

SERI PYTHON DARI NOL

Python dari Nol sampai **Sangat Advanced**

Jilid 1 • Basic

Bab 1–15



Badar Mubarak Yogaswara

Daftar Isi

Python dari Nol sampai Sangat Advanced	9
Tentang Buku Ini	11
Untuk siapa buku ini?	11
Apa yang dibahas, dan apa yang tidak	11
Asumsi tentang Anda	11
Cara membaca tulisan di buku ini	11
Ikon di buku ini	12
Peta seri: 13 seri dalam tiga tingkat	12
Pembagian tiga jilid	13
Cara membaca: berurutan atau melompat	14
Di luar buku	14
Versi Python yang dipakai	14
Catatan jujur: apa yang diuji dan tidak diuji	14
Siap?	15
Bab 1 – Komputer dan Program: Pegawai Super Patuh yang Tidak Punya Akal	17
Komputer itu pegawai yang sangat patuh	17
Tiga tugas pokok: masukan, proses, keluaran	17
Kenali “isi perut” komputer lewat analogi gudang	18
Apa itu program?	19
File dan folder: sistem laci komputer	19
Kenapa Python?	20
Latihan	20
Ringkasan Bab	20
Bab 2 – Berpikir Seperti Pemrogram: Menulis Resep untuk Komputer	21
Algoritma: resep yang tidak boleh ambigu	21
Tiga bahan dasar semua program	21
Flowchart: menggambar algoritma	22
Pecah masalah besar menjadi langkah kecil	24
Mengetes algoritma di kepala	24
Latihan	24
Ringkasan Bab	24

Bab 3 – Mengenal Terminal: Ngobrol dengan Komputer Lewat Ketikan	25
Terminal itu seperti berkirim pesan	25
Membuka terminal	25
Membaca “prompt”	25
Enam perintah dasar	26
Praktik: membuat folder belajar	26
Kalau salah ketik: pesan error	27
Latihan	27
Ringkasan Bab	28
Bab 4 – Mengenal Python dan Memasanginya: Program Pertama Anda	29
Apa itu Python?	29
Memasang Python 3.14 di Windows	29
Memeriksa pemasangan	30
Mencoba Python langsung: mode interaktif	30
Program pertama Anda	31
Proyek Bersambung B1: “Halo, Lahan!”	31
Kalau error: tenang, itu wajar	32
Jebakan 1: salah ketik nama perintah	32
Jebakan 2: tanda kutip penutup hilang	32
Latihan	33
Ringkasan Bab	33
Bab 5 – Variabel dan Tipe Data: Memberi Nama pada Segala Sesuatu	35
Variabel itu seperti label di karung	35
Aturan memberi nama (supaya Python tidak ngambek)	36
Empat tipe data yang akan Anda temui setiap hari	37
Mencetak kalimat rapi dengan f-string	37
Jebakan klasik: angka yang ternyata teks	38
Salah ketik? Python ikut membantu	40
Proyek Bersambung B2: data lapangan pertama	40
Latihan	41
Ringkasan Bab	41
Bab 6 – Operator dan Input/Output: Berhitung, Membandingkan, dan Ngobrol dengan Program	43
Berhitung seperti kalkulator	43
Tiga cara membagi	44
Membandingkan nilai	45
Menggabungkan syarat: and, or, not	46
Urutan pengerjaan operator	46

Kejutan angka desimal	47
Berbicara dengan program: print() dan input()	48
Proyek Bersambung B2: kalkulator kolam	48
Latihan	50
Ringkasan Bab	50
Bab 7 — Kondisional: Mengajari Program Mengambil Keputusan	51
if: kerjakan hanya bila syaratnya benar	51
else: jalan kedua	51
elif: banyak jalan	52
Menggabungkan syarat	53
Menjaga program dari error	53
match dan case: memilih satu dari banyak nilai	54
Jebakan: lupa menjorok	54
Proyek Bersambung B2: status kolam	55
Latihan	55
Ringkasan Bab	55
Bab 8 — Perulangan: Menyuruh Program Mengerjakan Hal yang Sama Berkali-kali	57
for: untuk setiap isi daftar	57
range: deretan angka	57
Pola penampung: menjumlahkan lewat perulangan	58
while: ulangi selama syaratnya benar	59
break dan continue	59
Perulangan di dalam perulangan	60
Jebakan: range tidak mau desimal	60
Proyek Bersambung B2: rekap panen kebun	60
Latihan	61
Ringkasan Bab	61
Bab 9 — Struktur Data Bawaan: Wadah untuk Data Pohon, Ikan, dan Ternak	63
list: daftar yang bisa diubah	63
Slicing: mengambil potongan	64
tuple: daftar yang dikunci	65
dict: data berlabel	65
set: kumpulan tanpa duplikat	66
Comprehension: list baru dalam satu baris	66
Proyek Bersambung B2: data pohon	67
Latihan	68
Ringkasan Bab	68

Bab 10 – Fungsi: Menulis Sekali, Memakai Berkali-kali	69
def: membuat fungsi	69
Argumen dan return	70
Nilai bawaan	70
Scope: siapa bisa melihat variabel apa	71
Mengembalikan beberapa nilai	71
Docstring dan lambda	72
Jebakan klasik: lupa return	72
Proyek Bersambung B2: pencatat data lapangan	72
Latihan	73
Ringkasan Bab	73
Penutup Seri B2	73
 Bab 11 – Modul dan Package: Meminjam Alat dari Gudang Python	 75
Tiga cara memakai import	75
math dan random	76
Tanggal dengan datetime	77
statistics dan collections	77
itertools dan pathlib	78
Membuat modul sendiri	79
if <code>__name__ == "__main__"</code>	79
Package: folder berisi modul	80
pip: meminjam dari luar gudang	80
Proyek Bersambung B3: ringkasan lima sektor	81
Latihan	82
Ringkasan Bab	82
 Bab 12 – Penanganan Error: Program yang Tidak Gampang Runtuh	 83
try dan except	83
Kenali jenis error	84
else dan finally	85
raise: memunculkan error sendiri	85
Jenis error buatan sendiri	86
Membaca traceback	86
Mencari kutu: print, assert, logging	87
Proyek Bersambung B3: tahan terhadap input salah	89
Latihan	91
Ringkasan Bab	91

