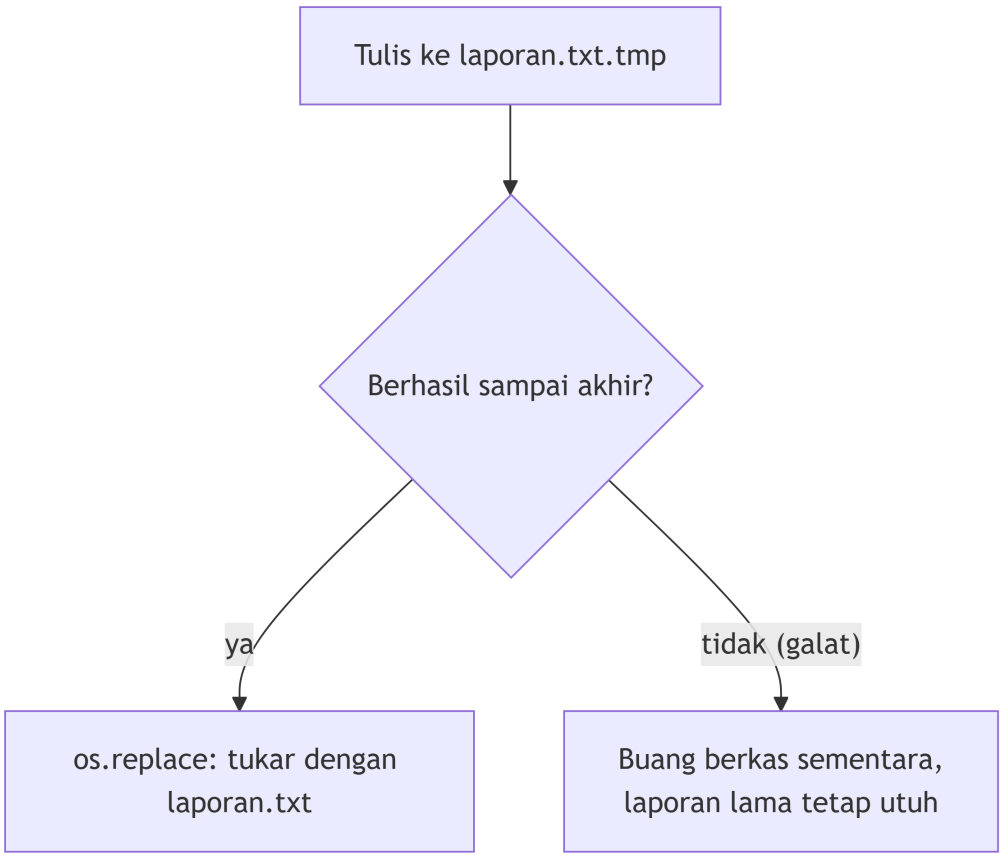
```
# Bab 32 - Salah: with pada objek yang bukan context manager
class Gudang:
    pass


with Gudang() as g:
    print("tidak akan sampai sini")
```

```
Traceback (most recent call last):
  File "06_bukan_context_manager.py", line 6, in <module>
    with Gudang() as g:
         ~~~~~~^^
TypeError: 'Gudang' object does not support the context manager protocol (missed
__exit__ method)
```

Pesan galatnya jelas: kelas `Gudang` tidak punya `__exit__` (dan `__enter__`), jadi tidak mendukung "protokol context manager". Solusinya: tambahkan kedua metode itu, atau pakai `@contextmanager`.

Proyek Bersambung I4: menulis laporan dengan aman

Skenario nyata: Anda menulis laporan harian ke berkas. Di tengah penulisan, listrik padam atau program error. Berkas yang tertulis setengah itu menimpa laporan lama yang utuh. Solusi umum: tulis ke **berkas sementara**, dan hanya bila seluruh penulisan berhasil, **tukar** dengan berkas asli.



Penulisan aman: berkas sementara baru menggantikan laporan asli bila semuanya berhasil.

```
# Bab 32 - Proyek bersambung I4: menulis laporan dengan aman
import os
from contextlib import contextmanager
from pathlib import Path

folder = Path(__file__).parent / "_tmp"
folder.mkdir(exist_ok=True)


@contextmanager
def tulis_aman(tujuan):
    """Menulis ke berkas sementara; hanya bila sukses, menggantikan berkas asli."""
    sementara = Path(str(tujuan) + ".tmp")
    f = open(sementara, "w", encoding="utf-8")
    try:
        yield f
    except BaseException:
        f.close()
        sementara.unlink()              # gagal: buang yang setengah jadi
        print("gagal, berkas sementara dibuang")
        raise
    else:
        f.close()
        os.replace(sementara, tujuan)    # sukses: tukar sekaligus
        print("sukses, laporan disimpan")
```

```
laporan = folder / "laporan.txt"

with tulis_aman(laporan) as f:
    f.write("Panen kebun: 1030.0 kg\n")

try:
    with tulis_aman(laporan) as f:
        f.write("Data setengah jadi...\n")
        raise RuntimeError("listrik padam")
except RuntimeError as e:
    print("galat:", e)

print("isi laporan tetap utuh:", laporan.read_text(encoding="utf-8").strip())
print("sisa berkas sementara:", Path(str(laporan) + ".tmp").exists())
```

```
sukses, laporan disimpan
gagal, berkas sementara dibuang
galat: listrik padam
isi laporan tetap utuh: Panen kebun: 1030.0 kg
sisa berkas sementara: False
```

Telusuri:

- Penulisan pertama sukses: "sukses, laporan disimpan", dan isi `Panen kebun: 1030.0 kg` menggantikan berkas.
- Penulisan kedua mensimulasikan listrik padam (`RuntimeError`) setelah menulis setengah isi. Pada blok `except BaseException`, berkas sementara ditutup dan **dibuang**, lalu galat dilempar ulang (`raise`) sehingga tertangkap di luar.
- Hasil akhir: isi laporan **tetap utuh** seperti sebelum penulisan kedua, dan tidak ada berkas `.tmp` yang tertinggal (`False`).
- `os.replace` menggantikan berkas tujuan. Menurut dokumentasi Python, bila berhasil operasi ini bersifat **atomik** (tidak terbelah di tengah), tetapi bisa gagal bila sumber dan tujuan berada di sistem berkas yang berbeda. Itu sebabnya berkas sementara ditaruh di folder yang sama dengan tujuan.

> BAGIAN TEKNIS (boleh dilewati)
>
> `BaseException` lebih luas daripada `Exception`: ia juga mencakup `KeyboardInterrupt` (Anda menekan Ctrl+C) dan `SystemExit`. Untuk pembersihan, kita ingin ikut tertangkap pada semua kasus itu, lalu **melempar ulang** agar galat tetap sampai ke pemanggil. Untuk menangani galat biasa, cukup `Exception`.

Latihan

1. Tulis context manager kelas `Pencatat` yang mencetak "mulai" di `__enter__` dan "selesai" di `__exit__`. Pakai dengan `with` dan tambahkan satu `print` di dalamnya.
2. Ubah `03_contextmanager.py`: tambahkan context manager `sesi_kerja("penyiraman")` ketiga yang mengalami `ValueError`, dan tangkap galat itu di luar. Apakah "dibersihkan" tetap tercetak?
3. Gunakan `contextlib.suppress(FileNotFoundError)` untuk menghapus berkas `_tmp/tidak_ada.txt` dengan `Path.unlink()` tanpa error bila berkasnya memang tidak ada.
4. **Tantangan:** tulis context manager `catat_langkah(nama)` memakai `@contextmanager` yang mencetak "[nama] mulai" dan "[nama] selesai"; bila terjadi galat, cetak "[nama] GAGAL" lalu lempar ulang galatnya. Uji dengan satu blok sukses dan satu blok gagal.

Ringkasan Bab

- **`with`** menjamin pembersihan (menutup berkas, melepas sumber daya) walau terjadi galat.
- Context manager = objek dengan **`__enter__`** dan **`__exit__`**; `__exit__` selalu jalan.

- `@contextmanager` mengubah fungsi generator menjadi context manager; bungkus `yield` dengan `try ... finally`.
- `__exit__` yang mengembalikan `True` **menelan** galat; pakai `suppress` hanya untuk jenis galat yang diharapkan.
- `contextlib` menyediakan `suppress`, `redirect_stdout`, dan `ExitStack` (jumlah sumber daya dinamis).
- Proyek I4: laporan ditulis lewat berkas sementara, sehingga laporan lama tetap utuh bila penulisan gagal.

Selanjutnya, di Bab 33: type hint, cara memberi tahu manusia dan alat pemeriksa tentang jenis data yang diharapkan.

Bab 33 — Type Hint: Memberi Label pada Data, Memeriksa Sebelum Berjalan

Seri I4 — Bahasa Python Tingkat Lanjut

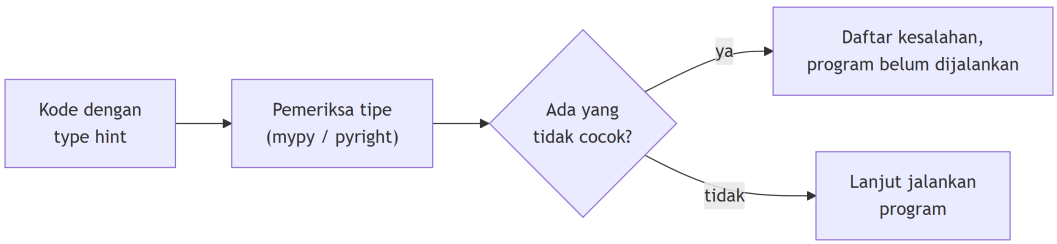
Di bab ini, Anda akan:

- Menulis **anotasi tipe** (*type hint*) pada variabel dan fungsi
- Memakai tipe untuk koleksi, pilihan (`X | None`), fungsi, dan generics
- Mengenal `TypedDict`, `Literal`, dan `Protocol`
- Menjalankan **mypy** sungguhan dan membaca keluarannya
- Memahami perubahan Python 3.14: anotasi dievaluasi belakangan

Semua skrip bab ini ada di folder `kode/bab33/`. Di versi web, Anda juga bisa mengunduh semuanya sekaligus (zip).

Bayangkan gudang pakan dengan karung tanpa label. Anda harus membuka tiap karung untuk tahu isinya, dan kadang salah mengambil. Sekarang bayangkan tiap karung berlabel "pelet ikan, 25 kg". Anda langsung tahu, dan petugas yang memeriksa gudang bisa menandai karung yang salah tempat sebelum pakan dipakai.

Type hint (petunjuk tipe) adalah label itu pada kode: catatan tentang jenis data yang diharapkan sebuah variabel atau fungsi. Python sendiri **tidak memaksakannya** saat program berjalan, tetapi manusia, editor, dan alat pemeriksa bisa memanfaatkannya untuk menemukan kesalahan **sebelum** program dijalankan.



Type hint dipakai alat pemeriksa (mypy, pyright) untuk menemukan kesalahan sebelum program berjalan.

Anotasi dasar

```
# Bab 33 - Anotasi tipe: catatan untuk manusia dan alat, bukan paksaan
def biomassa_kg(jumlah: int, bobot_g: float) -> float:
    return jumlah * bobot_g / 1000


umur_hari: int = 120
nama_kolam: str = "A"

print(biomassa_kg(800, 540.5))
print(biomassa_kg.__annotations__)

def gandakan(jumlah: int) -> int:
    return jumlah * 2
```

```
# Python TIDAK memeriksa anotasi saat program jalan
print(gandakan(21))
print(gandakan("ab"))          # bukan int, tetapi tetap jalan
```

```
432.4
{'jumlah': <class 'int'>, 'bobot_g': <class 'float'>, 'return': <class 'float'>}
42
abab
```

- `jumlah: int` artinya "jumlah diharapkan bilangan bulat". `-> float` di ujung artinya "fungsi mengembalikan angka desimal".
- Variabel juga bisa diberi label: `umur_hari: int = 120`.
- Anotasi disimpan di `__annotations__` sebagai catatan.
- **Python tidak memeriksanya saat berjalan.** `gandakan("ab")` bukan `int`, tetapi tetap jalan dan menghasilkan `abab` (teks dikali dua mengulang teks). Tidak ada galat. Pemeriksaan dilakukan alat terpisah, bukan Python.

> **i** INGAT
>
> Type hint adalah **catatan**, bukan **penjaga pintu**. Tanpa alat pemeriksa, salah tipe baru ketahuan saat program berjalan, atau malah tidak ketahuan seperti `abab` di atas. Untuk penjaga pintu saat berjalan, lihat Pydantic di Bab 34.

Tipe untuk koleksi, pilihan, dan fungsi

```
# Bab 33 - Tipe untuk koleksi, pilihan, dan fungsi
from collections.abc import Callable, Iterator


def rata_rata(angka: list[float]) -> float:
    return sum(angka) / len(angka)


def cari_kolam(data: dict[str, float], nama: str) -> float | None:
    """Mengembalikan bobot kolam, atau None bila tidak ada."""
    return data.get(nama)


def titik(x: float, y: float) -> tuple[float, float]:
    return (x, y)


def terapkan(fungsi: Callable[[float], float], nilai: list[float]) -> list[float]:
    return [fungsi(n) for n in nilai]


def hitung_mundur(n: int) -> Iterator[int]:
    while n > 0:
        yield n
        n -= 1


bobot = {"A": 540.5, "B": 498.0}
print(rata_rata([540.5, 498.0, 610.0]))
print(cari_kolam(bobot, "A"), cari_kolam(bobot, "Z"))
print(titik(1.5, 2.0))
```

```
print(terapkan(lambda g: g / 1000, [540.5, 498.0]))
print(list(hitung_mundur(3)))
```

```
549.5
540.5 None
(1.5, 2.0)
[0.5405, 0.498]
[3, 2, 1]
```

Penulisan	Artinya
`list[float]`	list yang isinya desimal
`dict[str, float]`	dict berkunci teks, bernilai desimal
`tuple[float, float]`	tuple dua desimal
`float | None`	desimal **atau** `None` (artinya "boleh kosong")
`Callable[[float], float]`	fungsi yang menerima satu desimal dan mengembalikan desimal
`Iterator[int]`	sesuatu yang bisa di-`next()` dan memberi bilangan bulat

`cari_kolam(bobot, "Z")` mengembalikan `None` karena kolam Z tidak ada; itu sebabnya tipenya `float | None`. Penulisan `float | None` memakai tanda `|` (Python 3.10 ke atas). `Callable` dan `Iterator` diambil dari `collections.abc`. Perhatikan: `hitung_mundur` adalah generator (Bab 30), dan tipe yang pas untuk generator sederhana adalah `Iterator[int]`.

Generics: tipe yang mengikuti isi

Fungsi `pertama(daftar)` bisa dipakai pada list desimal maupun list teks. Tipe hasilnya harus mengikuti: desimal bila diberi list desimal, teks bila diberi list teks. Itu tugas **generics**, dengan variabel tipe `T`.

```
# Bab 33 - Generics: tipe yang mengikuti isi
from collections.abc import Sequence


def pertama[T](daftar: Sequence[T]) -> T:
    """Mengembalikan elemen pertama; tipenya sama dengan isi daftar."""
    return daftar[0]


print(pertama([540.5, 498.0]))      # T = float
print(pertama(["A", "B"]))          # T = str


class Gudang[T]:
    """Tempat menyimpan barang bertipe sama."""

    def __init__(self) -> None:
        self.isi: list[T] = []

    def simpan(self, barang: T) -> None:
        self.isi.append(barang)

    def ambil(self) -> T:
        return self.isi.pop()


gudang_bobot = Gudang[float]()
```

```
gudang_bobot.simpan(540.5)
print(gudang_bobot.ambil())
```

```
540.5
A
540.5
```

Sintaks `def pertama[T](...)` dan `class Gudang[T]` adalah cara penulisan yang diperkenalkan di Python 3.12 (sebelumnya memakai `TypeVar` dan `Generic`). `Gudang[float]()` membuat gudang khusus desimal. Alat pemeriksa lalu akan memprotes bila Anda menyimpan teks ke dalamnya.

`TypedDict`, `Literal`, dan `Protocol`

Data lapangan sering berbentuk dict. `TypedDict` menyatakan kunci apa saja yang ada dan tipenya. `Literal` membatasi pilihan ke nilai tertentu. `Protocol` menyatakan "apa pun yang punya ini" tanpa mewajibkan pewarisan (disebut *duck typing*: "bila berjalan seperti bebek dan bersuara seperti bebek, ia bebek").

```
# Bab 33 - TypedDict, Literal, dan Protocol
from dataclasses import dataclass
from typing import Literal, Protocol, TypedDict

Sektor = Literal["hutan", "kebun", "sawah", "kolam", "kandang"]


class Rekaman(TypedDict):
    sektor: Sektor
    unit: str
    nilai: float


@dataclass
class Petak:
    nama: str
    luas_ha: float


class PunyaNama(Protocol):
    """Tipe 'bebek': apa pun yang punya atribut nama."""
    nama: str


def sapa(objek: PunyaNama) -> str:
    return "Halo, " + objek.nama


r: Rekaman = {"sektor": "kolam", "unit": "A", "nilai": 540.5}
print(r["sektor"], r["nilai"])
print(sapa(Petak("Blok A", 1.5)))
```

```
kolam 540.5
Halo, Blok A
```

`Rekaman` adalah dict dengan tiga kunci, dan `sektor` hanya boleh salah satu dari lima nilai. `sapa()` menerima apa saja yang punya atribut `nama`: di sini `Petak`, sebuah dataclass (Bab 16), tanpa mewarisi apa pun. Saat berjalan, semuanya biasa dict dan objek biasa; labelnya hanya untuk pemeriksa.

Menjalankan mypy sungguhan

mypy adalah alat pemeriksa tipe. Skrip di bawah menulis berkas kecil yang **sengaja salah**, menjalankan mypy atasnya, dan menampilkan keluaran aslinya.

```
# Bab 33 - Menjalankan mypy sungguhan pada berkas kecil
import subprocess
import sys
from pathlib import Path

folder = Path(__file__).parent / "_tmp"
folder.mkdir(exist_ok=True)
contoh = folder / "contoh_salah.py"
contoh.write_text('''def biomassa_kg(jumlah: int, bobot_g: float) -> float:
    return jumlah * bobot_g / 1000


total: int = biomassa_kg(800, 540.5)
print(biomassa_kg("800", 540.5))
print(biomassa_kg(800))
nama: str = None
''', encoding="utf-8")

hasil = subprocess.run(
    [sys.executable, "-m", "mypy", "--no-color-output", "--no-error-summary",
     "--cache-dir", str(folder / ".mypy_cache"), "contoh_salah.py"],
    cwd=folder, capture_output=True, text=True,
)
print(hasil.stdout.strip())
print("kode keluar mypy:", hasil.returncode)
```

```
contoh_salah.py:5: error: Incompatible types in assignment (expression has type
"float", variable has type "int")  [assignment]
contoh_salah.py:6: error: Argument 1 to "biomassa_kg" has incompatible type "str";
expected "int"  [arg-type]
contoh_salah.py:7: error: Missing positional argument "bobot_g" in call to
"biomassa_kg"  [call-arg]
contoh_salah.py:8: error: Incompatible types in assignment (expression has type "None",
variable has type "str")  [assignment]
kode keluar mypy: 1
```

Mypy menemukan empat kesalahan **tanpa menjalankan** berkas itu:

- Baris 5: `biomassa_kg` mengembalikan `float`, tetapi disimpan ke variabel yang dijanjikan `int`.
- Baris 6: argumen pertama berupa teks `"800"`, padahal diharapkan `int`.
- Baris 7: argumen `bobot_g` kurang.
- Baris 8: `None` diberikan ke variabel bertipe `str`.

Kode dalam kurung siku di ujung (`[assignment]`, `[arg-type]`, `[call-arg]`) adalah nama kategori galat. Kode keluar mypy `1` berarti "ada masalah", yang berguna bila mypy dijalankan otomatis (misalnya di pemeriksaan sebelum commit, Bab 19 dan 20).

> TIPS
>
> Mulailah dari yang kecil. Beri type hint pada fungsi yang paling sering dipakai, jalankan mypy, perbaiki, lalu perluas. Opsi `--strict` mypy lebih ketat (menuntut hampir semua fungsi diberi tipe); cocok bila proyek sudah rapi.

Pyright, alat pemeriksa lain

Selain mypy, ada **pyright**, pemeriksa tipe buatan Microsoft. Di halaman proyeknya, pyright digambarkan sebagai pemeriksa tipe statis untuk Python yang tersedia sebagai alat baris perintah dan ekstensi Visual Studio Code. Pyright **tidak dijalankan di buku ini**; cara memasangnya dan perbedaan perilakunya dengan mypy belum saya verifikasi di komputer penulis, jadi silakan lihat dokumentasinya sendiri bila ingin mencoba. Konsep yang Anda pelajari di sini (anotasi, generics, `TypedDict`) berlaku sama untuk keduanya.

Python 3.14: anotasi dievaluasi belakangan

Dulu, anotasi seperti `x: Kelas` langsung dievaluasi saat `def` dijalankan. Bila `Kelas` belum didefinisikan, terjadi galat, sehingga orang menulis `"Kelas"` (dalam tanda kutip) atau `from __future__ import annotations`. Sejak **Python 3.14**, anotasi disimpan dan dievaluasi **hanya bila diperlukan** (sumber: dokumentasi "What's New in Python 3.14", bagian anotasi tertunda). Akibatnya acuan ke nama yang belum ada tidak lagi memerlukan tanda kutip. Modul baru `annotationlib` membaca anotasi dalam tiga format.

```
# Bab 33 - Python 3.14: anotasi dievaluasi belakangan
from annotationlib import Format, get_annotations


def hitung(x: BelumDibuat) -> None:        # nama ini BELUM ada
    pass


print("fungsi berhasil dibuat tanpa galat")
print(get_annotations(hitung, format=Format.STRING))
acuan = get_annotations(hitung, format=Format.FORWARDREF)["x"]
print(type(acuan).__name__, acuan.__forward_arg__)
try:
    get_annotations(hitung, format=Format.VALUE)
except NameError as e:
    print("NameError:", e)


class BelumDibuat:
    pass


print(get_annotations(hitung, format=Format.VALUE))   # sekarang namanya ada
```

```
fungsi berhasil dibuat tanpa galat
{'x': 'BelumDibuat', 'return': 'None'}
ForwardRef BelumDibuat
NameError: name 'BelumDibuat' is not defined
{'x': <class '__main__.BelumDibuat'>, 'return': None}
```

- `hitung(x: BelumDibuat)` berhasil dibuat padahal `BelumDibuat` belum ada. Pada Python lama ini langsung galat `NameError`.
- Format `STRING` menampilkan anotasi sebagai teks, `FORWARDREF` sebagai acuan (`ForwardRef`) bagi nama yang belum ada, dan `VALUE` mengevaluasi nilainya sungguhan.
- Format `VALUE` sebelum kelas dibuat menghasilkan `NameError`. Setelah `BelumDibuat` didefinisikan, format `VALUE` berhasil dan mengembalikan kelasnya.

BAGIAN TEKNIS (boleh dilewati)

Skrip ini hanya berjalan di Python 3.14 ke atas, karena `annotationlib` baru ada di sana dan perilaku "fungsi dibuat walau nama belum ada" juga khas 3.14. Pembaca yang masih memakai Python lama sebaiknya tetap menulis acuan maju dalam tanda kutip. Kode `from __future__ import annotations` tetap dikenali dan mempertahankan perilaku lama.

Proyek Bersambung I4: memberi tipe pada kode proyek

Kita beri type hint pada fungsi pembaca rekaman dari Bab 30, lalu periksa dengan mypy mode ketat. Skrip menulis modul kecil `lapangan.py` ke `_tmp/`, menjalankan mypy, **menyisipkan satu kesalahan sengaja**, menjalankan mypy lagi, lalu menjalankan modulnya.

```
# Bab 33 - Proyek bersambung I4: memberi tipe pada kode proyek, lalu diperiksa mypy
import subprocess
import sys
from pathlib import Path

folder = Path(__file__).parent / "_tmp"
folder.mkdir(exist_ok=True)
modul = folder / "lapangan.py"
modul.write_text('''from collections.abc import Iterable, Iterator
from typing import Literal, TypedDict, cast

Sektor = Literal["hutan", "kebun", "sawah", "kolam", "kandang"]


class Rekaman(TypedDict):
    sektor: Sektor
    unit: str
    nilai: float


def baca(baris: Iterable[str]) -> Iterator[Rekaman]:
    for b in baris:
        sektor, unit, nilai = b.split(",")
        yield {"sektor": cast(Sektor, sektor), "unit": unit, "nilai": float(nilai)}


def rata_rata(rekaman: Iterable[Rekaman]) -> float | None:
    nilai = [r["nilai"] for r in rekaman]
    return sum(nilai) / len(nilai) if nilai else None


print(rata_rata(baca(["kolam,A,540.5", "kolam,B,498.0"])))
print(rata_rata(baca([])))
''', encoding="utf-8")


def mypy(*opsi: str) -> None:
    hasil = subprocess.run(
       [sys.executable, "-m", "mypy", "--no-color-output", "--cache-dir", str(folder /
".mypy_cache"),
         *opsi, "lapangan.py"],
        cwd=folder, capture_output=True, text=True,
    )
    print(hasil.stdout.strip())
```

```
print("== mypy ==")
mypy("--strict")
print("== sengaja disisipi satu kesalahan ==")
modul.write_text(modul.read_text(encoding="utf-8") + "print(rata_rata([1.5, 2.5]))" +
chr(10), encoding="utf-8")
mypy("--strict")
print("== menjalankan modul ==")
jalan     =     subprocess.run([sys.executable,     "lapangan.py"],     cwd=folder,
capture_output=True, text=True)
print(jalan.stdout.strip())
print("galat saat jalan:", jalan.stderr.strip().splitlines()[-1])
```

```
== mypy ==
Success: no issues found in 1 source file
== sengaja disisipi satu kesalahan ==
lapangan.py:26: error: List item 0 has incompatible type "float"; expected "Rekaman"
[list-item]
lapangan.py:26: error: List item 1 has incompatible type "float"; expected "Rekaman"
[list-item]
Found 2 errors in 1 file (checked 1 source file)
== menjalankan modul ==
519.25
None
galat saat jalan: TypeError: 'float' object is not subscriptable
```

Telusuri:

- Mypy `--strict` menyatakan modul bersih: `Success: no issues found in 1 source file`. `cast(Sektor, sektor)` memberi tahu mypy bahwa kita yakin teks itu salah satu nilai `Sektor` (mypy tidak bisa mengetahuinya dari teks bebas). Itu janji kita, bukan pemeriksaan; pemeriksaan sungguhan datang di Bab 34.
- Setelah baris `print(rata_rata([1.5, 2.5]))` disisipkan (seharusnya memberi `Rekaman`, bukan angka), mypy melaporkan dua kesalahan, satu untuk tiap elemen list. Program belum dijalankan saat itu.
- Bila modul tetap dijalankan, yang tercetak adalah 519.25 dan `None` dari dua baris pertama, lalu program jatuh di baris sisipan dengan `TypeError: 'float' object is not subscriptable`. Mypy menangkap kesalahan yang sama **tanpa** perlu menunggu program jatuh.

Latihan

1. Beri type hint pada fungsi `rata_rata_bobot(bobot)` yang menerima `list[float]` dan mengembalikan `float`.
2. Tulis fungsi `cari_pohon(daftar: list[str], nama: str) -> int | None` yang mengembalikan posisi `nama` dalam daftar, atau `None` bila tidak ada.
3. Ubah `05_jalankan_mypy.py`: perbaiki empat kesalahan pada berkas contoh, jalankan lagi, dan lihat pesan `Success`.
4. **Tantangan:** buat `TypedDict` bernama `Ternak` (kunci `kandang: str`, `jenis: Literal["sapi", "kambing", "ayam"]`, `bobot_kg: float`), lalu fungsi `bobot_total(data: list[Ternak]) -> float`. Periksa dengan mypy, termasuk satu data yang `jenis`-nya salah ketik.

Ringkasan Bab

- **Type hint** adalah label tipe pada variabel dan fungsi; Python **tidak memeriksanya** saat berjalan.
- Alat seperti **mypy** (dan pyright) memeriksa tipe **sebelum** program berjalan dan melaporkan ketidakcocokan.
- Tipe umum: `list[float]`, `dict[str, float]`, `float | None`, `Callable`, `Iterator`; **generics** (`[T]`) membuat tipe mengikuti isi.
- `TypedDict`, `Literal`, dan `Protocol` menyatakan bentuk dict, pilihan terbatas, dan “apa pun yang punya ini”.

- Python 3.14 menunda evaluasi anotasi; `annotationlib` membaca dalam format `VALUE`, `FORWARDREF`, dan `STRING`.
- Proyek I4: kode pembaca rekaman diberi tipe dan lulus `mypy --strict`; kesalahan sisipan terlihat sebelum dijalankan.

Selanjutnya, di Bab 34: validasi data dan konfigurasi dengan Pydantic, penjaga pintu yang bekerja saat program berjalan.

Bab 34 — Validasi Data dan Konfigurasi: Penjaga Pintu dengan Pydantic

Seri I4 — Bahasa Python Tingkat Lanjut

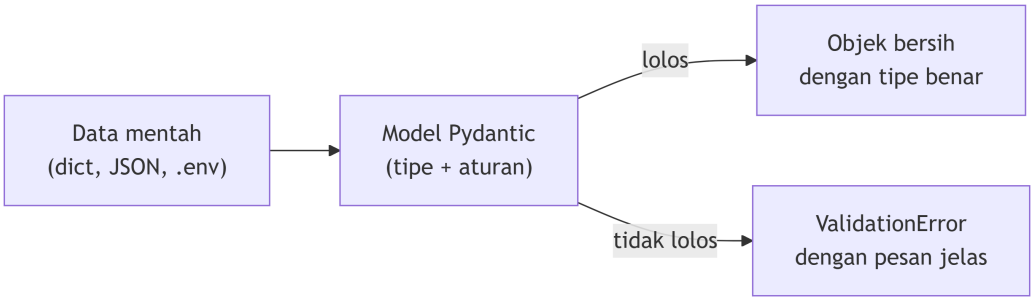
Di bab ini, Anda akan:

- Memahami mengapa data dari luar harus **divalidasi**, dan batas dataclass
- Membuat **model Pydantic**: validasi, konversi tipe, dan pesan galat
- Menambah aturan dengan `Field`, `field_validator`, dan `model_validator`
- Membaca pengaturan aplikasi dari variabel lingkungan dan `.env` (`pydantic-settings`)
- Memvalidasi data lapangan lima sektor sekaligus: proyek Seri I4

Semua skrip bab ini ada di folder `kode/bab34/`. Di versi web, Anda juga bisa mengunduh semuanya sekaligus (zip).

Di Bab 33, type hint berfungsi sebagai label yang dibaca alat pemeriksa. Tetapi saat program berjalan, label itu tidak menjaga apa-apa. Padahal data lapangan sering datang dari luar: formulir, berkas CSV, JSON dari API, ketikan petugas. Isinya bisa salah ketik, kosong, atau di luar akal sehat (pH air 17, diameter pohon negatif).

Perlu **penjaga pintu**: pemeriksa yang bekerja **saat program berjalan**, menolak data buruk dengan pesan jelas, dan merapikan data yang masih bisa diterima. Di Python, penjaga pintu yang paling populer adalah **Pydantic**.



Pydantic sebagai penjaga pintu: data mentah diperiksa, lalu menjadi objek bersih atau ditolak dengan pesan galat.

dataclass rapi, tetapi tidak menjaga

Di Bab 16 Anda mengenal `dataclass`: cara ringkas membuat kelas penyimpan data. Lihat apa yang terjadi bila isinya tidak masuk akal:

```
# Bab 34 - dataclass rapi, tetapi tidak memeriksa isi
from dataclasses import dataclass


@dataclass
class Pohon:
    jenis: str
    diameter_cm: float
    sehat: bool
```

```
bagus = Pohon("Jati", 32.5, True)
print(bagus)

kacau = Pohon("Jati", "tiga puluh", "mungkin")   # tidak ada protes
print(kacau)
print(type(kacau.diameter_cm).__name__)
```

```
Pohon(jenis='Jati', diameter_cm=32.5, sehat=True)
Pohon(jenis='Jati', diameter_cm='tiga puluh', sehat='mungkin')
str
```

Objek `kacau` dibuat tanpa protes: diameter berisi teks `'tiga puluh'`, `sehat` berisi `'mungkin'`. Tipenya `str`, padahal kita menulis `float`. `dataclass` hanya merapikan penulisan kelas, **tidak memeriksa isi**. Kesalahan baru terasa nanti, mungkin jauh dari sumbernya, ketika angka itu dihitung.

Model Pydantic

Pydantic adalah pustaka validasi data yang memakai type hint sebagai aturan. Versi yang terpasang di komputer penulis berseri utama 2 (skrip mencetak "versi utama" saja, karena nomor lengkapnya berubah dari waktu ke waktu). Buku ini memakai gaya Pydantic seri 2.

```
# Bab 34 - Model Pydantic: memeriksa dan mengubah tipe
import pydantic
from pydantic import BaseModel

print("Pydantic versi utama:", pydantic.VERSION.split(".")[0])


class Pohon(BaseModel):
    jenis: str
    diameter_cm: float
    sehat: bool = True          # nilai bawaan


p = Pohon(jenis="Jati", diameter_cm="32.5", sehat="true")  # teks diubah menjadi angka
dan bool
print(p)
print(type(p.diameter_cm).__name__, type(p.sehat).__name__)
print(p.model_dump())

p2 = Pohon(jenis="Meranti", diameter_cm=45)   # int menjadi float
print(p2.diameter_cm, p2.sehat)
```

```
Pydantic versi utama: 2
jenis='Jati' diameter_cm=32.5 sehat=True
float bool
{'jenis': 'Jati', 'diameter_cm': 32.5, 'sehat': True}
45.0 True
```

- Model dibuat dengan kelas turunan `BaseModel`, dan setiap atribut diberi tipe. `sehat: bool = True` punya nilai bawaan.
- Pydantic **mengonversi** bila masuk akal: teks `"32.5"` menjadi `float` 32.5, teks `"true"` menjadi `True`, dan bilangan bulat `45` menjadi `45.0`.
- Hasilnya objek dengan tipe yang benar: `float` dan `bool` (bukan `str`).
- `model_dump()` mengubah objek kembali menjadi dict biasa.

Di sini ada pelajaran kecil. Saat saya mencoba mengisi `sehat="ya"` (kata Indonesia untuk "ya"), Pydantic **menolaknya** (`bool_parsing`): ia tidak mengenal kata itu. Yang ia kenali adalah bentuk baku seperti `"true"`. Konversi Pydantic itu terbatas dan terdokumentasi, bukan menebak.

Pesan galat yang jelas

Bila data tidak bisa diterima, Pydantic melempar `ValidationError` yang menjelaskan **semua** masalah sekaligus, bukan hanya yang pertama.

```
# Bab 34 - Pesan galat yang jelas
from pydantic import BaseModel, ValidationError


class Pohon(BaseModel):
    jenis: str
    diameter_cm: float
    sehat: bool


try:
    Pohon(jenis="Jati", diameter_cm="tiga puluh", sehat="mungkin")
except ValidationError as e:
    print("jumlah masalah:", e.error_count())
    for masalah in e.errors(include_url=False, include_input=False):
        print(masalah["loc"], "|", masalah["type"], "|", masalah["msg"])

try:
    Pohon(jenis="Jati")
except ValidationError as e:
    for masalah in e.errors(include_url=False, include_input=False):
        print(masalah["loc"], "|", masalah["type"], "|", masalah["msg"])
```

```
jumlah masalah: 2
('diameter_cm',) | float_parsing | Input should be a valid number, unable to parse
string as a number
('sehat',) | bool_parsing | Input should be a valid boolean, unable to interpret input
('diameter_cm',) | missing | Field required
('sehat',) | missing | Field required
```

Pada percobaan pertama, ada **dua** masalah: `diameter_cm` tidak bisa dibaca sebagai angka (`float_parsing`) dan `sehat` tidak bisa dibaca sebagai ya/tidak (`bool_parsing`). Pada percobaan kedua, dua kolom wajib tidak diisi (`missing`: "Field required"). Setiap masalah punya:

- `loc`: letak masalah (nama kolom),
- `type`: kategori galat yang stabil dan bisa diproses program,
- `msg`: kalimat penjelas untuk manusia.

Opsi `include_url=False` dan `include_input=False` menghilangkan tautan dan salinan masukan dari tampilan, supaya keluaran ringkas dan stabil.

Aturan tambahan: `Field` dan validator

Tipe saja belum cukup. "Angka" belum berarti "masuk akal". Aturan tambahan bisa dipasang dengan `Field` (batas sederhana) dan validator (aturan buatan sendiri).

```
# Bab 34 - Aturan lewat Field dan validator
from typing import Literal

from pydantic import BaseModel, Field, ValidationError, field_validator,
model_validator
```

```python
class Ikan(BaseModel):
    kolam: str = Field(min_length=1, max_length=3)
    jenis: Literal["lele", "nila"]
    jumlah: int = Field(gt=0)
    bobot_rata_g: float = Field(ge=1, le=5000)
    ph_air: float = Field(ge=0, le=14)

    @field_validator("kolam")
    @classmethod
    def huruf_besar(cls, nilai: str) -> str:
        return nilai.upper()          # sekaligus merapikan

    @model_validator(mode="after")
    def periksa_bobot_lele(self):
        if self.jenis == "lele" and self.bobot_rata_g > 2000:
            raise ValueError("bobot rata-rata lele melebihi 2000 g tidak masuk akal")
        return self


print(Ikan(kolam="a", jenis="lele", jumlah=800, bobot_rata_g=540.5, ph_air=7.2))

for data in [
    dict(kolam="B", jenis="gurame", jumlah=100, bobot_rata_g=300, ph_air=7),
    dict(kolam="B", jenis="nila", jumlah=0, bobot_rata_g=300, ph_air=15),
    dict(kolam="C", jenis="lele", jumlah=10, bobot_rata_g=3000, ph_air=7),
]:
    try:
        Ikan(**data)
    except ValidationError as e:
        for m in e.errors(include_url=False, include_input=False):
            print(m["loc"], "|", m["msg"])
        print("--")
```

```
kolam='A' jenis='lele' jumlah=800 bobot_rata_g=540.5 ph_air=7.2
('jenis',) | Input should be 'lele' or 'nila'
--
('jumlah',) | Input should be greater than 0
('ph_air',) | Input should be less than or equal to 14
--
() | Value error, bobot rata-rata lele melebihi 2000 g tidak masuk akal
--
```

- `Field(gt=0)` berarti “lebih besar dari 0”, `ge`/`le` berarti “lebih besar sama dengan”/“lebih kecil sama dengan”, dan `min_length`/`max_length` membatasi panjang teks.
- `Literal["lele", "nila"]` membatasi pilihan jenis ikan. `"gurame"` ditolak: “Input should be ‘lele’ or ‘nila’”.
- `@field_validator("kolam")` memeriksa (dan boleh merapikan) satu kolom: nama kolam `"a"` menjadi `"A"`.
- `@model_validator(mode="after")` memeriksa **gabungan** kolom setelah semuanya valid: lele dengan bobot rata-rata di atas 2000 g ditolak, dan lokasi galatnya `()` (kosong) karena masalahnya milik seluruh model, bukan satu kolom.

> ⚠ AWAS
>
> Batas seperti `bobot_rata_g <= 5000` atau “lele tidak lebih dari 2000 g” di skrip ini hanyalah **angka contoh untuk latihan**, bukan anjuran budidaya. Tentukan batas sebenarnya bersama penyuluh atau pembudidaya berpengalaman.

Pengaturan aplikasi: variabel lingkungan dan `.env`

Aplikasi punya pengaturan yang berbeda di tiap komputer: lokasi berkas, batas unduhan, mode uji, kunci layanan. Menaruhnya di dalam kode merepotkan dan, untuk kunci rahasia, berbahaya. Praktik umum: simpan di **variabel lingkungan** (*environment variable*, pengaturan di luar program) atau di berkas `.env` (daftar `NAMA=nilai`), dan baca saat program mulai.

Paket `pydantic-settings` membaca pengaturan ke dalam kelas yang divalidasi seperti model Pydantic. Skrip ini memakai nilai **tiruan** (tidak ada rahasia sungguhan) dan menulis berkas `.env` contoh ke `_tmp/`.

```
# Bab 34 - Pengaturan aplikasi dari variabel lingkungan dan berkas .env (nilai tiruan)
import os
from pathlib import Path

from pydantic import SecretStr
from pydantic_settings import BaseSettings, SettingsConfigDict

folder = Path(__file__).parent / "_tmp"
folder.mkdir(exist_ok=True)
berkas_env = folder / "contoh.env"
berkas_env.write_text("LAPANGAN_NAMA_UNIT=Kolam     Contoh\nLAPANGAN_TOKEN_API=token-
tiruan-bukan-rahasia\n"
                      "LAPANGAN_BATAS_UNDUH=5\n", encoding="utf-8")


class Pengaturan(BaseSettings):
      model_config = SettingsConfigDict(env_prefix="LAPANGAN_", env_file=berkas_env,
env_file_encoding="utf-8")

    nama_unit: str = "Tanpa nama"
    batas_unduh: int = 10
    token_api: SecretStr = SecretStr("")
    mode_uji: bool = False


s = Pengaturan()
print("dari .env       :", s.nama_unit, s.batas_unduh, s.mode_uji)
print("token (disamarkan):", s.token_api)
print("token (sengaja dibuka):", s.token_api.get_secret_value() == "token-tiruan-
bukan-rahasia")

os.environ["LAPANGAN_BATAS_UNDUH"] = "25"        # variabel lingkungan mengalahkan .env
os.environ["LAPANGAN_MODE_UJI"] = "true"
s2 = Pengaturan()
print("env mengalahkan .env:", s2.batas_unduh, s2.mode_uji)

print("argumen langsung paling kuat:", Pengaturan(batas_unduh=1).batas_unduh)
```

```
dari .env       : Kolam Contoh 5 False
token (disamarkan): **********
token (sengaja dibuka): True
env mengalahkan .env: 25 True
argumen langsung paling kuat: 1
```

- `env_prefix="LAPANGAN_"` berarti pengaturan `batas_unduh` dibaca dari `LAPANGAN_BATAS_UNDUH`. Awalan ini mencegah bentrok dengan variabel milik program lain.
- Nilai dari `.env` otomatis dikonversi: `"5"` menjadi `int` 5.
- `SecretStr` menyamarkan nilai saat dicetak (`**********`), supaya kunci tidak ikut bocor ke log. Nilai aslinya hanya terbuka lewat `get_secret_value()`, yang kita pakai di sini hanya untuk membandingkan.