Bab 13 – File dan I/O Dasar: Data yang Tidak Hilang Saat Program Ditutup	93
Menulis berkas teks	93
Membaca berkas teks	94
Mode: w, a, x	95
Mengapa with	95
pathlib untuk berkas dan folder	96
Encoding: teks yang berubah jadi simbol aneh	97
Berkas yang tidak ada	98
Proyek Bersambung B3: menyimpan dan memuat data pohon	99
Latihan	100
Ringkasan Bab	100
Bab 14 – Format Data: CSV, JSON, dan Hati-hati dengan Pickle	101
CSV: tabel sebagai teks	101
Membaca CSV: semuanya teks	102
Mengapa bukan <code>split(",")</code>	103
JSON: data bersarang	103
JSON ke berkas	104
Dua masalah JSON	105
Pickle: praktis, tetapi berbahaya	105
Proyek Bersambung B3: simpan dan muat lima sektor	107
Latihan	108
Ringkasan Bab	109
Bab 15 – Otomasi Dokumen Kantor: Dari Data Menjadi Laporan Excel dan PDF	111
Memasang pustaka	111
Excel dengan <code>openpyxl</code>	112
Grafik di dalam Excel	114
Excel dengan <code>pandas</code>	114
Word dengan <code>python-docx</code>	115
PDF: membuat dengan <code>reportlab</code>	116
PDF: membaca dan mengambil tabel	117
PDF: menggabung dan memisah	117
Email: membuat pesan	118
Email: uji ke server lokal	119
Proyek Bersambung B3: rekap lapangan menjadi laporan	120
Latihan	123
Ringkasan Bab	124
Penutup Seri B3	124
Selamat: Tingkat Basic (Jilid 1) tamat!	124

Lampiran A — Pemasangan di Windows, Linux, dan macOS . .	125
Kejujuran dulu: apa yang diuji, apa yang tidak	125
Terminal: tempat mengetik perintah	125
Peta keputusan: cara memasang yang mana?	126
Windows	126
Python install manager	126
Hasil di komputer penulis	127
Memeriksa versi	127
Folder PATH	127
Pemasangan lewat skrip (opsional)	127
Linux	128
Python dari paket distro	128
pyenv: memasang banyak versi Python	128
Membangun dari kode sumber	128
macOS	129
Pemasang dari python.org	129
Homebrew	129
uv: jalan pintas di semua sistem	129
Lingkungan virtual (venv) di tiap sistem	130
Uji nyata: membuat venv dari skrip	130
Memeriksa lingkungan Anda	131
Alat lain yang dipakai buku ini	132
Masalah umum dan solusinya	132

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 sungguh dengan **Python 3.14.8** dan keluaran yang tampil adalah hasil aslinya, kecuali bagian yang ditandai terang-terangan sebagai tidak dijalankan. Penulis: **Badar Mubarak 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 Mubarak 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

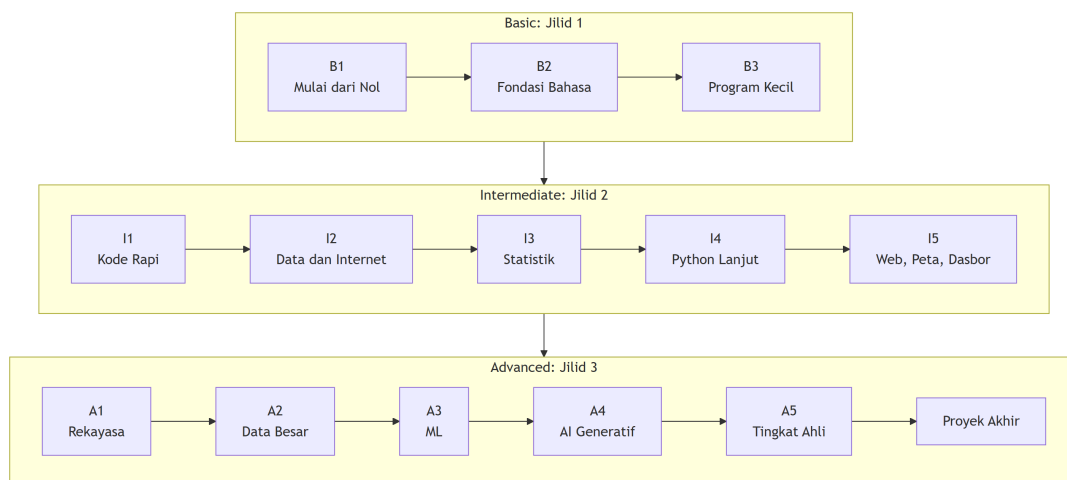
Buku ini berisi **68 bab**. Bab dikelompokkan menjadi **13 seri**, dan seri dikelompokkan menjadi **tiga tingkat**. Tiap seri punya prasyarat, satu proyek bersambung, dan “jembatan” ke seri berikutnya.

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.