- Urutan kekuatan, sesuai dokumentasi pydantic-settings: **argumen langsung** > **variabel lingkungan** > **berkas `.env`** > **nilai bawaan**. Skrip membuktikannya: `.env` memberi 5, variabel lingkungan menimpanya menjadi 25, dan argumen `Pengaturan(batas_unduh=1)` mengalahkan semuanya.

> TIPS
>
> Berkas `.env` berisi pengaturan pribadi dan sering rahasia. **Jangan** simpan di Git (masukkan ke `.gitignore`, Bab 20), dan sediakan berkas contoh (`.env.example`) tanpa nilai rahasia. Di buku ini semua nilai tiruan.

JSON, daftar model, dan skema

```
# Bab 34 - JSON, daftar model, dan skema JSON
import json

from pydantic import BaseModel, TypeAdapter


class Rekaman(BaseModel):
    sektor: str
    unit: str
    nilai: float


mentah = '{"sektor": "kolam", "unit": "A", "nilai": "540.5"}'
r = Rekaman.model_validate_json(mentah)       # JSON teks -> objek tervalidasi
print(r)
print(r.model_dump_json())

daftar = TypeAdapter(list[Rekaman]).validate_python([
    {"sektor": "hutan", "unit": "H1", "nilai": 32.5},
    {"sektor": "kebun", "unit": "K2", "nilai": "245.5"},
])
print(len(daftar), daftar[1].nilai)

skema = Rekaman.model_json_schema()          # dasar dokumentasi API dan alat agent AI
print(json.dumps(skema, indent=2))
```

```
sektor='kolam' unit='A' nilai=540.5
{"sektor":"kolam","unit":"A","nilai":540.5}
2 245.5
{
  "properties": {
    "sektor": {
      "title": "Sektor",
      "type": "string"
    },
    "unit": {
      "title": "Unit",
      "type": "string"
    },
    "nilai": {
      "title": "Nilai",
      "type": "number"
    }
  },
  "required": [
    "sektor",
```

```
    "unit",
    "nilai"
  ],
  "title": "Rekaman",
  "type": "object"
}
```

- `model_validate_json` memvalidasi teks JSON langsung (`"540.5"` menjadi angka). `model_dump_json()` mengubah objek kembali menjadi JSON.
- `TypeAdapter(list[Rekaman])` memvalidasi **daftar** model sekaligus.
- `model_json_schema()` menghasilkan **skema JSON**: deskripsi baku bentuk datanya (kolom, tipe, mana yang wajib). Skema inilah dasar dokumentasi otomatis sebuah API.

Dataclass atau Pydantic?

	`dataclass`	Pydantic `BaseModel`
Memeriksa isi saat berjalan	tidak	ya
Mengonversi tipe (`"32.5"` ke 32,5)	tidak	ya
Pesan galat terperinci	tidak	ya
Bagian pustaka standar	ya	tidak (pustaka tambahan)
Cocok untuk	data **di dalam** program yang sudah tepercaya	data yang **masuk dari luar**

Aturan praktis: data yang melewati "perbatasan" (formulir, berkas, API, pengaturan) divalidasi dengan Pydantic; data internal yang sudah bersih boleh berupa `dataclass`.

Kaitan dengan FastAPI dan agent AI

Pemeriksaan di perbatasan ini menjadi dasar dua topik berikutnya dalam buku. **FastAPI** (Bab 36), kerangka kerja web, memakai Pydantic: paket FastAPI yang terpasang di komputer penulis mencantumkan `pydantic` sebagai ketergantungan (hasil `pip show fastapi`), dan di Bab 36 Anda akan melihat sendiri model Pydantic dipakai untuk memeriksa masukan API (di bab ini hal itu belum dijalankan). Pada pembahasan agent AI di Tingkat Advanced, skema serupa dipakai untuk menjelaskan bentuk data kepada model bahasa; bagian itu belum diuji di bab ini.

Proyek Bersambung I4: validasi data lapangan lima sektor

Data lapangan lima sektor punya bentuk berbeda: pohon punya diameter, kebun punya hasil, sawah punya gabah, kolam punya bobot dan pH, kandang punya bobot ternak. Kita buat lima model, lalu satu "pintu" yang memilih model dengan membaca kolom `sektor` (disebut *discriminated union*, gabungan bertanda).

```
# Bab 34 - Proyek bersambung I4: validasi data lapangan lima sektor
from typing import Annotated, Literal

from pydantic import BaseModel, Field, TypeAdapter, ValidationError


class Hutan(BaseModel):
    sektor: Literal["hutan"]
    unit: str
    diameter_cm: float = Field(gt=0, le=300)
```

```
class Kebun(BaseModel):
    sektor: Literal["kebun"]
    unit: str
    hasil_kg: float = Field(ge=0)


class Sawah(BaseModel):
    sektor: Literal["sawah"]
    unit: str
    gabah_kg: float = Field(ge=0)


class Kolam(BaseModel):
    sektor: Literal["kolam"]
    unit: str
    bobot_rata_g: float = Field(gt=0)
    ph_air: float = Field(ge=0, le=14)


class Kandang(BaseModel):
    sektor: Literal["kandang"]
    unit: str
    bobot_kg: float = Field(gt=0)


Lapangan = Annotated[Hutan | Kebun | Sawah | Kolam | Kandang,
Field(discriminator="sektor")]
pemeriksa = TypeAdapter(Lapangan)

mentah = [
    {"sektor": "hutan", "unit": "H1", "diameter_cm": "32.5"},
    {"sektor": "kebun", "unit": "K1", "hasil_kg": 245.5},
    {"sektor": "sawah", "unit": "S1", "gabah_kg": -10},
    {"sektor": "kolam", "unit": "A", "bobot_rata_g": 540.5, "ph_air": 7.2},
    {"sektor": "kolam", "unit": "B", "bobot_rata_g": 498, "ph_air": 17},
    {"sektor": "kandang", "unit": "D1", "bobot_kg": 412.5},
    {"sektor": "empang", "unit": "E1", "bobot_rata_g": 100},
    {"sektor": "kandang", "unit": "D2"},
]

sah, ditolak = [], []
for nomor, baris in enumerate(mentah, start=1):
    try:
        sah.append(pemeriksa.validate_python(baris))
    except ValidationError as e:
        pesan = e.errors(include_url=False, include_input=False)[0]
          ditolak.append((nomor, ".".join(str(x) for x in pesan["loc"]) or "sektor",
pesan["msg"]))

print("sah:", len(sah), "| ditolak:", len(ditolak))
print("jenis objek sah:", sorted({type(o).__name__ for o in sah}))
for nomor, tempat, alasan in ditolak:
    print(f"baris {nomor}: {tempat} -> {alasan}")
```

```
sah: 4 | ditolak: 4
jenis objek sah: ['Hutan', 'Kandang', 'Kebun', 'Kolam']
baris 3: sawah.gabah_kg -> Input should be greater than or equal to 0
baris 5: kolam.ph_air -> Input should be less than or equal to 14
baris 7: sektor -> Input tag 'empang' found using 'sektor' does not match any of the
```

```
expected tags: 'hutan', 'kebun', 'sawah', 'kolam', 'kandang'
baris 8: kandang.bobot_kg -> Field required
```

Delapan baris mentah masuk, **empat lolos** dan **empat ditolak**:

- Baris 1: `diameter_cm` berupa teks `"32.5"`, dikonversi dan diterima sebagai `Hutan`.
- Baris 3: `gabah_kg` bernilai −10, ditolak karena harus `>= 0`.
- Baris 5: pH 17, ditolak karena di atas 14.
- Baris 7: sektor `"empang"` bukan salah satu dari lima, ditolak, dan pesannya menyebut pilihan yang sah.
- Baris 8: kolom `bobot_kg` tidak ada (“Field required”).
- Jenis objek yang lolos: `Hutan`, `Kandang`, `Kebun`, `Kolam`, masing-masing berbentuk berbeda tetapi semuanya sudah bertipe benar.

Data yang ditolak tidak membuat program berhenti: ia dicatat beserta alasannya, dan sisanya diproses. Itulah pola kerja yang aman untuk data lapangan yang kotor.

Latihan

1. Buat model `Ternak` dengan kolom `kandang: str`, `jenis: Literal["sapi", "kambing", "ayam"]`, `umur_bulan: int` (harus `>= 0`), dan `bobot_kg: float` (harus `> 0`). Coba dengan satu data benar dan satu data salah.
2. Ubah `03_pesan_galat.py`: berikan `diameter_cm=-5` pada model yang punya `Field(gt=0)`. Baca pesan galatnya.
3. Di `05_pengaturan_env.py`, tambahkan pengaturan `folder_keluaran: str` bernilai bawaan `"hasil"`, lalu timpa lewat variabel lingkungan dan lewat `.env`. Amati mana yang menang.
4. **Tantangan:** tambahkan model keenam `Peralatan` ke `07_proyek_validasi_lima_sektor.py` (dengan `sektor` bernilai `"alat"`), masukkan ke gabungan, dan tambahkan dua baris mentah (satu benar, satu salah). Hitung ulang jumlah lolos dan ditolak.

Ringkasan Bab

- Data dari luar harus **divalidasi saat program berjalan**; type hint saja tidak menjaga apa-apa.
- `dataclass` tidak memeriksa isi; **Pydantic** `BaseModel` memeriksa, mengonversi, dan memberi pesan galat per kolom.
- Aturan tambahan: `Field` (batas), `field_validator` (satu kolom), `model_validator` (gabungan kolom).
- **`pydantic-settings`** membaca pengaturan dari variabel lingkungan dan `.env`; urutan kekuatan: argumen > lingkungan > `.env` > bawaan; `SecretStr` menyamarkan rahasia.
- `TypeAdapter`, `model_validate_json`, dan `model_json_schema` memvalidasi daftar, JSON, dan menghasilkan skema.
- Proyek I4: data lima sektor divalidasi lewat gabungan bertanda; empat lolos, empat ditolak dengan alasan.

Selanjutnya, di Bab 35: async dan concurrency, cara mengerjakan banyak pekerjaan menunggu sekaligus dan menutup Seri I4.

Bab 35 — Async dan Concurrency: Mengerjakan Banyak Hal Sekaligus

Seri I4 — Bahasa Python Tingkat Lanjut

Di bab ini, Anda akan:

- Membedakan pekerjaan **menunggu** (I/O) dan pekerjaan **menghitung** (CPU)
- Menulis kode `async`/`await` dengan `asyncio` (`gather`, `TaskGroup`, `timeout`, `Semaphore`)
- Memakai **thread** dan **proses** lewat `concurrent.futures`
- Memahami GIL dan mode *free-threaded* Python 3.14, serta kapan memakai yang mana
- Mengunduh banyak data cuaca secara serempak dari server tiruan lokal: puncak proyek Seri I4

Semua skrip bab ini ada di folder `kode/bab35/`. Di versi web, Anda juga bisa mengunduh semuanya sekaligus (zip).

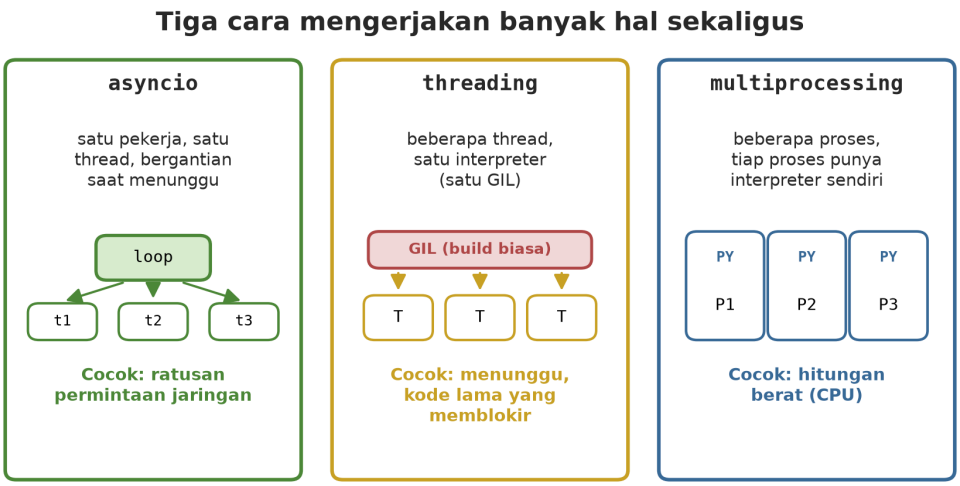
Bayangkan Anda mengambil data curah hujan dari 20 stasiun. Setiap permintaan ke jaringan butuh waktu menunggu, misalnya setengah detik. Dikerjakan satu per satu: 20 x 0,5 = 10 detik, dan hampir semuanya dihabiskan untuk **menunggu**. Seperti petani yang menyiram 20 petak dengan satu selang, menunggu air sampai di tiap petak sebelum pindah ke petak berikutnya. Padahal selama menunggu, tangannya bisa mengurus petak lain.

Itulah inti bab ini: mengatur supaya waktu menunggu tidak terbuang.

Dua jenis pekerjaan

Jenis	Contoh	Yang jadi penghambat
I/O-bound (banyak menunggu)	unduh data, baca disk, tanya basis data	menunggu jaringan atau disk
CPU-bound (banyak menghitung)	simulasi, olah angka berat	kecepatan prosesor

Pilihan alat bergantung pada jenis ini. Tiga alat utamanya:



Gambar 0.1: Tiga cara mengerjakan banyak hal sekaligus: asyncio, threading, multiprocessing.

`asyncio`: satu pekerja yang pandai bergantian

Fungsi biasa dikerjakan dari awal sampai akhir. Fungsi `async def` (disebut **coroutine**) bisa **berhenti sementara** di titik `await` dan menyerahkan giliran ke pekerjaan lain yang sedang menunggu. Satu pekerja (satu thread, disebut **event loop**) mengatur gilirannya.

```
# Bab 35 - Dasar async: coroutine, await, dan asyncio.run
import asyncio


async def ukur_ph(kolam):
    await asyncio.sleep(0.01)         # menunggu tanpa memblokir program lain
    return f"pH kolam {kolam} = 7.2"


# Memanggil fungsi async TIDAK menjalankannya: ia hanya membuat objek coroutine
objek = ukur_ph("A")
print(type(objek).__name__)
objek.close()                          # dibuang tanpa dijalankan


async def utama():
    hasil = await ukur_ph("A")        # await: jalankan dan tunggu hasilnya
    print(hasil)


asyncio.run(utama())                   # pintu masuk: jalankan event loop
```

```
coroutine
pH kolam A = 7.2
```

- Memanggil `ukur_ph("A")` **tidak menjalankannya**; hanya membuat objek bertipe `coroutine`. (Di skrip, objek itu langsung ditutup dengan `close()`.) Melupakan `await` adalah kesalahan pemula yang sangat umum, dan biasanya hanya menghasilkan peringatan.
- `await ukur_ph("A")` menjalankan dan menunggu hasilnya.
- `asyncio.run(utama())` adalah pintu masuk yang menyalakan event loop. Biasanya dipanggil **sekali** di bagian paling luar program.
- `asyncio.sleep(0.01)` adalah menunggu yang **tidak memblokir**. Bandingkan dengan `time.sleep()` biasa yang membekukan seluruh program.

`gather`: banyak pekerjaan menunggu bersama

```
# Bab 35 - Banyak pekerjaan menunggu sekaligus dengan gather
import asyncio

selesai = []


async def unduh(nama, detik):
    await asyncio.sleep(detik)
    selesai.append(nama)
    return nama, detik


async def secara_berurutan():
    for nama, detik in [("S1", 0.06), ("S2", 0.02), ("S3", 0.04)]:
        await unduh(nama, detik)
```

```
async def secara_serempak():
   return await asyncio.gather(unduh("S1", 0.06), unduh("S2", 0.02), unduh("S3", 0.04))


async def utama():
    loop = asyncio.get_running_loop()
    t0 = loop.time()
    await secara_berurutan()
    waktu_urut = loop.time() - t0
    print("urutan selesai (berurutan):", selesai)

    selesai.clear()
    t0 = loop.time()
    hasil = await secara_serempak()
    waktu_serempak = loop.time() - t0
    print("urutan selesai (serempak):", selesai)
    print("urutan hasil gather mengikuti urutan penulisan:", [n for n, _ in hasil])
     print("serempak lebih cepat dari 70% waktu berurutan:", waktu_serempak < 0.7
* waktu_urut)


asyncio.run(utama())
```

```
urutan selesai (berurutan): ['S1', 'S2', 'S3']
urutan selesai (serempak): ['S2', 'S3', 'S1']
urutan hasil gather mengikuti urutan penulisan: ['S1', 'S2', 'S3']
serempak lebih cepat dari 70% waktu berurutan: True
```

Tiga unduhan tiruan dengan lama menunggu 0,06, 0,02, dan 0,04 detik:

- **Berurutan**: selesai sesuai urutan dikerjakan: S1, S2, S3.
- **Serempak (`gather`)**: selesai sesuai **lama menunggu**: S2 (0,02), S3 (0,04), lalu S1 (0,06). Yang sebentar tidak harus menunggu yang lama.
- **Urutan hasil** `gather` tetap mengikuti urutan **penulisan** (S1, S2, S3), apa pun urutan selesainya. Itu terdokumentasi dan memudahkan: hasil ke-*n* selalu milik pekerjaan ke-*n*.
- Skrip tidak mencetak waktu mentah (berubah tiap dijalankan), hanya perbandingan: waktu serempak kurang dari 70% waktu berurutan (`True`). Total waktu berurutan sekitar 0,12 detik, serempak sekitar 0,06 detik.

`TaskGroup`, batas waktu, dan pembatas jumlah

```
# Bab 35 - TaskGroup, batas waktu, dan pembatas jumlah (Semaphore)
import asyncio

aktif = 0
puncak = 0


async def kerja(nomor, semaphore):
    global aktif, puncak
    async with semaphore:              # paling banyak 2 yang masuk sekaligus
        aktif += 1
        puncak = max(puncak, aktif)
        await asyncio.sleep(0.02)
        aktif -= 1
    return nomor * 10


async def lambat():
```

```
    await asyncio.sleep(1)


async def utama():
    semaphore = asyncio.Semaphore(2)
    async with asyncio.TaskGroup() as grup:
        tugas = [grup.create_task(kerja(i, semaphore)) for i in range(1, 7)]
    print("hasil:", [t.result() for t in tugas])
    print("paling banyak berjalan bersamaan:", puncak)

    try:
        async with asyncio.timeout(0.05):
            await lambat()
    except TimeoutError:
        print("dibatalkan: melewati batas waktu")


asyncio.run(utama())
```

```
hasil: [10, 20, 30, 40, 50, 60]
paling banyak berjalan bersamaan: 2
dibatalkan: melewati batas waktu
```

- **`asyncio.TaskGroup`** (Python 3.11+) memulai sekelompok tugas dan **menunggu semuanya** saat blok `async with` berakhir. Bila satu tugas gagal, yang lain dibatalkan: lebih aman daripada `gather` mentah.
- **`asyncio.Semaphore(2)`** membatasi hanya dua tugas yang masuk bagian dalamnya sekaligus. Buktinya: dari enam tugas, `puncak` tertinggi yang berjalan bersamaan adalah 2. Ini penting agar server tidak dibanjiri ratusan permintaan sekaligus.
- **`asyncio.timeout(0.05)`** (Python 3.11+) membatalkan pekerjaan yang melebihi batas, lalu menimbulkan `TimeoutError`. Pekerjaan `lambat()` yang butuh 1 detik dihentikan setelah 0,05 detik.

> TIPS
>
> Selalu beri **batas waktu** (`timeout`) dan **batas jumlah** (`Semaphore`) pada pekerjaan jaringan. Server luar bisa lambat atau macet, dan mengirim 1.000 permintaan sekaligus bisa dianggap serangan.

Thread: untuk pekerjaan menunggu pada kode biasa

Bagaimana bila fungsi Anda bukan `async` (misalnya pustaka lama yang memblokir)? Pakai **thread**: beberapa alur kerja dalam satu program. Cara termudah: `ThreadPoolExecutor`.

```
# Bab 35 - Thread untuk pekerjaan menunggu (I/O) dengan pustaka biasa
import threading
import time
from concurrent.futures import ThreadPoolExecutor


def baca_sensor(nama):
    time.sleep(0.05)                    # meniru menunggu jaringan atau disk
    return nama, len(nama) * 1.5


with ThreadPoolExecutor(max_workers=4) as kolam:
    hasil = list(kolam.map(baca_sensor, ["suhu", "ph", "oksigen", "keruh"]))
print(hasil)                            # map menjaga urutan masukan

# Data bersama di antara thread harus dilindungi Lock
total = 0
```

```
kunci = threading.Lock()


def tambah_banyak():
    global total
    for _ in range(100_000):
        with kunci:
            total += 1


threads = [threading.Thread(target=tambah_banyak) for _ in range(4)]
for t in threads:
    t.start()
for t in threads:
    t.join()
print("total dengan Lock:", total)
```

```
[('suhu', 6.0), ('ph', 3.0), ('oksigen', 10.5), ('keruh', 7.5)]
total dengan Lock: 400000
```

- `kolam.map(...)` menjalankan `baca_sensor` pada empat thread dan **menjaga urutan hasil** sesuai masukan: `suhu`, `ph`, `oksigen`, `keruh`. Itu membuat keluaran stabil.
- Di bagian bawah, empat thread menambah `total` sama-sama. Tanpa perlindungan, dua thread bisa membaca nilai lama bersamaan dan salah satu tambahan hilang (*race condition*). Dengan **`Lock`**, hanya satu thread masuk pada satu saat: totalnya tepat 400.000 (4 x 100.000).
- Untuk menjalankan fungsi biasa yang memblokir dari dalam kode `async`, ada `asyncio.to_thread(...)` (terdokumentasi sejak Python 3.9).

Proses: untuk pekerjaan menghitung

```
# Bab 35 - Proses terpisah untuk pekerjaan menghitung (CPU)
from concurrent.futures import ProcessPoolExecutor


def hitung_berat(n):
    """Pekerjaan CPU murni: jumlah kuadrat 1..n."""
    return sum(i * i for i in range(1, n + 1))


if __name__ == "__main__":            # WAJIB di Windows: tiap proses anak mengimpor
berkas ini
    tugas = [200_000, 300_000, 400_000, 500_000]
    with ProcessPoolExecutor(max_workers=4) as kolam:
        hasil = list(kolam.map(hitung_berat, tugas))
    for n, h in zip(tugas, hasil):
        print(n, h)
```

```
200000 2666686666700000
300000 9000045000050000
400000 21333413333400000
500000 41666791666750000
```

`ProcessPoolExecutor` menjalankan fungsi di **proses terpisah**: tiap proses adalah program Python tersendiri dengan memorinya sendiri. Hasil dikirim kembali ke program utama, dan `map` menjaga urutannya (200.000, 300.000, 400.000, 500.000).

⚠ AWAS

Di **Windows**, kode yang membuat proses baru **wajib** berada di bawah `if __name__ == "__main__":`. Alasannya: Windows memulai proses anak dengan metode *spawn*, yaitu menjalankan interpreter baru yang **mengimpor** berkas Anda dari awal. Tanpa pelindung itu, setiap proses anak akan mengulang kode pembuat proses dan menghasilkan galat `RuntimeError`. Dokumentasi multiprocessing menyebut spawn sebagai metode bawaan di Windows dan macOS, dan menganjurkan pelindung ini ("safe importing of main module"). Skrip di bab ini selalu memakainya bila membuat proses. Karena fungsi dan data dikirim antar-proses, keduanya harus bisa **di-pickle** (diserialisasi); fungsi harus didefinisikan di tingkat atas modul, bukan lambda (penulis mencoba lambda dan mendapat `PicklingError`).

GIL: mengapa thread tidak selalu mempercepat

Python standar punya **GIL** (*Global Interpreter Lock*), kunci yang memastikan hanya **satu thread** menjalankan kode Python pada satu saat dalam satu proses. Akibatnya, thread **tidak mempercepat pekerjaan CPU**; thread berguna untuk pekerjaan menunggu karena saat satu thread menunggu, yang lain bisa jalan. Mari buktikan di komputer ini:

```
# Bab 35 - GIL: thread tidak mempercepat pekerjaan CPU (pada build biasa), proses bisa
import sys
import time
from concurrent.futures import ProcessPoolExecutor, ThreadPoolExecutor


def hitung_berat(n):
    return sum(i * i for i in range(n))


def ukur(fungsi):
    t0 = time.perf_counter()
    fungsi()
    return time.perf_counter() - t0


if __name__ == "__main__":
    print("GIL aktif di Python ini:", sys._is_gil_enabled())
    beban = [5_000_000] * 4

    urut = ukur(lambda: [hitung_berat(n) for n in beban])
    with ThreadPoolExecutor(max_workers=4) as kolam:
        thread = ukur(lambda: list(kolam.map(hitung_berat, beban)))
    with ProcessPoolExecutor(max_workers=4) as kolam:
        proses = ukur(lambda: list(kolam.map(hitung_berat, beban)))

    print("4 thread mempercepat lebih dari 1,5x:", urut / thread > 1.5)
    print("4 proses mempercepat lebih dari 1,3x:", urut / proses > 1.3)
```

```
GIL aktif di Python ini: True
4 thread mempercepat lebih dari 1,5x: False
4 proses mempercepat lebih dari 1,3x: True
```

- `sys._is_gil_enabled()` melaporkan apakah GIL aktif di Python yang menjalankan skrip: `True`.
- Empat thread **tidak** mempercepat hitungan berat lebih dari 1,5 kali (`False`): kuncinya membuat mereka bergantian.
- Empat proses mempercepat lebih dari 1,3 kali (`True`), karena tiap proses punya interpreter dan GIL sendiri sehingga benar-benar berjalan paralel.

- Seperti sebelumnya, skrip hanya mencetak perbandingan, bukan detik. Hasil `True/False` ini diperiksa berkali-kali di komputer penulis; pada komputer Anda angka pastinya akan berbeda (misalnya bila inti prosesor lebih sedikit), sehingga ambang 1,3 kali bisa tidak tercapai di komputer dengan sedikit inti.

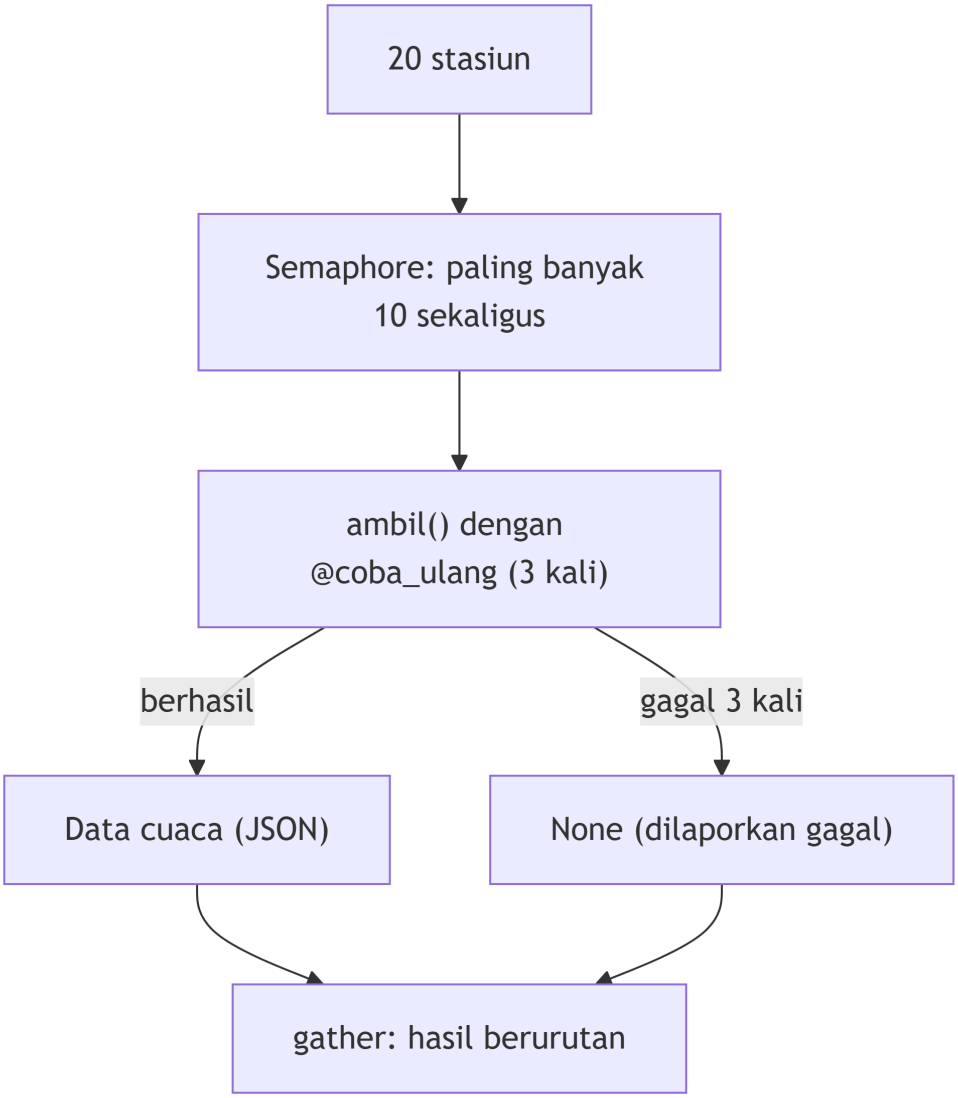
Mode *free-threaded*

Sejak Python 3.13 ada build opsional **free-threaded** dengan GIL dimatikan, sehingga thread bisa berjalan paralel di banyak inti. Menurut dokumentasi, Python 3.14 menyatakan dukungan resmi untuk mode ini (rujukan PEP 779 di bagian "What's New"), tetapi ia **bukan bawaan**: harus dipilih saat memasang atau membuat build tersendiri, dan tidak semua pustaka C siap untuk itu. Cara mengeceknya: `sys._is_gil_enabled()` dan `sysconfig.get_config_var("Py_GIL_DISABLED")`. Di komputer penulis hanya build biasa (GIL aktif) yang diuji; build free-threaded **belum diuji** di bab ini.

> **i** INGAT
>
> Panduan memilih:
>
> | Masalah Anda | Pilihan | Alasan |
> |---|---|---|
> | Ratusan permintaan jaringan, pustaka mendukung `async` | **asyncio** | ringan, satu thread, mudah dibatasi |
> | Menunggu, tetapi pustaka lama memblokir | **thread** (`ThreadPoolExecutor`) | tanpa menulis ulang jadi `async` |
> | Hitungan berat murni (CPU) | **proses** (`ProcessPoolExecutor`) | melewati GIL dengan banyak interpreter |
> | Hanya beberapa pekerjaan, sederhana | tidak perlu semuanya | kode biasa lebih mudah dirawat |

Proyek Bersambung I4: mengunduh banyak data cuaca secara async

Puncak proyek Seri I4: mengambil data 20 stasiun cuaca/harga sekaligus. Agar aman dan stabil, **server-nya tiruan**: dibuat di komputer ini sendiri dengan `aiohttp`, di alamat lokal `127.0.0.1` pada port yang dipilih sistem. **Tidak ada internet** yang dipakai. Server meniru jaringan lambat (0,02 detik) dan gangguan: stasiun S07 gagal sekali (status 503), S13 gagal tiga kali.



Alur proyek: 20 permintaan dibatasi 10 sekaligus, tiap permintaan dicoba ulang bila gagal, hasil dikumpulkan berurutan.

```
# Bab 35 - Proyek bersambung I4: mengunduh banyak data cuaca dari server TIRUAN lokal
import asyncio
import logging
import sys
from functools import wraps

import aiohttp
from aiohttp import web

logging.basicConfig(stream=sys.stdout,  level=logging.WARNING,  format="%(levelname)s
%(message)s")
log = logging.getLogger("unduh")

gangguan = {"S07": 1, "S13": 3}          # stasiun tiruan yang gagal beberapa kali
```

```
# ---------- server tiruan (hanya di komputer ini, tanpa internet) ----------
async def beri_data(request):
    stasiun = request.match_info["stasiun"]
    await asyncio.sleep(0.02)            # meniru lambatnya jaringan
    if gangguan.get(stasiun, 0) > 0:
        gangguan[stasiun] -= 1
        return web.Response(status=503, text="sibuk")
    nomor = int(stasiun[1:])
    return web.json_response({"stasiun": stasiun, "curah_hujan_mm": nomor * 1.5,
                              "harga_gabah": 6000 + nomor * 10})


# ---------- klien async dengan coba-ulang (decorator Bab 31, versi async) ----------
def coba_ulang(maksimal=3):
    def decorator(fungsi):
        @wraps(fungsi)
        async def pembungkus(sesi, alamat, stasiun):
            for percobaan in range(1, maksimal + 1):
                try:
                    return await fungsi(sesi, alamat, stasiun)
                except aiohttp.ClientResponseError as e:
                    log.warning("%s percobaan %d: status %d", stasiun, percobaan, e.status)
                    if percobaan == maksimal:
                        raise
                    await asyncio.sleep(0.01)
        return pembungkus
    return decorator


@coba_ulang(maksimal=3)
async def ambil(sesi, alamat, stasiun):
    async with sesi.get(f"{alamat}/data/{stasiun}") as jawaban:
        jawaban.raise_for_status()
        return await jawaban.json()


async def ambil_semua(sesi, alamat, daftar, batas):
    semaphore = asyncio.Semaphore(batas)

    async def satu(stasiun):
        async with semaphore:
            try:
                return await ambil(sesi, alamat, stasiun)
            except aiohttp.ClientResponseError:
                return None

    return await asyncio.gather(*(satu(s) for s in daftar))


async def utama():
    aplikasi = web.Application()
    aplikasi.add_routes([web.get("/data/{stasiun}", beri_data)])
    runner = web.AppRunner(aplikasi)
    await runner.setup()
     situs = web.TCPSite(runner, "127.0.0.1", 0)        # port 0: sistem memilihkan
port bebas
    await situs.start()
    port = situs._server.sockets[0].getsockname()[1]
    alamat = f"http://127.0.0.1:{port}"
```

```
    daftar = [f"S{n:02d}" for n in range(1, 21)]
    async with aiohttp.ClientSession() as sesi:
        loop = asyncio.get_running_loop()
        t0 = loop.time()
        hasil = await ambil_semua(sesi, alamat, daftar, batas=10)
        waktu = loop.time() - t0

    await runner.cleanup()
    berhasil = [h for h in hasil if h]
    gagal = [s for s, h in zip(daftar, hasil) if h is None]
    print("stasiun diminta:", len(daftar), "| berhasil:", len(berhasil), "| gagal:",
gagal)
    rata = sum(h["curah_hujan_mm"] for h in berhasil) / len(berhasil)
    print("rata-rata curah hujan (mm):", round(rata, 2))
        print("harga gabah tertinggi di stasiun:", max(berhasil, key=lambda h:
h["harga_gabah"])["stasiun"])
    print("lebih cepat dari satu per satu (20 x 0,02 detik = 0,4 detik):", waktu < 0.3)


if __name__ == "__main__":
    asyncio.run(utama())
```

```
WARNING S07 percobaan 1: status 503
WARNING S13 percobaan 1: status 503
WARNING S13 percobaan 2: status 503
WARNING S13 percobaan 3: status 503
stasiun diminta: 20 | berhasil: 19 | gagal: ['S13']
rata-rata curah hujan (mm): 15.55
harga gabah tertinggi di stasiun: S20
lebih cepat dari satu per satu (20 x 0,02 detik = 0,4 detik): True
```

Telusuri hasilnya:

- Decorator `coba_ulang` dari Bab 31 dipakai lagi, kini versi `async` (dengan `await asyncio.sleep` untuk jeda). Ini menyambungkan Bab 31 dan 35.
- Log peringatan menunjukkan: S07 gagal pada percobaan 1 lalu berhasil pada percobaan 2 (tidak ada peringatan kedua), sedangkan S13 gagal pada percobaan 1, 2, dan 3, sehingga menyerah dan hasilnya `None`.
- Ringkasan: 20 diminta, **19 berhasil**, gagal: `['S13']`. Rata-rata curah hujan dihitung hanya dari yang berhasil: 15,55 mm (angka tiruan: stasiun nomor *n* mengembalikan *n* x 1,5). Harga gabah tertinggi di stasiun S20.
- Dengan `Semaphore(10)`, dua puluh permintaan selesai jauh lebih cepat daripada satu per satu (20 x 0,02 = 0,4 detik): perbandingannya `True`. Skrip tidak mencetak detik.
- Server dan klien berjalan **dalam satu event loop** yang sama, sehingga seluruh percobaan muat dalam satu skrip.
- Server dimatikan rapi dengan `runner.cleanup()`; sesi klien ditutup oleh `async with`, yaitu **context manager asinkron** (kerabat async dari Bab 32).

> ⚠ AWAS
>
> Ini **simulasi**. Pada API sungguhan, ikuti batas laju (*rate limit*) dan aturan penggunaan penyedia data, dan jangan menembaknya dengan ratusan permintaan tanpa izin. Contoh di buku ini tidak menghubungi layanan luar mana pun.

Latihan

1. Ubah `02_gather_serempak.py`: tambahkan stasiun keempat dengan lama menunggu 0,01 detik. Perkirakan urutan selesai serempak lebih dulu, lalu jalankan dan cocokkan.

2. Ubah `Semaphore(2)` di `03_taskgroup_timeout.py` menjadi 3. Berapa `puncak` yang tercetak sekarang? Mengapa?
3. Ubah `batas=10` di `07_proyek_unduh_async.py` menjadi `batas=1`. Hasil ringkasan (berhasil dan gagal) tetap sama; hanya baris terakhir (perbandingan kecepatan) berubah. Mengapa?
4. **Tantangan:** tambahkan stasiun ketiga yang rusak, `S18`, ke `gangguan` dengan nilai 5 (gagal lima kali), jalankan ulang, dan hitung ulang rata-rata curah hujan secara manual untuk memastikan cocok dengan keluaran.

Ringkasan Bab

- Bedakan pekerjaan **menunggu (I/O)** dan **menghitung (CPU)**; pilihan alat bergantung padanya.
- **`asyncio`**: `async def` + `await` + `asyncio.run`; `gather` menjaga urutan hasil; `TaskGroup`, `timeout`, dan `Semaphore` mengatur dan membatasi tugas.
- **Thread** (`ThreadPoolExecutor`) cocok untuk menunggu pada kode biasa; data bersama dilindungi `Lock`.
- **Proses** (`ProcessPoolExecutor`) cocok untuk menghitung; di Windows wajib `if __name__ == "__main__":`.
- **GIL** membuat thread tidak mempercepat pekerjaan CPU di Python standar; build **free-threaded** adalah pilihan terpisah.
- Proyek I4: 20 stasiun diunduh serempak dari server tiruan, dengan coba-ulang dan batas jumlah.

Penutup Seri I4

Selamat! Anda menyelesaikan **Seri I4** dan Tingkat Intermediate jilid kedua ini. Perhatikan apa yang sekarang bisa dilakukan kode Anda dibandingkan di awal seri:

- membaca berkas besar **satu baris per satu** tanpa membebani memori (generator, Bab 30),
- membungkus fungsi dengan **log dan coba-ulang** tanpa mengubah isinya (decorator, Bab 31),
- **membuka dan menutup** sumber daya dengan pasti, dan menulis laporan dengan aman (context manager, Bab 32),
- **berlabel tipe** dan diperiksa mypy sebelum berjalan (type hint, Bab 33),
- menolak data lapangan yang buruk dengan **pesan jelas** dan membaca pengaturan dengan aman (Pydantic, Bab 34),
- mengunduh banyak data **serempak** dengan batas dan coba-ulang (async, Bab 35).

Kode sudah kuat. Seri I5 mengubahnya menjadi aplikasi yang bisa dipakai orang lain: layanan web (Bab 36), Python di browser (Bab 37), peta dan data spasial (Bab 38 sampai 40), otomasi dan antarmuka (Bab 41 dan 42), serta dasbor (Bab 43). Pydantic dan async yang baru Anda pelajari akan langsung dipakai di FastAPI.

Selanjutnya, di Bab 36: web development, yaitu Flask, FastAPI, pengantar Django, dan REST API.