Tiga jilid, satu perjalanan



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 menuliskannya 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 1 — Komputer dan Program: Pegawai Super Patuh yang Tidak Punya Akal

Seri B1 — Mulai dari Nol

Di bab ini, Anda akan:

- Memahami apa sebenarnya yang dikerjakan komputer (bocoran: hanya tiga hal)
- Mengenal bagian-bagian komputer lewat analogi gudang dan meja kerja
- Tahu apa itu program, file, folder, dan alamat file
- Siap melangkah ke Bab 2 tanpa rasa takut

Bab ini tidak punya satu pun baris kode. Tenang saja: justru di sinilah fondasi paling penting diletakkan. Orang yang paham *apa yang dilakukan komputer* akan jauh lebih mudah belajar memprogramnya.

Komputer itu pegawai yang sangat patuh

Bayangkan Anda mempekerjakan seorang pegawai gudang dengan tiga sifat aneh:

1. **Sangat cepat.** Ia bisa mengerjakan jutaan langkah dalam satu detik.
2. **Sangat teliti.** Ia tidak pernah lelah, tidak pernah bosan, dan tidak pernah salah hitung.
3. **Tidak punya akal sendiri.** Ia hanya mengerjakan *persis* apa yang Anda tuliskan. Tidak kurang, tidak lebih.

Itulah komputer. Kalau Anda menulis “pindahkan karung itu” tanpa menyebut karung yang mana, ia tidak akan menebak. Ia berhenti dan bingung (atau mengerjakan hal yang aneh).

i INGAT

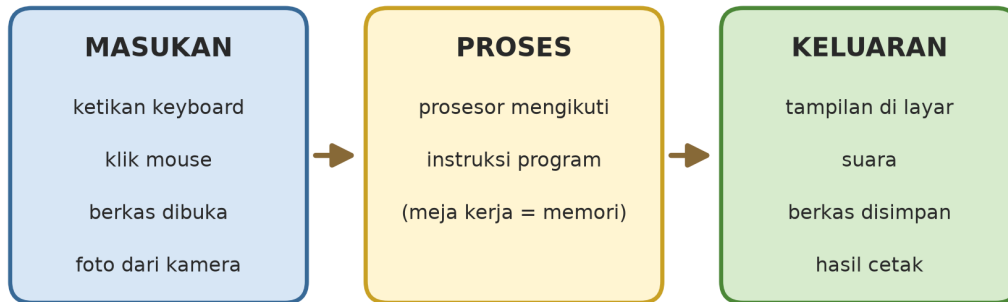
Komputer **tidak pintar**. Ia hanya **sangat cepat dan sangat patuh**. Semua “kepintaran” yang Anda lihat berasal dari instruksi yang ditulis manusia.

Kabar baiknya: karena komputer patuh, kalau hasilnya salah, hampir selalu yang keliru adalah instruksinya, bukan komputernya. Dan instruksi bisa diperbaiki.

Tiga tugas pokok: masukan, proses, keluaran

Apa pun yang dilakukan komputer, dari memutar lagu sampai menghitung gaji, selalu bisa diurai menjadi tiga tahap:

Tiga tugas pokok komputer



Program = daftar instruksi yang memberi tahu komputer apa yang harus dilakukan pada masukan

Gambar 0.1: Tiga tugas pokok komputer: menerima masukan, memprosesnya mengikuti instruksi program, lalu menghasilkan keluaran.

Tahap	Artinya	Contoh dari lapangan
Masukan (<i>input</i>)	Data yang masuk ke komputer	Anda mengetik diameter pohon 32.5
Proses	Komputer mengolah data itu	Menghitung luas penampang batang
Keluaran (<i>output</i>)	Hasil yang dikeluarkan	Angka hasil tampil di layar, atau tersimpan ke file

Jadi ketika nanti Anda belajar Python, sebenarnya Anda sedang belajar menuliskan **bagian tengah** dari bagian ini: instruksi apa yang harus dikerjakan komputer terhadap masukan.

Kenali “isi perut” komputer lewat analogi gudang

Anda tidak perlu jadi teknisi. Cukup kenal empat bagian ini:

Bagian	Analogi	Fungsinya
Prosesor (CPU)	Pegawai gudang	Menjalankan instruksi satu per satu, sangat cepat
Memori (RAM)	Meja kerja	Tempat sementara untuk barang yang <i>sedang</i> dikerjakan. Cepat dijangkau, tetapi dibersihkan saat komputer dimatikan
Penyimpanan (SSD/hard disk)	Lemari arsip	Tempat menyimpan file secara tetap. Lebih lambat dijangkau, tetapi isinya awet walau listrik mati
Layar, keyboard, mouse	Jendela dan pintu	Jalan masuk (masukan) dan jalan keluar (keluaran)

💡 TIPS

Pernah kehilangan ketikan karena lupa menekan *Save* lalu listrik padam? Itu karena pekerjaan Anda masih ada di “meja kerja” (RAM) dan belum dipindah ke “lemari arsip” (penyimpanan). **Simpan sering-sering.**

Apa itu program?

Program adalah daftar instruksi yang ditulis rapi supaya komputer bisa mengerjakan sesuatu. Mirip resep: bahan masuk, langkah dijalankan berurutan, hidangan keluar.

Hampir semua yang Anda pakai setiap hari adalah program: peramban (*browser*), kalkulator, aplikasi pesan, bahkan Windows sendiri. Tanpa program, komputer hanyalah benda diam yang hangat.