Bab 36 — Web Development: Membuka Warung Data di Internet

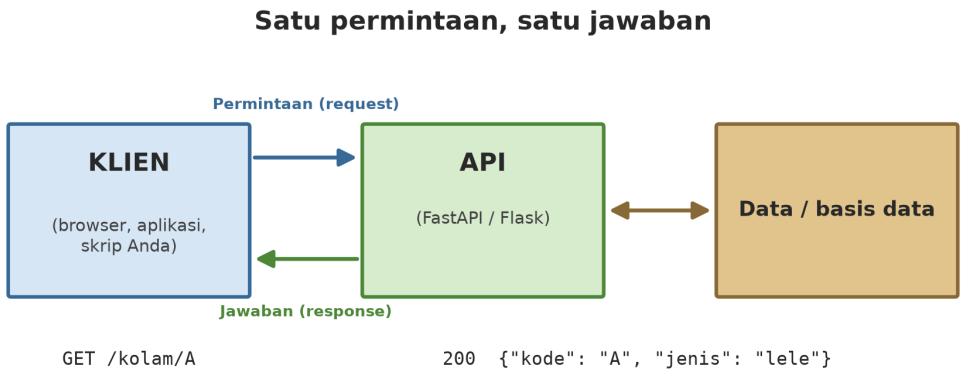
Seri I5 — Aplikasi Pertama: Web, Peta, dan Dasbor

Di bab ini, Anda akan:

- Memahami cara kerja web: **permintaan** (request) dan **jawaban** (response)
- Membuat aplikasi web dan **REST API** dengan **Flask**
- Membuat API modern dengan **FastAPI**, lengkap dengan validasi Pydantic dan dokumentasi otomatis
- Mengenal sekilas **Django**, kerangka kerja "serba ada"
- Menguji aplikasi web tanpa menyalakan server, lalu menyalakan server asli sebentar saja
- Memulai API data unit usaha: proyek bersambung Seri I5

Semua skrip bab ini ada di folder `kode/bab36/`. Di versi web, Anda juga bisa mengunduh semuanya sekaligus (zip).

Sampai Bab 35, program Anda hidup di satu komputer: komputer Anda sendiri. Padahal data kebun, kolam, dan kandang sering dibutuhkan orang lain: penyuluh di kantor, petugas di lapangan, atau aplikasi di ponsel. Cara paling umum untuk berbagi adalah **membuka warung data**: program Anda duduk menunggu pesanan ("tolong data kolam A"), lalu menjawab. Itulah **web server** dan **API** (*Application Programming Interface*, "pintu" yang dipakai program lain untuk meminta data).



Gambar 0.1: Satu permintaan, satu jawaban: dasar semua aplikasi web.

Cara kerja web dalam satu paragraf

Setiap interaksi web adalah **pesanan dan jawaban**. Klien (browser, aplikasi, atau skrip Python Anda) mengirim **permintaan** berisi *metode* dan *alamat*, misalnya `GET /kolam/A`. Server membalas dengan **kode status** dan isi. Isinya biasanya halaman HTML atau data **JSON** (teks berformat seperti dict, Bab 14).

Metode	Artinya	Contoh
`GET`	Minta data	Lihat daftar kolam
`POST`	Tambah data baru	Catat kolam baru
`PUT` / `PATCH`	Ubah data	Perbarui jumlah ikan

Metode	Artinya	Contoh
`DELETE`	Hapus data	Hapus kolam

Kode status	Artinya
`200`	Berhasil
`201`	Berhasil dibuat (setelah `POST`)
`404`	Tidak ditemukan
`409`	Bentrok (mis. kode sudah ada)
`422`	Isian tidak lolos pemeriksaan

REST API hanyalah kebiasaan menata alamat dan metode itu: alamat menyebut *benda* (`/kolam/A`), metode menyebut *tindakan*.

> **i** INGAT
>
> Kode di bab ini **tidak menyalakan server** untuk ditunggu terus-menerus. Kita memakai **klien uji** (peniru browser yang menelepon aplikasi langsung dari dalam Python). Hasilnya sama persis, tetapi cepat, aman, dan keluarannya tetap. Satu skrip (07) menyalakan server asli sebentar di thread lalu mematikannya.

Flask: warung paling kecil

Flask adalah kerangka kerja web yang ringan: Anda mulai dengan beberapa baris, lalu menambah sendiri sesuai kebutuhan. Setiap alamat dihubungkan ke sebuah fungsi dengan *decorator* `@app.get(...)` (dari Bab 31).

```
# Bab 36 - Flask: aplikasi web terkecil, diuji tanpa menyalakan server
from flask import Flask, jsonify, request

app = Flask(__name__)


@app.get("/")
def beranda():
    return "Halo dari kebun!"


@app.get("/sapa/<nama>")                          # bagian <nama> dari alamat masuk
ke fungsi
def sapa(nama):
    return {"pesan": f"Selamat pagi, {nama}"}   # dict otomatis menjadi JSON


@app.get("/cari")
def cari():
    komoditas = request.args.get("komoditas", "semua")   # ?komoditas=kopi
    return jsonify(dicari=komoditas)


klien = app.test_client()                          # peniru browser; tidak membuka
port jaringan
r = klien.get("/")
print(r.status_code, r.content_type, r.get_data(as_text=True))
r = klien.get("/sapa/Budi")
print(r.status_code, r.content_type, r.get_json())
```

```
print(klien.get("/cari?komoditas=kopi").get_json())
r = klien.get("/tidak-ada")
print(r.status_code, r.status)
```

```
200 text/html; charset=utf-8 Halo dari kebun!
200 application/json {'pesan': 'Selamat pagi, Budi'}
{'dicari': 'kopi'}
404 404 NOT FOUND
```

- `"/"` mengembalikan teks, jadi jenis isinya `text/html`. Mengembalikan **dict** otomatis menjadi JSON (`application/json`).
- `<nama>` pada alamat adalah **parameter alamat**: isi `Budi` masuk ke fungsi sebagai variabel `nama`.
- `request.args` membaca **parameter query**, bagian setelah tanda `?` di alamat.
- Alamat yang tidak terdaftar menghasilkan `404 NOT FOUND` otomatis.
- `app.test_client()` adalah klien uji: ia memanggil aplikasi tanpa membuka port jaringan.

REST API dengan Flask

Sekarang warung yang bisa mencatat: daftar kolam disimpan di sebuah dict (di memori, jadi hilang saat program berhenti; penyimpanan tetap datang di bab-bab berikutnya).

```
# Bab 36 - REST API dengan Flask: daftar kolam (disimpan di memori)
from flask import Flask, abort, jsonify, request

app = Flask(__name__)
kolam = {"A": {"jenis": "lele", "jumlah": 800}, "B": {"jenis": "nila", "jumlah": 500}}


@app.get("/kolam")
def daftar():
    return jsonify(kolam)


@app.get("/kolam/<kode>")
def satu(kode):
    if kode not in kolam:
        abort(404, description=f"kolam {kode} tidak ada")
    return jsonify(kolam[kode])


@app.post("/kolam/<kode>")
def tambah(kode):
    isi = request.get_json()                    # tubuh permintaan (JSON)
    kolam[kode] = {"jenis": isi["jenis"], "jumlah": isi["jumlah"]}
    return jsonify(kolam[kode]), 201            # 201 = Created


@app.delete("/kolam/<kode>")
def hapus(kode):
    kolam.pop(kode, None)
    return "", 204                              # 204 = berhasil, tanpa isi


@app.errorhandler(404)
def tidak_ada(galat):
    return jsonify(galat=galat.description), 404


k = app.test_client()
```

```
print("GET       /kolam           ->",  k.get("/kolam").status_code,  sorted(k.get("/
kolam").get_json()))
r = k.post("/kolam/C", json={"jenis": "lele", "jumlah": 650})
print("POST   /kolam/C    ->", r.status_code, r.get_json())
print("GET    /kolam/C    ->", k.get("/kolam/C").get_json())
print("DELETE /kolam/C    ->", k.delete("/kolam/C").status_code)
r = k.get("/kolam/C")
print("GET    /kolam/C    ->", r.status_code, r.get_json())
```

```
GET    /kolam      -> 200 ['A', 'B']
POST   /kolam/C    -> 201 {'jenis': 'lele', 'jumlah': 650}
GET    /kolam/C    -> {'jenis': 'lele', 'jumlah': 650}
DELETE /kolam/C    -> 204
GET    /kolam/C    -> 404 {'galat': 'kolam C tidak ada'}
```

Telusuri lima baris keluaran itu: `GET` daftar menghasilkan 200 dengan kode `A` dan `B`; `POST` menambah kolam C dan dijawab **201**; `DELETE` dijawab **204** (berhasil, tanpa isi); `GET` kolam C sesudah dihapus dijawab **404** dengan pesan yang kita tulis lewat `errorhandler`. Perhatikan: Flask **tidak memeriksa** isi `POST` kita. Bila `jumlah` hilang, kode kita akan berhenti dengan `KeyError`. Pemeriksaan itu kita tulis sendiri. Di sinilah FastAPI unggul.

FastAPI: tipe data yang bekerja untuk Anda

FastAPI memakai *type hint* (Bab 33) dan **Pydantic** (Bab 34) untuk memeriksa data otomatis dan membuat dokumentasi sendiri. Fungsinya boleh biasa (`def`) atau **async** (Bab 35).

```
# Bab 36 - FastAPI: parameter alamat dan parameter query dengan tipe data
from fastapi import FastAPI
from fastapi.testclient import TestClient

app = FastAPI()


@app.get("/")
def beranda():
    return {"pesan": "Halo dari kebun!"}


@app.get("/kolam/{kode}")
def kolam(kode: str, jumlah_min: int = 0):            # kode dari alamat; jumlah_min
dari ?jumlah_min=
    return {"kode": kode, "jumlah_min": jumlah_min}


@app.get("/panen/{blok}")
async def panen(blok: str):                          # fungsi async boleh (lihat Bab 35)
    return {"blok": blok, "hasil_kg": 245.5}


klien = TestClient(app)
print(klien.get("/").json())
print(klien.get("/kolam/A?jumlah_min=500").json())
print(klien.get("/panen/A").json())

# Tipe data diperiksa otomatis: "banyak" bukan bilangan bulat
r = klien.get("/kolam/A?jumlah_min=banyak")
print(r.status_code)
print(r.json()["detail"][0]["msg"])
```

```
{'pesan': 'Halo dari kebun!'}
{'kode': 'A', 'jumlah_min': 500}
{'blok': 'A', 'hasil_kg': 245.5}
422
Input should be a valid integer, unable to parse string as an integer
```

`jumlah_min: int = 0` berarti "bilangan bulat, bawaan 0". Saat kita mengirim `?jumlah_min=banyak`, FastAPI **menolak** dengan kode `422` dan pesan jelas (`Input should be a valid integer...`) tanpa satu baris pemeriksaan pun yang kita tulis.

Tubuh permintaan dengan model Pydantic

```
# Bab 36 - FastAPI + Pydantic: tubuh permintaan yang divalidasi
from fastapi import FastAPI, HTTPException
from fastapi.testclient import TestClient
from pydantic import BaseModel, Field

app = FastAPI()


class KolamBaru(BaseModel):
    jenis: str
    jumlah: int = Field(gt=0)
    ph_air: float = Field(ge=0, le=14)


class KolamKeluar(BaseModel):                       # bentuk jawaban
    kode: str
    jenis: str
    jumlah: int


gudang: dict[str, KolamBaru] = {}


@app.post("/kolam/{kode}", response_model=KolamKeluar, status_code=201)
def tambah(kode: str, kolam: KolamBaru):
    gudang[kode] = kolam
     return {"kode": kode, **kolam.model_dump()}        # ph_air tidak ikut keluar:
disaring response_model


@app.get("/kolam/{kode}", response_model=KolamKeluar)
def baca(kode: str):
    if kode not in gudang:
        raise HTTPException(status_code=404, detail=f"kolam {kode} tidak ada")
    return {"kode": kode, **gudang[kode].model_dump()}


k = TestClient(app)
r = k.post("/kolam/A", json={"jenis": "lele", "jumlah": 800, "ph_air": 7.2})
print(r.status_code, r.json())
r = k.post("/kolam/B", json={"jenis": "nila", "jumlah": -5, "ph_air": 17})
print(r.status_code)
for galat in r.json()["detail"]:
    print(" ", galat["loc"], "->", galat["msg"])
r = k.get("/kolam/Z")
print(r.status_code, r.json())
```

```
201 {'kode': 'A', 'jenis': 'lele', 'jumlah': 800}
422
  ['body', 'jumlah'] -> Input should be greater than 0
  ['body', 'ph_air'] -> Input should be less than or equal to 14
404 {'detail': 'kolam Z tidak ada'}
```

- `KolamBaru` adalah model Pydantic untuk data **masuk**: `jumlah` harus lebih dari 0 dan `ph_air` antara 0 dan 14.
- `KolamKeluar` adalah bentuk jawaban. `response_model` **menyaring** isi jawaban: `ph_air` tidak ikut keluar pada baris pertama keluaran. Ini cara aman agar data internal tidak bocor.
- Pada kolam B, dua aturan dilanggar sekaligus, dan FastAPI melaporkan **keduanya**: `jumlah` dan `ph_air`, lengkap dengan lokasinya (`['body', 'jumlah']`).
- `HTTPException` mengubah masalah kita menjadi kode status yang tepat (404).

Dokumentasi yang tertulis sendiri

```python
# Bab 36 - FastAPI membuat dokumentasi API sendiri (OpenAPI)
from fastapi import FastAPI
from fastapi.testclient import TestClient

app = FastAPI(title="API Kebun", version="0.1")


@app.get("/kolam/{kode}", summary="Ambil satu kolam", tags=["kolam"])
def kolam(kode: str):
    """Mengembalikan data satu kolam menurut kodenya."""
    return {"kode": kode}


@app.get("/panen", summary="Daftar panen", tags=["panen"])
def panen(blok: str | None = None):
    return []


k = TestClient(app)
skema = k.get("/openapi.json").json()
print("judul:", skema["info"]["title"], "| versi:", skema["info"]["version"])
for alamat, metode in skema["paths"].items():
    for nama, rincian in metode.items():
        print(f"{nama.upper():4s} {alamat:15s} {rincian['summary']}")
print("halaman /docs (Swagger UI) ada:", k.get("/docs").status_code == 200)
print("halaman /redoc ada:", k.get("/redoc").status_code == 200)
```

```
judul: API Kebun | versi: 0.1
GET  /kolam/{kode}   Ambil satu kolam
GET  /panen          Daftar panen
halaman /docs (Swagger UI) ada: True
halaman /redoc ada: True
```

Dari kode saja, FastAPI membuat **skema OpenAPI** (standar penjelasan API) di `/openapi.json`, serta halaman dokumentasi interaktif `/docs` (Swagger UI) dan `/redoc`. Penyuluh atau pengembang aplikasi ponsel bisa mencoba API Anda dari browser tanpa membaca kode.

> **TIPS**
>
> Untuk membuka dokumentasi itu di komputer Anda, jalankan server sungguhan di terminal: `uvicorn nama_berkas:app --reload`, lalu buka `http://127.0.0.1:8000/docs`. Kita tidak menjalankannya di bab ini karena server menunggu tanpa akhir; itu tugas terminal Anda.

Pengantar Django

Django adalah kerangka kerja "serba ada" (*batteries included*): sistem pengguna, panel admin, basis data (ORM), dan templat HTML sudah tersedia. Cocok untuk situs besar yang penuh halaman dan data. Proyek Django sungguhan dibuat dengan perintah `django-admin startproject nama` yang menghasilkan beberapa berkas. Perintah itu tidak kita jalankan di sini. Sebagai perkenalan, berikut proyek Django **satu berkas**:

```
# Bab 36 - Django: proyek satu berkas (hanya untuk perkenalan)
import django
from django.conf import settings

settings.configure(
    DEBUG=False,
    SECRET_KEY="contoh-saja-bukan-rahasia",
    ROOT_URLCONF=__name__,                          # daftar alamat ada di berkas ini
    ALLOWED_HOSTS=["testserver"],
)
django.setup()

from django.http import JsonResponse
from django.test import Client
from django.urls import path


def beranda(request):
    return JsonResponse({"pesan": "Halo dari Django"})


def kolam(request, kode):
    return JsonResponse({"kode": kode, "jenis": request.GET.get("jenis", "?")})


urlpatterns = [path("", beranda), path("kolam/<str:kode>/", kolam)]

klien = Client()
print(klien.get("/").json())
print(klien.get("/kolam/A/?jenis=lele").json())
print(klien.get("/tidak-ada/").status_code)
```

```
{'pesan': 'Halo dari Django'}
{'kode': 'A', 'jenis': 'lele'}
404
```

`settings.configure(...)` mengganti berkas `settings.py`; `urlpatterns` adalah daftar alamat; `Client` adalah klien uji milik Django. Bagian `SECRET_KEY` di sini hanya teks contoh. Pada proyek nyata, kunci itu harus rahasia dan dibaca dari variabel lingkungan (Bab 34).

	Flask	FastAPI	Django
Gaya	Mikro, bebas	Modern, berbasis tipe	Serba ada
Validasi data	Tulis sendiri	Otomatis (Pydantic)	Formulir dan model
Dokumentasi API	Tambahan	Otomatis	Tambahan
Cocok untuk	Prototipe, layanan kecil	**API** data dan model	Situs lengkap dengan admin

Kita memakai **FastAPI** untuk proyek bersambung karena tujuan kita adalah API data.

Server asli, sebentar saja

Klien uji tidak melewati jaringan. Untuk membuktikan bahwa aplikasi benar-benar bisa dipanggil lewat HTTP, skrip berikut menyalakan **uvicorn** (server untuk FastAPI) di sebuah *thread* (Bab 35), memanggilnya dengan `httpx`, lalu mematikannya.

```
# Bab 36 - Menyalakan server asli (uvicorn) di thread, memanggilnya lewat jaringan, lalu mematikannya
import socket
import threading
import time

import httpx
import uvicorn
from fastapi import FastAPI

app = FastAPI()


@app.get("/ping")
def ping():
    return {"hidup": True}


with socket.socket() as s:                          # minta nomor port bebas dari sistem
    s.bind(("127.0.0.1", 0))
    port = s.getsockname()[1]

server    =    uvicorn.Server(uvicorn.Config(app,    host="127.0.0.1",    port=port, log_level="warning"))
thread = threading.Thread(target=server.run, daemon=True)
thread.start()
while not server.started:                           # tunggu sampai siap
    time.sleep(0.05)

r  =  httpx.get(f"http://127.0.0.1:{port}/ping")   #  panggilan  HTTP  sungguhan  lewat jaringan lokal
print(r.status_code, r.json())

server.should_exit = True                           # minta server berhenti dengan sopan
thread.join(timeout=5)
print("server masih hidup:", thread.is_alive())
```

```
200 {'hidup': True}
server masih hidup: False
```

`server.should_exit = True` meminta server berhenti dengan sopan; `thread.join()` menunggu sampai selesai. Hasil terakhir `False` membuktikan server sudah mati. Port diminta dari sistem (angka 0 berarti "pilihkan yang kosong") supaya tidak bentrok dengan program lain.

> ⚠ AWAS
>
> Server pengembangan (uvicorn tanpa pengaman, atau `app.run()` milik Flask) **tidak** untuk dibuka ke internet. Menyajikan aplikasi dengan aman (HTTPS, proses banyak, pembatasan akses) dibahas di tingkat Advanced.

Proyek Bersambung I5: API data unit usaha

Awal dari "Sistem Informasi Hutan, Kebun, Pertanian, Perikanan, Peternakan" versi web: daftar **unit usaha** (petak, kolam, kandang) yang bisa dilihat, disaring per sektor, dan ditambah. Luas petak diambil dari angka Bab 38 (data tiruan).

```
# Bab 36 - Proyek bersambung I5: API data unit usaha (awal)
from typing import Literal

from fastapi import FastAPI, HTTPException
from fastapi.testclient import TestClient
from pydantic import BaseModel, Field


class Unit(BaseModel):
    sektor: Literal["hutan", "kebun", "sawah", "kolam", "kandang"]
    kode: str
    luas_ha: float = Field(gt=0)


app = FastAPI(title="Sistem Informasi Usaha")
unit_db: dict[str, Unit] = {
    "H1": Unit(sektor="hutan", kode="H1", luas_ha=9.0),
    "S1": Unit(sektor="sawah", kode="S1", luas_ha=3.0),
    "S2": Unit(sektor="sawah", kode="S2", luas_ha=2.25),
}


@app.get("/unit", response_model=list[Unit])
def daftar(sektor: str | None = None):
    return [u for u in unit_db.values() if sektor is None or u.sektor == sektor]


@app.get("/unit/{kode}", response_model=Unit)
def satu(kode: str):
    if kode not in unit_db:
        raise HTTPException(404, f"unit {kode} tidak ada")
    return unit_db[kode]


@app.post("/unit", response_model=Unit, status_code=201)
def tambah(unit: Unit):
    if unit.kode in unit_db:
        raise HTTPException(409, f"unit {unit.kode} sudah ada")        # 409 = konflik
    unit_db[unit.kode] = unit
    return unit


k = TestClient(app)
print("sawah:", [u["kode"] for u in k.get("/unit", params={"sektor": "sawah"}).json()])
print("tambah K1:", k.post("/unit", json={"sektor": "kebun", "kode": "K1", "luas_ha":
4.0}).status_code)
print("tambah K1 lagi:", k.post("/unit", json={"sektor": "kebun", "kode": "K1",
"luas_ha": 4.0}).status_code)
print("sektor salah:", k.post("/unit", json={"sektor": "empang", "kode": "E1",
"luas_ha": 1}).status_code)
print("total luas (ha):", sum(u["luas_ha"] for u in k.get("/unit").json()))
```

```
sawah: ['S1', 'S2']
tambah K1: 201
tambah K1 lagi: 409
```

```
sektor salah: 422
total luas (ha): 18.25
```

`Literal[...]` membatasi sektor pada lima pilihan, jadi sektor `empang` ditolak dengan **422**. Kode yang sudah ada ditolak dengan **409**. Total luas 18,25 ha adalah 9,0 + 3,0 + 2,25 (unit awal) + 4,0 (K1 yang berhasil ditambah). Data masih di memori; di bab-bab berikutnya API ini akan membaca data peta dan dasbor.

Latihan

1. Di `01_flask_dasar.py`, tambah alamat `/panen/<blok>` yang mengembalikan `{"blok": ..., "hasil_kg": 245.5}`. Uji dengan `test_client`.
2. Di `03_fastapi_dasar.py`, ubah `jumlah_min: int` menjadi `float` dan coba `?jumlah_min=2.5`. Lalu kembalikan ke `int`: apa yang terjadi?
3. Di `08_proyek_api_unit.py`, tambah alamat `DELETE /unit/{kode}` yang menjawab 204, dan 404 bila kode tidak ada. Uji kedua kasus.
4. **Tantangan:** tambah alamat `GET /ringkasan` yang mengembalikan total luas per sektor sebagai dict, memakai `defaultdict` atau `dict.get`.

Ringkasan Bab

- Web = **permintaan** (metode + alamat) dan **jawaban** (kode status + isi, sering JSON). **REST** menata alamat sebagai benda dan metode sebagai tindakan.
- **Flask** ringan dan bebas; validasi harus ditulis sendiri.
- **FastAPI** memeriksa tipe data, memakai Pydantic, menyaring jawaban dengan `response_model`, dan membuat dokumentasi `/docs` otomatis.
- **Django** serba ada, cocok untuk situs lengkap.
- Aplikasi bisa diuji tanpa server memakai klien uji (`test_client`, `TestClient`, `Client`).
- Server pengembangan tidak untuk dibuka ke internet.

Selanjutnya, di Bab 37: Python yang berjalan di dalam browser, tanpa server sama sekali.

Bab 37 — Python di Browser: Tanpa Server, Tanpa Pasang

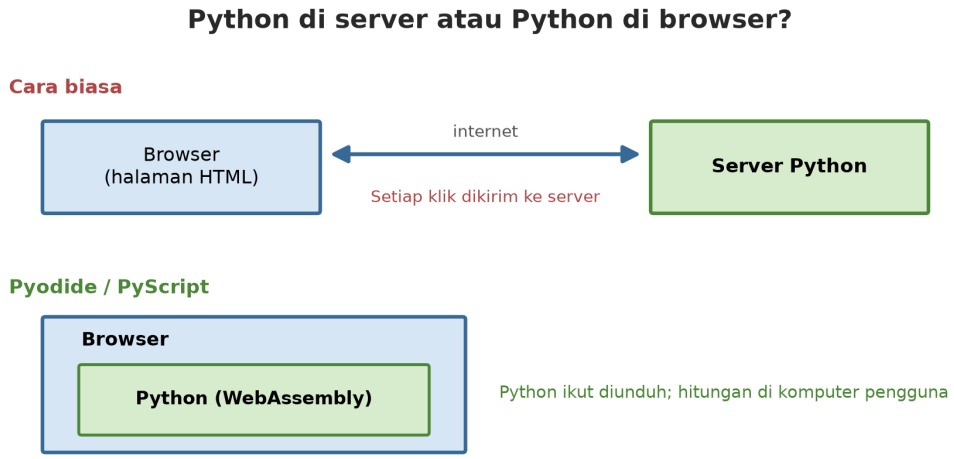
Seri I5 — Aplikasi Pertama: Web, Peta, dan Dasbor

Di bab ini, Anda akan:

- Memahami bagaimana Python bisa berjalan **di dalam browser** lewat **WebAssembly**
- Menjalankan kode Python di **Pyodide** dan memahami batasnya
- Membuat halaman **PyScript** yang interaktif tanpa server
- Mengenal **JupyterLite** (notebook yang berjalan di browser)
- Menilai kapan cara ini cocok, dan kapan tidak

Semua skrip dan halaman demo bab ini ada di folder `kode/bab37/`. Di versi web, Anda juga bisa mengunduh semuanya sekaligus (zip).

Di Bab 36, setiap hitungan terjadi di **server**: browser meminta, server menghitung, browser menampilkan. Bayangkan penyuluh di desa yang sinyalnya putus-nyambung dan butuh **kalkulator biomassa kolam**. Menyalakan dan merawat server hanya untuk rumus sekecil itu terasa berlebihan. Bagaimana kalau Python ikut **diunduh bersama halaman**, dan semua hitungan terjadi di ponsel atau laptop pengguna? Itulah Python di browser.



Gambar 0.1: Kiri-kanan: cara biasa (hitungan di server) dan Python di browser (hitungan di komputer pengguna).

WebAssembly, singkatnya

Browser aslinya hanya mengerti **JavaScript**. Tetapi browser modern juga bisa menjalankan **WebAssembly** (disingkat *WASM*): kode mesin kecil yang aman dan cepat. Para pengembang mengompilasi interpreter Python (CPython) menjadi WebAssembly, lalu browser menjalankannya seperti program biasa di dalam "kotak pasir" yang aman. Hasilnya:

Nama	Apa itu
Pyodide	CPython yang dikompilasi ke WebAssembly, lengkap dengan banyak pustaka sains (NumPy dan sejenisnya)
PyScript	Kerangka kerja di atas Pyodide (atau MicroPython): Python ditulis langsung di halaman HTML
JupyterLite	JupyterLab/notebook yang seluruhnya berjalan di browser, memakai kernel Pyodide

Pyodide: Python di dalam WebAssembly

Pyodide bisa dijalankan di browser dan juga di Node.js. Kita memakai Node.js supaya bisa **menangkap keluaran nyata** untuk buku ini. Skrip Python di bawah **dijalankan oleh Pyodide**, bukan oleh Python di komputer Anda:

```
# Bab 37 - Python yang berjalan di dalam Pyodide
import platform
import sys

print("sistem:", sys.platform)                    # 'emscripten' = dunia WebAssembly
print("implementasi:",       platform.python_implementation(),       "|       Python",
sys.version_info[0], sys.version_info[1])


def biomassa_kg(jumlah_ikan, bobot_rata_g):
    return jumlah_ikan * bobot_rata_g / 1000


print("biomassa kolam A (kg):", biomassa_kg(800, 540.5))

# Memanggil JavaScript dari Python
from js import Math

print("akar 1600 lewat JavaScript:", Math.sqrt(1600))

# NumPy tersedia sebagai paket Pyodide (dimuat dengan loadPackage di sisi JavaScript)
import numpy as np

bobot = np.array([480, 520, 505, 495, 540.5])
print("rata-rata bobot:", bobot.mean(), "| siap panen (>=500 g):", int((bobot >=
500).sum()))

# Batasan: tidak ada proses lain dan tidak ada soket jaringan biasa
import subprocess

try:
    subprocess.run(["ls"])
except Exception as galat:
    print("subprocess ->", type(galat).__name__)
```

```
sistem: emscripten
implementasi: CPython | Python 3 14
biomassa kolam A (kg): 432.4
akar 1600 lewat JavaScript: 40
rata-rata bobot: 508.1 | siap panen (>=500 g): 3
subprocess -> OSError
```

Keluaran itu diambil asli sekali lewat pembungkus kecil `kode/bab37/node/jalankan.mjs` (pustaka `pyodide` versi 314.0.7 dipasang dengan `npm install`, dan NumPy diunduh dari CDN resmi Pyodide). Karena itu skrip diberi tanda `@statis`: bangun buku tidak menjalankannya ulang.

Telusuri keluarannya:

- `sistem: emscripten` menandakan Python berjalan di dunia WebAssembly, bukan Windows atau Linux.
- Python-nya adalah **CPython 3.14**, bahasa yang sama dengan buku ini. Rumus `biomassa_kg` dan NumPy bekerja seperti biasa (508,1 adalah rata-rata dari lima bobot).
- `from js import Math` memanggil **JavaScript** dari Python: dua dunia bisa saling bicara.
- `subprocess` gagal dengan `OSError`: browser tidak mengizinkan program memulai proses lain. Ini salah satu **batasan** kotak pasir.

> **i** INGAT
>
> Pada Pyodide, nomor versi mengikuti versi Python: seri **314** berarti Python 3.14. Kita mematok versi di alamat (`.../pyodide/v314.0.7/full/pyodide.js`) supaya halaman tidak tiba-tiba berubah perilakunya saat Pyodide diperbarui.

Pyodide langsung di halaman HTML

Halaman `kode/bab37/demo/peta_pyodide.html` memuat Pyodide dari CDN (jaringan pengantar berkas), menjalankan Python yang menggambar **peta petak sederhana** sebagai SVG (gambar vektor untuk web), lalu menempelkannya ke halaman. Python menghitung luas dengan rumus tali sepatu; tidak ada server, tidak ada GeoPandas. Inilah kerangkanya:

```
<script src="https://cdn.jsdelivr.net/pyodide/v314.0.7/full/pyodide.js"></script>
<script>
  const pyodide = await loadPyodide();            // unduh dan siapkan Python
  const hasil = pyodide.runPython("1 + 2");       // jalankan kode Python
  await pyodide.loadPackage("numpy");             // paket Pyodide dimuat bila perlu
</script>
```

Demo itu **sudah saya coba di browser sungguhan** (peta dan tabel luas tampil, dengan H1 9,00 ha, K1 4,00 ha, S1 3,00 ha, S2 2,25 ha). Untuk mencobanya sendiri, buka `demo/peta_pyodide.html` lewat server lokal sederhana (`python -m http.server` di folder demo), karena sebagian browser membatasi halaman yang dibuka langsung dari berkas.

PyScript: Python di dalam HTML

Dengan **PyScript**, Python ditulis langsung di dalam halaman memakai `<script type="py">`. Kalkulator biomassa kolam di `demo/kalkulator_pyscript.html` hanya butuh dua baris pemuatan:

```
<link rel="stylesheet" href="https://pyscript.net/releases/2026.7.3/core.css">
<script type="module" src="https://pyscript.net/releases/2026.7.3/core.js"></script>
```

lalu Python-nya:

```
from pyscript import document, when

@when("click", "#hitung")
def hitung(event):
    jumlah = float(document.querySelector("#jumlah").value)
    bobot = float(document.querySelector("#bobot").value)
   document.querySelector("#hasil").innerText = f"Biomassa: {jumlah * bobot / 1000:.1f}
kg"
```

`@when("click", "#hitung")` berarti: "bila tombol ber-id `hitung` diklik, jalankan fungsi ini". Saya mencoba halaman itu di browser sungguhan: klik **Hitung** dengan isian bawaan (800 ikan, 540,5 g) menampilkan *Biomassa: 432.4 kg*, sama dengan hitungan Bab 6 dan 10. Tipe `type="mpy"` memakai **MicroPython** (Python mini, unduhan lebih kecil dan cepat mulai) sebagai pengganti Pyodide; pada bab ini kita hanya mencoba `type="py"` (Pyodide).

Skrip 02 memeriksa kedua halaman demo: semua alamat CDN-nya memuat **nomor versi**.

```
# Bab 37 - Memeriksa halaman demo: versi CDN dipatok, tanpa server
import re
from pathlib import Path

demo = Path(__file__).parent / "demo"
for halaman in sorted(demo.glob("*.html")):
    html = halaman.read_text(encoding="utf-8")
    alamat = sorted(set(re.findall(r'(?:src|href)="(https://[^"]+)"', html)))
    print(halaman.name)
    for a in alamat:
        print("  memuat:", a)
    print("  versi dipatok (ada angka versi di alamat):", all(re.search(r"\d+\.\d+\.
\d+", a) for a in alamat))
```

```
kalkulator_pyscript.html
  memuat: https://pyscript.net/releases/2026.7.3/core.css
  memuat: https://pyscript.net/releases/2026.7.3/core.js
  versi dipatok (ada angka versi di alamat): True
peta_pyodide.html
  memuat: https://cdn.jsdelivr.net/pyodide/v314.0.7/full/pyodide.js
  versi dipatok (ada angka versi di alamat): True
```

JupyterLite

JupyterLite adalah distribusi Jupyter yang seluruhnya berjalan di browser dengan kernel Pyodide. Pembuatannya memakai dua paket: `jupyterlite-core` (perintah `jupyter lite build`) dan `jupyterlite-pyodide-kernel` (kernel Python). Hasil build adalah **situs statis** yang bisa ditaruh di hosting berkas biasa, jadi cocok untuk **pengajaran**: murid cukup membuka tautan, tanpa memasang apa pun. Sesuai dokumentasi resmi, server yang menyajikannya harus mengirim berkas `.wasm` dengan jenis `application/wasm`.

BAGIAN TEKNIS (boleh dilewati)

Bagian ini **tidak saya jalankan**: `jupyter lite build` mengunduh banyak berkas kernel dan saya tidak membangunnya di komputer penulis. Versi paket yang tercatat di PyPI saat bab ini ditulis: `jupyterlite-core` dan `jupyterlite-pyodide-kernel` sama-sama 0.8.6. Cara pemakaian di atas diambil dari dokumentasi JupyterLite (halaman *standalone quickstart*).

Kapan cocok, kapan tidak

Cocok	Kurang cocok
Kalkulator dan alat bantu kecil	Data besar (ratusan MB)
Demo dan bahan ajar tanpa pemasangan	Hitungan berat yang butuh GPU
Alat yang harus jalan **tanpa sinyal** setelah diunduh	Program yang butuh proses lain, soket jaringan, atau berkas sistem
Prototipe yang tidak butuh server	Rahasia: **semua kode dan data ikut diunduh pengguna**

Dua catatan penting. Pertama, **waktu mulai**: berkas inti Pyodide berukuran belasan megabita; pada demo saya, halaman menunggu beberapa detik sebelum Python siap. Kedua, **tidak ada rahasia**: apa pun yang Anda taruh di halaman (kunci API, kata sandi, data pribadi) bisa dibaca siapa saja yang membukanya.

⚠ AWAS

Tidak semua pustaka tersedia di Pyodide. Pustaka murni Python biasanya bisa dipasang, tetapi pustaka berkode C harus sudah dikompilasi untuk WebAssembly. Periksa daftar paket Pyodide sebelum merencanakan aplikasi.

Proyek Bersambung I5: ringkasan luas petak yang tampil di browser

Pola yang masuk akal: **data disiapkan di komputer penulis** dengan GeoPandas (berat), lalu **hasilnya ditampilkan di browser** dengan PyScript (ringan). Skrip berikut menghitung luas petak, lalu menulis `main.py` (dengan data tertanam) dan `index.html` ke `_tmp/web/`:

```
# Bab 37 - Proyek bersambung I5: GeoPandas menyiapkan data, browser menampilkannya
dengan PyScript
import json
from pathlib import Path

import geopandas as gpd

web = Path(__file__).parent / "_tmp" / "web"
web.mkdir(parents=True, exist_ok=True)

# Langkah 1 (di komputer penulis, Python biasa): hitung luas petak dengan GeoPandas
petak = gpd.read_file("data/petak.geojson").to_crs(32748)
petak["luas_ha"] = (petak.area / 10_000).round(2)
data = petak[["kode", "sektor", "luas_ha"]].to_dict("records")

# Langkah 2: tulis program yang akan berjalan di browser (main.py) dan halamannya
(web / "main.py").write_text(f'''from pyscript import document

DATA = {json.dumps(data)}

total = {{}}
for baris in DATA:
    total[baris["sektor"]] = total.get(baris["sektor"], 0) + baris["luas_ha"]

teks = "\\n".join(f"{{s:7s}} {{ha:6.2f}} ha" for s, ha in sorted(total.items()))
document.querySelector("#hasil").innerText = teks
''', encoding="utf-8")

(web / "index.html").write_text("""<!doctype html>
<html lang="id"><head><meta charset="utf-8"><title>Luas per sektor</title>
<link rel="stylesheet" href="https://pyscript.net/releases/2026.7.3/core.css">
<script     type="module"     src="https://pyscript.net/releases/2026.7.3/core.js"></
script></head>
<body><h1>Luas per sektor</h1><pre id="hasil">menghitung...</pre>
<script type="py" src="./main.py"></script></body></html>
""", encoding="utf-8")

for berkas in sorted(web.iterdir()):
    print(berkas.name, "-", berkas.stat().st_size, "byte")
print("\nisi main.py (3 baris pertama):")
print("\n".join((web    /    "main.py").read_text(encoding="utf-8").splitlines()[:3])
[:140])
```

```
index.html - 392 byte
main.py - 593 byte
```

```
isi main.py (3 baris pertama):
from pyscript import document

DATA = [{"kode": "H1", "sektor": "hutan", "luas_ha": 9.0}, {"kode": "H2", "sektor":
"hutan", "luas_ha": 6.93}
```

Saya membuka halaman hasil itu lewat server lokal dan browser: tampil `hutan 15.93 ha`, `kebun 7.00 ha`, `sawah 5.25 ha`, sama dengan hitungan GeoPandas di Bab 38. Perhatikan bahwa `main.py` hanya memakai Python biasa: ia tidak butuh GeoPandas.

Latihan

1. Buka `demo/kalkulator_pyscript.html` lewat server lokal, ubah jumlah ikan menjadi 1250, dan cocokkan hasilnya dengan hitungan tangan.
2. Ubah `kalkulator_pyscript.html` agar menampilkan juga "perkiraan panen (kg)" bila 85 persen ikan hidup (angka contoh).
3. Tambah satu petak lagi ke dict `petak` di `peta_pyodide.html` dan lihat petanya berubah.
4. **Tantangan:** buat halaman PyScript yang menghitung rata-rata bobot dari beberapa angka yang diketik pengguna dipisah koma.

Ringkasan Bab

- **WebAssembly** membuat Python bisa berjalan di dalam browser, dalam kotak pasir yang aman.
- **Pyodide** = CPython di WebAssembly; **PyScript** = Python di dalam HTML; **JupyterLite** = notebook di browser.
- Cocok untuk demo, pengajaran, dan alat kecil tanpa server; tidak cocok untuk data besar, rahasia, atau program yang butuh proses lain.
- Patok nomor versi pada alamat CDN. Kode dan data **ikut terunduh** oleh pengguna.

Selanjutnya, di Bab 38: data peta pertama Anda: titik, garis, dan poligon dengan GeoPandas.

Bab 38 — GIS 1: Data Vektor: Peta yang Bisa Dihitung

Seri I5 — Aplikasi Pertama: Web, Peta, dan Dasbor

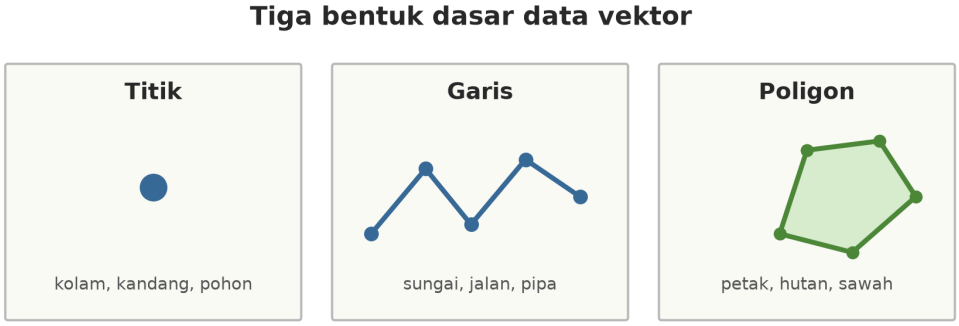
Di bab ini, Anda akan:

- Memahami data spasial: **titik, garis, dan poligon**
- Memakai **Shapely** (bentuk) dan **GeoPandas** (tabel berbentuk)
- Membedakan format **GeoJSON, GeoPackage, dan Shapefile**
- Memahami **CRS** (sistem koordinat) dan **proyeksi**, serta menghindari jebakan derajat
- Menghitung **luas petak**, **jarak kolam ke sungai**, **buffer**, **overlay**, dan **spatial join**

Semua skrip bab ini ada di folder `kode/bab38/`. Di versi web, Anda juga bisa mengunduh semuanya sekaligus (zip).

Sampai sekarang, "lokasi" di buku ini hanya nama: kolam A, petak H1. Padahal pertanyaan lapangan sering menyangkut **letak**: *Berapa hektare sawah kita? Kolam mana yang paling dekat ke sungai? Petak mana yang masuk zona penyangga?* **GIS** (*Geographic Information System*, sistem informasi geografis) menjawabnya dengan menyimpan bentuk dan letak benda di bumi sebagai data, lalu menghitungnya. Bab ini membahas data **vektor**, yaitu benda yang digambar dengan koordinat. (Data **raster**, yaitu kisi piksel seperti foto satelit, ada di Bab 39.)

Seluruh data bab ini **tiruan** buatan skrip dan letaknya **fiktif**, jadi angkanya hanya untuk belajar. Isinya enam petak (hutan, kebun, sawah), empat kolam, tiga kandang, satu sungai, dan dua dusun, tersimpan di `kode/bab38/data/` sebagai berkas GeoJSON.