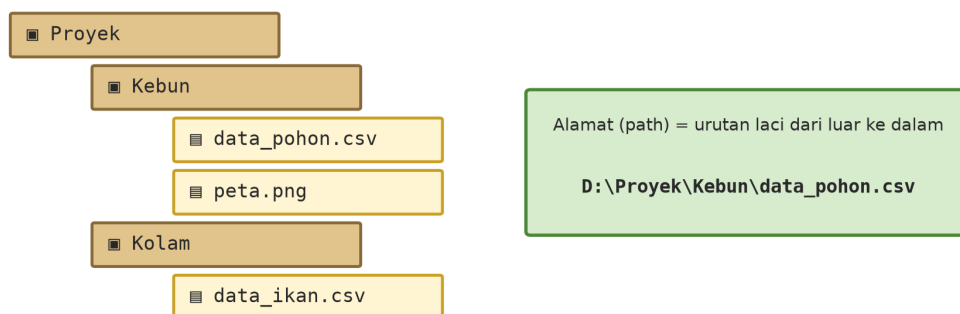
Orang yang menulis program disebut **pemrogram** (*programmer*). Dan setelah membaca buku ini, itu akan menjadi Anda.

File dan folder: sistem laci komputer

Karena penyimpanan berisi ribuan item, komputer mengaturnya dengan dua konsep sederhana:

- **File (berkas)** = satu kumpulan data yang punya nama. Bisa berupa foto, dokumen, lagu, atau (nanti) program Python Anda.
- **Folder (direktori)** = laci tempat menyimpan file dan folder lain. Karena folder boleh berisi folder lagi, hasilnya seperti laci bersarang.

Folder itu seperti laci bersarang



Gambar 0.2: Folder bersarang seperti laci. Alamat (path) sebuah file adalah urutan laci yang harus dibuka dari luar ke dalam.

Alamat file (disebut *path*) menunjukkan jalan menuju sebuah file, dari “laci terluar” sampai nama file-nya. Di Windows, contohnya:

```
D:\Proyek\Kebun\data_pohon.csv
```

Dibaca: *di drive D, buka folder Proyek, lalu folder Kebun, lalu ambil file data_pohon.csv*. Tanda \ memisahkan satu laci dari laci berikutnya.

Bagian setelah titik terakhir pada nama file, misalnya *.csv*, *.jpg*, atau *.py*, disebut **ekstensi**. Ekstensi memberi tahu komputer jenis file itu dan program mana yang cocok membukanya. Nanti file Python Anda berekstensi *.py*.

⚠️ AWAS

Windows sering **menyembunyikan ekstensi** secara bawaan, sehingga `catatan.txt` tampak hanya sebagai `catatan`. Ini menjebak pemula: Anda mengira sudah membuat `halo.py`, padahal namanya sebenarnya `halo.py.txt`. Di File Explorer, aktifkan tampilan ekstensi lewat menu **Lihat > Tampilkan > Ekstensi nama file**. Latihan 3 di bawah akan memandu Anda.

🔥 BAGIAN TEKNIS (boleh dilewati)

Di tingkat paling dalam, komputer hanya mengenal dua keadaan: arus **ada** atau **tidak ada**, ditulis sebagai **1** dan **0**. Huruf, foto, dan lagu semuanya akhirnya disimpan sebagai deretan angka 1 dan 0 yang sangat panjang. Anda **tidak perlu** menghafal ini; Python akan menyembunyikan semua detail itu dari Anda.

Kenapa Python?

Komputer hanya paham bahasa angka. Manusia hanya paham bahasa manusia. Dibutuhkan penerjemah: **bahasa pemrograman**. Python adalah salah satu bahasa pemrograman yang tulisannya paling mirip bahasa manusia, sehingga cocok untuk pemula, tetapi cukup kuat untuk dipakai para ilmuwan data. Ceritanya kita mulai di Bab 4.

Latihan

1. Sebutkan **tiga masukan** dan **tiga keluaran** yang ada pada ponsel Anda. (Petunjuk: layar sentuh itu masukan *sekaligus* keluaran.)
2. Buka File Explorer dan tulis alamat lengkap (*path*) folder `Documents` milik Anda. (Petunjuk: klik kotak alamat di bagian atas jendela.)
3. **Tantangan:** aktifkan tampilan ekstensi nama file, lalu cari satu file berekstensi `.jpg` atau `.png` di komputer Anda. Apa ekstensi file dokumen Word Anda?

Ringkasan Bab

- Komputer = pegawai **sangat cepat, sangat teliti, tetapi tidak punya akal**: ia hanya mengikuti instruksi persis.
- Semua pekerjaan komputer terurai menjadi **masukan → proses → keluaran**.
- **RAM** = meja kerja (sementara), **penyimpanan** = lemari arsip (tetap). Simpan pekerjaan sering-sering.
- **Program** = daftar instruksi. **File** = data bernama, **folder** = laci, **path** = alamat menuju file.
- Aktifkan tampilan **ekstensi** nama file supaya tidak tertipu nama file.

Selanjutnya, di Bab 2: sebelum menulis instruksi untuk pegawai yang patuh ini, kita latihan menyusun instruksi yang jelas, dimulai dari urusan sehari-hari seperti memberi pakan ikan.

Bab 2 — Berpikir Seperti Pemrogram: Menulis Resep untuk Komputer

Seri B1 — Mulai dari Nol

Di bab ini, Anda akan:

- Memahami apa itu algoritma (bocoran: resep yang sangat teliti)
- Mengenal tiga bahan dasar semua program: urutan, pilihan, pengulangan
- Membuat dan membaca flowchart sederhana
- Memecah masalah besar menjadi langkah-langkah kecil

Masih belum ada kode di bab ini. Pemrogram yang baik **berpikir dulu, mengetik kemudian**. Bab ini melatih otot berpikir itu, memakai urusan lapangan yang sudah Anda kenal.

Algoritma: resep yang tidak boleh ambigu

Algoritma adalah urutan langkah yang jelas untuk mencapai sebuah tujuan. Anda sudah sering memakainya tanpa sadar: resep masakan, petunjuk arah, cara mengisi formulir.