Gambar 0.1: Tiga bentuk dasar data vektor.

Titik, garis, poligon: Shapely

Shapely adalah pustaka untuk membuat dan menghitung bentuk geometri. Ia tidak tahu apa-apa tentang bumi: ia hanya melihat angka pada bidang datar.

```
# Bab 38 - Titik, garis, dan poligon dengan Shapely
from shapely.geometry import LineString, Point, Polygon

# Koordinat di bawah ini dalam meter (bidang datar, satuan sama di kedua sumbu)
kolam = Point(120, 285)                                  # titik: satu lokasi
sungai = LineString([(0, 330), (200, 310), (400, 290)])   # garis: urutan titik
petak = Polygon([(50, 100), (250, 100), (250, 250), (50, 250)]) # poligon: area tertutup
```

```
print(kolam.geom_type, sungai.geom_type, petak.geom_type)
print("panjang sungai (m):", round(sungai.length, 1))
print("luas petak (m2)  :", petak.area)
print("keliling petak (m):", petak.length)
print("titik tengah petak:", petak.centroid)

print("kolam di dalam petak?  ", petak.contains(kolam))
print("sungai memotong petak? ", sungai.intersects(petak))
print("jarak kolam ke sungai (m):", round(kolam.distance(sungai), 1))
print("batas luar (minx, miny, maxx, maxy):", petak.bounds)
```

```
Point LineString Polygon
panjang sungai (m): 402.0
luas petak (m2)  : 30000.0
keliling petak (m): 700.0
titik tengah petak: POINT (150 175)
kolam di dalam petak?   False
sungai memotong petak?  False
jarak kolam ke sungai (m): 32.8
batas luar (minx, miny, maxx, maxy): (50.0, 100.0, 250.0, 250.0)
```

- `Point` untuk **satu lokasi** (kolam, kandang, pohon), `LineString` untuk **urutan titik** (sungai, jalan), `Polygon` untuk **area tertutup** (petak).
- Luas petak 30000 m² (200 m kali 150 m) dan kelilingnya 700 m. Sungai (dari tiga titik) panjangnya 402 m.
- `contains`, `intersects`, `distance` menjawab pertanyaan letak: kolam tidak di dalam petak, sungai tidak memotong petak, dan jarak kolam ke sungai 32,8 m.

> **i** INGAT
>
> Shapely menghitung dalam **satuan angka koordinat**. Bila koordinat dalam meter, luasnya meter persegi. Bila dalam derajat, "luas"-nya derajat persegi, yang tidak berguna. Itu persoalan CRS, dibahas sebentar lagi.

GeoPandas: tabel yang punya bentuk

GeoPandas menambahkan satu kolom khusus, `geometry`, ke DataFrame pandas (Bab 23). Semua yang Anda tahu tentang pandas (menyaring, `groupby`, menggabung) tetap berlaku. Mesin yang membaca dan menulis berkas peta di belakang layar adalah **pyogrio** (pengganti fiona, yang belum tersedia untuk Python 3.14 saat bab ini ditulis).

```
# Bab 38 - GeoDataFrame dan tiga format berkas: GeoJSON, GeoPackage, Shapefile
import warnings
from pathlib import Path

import geopandas as gpd
from shapely.geometry import Point

tmp = Path(__file__).parent / "_tmp"
tmp.mkdir(exist_ok=True)

# GeoDataFrame = tabel pandas + satu kolom bernama geometry + CRS
kandang = gpd.GeoDataFrame(
    {"kode": ["D1", "D2"], "jenis": ["sapi", "ayam"], "populasi_ekor_hari_ini": [12,
200]},
   geometry=[Point(106.8168, -6.9955), Point(106.8118, -6.9945)],   # (bujur, lintang)
    crs="EPSG:4326",
)
```

```
print(kandang)
print("CRS:", kandang.crs)

kandang.to_file(tmp / "kandang.geojson", driver="GeoJSON")
kandang.to_file(tmp / "kandang.gpkg", layer="kandang", driver="GPKG")
with warnings.catch_warnings(record=True) as w:
    warnings.simplefilter("always")
    kandang.to_file(tmp / "kandang.shp")                          # Shapefile
    print("peringatan Shapefile:", str(w[0].message))

print("berkas Shapefile:", sorted(p.name for p in tmp.glob("kandang.*") if p.suffix
not in (".geojson", ".gpkg")))
print("kolom dari Shapefile  :", list(gpd.read_file(tmp / "kandang.shp").columns))
print("kolom dari GeoPackage :", list(gpd.read_file(tmp / "kandang.gpkg").columns))
```

```
  kode jenis  populasi_ekor_hari_ini                 geometry
0   D1  sapi                      12  POINT (106.8168 -6.9955)
1   D2  ayam                     200  POINT (106.8118 -6.9945)
CRS: EPSG:4326
peringatan Shapefile: Column names longer than 10 characters will be truncated when
saved to ESRI Shapefile.
berkas  Shapefile:  ['kandang.cpg',  'kandang.dbf',  'kandang.prj',  'kandang.shp',
'kandang.shx']
kolom dari Shapefile  : ['kode', 'jenis', 'populasi_e', 'geometry']
kolom dari GeoPackage : ['kode', 'jenis', 'populasi_ekor_hari_ini', 'geometry']
```

Tiga format yang paling sering Anda temui:

Format	Berkas	Kelebihan	Kekurangan
GeoJSON	satu `.geojson` (teks)	Mudah dibaca manusia dan web; selalu EPSG:4326	Boros ruang; lambat untuk data besar
GeoPackage	satu `.gpkg` (SQLite)	Satu berkas, **banyak lapisan**, nama kolom bebas, standar terbuka	Tidak terbaca manusia
Shapefile	**lima berkas atau lebih** (`.shp`, `.shx`, `.dbf`, `.prj`, `.cpg`)	Dibaca hampir semua perangkat lunak lama	Nama kolom maksimum 10 huruf

Keluarannya menunjukkan batasan Shapefile: kolom `populasi_ekor_hari_ini` **dipotong** menjadi `populasi_e` (dan Anda diberi peringatan), sedangkan GeoPackage menyimpannya utuh. Untuk data baru, pilih **GeoPackage**.

CRS: derajat atau meter?

Bumi bulat, peta datar. **CRS** (*Coordinate Reference System*) menjelaskan arti angka koordinat. Dua yang paling penting:

- **EPSG:4326** (WGS 84): koordinat **bujur dan lintang dalam derajat**. Dipakai GPS dan web.
- **Proyeksi UTM**, misalnya **EPSG:32748** (UTM zona 48 Selatan): koordinat **dalam meter** di bidang datar. Cocok untuk luas dan jarak di satu wilayah. Indonesia terbagi dalam beberapa zona UTM.

```
# Bab 38 - Sistem koordinat (CRS): derajat atau meter?
import geopandas as gpd
from pyproj import CRS, Transformer

petak = gpd.read_file("data/petak.geojson")
print("CRS berkas:", petak.crs)
print("satuan sumbu:", [s.unit_name for s in petak.crs.axis_info])
```

```
print("pojok petak H1 (bujur, lintang):", [round(v, 5) for v in
petak.geometry.iloc[0].exterior.coords[0]])
print("'luas' H1 dalam derajat persegi:", petak.geometry.iloc[0].area)   # tidak ada
artinya!

utm = petak.estimate_utm_crs()                    # menebak zona UTM dari letak data
print("zona UTM yang cocok:", utm.to_string(), "|", utm.name)
datar = petak.to_crs(utm)
print("satuan sumbu UTM:", [s.unit_name for s in datar.crs.axis_info])
print("luas H1 (m2):", round(datar.geometry.iloc[0].area))

# Pyproj: pindah satu titik dari 4326 ke UTM. always_xy=True = urutan (bujur, lintang).
pindah = Transformer.from_crs(CRS.from_epsg(4326), CRS.from_epsg(32748),
always_xy=True)
timur, utara = pindah.transform(106.8125, -6.9950)
print("titik di UTM 48S: timur =", round(timur, 1), "| utara =", round(utara, 1))
```

```
CRS berkas: EPSG:4326
satuan sumbu: ['degree', 'degree']
pojok petak H1 (bujur, lintang): [106.81361, -6.99516]
'luas' H1 dalam derajat persegi: 7.363732839988365e-06
zona UTM yang cocok: EPSG:32748 | WGS 84 / UTM zone 48S
satuan sumbu UTM: ['metre', 'metre']
luas H1 (m2): 90001
titik di UTM 48S: timur = 700227.2 | utara = 9226418.1
```

- File kita berasal dari EPSG:4326, sumbunya **derajat**. "Luas" H1 di situ `7.36e-06` derajat persegi: angka tanpa arti praktis.
- `estimate_utm_crs()` menebak zona UTM yang cocok dari letak data: EPSG:32748.
- Setelah `to_crs(...)`, sumbu berubah menjadi **meter** dan luas H1 menjadi 90001 m² (kira-kira 9 hektare; selisih 1 m² berasal dari pembulatan koordinat GeoJSON ke 7 desimal, sekitar 1 cm).
- `pyproj` bisa memindahkan satu titik sendiri. `always_xy=True` memastikan urutannya (bujur, lintang), karena EPSG:4326 resminya berurutan (lintang, bujur) dan sering membingungkan.

> ⚠ AWAS
>
> **Jangan** menghitung luas atau jarak langsung dari data berderajat. Ubah dulu ke CRS yang berunit meter dengan `to_crs`. Ini kesalahan paling sering di GIS. Setiap berkas membawa CRS-nya sendiri (`gdf.crs`): periksa sebelum menggabung.

Luas petak dan jarak kolam ke sungai

```
# Bab 38 - Luas petak dan jarak kolam ke sungai
import geopandas as gpd

petak = gpd.read_file("data/petak.geojson").to_crs(32748)    # meter
kolam = gpd.read_file("data/kolam.geojson").to_crs(32748)
sungai = gpd.read_file("data/sungai.geojson").to_crs(32748)

petak["luas_ha"] = (petak.area / 10_000).round(2)
print(petak[["kode", "sektor", "komoditas", "luas_ha"]].to_string(index=False))
print()
print("luas per sektor (ha):")
print(petak.groupby("sektor")["luas_ha"].sum().to_string())

garis_sungai = sungai.geometry.iloc[0]
kolam["jarak_sungai_m"] = kolam.distance(garis_sungai).round(1)
```

```
print()
print(kolam[["kode",                                                    "jenis",
"jumlah_ikan", "jarak_sungai_m"]].to_string(index=False))
dekat = kolam[kolam["jarak_sungai_m"] < 25]
print("kolam kurang dari 25 m dari sungai:", list(dekat["kode"]))
```

```
kode sektor komoditas  luas_ha
  H1  hutan      Jati     9.00
  H2  hutan   Meranti     6.93
  K1  kebun      kopi     4.00
  K2  kebun     kakao     3.00
  S1  sawah      padi     3.00
  S2  sawah      padi     2.25

luas per sektor (ha):
sektor
hutan    15.93
kebun     7.00
sawah     5.25

kode jenis  jumlah_ikan  jarak_sungai_m
   A  lele          800            15.4
   B  nila          500            29.8
   C  lele          650            20.0
   D  nila          300           116.4
kolam kurang dari 25 m dari sungai: ['A', 'C']
```

Hutan 15,93 ha (9,00 + 6,93), kebun 7,00 ha, sawah 5,25 ha. `kolam.distance(garis)` menghitung jarak terdekat dari tiap kolam ke sungai: A dan C kurang dari 25 m, kolam D jauh (116,4 m). Pada contoh nyata, kolam dekat sungai berguna untuk air, tetapi berisiko banjir; angka batas 25 m di sini hanya contoh.

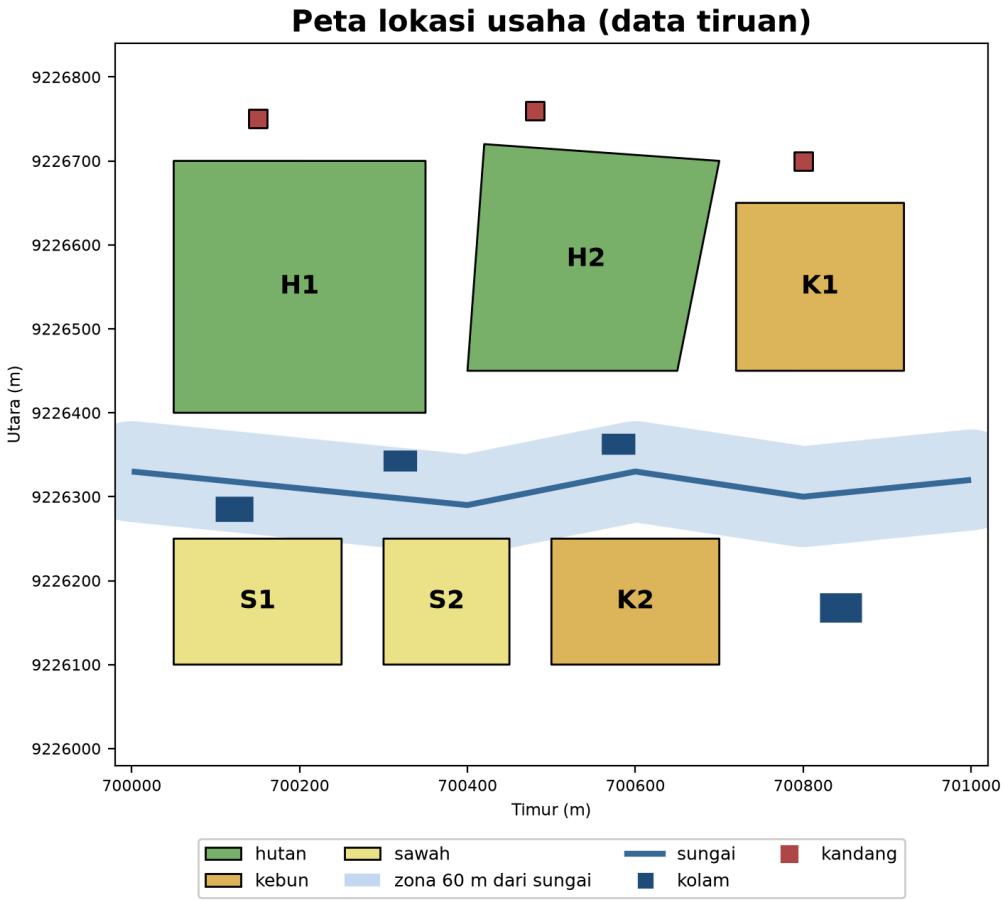
Buffer dan overlay

Buffer membuat area di sekitar suatu bentuk dengan jarak tertentu. **Overlay** menumpuk dua lapisan dan menghitung potongannya: `intersection` (bagian yang beririsan) atau `difference` (bagian yang tidak).

```
# Bab 38 - Buffer (zona penyangga) dan overlay (tumpang susun)
import geopandas as gpd

petak = gpd.read_file("data/petak.geojson").to_crs(32748)
sungai = gpd.read_file("data/sungai.geojson").to_crs(32748)

# Buffer: area dalam jarak 60 m dari sungai (angka 60 hanya contoh belajar)
zona   =   gpd.GeoDataFrame({"zona":   ["penyangga"]},   geometry=sungai.buffer(60),
crs=sungai.crs)
print("luas zona penyangga (ha):", round(zona.area.iloc[0] / 10_000, 2))

# Overlay intersection: potongan petak yang masuk zona
masuk = gpd.overlay(petak, zona, how="intersection")
masuk["luas_masuk_m2"] = masuk.area.round(0)
print("\npetak yang masuk zona penyangga:")
print(masuk[["kode", "sektor", "luas_masuk_m2"]].to_string(index=False))

# Overlay difference: bagian petak yang di luar zona
aman = gpd.overlay(petak, zona, how="difference")
aman["luas_aman_ha"] = (aman.area / 10_000).round(2)
print("\nluas petak di luar zona (ha):")
print(aman[["kode", "luas_aman_ha"]].to_string(index=False))
```

```
luas zona penyangga (ha): 13.24

petak yang masuk zona penyangga:
kode sektor  luas_masuk_m2
  K2  kebun            4.0
  S1  sawah          141.0
  S2  sawah         2333.0

luas petak di luar zona (ha):
kode  luas_aman_ha
  H1          9.00
  H2          6.93
  K1          4.00
  K2          3.00
  S1          2.99
  S2          2.02
```

Zona 60 m di kiri-kanan sungai luasnya 13,24 ha. Tiga petak menjorok masuk: S2 paling banyak (2333 m²), S1 hanya 141 m², dan K2 cuma 4 m². Baris terakhir menunjukkan sisanya: S2 tinggal 2,02 ha (dari 2,25) dan S1 2,99 ha. Angka 60 m hanyalah contoh; aturan sempadan sungai yang sebenarnya ditentukan peraturan setempat.



Gambar 0.2: Peta lokasi usaha (data tiruan): petak, kolam, kandang, sungai, dan zona 60 m.

Spatial join: menggabung menurut letak

Di pandas, `merge` menggabung dua tabel menurut **nilai kolom yang sama**. Di GeoPandas, **spatial join** (`sjoin`) menggabung menurut **hubungan letak**: “titik ini di dalam poligon yang mana?”.

```python
# Bab 38 - Spatial join: menggabungkan tabel berdasarkan letak
import warnings

import geopandas as gpd

dusun = gpd.read_file("data/dusun.geojson").to_crs(32748)
kolam = gpd.read_file("data/kolam.geojson").to_crs(32748)
kandang = gpd.read_file("data/kandang.geojson").to_crs(32748)

# Titik pusat tiap kolam jatuh di dusun mana? (predicate="within": di dalam)
kolam_titik = kolam.copy()
kolam_titik["geometry"] = kolam.centroid
gabung = gpd.sjoin(kolam_titik, dusun[["nama", "geometry"]], predicate="within")
print(gabung[["kode", "jumlah_ikan", "nama"]].to_string(index=False))
print("\nikan per dusun:")
print(gabung.groupby("nama")["jumlah_ikan"].sum().to_string())

# Kandang terdekat dari tiap kolam (sjoin_nearest) beserta jaraknya
dekat = gpd.sjoin_nearest(kolam_titik, kandang[["kode", "geometry"]],
distance_col="jarak_m",
                          lsuffix="kolam", rsuffix="kandang")
print("\nkandang terdekat dari tiap kolam:")
print(dekat[["kode_kolam",
"kode_kandang", "jarak_m"]].round(0).to_string(index=False))

# Jebakan: CRS berbeda -> hanya peringatan, hasilnya KOSONG
with warnings.catch_warnings(record=True) as w:
    warnings.simplefilter("always")
    salah = gpd.sjoin(gpd.read_file("data/kolam.geojson"), dusun)     # kiri 4326,
kanan UTM
print("\nCRS beda ->", str(w[0].message).splitlines()[0])
print("jumlah baris hasil:", len(salah))
```

```
kode  jumlah_ikan  nama
   A          800  Hulu
   B          500  Hulu
   C          650 Hilir
   D          300 Hilir

ikan per dusun:
nama
Hilir     950
Hulu     1300

kandang terdekat dari tiap kolam:
kode_kolam kode_kandang  jarak_m
         A           D2    466.0
         B           D2    442.0
         C           D1    403.0
         D           D1    534.0

CRS beda -> CRS mismatch between the CRS of left geometries and the CRS of right
geometries.
jumlah baris hasil: 0
```

- Kolam A dan B jatuh di dusun Hulu, C dan D di Hilir, sehingga ikan per dusun: Hulu 1300, Hilir 950.
- `sjoin_nearest` mencari **yang terdekat**; kandang D2 terdekat dari kolam A dan B, sedangkan D1 dari C dan D.
- **Jebakan di akhir:** menggabung data berCRS beda tidak menimbulkan error, hanya **peringatan**, dan hasilnya **kosong** (0 baris). Selalu `to_crs` dulu, dan baca peringatan.

Memasang GIS: pip atau conda-forge?

Di komputer penulis, semua pustaka GIS terpasang dengan **pip** karena tersedia sebagai *wheel* (paket jadi) yang sudah membawa pustaka C di dalamnya. Tidak selalu begitu. Pustaka dasarnya, **GDAL**, **GEOS**, dan **PROJ**, ditulis dalam C/C++, dan di beberapa sistem pip tidak menemukan wheel yang cocok sehingga pemasangan gagal. **conda-forge** (saluran paket komunitas untuk `conda` atau `mamba`) memasang pustaka C itu **bersama** paket Python-nya sehingga biasanya lebih mulus:

	pip	conda-forge
Perintah	`pip install geopandas rasterio`	`conda install -c conda-forge geopandas rasterio`
Pustaka C (GDAL, PROJ)	Ikut di wheel, bila ada untuk sistem Anda	Dipasang sebagai paket sendiri, versinya serasi
Kelebihan	Ringan, satu cara dengan paket lain	Paling andal untuk GDAL, banyak alat GIS lain
Kekurangan	Bisa gagal bila tidak ada wheel	Perlu memasang conda/mamba; lingkungan terpisah dari venv

BAGIAN TEKNIS (boleh dilewati)

Perintah `conda` di atas **tidak saya jalankan**: conda belum terpasang di komputer penulis, jadi bagian conda-forge belum diuji di sini. Perbandingan ini berdasar cara kerja umum kedua alat. Bila `pip install geopandas` berhasil di komputer Anda, Anda tidak perlu conda.

Proyek Bersambung I5: peta lokasi usaha

Kita satukan semuanya: membaca lima lapisan, mengubahnya ke UTM, menyimpannya sebagai **satu GeoPackage berisi lima lapisan**, merangkum luas per sektor, dan membuat peta cepat dengan `.plot()`. GeoPackage ini bahan untuk Bab 40 (PostGIS dan QGIS).

```
# Bab 38 - Proyek bersambung I5: peta lokasi usaha, disimpan ke GeoPackage
from pathlib import Path

import geopandas as gpd
import matplotlib.pyplot as plt

tmp = Path(__file__).parent / "_tmp"
tmp.mkdir(exist_ok=True)
lapisan = ["petak", "kolam", "kandang", "sungai", "dusun"]
data = {nama: gpd.read_file(f"data/{nama}.geojson").to_crs(32748) for nama in lapisan}

# Satu berkas GeoPackage berisi banyak lapisan (layer)
gpkg = tmp / "lokasi_usaha.gpkg"
gpkg.unlink(missing_ok=True)
for nama, gdf in data.items():
    gdf.to_file(gpkg, layer=nama, driver="GPKG")
print("lapisan di dalam GeoPackage:", gpd.list_layers(gpkg)["name"].tolist())

# Ringkasan per sektor: luas petak dan jumlah titik
petak = data["petak"]
print("\nluas per sektor (ha):")
luas_ha = petak.assign(luas_ha=petak.area / 10_000)
print(luas_ha.groupby("sektor")["luas_ha"].sum().round(2).to_string())
print("jumlah kolam:", len(data["kolam"]), "| jumlah kandang:", len(data["kandang"]))

# Peta cepat dengan .plot()
```

```
fig, ax = plt.subplots(figsize=(7, 5.6))
petak.plot(ax=ax, column="sektor", legend=True, alpha=0.6, edgecolor="black")
data["sungai"].plot(ax=ax, color="tab:blue", linewidth=2)
data["kolam"].plot(ax=ax, color="navy")
data["kandang"].plot(ax=ax, color="red", markersize=40)
ax.set_title("Peta lokasi usaha (data tiruan)")
fig.savefig(tmp / "peta_cepat.png", dpi=100)
print("peta tersimpan:", (tmp / "peta_cepat.png").name)
```

```
lapisan di dalam GeoPackage: ['petak', 'kolam', 'kandang', 'sungai', 'dusun']

luas per sektor (ha):
sektor
hutan    15.93
kebun     7.00
sawah     5.25
jumlah kolam: 4 | jumlah kandang: 3
peta tersimpan: peta_cepat.png
```

`gpd.list_layers` membuktikan satu berkas memuat lima lapisan. Peta cepatnya tersimpan di `_tmp/peta_cepat.png` (peta yang lebih rapi kita buat di Bab 40).

Latihan

1. Di `01_geometri_shapely.py`, buat poligon segitiga dengan titik (0,0), (100,0), (0,60) dan cetak luasnya. (Cocokkan dengan rumus setengah alas kali tinggi.)
2. Di `04_luas_jarak.py`, tambahkan kolom `jarak_sungai_m` untuk **kandang** dan cetak kandang yang paling jauh dari sungai.
3. Di `05_buffer_overlay.py`, ganti 60 m menjadi 100 m. Petak mana saja yang masuk, dan berapa luas yang masuk di S2?
4. **Tantangan:** hitung, untuk setiap dusun, luas total petak yang berada di dalamnya memakai `overlay(petak, dusun, how="intersection")` dan `groupby`. Petak H2 melintasi batas dusun: periksa bahwa jumlah luasnya tetap 6,93 ha.

Ringkasan Bab

- Data vektor = **titik, garis, poligon**. Shapely menghitung bentuknya; GeoPandas menaruhnya dalam tabel.
- Pilih **GeoPackage** untuk data baru; Shapefile memotong nama kolom dan butuh banyak berkas.
- **CRS** menentukan arti koordinat. Untuk luas dan jarak, ubah ke CRS bermeter (mis. UTM) dengan `to_crs`.
- `area`, `distance`, `buffer`, `overlay`, `sjoin`, dan `sjoin_nearest` menjawab pertanyaan lapangan tentang letak.
- CRS berbeda pada `sjoin` atau `overlay` hanya memberi peringatan dan hasil keliru: waspada.
- pip cukup bila ada wheel; conda-forge lebih andal untuk GDAL (belum diuji di sini).

Selanjutnya, di Bab 39: data raster: foto satelit tiruan, piksel, dan NDVI.

Bab 39 — GIS 2: Data Raster: Foto Udara yang Bisa Dihitung

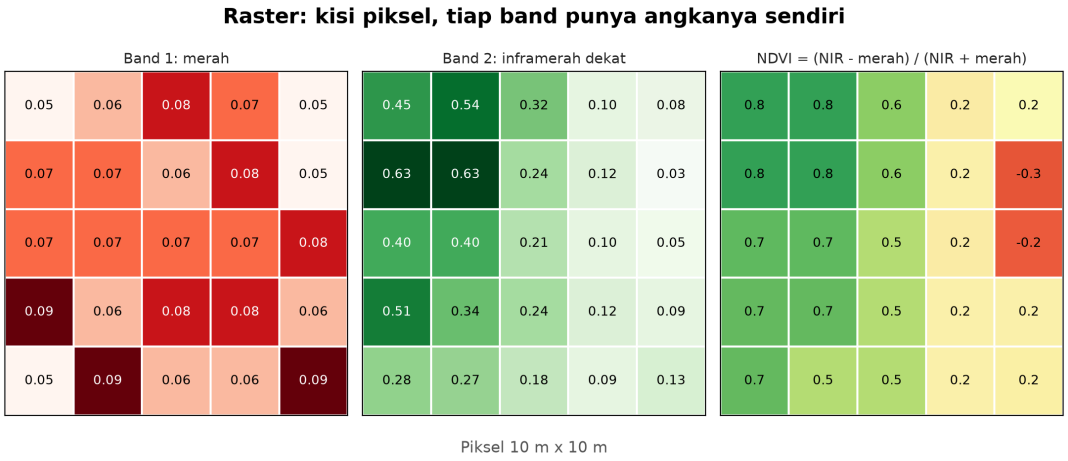
Seri I5 — Aplikasi Pertama: Web, Peta, dan Dasbor

Di bab ini, Anda akan:

- Memahami **raster**: piksel, resolusi, dan **band**
- Membaca citra dengan **Rasterio** dan berpindah antara piksel dan koordinat
- Menghitung **NDVI**, indeks kehijauan tumbuhan
- **Memotong** citra dengan batas lahan, lalu mengubah CRS dan resolusinya
- Menghitung **statistik zonal** per petak dan per kolam, lalu menyimpan **GeoTIFF**

Semua skrip bab ini ada di folder `kode/bab39/`. Di versi web, Anda juga bisa mengunduh semuanya sekaligus (zip).

Bab 38 menggambar lahan dengan garis. Tetapi foto satelit atau foto drone tidak berisi garis: isinya **kisi kotak-kotak kecil**, masing-masing punya satu angka. Itulah **raster**. Bayangkan selembar kain kotak-kotak di atas kebun Anda; tiap kotak dicat sesuai warna rata-rata tanah di bawahnya. Kotak itu disebut **piksel**.



Gambar 0.1: Raster = kisi piksel. Tiap band punya angkanya sendiri; NDVI dihitung dari dua band.

Tiga istilah pokok:

- **Piksel**: satu kotak pada kisi, dengan satu nilai per band.
- **Resolusi**: ukuran sisi satu piksel di lapangan. Resolusi 10 m berarti satu piksel mewakili 10 m kali 10 m (100 m²). Makin kecil angkanya, makin halus citranya, dan makin besar berkasnya.
- **Band**: satu "lapisan" nilai. Foto biasa punya tiga band (merah, hijau, biru). Citra satelit punya band lain yang tak terlihat mata, misalnya **inframerah dekat** (*NIR*).

i INGAT

Tumbuhan sehat **memantulkan banyak** inframerah dekat dan **menyerap banyak** cahaya merah. Air hampir tidak memantulkan inframerah. Perbedaan ini yang dipakai untuk mengukur "kehijauan".

Dilarang mengunduh citra besar untuk buku ini, maka kita membuat **citra satelit tiruan** dengan skrip: kisi 100 kolom kali 80 baris, piksel 10 m, menutupi area 1 km × 800 m yang sama dengan Bab 38, dengan dua band (merah dan NIR) dan sedikit derau acak. Tiap petak sengaja diberi tingkat kehijauan tertentu.

```
# Bab 39 - Membuat citra satelit tiruan (2 band: merah dan inframerah dekat)
from pathlib import Path

import geopandas as gpd
import numpy as np
import rasterio
from rasterio.features import rasterize
from rasterio.transform import from_origin

tmp = Path(__file__).parent / "_tmp"
tmp.mkdir(exist_ok=True)

# Kisi 100 kolom x 80 baris, piksel 10 m, pojok kiri-atas di (700000, 9226800) UTM 48S
transform = from_origin(700000, 9226800, 10, 10)
bentuk = (80, 100)

petak = gpd.read_file("data/petak.geojson").to_crs(32748)
kolam = gpd.read_file("data/kolam.geojson").to_crs(32748)
sungai = gpd.read_file("data/sungai.geojson").to_crs(32748)

# Peta "tutupan lahan": tiap piksel diberi nilai NDVI yang diinginkan
ndvi_target = np.full(bentuk, 0.15)                  # tanah terbuka
nilai_petak = {"H1": 0.80, "H2": 0.78, "K1": 0.60, "K2": 0.55, "S1": 0.45, "S2": 0.70}
for _, baris in petak.iterrows():
     mask = rasterize([(baris.geometry, 1)], out_shape=bentuk, transform=transform,
dtype="uint8")
    ndvi_target[mask == 1] = nilai_petak[baris["kode"]]
air = list(kolam.geometry) + list(sungai.buffer(8).geometry)       # kolam dan badan
sungai
ndvi_target[rasterize(air, out_shape=bentuk, transform=transform, dtype="uint8") == 1]
= -0.20

# Dari NDVI target, susun band merah dan NIR (ditambah sedikit derau acak)
rng = np.random.default_rng(42)
ndvi_target = ndvi_target + rng.normal(0, 0.03, bentuk)
merah = rng.uniform(0.05, 0.09, bentuk)
nir = merah * (1 + ndvi_target) / (1 - ndvi_target)
citra = np.stack([merah, nir]).astype("float32")

profil = dict(driver="GTiff", height=80, width=100, count=2, dtype="float32",
              crs="EPSG:32748", transform=transform)
with rasterio.open(tmp / "citra_tiruan.tif", "w", **profil) as tujuan:
    tujuan.write(citra)
    tujuan.set_band_description(1, "merah")
    tujuan.set_band_description(2, "nir")
print("tersimpan:", (tmp / "citra_tiruan.tif").name, "| bentuk:", citra.shape)

# Bukti: hasilnya sama persis dengan berkas yang disediakan di data/
with rasterio.open("data/citra_tiruan.tif") as ada:
    print("identik dengan data/citra_tiruan.tif:", np.array_equal(ada.read(), citra))
```

```
tersimpan: citra_tiruan.tif | bentuk: (2, 80, 100)
identik dengan data/citra_tiruan.tif: True
```

Skrip ini membuat citra di `_tmp/` dan **membuktikan** bahwa hasilnya identik dengan berkas `data/citra_tiruan.tif` yang disediakan. Skrip-skrip berikutnya membaca berkas yang disediakan itu. Dua alat baru terlihat di sini: `rasterize` mengubah poligon menjadi piksel, dan `from_origin(x, y, lebar, tinggi)` menentukan **letak** kisi di bumi (pojok kiri-atas dan ukuran piksel).

Membaca raster dengan Rasterio

Rasterio membaca dan menulis raster (memakai GDAL di belakangnya). Isi tiap band dibaca sebagai **array NumPy** (Bab 23).

```
# Bab 39 - Membaca raster: piksel, resolusi, band, dan koordinat
import rasterio

with rasterio.open("data/citra_tiruan.tif") as citra:
    print("ukuran (lebar x tinggi):", citra.width, "x", citra.height, "piksel")
    print("jumlah band:", citra.count, "| deskripsi:", citra.descriptions)
    print("tipe data:", citra.dtypes[0])
    print("CRS:", citra.crs)
    print("resolusi (m):", citra.res)
    print("batas (kiri, bawah, kanan, atas):", tuple(citra.bounds))

    merah = citra.read(1)          # band 1 -> array NumPy dua dimensi
    nir = citra.read(2)
    print("\nbentuk array:", merah.shape, "(baris, kolom)")
    print("merah: min %.3f maks %.3f" % (merah.min(), merah.max()))
    print("nir  : min %.3f maks %.3f" % (nir.min(), nir.max()))

    # Piksel (baris 10, kolom 20) -> koordinat pusatnya di peta, dan sebaliknya
    x, y = citra.xy(10, 20)
    print("\npusat piksel (10, 20):", x, y)
    print("koordinat itu kembali ke (baris, kolom):", citra.index(x, y))

    # Nilai di satu titik, mis. di dalam kolam A (timur 700122, utara 9226285)
      print("nilai [merah, nir] di titik kolam A:", [round(float(v), 3) for v in
next(citra.sample([(700122, 9226285)]))])
```

```
ukuran (lebar x tinggi): 100 x 80 piksel
jumlah band: 2 | deskripsi: ('merah', 'nir')
tipe data: float32
CRS: EPSG:32748
resolusi (m): (10.0, 10.0)
batas (kiri, bawah, kanan, atas): (700000.0, 9226000.0, 701000.0, 9226800.0)

bentuk array: (80, 100) (baris, kolom)
merah: min 0.050 maks 0.090
nir  : min 0.030 maks 1.314

pusat piksel (10, 20): 700205.0 9226695.0
koordinat itu kembali ke (baris, kolom): (10, 20)
nilai [merah, nir] di titik kolam A: [0.057, 0.038]
```

- Citra 100 × 80 piksel, dua band bernama `merah` dan `nir`, tipe `float32`, CRS EPSG:32748 (meter), resolusi 10 m.
- `bounds` menunjukkan letaknya: dari easting 700000 sampai 701000 dan northing 9226000 sampai 9226800.
- Array berbentuk `(80, 100)`: **baris dulu, baru kolom**. Itu urutan NumPy, kebalikan dari (x, y) yang biasa kita pikirkan pada peta.
- `xy(baris, kolom)` mengubah nomor piksel menjadi koordinat pusatnya; `index(x, y)` membalikkannya. `sample` membaca nilai di satu titik: di dalam kolam A, merah 0,057 dan NIR 0,038. NIR lebih rendah daripada merah, ciri khas air.

NDVI: mengukur kehijauan

NDVI (*Normalized Difference Vegetation Index*) = (NIR − Merah) ÷ (NIR + Merah). Hasilnya antara −1 dan 1: air bernilai negatif, tanah terbuka dekat 0,1 sampai 0,2, dan tumbuhan lebat mendekati 0,8 ke atas.

```
# Bab 39 - NDVI: indeks kehijauan tumbuhan
from pathlib import Path

import numpy as np
import rasterio

tmp = Path(__file__).parent / "_tmp"
tmp.mkdir(exist_ok=True)

with rasterio.open("data/citra_tiruan.tif") as citra:
    merah, nir = citra.read(1), citra.read(2)
    profil = citra.profile

# NDVI = (NIR - Merah) / (NIR + Merah), nilainya antara -1 dan 1
ndvi = (nir - merah) / (nir + merah)
print("NDVI: min %.2f | rata-rata %.2f | maks %.2f" % (ndvi.min(), ndvi.mean(),
ndvi.max()))

# Kelas sederhana (batas angka hanya contoh belajar)
kelas = np.digitize(ndvi, [0.0, 0.3, 0.6])        # 0=air, 1=terbuka, 2=sedang, 3=rapat
nama = ["air", "tanah terbuka", "vegetasi sedang", "vegetasi rapat"]
for i, n in enumerate(nama):
    jumlah = int((kelas == i).sum())
    print(f"{n:16s}: {jumlah:5d} piksel = {jumlah * 100 / 10_000:5.1f} ha")

# Simpan NDVI sebagai GeoTIFF satu band
profil.update(count=1, dtype="float32", nodata=-9999.0)
with rasterio.open(tmp / "ndvi.tif", "w", **profil) as tujuan:
    tujuan.write(ndvi.astype("float32"), 1)
print("tersimpan:", "ndvi.tif")
```

```
NDVI: min -0.29 | rata-rata 0.33 | maks 0.89
air             :   216 piksel =   2.2 ha
tanah terbuka   :  4964 piksel =  49.6 ha
vegetasi sedang :   777 piksel =   7.8 ha
vegetasi rapat  :  2043 piksel =  20.4 ha
tersimpan: ndvi.tif
```

Rata-rata NDVI seluruh citra 0,33 dan terkecil −0,29 (air). Kelas dibuat dengan `np.digitize`; batas 0, 0,3, dan 0,6 hanya contoh belajar (nilai sebenarnya bergantung pada sensor dan tanaman). Misalnya vegetasi rapat 2043 piksel × 100 m² = 20,4 ha. Operasi `(nir - merah) / (nir + merah)` bekerja pada **seluruh kisi sekaligus**, tanpa perulangan: kekuatan NumPy. Hasilnya disimpan sebagai GeoTIFF (`nodata` ditandai −9999).

Gambar 0.2: NDVI citra tiruan dengan batas petak dan kolam. Hijau tua = rapat, oranye = air.

Memotong citra dengan batas lahan

Biasanya kita hanya butuh citra di dalam lahan tertentu. `rasterio.mask.mask` memotongnya dengan poligon.

```
# Bab 39 - Memotong citra dengan batas lahan (clip)
from pathlib import Path

import geopandas as gpd
import numpy as np
import rasterio
from rasterio.mask import mask

tmp = Path(__file__).parent / "_tmp"
tmp.mkdir(exist_ok=True)

petak = gpd.read_file("data/petak.geojson")
with rasterio.open("data/citra_tiruan.tif") as citra:
    s1 = petak[petak["kode"] == "S1"].to_crs(citra.crs)       # CRS vektor = CRS raster
    potongan, transform = mask(citra, s1.geometry, crop=True, nodata=np.nan)
    profil = citra.profile

merah, nir = potongan
ndvi = (nir - merah) / (nir + merah)
print("bentuk potongan (band, baris, kolom):", potongan.shape)
print("piksel di dalam S1:", int(np.isfinite(ndvi).sum()))
print("NDVI rata-rata S1: %.2f" % np.nanmean(ndvi))

profil.update(height=potongan.shape[1], width=potongan.shape[2], transform=transform,
nodata=np.nan)
with rasterio.open(tmp / "potongan_S1.tif", "w", **profil) as tujuan:
    tujuan.write(potongan)
print("tersimpan:", "potongan_S1.tif")
```

```
bentuk potongan (band, baris, kolom): (2, 17, 21)
piksel di dalam S1: 300
NDVI rata-rata S1: 0.45
tersimpan: potongan_S1.tif
```

Potongan berukuran 17 baris kali 21 kolom: kotak yang memuat S1 (aslinya 15 baris × 20 kolom piksel), sedikit lebih besar karena batas kotak dibulatkan keluar (penyebab pastinya tidak saya periksa). Yang benar-benar di dalam petak: **300 piksel = 3 hektare**, sama dengan luas S1 di Bab 38. NDVI rata-ratanya 0,45. Perhatikan `.to_crs(citra.crs)`: poligon **harus** berCRS sama dengan raster. Bila lupa:

```
# Bab 39 - Jebakan: batas lahan berbeda CRS dengan raster
import geopandas as gpd
import rasterio
from rasterio.mask import mask

petak = gpd.read_file("data/petak.geojson")       # EPSG:4326 (derajat), lupa to_crs!
with rasterio.open("data/citra_tiruan.tif") as citra:    # raster EPSG:32748 (meter)
    print("CRS raster:", citra.crs, "| CRS vektor:", petak.crs)
    mask(citra, petak.geometry, crop=True)
```

```
CRS raster: EPSG:32748 | CRS vektor: EPSG:4326
Traceback (most recent call last):
    File  "<folder-proyek>\.venv\Lib\site-packages\rasterio\mask.py",  line  80,  in
raster_geometry_mask
    window = geometry_window(dataset, shapes, pad_x=pad_x, pad_y=pad_y)
   File "<folder-proyek>\.venv\Lib\site-packages\rasterio\features.py", line 522, in
geometry_window
    bounding_window = bounding_window.intersection(raster_window)
   File  "<folder-proyek>\.venv\Lib\site-packages\rasterio\windows.py",  line  787,  in
intersection
    return intersection([self, other])
   File  "<folder-proyek>\.venv\Lib\site-packages\rasterio\windows.py",  line  125,  in
wrapper
    return function(*args[0])
   File  "<folder-proyek>\.venv\Lib\site-packages\rasterio\windows.py",  line  239,  in
intersection
    return functools.reduce(_intersection, windows)
           ~~~~~~~~~~~~~~~~^^^^^^^^^^^^^^^^^^^^^^^
   File  "<folder-proyek>\.venv\Lib\site-packages\rasterio\windows.py",  line  257,  in
_intersection
    raise WindowError(f"Intersection is empty {w1} {w2}")
rasterio.errors.WindowError:    Intersection    is    empty    Window(col_off=-69990,
row_off=922680, width=1, height=1) Window(col_off=0, row_off=0, width=100, height=80)

During handling of the above exception, another exception occurred:

Traceback (most recent call last):
  File "08_salah_crs_mask.py", line 9, in <module>
    mask(citra, petak.geometry, crop=True)
    ~~~~^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "<folder-proyek>\.venv\Lib\site-packages\rasterio\mask.py", line 180, in mask
    shape_mask, transform, window = raster_geometry_mask(
                                    ~~~~~~~~~~~~~~~~~~~~^
        dataset, shapes, all_touched=all_touched, invert=invert, crop=crop,
        ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
        pad=pad, pad_width=pad_width)
        ^^^^^^^^^^^^^^^^^^^^^^^^^^^^
    File  "<folder-proyek>\.venv\Lib\site-packages\rasterio\mask.py",  line  86,  in
raster_geometry_mask
```

```
    raise ValueError('Input shapes do not overlap raster.')
ValueError: Input shapes do not overlap raster.
```

Rasterio menolak dengan `Input shapes do not overlap raster`: koordinat derajat (sekitar 106) tidak tumpang tindih dengan koordinat meter (sekitar 700000). Syukurlah di sini ada error. Bahkan tanpa error, hasil seperti ini selalu patut dicurigai.

Reproject dan resample

Reproject mengubah CRS raster; **resample** mengubah ukuran piksel. Keduanya memaksa komputer **menghitung ulang** nilai piksel, dan ada pilihan cara: `bilinear` (rata-rata berbobot dari tetangga, baik untuk data kontinu seperti NDVI), `average`, atau `nearest` (menyalin tetangga terdekat, wajib untuk data kelas seperti peta tutupan lahan, supaya tidak muncul "kelas 1,5").

```
# Bab 39 - Reproject (ganti CRS) dan resample (ganti resolusi)
import numpy as np
import rasterio
from rasterio.enums import Resampling
from rasterio.warp import calculate_default_transform, reproject

with rasterio.open("data/citra_tiruan.tif") as citra:
    nir = citra.read(2)

    # Reproject: UTM (meter) -> WGS 84 (derajat)
    transform, lebar, tinggi = calculate_default_transform(
        citra.crs, "EPSG:4326", citra.width, citra.height, *citra.bounds)
    hasil = np.zeros((tinggi, lebar), dtype="float32")
          reproject(source=nir,  destination=hasil,  src_transform=citra.transform,
src_crs=citra.crs,
                                    dst_transform=transform,  dst_crs="EPSG:4326",
resampling=Resampling.bilinear)
    print("sebelum:", citra.width, "x", citra.height, "piksel, satuan meter")
     print("sesudah:", lebar, "x", tinggi, "piksel, ukuran piksel (derajat): %.6f"
% transform.a)

   # Resample: baca langsung dengan resolusi 3x lebih kasar (30 m), nilai dirata-ratakan
       kasar = citra.read(2, out_shape=(citra.height // 3, citra.width // 3),
resampling=Resampling.average)
    print("\nresolusi 10 m :", nir.shape, "piksel")
    print("resolusi 30 m :", kasar.shape, "piksel")
    print("rata-rata NIR 10 m vs 30 m: %.4f vs %.4f" % (nir.mean(), kasar.mean()))
```

```
sebelum: 100 x 80 piksel, satuan meter
sesudah: 100 x 80 piksel, ukuran piksel (derajat): 0.000090

resolusi 10 m : (80, 100) piksel
resolusi 30 m : (26, 33) piksel
rata-rata NIR 10 m vs 30 m: 0.2249 vs 0.2249
```

Setelah reproject ke EPSG:4326, ukuran piksel menjadi sekitar 0,00009 derajat (kira-kira 10 m). Resolusi 10 m menjadi 30 m membuat kisi 26 × 33 piksel (sepertiga ukuran, dibulatkan ke bawah). Rata-rata NIR sebelum dan sesudah sama (0,2249): `Resampling.average` menjaga nilai rata-rata.

Statistik zonal

Statistik zonal merangkum nilai raster di dalam tiap **zona** (poligon): rata-rata, terkecil, terbesar. Kita membuat **masker** untuk tiap poligon dengan `geometry_mask`, lalu menghitung dengan NumPy.

```
# Bab 39 - Statistik zonal: ringkasan nilai raster di dalam tiap poligon
import geopandas as gpd
```

```
import rasterio
from rasterio.features import geometry_mask

with rasterio.open("data/citra_tiruan.tif") as citra:
    merah, nir = citra.read(1), citra.read(2)
    ndvi = (nir - merah) / (nir + merah)
    petak = gpd.read_file("data/petak.geojson").to_crs(citra.crs)
    kolam = gpd.read_file("data/kolam.geojson").to_crs(citra.crs)

    def zonal(zona):
        hasil = []
        for _, z in zona.iterrows():
            # invert=True: True di DALAM poligon, False di luarnya
                        dalam  =  geometry_mask([z.geometry],  out_shape=ndvi.shape,
transform=citra.transform, invert=True)
            nilai = ndvi[dalam]
               hasil.append((z["kode"], int(dalam.sum()), nilai.mean(), nilai.min(),
nilai.max()))
        return hasil

    print("zona     piksel  NDVI_rata  NDVI_min  NDVI_maks")
    for kode, n, rata, kecil, besar in zonal(petak):
        print(f"petak {kode:>2s}  {n:4d}  {rata:9.2f}  {kecil:8.2f}  {besar:9.2f}")
    for kode, n, rata, kecil, besar in zonal(kolam):
        print(f"kolam {kode:>2s}  {n:4d}  {rata:9.2f}  {kecil:8.2f}  {besar:9.2f}")
```

```
zona     piksel  NDVI_rata  NDVI_min  NDVI_maks
petak H1   900       0.80      0.71       0.89
petak H2   695       0.78      0.67       0.88
petak K1   400       0.60      0.51       0.68
petak K2   300       0.55      0.45       0.63
petak S1   300       0.45      0.36       0.55
petak S2   225       0.70      0.62       0.76
kolam  A    12      -0.20     -0.24      -0.15
kolam  B    12      -0.21     -0.25      -0.15
kolam  C    12      -0.20     -0.25      -0.14
kolam  D    16      -0.19     -0.26      -0.14
```

Hutan H1 dan H2 hijau lebat (0,80 dan 0,78), sawah S1 paling rendah (0,45; pada data tiruan ini nilai itu sengaja dibuat rendah, seolah padinya masih muda), dan keempat kolam berNDVI negatif (sekitar −0,2): air. Jumlah piksel pun cocok dengan luas: H1 900 piksel × 100 m² = 9 ha.

Proyek Bersambung I5: NDVI per petak

Hasil zonal dimasukkan kembali ke tabel petak dan ditandai.

```
# Bab 39 - Proyek bersambung I5: NDVI per petak, disimpan ke GeoPackage
from pathlib import Path

import geopandas as gpd
import numpy as np
import rasterio
from rasterio.features import geometry_mask

tmp = Path(__file__).parent / "_tmp"
tmp.mkdir(exist_ok=True)

with rasterio.open("data/citra_tiruan.tif") as citra:
    merah, nir = citra.read(1), citra.read(2)
    ndvi = (nir - merah) / (nir + merah)
```

```
    petak = gpd.read_file("data/petak.geojson").to_crs(citra.crs)
    rata = []
    for geom in petak.geometry:
        # invert=True: True DI DALAM poligon
        dalam = geometry_mask([geom], out_shape=ndvi.shape, transform=citra.transform,
invert=True)
        rata.append(round(float(ndvi[dalam].mean()), 2))

petak["ndvi_rata"] = rata
petak["status"] = np.where(petak["ndvi_rata"] < 0.5, "perlu dicek", "baik")       #
batas 0,5 = contoh
petak["luas_ha"] = (petak.area / 10_000).round(2)
print(petak[["kode",     "sektor",     "komoditas",     "luas_ha",     "ndvi_rata",
"status"]].to_string(index=False))

petak.to_file(tmp / "petak_ndvi.gpkg", layer="petak_ndvi", driver="GPKG")
print("\ntersimpan:    petak_ndvi.gpkg    |    kolom:",    list(gpd.read_file(tmp    /
"petak_ndvi.gpkg").columns))
```

```
kode sektor komoditas  luas_ha  ndvi_rata      status
  H1  hutan      Jati     9.00       0.80        baik
  H2  hutan   Meranti     6.93       0.78        baik
  K1  kebun      kopi     4.00       0.60        baik
  K2  kebun     kakao     3.00       0.55        baik
  S1  sawah      padi     3.00       0.45 perlu dicek
  S2  sawah      padi     2.25       0.70        baik

tersimpan:  petak_ndvi.gpkg  |  kolom:  ['kode',  'sektor',  'komoditas',  'ndvi_rata',
'status', 'luas_ha', 'geometry']
```

Hanya S1 (0,45) berstatus “perlu dicek” karena NDVI di bawah batas 0,5, yang di sini hanya angka contoh. Tabel itu disimpan ke GeoPackage: data vektor dan hasil raster akhirnya bersatu. Ingat bahwa citra ini **tiruan**; pada citra nyata, NDVI perlu dibandingkan dengan musim tanam dan pengamatan lapangan.

Latihan

1. Di `03_ndvi.py`, ubah batas kelas menjadi `[0.0, 0.2, 0.5]` dan lihat luas tiap kelas berubah.
2. Di `04_potong_batas_lahan.py`, potong petak `K1` (kebun kopi) dan cetak NDVI rata-ratanya.
3. Di `06_statistik_zonal.py`, tambahkan kolom simpangan baku (`values.std()`). Petak mana yang paling seragam?
4. **Tantangan:** buat kolom `jumlah_piksel_di_bawah_0_5` untuk tiap petak dan tampilkan persentasenya. Bandingkan dengan status “perlu dicek”.

Ringkasan Bab

- **Raster** = kisi piksel; **resolusi** = ukuran piksel di lapangan; **band** = lapisan nilai.
- **Rasterio** membaca band sebagai array NumPy; `xy` dan `index` berpindah antara piksel dan koordinat.
- **NDVI** = (NIR − merah) / (NIR + merah): air negatif, tanah terbuka rendah, tumbuhan lebat tinggi.
- `mask` memotong dengan poligon berCRS sama; `reproject` dan resample menghitung ulang piksel (pilih metode sesuai jenis data).
- **Statistik zonal** merangkum raster per poligon; hasilnya bisa digabung ke tabel vektor.

Selanjutnya, di Bab 40: peta statis dan interaktif, PostGIS, dan mengendalikan QGIS dari Python.

Bab 40 — GIS 3: Peta dan Integrasi: PostGIS dan QGIS

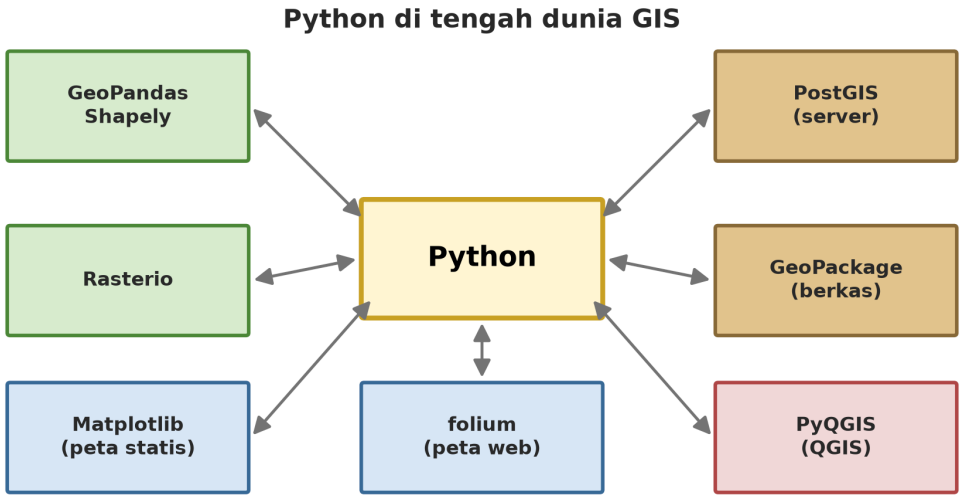
Seri I5 — Aplikasi Pertama: Web, Peta, dan Dasbor

Di bab ini, Anda akan:

- Membuat **peta statis** (Matplotlib) dan **peta interaktif** (folium)
- Menyimpan dan menanyai data spasial di **PostGIS** dari Python
- Mengendalikan **QGIS** dengan **PyQGIS** tanpa membuka jendelanya
- Memahami perbedaan PyQGIS di **QGIS 4** (Qt6) dibanding QGIS 3
- Menyimpan data lokasi usaha di PostGIS **dan** GeoPackage: proyek bersambung

Semua skrip bab ini ada di folder `kode/bab40/`. Di versi web, Anda juga bisa mengunduh semuanya sekaligus (zip).

Dua bab GIS sebelumnya membuat Anda bisa **menghitung**. Sekarang kita **menyajikan** dan **menyimpan**. Peta adalah cara data spasial dibaca orang; basis data spasial adalah gudang yang tahan lama; dan QGIS adalah "dapur peta" yang dipakai banyak instansi. Python bisa duduk di tengah ketiganya.



Gambar 0.1: Python sebagai pusat: membaca dan menulis ke berbagai bentuk penyimpanan dan penyaji peta.

Peta statis dengan Matplotlib

Peta statis adalah gambar (PNG, PDF) untuk laporan dan buku. GeoPandas menggambar lewat Matplotlib (Bab 25), jadi semua kendali Matplotlib tersedia: judul, legenda, label.

```
# Bab 40 - Peta statis dengan GeoPandas dan Matplotlib
from pathlib import Path

import geopandas as gpd
import matplotlib.pyplot as plt
from matplotlib.lines import Line2D
```

```
tmp = Path(__file__).parent / "_tmp"
tmp.mkdir(exist_ok=True)
petak = gpd.read_file("data/petak.geojson").to_crs(32748)
kolam = gpd.read_file("data/kolam.geojson").to_crs(32748)
kandang = gpd.read_file("data/kandang.geojson").to_crs(32748)
sungai = gpd.read_file("data/sungai.geojson").to_crs(32748)

warna = {"hutan": "#4f8a3b", "kebun": "#d9a441", "sawah": "#e8dc7a"}
fig, ax = plt.subplots(figsize=(7, 5.6), dpi=100)
for sektor, grup in petak.groupby("sektor"):
    grup.plot(ax=ax, color=warna[sektor], edgecolor="black", linewidth=0.8)
sungai.plot(ax=ax, color="#3b6a9a", linewidth=2.5)
kolam.plot(ax=ax, color="#1f4e79")
kandang.plot(ax=ax, color="#b04a4a", markersize=60, marker="s")
for _, p in petak.iterrows():                                    # label kode petak
   ax.annotate(p["kode"], (p.geometry.centroid.x, p.geometry.centroid.y), ha="center",
weight="bold")

ax.set_title("Peta lokasi usaha (data tiruan)")
ax.set_xlabel("Timur (m, UTM 48S)")
ax.set_ylabel("Utara (m, UTM 48S)")
ax.ticklabel_format(style="plain", useOffset=False)
ax.set_aspect("equal")                                           # 1 m di sumbu X = 1 m
di sumbu Y
legenda = [Line2D([0], [0], marker="s", color="w", markerfacecolor=w, markersize=12,
label=s)
           for s, w in warna.items()]
legenda += [Line2D([0], [0], color="#3b6a9a", lw=2.5, label="sungai"),
                Line2D([0], [0], marker="s", color="w", markerfacecolor="#1f4e79",
markersize=9, label="kolam"),
                Line2D([0], [0], marker="s", color="w", markerfacecolor="#b04a4a",
markersize=9, label="kandang")]
ax.legend(handles=legenda, loc="upper left", fontsize=8)
fig.tight_layout()
fig.savefig(tmp / "peta_statis.png")
print("peta tersimpan:", "peta_statis.png", "| ukuran piksel:", fig.get_size_inches()
* fig.dpi)
```

```
peta tersimpan: peta_statis.png | ukuran piksel: [700. 560.]
```

Hal yang membedakan peta dari grafik biasa:

- **`ax.set_aspect("equal")`**: 1 meter di sumbu X digambar sama panjang dengan 1 meter di sumbu Y. Tanpa ini, petak persegi tampak gepeng.
- Data digambar dalam **UTM** (meter), jadi sumbu bisa diberi label meter dan jarak terbaca benar.
- Warna dipilih per sektor lewat `groupby`, lalu legenda dibuat sendiri dari `Line2D`.

Gambar 700 × 560 piksel itu tersimpan di `_tmp/peta_statis.png`. Contoh peta yang sama dengan zona 60 m sudah Anda lihat di Bab 38.

Peta interaktif dengan folium

folium membungkus pustaka JavaScript **Leaflet** dan menghasilkan **berkas HTML** yang bisa digeser dan diperbesar di browser, lengkap dengan penanda dan *popup*.

```
# Bab 40 - Peta interaktif dengan folium (hasilnya berkas HTML)
from pathlib import Path

import folium
import geopandas as gpd
```

```
tmp = Path(__file__).parent / "_tmp"
tmp.mkdir(exist_ok=True)
petak = gpd.read_file("data/petak.geojson")          # folium butuh EPSG:4326 (derajat)
kolam = gpd.read_file("data/kolam.geojson")
kandang = gpd.read_file("data/kandang.geojson")
sungai = gpd.read_file("data/sungai.geojson")

pusat = petak.union_all().centroid
peta = folium.Map(location=[pusat.y, pusat.x], zoom_start=16, tiles="OpenStreetMap")
# urutan: lintang, bujur

folium.GeoJson(petak, name="petak",
                           tooltip=folium.GeoJsonTooltip(fields=["kode",  "sektor",
"komoditas"])).add_to(peta)
folium.GeoJson(sungai,  name="sungai",  style_function=lambda  f:  {"color":  "#3b6a9a",
"weight": 4}).add_to(peta)
for _, k in kolam.iterrows():
    folium.Marker([k.geometry.centroid.y, k.geometry.centroid.x],
                  popup=f"Kolam {k['kode']}: {k['jumlah_ikan']} ikan {k['jenis']}",
                  icon=folium.Icon(color="blue", icon="tint")).add_to(peta)
for _, k in kandang.iterrows():
        folium.Marker([k.geometry.y,  k.geometry.x],  popup=f"Kandang  {k['kode']}:
{k['jenis']}",
                  icon=folium.Icon(color="red")).add_to(peta)
folium.LayerControl().add_to(peta)

peta.save(tmp / "peta_usaha.html")
html = (tmp / "peta_usaha.html").read_text(encoding="utf-8")
print("berkas HTML tersimpan:", "peta_usaha.html")
print("memuat pustaka Leaflet:", "leaflet" in html.lower())
print("jumlah penanda (Marker):", html.count("L.marker("))
print("berisi nama petak 'Meranti':", "Meranti" in html)
```

```
berkas HTML tersimpan: peta_usaha.html
memuat pustaka Leaflet: True
jumlah penanda (Marker): 7
berisi nama petak 'Meranti': True
```

- folium memakai **EPSG:4326** (derajat), dan urutan lokasinya **[lintang, bujur]**: kebalikan dari (x, y). Kekeliruan urutan ini membuat peta "mendarat" di laut.
- Skrip memasang lapisan petak (dengan *tooltip* yang muncul saat kursor mendekat), sungai, tujuh penanda (empat kolam dan tiga kandang), dan kontrol lapisan.
- Bukti yang bisa kita cetak: berkas memuat Leaflet, memuat tujuh `L.marker(`, dan memuat nama komoditas `Meranti`.

> BAGIAN TEKNIS (boleh dilewati)
>
> Saya **tidak membuka** `peta_usaha.html` di browser dalam pembuatan buku ini; yang diperiksa hanya isi berkasnya, seperti pada keluaran di atas. Perhatikan juga bahwa peta dasar (`OpenStreetMap`) dimuat dari internet saat peta dibuka, jadi tanpa sinyal Anda hanya melihat bentuk petak tanpa latar. Menyalin data sensitif ke berkas HTML berarti data itu ikut terkirim ke siapa pun yang menerima berkasnya.

PostGIS: basis data yang mengerti peta

PostgreSQL adalah basis data relasional (Bab 24); **PostGIS** adalah ekstensinya yang menambah tipe `geometry`, ratusan fungsi spasial (`ST_Area`, `ST_Distance`, ...), dan **indeks spasial**. Keuntungannya dibanding berkas: banyak pengguna sekaligus, data besar, dan pencarian cepat berdasarkan letak.

Untuk bab ini, pembangun buku menyalakan **klaster PostgreSQL sementara** (port acak, hanya di komputer ini, dihapus sesudahnya) dan memberi alamatnya lewat variabel `DATABASE_URL`. Skrip yang memakai ini ditandai `@pg`. Server PostgreSQL milik penulis tidak disentuh. Di komputer Anda, nilai bawaan `postgresql://postgres@localhost:5432/belajar` hanyalah **contoh**: ganti dengan alamat server milik Anda (dan jangan menaruh kata sandi di kode; ambil dari variabel lingkungan, Bab 34).

Membuat tabel dan memuat data

```
# Bab 40 - PostGIS: membuat tabel spasial dan memuat data dengan psycopg
import os

import geopandas as gpd
import psycopg

# Di komputer Anda, ganti alamat bawaan ini dengan alamat server PostgreSQL+PostGIS
milik Anda
alamat   =   os.environ.get("DATABASE_URL",   "postgresql://postgres@localhost:5432/
belajar")

petak = gpd.read_file("data/petak.geojson")
kolam = gpd.read_file("data/kolam.geojson")
sungai = gpd.read_file("data/sungai.geojson")

with psycopg.connect(alamat) as sambungan:
    sambungan.execute("CREATE EXTENSION IF NOT EXISTS postgis")
    versi = sambungan.execute("SELECT PostGIS_Lib_Version()").fetchone()[0]
    print("PostGIS aktif, versi mayor:", versi.split(".")[0])

    sambungan.execute("DROP TABLE IF EXISTS petak, kolam, sungai")
    sambungan.execute("""CREATE TABLE petak (
        kode text PRIMARY KEY, sektor text, komoditas text,
         geom geometry(Polygon, 4326))""")                      # tipe geometri DAN
SRID dikunci
    sambungan.execute("""CREATE TABLE kolam (
        kode text PRIMARY KEY, jenis text, jumlah_ikan int,
        geom geometry(Polygon, 4326))""")

    sambungan.execute("""CREATE TABLE sungai (
        nama text, geom geometry(LineString, 4326))""")

    for _, p in petak.iterrows():                              # WKT = teks geometri
         sambungan.execute("INSERT INTO petak VALUES (%s, %s, %s, ST_GeomFromText(%s,
4326))",
                           (p["kode"], p["sektor"], p["komoditas"], p.geometry.wkt))
    for _, k in kolam.iterrows():
         sambungan.execute("INSERT INTO kolam VALUES (%s, %s, %s, ST_GeomFromText(%s,
4326))",
                       (k["kode"], k["jenis"], int(k["jumlah_ikan"]), k.geometry.wkt))

    sambungan.execute("INSERT INTO sungai VALUES (%s, ST_GeomFromText(%s, 4326))",
                      (sungai["nama"].iloc[0], sungai.geometry.iloc[0].wkt))

    # Indeks spasial GiST: membuat pencarian berdasarkan letak cepat
    sambungan.execute("CREATE INDEX petak_geom_idx ON petak USING GIST (geom)")
    sambungan.execute("CREATE INDEX kolam_geom_idx ON kolam USING GIST (geom)")

    for tabel in ("petak", "kolam", "sungai"):
        n = sambungan.execute(f"SELECT count(*) FROM {tabel}").fetchone()[0]
        print(f"baris di tabel {tabel}:", n)
    print("jenis geometri tabel petak:",
```

```
                    sambungan.execute("SELECT  DISTINCT  ST_GeometryType(geom)  FROM
petak").fetchone()[0])
```

```
PostGIS aktif, versi mayor: 3
baris di tabel petak: 6
baris di tabel kolam: 4
baris di tabel sungai: 1
jenis geometri tabel petak: ST_Polygon
```

- `CREATE EXTENSION IF NOT EXISTS postgis` mengaktifkan PostGIS di basis data (butuh hak yang cukup di server Anda).
- `geometry(Polygon, 4326)` mengunci **jenis bentuk** dan **SRID** (nomor CRS). Memasukkan jenis lain akan ditolak: pengaman yang tidak ada pada berkas.
- Bentuk dikirim sebagai **WKT** (*Well-Known Text*), teks seperti `POLYGON((...))`, dan diubah oleh `ST_GeomFromText`.
- **Indeks GiST** membuat pencarian berdasarkan letak cepat. Hasil: 6 petak, 4 kolam, 1 sungai.

Query spasial

```
# Bab 40 - Query spasial di PostGIS (memakai tabel dari skrip 03)
import os

import psycopg

alamat   =   os.environ.get("DATABASE_URL",   "postgresql://postgres@localhost:5432/
belajar")

with psycopg.connect(alamat) as sambungan:
    # Luas: tipe geography menghitung di bumi bulat, hasilnya meter persegi
    print("luas petak (ha):")
    for kode, ha in sambungan.execute(
            "SELECT kode, round((ST_Area(geom::geography) / 10000)::numeric, 2) FROM
petak ORDER BY kode"):
        print(f"  {kode}: {ha}")

     # Petak dalam jarak 60 m dari sungai: ST_DWithin pada tipe geography (jarak
dalam meter)
    print("\npetak dalam 60 m dari sungai:")
    for (kode,) in sambungan.execute(
            "SELECT p.kode FROM petak p, sungai s "
         "WHERE ST_DWithin(p.geom::geography, s.geom::geography, 60) ORDER BY p.kode"):
        print(" ", kode)

   # Kolam terdekat ke titik gudang (operator <-> = urut menurut jarak, memakai indeks)
    print("\ntiga kolam terdekat ke titik gudang:")
    gudang = "ST_SetSRID(ST_MakePoint(106.8140, -6.9950), 4326)"
    for kode, jarak in sambungan.execute(
                                f"SELECT   kode,   round(ST_Distance(geom::geography,
{gudang}::geography)::numeric) "
            f"FROM kolam ORDER BY geom <-> {gudang} LIMIT 3"):
        print(f"  kolam {kode}: {jarak} m")

     # Menggabungkan dua tabel menurut letak: tabel petak dan kolam yang saling
bersinggungan
        n  =  sambungan.execute("SELECT  count(*)  FROM  petak  p  JOIN  kolam  k  ON
ST_Intersects(p.geom, k.geom)").fetchone()[0]
    print("\npasangan petak-kolam yang saling tumpang tindih:", n)
```

```
luas petak (ha):
  H1: 9.00
  H2: 6.92
  K1: 4.00
  K2: 3.00
  S1: 3.00
  S2: 2.25

petak dalam 60 m dari sungai:
  K2
  S1
  S2

tiga kolam terdekat ke titik gudang:
  kolam B: 82 m
  kolam C: 172 m
  kolam A: 274 m

pasangan petak-kolam yang saling tumpang tindih: 0
```

Perhatikan:

- Memakai `::geography` membuat PostGIS menghitung di **bumi bulat** dan menjawab dalam **meter** walau datanya berderajat. H2 muncul 6,92 ha di sini, sedangkan GeoPandas (UTM) menghitung 6,93 ha: **dua metode berbeda menghasilkan selisih kecil** karena proyeksi UTM sedikit mengubah ukuran. Untuk keperluan lapangan keduanya cukup dekat; untuk keperluan hukum, ikuti ketentuan resminya.
- `ST_DWithin(..., 60)` menemukan petak dalam 60 m dari sungai: K2, S1, S2, sama dengan hasil overlay Bab 38.
- Operator `<->` mengurutkan menurut jarak dan memanfaatkan indeks: tiga kolam terdekat ke titik gudang (titik contoh) adalah B (82 m), C (172 m), A (274 m).
- Tidak ada petak yang tumpang tindih dengan kolam (hasil 0).

> ⚠ AWAS
>
> Jangan menyusun SQL dengan menempel teks pengguna ke dalam string (`f"... {isian} ..."`): itu membuka celah **SQL injection**. Untuk **nilai**, selalu pakai parameter `%s` seperti pada `INSERT` di skrip 03. Skrip 04 menempel teks hanya karena isinya konstanta tetap dari kita sendiri, bukan dari pengguna.

PostGIS dan GeoPandas

```
# Bab 40 - PostGIS dan GeoPandas: baca query, tulis tabel baru
import os

import geopandas as gpd
from sqlalchemy import create_engine, text

alamat   =   os.environ.get("DATABASE_URL",   "postgresql://postgres@localhost:5432/
belajar")
mesin = create_engine(alamat.replace("postgresql://", "postgresql+psycopg://"))

# Baca hasil query langsung menjadi GeoDataFrame
sql = "SELECT kode, sektor, geom FROM petak WHERE sektor <> 'hutan'"
bukan_hutan = gpd.read_postgis(sql, mesin, geom_col="geom")
print(bukan_hutan[["kode", "sektor"]].to_string(index=False))
print("CRS terbaca:", bukan_hutan.crs)

# Tulis GeoDataFrame ke tabel baru (geoalchemy2 dibutuhkan di belakang layar)
```

```
kandang = gpd.read_file("data/kandang.geojson")
kandang.to_postgis("kandang", mesin, if_exists="replace", index=False)
with mesin.connect() as k:
        print("baris di tabel kandang:", k.execute(text("SELECT count(*) FROM
kandang")).scalar())
    print("tipe kolom geometri:", k.execute(text(
        "SELECT type || ' / SRID ' || srid FROM geometry_columns WHERE f_table_name
= 'kandang'")).scalar())
```

```
kode sektor
  K1  kebun
  K2  kebun
  S1  sawah
  S2  sawah
CRS terbaca: EPSG:4326
baris di tabel kandang: 3
tipe kolom geometri: POINT / SRID 4326
```

`read_postgis` mengubah hasil query menjadi GeoDataFrame, dengan CRS terbaca otomatis (EPSG:4326). `to_postgis` menulis GeoDataFrame ke tabel baru dan membuat kolom bertipe geometri (`POINT / SRID 4326`). Keduanya membutuhkan **SQLAlchemy** dan, untuk menulis, **GeoAlchemy2** (versi 0.20.0 saya pasang lewat `alat/pasang.py`).

PyQGIS: mengendalikan QGIS dari Python

QGIS adalah program GIS bebas yang sangat populer. Ia punya Python sendiri dan modul bernama **PyQGIS** (`qgis.core`, `qgis.analysis`) untuk mengotomasi pekerjaan: memuat lapisan, memproses, memberi gaya, mencetak peta. Cara pakai: skrip dijalankan oleh **Python milik QGIS**, bukan Python proyek ini.

Di komputer penulis, QGIS **4.0.2** terpasang di `C:\Program Files\QGIS 4.0.2` dengan Python 3.12 miliknya sendiri. Perintah `bin\python-qgis.bat` menyiapkan semua jalur lalu menjalankan Python itu. Karena itulah dua skrip berikut diberi arahan `@python:` yang menunjuk ke berkas itu; **di komputer Anda jalurnya akan berbeda** (atau memakai konsol Python di dalam QGIS).

Pola dasar skrip tanpa antarmuka (sesuai cookbook PyQGIS resmi):

1. `QgsApplication.setPrefixPath("<folder qgis>", True)`: beri tahu lokasi pemasangan.
2. `app = QgsApplication([], False)`: `False` = **tanpa GUI**.
3. `app.initQgis()`: muat penyedia data.
4. … kerja Anda …
5. `app.exitQgis()`: bersih-bersih.

Tambahan kita: `os.environ["QT_QPA_PLATFORM"] = "offscreen"` sebelum `QgsApplication` dibuat, supaya Qt tidak mencari layar.

```
# Bab 40 - PyQGIS tanpa antarmuka: memuat lapisan, membaca atribut
import os

os.environ["QT_QPA_PLATFORM"] = "offscreen"          # tanpa jendela (harus sebelum
QgsApplication dibuat)

from qgis.core import Qgis, QgsApplication, QgsProject, QgsVectorLayer

QgsApplication.setPrefixPath("C:/Program Files/QGIS 4.0.2/apps/qgis", True)
app = QgsApplication([], False)                   # False = tanpa GUI
app.initQgis()
print("versi mayor QGIS:", Qgis.version().split(".")[0])

petak = QgsVectorLayer("data/petak.geojson", "petak", "ogr")
print("lapisan valid:", petak.isValid(), "| jumlah fitur:", petak.featureCount())
print("CRS:", petak.crs().authid(), "| jenis geometri:", petak.geometryType().name)
```

```
print("kolom:", [f.name() for f in petak.fields()])

for fitur in petak.getFeatures():
    if fitur["sektor"] == "sawah":
        print("sawah:", fitur["kode"], fitur["komoditas"])

QgsProject.instance().addMapLayer(petak)
print("lapisan di proyek:", [l.name() for l in
QgsProject.instance().mapLayers().values()])
app.exitQgis()
```

```
versi mayor QGIS: 4
lapisan valid: True | jumlah fitur: 6
CRS: EPSG:4326 | jenis geometri: Polygon
kolom: ['kode', 'sektor', 'komoditas']
sawah: S1 padi
sawah: S2 padi
lapisan di proyek: ['petak']
```

Keluaran membuktikan skrip berjalan di **QGIS 4** (versi mayor 4): enam fitur terbaca, CRS EPSG:4326, jenis geometri poligon, dan kolom `kode`, `sektor`, `komoditas`. Sawah S1 dan S2 ditemukan dengan menyaring atribut.

Algoritma Processing, gaya, dan menggambar peta

QGIS punya ratusan algoritma pengolah (*Processing*), mis. `native:buffer` dan `native:reprojectlayer`. Dari skrip, panggil dengan `processing.run(nama, parameter)`.

```
# Bab 40 - PyQGIS: algoritma Processing, memberi gaya, menggambar peta ke PNG
import os

os.environ["QT_QPA_PLATFORM"] = "offscreen"

import sys
from pathlib import Path

from qgis.core import (QgsApplication, QgsCategorizedSymbolRenderer,
QgsCoordinateReferenceSystem,
                       QgsMapRendererSequentialJob, QgsMapSettings, QgsProject,
QgsRendererCategory,
                       QgsSymbol, QgsVectorLayer)
from qgis.PyQt.QtCore import QSize
from qgis.PyQt.QtGui import QColor

tmp = Path(__file__).parent / "_tmp"
tmp.mkdir(exist_ok=True)
QgsApplication.setPrefixPath("C:/Program Files/QGIS 4.0.2/apps/qgis", True)
app = QgsApplication([], False)
app.initQgis()

# Processing: modul `processing` ada di folder plugin QGIS, jadi jalurnya ditambahkan
dulu
sys.path.append("C:/Program Files/QGIS 4.0.2/apps/qgis/python/plugins")
import processing
from processing.core.Processing import Processing
from qgis.analysis import QgsNativeAlgorithms

QgsApplication.processingRegistry().addProvider(QgsNativeAlgorithms())
Processing.initialize()
```

```
petak = QgsVectorLayer("data/petak.geojson", "petak", "ogr")
sungai = QgsVectorLayer("data/sungai.geojson", "sungai", "ogr")

datar = processing.run("native:reprojectlayer", {
   "INPUT": petak, "TARGET_CRS": QgsCoordinateReferenceSystem("EPSG:32748"), "OUTPUT":
"memory:"})["OUTPUT"]
print("luas   petak   di   UTM   (m2):",   [round(f.geometry().area())   for   f   in
datar.getFeatures()])

sungai_utm = processing.run("native:reprojectlayer", {
        "INPUT":  sungai,  "TARGET_CRS":  QgsCoordinateReferenceSystem("EPSG:32748"),
"OUTPUT": "memory:"})["OUTPUT"]
zona   =   processing.run("native:buffer",   {"INPUT":   sungai_utm,   "DISTANCE":   60,
"SEGMENTS": 8, "OUTPUT": "memory:"})["OUTPUT"]
print("jumlah fitur zona penyangga:", zona.featureCount(), "| luas (ha):",
      round(next(zona.getFeatures()).geometry().area() / 10_000, 2))

# Gaya: warna berdasarkan kolom sektor
kategori = []
for nilai, warna in [("hutan", "#4f8a3b"), ("kebun", "#d9a441"), ("sawah", "#e8dc7a")]:
    simbol = QgsSymbol.defaultSymbol(datar.geometryType())
    simbol.setColor(QColor(warna))
    kategori.append(QgsRendererCategory(nilai, simbol, nilai))
datar.setRenderer(QgsCategorizedSymbolRenderer("sektor", kategori))
zona.renderer().symbol().setColor(QColor(59, 106, 154, 90))     # biru transparan

# Menggambar peta ke gambar PNG
pengaturan = QgsMapSettings()
pengaturan.setLayers([zona, datar])
pengaturan.setDestinationCrs(datar.crs())
pengaturan.setExtent(zona.extent())
pengaturan.setOutputSize(QSize(700, 450))
pengaturan.setBackgroundColor(QColor("white"))
tugas = QgsMapRendererSequentialJob(pengaturan)
tugas.start()
tugas.waitForFinished()
gambar = tugas.renderedImage()
print("ukuran  gambar:",  gambar.width(),  "x",  gambar.height(),  "|  tersimpan:",
gambar.save(str(tmp / "peta_qgis.png")))

proyek = QgsProject.instance()
proyek.setCrs(datar.crs())
proyek.addMapLayers([zona, datar])
print("proyek QGIS tersimpan:", proyek.write(str(tmp / "usaha.qgz")))
app.exitQgis()
```

```
luas petak di UTM (m2): [90001, 69252, 39999, 30000, 29999, 22500]
jumlah fitur zona penyangga: 1 | luas (ha): 13.23
ukuran gambar: 700 x 450 | tersimpan: True
proyek QGIS tersimpan: True
```

Luas petak setelah diubah ke UTM (90001, 69252, 39999, … m²) cocok dengan hitungan GeoPandas. Zona penyangga 60 m berluas 13,23 ha (GeoPandas 13,24 ha: beda sedikit karena banyaknya segmen pada ujung buffer). Sesudah itu skrip memberi **gaya** per kategori (warna menurut kolom `sektor`), menggambar peta 700 × 450 piksel ke PNG, dan menyimpan **proyek QGIS** (`.qgz`) yang bisa dibuka di QGIS sungguhan.

TIPS

Karena `modul processing` ada di folder plugin QGIS, skrip menambahkan `.../apps/qgis/python/plugins` ke `sys.path` dan mendaftarkan `QgsNativeAlgorithms` sebelum `processing.run`. Tanpa dua langkah itu, algoritma tidak ditemukan.

QGIS 4 berbeda dari QGIS 3

QGIS 4.0 (dirilis Maret 2026) pindah dari Qt5/PyQt5 ke **Qt6/PyQt6**. Akibatnya banyak kode PyQGIS lama perlu disesuaikan. Dua perubahan yang paling sering menabrak:

1. **Enum Qt harus ditulis lengkap**: `Qt.AlignLeft` menjadi `Qt.AlignmentFlag.AlignLeft`.
2. **Tipe kolom memakai `QMetaType.Type`** (bukan `QVariant`) pada `QgsField`.

```
# Bab 40 - QGIS 4 (Qt6): enum Qt harus ditulis lengkap, tipe kolom memakai QMetaType
import os

os.environ["QT_QPA_PLATFORM"] = "offscreen"

from qgis.core import Qgis, QgsField
from qgis.PyQt.QtCore import QMetaType, Qt

# Gaya QGIS 3 (PyQt5): Qt.AlignLeft. Di QGIS 4 (PyQt6) nama pendek itu sudah tidak ada.
try:
    Qt.AlignLeft
except AttributeError as galat:
    print("gaya lama gagal ->", galat)
print("gaya baru Qt.AlignmentFlag.AlignLeft ->", int(Qt.AlignmentFlag.AlignLeft))

# Tipe kolom: QMetaType.Type menggantikan QVariant pada QgsField
kolom = QgsField("jumlah_ikan", QMetaType.Type.Int)
print("nama kolom:", kolom.name(), "| tipe:", kolom.type())

# Enum milik QGIS sendiri juga dianjurkan ditulis lengkap
print("jenis geometri poligon:", Qgis.GeometryType.Polygon.name)
```

```
gaya lama gagal -> type object 'Qt' has no attribute 'AlignLeft'
gaya baru Qt.AlignmentFlag.AlignLeft -> 1
nama kolom: jumlah_ikan | tipe: Type.Int
jenis geometri poligon: Polygon
```

Yang saya buktikan di QGIS 4.0.2 komputer ini: gaya lama `Qt.AlignLeft` gagal dengan `AttributeError`, gaya baru berhasil, dan `QgsField` dengan `QMetaType.Type.Int` bekerja. Sebagian nama QGIS lama masih berfungsi lewat lapisan kompatibilitas (saya mencoba `QgsWkbTypes.PolygonGeometry` dan `QgsLayoutExporter.Success`, keduanya masih menjawab), tetapi bentuk lengkap seperti `Qgis.GeometryType.Polygon` lebih aman ke depan. Daftar lengkap perubahan ada di panduan kompatibilitas Qt6 di situs QGIS; saya tidak membaca seluruhnya, jadi rujuklah panduan itu untuk kode Anda sendiri.