Bedanya dengan instruksi biasa: algoritma untuk komputer **tidak boleh ambigu**. Ingat dari Bab 1, komputer adalah pegawai yang tidak punya akal. Bandingkan:

Instruksi untuk manusia	Instruksi untuk komputer
“Beri pakan ikan secukupnya.”	“Tebar 2 kg pakan. Selama ikan masih berebut, tambah 200 g.”
“Panen kalau sudah besar.”	“Panen kalau bobot rata-rata sudah 500 g atau lebih.”
“Siram tanaman pagi-pagi.”	“Siram pukul 06.00 selama 10 menit.”

⚠ AWAS

Kata seperti *secukupnya*, *kira-kira*, *agak*, dan *kalau sudah besar* mudah dipahami manusia, tetapi **tidak bisa dikerjakan komputer**. Setiap kali menulis instruksi, tanyakan: “*Bisakah orang yang sama sekali tidak tahu apa-apa mengerjakan ini dengan benar?*”

Tiga bahan dasar semua program

Rahasia yang jarang diceritakan: program sebesar apa pun, dari kalkulator sampai aplikasi peta, tersusun dari **tiga bahan dasar** saja.

Bahan	Artinya	Contoh dari lapangan
Urutan	Langkah dikerjakan satu per satu, dari atas ke bawah	Ambil pakan → tebar pakan → catat jumlahnya
Pilihan	<i>Jika</i> syarat terpenuhi, kerjakan A; <i>kalau tidak</i> , kerjakan B	Jika pH air di bawah 6, tambahkan kapur; kalau tidak, biarkan

Bahan	Artinya	Contoh dari lapangan
Pengulangan	Ulangi langkah <i>selama</i> syarat masih berlaku	Selama ikan masih berebut pakan, tambah pakan sedikit lagi

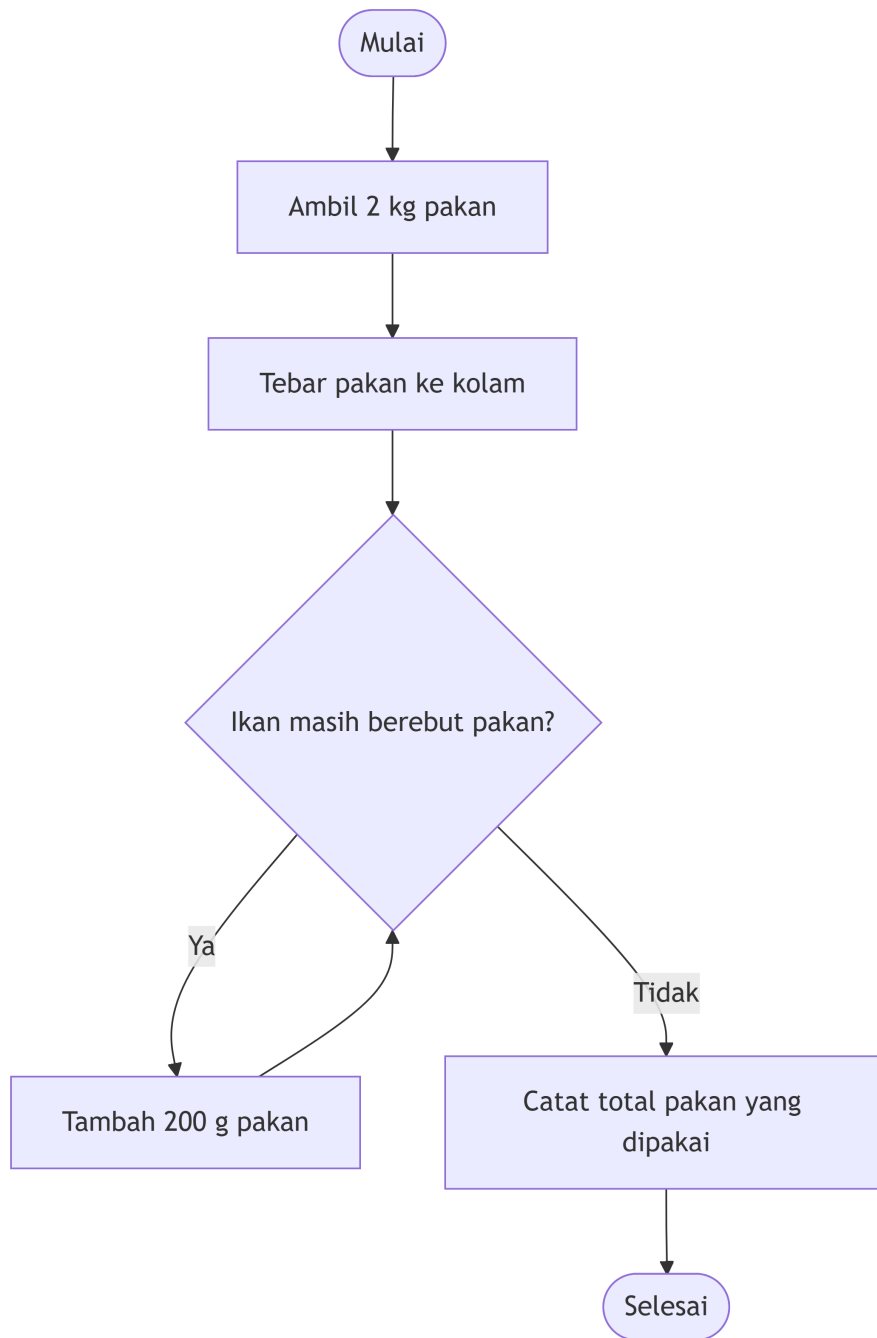
Itu saja. Seluruh sisa buku ini pada dasarnya adalah cara menuliskan ketiga hal ini dalam bahasa Python, lalu menerapkannya pada data hutan, kebun, sawah, kolam, dan kandang.