BAGIAN TEKNIS (boleh dilewati)

Dua temuan dari percobaan di komputer penulis: (1) layout cetak (`QgsPrintLayout`) bisa diekspor ke PNG tanpa antarmuka, tetapi **teks pada label tampil sebagai kotak hitam** pada mode `offscreen` (kemungkinan karena tidak ada font yang dimuat; penyebab pastinya belum saya periksa). Karena itu skrip bab ini memakai `QgsMapRendererSequentialJob` untuk menggambar peta, bukan layout bertulisan. (2) Ekspor layout memancarkan pesan GDAL "ERROR 6" di keluaran walau berhasil; saya tidak memakainya agar keluaran bersih.

Alur kerja Python dan QGIS

Kebutuhan	Alat yang cocok
Hitungan dan analisis berulang	**GeoPandas** / **Rasterio** (Python proyek Anda)
Penyimpanan bersama, banyak pengguna	**PostGIS**
Memeriksa dan menyunting peta dengan mata dan tikus	**QGIS** (antarmuka)
Mengotomasi langkah QGIS (cetak 20 peta serupa)	**PyQGIS**
Peta untuk laporan atau buku	**Matplotlib**
Peta untuk dibagikan di web	**folium**

Pola kerja yang umum: data masuk ke **PostGIS atau GeoPackage**; analis memeriksanya di **QGIS**; skrip **GeoPandas** menghitung ringkasannya; dan **PyQGIS** atau Matplotlib mencetak peta mingguan.

Proyek Bersambung I5: data lokasi usaha di dua tempat

Lima lapisan lokasi (petak, kolam, kandang, sungai, dusun) kita simpan ke **PostGIS** (untuk dipakai bersama dan nanti dibaca API) **dan** ke **GeoPackage** (satu berkas yang bisa dikirim lewat surel dan dibuka di QGIS), lalu dicek bahwa keduanya berisi hal yang sama.

```
# Bab 40 - Proyek bersambung I5: data lokasi disimpan di PostGIS dan GeoPackage
import os
from pathlib import Path

import geopandas as gpd
import pandas as pd
from sqlalchemy import create_engine

tmp = Path(__file__).parent / "_tmp"
tmp.mkdir(exist_ok=True)
alamat   =   os.environ.get("DATABASE_URL",   "postgresql://postgres@localhost:5432/
belajar")
mesin = create_engine(alamat.replace("postgresql://", "postgresql+psycopg://"))

lapisan = ["petak", "kolam", "kandang", "sungai", "dusun"]
gpkg = tmp / "usaha.gpkg"
gpkg.unlink(missing_ok=True)
for nama in lapisan:
    gdf = gpd.read_file(f"data/{nama}.geojson")
    gdf.to_postgis(f"usaha_{nama}", mesin, if_exists="replace", index=False)       #
server
    gdf.to_file(gpkg, layer=nama, driver="GPKG")                                    #
berkas portabel

print("lapisan  PostGIS  GeoPackage")
for nama in lapisan:
   a = len(gpd.read_postgis(f"SELECT * FROM usaha_{nama}", mesin, geom_col="geometry"))
    b = len(gpd.read_file(gpkg, layer=nama))
    print(f"{nama:8s} {a:7d} {b:11d}")

# Luas per sektor: dihitung di server PostGIS dan di Python dari GeoPackage; harus sama
dari_server = pd.read_sql(
    "SELECT sektor, round((sum(ST_Area(geometry::geography)) / 10000)::numeric, 1) AS
ha "
    "FROM usaha_petak GROUP BY sektor ORDER BY sektor", mesin)
```

```
print("\nluas per sektor (ha) dari PostGIS:")
print(dari_server.to_string(index=False))
petak = gpd.read_file(gpkg, layer="petak").to_crs(32748)
print("\nluas per sektor (ha) dari GeoPackage:")
print(petak.assign(ha=petak.area                                                /
10_000).groupby("sektor")["ha"].sum().round(1).to_string())
```

```
lapisan  PostGIS  GeoPackage
petak          6           6
kolam          4           4
kandang        3           3
sungai         1           1
dusun          2           2

luas per sektor (ha) dari PostGIS:
sektor   ha
 hutan 15.9
 kebun  7.0
 sawah  5.2

luas per sektor (ha) dari GeoPackage:
sektor
hutan    15.9
kebun     7.0
sawah     5.2
```

Lima lapisan, jumlah baris sama di kedua tempat, dan luas per sektor cocok: hutan 15,9 ha, kebun 7,0 ha, sawah 5,2 ha (ketiganya dibulatkan satu desimal; sawah 5,25 menjadi 5,2 karena pembulatan Python). Tabel di PostGIS diberi awalan `usaha_` supaya tidak bertabrakan dengan tabel skrip 03. Itu mengakhiri bagian peta dari proyek Seri I5: data lokasi sudah rapi dan tersimpan.

Latihan

1. Di `01_peta_statis.py`, tambah garis skala sederhana (sebuah garis 100 m dengan teks “100 m”) di pojok kanan bawah peta.
2. Di `02_peta_folium.py`, ganti `tiles="OpenStreetMap"` menjadi `tiles="CartoDB positron"` dan lihat perbedaan isi berkas.
3. Di `04_postgis_query_spasial.py`, tambah query yang menghitung jumlah ikan per dusun memakai `ST_Within` antara titik pusat kolam (`ST_Centroid`) dan poligon dusun. (Tabel dusun belum dimuat: tambahkan di skrip 03.)
4. **Tantangan:** di `07_pyqgis_proses_dan_cetak.py`, tambah algoritma `native:intersection` antara petak dan zona, lalu cetak luas potongan tiap petak, dan cocokkan dengan Bab 38.

Ringkasan Bab

- **Matplotlib** untuk peta statis (jangan lupa `set_aspect("equal")`); **folium** untuk peta interaktif (HTML, urutan [lintang, bujur]).
- **PostGIS** menambah tipe `geometry`, fungsi `ST_*`, dan indeks spasial pada PostgreSQL; kirim nilai lewat parameter, bukan tempelan teks.
- `::geography` menghitung di bumi bulat dalam meter; hasilnya dapat sedikit berbeda dari hitungan UTM.
- **PyQGIS** berjalan di Python milik QGIS; skrip tanpa antarmuka butuh `setPrefixPath`, `QgsApplication([], False)`, `initQgis`, dan `exitQgis`.
- **QGIS 4** memakai Qt6: tulis enum Qt lengkap dan pakai `QMetaType.Type`.
- Penyimpanan ganda: PostGIS untuk bersama, GeoPackage untuk portabel.

Selanjutnya, di Bab 41: otomasi tugas dan web scraping.

Bab 41 — Automasi dan Scripting: Biarkan Python Bekerja Sendiri

Seri I5 — Aplikasi Pertama: Web, Peta, dan Dasbor

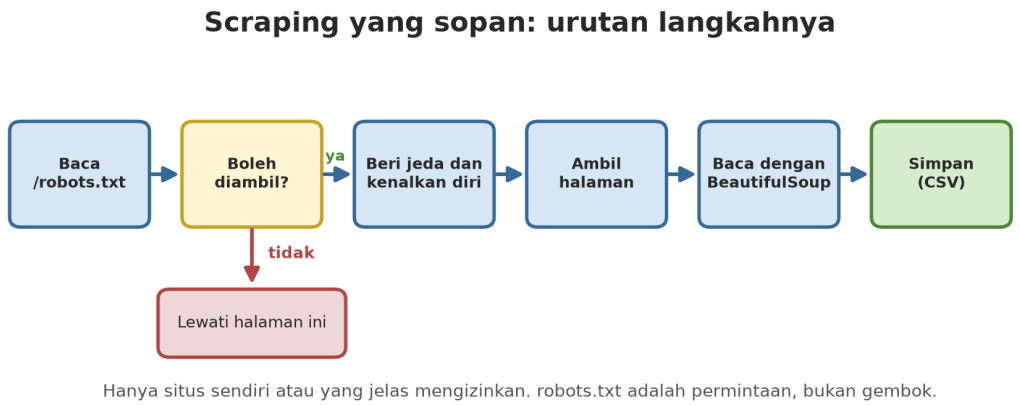
Di bab ini, Anda akan:

- Menjalankan program lain dari Python dengan **subprocess**
- Mengotomasi tugas berkas dan pemeriksaan layanan (infrastruktur)
- Memahami cara menjadwalkan skrip di Windows (Task Scheduler) dan Linux/macOS (cron)
- Membaca halaman web dengan **BeautifulSoup** dan **Scrapy**, dengan etika yang benar
- Menyusun otomasi laporan mingguan: proyek bersambung Seri I5

Semua skrip bab ini ada di folder `kode/bab41/`. Di versi web, Anda juga bisa mengunduh semuanya sekaligus (zip).

Di Bab 36 sampai 40, Anda membuat aplikasi web dan peta. Sekarang bayangkan pekerjaan yang berulang: setiap Senin pagi seseorang mengumpulkan angka dari tiap unit usaha, merapikannya, lalu menulis laporan. Itu seperti petugas yang tiap pekan keliling membawa buku catatan. Python bisa menjadi **petugas yang tidak pernah lupa dan tidak pernah bosan**. Pegangan sederhananya: **bila Anda mengerjakan sesuatu yang sama untuk ketiga kalinya, tulislah skripnya.**

Bab ini punya dua bagian besar. Pertama, **automasi**: menjalankan program, merapikan berkas, memeriksa layanan, dan menjadwalkan semuanya. Kedua, **web scraping**: membaca data dari halaman web. Untuk yang kedua ada aturan keras di buku ini: kita hanya berlatih pada **situs buatan sendiri yang berjalan di komputer sendiri**, dan kita membahas etikanya lebih dulu.



Gambar 0.1: Scraping yang sopan: urutan langkahnya.

subprocess: memanggil program lain

Python punya modul **subprocess** untuk menjalankan program lain (disebut **proses anak**) lalu menunggu hasilnya. Ini seperti menyuruh asisten mengerjakan sesuatu dan melaporkan hasilnya.

```
# Bab 41 - subprocess: menjalankan program lain dari Python
import subprocess
import sys

# sys.executable = jalur Python yang sedang kita pakai; "-c" = jalankan kode pendek
```

```
hasil = subprocess.run(
    [sys.executable, "-c", "print('halo dari proses anak')"],
    capture_output=True, text=True, encoding="utf-8",
)
print("kode keluar:", hasil.returncode)
print("stdout     :", hasil.stdout.strip())
print("stderr kosong:", hasil.stderr == "")

# Argumen dikirim sebagai LIST, satu butir per kata (tanpa shell)
perintah = [sys.executable, "-c", "import sys; print(' + '.join(sys.argv[1:]))",
"padi", "jagung"]
hasil = subprocess.run(perintah, capture_output=True, text=True, encoding="utf-8")
print("argumen sampai :", hasil.stdout.strip())
```

```
kode keluar: 0
stdout     : halo dari proses anak
stderr kosong: True
argumen sampai : padi + jagung
```

- `sys.executable` adalah jalur Python yang sedang dipakai, jadi proses anak memakai Python yang sama. `-c` berarti "jalankan kode pendek ini".
- `capture_output=True` menangkap apa yang dicetak anak (`stdout`) dan keluhannya (`stderr`). `text=True` membuat hasilnya berupa teks, bukan bytes.
- **Kode keluar** (`returncode`) 0 berarti sukses. Angka lain berarti gagal.
- Argumen dikirim sebagai **list**, satu butir per kata. Menurut dokumentasi resmi Python, dengan cara ini modul sendiri yang mengurus tanda kutip dan spasi pada nama berkas.

> ⚠ AWAS
>
> Jangan tergoda memakai `shell=True` dengan teks yang berasal dari pengguna atau dari berkas. Dokumentasi Python memperingatkan bahwa dengan `shell=True` Andalah yang harus mengamankan semua spasi dan tanda khusus, bila tidak, orang bisa menyisipkan perintah lain (**shell injection**). Dengan list, tanda khusus dikirim apa adanya kepada anak dan aman.

Program anak bisa gagal dengan tiga cara. Ketiganya bisa ditangani:

```
# Bab 41 - subprocess: tiga cara gagal dan cara menanganinya
import subprocess
import sys

# 1. Program anak berakhir dengan galat; check=True mengubahnya menjadi pengecualian
try:
    subprocess.run([sys.executable, "-c", "raise SystemExit(3)"], check=True)
except subprocess.CalledProcessError as e:
    print("1. kode keluar bukan 0:", e.returncode)

# 2. Program anak terlalu lama; timeout memutus penantian
try:
    subprocess.run([sys.executable, "-c", "import time; time.sleep(5)"], timeout=0.5)
except subprocess.TimeoutExpired:
    print("2. melewati batas waktu 0.5 detik, proses anak dihentikan")

# 3. Programnya tidak ada sama sekali
try:
    subprocess.run(["program-yang-tidak-ada-123"])
except FileNotFoundError:
    print("3. program tidak ditemukan")
```

```
1. kode keluar bukan 0: 3
2. melewati batas waktu 0.5 detik, proses anak dihentikan
3. program tidak ditemukan
```

`check=True` mengubah kode keluar bukan-0 menjadi pengecualian `CalledProcessError`. `timeout` menghentikan proses anak yang terlalu lama dan memunculkan `TimeoutExpired`. Program yang tidak ada menghasilkan `FileNotFoundError`. Skrip otomasi yang baik **selalu memikirkan ketiganya**, karena tidak ada orang yang melihat layar saat skrip berjalan pukul tiga pagi.

Otomasi berkas: merapikan folder

Tugas klasik: folder unduhan menumpuk berbagai jenis berkas. Skrip ini memindahkan tiap berkas ke folder menurut ekstensinya.

```
# Bab 41 - Otomasi tugas harian: merapikan folder unduhan berdasarkan jenis berkas
import shutil
from pathlib import Path

kerja = Path(__file__).parent / "_tmp" / "unduhan"
shutil.rmtree(kerja, ignore_errors=True)
kerja.mkdir(parents=True)
for nama in ["panen_a.csv", "panen_b.csv", "peta_kolam.png", "laporan_mei.pdf",
"catatan.txt", "foto_kebun.png"]:
    (kerja / nama).write_text("isi contoh", encoding="utf-8")

PETA_FOLDER = {".csv": "data", ".png": "gambar", ".pdf": "dokumen"}

dipindah = 0
for berkas in sorted(kerja.iterdir()):
    if not berkas.is_file():
        continue
    tujuan = kerja / PETA_FOLDER.get(berkas.suffix, "lainnya")
    tujuan.mkdir(exist_ok=True)
    shutil.move(str(berkas), str(tujuan / berkas.name))
    dipindah += 1

print("berkas dipindah:", dipindah)
for folder in sorted(p for p in kerja.iterdir() if p.is_dir()):
    print(f"{folder.name:8}", sorted(p.name for p in folder.iterdir()))
```

```
berkas dipindah: 6
data     ['panen_a.csv', 'panen_b.csv']
dokumen  ['laporan_mei.pdf']
gambar   ['foto_kebun.png', 'peta_kolam.png']
lainnya  ['catatan.txt']
```

Perhatikan dua kebiasaan baik. Pertama, skrip bekerja di folder `_tmp` miliknya sendiri, bukan di folder asli Anda, supaya aman dicoba. Kedua, `PETA_FOLDER.get(berkas.suffix, "lainnya")` menyediakan folder cadangan untuk jenis yang tidak dikenal (`catatan.txt` masuk ke `lainnya`), jadi tidak ada berkas yang tertinggal.

> TIPS
>
> Sebelum memindah atau menghapus berkas secara otomatis, jalankan dulu versi yang **hanya mencetak** apa yang akan dilakukan. Setelah daftarnya benar, barulah ganti dengan perintah yang sungguhan.

Otomasi infrastruktur: memeriksa layanan

Infrastruktur adalah semua "mesin di belakang layar" yang menjalankan aplikasi Anda: server, layanan web, basis data, ruang disk. Otomasi infrastruktur berarti membuat skrip yang memeriksa dan merawatnya. Contoh paling dasar adalah **pemeriksa kesehatan** (*health check*): minta halaman utama, lalu lihat apakah jawabannya sehat.

```
# Bab 41 - Otomasi infrastruktur: pemeriksa kesehatan layanan (health check)
import shutil
import urllib.error
import urllib.request

from situs_lokal import SitusLokal


def periksa(alamat):
    """Mengembalikan teks status: 'sehat', 'galat HTTP <kode>', atau 'tidak terhubung'."""
    try:
        with urllib.request.urlopen(alamat, timeout=2) as jawaban:
            return "sehat" if jawaban.status == 200 else f"status {jawaban.status}"
    except urllib.error.HTTPError as e:
        return f"galat HTTP {e.code}"
    except urllib.error.URLError:
        return "tidak terhubung"


with SitusLokal() as situs:
    alamat = situs.url
    print("halaman utama     :", periksa(alamat + "/"))
    print("halaman tak ada   :", periksa(alamat + "/tidak-ada"))

# Setelah blok `with`, server dimatikan: pemeriksa harus menyadarinya
print("setelah dimatikan :", periksa(alamat + "/"))

# Cek lain yang sering diotomasi: sisa ruang disk (hanya dicetak benar/salah supaya stabil)
print("disk masih ada ruang:", shutil.disk_usage(".").free > 100 * 1024 * 1024)
```

```
halaman utama     : sehat
halaman tak ada   : galat HTTP 404
setelah dimatikan : tidak terhubung
disk masih ada ruang: True
```

Server yang diperiksa di sini adalah server kecil buatan kita sendiri di komputer ini (modul bantu `situs_lokal.py`, lihat penjelasannya di bagian scraping). Tiga hasil berbeda patut dibedakan: **sehat** (kode 200), **galat HTTP** (server hidup tetapi halamannya tidak ada, 404), dan **tidak terhubung** (server mati). Dalam praktik, skrip seperti ini dijadwalkan tiap beberapa menit dan mengirim peringatan bila hasilnya bukan "sehat". Mengirim peringatan (email atau pesan) tidak kita praktikkan di sini karena butuh layanan di luar komputer ini.

Menjadwalkan skrip

Skrip otomasi baru berguna bila berjalan **sendiri pada waktunya**. Python tidak menjadwalkan dirinya: tugas itu diserahkan ke penjadwal milik sistem operasi.

Sistem	Penjadwal	Cara
Windows	**Task Scheduler**	perintah `schtasks /create`, atau aplikasi bergambar Task Scheduler

Sistem	Penjadwal	Cara
Linux dan macOS	**cron**	baris di berkas **crontab** (`crontab -e`)

```
# Bab 41 - Menjadwalkan skrip: perintahnya HANYA dicetak, tidak dijalankan
from pathlib import Path

# Windows Task Scheduler: satu tugas = satu program, jadi kita bungkus dengan berkas .bat
bat = "\n".join([
    "@echo off",
    r"C:\proyek\.venv\Scripts\python.exe C:\proyek\laporan_mingguan.py",
    "",
])
tulis = Path(__file__).parent / "_tmp" / "jalankan_laporan.bat"
tulis.parent.mkdir(exist_ok=True)
tulis.write_text(bat, encoding="ascii")
print("isi jalankan_laporan.bat:")
print(bat)

print("Perintah Windows (jalankan sendiri di terminal Anda):")
print(r"schtasks /create /tn LaporanMingguan /tr C:\proyek\jalankan_laporan.bat /sc
weekly /d MON /st 07:00")

# cron (Linux/macOS): lima kolom waktu, lalu perintah
cron  =  "0  7  *  *  1  /home/anda/proyek/.venv/bin/python  /home/anda/proyek/
laporan_mingguan.py"
print("\nBaris crontab (tempel lewat `crontab -e`):")
print(cron)

BATAS = [("menit", 0, 59), ("jam", 0, 23), ("tanggal", 1, 31), ("bulan", 1, 12),
("hari", 0, 6)]


def periksa_cron(baris):
    """Pemeriksa sederhana: hanya '*' atau satu angka per kolom, dan angkanya dalam
rentang."""
    kolom = baris.split()[:5]
    for isi, (nama, kecil, besar) in zip(kolom, BATAS):
        if isi != "*" and not (isi.isdigit() and kecil <= int(isi) <= besar):
            return f"kolom {nama} salah: {isi}"
    return "lima kolom waktu masuk akal"


print("\nPemeriksaan:", periksa_cron(cron))
print("Contoh salah:", periksa_cron("0 25 * * 1 perintah"))
```

```
isi jalankan_laporan.bat:
@echo off
C:\proyek\.venv\Scripts\python.exe C:\proyek\laporan_mingguan.py

Perintah Windows (jalankan sendiri di terminal Anda):
schtasks /create /tn LaporanMingguan /tr C:\proyek\jalankan_laporan.bat /sc weekly /d
MON /st 07:00

Baris crontab (tempel lewat `crontab -e`):
0 7 * * 1 /home/anda/proyek/.venv/bin/python /home/anda/proyek/laporan_mingguan.py

Pemeriksaan: lima kolom waktu masuk akal
Contoh salah: kolom jam salah: 25
```

Kode ini hanya mencetak perintahnya. Perintah penjadwalan tidak dijalankan di buku ini karena mengubah pengaturan komputer Anda. Anda menjalankannya sendiri bila siap. Keterangannya, menurut dokumentasi resmi:

- **Windows (`schtasks`)**: `/tn` memberi nama tugas, `/tr` menentukan program yang dijalankan, `/sc weekly` berarti mingguan, `/d MON` hari Senin, `/st 07:00` jam mulai (format 24 jam). Satu tugas hanya menjalankan **satu program**, sehingga dokumentasi menyarankan berkas batch (`.bat`) untuk menjalankan beberapa langkah. Itu sebabnya kita membungkus Python dengan `jalankan_laporan.bat`.
- **cron**: lima kolom waktu berurutan: menit (0-59), jam (0-23), tanggal (1-31), bulan (1-12), hari dalam pekan (0-6, dengan 0 = Minggu). Jadi `0 7 * * 1` berarti **pukul 07.00 setiap Senin**. Tanda `*` berarti "semua nilai".

> **i** INGAT
>
> Tiga kebiasaan penjadwalan yang menyelamatkan Anda: (1) pakai **jalur lengkap** ke Python dan ke skrip, karena penjadwal tidak berjalan dari folder Anda; (2) selalu **tulis log** ke berkas, karena tidak ada layar yang bisa dibaca; (3) **coba dulu dengan tangan** sebelum dijadwalkan.

Web scraping: etika dulu, kode kemudian

Web scraping adalah membaca halaman web dengan program, lalu mengambil bagian yang berguna (misalnya isi tabel). Tekniknya mudah. Yang sulit, dan yang lebih penting, adalah **bersikap benar**. Sebelum menulis satu baris pun, pegang aturan ini:

Pertanyaan	Yang harus Anda lakukan
Adakah cara resmi?	Cari dulu **API** atau unduhan data resmi. Itu selalu lebih baik daripada scraping.
Apa kata `robots.txt`?	Baca `/robots.txt` di situs itu dan patuhi bagian yang melarang.
Apa kata syarat penggunaan?	Baca syarat penggunaan (*terms of service*) situs. Bila melarang scraping, jangan lakukan.
Seberapa sering saya meminta?	Beri **jeda** antar permintaan dan jangan membanjiri server (batas laju).
Siapa saya bagi server?	Perkenalkan diri lewat **User-Agent** yang jujur dan bisa dihubungi.
Datanya apa?	Hati-hati dengan **data pribadi** dan hak cipta. Data boleh dibaca belum tentu boleh disebarkan.

`robots.txt` dijelaskan dalam standar **RFC 9309** (Robots Exclusion Protocol). Standar itu menegaskan bahwa aturan di dalamnya **bukan bentuk izin akses**: ia adalah permintaan sopan yang dipatuhi oleh perayap yang baik, bukan gembok. Artinya, tidak ada yang menghentikan program Anda melanggarnya. Yang menghentikan adalah **sikap Anda**. Standar itu juga tidak mendefinisikan `Crawl-delay`; itu kebiasaan tambahan yang dipahami sebagian perayap.

Aturan buku ini: semua latihan scraping di sini berjalan terhadap halaman HTML **buatan sendiri** yang disajikan server di komputer Anda (`127.0.0.1`, alamat yang hanya bisa dijangkau komputer ini). Kita tidak mengambil halaman situs nyata mana pun. Itu sengaja: kita bisa berlatih penuh tanpa membebani siapa pun.

Situs latihannya dibuat oleh modul bantu `situs_lokal.py` (ada di folder skrip): satu halaman utama berisi daftar sektor, lima halaman sektor yang masing-masing memuat sebuah tabel unit usaha, satu halaman "internal" yang dilarang oleh `robots.txt`, dan berkas `robots.txt` itu sendiri. Datanya adalah data tiruan unit usaha kita, **semua angka fiktif**.

BeautifulSoup: membaca HTML

BeautifulSoup (paket `beautifulsoup4`) mengubah teks HTML menjadi pohon yang bisa ditelusuri. Pemilah (*parser*) bawaan Python bernama `html.parser`; ada juga `lxml` dan `html5lib` yang dipasang terpisah.

```
# Bab 41 - BeautifulSoup: membaca halaman HTML (situs buatan sendiri di komputer sendiri)
import urllib.request

from bs4 import BeautifulSoup

from situs_lokal import SitusLokal

with SitusLokal() as situs:
    with urllib.request.urlopen(situs.url + "/sektor/kolam.html", timeout=5) as r:
        html = r.read().decode("utf-8")

print("--- 8 baris pertama HTML ---")
print("\n".join(html.splitlines()[:8]))
print("...")

sup = BeautifulSoup(html, "html.parser")
print("\njudul halaman :", sup.title.get_text())
print("judul utama   :", sup.h1.get_text())

# Selektor CSS: cari semua baris tabel yang punya atribut data-id
total = 0.0
for tr in sup.select("table#unit tbody tr[data-id]"):
    sel = [td.get_text(strip=True) for td in tr.find_all("td")]
    print(tr["data-id"], sel, "lat =", tr["data-lat"])
    total += float(tr.select_one("td.nilai").get_text())
print("total nilai   :", total)

# Jebakan: find() mengembalikan None bila tidak ketemu
tabel = sup.find("table", id="tidak-ada")
print("\ntabel yang dicari:", tabel)
try:
    tabel.find_all("tr")
except AttributeError as e:
    print("galat:", e)
```

```
--- 8 baris pertama HTML ---
<!doctype html>
<html lang="id">
<head><meta charset="utf-8"><title>Sektor kolam</title></head>
<body>
  <h1>Sektor kolam</h1>
  <table id="unit">
    <thead><tr><th>Unit</th><th>Jenis</th><th>Nilai</th><th>Satuan</th></tr></thead>
    <tbody>
...

judul halaman : Sektor kolam
judul utama   : Sektor kolam
L1 ['Kolam A', 'nila', '432.4', 'kg'] lat = -6.644
L2 ['Kolam B', 'lele', '380.0', 'kg'] lat = -6.6452
L3 ['Kolam C', 'nila', '295.5', 'kg'] lat = -6.6466
total nilai   : 1107.9

tabel yang dicari: None
galat: 'NoneType' object has no attribute 'find_all'
```

Telusuri hasilnya:

- Delapan baris pertama HTML menunjukkan bentuk halaman: ada `<table id="unit">` dengan kepala tabel dan badan tabel.
- `select("table#unit tbody tr[data-id]")` memakai **selektor CSS**: "baris di dalam badan tabel `unit` yang punya atribut `data-id`". `select()` mengembalikan list semua yang cocok; `select_one()` hanya yang pertama.
- `get_text(strip=True)` mengambil teks tanpa spasi berlebih. Atribut seperti `data-lat` dibaca seperti kamus: `tr["data-lat"]`.
- Total tiga kolam adalah 432,4 + 380,0 + 295,5 = **1107,9** kg, cocok dengan keluaran.
- **Jebakan:** bila elemen yang dicari tidak ada, `find()` mengembalikan `None` (sedangkan `find_all()` mengembalikan list kosong). Memanggil metode pada `None` menghasilkan `AttributeError`. Itulah sebabnya kode yang baik memeriksa `None` dan **berteriak jelas** bila halaman berubah bentuk.

robots.txt di kode

Python punya pembaca `robots.txt` bawaan: `urllib.robotparser`.

```
# Bab 41 - robots.txt: membaca aturan situs sebelum mengambil halaman
import urllib.request
import urllib.robotparser

from situs_lokal import SitusLokal

AGEN = "BukuPythonBot/1.0"

with SitusLokal() as situs:
    with urllib.request.urlopen(situs.url + "/robots.txt", timeout=5) as r:
        isi = r.read().decode("utf-8")
    print("isi robots.txt:")
    print(isi)

    robot = urllib.robotparser.RobotFileParser()
    robot.parse(isi.splitlines())
    for path in ["/", "/sektor/kolam.html", "/privat/catatan.html"]:
        boleh = robot.can_fetch(AGEN, situs.url + path)
        print(f"{path:24} boleh diambil? {boleh}")
    print("Crawl-delay:", robot.crawl_delay(AGEN), "detik")

    # robots.txt hanyalah permintaan, bukan gembok: server tetap menjawab bila kita
memaksa
    with urllib.request.urlopen(situs.url + "/privat/catatan.html", timeout=5) as r:
          print("\nkalau dipaksa, server menjawab:", r.status, "(tetap TIDAK boleh
kita lakukan)")
```

```
isi robots.txt:
User-agent: *
Disallow: /privat/
Crawl-delay: 1

/                        boleh diambil? True
/sektor/kolam.html       boleh diambil? True
/privat/catatan.html     boleh diambil? False
Crawl-delay: 1 detik

kalau dipaksa, server menjawab: 200 (tetap TIDAK boleh kita lakukan)
```

`can_fetch(agen, alamat)` menjawab apakah agen kita boleh mengambil alamat itu. Halaman `/privat/` dilarang (`False`). `crawl_delay()` mengembalikan jeda yang diminta situs (1 detik), atau `None` bila tidak ada. Perhatikan baris terakhir: **server tetap menjawab 200 bila kita memaksa**. Itu persis pesan RFC 9309 tadi: `robots.txt` bukan gembok, jadi kepatuhan sepenuhnya di tangan kita.

Scraper yang sopan

Kini kita gabungkan semuanya menjadi satu kelas: membaca `robots.txt` dulu, memberi jeda, dan memperkenalkan diri.

```
# Bab 41 - Modul bantu: pengambil halaman yang sopan (patuh robots.txt, memberi jeda,
memperkenalkan diri)
import time
import urllib.error
import urllib.request
import urllib.robotparser

from bs4 import BeautifulSoup

AGEN = "BukuPythonBot/1.0 (contoh belajar; situs lokal sendiri)"


def baca_tabel(html):
    """Membaca tabel unit dari satu halaman sektor menjadi daftar dict."""
    sup = BeautifulSoup(html, "html.parser")
    tabel = sup.find("table", id="unit")
    if tabel is None:                          # halaman berubah bentuk: jangan diam-
diam salah
        raise ValueError("tabel #unit tidak ditemukan")
    hasil = []
    for tr in tabel.select("tbody tr"):
        sel = [td.get_text(strip=True) for td in tr.find_all("td")]
        hasil.append({
            "id": tr["data-id"], "unit": sel[0], "jenis": sel[1],
            "lat": float(tr["data-lat"]), "lon": float(tr["data-lon"]),
            "nilai": float(sel[2]), "satuan": sel[3],
        })
    return hasil


class PengambilSopan:
    def __init__(self, dasar, agen=AGEN, jeda_bawaan=1.0, batas_waktu=5):
        self.dasar, self.agen, self.batas_waktu = dasar, agen, batas_waktu
       self.ditolak = []                # alamat yang tidak diambil karena robots.txt
        self._terakhir = 0.0
        self.robot = urllib.robotparser.RobotFileParser()
        teks = self._unduh("/robots.txt")
        self.robot.parse(teks.splitlines() if teks else [])
       # Jeda: ikuti Crawl-delay bila ada (itu kebiasaan, bukan bagian standar), kalau
tidak pakai bawaan
        self.jeda = self.robot.crawl_delay(agen) or jeda_bawaan

    def _unduh(self, path):
        permintaan = urllib.request.Request(self.dasar + path, headers={"User-Agent":
self.agen})
        try:
            with urllib.request.urlopen(permintaan, timeout=self.batas_waktu) as r:
                return r.read().decode("utf-8")
        except urllib.error.HTTPError:
            return ""

    def ambil(self, path):
        """Mengembalikan HTML, atau None bila robots.txt melarang."""
        if not self.robot.can_fetch(self.agen, self.dasar + path):
            self.ditolak.append(path)
            return None
```

```
        tunggu = self.jeda - (time.monotonic() - self._terakhir)
        if tunggu > 0:
            time.sleep(tunggu)
        try:
            return self._unduh(path)
        finally:
            self._terakhir = time.monotonic()
```

Hal penting di kelas ini:

- **User-Agent** jujur: `BukuPythonBot/1.0 (contoh belajar; situs lokal sendiri)`. Pada situs nyata, sertakan cara menghubungi Anda.
- `self.jeda`: memakai `Crawl-delay` dari situs bila ada, bila tidak, 1 detik.
- `ambil()` **menolak** alamat yang dilarang `robots.txt` (dicatat di `ditolak`) dan menunggu sampai jeda terpenuhi sebelum meminta lagi.
- `baca_tabel()` melempar `ValueError` bila tabel tidak ditemukan.

Skrip yang memakainya:

```
# Bab 41 - Scraper sopan: patuh robots.txt, memberi jeda, memperkenalkan diri
import csv
from pathlib import Path

from bs4 import BeautifulSoup

from data_unit import baca_unit
from pengambil import PengambilSopan, baca_tabel
from situs_lokal import SitusLokal

with SitusLokal() as situs:
    bot = PengambilSopan(situs.url)
    print("jeda yang dipakai:", bot.jeda, "detik")

    indeks = BeautifulSoup(bot.ambil("/"), "html.parser")
    tautan = [a["href"] for a in indeks.select("#menu a")]
    print("tautan di halaman utama:", len(tautan))

    semua = []
    for path in tautan:
        html = bot.ambil(path)
        if html is None:
            print("dilewati (robots.txt melarang):", path)
            continue
        semua += baca_tabel(html)
    catatan = list(situs.catatan)

keluar = Path(__file__).parent / "_tmp" / "unit_hasil_scrape.csv"
keluar.parent.mkdir(exist_ok=True)
with open(keluar, "w", encoding="utf-8", newline="") as f:
    tulis = csv.DictWriter(f, fieldnames=list(semua[0]))
    tulis.writeheader()
    tulis.writerows(semua)

print("baris hasil scraping:", len(semua))
print("sama dengan data sumber:", semua == [{k: u[k] for k in semua[0]} for u in
baca_unit()])
print("permintaan yang masuk ke server:", len(catatan))
print("halaman terlarang yang diminta :", [p for p, _ in catatan if p.startswith("/
privat")])
print("pengenal diri (User-Agent)     :", sorted({ua for _, ua in catatan}))
```

```
jeda yang dipakai: 1 detik
tautan di halaman utama: 6
dilewati (robots.txt melarang): /privat/catatan.html
baris hasil scraping: 15
sama dengan data sumber: True
permintaan yang masuk ke server: 7
halaman terlarang yang diminta : []
pengenal diri (User-Agent)      : ['BukuPythonBot/1.0 (contoh belajar; situs lokal
sendiri)']
```

Mari periksa angkanya. Halaman utama punya **6 tautan**: lima sektor dan satu halaman internal. Yang internal **dilewati** karena dilarang. Hasilnya **15 baris**, sama persis dengan data sumber (`True`). Server menerima **7 permintaan**: 1 untuk `robots.txt`, 1 untuk halaman utama, dan 5 untuk halaman sektor. Daftar halaman terlarang yang diminta kosong, dan pengenal diri kita tercatat jujur di server.

> BAGIAN TEKNIS (boleh dilewati)
>
> Skrip ini menunggu 1 detik sebelum tiap permintaan, jadi berjalan beberapa detik. Itu disengaja dan bukan kelambatan: itulah batas laju. Pada situs nyata, jeda yang wajar sering lebih lama, dan scraping skala besar sebaiknya diizinkan tertulis oleh pemilik situs.

Scrapy: kerangka kerja untuk scraping besar

Scrapy adalah kerangka kerja (*framework*) lengkap untuk perayapan: mengelola antrean alamat, jeda, konkurensi, kepatuhan `robots.txt`, dan penyimpanan hasil. Bila BeautifulSoup adalah **pisau**, Scrapy adalah **pabrik kecil**. Untuk satu-dua halaman, BeautifulSoup cukup; untuk ratusan halaman yang rutin, Scrapy lebih rapi.

	BeautifulSoup	Scrapy
Sifat	pustaka pembaca HTML	kerangka kerja perayap penuh
Mengambil halaman?	tidak (pakai `urllib` atau `httpx`)	ya
Jeda dan `robots.txt`	Anda yang menulis	pengaturan (`DOWNLOAD_DELAY`, `ROBOTSTXT_OBEY`)
Cocok untuk	tugas kecil	proyek perayapan besar dan berulang

Scrapy 2.19.0 terpasang di komputer penulis lewat `alat/pasang.py` untuk Python 3.14, dan skrip di bawah berhasil berjalan. (Di komputer Anda, pemasangannya `pip install scrapy`.) Berikut sebuah **spider** (perayap) kecil:

```
# Bab 41 - Scrapy: kerangka kerja scraping, dijalankan terhadap situs lokal buatan
sendiri
import json
from pathlib import Path

import scrapy
from scrapy.crawler import CrawlerProcess

from situs_lokal import SitusLokal

KELUAR = Path(__file__).parent / "_tmp" / "scrapy_unit.json"
KELUAR.parent.mkdir(exist_ok=True)


class LabaUnit(scrapy.Spider):
    name = "laba_unit"
```

```
    def __init__(self, dasar, **kw):
        super().__init__(**kw)
        self.dasar = dasar

    async def start(self):
        yield scrapy.Request(self.dasar + "/")

    def parse(self, response):
        # menu berisi 5 tautan sektor + 1 tautan internal yang dilarang robots.txt
        for a in response.css("#menu a"):
            yield response.follow(a, callback=self.parse_sektor)

    def parse_sektor(self, response):
        for tr in response.css("table#unit tbody tr"):
            yield {
                "id": tr.attrib["data-id"],
                "unit": tr.css("td:nth-child(1)::text").get(),
                "nilai": float(tr.css("td.nilai::text").get()),
            }


with SitusLokal() as situs:
    proses = CrawlerProcess(settings={
        "USER_AGENT": "BukuPythonBot/1.0 (contoh belajar; situs lokal sendiri)",
        "ROBOTSTXT_OBEY": True,
        "DOWNLOAD_DELAY": 1,
        "CONCURRENT_REQUESTS_PER_DOMAIN": 1,
        "LOG_LEVEL": "ERROR",
        "FEEDS": {str(KELUAR): {"format": "json", "overwrite": True}},
    })
    perayap = proses.create_crawler(LabaUnit)
    proses.crawl(perayap, dasar=situs.url)
    proses.start()
    statistik = perayap.stats.get_stats()

item = json.loads(KELUAR.read_text(encoding="utf-8"))
print("item terkumpul          :", len(item))
print("contoh item pertama     :", sorted(item, key=lambda x: x["id"])[0])
print("dilarang robots.txt     :", statistik.get("robotstxt/forbidden"))
print("halaman      berhasil      (200)           :",      statistik.get("downloader/
response_status_count/200"))
```

```
item terkumpul          : 15
contoh item pertama     : {'id': 'D1', 'unit': 'Kandang 1', 'nilai': 350.0}
dilarang robots.txt     : 1
halaman berhasil (200)  : 7
```

Cara membacanya:

- `start()` memberi alamat awal. `parse()` menerima halaman utama dan mengikuti tiap tautan di menu dengan `response.follow`. `parse_sektor()` mengeluarkan satu **item** (kamus) untuk tiap baris tabel.
- Pengaturan di `CrawlerProcess` memuat etika kita: `ROBOTSTXT_OBEY` menyalakan kepatuhan pada `robots.txt` (dokumentasi Scrapy mencatat bahwa proyek hasil `scrapy startproject` menyalakannya, tetapi kita menuliskannya eksplisit), `DOWNLOAD_DELAY` memberi jeda 1 detik, `CONCURRENT_REQUESTS_PER_DOMAIN` membatasi permintaan serentak ke satu situs, dan `USER_AGENT` memperkenalkan diri. Dokumentasi Scrapy menganjurkan User-Agent yang menyebut proyek dan cara menghubungi Anda.
- Hasilnya: **15 item**, **1** permintaan ditolak oleh `robots.txt` (halaman internal), dan **7** halaman berstatus 200: `robots.txt`, halaman utama, dan lima halaman sektor. Angka ini sama dengan hitungan scraper buatan sendiri tadi.

> **INGAT**
>
> Scrapy punya **jeda acak**: secara bawaan jedanya digoyang sekitar setengah sampai satu setengah kali nilai `DOWNLOAD_DELAY`, supaya tidak terlalu seragam. Karena itu waktu proses tidak dicetak di buku ini, hanya jumlah yang stabil.