Flowchart: menggambar algoritma

Flowchart (diagram alur) adalah cara menggambar algoritma supaya mudah dilihat. Hanya ada beberapa bentuk:

Bentuk	Artinya
Oval	Mulai atau Selesai
Kotak	Satu langkah yang dikerjakan
Belah ketupat	Satu pertanyaan yang jawabannya “Ya” atau “Tidak”
Panah	Arah alur, ke langkah berikutnya

Berikut algoritma memberi pakan di kolam, memakai ketiga bahan dasar sekaligus:



Algoritma memberi pakan ikan: urutan, pilihan, dan pengulangan dalam satu gambar. Coba telusuri dengan jari: mulai dari oval **Mulai**, ikuti panah. Di belah ketupat Anda harus memilih jalan “Ya” atau “Tidak”. Jalan “Ya” **kembali** ke pertanyaan yang sama. Itulah **pengulangan**: berputar sampai jawabannya “Tidak”.

TIPS

Tidak perlu aplikasi khusus. Flowchart cukup digambar di kertas dengan pensil. Kalau Anda bisa menelusuri gambarnya tanpa bingung, berarti algoritmanya sudah cukup jelas untuk diubah menjadi kode.

Pecah masalah besar menjadi langkah kecil

Misalkan atasan meminta: “**Hitung hasil panen kebun bulan ini.**” Terdengar besar dan kabur. Pemrogram memecahnya sampai setiap potongan jelas:

1. Kumpulkan catatan panen setiap blok kebun.
2. Untuk **setiap** blok: jumlahkan berat panen semua hari.
3. Jumlahkan hasil semua blok.
4. Bandingkan dengan target bulan ini.
5. Tampilkan hasilnya dengan rapi.

Setiap nomor masih bisa dipecah lagi, sampai langkah terkecilnya begitu jelas sehingga bisa ditulis sebagai satu atau dua baris kode. Kebiasaan **memecah masalah** ini lebih berharga daripada menghafal aturan bahasa apa pun.

Mengetes algoritma di kepala

Sebelum menulis kode, uji algoritma dengan angka kecil. Contoh: pakan 2 kg, ikan berebut dua kali sebelum tenang.

Langkah	Pakan yang sudah ditebar	Ikan masih berebut?
Tebar pertama	2000 g	Ya
Tambah 200 g	2200 g	Ya
Tambah 200 g	2400 g	Tidak

Hasil: total **2400 g**. Mengetes secara manual seperti ini disebut **uji coba di atas kertas** (*dry run*). Kalau hasilnya sudah masuk akal di kertas, barulah kita yakin untuk mengetiknya.

BAGIAN TEKNIS (boleh dilewati)

Selain flowchart, pemrogram sering menulis algoritma sebagai **pseudokode**: kalimat bahasa sehari-hari yang disusun mirip kode, misalnya “**ULANGI** selama ikan berebut: tambah 200 g”. Pseudokode tidak dijalankan komputer. Ia hanya alat berpikir sebelum menulis kode sungguhan.

Latihan

1. Tulis algoritma **menyeduh teh manis** dalam enam langkah atau lebih. Pastikan tidak ada kata *secukupnya*.
2. Gambar flowchart untuk: “*Periksa pH air kolam. Jika di bawah 6, tambahkan kapur lalu periksa lagi. Jika sudah 6 atau lebih, catat hasilnya.*” Bagian mana yang berupa pengulangan?
3. **Tantangan**: pecah tugas “*Hitung jumlah pohon sehat di satu petak hutan*” menjadi minimal lima langkah, lalu tandai langkah mana yang berupa pilihan dan mana yang berupa pengulangan.

Ringkasan Bab

- **Algoritma** = urutan langkah yang jelas dan tidak ambigu untuk mencapai tujuan.
- Semua program tersusun dari tiga bahan: **urutan, pilihan, pengulangan**.
- **Flowchart** menggambar algoritma: oval (mulai/selesai), kotak (langkah), belah ketupat (pertanyaan), panah (arah).
- **Pecah** masalah besar menjadi langkah kecil, lalu **uji coba di atas kertas** sebelum mengetik.

Selanjutnya, di Bab 3: kita berkenalan dengan *terminal*, jendela hitam tempat Anda “mengobrol” dengan komputer lewat ketikan, tanpa perlu takut salah.

Bab 3 — Mengenal Terminal: Ngobrol dengan Komputer Lewat Ketikan

Seri B1 — Mulai dari Nol

Di bab ini, Anda akan:

- Tahu apa itu terminal dan kenapa pemrogram memakainya
- Membuka terminal dan membaca “prompt”-nya
- Berpindah folder, melihat isi folder, dan membuat folder baru
- Membaca pesan error terminal tanpa panik

Banyak pemula merasa terminal itu menakutkan: jendela hitam, tulisan putih, dan konon sekali salah ketik semuanya rusak. Kenyataannya jauh lebih jinak. Di akhir bab ini, Anda akan membuat folder kerja untuk belajar Python, memakai terminal, dan bisa tertawa kecil melihat “seram”-nya dulu.

Terminal itu seperti berkirim pesan

Selama ini Anda memakai komputer dengan **klik**: klik folder, klik file, klik tombol. Terminal adalah cara lain: Anda **mengetik perintah**, komputer **menjawab dengan teks**. Seperti berkirim pesan dengan pegawai gudang dari Bab 1: Anda menulis permintaan, ia membalas.

Kenapa pemrogram suka terminal?