Proyek Bersambung I5: otomasi laporan mingguan

Sekarang semua bahan bergabung. Skrip proyek ini mengambil data semua sektor dari situs lokal dengan pengambil yang sopan, merangkum per sektor, membandingkannya dengan rata-rata minggu lalu, menulis laporan Markdown, dan mencatat log.

```
# Bab 41 - Proyek bersambung I5: otomasi laporan mingguan (angka FIKTIF)
import logging
from pathlib import Path

from data_unit import ringkas_per_sektor
from pengambil import PengambilSopan, baca_tabel
from situs_lokal import SitusLokal

KELUAR = Path(__file__).parent / "_tmp"
KELUAR.mkdir(exist_ok=True)
logging.basicConfig(filename=KELUAR / "laporan.log", filemode="w", level=logging.INFO,
                    format="%(levelname)s %(message)s")

# Rata-rata minggu lalu (angka contoh) untuk dibandingkan
MINGGU_LALU = {"hutan": 34.8, "kebun": 250.0, "sawah": 1100.0, "kolam": 360.0,
"kandang": 380.0}


def kumpulkan(dasar):
    bot = PengambilSopan(dasar)
    unit = []
    for sektor in MINGGU_LALU:
        html = bot.ambil(f"/sektor/{sektor}.html")
        if html is None:
            logging.warning("dilewati: %s", sektor)
            continue
        baris = [dict(u, sektor=sektor) for u in baca_tabel(html)]
        unit += baris
        logging.info("sektor %s: %d unit", sektor, len(baris))
    return unit


def susun_laporan(unit):
    baris = ["# Laporan mingguan (data fiktif)", "",
                  "| Sektor | Unit | Rata-rata | Satuan | Beda dari minggu lalu
|", "|---|---|---|---|---|"]
    for sektor, r in ringkas_per_sektor(unit).items():
        beda = (r["rata"] - MINGGU_LALU[sektor]) / MINGGU_LALU[sektor] * 100
        baris.append(f"| {sektor} | {r['jumlah']} | {r['rata']:.1f} | {r['satuan']} |
{beda:+.1f}% |")
    return "\n".join(baris) + "\n"


def main():
    with SitusLokal() as situs:
        unit = kumpulkan(situs.url)
    teks = susun_laporan(unit)
```

```
    (KELUAR / "laporan_mingguan.md").write_text(teks, encoding="utf-8")
    logging.info("laporan selesai: %d unit", len(unit))
    return teks


if __name__ == "__main__":
    print(main())
    print("baris log:", len((KELUAR / "laporan.log").read_text().splitlines()))
```

```
# Laporan mingguan (data fiktif)

| Sektor | Unit | Rata-rata | Satuan | Beda dari minggu lalu |
|---|---|---|---|---|
| hutan | 3 | 35.2 | cm | +1.2% |
| kebun | 4 | 257.5 | kg | +3.0% |
| sawah | 2 | 1115.2 | kg | +1.4% |
| kolam | 3 | 369.3 | kg | +2.6% |
| kandang | 3 | 383.8 | kg | +1.0% |

baris log: 6
```

Telusuri satu baris: sektor **kolam** punya 3 unit dengan rata-rata (432,4 + 380,0 + 295,5) / 3 = **369,3** kg. Rata-rata minggu lalu (angka contoh) 360,0, jadi (369,3 − 360,0) / 360,0 = **+2,6%**. Skrip juga menulis `laporan.log` yang berisi 6 baris (5 sektor dan 1 baris "laporan selesai"), bukti bahwa ia bisa dipantau tanpa layar.

Struktur skrip sengaja dibuat agar bisa dijadwalkan: semua pekerjaan ada dalam fungsi `main()`, dan bagian `if __name__ == "__main__":` hanya memanggilnya. Dengan bungkus `.bat` atau baris cron dari bagian penjadwalan, laporan ini akan tersusun sendiri tiap Senin pukul 07.00. Angka rata-rata sektor hutan (cm) dan lainnya (kg) tidak boleh dijumlahkan lintas sektor; itulah sebabnya laporan dibuat per sektor.

Latihan

1. Ubah `01_subprocess_dasar.py` supaya proses anak mencetak penjumlahan dua angka yang Anda kirim sebagai argumen, lalu baca hasilnya di induk.
2. Tambahkan jenis berkas `.xlsx` (folder `lembar`) pada `03_otomasi_berkas.py`. Berapa berkas yang masuk `lainnya` sekarang?
3. Pada `07_robots_etika.py`, ubah `Disallow` di modul `situs_lokal.py` (salinan Anda sendiri) agar `/sektor/` juga dilarang. Pastikan `can_fetch` menjadi `False` untuk halaman sektor.
4. **Tantangan:** tambahkan ke laporan mingguan satu kolom "unit tertinggi" (nama unit dengan nilai terbesar di tiap sektor), lalu tulis baris `schtasks` atau cron untuk menjadwalkannya hari Jumat pukul 16.00 (cukup dicetak, jangan dijalankan).

Ringkasan Bab

- **`subprocess.run([...], capture_output=True, text=True)`** menjalankan program lain; kirim argumen sebagai **list**, hindari `shell=True` dengan teks tak tepercaya.
- Tangani tiga kegagalan: kode keluar (`check=True`), waktu habis (`timeout`), program tidak ada (`FileNotFoundError`).
- Penjadwalan diserahkan ke **Task Scheduler** (Windows) atau **cron** (Linux/macOS); pakai jalur lengkap, tulis log, uji dengan tangan dulu.
- **Scraping**: cari API resmi dulu; patuhi `robots.txt` dan syarat penggunaan; beri jeda; kenalkan diri; hati-hati data pribadi. `robots.txt` adalah permintaan, bukan gembok.
- **BeautifulSoup** membaca HTML (`select`, `get_text`); `find()` bisa mengembalikan `None`. **Scrapy** mengurus perayapan besar lewat pengaturan.
- Buku ini hanya melatih scraping pada **situs buatan sendiri** di komputer sendiri.

Selanjutnya, di Bab 42: antarmuka. Skrip tidak harus dipakai lewat berkas Python: Anda akan membuat perintah di terminal (CLI), jendela sederhana (GUI), dan melihat dasbor data pertama.

Bab 42 — Antarmuka: Perintah, Jendela, dan Dasbor

Seri I5 — Aplikasi Pertama: Web, Peta, dan Dasbor

Di bab ini, Anda akan:

- Membuat alat **baris perintah (CLI)** dengan `argparse` dan **Typer**
- Menguji CLI tanpa membuka terminal (`subprocess` dan `CliRunner`)
- Membuat jendela sederhana (**GUI**) dengan `tkinter` dan mengujinya tanpa menampilkannya
- Menyusun **dasbor** data pertama sebagai halaman HTML mandiri
- Membungkus data unit usaha dengan CLI: proyek bersambung Seri I5

Semua skrip bab ini ada di folder `kode/bab42/`. Di versi web, Anda juga bisa mengunduh semuanya sekaligus (zip).

Skrip yang Anda tulis sejauh ini dijalankan dengan mengetik `python nama.py`. Itu cukup untuk Anda, tetapi orang lain butuh **pintu masuk** yang lebih ramah. Bayangkan warung: ada yang memesan lewat loket (cepat, tinggal sebut pesanan), ada yang duduk di meja dengan daftar menu bergambar, dan ada yang melihat papan pengumuman penjualan hari ini. Satu dapur yang sama, tiga cara berinteraksi. Itulah tiga jenis **antarmuka** (*interface*) yang kita bahas.

Gambar 0.1: Tiga jenis antarmuka untuk program yang sama.

Di bab ini kita memakai data unit usaha tiruan (15 unit di lima sektor, **semua angka fiktif**) dari berkas `data/unit_usaha.csv`, dibaca oleh modul bantu `data_unit.py`.

CLI dengan argparse

CLI (*Command-Line Interface*) adalah program yang dipakai dengan mengetik perintah di terminal, seperti `git commit -m "pesan"`. **argparse** adalah modul bawaan Python untuk membaca ketikan itu.

```
# Bab 42 - argparse: membaca argumen dari baris perintah
import argparse

parser = argparse.ArgumentParser(prog="hitung_pakan", description="Menghitung pakan
```

```
harian kolam.")
parser.add_argument("biomassa_kg", type=float, help="biomassa ikan (kg)")           #
posisi, wajib
parser.add_argument("--persen", type=float, default=3.0, help="persen pakan (bawaan
3.0)")  # opsi
parser.add_argument("--rinci", action="store_true", help="tampilkan rincian")       #
sakelar

# Biasanya argparse membaca ketikan terminal. Di sini kita beri daftar sendiri agar
bisa dicoba.
for ketikan in (["432.4"], ["432.4", "--persen", "2.5", "--rinci"]):
    args = parser.parse_args(ketikan)
    pakan = args.biomassa_kg * args.persen / 100
    print(f"ketikan {ketikan} -> {args}")
     print(f"  pakan: {pakan:.2f} kg" + (" (rincian: biomassa x persen / 100)" if
args.rinci else ""))

print()
parser.print_help()
```

```
ketikan ['432.4'] -> Namespace(biomassa_kg=432.4, persen=3.0, rinci=False)
  pakan: 12.97 kg
ketikan  ['432.4',  '--persen',  '2.5',  '--rinci']  ->  Namespace(biomassa_kg=432.4,
persen=2.5, rinci=True)
  pakan: 10.81 kg (rincian: biomassa x persen / 100)

usage: hitung_pakan [-h] [--persen PERSEN] [--rinci] biomassa_kg

Menghitung pakan harian kolam.

positional arguments:
  biomassa_kg      biomassa ikan (kg)

options:
  -h, --help       show this help message and exit
  --persen PERSEN  persen pakan (bawaan 3.0)
  --rinci          tampilkan rincian
```

Tiga jenis argumen muncul di sini:

- **Posisi** (`biomassa_kg`): wajib, dikenali dari urutannya.
- **Opsi** (`--persen`): diawali tanda hubung ganda, boleh dilewati, punya nilai bawaan.
- **Sakelar** (`--rinci`, `action="store_true"`): bernilai `True` bila ditulis, `False` bila tidak.

`type=float` membuat argparse mengubah teks `"432.4"` menjadi angka. Hasil `parse_args()` adalah objek `Namespace` yang atributnya Anda baca (`args.persen`). Biasanya `parse_args()` membaca ketikan terminal; di skrip ini kita memberinya **list** supaya bisa dicoba tanpa terminal (cara yang sama dipakai untuk menguji). Hitungannya: 432,4 × 3 / 100 = 12,97 kg dan 432,4 × 2,5 / 100 = 10,81 kg. Bagian bawah keluaran adalah **bantuan otomatis** (`--help`) yang dibuat argparse dari deklarasi tadi, gratis.

Bila pengguna salah ketik

Alat yang baik **menolak masukan salah dengan jelas**. Kita uji alat `unit` sungguhan (modul bantu `cli_argparse.py`, ditampilkan di bawah) lewat `subprocess` dari Bab 41:

```
# Bab 42 - Alat baris perintah "unit" versi argparse (modul bantu; dijalankan lewat
skrip 02 dan 03)
import argparse
import sys

from data_unit import baca_unit, ringkas_per_sektor
```

```
SEKTOR = ["hutan", "kebun", "sawah", "kolam", "kandang"]


def buat_parser():
    parser = argparse.ArgumentParser(prog="unit", description="Melihat data unit usaha
(data fiktif).")
    sub = parser.add_subparsers(dest="perintah", required=True)

    daftar = sub.add_parser("daftar", help="menampilkan unit")
    daftar.add_argument("--sektor", choices=SEKTOR, help="hanya satu sektor")
     daftar.add_argument("--maks", type=int, default=0, help="batas jumlah baris (0
= semua)")

    sub.add_parser("ringkas", help="ringkasan per sektor")
    return parser


def main(argv=None):
    args = buat_parser().parse_args(argv)
    unit = baca_unit()
    if args.perintah == "daftar":
        if args.sektor:
            unit = [u for u in unit if u["sektor"] == args.sektor]
        if args.maks:
            unit = unit[: args.maks]
        for u in unit:
                print(f'{u["id"]:3} {u["sektor"]:8} {u["unit"]:10} {u["nilai"]:8.1f}
{u["satuan"]}')
    else:
        for sektor, r in ringkas_per_sektor(unit).items():
                  print(f'{sektor:8} {r["jumlah"]} unit, rata-rata {r["rata"]:.1f}
{r["satuan"]}')
    return 0


if __name__ == "__main__":
    sys.exit(main())
```

```
# Bab 42 - argparse: apa yang terjadi bila pengguna salah mengetik
import subprocess
import sys

for argumen in (["daftar", "--sektor", "empang"], ["daftar", "--maks", "dua"], []):
    hasil = subprocess.run(
        [sys.executable, "cli_argparse.py", *argumen],
        capture_output=True, text=True, encoding="utf-8",
    )
    print("argumen:", argumen)
    print("kode keluar:", hasil.returncode)
    print(hasil.stderr.strip().splitlines()[-1])
    print()
```

```
argumen: ['daftar', '--sektor', 'empang']
kode keluar: 2
unit daftar: error: argument --sektor: invalid choice: 'empang' (choose from 'hutan',
'kebun', 'sawah', 'kolam', 'kandang')

argumen: ['daftar', '--maks', 'dua']
kode keluar: 2
```

```
unit daftar: error: argument --maks: invalid int value: 'dua'

argumen: []
kode keluar: 2
unit: error: the following arguments are required: perintah
```

Dokumentasi resmi menyebut perilaku ini: saat masukan tidak sah, `parse_args()` mencetak pesan penggunaan ke `stderr` lalu keluar dengan **kode keluar 2**. Pilihan `choices=SEKTOR` membuat sektor `empang` ditolak, `type=int` menolak `dua`, dan subperintah yang `required=True` menolak ketikan kosong. Pesan galat argparse ditulis dalam bahasa Inggris oleh Python; itu bukan kesalahan skrip kita.

Subperintah

Alat besar punya beberapa perintah (`git commit`, `git push`). Di argparse, itu dibuat dengan `add_subparsers()`: `daftar` dan `ringkas` pada alat `unit` kita. Uji lewat program anak, persis seperti dipakai orang:

```
# Bab 42 - argparse: alat "unit" dengan subperintah, diuji lewat subprocess
import subprocess
import sys


def unit(*argumen):
    hasil = subprocess.run([sys.executable, "cli_argparse.py", *argumen],
                           capture_output=True, text=True, encoding="utf-8")
    return hasil.returncode, hasil.stdout


for argumen in (["ringkas"], ["daftar", "--sektor", "kolam"], ["daftar", "--maks",
"2"]):
    kode, keluaran = unit(*argumen)
    print(f"$ unit {' '.join(argumen)}   (kode keluar {kode})")
    print(keluaran)
```

```
$ unit ringkas   (kode keluar 0)
hutan    3 unit, rata-rata 35.2 cm
kebun    4 unit, rata-rata 257.5 kg
sawah    2 unit, rata-rata 1115.2 kg
kolam    3 unit, rata-rata 369.3 kg
kandang  3 unit, rata-rata 383.8 kg

$ unit daftar --sektor kolam   (kode keluar 0)
L1  kolam    Kolam A       432.4 kg
L2  kolam    Kolam B       380.0 kg
L3  kolam    Kolam C       295.5 kg

$ unit daftar --maks 2   (kode keluar 0)
H1  hutan    Plot A         32.5 cm
H2  hutan    Plot B         45.2 cm
```

CLI dengan Typer

argparse kuat, tetapi banyak tulisan. **Typer** (pustaka pihak ketiga, dibangun di atas Click) membalik caranya: Anda menulis **fungsi biasa dengan type hint** (Bab 33), dan Typer membuat perintahnya. Parameter tanpa nilai bawaan menjadi argumen wajib; parameter dengan nilai bawaan menjadi opsi; `bool` menjadi sakelar.

```
# Bab 42 - Typer: fungsi biasa menjadi perintah, lengkap dengan pengujian tanpa terminal
import typer
```

```
from typer.testing import CliRunner

app = typer.Typer(add_completion=False)


@app.command()
def pakan(biomassa_kg: float, persen: float = 3.0, rinci: bool = False):
    """Menghitung pakan harian kolam."""
    hasil = biomassa_kg * persen / 100
    typer.echo(f"pakan: {hasil:.2f} kg")
    if rinci:
        typer.echo("rincian: biomassa x persen / 100")


runner = CliRunner()       # penguji bawaan Typer: memanggil aplikasi tanpa membuka
terminal

for ketikan in (["432.4"], ["432.4", "--persen", "2.5", "--rinci"], ["banyak"]):
    hasil = runner.invoke(app, ketikan)
    print(ketikan, "-> kode keluar", hasil.exit_code)
     print(hasil.output.strip().splitlines()[0] if hasil.exit_code == 0 else "(galat
penggunaan)")

hasil = runner.invoke(app, ["--help"])
print()
print(hasil.output)
```

```
['432.4'] -> kode keluar 0
pakan: 12.97 kg
['432.4', '--persen', '2.5', '--rinci'] -> kode keluar 0
pakan: 10.81 kg
['banyak'] -> kode keluar 2
(galat penggunaan)


 Usage: pakan [OPTIONS] {biomassa_kg}

 Menghitung pakan harian kolam.

╭─ Arguments ──────────────────────────────────────────────────────────────────╮
│ *    biomassa_kg      <float>  [required]                                    │
╰──────────────────────────────────────────────────────────────────────────────╯
╭─ Options ────────────────────────────────────────────────────────────────────╮
│ --persen                 <float>  [default: 3.0]                             │
│ --rinci     --no-rinci            [default: no-rinci]                       │
│ --help                            Show this message and exit.               │
╰──────────────────────────────────────────────────────────────────────────────╯
```

- Ketiga ketikan menghasilkan: sukses (12,97), sukses dengan opsi (10,81), dan galat penggunaan (kode keluar 2) untuk `banyak`, karena bukan angka.
- `CliRunner` adalah **penguji bawaan Typer**: ia memanggil aplikasi di dalam proses yang sama dan menyimpan `exit_code` serta `output`. Tidak ada terminal yang perlu dibuka, jadi cepat dan cocok untuk tes otomatis.
- Bantuan otomatis memakai tampilan kotak (dibuat pustaka `rich`). Perhatikan `--rinci` dan `--no-rinci`: Typer menambah pasangan sakelar sendiri.

Berikut alat `unit` versi Typer, dengan tiga perintah (`daftar`, `ringkas`, `laporan`):

```
# Bab 42 - Alat baris perintah "unit" versi Typer (modul bantu; dipakai skrip 04
sampai 09)
```

```
from enum import Enum
from pathlib import Path
from typing import Annotated, Optional

import typer

from data_unit import baca_unit, ringkas_per_sektor

app = typer.Typer(help="Melihat data unit usaha (data fiktif).", no_args_is_help=True,
add_completion=False)


class Sektor(str, Enum):
    hutan = "hutan"
    kebun = "kebun"
    sawah = "sawah"
    kolam = "kolam"
    kandang = "kandang"


@app.command()
def daftar(
   sektor: Annotated[Optional[Sektor], typer.Option(help="hanya satu sektor")] = None,
    maks: Annotated[int, typer.Option(min=0, help="batas jumlah baris (0 = semua)")]
= 0,
):
    """Menampilkan unit."""
    unit = baca_unit()
    if sektor:
        unit = [u for u in unit if u["sektor"] == sektor.value]
    if maks:
        unit = unit[:maks]
    for u in unit:
           typer.echo(f'{u["id"]:3}  {u["sektor"]:8}  {u["unit"]:10}  {u["nilai"]:8.1f}
{u["satuan"]}')


@app.command()
def ringkas():
    """Ringkasan per sektor."""
    for sektor, r in ringkas_per_sektor(baca_unit()).items():
             typer.echo(f'{sektor:8}  {r["jumlah"]}  unit,  rata-rata  {r["rata"]:.1f}
{r["satuan"]}')


@app.command()
def laporan(keluar: Annotated[Path, typer.Argument(help="berkas Markdown tujuan")]):
    """Menulis laporan ringkas ke berkas."""
    baris = ["# Ringkasan unit usaha (data fiktif)", ""]
    for sektor, r in ringkas_per_sektor(baca_unit()).items():
           baris.append(f'- {sektor}: {r["jumlah"]} unit, rata-rata {r["rata"]:.1f}
{r["satuan"]}')
    keluar.parent.mkdir(parents=True, exist_ok=True)
    keluar.write_text("\n".join(baris) + "\n", encoding="utf-8")
    typer.echo(f"laporan ditulis: {keluar.name}")


if __name__ == "__main__":
    app()
```

`Enum` (daftar pilihan tetap) menggantikan `choices`. `Annotated[..., typer.Option(...)]` menambah keterangan dan batas (`min=0`). Fungsi di atas bisa dijalankan sebagai program:

```
# Bab 42 - Typer: alat "unit" lengkap dijalankan sebagai program sungguhan lewat
subprocess
import subprocess
import sys


def unit(*argumen):
    hasil = subprocess.run([sys.executable, "cli_typer.py", *argumen],
                           capture_output=True, text=True, encoding="utf-8")
    return hasil.returncode, hasil.stdout + hasil.stderr


for argumen in (["--help"], ["daftar", "--sektor", "kandang"], ["daftar", "--sektor",
"empang"]):
    kode, teks = unit(*argumen)
    print(f"$ unit {' '.join(argumen)}   (kode keluar {kode})")
    print(teks)
```

```
$ unit --help   (kode keluar 0)

 Usage: cli_typer.py [OPTIONS] COMMAND [ARGS]...

 Melihat data unit usaha (data fiktif).

╭─ Options ────────────────────────────────────────────────────────────────────╮
│ --help          Show this message and exit.                                  │
╰──────────────────────────────────────────────────────────────────────────────╯
╭─ Commands ───────────────────────────────────────────────────────────────────╮
│ daftar   Menampilkan unit.                                                   │
│ ringkas  Ringkasan per sektor.                                               │
│ laporan  Menulis laporan ringkas ke berkas.                                  │
╰──────────────────────────────────────────────────────────────────────────────╯


$ unit daftar --sektor kandang   (kode keluar 0)
D1  kandang  Kandang 1     350.0 kg
D2  kandang  Kandang 2     412.5 kg
D3  kandang  Kandang 3     389.0 kg

$ unit daftar --sektor empang   (kode keluar 2)
Usage: cli_typer.py daftar [OPTIONS]
Try 'cli_typer.py daftar --help' for help.
╭─ Error ──────────────────────────────────────────────────────────────────────╮
│ Invalid value for '--sektor': 'empang' is not one of 'hutan', 'kebun',       │
│ 'sawah', 'kolam', 'kandang'.                                                 │
╰──────────────────────────────────────────────────────────────────────────────╯
```

Sektor `empang` ditolak dengan kode keluar 2 dan kotak galat yang menyebut pilihan yang sah.

> TIPS
>
> Pilih **argparse** bila Anda ingin nol ketergantungan (bawaan Python). Pilih **Typer** bila alat Anda punya banyak perintah dan Anda suka kode ringkas berbasis type hint. Keduanya sama-sama menghasilkan bantuan otomatis dan kode keluar yang jelas.

Menguji CLI

CLI yang tidak diuji akan rusak diam-diam. Dengan `CliRunner`, tiap perilaku jadi satu pemeriksaan:

```
# Bab 42 - Menguji CLI Typer dengan CliRunner (gaya tes pytest, ditulis sebagai skrip)
from pathlib import Path

from typer.testing import CliRunner

from cli_typer import app

runner = CliRunner()
lulus = 0


def periksa(kondisi, nama):
    global lulus
    print(("LULUS  " if kondisi else "GAGAL  ") + nama)
    lulus += bool(kondisi)


hasil = runner.invoke(app, ["ringkas"])
periksa(hasil.exit_code == 0 and "kolam" in hasil.output, "ringkas menampilkan sektor
kolam")

hasil = runner.invoke(app, ["daftar", "--sektor", "sawah"])
periksa(len(hasil.output.strip().splitlines()) == 2, "daftar --sektor sawah memberi
2 baris")

hasil = runner.invoke(app, ["daftar", "--maks", "4"])
periksa(len(hasil.output.strip().splitlines()) == 4, "daftar --maks 4 memberi 4 baris")

hasil = runner.invoke(app, ["daftar", "--sektor", "empang"])
periksa(hasil.exit_code == 2, "sektor tak dikenal ditolak dengan kode keluar 2")

tujuan = Path(__file__).parent / "_tmp" / "ringkasan.md"
hasil = runner.invoke(app, ["laporan", str(tujuan)])
periksa(hasil.exit_code  ==  0  and  tujuan.read_text(encoding="utf-8").startswith("#
Ringkasan"),
        "laporan menulis berkas Markdown")

print(f"\n{lulus} dari 5 pemeriksaan lulus")
```

```
LULUS  ringkas menampilkan sektor kolam
LULUS  daftar --sektor sawah memberi 2 baris
LULUS  daftar --maks 4 memberi 4 baris
LULUS  sektor tak dikenal ditolak dengan kode keluar 2
LULUS  laporan menulis berkas Markdown

5 dari 5 pemeriksaan lulus
```

Ini gaya tes `pytest` (Bab 18) yang ditulis sebagai skrip agar keluarannya tampil. Perhatikan pengujian **jalur gagal** (sektor tak dikenal harus kode 2) sama pentingnya dengan jalur sukses. Ada dua cara menguji CLI di bab ini: `CliRunner` (cepat, dalam satu proses) dan `subprocess` (lambat sedikit, tetapi persis seperti dipakai orang di terminal).

GUI sederhana dengan tkinter

GUI (*Graphical User Interface*) adalah program berjendela: tombol, kotak pilihan, tabel. **tkinter** adalah pustaka GUI yang menyertai Python (membungkus Tk). Dokumentasi resmi mencatat bahwa tkinter adalah modul **opsional**: bila tidak ada di salinan Python Anda, cari petunjuk dari penyedia Python Anda. Di komputer penulis ia ada (versi Tk 9.0). **ttk** adalah kumpulan widget bertema yang dianjurkan untuk kode baru.

```
# Bab 42 - Jendela tkinter untuk melihat unit usaha (modul bantu)
import tkinter as tk
from tkinter import ttk

from data_unit import baca_unit

SEKTOR = ["semua", "hutan", "kebun", "sawah", "kolam", "kandang"]


def bangun_jendela(root):
    """Menyusun isi jendela dan mengembalikan widget yang perlu diuji."""
    root.title("Unit usaha (data fiktif)")
    pilihan = tk.StringVar(value="semua")
    status = tk.StringVar()

    atas = ttk.Frame(root, padding=8)
    atas.pack(fill="x")
    ttk.Label(atas, text="Sektor:").pack(side="left")
    kotak = ttk.Combobox(atas, textvariable=pilihan, values=SEKTOR, state="readonly",
width=10)
    kotak.pack(side="left", padx=6)

     tabel = ttk.Treeview(root, columns=("unit", "jenis", "nilai"), show="headings",
height=8)
    for kolom, judul in (("unit", "Unit"), ("jenis", "Jenis"), ("nilai", "Nilai")):
        tabel.heading(kolom, text=judul)
    tabel.pack(fill="both", expand=True, padx=8)
    ttk.Label(root, textvariable=status, padding=8).pack(anchor="w")

    def muat(*_):
        tabel.delete(*tabel.get_children())
       terpilih = [u for u in baca_unit() if pilihan.get() in ("semua", u["sektor"])]
        for u in terpilih:
               tabel.insert("", "end", values=(u["unit"], u["jenis"], f'{u["nilai"]}
{u["satuan"]}'))
        status.set(f"{len(terpilih)} unit ditampilkan")

    kotak.bind("<<ComboboxSelected>>", muat)
    muat()
    return pilihan, tabel, status, muat


if __name__ == "__main__":
    jendela = tk.Tk()
    bangun_jendela(jendela)
    jendela.mainloop()          # menampilkan jendela dan menunggu pengguna
```

Jendela ini punya kotak pilihan sektor, tabel unit, dan baris status. Memilih sektor memanggil `muat()`, yang mengosongkan tabel dan mengisinya kembali. Perhatikan pola yang baik: seluruh isi jendela dibangun oleh fungsi `bangun_jendela(root)`, sedangkan `mainloop()` ada **hanya** di bagian `if __name__ == "__main__":`.

Kode ini **tidak dijalankan** saat buku dibuat (bagian `mainloop()`) karena jendela yang tampil butuh layar dan pengguna, jadi keluarannya tidak ditampilkan. Anda menjalankannya sendiri dengan `python gui_unit.py`. Tetapi isinya tetap bisa diuji tanpa menampilkan apa-apa, karena strukturnya terpisah:

```
# Bab 42 - tkinter: menguji jendela TANPA menampilkannya (jendela disembunyikan)
import tkinter as tk

from gui_unit import bangun_jendela

root = tk.Tk()
```

```
root.withdraw()                     # sembunyikan: tidak ada jendela muncul di layar
pilihan, tabel, status, muat = bangun_jendela(root)
root.update()

print("judul jendela :", root.title())
print("awal          :", status.get(), "| baris tabel:", len(tabel.get_children()))

for sektor in ("kolam", "hutan"):
    pilihan.set(sektor)             # seolah-olah pengguna memilih di kotak pilihan
    muat()
    baris = [tabel.item(i, "values")[0] for i in tabel.get_children()]
    print(f"{sektor:6}        :", status.get(), "->", baris)

root.destroy()
```

```
judul jendela : Unit usaha (data fiktif)
awal          : 15 unit ditampilkan | baris tabel: 15
kolam         : 3 unit ditampilkan -> ['Kolam A', 'Kolam B', 'Kolam C']
hutan         : 3 unit ditampilkan -> ['Plot A', 'Plot B', 'Plot C']
```

`root.withdraw()` menyembunyikan jendela sehingga tidak ada yang muncul di layar. Lalu kita **meniru pengguna**: `pilihan.set("kolam")` seperti memilih di kotak, `muat()` seperti peristiwa pilihan. Hasilnya cocok: 15 unit di awal, 3 unit kolam (Kolam A, B, C), dan 3 unit hutan (Plot A, B, C).

> **i** INGAT
>
> Yang diuji di sini adalah **logika** jendela (isi tabel, teks status). Tampilannya (posisi tombol, warna, ukuran) tidak bisa diperiksa tanpa melihat jendela itu sendiri. Pemeriksaan tampilan tetap tugas Anda, dengan mata, di komputer Anda.

Dasbor data pengantar

Dasbor (*dashboard*) adalah satu halaman yang merangkum angka penting dengan tabel dan grafik, seperti panel instrumen mobil. Dasbor paling sederhana tidak butuh server: satu berkas HTML mandiri yang bisa dibuka dengan klik ganda.

```
# Bab 42 - Dasbor pengantar: satu halaman HTML mandiri berisi tabel dan grafik
import base64
import io
from pathlib import Path

import matplotlib.pyplot as plt
import pandas as pd

from data_unit import BERKAS

df = pd.read_csv(BERKAS)
ringkas = df.groupby("sektor", sort=False)["nilai"].agg(["count", "mean"]).round(1)
print(ringkas)

fig, ax = plt.subplots(figsize=(6, 3))
ringkas["count"].plot.bar(ax=ax, color="#4f8a3b")
ax.set_ylabel("jumlah unit")
ax.set_xlabel("")
fig.tight_layout()
buf = io.BytesIO()
fig.savefig(buf, format="png", dpi=100)
plt.close(fig)
gambar = base64.b64encode(buf.getvalue()).decode("ascii")
```

```
halaman = f"""<!doctype html>
<html lang="id"><head><meta charset="utf-8"><title>Dasbor unit usaha</title></head>
<body style="font-family: sans-serif; max-width: 640px; margin: 2em auto">
<h1>Dasbor unit usaha (data fiktif)</h1>
<p>Total unit: <b>{len(df)}</b> di <b>{df['sektor'].nunique()}</b> sektor.</p>
<img alt="jumlah unit per sektor" src="{gambar}">
{ringkas.to_html(classes="ringkas")}
</body></html>
"""
tujuan = Path(__file__).parent / "_tmp" / "dasbor.html"
tujuan.parent.mkdir(exist_ok=True)
tujuan.write_text(halaman, encoding="utf-8")
print("\nberkas dasbor.html dibuat:", tujuan.stat().st_size > 5000, "(bisa dibuka
dengan klik ganda)")
print("memuat grafik tertanam :", "" in halaman)
```

```
         count    mean
sektor
hutan        3    35.2
kebun        4   257.5
sawah        2  1115.2
kolam        3   369.3
kandang      3   383.8

berkas dasbor.html dibuat: True (bisa dibuka dengan klik ganda)
memuat grafik tertanam : True
```

Grafik dibuat dengan matplotlib, diubah menjadi teks **base64**, lalu ditanam langsung di HTML (`...`), sehingga berkas tidak butuh gambar terpisah. Tabel ringkasan datang dari `to_html()` milik pandas. Kita memeriksa bahwa berkasnya terbentuk dan memuat grafik; kita **tidak** bisa melihat tampilannya dari skrip ini, jadi buka `_tmp/dasbor.html` sendiri bila ingin melihatnya.

Kelemahan dasbor statis: angkanya **beku** saat berkas dibuat dan tidak ada interaksi. Dasbor yang hidup, bisa dipilih-pilih, dan menampilkan peta dibahas di Bab 43.

Proyek Bersambung I5: CLI untuk unit usaha

Di Bab 41 laporan mingguan diambil lewat scraping. Sekarang kita beri alat `unit` kepada rekan kerja: mereka cukup mengetik perintah. Skrip otomasi harian di bawah memanggil CLI itu seperti seseorang di terminal, memeriksa kode keluar tiap langkah, dan berhenti bila ada yang gagal.

```
# Bab 42 - Proyek bersambung I5: CLI "unit" dipanggil oleh skrip otomasi harian
import subprocess
import sys
from pathlib import Path

tujuan = Path(__file__).parent / "_tmp" / "harian" / "ringkasan.md"


def unit(*argumen):
    hasil = subprocess.run([sys.executable, "cli_typer.py", *argumen],
                           capture_output=True, text=True, encoding="utf-8")
    return hasil.returncode, hasil.stdout.strip()


langkah = [
    ("ringkas", ["ringkas"]),
    ("dua kolam pertama", ["daftar", "--sektor", "kolam", "--maks", "2"]),
    ("tulis laporan", ["laporan", str(tujuan)]),
]
```

```
for nama, argumen in langkah:
    kode, keluaran = unit(*argumen)
    print(f"[{nama}] kode keluar {kode}")
    print(keluaran)
    if kode != 0:                 # otomasi yang baik berhenti bila satu langkah gagal
        raise SystemExit(f"langkah '{nama}' gagal")

print("\nisi berkas laporan:")
print(tujuan.read_text(encoding="utf-8"))
```

```
[ringkas] kode keluar 0
hutan    3 unit, rata-rata 35.2 cm
kebun    4 unit, rata-rata 257.5 kg
sawah    2 unit, rata-rata 1115.2 kg
kolam    3 unit, rata-rata 369.3 kg
kandang  3 unit, rata-rata 383.8 kg
[dua kolam pertama] kode keluar 0
L1  kolam    Kolam A       432.4 kg
L2  kolam    Kolam B       380.0 kg
[tulis laporan] kode keluar 0
laporan ditulis: ringkasan.md

isi berkas laporan:
# Ringkasan unit usaha (data fiktif)

- hutan: 3 unit, rata-rata 35.2 cm
- kebun: 4 unit, rata-rata 257.5 kg
- sawah: 2 unit, rata-rata 1115.2 kg
- kolam: 3 unit, rata-rata 369.3 kg
- kandang: 3 unit, rata-rata 383.8 kg
```

Tiga langkah berjalan, semuanya berkode keluar 0. Berkas laporan berisi rata-rata tiap sektor. Contohnya kebun: 4 unit dengan rata-rata (245,5 + 310,0 + 198,5 + 276,0) / 4 = 1030,0 / 4 = **257,5** kg. Kebiasaan `raise SystemExit(...)` saat langkah gagal membuat otomasi tidak melanjutkan pekerjaan di atas hasil yang rusak.

Latihan

1. Tambahkan opsi `--tinggi` (sakelar) pada `01_argparse_dasar.py`; bila ditulis, cetak juga "pakan per ekor" untuk 800 ekor ikan.
2. Pada `cli_typer.py`, tambahkan perintah `satu` yang menerima `id_unit` (mis. `D2`) dan mencetak unit itu, atau "tidak ada" dengan kode keluar 1. Tambahkan dua pemeriksaan di `06_uji_cli.py`.
3. Ubah `gui_unit.py` supaya status juga menampilkan nilai tertinggi yang sedang tampil, dan perbarui `07_tkinter_uji.py` untuk memeriksanya.
4. **Tantangan:** buat CLI `unit` memiliki perintah `dasbor` yang memanggil kode `08_dasbor_statis.py` dan mencetak lokasi berkasnya. Uji dengan `CliRunner`.

Ringkasan Bab

- **Antarmuka** ada tiga: baris perintah (CLI), jendela (GUI), dan halaman web (dasbor).
- **argparse** (bawaan): argumen posisi, opsi, sakelar, `choices`, subperintah. Masukan salah: pesan ke `stderr` dan kode keluar **2**.
- **Typer**: fungsi bertype hint menjadi perintah; uji dengan `CliRunner` (`exit_code`, `output`).
- Uji CLI di dua jalur: sukses **dan** gagal; `CliRunner` atau `subprocess`.
- **tkinter**: pisahkan pembangunan jendela dari `mainloop()` supaya logikanya bisa diuji dengan `withdraw()`; tampilan tetap harus dilihat manusia.
- **Dasbor** statis = satu HTML mandiri dengan grafik tertanam; dasbor interaktif menunggu di Bab 43.

Selanjutnya, di Bab 43: aplikasi data cepat dan dasbor hidup: Streamlit, Gradio, Dash, peta interaktif, HTMX, cara menyajikannya ke pengguna, dan penutup Seri I5.

Bab 43 — Aplikasi Data Cepat dan Dasbor: Dari Skrip ke Layar Orang Lain

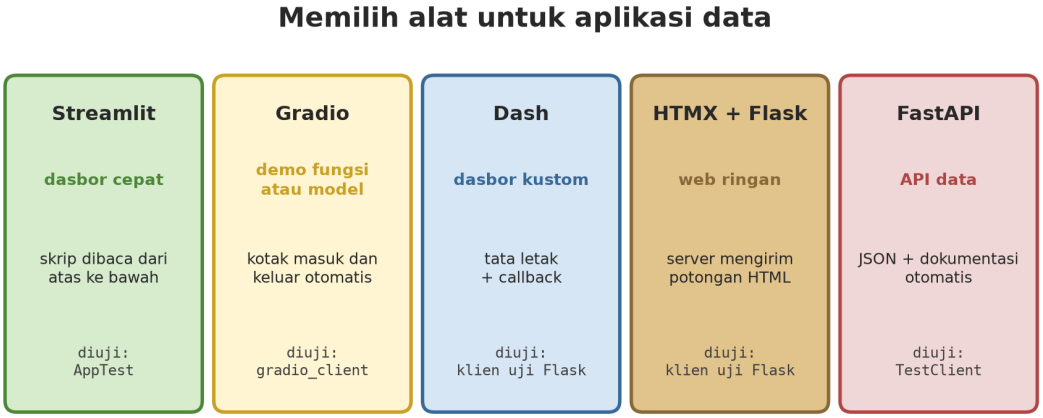
Seri I5 — Aplikasi Pertama: Web, Peta, dan Dasbor

Di bab ini, Anda akan:

- Membuat dasbor dengan **Streamlit** dan mengujinya tanpa peramban (`AppTest`)
- Membuat demo fungsi dengan **Gradio** dan **Dash** untuk dasbor kustom, lalu mengujinya
- Menampilkan **peta interaktif** (folium dan pydeck) di dasbor
- Mengenal **HTMX** untuk web ringan dan **FastAPI** untuk API data
- Memahami cara **menyajikan** aplikasi ke pengguna, lalu menutup Seri I5

Semua skrip bab ini ada di folder `kode/bab43/`. Di versi web, Anda juga bisa mengunduh semuanya sekaligus (zip).

Bab 42 menutup dengan dasbor statis: bagus, tetapi beku. Kali ini kita membuat dasbor yang **hidup**: pengguna memilih sektor, angkanya berubah, peta bergeser. Untungnya Python punya pustaka yang membuat ini mungkin tanpa harus belajar JavaScript. Bayangkan sebuah toko: dulu Anda menempel daftar harga di dinding (dasbor statis); sekarang Anda punya kasir yang menjawab pertanyaan pembeli satu per satu.



Gambar 0.1: Memilih alat untuk aplikasi data.

Kita memakai data unit usaha tiruan yang sama dengan bab sebelumnya (15 unit, lima sektor, **semua angka fiktif**). Semua pustaka di bawah sudah terpasang di komputer penulis dan **diuji dengan menjalankannya**; versi pustaka terus berubah, jadi bila perilakunya berbeda di komputer Anda, periksa dokumentasi resmi terbaru.

Satu keterbatasan yang jujur: bab ini **tidak bisa memperlihatkan tampilan** di peramban. Semua yang dijalankan di sini diuji lewat kode (tanpa membuka peramban), sehingga yang terbukti adalah **logika dan isinya**, bukan penampilannya.

Streamlit: dasbor dari satu skrip

Streamlit mengubah skrip Python biasa menjadi halaman web. Idenya sederhana: skrip dibaca **dari atas ke bawah**, dan setiap kali pengguna mengubah sesuatu (memilih, menggeser), **seluruh skrip dijalankan ulang**. Tidak ada HTML, tidak ada JavaScript.

```
# Bab 43 - Dasbor Streamlit per sektor (modul bantu; dijalankan dengan: streamlit
run app_streamlit.py)
import pandas as pd
import streamlit as st

from data_unit import baca_unit
from peta import buat_peta

SEKTOR = ["semua", "hutan", "kebun", "sawah", "kolam", "kandang"]


@st.cache_data                          # hasil disimpan; tidak membaca CSV ulang
setiap interaksi
def muat_unit():
    return pd.DataFrame(baca_unit())


st.set_page_config(page_title="Dasbor unit usaha", layout="wide")
st.title("Dasbor unit usaha (data fiktif)")

sektor = st.sidebar.selectbox("Sektor", SEKTOR)
unit = muat_unit()
tampil = unit if sektor == "semua" else unit[unit["sektor"] == sektor]

kiri, tengah, kanan = st.columns(3)
kiri.metric("Jumlah unit", len(tampil))
tengah.metric("Rata-rata nilai", f"{tampil['nilai'].mean():.1f}")
kanan.metric("Nilai tertinggi", f"{tampil['nilai'].max():.1f}")

if sektor == "semua":
    st.info("Nilai tiap sektor berbeda satuan (cm atau kg): pilih satu sektor untuk
membandingkan.")
else:
    st.bar_chart(tampil.set_index("unit")["nilai"])

st.dataframe(tampil[["id",   "sektor",   "unit",   "jenis",   "nilai",   "satuan"]],
hide_index=True)
st.map(tampil, latitude="lat", longitude="lon")

with st.expander("Peta interaktif (folium)"):
    st.iframe(buat_peta(tampil.to_dict("records")).get_root().render(), height=420)
```

Bacalah seperti resep:

- `st.sidebar.selectbox(...)` membuat kotak pilihan di bilah samping dan **mengembalikan** pilihan sekarang.
- `st.columns(3)` dan `.metric(...)` membuat tiga kartu angka.
- `st.bar_chart`, `st.dataframe`, dan `st.map` membuat grafik, tabel, dan peta titik. `st.map` diberi nama kolom lintang dan bujur (`lat` dan `lon`).
- `@st.cache_data` menyimpan hasil `muat_unit()` supaya CSV tidak dibaca ulang setiap interaksi.
- Bagian terakhir menanam **peta folium** dalam sebuah lipatan (dibahas di bagian peta).