- **Cepat**. Satu baris ketikan bisa menggantikan belasan klik.
- **Bisa diulang**. Perintah yang sama bisa dipakai lagi tanpa mengulang klik yang sama.
- **Dipakai di mana-mana**. Server, alat data, dan hampir semua perangkat pemrograman bisa dikendalikan lewat terminal.
- **Python dijalankan dari sini**. Mulai Bab 4, Anda akan menjalankan program lewat terminal.

Membuka terminal

Contoh di buku ini memakai **Windows** dan program bernama **PowerShell**.

- **Windows**: tekan tombol *Start*, ketik **Terminal** (atau **PowerShell**), lalu tekan Enter.
- **macOS**: buka aplikasi *Terminal* lewat Spotlight (Cmd + Spasi).
- **Linux**: biasanya aplikasi *Terminal* ada di menu aplikasi; caranya sedikit berbeda antar distribusi.

Perintah dasar di bab ini bekerja serupa di ketiganya, tetapi beberapa nama dan tanda garis miring berbeda. Contoh di buku mengikuti Windows PowerShell.

Membaca “prompt”

Begitu terminal terbuka, Anda melihat satu baris yang berkedip menunggu ketikan, kira-kira seperti ini (isinya di komputer Anda akan berbeda):

```
PS C:\Users\Anda>
```

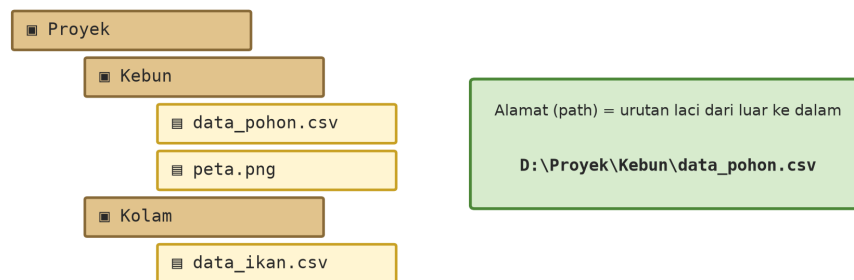
Baris ini disebut **prompt**, artinya “komputer siap menerima perintah”. Bagian penting darinya adalah **C:\Users\Anda**: itu **folder tempat Anda sedang berada**. Bayangkan Anda berdiri di depan salah satu laci; perintah yang diketik akan bekerja di laci itu.

i INGAT

Terminal selalu “berdiri” di satu folder. Kalau perintah tidak bekerja seperti harapan, pertanyaan pertama yang harus Anda ajukan: **“Saya sedang berdiri di folder mana?”**

Ingat gambar laci bersarang dari Bab 1? Setiap perintah di bawah bekerja pada salah satu laci itu:

Folder itu seperti laci bersarang



Gambar 0.1: Folder bersarang: terminal selalu “berdiri” di salah satu laci, dan perintah `cd` berpindah antar laci.

Enam perintah dasar

Ketik perintah lalu tekan **Enter**. (Bagian di dalam `< >` adalah nama yang Anda ganti sendiri; jangan ketik tanda `<` dan `>`.)

Perintah	Artinya	Analogi
<code>pwd</code>	Tampilkan folder tempat saya berdiri	“Saya di laci mana?”
<code>dir</code>	Tampilkan isi folder ini	Mengintip isi laci
<code>cd <nama-folder></code>	Masuk ke folder itu	Membuka laci yang lebih dalam
<code>cd ..</code>	Naik satu tingkat (dua titik)	Mundur ke laci di luarnya
<code>mkdir <nama-folder></code>	Buat folder baru	Menambah laci baru
<code>cls</code>	Bersihkan layar	Menghapus papan tulis (isi folder tidak terhapus)

💡 TIPS

Dua jalan pintas yang akan sering Anda syukuri. Tombol **panah atas** mengulang perintah sebelumnya. Tombol **Tab** melengkapi nama folder: ketik `cd Docu` lalu tekan Tab, nama lengkapnya muncul sendiri.

Praktik: membuat folder belajar

Ayo membuat folder kerja untuk seluruh buku ini. Ketik satu per satu (tekan Enter setelah tiap baris):

```
cd Documents
mkdir belajar-python
cd belajar-python
pwd
```

Setelah baris terakhir, terminal menampilkan alamat folder baru Anda, yang berakhir dengan `belajar-python`. Selamat: Anda baru saja membuat folder dan masuk ke dalamnya tanpa satu pun klik mouse.

Untuk memastikan, ketik `dir`. Folder ini masih kosong, jadi tidak ada yang ditampilkan. Folder inilah yang akan kita isi dengan program Python di Bab 4.

AWAS

Bila folder *Documents* Anda disinkronkan dengan **OneDrive**, folder yang tampil di File Explorer bisa berada di tempat lain dari `C:\Users\Anda\Documents`, misalnya `C:\Users\Anda\OneDrive\Documents`. Di komputer penulis, keduanya ada. Itu tidak masalah untuk belajar, **asal Anda selalu memakai folder yang sama**. Tuliskan hasil `pwd` Anda, lalu buka folder itu di File Explorer untuk memastikan Anda tahu letak aslinya.

Kalau salah ketik: pesan error

Cobalah dengan sengaja salah. Ketik nama folder yang tidak ada:

```
cd Dokumen
```

Terminal menjawab dengan pesan seperti ini (nama folder di baris Anda akan berbeda; di PowerShell 7.6.6 pesan ini awalnya diawali nama perintah):

```
Cannot find path 'C:\Users\Anda\Dokumen' because it does not exist.
```

Artinya: “Tidak menemukan alamat itu karena memang tidak ada.” Tidak ada yang rusak. Pesannya bahkan menyebutkan nama persis yang gagal ditemukan. Obatnya: ketik `dir` untuk melihat nama folder yang benar (misalnya *Documents*), lalu coba lagi.