Aplikasi ini dijalankan dengan `streamlit run app_streamlit.py`, yang membuka alamat di komputer Anda. **Perintah itu tidak dijalankan saat buku dibuat**, sebab ia menyalakan server yang menunggu peramban. Sebagai gantinya, Streamlit menyediakan **`AppTest`** (dokumentasi resmi: pengujian tanpa peramban): ia menjalankan skrip di memori dan memberi akses ke elemennya.

```
# Bab 43 - Streamlit: menjalankan dasbor TANPA peramban memakai AppTest
import logging

from streamlit.testing.v1 import AppTest
```

```
# "bare mode": AppTest tidak punya peramban, jadi Streamlit mengeluh pelan-pelan;
diamkan peringatannya
logging.getLogger("streamlit.runtime.scriptrunner_utils.script_run_context").setLevel(logging.ERROR)

at = AppTest.from_file("app_streamlit.py", default_timeout=30).run()

print("ada galat      :", bool(at.exception))
print("judul          :", at.title[0].value)
print("pilihan sektor :", at.selectbox[0].options)
print("sektor terpilih:", at.selectbox[0].value)
for m in at.metric:
    print(f"metrik         : {m.label} = {m.value}")
print("baris tabel    :", len(at.dataframe[0].value))
print("kotak info     :", at.info[0].value[:40] + "...")
```

```
ada galat      : False
judul          : Dasbor unit usaha (data fiktif)
pilihan sektor : ['semua', 'hutan', 'kebun', 'sawah', 'kolam', 'kandang']
sektor terpilih: semua
metrik         : Jumlah unit = 15
metrik         : Rata-rata nilai = 375.0
metrik         : Nilai tertinggi = 1250.0
baris tabel    : 15
kotak info     : Nilai tiap sektor berbeda satuan (cm ata...
```

Pada awal, sektor terpilih adalah `semua`, dan angka-angkanya benar: 15 unit, rata-rata nilai 375,0 (angka ini mencampur cm dan kg, sehingga aplikasi menampilkan kotak info yang memperingatkannya), dan nilai tertinggi 1250,0 (petak sawah 1). Kini kita meniru pengguna:

```
# Bab 43 - Streamlit: meniru pengguna memilih sektor, lalu memeriksa hasilnya
import logging

from streamlit.testing.v1 import AppTest

logging.getLogger("streamlit.runtime.scriptrunner_utils.script_run_context").setLevel(logging.ERROR)

at = AppTest.from_file("app_streamlit.py", default_timeout=30).run()

for sektor in ["hutan", "kolam", "kandang"]:
    at.selectbox[0].select(sektor).run()          # seperti pengguna mengklik pilihan
di bilah samping
    nilai = {m.label: m.value for m in at.metric}
     grafik = len(at.get("vega_lite_chart"))          # st.bar_chart = elemen grafik
Vega-Lite
    peta = len(at.get("deck_gl_json_chart"))       # st.map = elemen peta deck.gl
   print(f"{sektor:8} unit={nilai['Jumlah unit']} rata-rata={nilai['Rata-rata nilai']}
"
                f"tertinggi={nilai['Nilai  tertinggi']}  grafik={grafik}  peta={peta}
baris={len(at.dataframe[0].value)}")

# Memilih sesuatu yang tidak ada dalam daftar gagal ketika aplikasi dijalankan ulang
try:
    at.selectbox[0].select("empang").run()
except ValueError as e:
    print("pilihan tidak sah ditolak:", type(e).__name__)
```

```
hutan    unit=3 rata-rata=35.2 tertinggi=45.2 grafik=1 peta=1 baris=3
kolam    unit=3 rata-rata=369.3 tertinggi=432.4 grafik=1 peta=1 baris=3
kandang  unit=3 rata-rata=383.8 tertinggi=412.5 grafik=1 peta=1 baris=3
pilihan tidak sah ditolak: ValueError
```

`at.selectbox[0].select("kolam").run()` adalah "pengguna memilih kolam, skrip berjalan ulang". Hasilnya cocok dengan hitungan tangan: kolam punya 3 unit, rata-rata (432,4 + 380,0 + 295,5) / 3 = **369,3**, tertinggi **432,4**. Grafik batang dan peta muncul sebagai satu elemen masing-masing. Memilih sesuatu yang tidak ada di daftar (`empang`) dijawab `ValueError` saat aplikasi dijalankan ulang.

> **i** INGAT
>
> `AppTest` membuktikan bahwa logika aplikasi benar dan tidak ada pengecualian. Ia **tidak** membuktikan bahwa grafik tampak cantik. Grafik dan peta hanya terlihat sebagai elemen bertipe tertentu (`vega_lite_chart`, `deck_gl_json_chart`); bentuk di layar tetap harus dilihat manusia.

Gradio: demo fungsi dan model

Gradio punya tujuan sedikit berbeda dari Streamlit: membungkus **sebuah fungsi** dengan kotak masukan dan keluaran otomatis. Sangat disukai untuk memamerkan **model**: pengguna mengetik sesuatu, model menjawab. Di sini "model" kita sengaja sederhana: rumus taksiran bobot ikan dari panjangnya, **berupa contoh fiktif**, bukan hasil penelitian.

```
# Bab 43 - Demo Gradio: cari unit dan "model" taksir bobot ikan (modul bantu)
import gradio as gr

from data_unit import baca_unit

SEKTOR = ["semua", "hutan", "kebun", "sawah", "kolam", "kandang"]


def cari_unit(sektor, nilai_minimal):
    """Mengembalikan tabel unit dan kalimat ringkasan."""
    terpilih = [
        [u["id"], u["sektor"], u["unit"], u["nilai"], u["satuan"]]
        for u in baca_unit()
        if sektor in ("semua", u["sektor"]) and u["nilai"] >= nilai_minimal
    ]
    return terpilih, f"{len(terpilih)} unit cocok"


def taksir_bobot(panjang_cm):
      """Pengganti 'model': bobot = 0.01 x panjang^3 (rumus contoh FIKTIF, bukan
hasil penelitian)."""
    return round(0.01 * panjang_cm ** 3, 1)


antarmuka_cari = gr.Interface(
    fn=cari_unit,
    inputs=[gr.Dropdown(SEKTOR, value="semua", label="Sektor"),
            gr.Slider(0, 1000, value=0, step=10, label="Nilai minimal")],
       outputs=[gr.Dataframe(headers=["id", "sektor", "unit", "nilai", "satuan"],
label="Unit"),
             gr.Textbox(label="Ringkasan")],
    api_name="cari",
    flagging_mode="never",
)
antarmuka_taksir = gr.Interface(
    fn=taksir_bobot,
    inputs=gr.Number(value=25, label="Panjang ikan (cm)"),
    outputs=gr.Number(label="Taksiran bobot (g)"),
    api_name="taksir",
    flagging_mode="never",
)
demo = gr.TabbedInterface([antarmuka_cari, antarmuka_taksir], ["Cari unit", "Taksir
```

```
bobot"],
                          title="Demo unit usaha (data fiktif)")

if __name__ == "__main__":
    demo.launch()          # alamat lokal saja; share=True TIDAK dipakai
```

Dua fungsi, dua `gr.Interface`, digabung dalam tab. `api_name` memberi nama pada tiap titik akses supaya bisa dipanggil dari program lain. Ada dua cara menguji:

```
# Bab 43 - Gradio: fungsinya diuji langsung, lalu aplikasinya dinyalakan di komputer
sendiri
import contextlib
import io
import os

os.environ["GRADIO_ANALYTICS_ENABLED"] = "False"     # jangan kirim statistik pemakaian
os.environ["HF_HUB_OFFLINE"] = "1"                  # jangan menghubungi huggingface.co

from gradio_client import Client

from app_gradio import cari_unit, demo, taksir_bobot

# 1. Fungsi biasa: tidak perlu server untuk mengujinya
tabel, ringkasan = cari_unit("kolam", 300)
print("langsung    :", ringkasan, "->", [baris[2] for baris in tabel])
print("langsung    :", taksir_bobot(25), "g untuk ikan 25 cm")

# 2. Nyalakan server lokal, lalu panggil lewat klien resmi (seperti pengguna lewat
jaringan)
demo.launch(server_name="127.0.0.1",      prevent_thread_lock=True,      share=False,
quiet=True)
try:
    klien = Client(demo.local_url, verbose=False)
    data, teks = klien.predict("kandang", 0, api_name="/cari")
    print("klien cari  :", teks, "->", [b[2] for b in data["data"]])
   print("klien taksir:", klien.predict(30, api_name="/taksir"), "g untuk ikan 30 cm")
finally:
   with contextlib.redirect_stdout(io.StringIO()):   # menyembunyikan pesan nomor port
        demo.close()
print("server ditutup")
```

```
langsung    : 2 unit cocok -> ['Kolam A', 'Kolam B']
langsung    : 156.2 g untuk ikan 25 cm
klien cari  : 3 unit cocok -> ['Kandang 1', 'Kandang 2', 'Kandang 3']
klien taksir: 270.0 g untuk ikan 30 cm
server ditutup
```

1. **Panggil fungsinya langsung**: 2 unit kolam bernilai ≥ 300 (Kolam A dan B), dan $0{,}01 \times 25^3 = 156{,}25$, yang dibulatkan Python menjadi 156,2 g.
2. **Nyalakan server lokal lalu panggil lewat `gradio_client`**, klien resmi: seperti pengguna jarak jauh. 3 unit kandang cocok, dan ikan 30 cm: $0{,}01 \times 30^3 = 270{,}0$ g.

Tiga catatan keamanan dan privasi:

- Kita **tidak memakai `share=True`**. Menurut dokumentasi Gradio, opsi itu membuat **tautan publik sementara** (berlaku sekitar satu pekan) yang meneruskan lalu lintas ke komputer Anda; siapa pun yang punya tautan bisa memakai aplikasi, sementara komputer Anda yang bekerja. Jangan lakukan itu pada data yang bukan untuk umum.
- Saat menguji, kami memantau sambungan jaringan dan menemukan bahwa **Gradio menghubungi `huggingface.co`** bila tidak dicegah. Kami tidak menyelidiki tujuannya. Dengan `HF_HUB_OFFLINE=1` (dan `GRADIO_ANALYTICS_ENABLED=False` untuk statistik pemakaian), tidak ada lagi sambungan ke luar komputer yang terdeteksi. Bila Anda memakai Gradio untuk data sensitif, periksa pengaturan ini.

- `demo.launch()` tanpa argumen hanya melayani komputer Anda sendiri.

Dash: dasbor yang sepenuhnya Anda atur

Dash (dari Plotly) lebih "dari bawah": Anda menyusun **tata letak** (kerangka halaman dari komponen) dan **callback** (fungsi yang dipanggil bila sebuah komponen berubah). Lebih banyak kode daripada Streamlit, tetapi kendalinya lebih halus.

```
# Bab 43 - Dasbor Dash (modul bantu; dijalankan dengan: python app_dash.py)
import pandas as pd
import plotly.express as px
from dash import Dash, Input, Output, dcc, html

from data_unit import baca_unit

SEKTOR = ["hutan", "kebun", "sawah", "kolam", "kandang"]
unit = pd.DataFrame(baca_unit())

app = Dash(__name__)
app.layout = html.Div([
    html.H1("Dasbor Dash (data fiktif)"),
    dcc.Dropdown(id="sektor", options=SEKTOR, value="kolam", clearable=False),
    html.Div(id="ringkasan"),
    dcc.Graph(id="grafik"),
])


@app.callback(Output("grafik",     "figure"),     Output("ringkasan",     "children"),
Input("sektor", "value"))
def perbarui(sektor):
    bagian = unit[unit["sektor"] == sektor]
    gambar = px.bar(bagian, x="unit", y="nilai", title=f"Sektor {sektor}")
      return  gambar,  f"{len(bagian)}  unit,  rata-rata  {bagian['nilai'].mean():.1f}
{bagian['satuan'].iloc[0]}"


if __name__ == "__main__":
    app.run(debug=False)       # http://127.0.0.1:8050
```

Dekorator `@app.callback` berkata: *"bila nilai `sektor` berubah, panggil fungsi ini dan taruh hasilnya di `grafik.figure` dan `ringkasan.children`."* Dash berdiri di atas Flask, sehingga kita bisa memakai **klien uji Flask** tanpa peramban:

```
# Bab 43 - Dash: menguji dasbor lewat klien uji Flask, tanpa peramban
from app_dash import app, perbarui

klien = app.server.test_client()          # Dash berdiri di atas Flask

halaman = klien.get("/")
print("halaman  utama:",  halaman.status_code,  "|  memuat  penyaji  Dash:",  b"_dash-
renderer" in halaman.data)

tata_letak = klien.get("/_dash-layout").get_json()
anak = tata_letak["props"]["children"]
print("tata letak   :", [(a["type"], a["props"].get("id")) for a in anak])

# Meniru permintaan yang dikirim peramban saat pengguna mengganti sektor
isi = {
    "output": "..grafik.figure...ringkasan.children..",
   "outputs": [{"id": "grafik", "property": "figure"}, {"id": "ringkasan", "property":
"children"}],
```

```
    "inputs": [{"id": "sektor", "property": "value", "value": "sawah"}],
    "changedPropIds": ["sektor.value"],
}
jawab = klien.post("/_dash-update-component", json=isi)
print("callback     :", jawab.status_code, jawab.get_json()["response"]["ringkasan"]
["children"])

# Fungsinya juga bisa dipanggil langsung seperti fungsi biasa
print("langsung     :", perbarui("kandang")[1])
```

```
halaman utama: 200 | memuat penyaji Dash: True
tata letak   : [('H1', None), ('Dropdown', 'sektor'), ('Div', 'ringkasan'), ('Graph',
'grafik')]
callback     : 200 2 unit, rata-rata 1115.2 kg
langsung     : 3 unit, rata-rata 383.8 kg
```

Halaman utama menjawab 200 dan memuat penyaji Dash. Tata letak berisi judul `H1`, daftar pilihan `sektor`, `ringkasan`, dan `grafik`. Kita lalu mengirim permintaan yang sama seperti yang dikirim peramban saat pengguna memilih **sawah**: jawabannya "2 unit, rata-rata 1115.2 kg", cocok dengan (1250,0 + 980,5) / 2 = 1115,25 yang dibulatkan satu desimal. Fungsi callback juga bisa dipanggil langsung.

> TIPS
>
> Pegangan memilih: **Streamlit** untuk dasbor cepat dan prototipe; **Gradio** untuk memamerkan satu fungsi atau model; **Dash** bila tata letak dan interaksi perlu diatur rinci. Untuk dasbor internal sederhana, mulailah dari Streamlit.

Peta interaktif di dasbor

Di Bab 38 sampai 40 Anda membuat peta. Untuk peta di dasbor, **folium** membungkus pustaka peta JavaScript **Leaflet**; **pydeck** membungkus **deck.gl** untuk lapisan titik atau data besar. Keduanya menghasilkan HTML.

```
# Bab 43 - Modul bantu: peta interaktif lokasi unit usaha dengan folium (koordinat
FIKTIF)
import folium

WARNA = {"hutan": "darkgreen", "kebun": "orange", "sawah": "green", "kolam": "blue",
"kandang": "red"}


def buat_peta(unit):
    """Satu penanda per unit, diberi warna menurut sektor dan keterangan saat diklik."""
    tengah = [sum(u["lat"] for u in unit) / len(unit), sum(u["lon"] for u in unit) /
len(unit)]
    peta = folium.Map(location=tengah, zoom_start=13, tiles="OpenStreetMap")
    for u in unit:
        folium.Marker(
            [u["lat"], u["lon"]],
            tooltip=f'{u["unit"]} ({u["sektor"]})',
            popup=f'{u["unit"]}: {u["nilai"]} {u["satuan"]}',
            icon=folium.Icon(color=WARNA[u["sektor"]]),
        ).add_to(peta)
    return peta
```

```
# Bab 43 - Peta interaktif: folium (Leaflet) dan pydeck (deck.gl), disimpan sebagai
berkas HTML
from pathlib import Path
```

```
import pydeck as pdk

from data_unit import baca_unit
from peta import buat_peta

unit = baca_unit()
keluar = Path(__file__).parent / "_tmp"
keluar.mkdir(exist_ok=True)

# 1. folium: satu penanda per unit
peta = buat_peta(unit)
berkas = keluar / "peta_folium.html"
peta.save(berkas)
html = berkas.read_text(encoding="utf-8")
print("folium : penanda =", html.count("L.marker("), "| rujukan Leaflet:", "leaflet"
in html.lower())
print("         berkas HTML dibuat:", berkas.stat().st_size > 5000)

# 2. pydeck: lapisan titik dengan ukuran sebanding nilai (untuk perbandingan sektor
yang sama satuannya)
kolam = [u for u in unit if u["sektor"] == "kolam"]
lapisan = pdk.Layer("ScatterplotLayer", data=kolam, get_position=["lon", "lat"],
                    get_radius="nilai", radius_scale=0.5, get_fill_color=[30, 100,
200, 160])
tampilan = pdk.ViewState(latitude=kolam[0]["lat"], longitude=kolam[0]["lon"], zoom=13)
deck = pdk.Deck(layers=[lapisan], initial_view_state=tampilan)
berkas2 = keluar / "peta_pydeck.html"
deck.to_html(str(berkas2), open_browser=False)
print("pydeck : titik kolam =", len(kolam), "| berkas HTML dibuat:",
berkas2.stat().st_size > 1000)

# Isi kedua berkas memuat pustaka peta dari internet saat DIBUKA di peramban (bukan
saat dibuat di sini)
print("pustaka dimuat dari CDN:", "https://" in html and "https://" in
berkas2.read_text(encoding="utf-8"))
```

```
folium : penanda = 15 | rujukan Leaflet: True
         berkas HTML dibuat: True
pydeck : titik kolam = 3 | berkas HTML dibuat: True
pustaka dimuat dari CDN: True
```

Ada 15 penanda (satu per unit) pada peta folium, dan 3 titik kolam pada peta pydeck. Perhatikan baris terakhir: kedua berkas **memuat pustaka petanya dari internet saat dibuka** di peramban, bukan saat dibuat. Jadi peta ini butuh sambungan internet untuk dilihat; kami tidak membukanya di sini. Streamlit menanam peta folium memakai `st.iframe(...)`. (Fungsi lama `st.components.v1.html` ditandai usang oleh Streamlit 1.65 dengan `st.iframe` sebagai penggantinya.) Dokumentasi Streamlit juga memperingatkan: **jangan menanam HTML dari sumber tak tepercaya** lewat cara ini; peta kita dibuat sendiri dari data kita.

> ⚠ AWAS
>
> Ukuran titik pada peta pydeck memakai kolom `nilai`. Itu **adil hanya untuk sektor dengan satuan sama**. Kita memilih sektor kolam (semua dalam kg). Menggambar hutan (cm) berdampingan dengan kolam (kg) pada satu skala akan menyesatkan.

HTMX: web ringan tanpa kerangka besar

Kadang yang Anda butuhkan hanya halaman yang sebagian isinya berganti tanpa memuat ulang seluruh halaman. **HTMX** adalah pustaka JavaScript kecil yang memberi HTML kemampuan itu lewat **atribut**: `hx-get` (minta ke alamat ini), `hx-trigger` (kapan), `hx-target` (isi bagian mana), `hx-swap` (cara mengganti). Servernya, di sini **Flask**, cukup mengirim **potongan HTML**, bukan JSON.

```
# Bab 43 - Web ringan dengan HTMX + Flask (modul bantu; dijalankan dengan: flask --app
app_htmx run)
from flask import Flask, abort, render_template_string, request

from data_unit import baca_unit

app = Flask(__name__)
SEKTOR = ["hutan", "kebun", "sawah", "kolam", "kandang"]

HALAMAN = """<!doctype html>
<html lang="id">
<head>
  <meta charset="utf-8"><title>Unit usaha (HTMX)</title>
     <script   src="https://cdn.jsdelivr.net/npm/htmx.org@2.0.11/dist/htmx.min.js"></
script>
</head>
<body>
  <h1>Unit usaha (data fiktif)</h1>
    <select  name="s"  hx-get="/sektor"  hx-trigger="change"  hx-target="#hasil"  hx-
swap="innerHTML">
    <option value="">- pilih sektor -</option>
    {% for s in sektor %}<option value="{{ s }}">{{ s }}</option>{% endfor %}
  </select>
  <div id="hasil">Pilih sektor di atas.</div>
</body>
</html>"""

TABEL = """<table>
  {% for u in unit %}<tr><td>{{ u.unit }}</td><td>{{ u.jenis }}</td><td>{{ u.nilai }}
{{ u.satuan }}</td></tr>
  {% endfor %}
</table>"""


@app.get("/")
def indeks():
    return render_template_string(HALAMAN, sektor=SEKTOR)


@app.get("/sektor")
def per_sektor():
    s = request.args.get("s", "")
    if s not in SEKTOR:
        abort(400, "sektor tidak dikenal")
    unit = [u for u in baca_unit() if u["sektor"] == s]
    return render_template_string(TABEL, unit=unit)      # hanya potongan HTML, bukan
satu halaman penuh
```

Pemilihan sektor mengirim `GET /sektor?s=kolam`, dan server menjawab tabel kecil yang ditaruh di `#hasil`. Skrip HTMX dimuat dari CDN jsDelivr; versi 2.0.11 adalah versi terbaru di registri npm saat ini ditulis (dibaca 2026-10-10). Pada penerapan sungguhan, Anda bisa menambahkan atribut `integrity` (*Subresource Integrity*) seperti pada contoh dokumentasi HTMX; kami tidak menyalin hash itu dari ingatan.

```
# Bab 43 - HTMX + Flask: sisi server diuji dengan klien uji (JavaScript htmx TIDAK
dijalankan di sini)
from app_htmx import app

klien = app.test_client()

halaman = klien.get("/").get_data(as_text=True)
print("halaman utama:", len(halaman.splitlines()), "baris")
for atribut in ("hx-get=", "hx-trigger=", "hx-target=", "htmx.org@2.0.11"):
    print(f"  memuat {atribut:18}", atribut in halaman)

# Yang dikirim htmx saat pengguna memilih: GET /sektor?s=kolam dengan header HX-Request
jawab = klien.get("/sektor?s=kolam", headers={"HX-Request": "true"})
potongan = jawab.get_data(as_text=True)
print("\nGET /sektor?s=kolam ->", jawab.status_code, "| potongan HTML, bukan halaman
penuh:", "<html" not in potongan)
print(potongan)

print("sektor aneh ->", klien.get("/sektor?s=empang").status_code)
print("tanpa sektor ->", klien.get("/sektor").status_code)
```

```
halaman utama: 15 baris
  memuat hx-get=            True
  memuat hx-trigger=        True
  memuat hx-target=         True
  memuat htmx.org@2.0.11    True

GET /sektor?s=kolam -> 200 | potongan HTML, bukan halaman penuh: True
<table>
  <tr><td>Kolam A</td><td>nila</td><td>432.4 kg</td></tr>
  <tr><td>Kolam B</td><td>lele</td><td>380.0 kg</td></tr>
  <tr><td>Kolam C</td><td>nila</td><td>295.5 kg</td></tr>

</table>
sektor aneh -> 400
tanpa sektor -> 400
```

Yang terbukti: halaman memuat atribut HTMX dan skrip versi 2.0.11; endpoint `/sektor` menjawab potongan HTML (bukan halaman penuh) berisi tiga kolam; sektor aneh atau kosong ditolak dengan kode 400. **Yang tidak teruji:** JavaScript HTMX di peramban, yaitu bahwa memilih sektor benar-benar mengganti isi `#hasil`. Itu butuh peramban dan kami tidak membukanya.

FastAPI: pintu untuk program lain

Dasbor adalah pintu untuk **manusia**. **API** adalah pintu untuk **program lain**: aplikasi ponsel, skrip lain, atau dasbor di tempat lain yang meminta data dalam bentuk JSON. Di Bab 36 Anda mengenal FastAPI; kini kita memakainya untuk data semua unit usaha, dengan model Pydantic dari Bab 34.

```
# Bab 43 - API FastAPI untuk data semua unit usaha (modul bantu)
from typing import Literal

from fastapi import FastAPI, HTTPException
from pydantic import BaseModel

from data_unit import baca_unit, ringkas_per_sektor

Sektor = Literal["hutan", "kebun", "sawah", "kolam", "kandang"]


class Unit(BaseModel):
```

```
    id: str
    sektor: Sektor
    unit: str
    jenis: str
    lat: float
    lon: float
    nilai: float
    satuan: str


class Ringkasan(BaseModel):
    sektor: Sektor
    jumlah: int
    total: float
    rata: float
    satuan: str


app = FastAPI(title="API unit usaha (data fiktif)")


@app.get("/unit", response_model=list[Unit])
def daftar_unit(sektor: Sektor | None = None):
    return [u for u in baca_unit() if sektor is None or u["sektor"] == sektor]


@app.get("/unit/{id_unit}", response_model=Unit)
def satu_unit(id_unit: str):
    for u in baca_unit():
        if u["id"] == id_unit:
            return u
    raise HTTPException(status_code=404, detail=f"unit {id_unit} tidak ada")


@app.get("/ringkasan", response_model=list[Ringkasan])
def ringkasan():
    return [{"sektor": s, **r} for s, r in ringkas_per_sektor(baca_unit()).items()]
```

```
# Bab 43 - FastAPI: API data semua unit usaha, diuji dengan TestClient
from fastapi.testclient import TestClient

from api_unit import app

klien = TestClient(app)

semua = klien.get("/unit").json()
print("GET /unit                 :", len(semua), "unit; pertama =", semua[0]["id"],
semua[0]["unit"])

kolam = klien.get("/unit", params={"sektor": "kolam"}).json()
print("GET /unit?sektor=kolam :", [u["unit"] for u in kolam])

satu = klien.get("/unit/D2")
print("GET /unit/D2           :", satu.status_code, satu.json()["nilai"], satu.json()
["satuan"])

hilang = klien.get("/unit/X9")
print("GET /unit/X9           :", hilang.status_code, hilang.json())

salah = klien.get("/unit", params={"sektor": "empang"})
```

```
print("GET   /unit?sektor=empang:",   salah.status_code,   salah.json()["detail"][0]
["type"])

print("GET /ringkasan        :", [(r["sektor"], r["jumlah"], round(r["rata"], 1)) for
r in klien.get("/ringkasan").json()])
print("dokumentasi  otomatis      :",  klien.get("/docs").status_code,  "|  skema:",
klien.get("/openapi.json").status_code)
```

```
GET /unit              : 15 unit; pertama = H1 Plot A
GET /unit?sektor=kolam : ['Kolam A', 'Kolam B', 'Kolam C']
GET /unit/D2           : 200 412.5 kg
GET /unit/X9           : 404 {'detail': 'unit X9 tidak ada'}
GET /unit?sektor=empang: 422 literal_error
GET /ringkasan       : [('hutan', 3, 35.2), ('kebun', 4, 257.5), ('sawah', 2, 1115.2),
('kolam', 3, 369.3), ('kandang', 3, 383.8)]
dokumentasi otomatis   : 200 | skema: 200
```

`TestClient` memanggil API tanpa menyalakan server. Hasilnya menunjukkan: 15 unit; tiga kolam bila disaring; `D2` ditemukan (412,5 kg); `X9` menghasilkan **404** dengan pesan jelas; sektor `empang` menghasilkan **422**, galat validasi otomatis karena bukan salah satu isi `Literal`; dan FastAPI membuatkan dokumentasi interaktif (`/docs`) dan skema (`/openapi.json`) tanpa Anda menulisnya.

Menyajikan aplikasi kepada pengguna

Sejauh ini semuanya berjalan di `127.0.0.1`, alamat yang hanya bisa dijangkau **komputer ini**. Agar orang lain memakainya, aplikasi harus berjalan di mesin yang **bisa dijangkau** mereka. Mari lihat server sungguhan pertama, **uvicorn**, yang menjalankan API kita lewat HTTP asli:

```
# Bab 43 - Menyajikan aplikasi: server sungguhan (uvicorn) di komputer sendiri +
berkas penyajian
import socket
import threading
import time
from pathlib import Path

import httpx
import uvicorn

from api_unit import app

# Cari port yang sedang kosong, lalu nyalakan server di thread terpisah
with socket.socket() as s:
    s.bind(("127.0.0.1", 0))
    port = s.getsockname()[1]

server    =    uvicorn.Server(uvicorn.Config(app,    host="127.0.0.1",    port=port,
log_level="warning"))
thread = threading.Thread(target=server.run, daemon=True)
thread.start()
while not server.started:
    time.sleep(0.05)

try:
    jawab = httpx.get(f"http://127.0.0.1:{port}/ringkasan", timeout=5)
   print("lewat HTTP sungguhan:", jawab.status_code, "|", len(jawab.json()), "sektor")
finally:
    server.should_exit = True          # minta server berhenti dengan rapi
    thread.join(timeout=10)
print("server sudah berhenti:", not thread.is_alive())
```

```
# Berkas yang biasanya menyertai aplikasi saat dipindah ke server lain
keluar = Path(__file__).parent / "_tmp" / "penyajian"
keluar.mkdir(parents=True, exist_ok=True)
(keluar                                                                    /
"requirements.txt").write_text("fastapi\nuvicorn\nstreamlit\npandas\nfolium\n",
encoding="utf-8")
perintah = {
    "API (FastAPI)": "uvicorn api_unit:app --host 127.0.0.1 --port 8000",
    "Dasbor (Streamlit)": "streamlit run app_streamlit.py",
    "Demo (Gradio)": "python app_gradio.py",
    "Dasbor (Dash)": "python app_dash.py",
    "Web ringan (Flask)": "flask --app app_htmx run",
}
print("\nPerintah menyalakan tiap aplikasi (dijalankan SENDIRI di terminal, tidak
di sini):")
for nama, p in perintah.items():
    print(f"  {nama:20} {p}")
```

```
lewat HTTP sungguhan: 200 | 5 sektor
server sudah berhenti: True

Perintah menyalakan tiap aplikasi (dijalankan SENDIRI di terminal, tidak di sini):
  API (FastAPI)        uvicorn api_unit:app --host 127.0.0.1 --port 8000
  Dasbor (Streamlit)   streamlit run app_streamlit.py
  Demo (Gradio)        python app_gradio.py
  Dasbor (Dash)        python app_dash.py
  Web ringan (Flask)   flask --app app_htmx run
```

Server dinyalakan di port yang sedang kosong, dijawab 200 dengan 5 sektor, lalu dihentikan dengan rapi. Daftar di bawahnya adalah perintah penyalaan tiap aplikasi di bab ini, yang **Anda jalankan sendiri di terminal**; tidak ada yang dijalankan di sini.

Bagaimana memindahkannya agar orang lain bisa memakai? Berikut pilihan yang disebut **dokumentasi resmi masing-masing**. **Kami tidak mendaftar, mengunggah, atau mencoba satu pun**; syarat, harga, dan batasan bisa berubah, jadi baca halaman resminya sebelum memutuskan.

Aplikasi	Pilihan menurut dokumentasi
Streamlit	**Streamlit Community Cloud** (dijelaskan sebagai hosting gratis untuk aplikasi nonkomersial, pribadi, dan pendidikan), **Snowflake**, atau platform lain lewat tutorial
Gradio	`share=True` (tautan sementara dari komputer Anda), **Hugging Face Spaces** (tautan permanen), atau **dipasang pada aplikasi FastAPI** milik sendiri dengan `gr.mount_gradio_app()`
Dash	**Plotly Cloud** (terkelola) atau **Dash Enterprise** (untuk organisasi)
FastAPI, Flask	server milik sendiri atau sewaan, menjalankan `uvicorn` seperti di atas

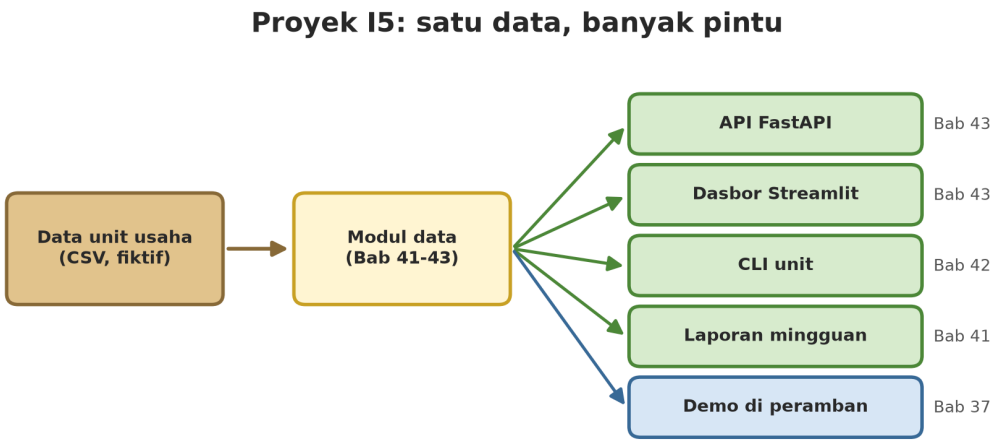
Catatan dari dokumentasi Hugging Face yang perlu dibaca teliti: saat ditulis, ruang (*Space*) bertipe Gradio atau Docker butuh paket berbayar untuk dibuat, sedangkan ruang statis gratis, dan ruang di perangkat keras gratis akan "tidur" bila lama tak dipakai. Dokumentasinya juga menyarankan menaruh rahasia (kunci, token) sebagai **secret** pengaturan, **bukan** menuliskannya di kode.

⚠ AWAS

Sebelum menyajikan apa pun ke publik, tanya diri Anda: (1) apakah datanya boleh dilihat semua orang? Dasbor tanpa kata sandi terbuka bagi siapa pun yang punya alamatnya; (2) apakah ada rahasia di dalam kode? Kunci dan kata sandi **tidak boleh** ada di kode atau repositori; (3) siapa yang merawatnya bila rusak? Bab-bab Seri A1 membahas keamanan dan penerapan (*deployment*) secara serius.

Proyek Bersambung I5 (penutup): satu data, banyak pintu

Proyek Seri I5 kini lengkap: data unit usaha dibaca oleh satu modul, lalu dibuka lewat banyak pintu.



Gambar 0.2: Proyek I5: satu data, banyak pintu.

- **API FastAPI** (bab ini) untuk program lain.
- **Dasbor Streamlit per sektor** (bab ini) untuk manusia.
- **CLI `unit`** (Bab 42) untuk terminal dan otomasi.
- **Laporan mingguan** dengan scraping yang sopan (Bab 41).
- **Demo di peramban** tanpa server (Bab 37), dan peta (Bab 38 sampai 40).

Karena semua pintu membaca data yang sama, angkanya **harus sama**. Skrip penutup memeriksa itu: rata-rata per sektor dihitung langsung dari modul, lewat API, dan lewat dasbor, lalu dibandingkan.

```
# Bab 43 - Proyek bersambung I5 (penutup): satu data, tiga jalur akses, hasilnya
harus sama
import logging

from fastapi.testclient import TestClient
from streamlit.testing.v1 import AppTest

from api_unit import app
from data_unit import baca_unit, ringkas_per_sektor

logging.getLogger("streamlit.runtime.scriptrunner_utils.script_run_context").setLevel(logging.ERROR)

langsung = ringkas_per_sektor(baca_unit())                    # jalur 1: modul data
lewat_api = {r["sektor"]: r for r in TestClient(app).get("/ringkasan").json()}    #
jalur 2: API

dasbor = AppTest.from_file("app_streamlit.py", default_timeout=30).run()
```

```
print(f"{'sektor':8} {'unit':>4} {'rata-rata (modul)':>18} {'API':>8} {'dasbor':>8}
cocok")
semua_cocok = True
for sektor, r in langsung.items():
     dasbor.selectbox[0].select(sektor).run()                          # jalur 3:
dasbor Streamlit
    metrik = {m.label: m.value for m in dasbor.metric}
    cocok = (f"{r['rata']:.1f}" == f"{lewat_api[sektor]['rata']:.1f}" == metrik["Rata-
rata nilai"]
             and str(r["jumlah"]) == metrik["Jumlah unit"])
    semua_cocok &= cocok
        print(f"{sektor:8}  {r['jumlah']:>4}  {r['rata']:>18.1f}  {lewat_api[sektor]
['rata']:>8.1f} "
          f"{metrik['Rata-rata nilai']:>8}  {'ya' if cocok else 'TIDAK'}")

print("\nsemua jalur konsisten:", semua_cocok)
```

```
sektor   unit  rata-rata (modul)      API   dasbor  cocok
hutan       3              35.2     35.2     35.2  ya
kebun       4             257.5    257.5    257.5  ya
sawah       2            1115.2   1115.2   1115.2  ya
kolam       3             369.3    369.3    369.3  ya
kandang     3             383.8    383.8    383.8  ya

semua jalur konsisten: True
```

Kelima sektor cocok di ketiga jalur (kolom terakhir "ya", dan `semua jalur konsisten: True`). Pemeriksaan semacam ini adalah tes yang murah dan menyelamatkan: bila suatu hari satu jalur mulai menghitung lain, Anda tahu **sebelum pengguna tahu**.

Latihan

1. Pada `app_streamlit.py`, tambahkan `st.slider` "Nilai minimal" yang menyaring tabel. Tambahkan pemeriksaan di `02_streamlit_interaksi.py` yang menggeser slider dan membaca jumlah baris tabel.
2. Pada `api_unit.py`, tambahkan titik akses `GET /sektor` yang mengembalikan daftar nama sektor, lalu uji dengan `TestClient`.
3. Pada `app_dash.py`, tambahkan komponen kedua (misalnya `dcc.RadioItems` untuk memilih urutan naik atau turun) dan callback-nya. Uji lewat klien uji Flask.
4. **Tantangan:** buat `app_gradio.py` menerima nomor unit (`D1`, `L2`, …) dan mengembalikan unit itu dengan memanggil API FastAPI Anda lewat `TestClient` (jangan lewat jaringan). Tampilkan galat yang ramah bila nomornya tidak ada.

Ringkasan Bab

- **Streamlit**: skrip dari atas ke bawah, dijalankan ulang tiap interaksi; uji dengan **`AppTest`** tanpa peramban.
- **Gradio**: membungkus fungsi/model dengan kotak otomatis; uji fungsinya langsung atau lewat **`gradio_client`**. Jangan pakai `share=True` untuk data yang bukan untuk umum.
- **Dash**: tata letak + callback, berdiri di atas Flask; uji dengan klien uji.
- **folium** dan **pydeck** menghasilkan peta HTML interaktif yang memuat pustakanya dari internet saat dibuka.
- **HTMX** + Flask: server mengirim potongan HTML; yang teruji di sini hanya sisi server.
- **FastAPI**: API JSON dengan validasi dan dokumentasi otomatis; uji dengan `TestClient`.
- Menyajikan = menjalankan di mesin yang bisa dijangkau pengguna; pertimbangkan data, rahasia, dan perawatan lebih dulu.

Penutup Seri I5

Selamat! Anda menyelesaikan Seri I5, dan itu berarti **Tingkat Intermediate (Jilid 2) tamat**. Dari kode yang rapi dan terawat (Seri I1), data dan internet (I2), statistik dan pemodelan (I3), bahasa Python tingkat lanjut (I4), sampai aplikasi pertama (I5), Anda sekarang bisa:

- menulis kode yang rapi, diuji, dan terkontrol versinya;
- mengambil, membersihkan, dan menganalisis data, termasuk statistik dasar;
- memakai generator, decorator, type hint, Pydantic, dan async;
- membuat API web, memproses data spasial, dan menampilkannya di peta;
- mengotomasi tugas berulang, membuat alat baris perintah, dan menyajikan data lewat dasbor.

Tetapi aplikasi Anda masih **berjalan di komputer sendiri**. Agar ia **cepat**, **aman**, **bisa di-server** untuk banyak pengguna, **menangani data besar**, dan **cerdas**, Anda perlu masuk ke Tingkat **Advanced**: rekayasa perangkat lunak profesional, data skala besar, machine learning, sampai AI.

Selanjutnya, di Bab 44: performa. Kita belajar mengukur dan mempercepat kode (profiling, struktur data yang tepat, vektorisasi, dan caching).

Naik tingkat: dari program kecil ke sistem nyata

- Kode rapi: OOP, pengujian, Git, dan kualitas kode
- Data dan internet: API, regex, database, NumPy, pandas
- Statistika, regresi, dan analisis data
- Python lanjut, web, peta (GIS), dan dasbor

Jilid 2 untuk Anda yang sudah menulis program kecil dan ingin menggarap data, peta, dan aplikasi web. Contoh tetap dari hutan, kebun, sawah, kolam, dan kandang, dengan data tiruan yang aman dijalankan.

Disusun oleh

Badar Mubarok Yogaswara