AWAS

Terminal tidak punya tombol “Undo” untuk **perintah penghapus** (misalnya `del`, `rm`, atau `Remove-Item`). Buku ini tidak mengajarkannya sekarang. Aturan aman: **jangan menyalin perintah dari internet yang Anda tidak mengerti artinya**, dan kalau ragu, tanyakan dulu. Selama Anda memakai enam perintah di atas, tidak ada yang bisa dirusak.

TIPS

Perintah macet atau terminal tampak “beku”? Tekan **Ctrl + C** untuk membatalkan perintah yang sedang berjalan.

BAGIAN TEKNIS (boleh dilewati)

Nama “terminal” sebenarnya merujuk pada *jendela*-nya, sedangkan program yang menerjemahkan ketikan Anda disebut **shell**. PowerShell, Command Prompt (`cmd`), dan `bash` adalah contoh shell. Perintah di bab ini disengaja dipilih yang sama di banyak shell supaya mudah dibawa pindah.

Latihan

1. Buka terminal, lalu ketik `pwd`. Tuliskan alamat yang muncul, lalu cocokkan dengan nama folder yang Anda lihat di File Explorer.
2. Di dalam folder `belajar-python`, buat tiga folder: `bab04`, `bab05`, dan `proyek`. Pakai `dir` untuk membuktikannya. Lalu kembali ke folder induk dengan `cd ..`
3. **Tantangan:** buat folder `kebun`, masuk ke dalamnya, buat folder `blok-a`, masuk ke `blok-a`, lalu kembali ke `belajar-python` hanya dengan perintah `cd ..`. Berapa kali Anda mengetik `cd ..`?

Ringkasan Bab

- **Terminal** = jendela untuk mengetik perintah; komputer menjawab dengan teks.
- **Prompt** menunjukkan **folder tempat Anda berdiri**.
- Enam perintah dasar: `pwd`, `dir`, `cd <folder>`, `cd ..`, `mkdir <folder>`, `cls`.
- **Panah atas** mengulang perintah, **Tab** melengkapi nama, **Ctrl + C** membatalkan.
- Pesan error menyebutkan sendiri apa yang salah; baca pelan-pelan, bukan panik.

Selanjutnya, di Bab 4: kita berkenalan dengan Python, memasangnya, lalu menjalankan program pertama di folder `belajar-python` yang baru saja Anda buat.

Bab 4 — Mengetahui Python dan Memasanginya: Program Pertama Anda

Seri B1 — Mulai dari Nol

Di bab ini, Anda akan:

- Tahu apa itu Python dan untuk apa dipakai
- Memasang Python 3.14 dan memeriksa bahwa pemasangannya berhasil
- Mencoba Python secara langsung lewat mode interaktif
- Menulis dan menjalankan program pertama: “Halo, Lahan!”
- Membaca pesan error Python tanpa panik

Semua skrip bab ini ada di folder `kode/bab04/`. Di versi web, Anda juga bisa [mengunduh semuanya sekaligus](#) (zip).

Apa itu Python?

Python adalah bahasa pemrograman: bahasa yang dipakai manusia untuk menulis instruksi bagi komputer (ingat pegawai gudang di Bab 1). Namanya, konon, diambil dari grup komedi *Monty Python*, bukan dari ular.

Kenapa Python cocok untuk Anda?

- **Mudah dibaca.** Tulisannya mirip bahasa Inggris sederhana, sehingga pemula cepat paham.
- **Serbaguna.** Dipakai untuk mengolah data, otomatisasi pekerjaan, pemetaan (GIS), situs web, kecerdasan buatan, dan banyak lagi.
- **Gratis dan terbuka.** Siapa pun boleh memasang dan memakainya.
- **Komunitas besar.** Hampir setiap masalah pemula sudah pernah ditanyakan dan dijawab orang lain.

Memasang Python 3.14 di Windows

Buku ini memakai **Python 3.14**. Dokumentasi resmi Python menyarankan pemasangan di Windows lewat **Python install manager** (pengelola pemasangan Python); pemasang lama yang berbentuk satu berkas besar sudah dinyatakan usang sejak versi 3.14.

Langkahnya:

1. Buka python.org/downloads dan unduh **Python install manager** untuk Windows (atau pasang dari Microsoft Store).
2. Jalankan berkas yang diunduh dan ikuti petunjuk di layar.
3. Buka terminal (Bab 3), lalu ketik:

```
py install 3.14
```

Perintah ini meminta pengelola memasang Python 3.14. Bila muncul pertanyaan, baca dan ikuti petunjuknya. Menambahkan folder ke `PATH` itu **opsional** menurut dokumentasi; perintah `py` bisa dipakai tanpa itu.

⚠️ AWAS

Kalau komputer Anda pernah memasang Python dengan cara lama, bisa ada **lebih dari satu** program bernama `py.exe`: peluncur lama dan pengelola baru. Di komputer penulis ada tiga. Peluncur lama tidak mengenal perintah pengelola: `py list` dan `py install --help` dibaca sebagai *nama berkas* dan dijawab dengan pesan seperti `can't open file ... list`. Bila itu terjadi pada Anda, pakai nama yang tidak ambigu, yaitu **pymanager**: `pymanager install 3.14` dan `pymanager list`. Menurut dokumentasi resmi, **pymanager** memang disediakan untuk menghindari bentrok dengan `py.exe` lama. Daftar semua `py.exe` di komputer Anda: `where.exe py`.

i INGAT

Pemasangan di macOS dan Linux caranya berbeda. Petunjuk resminya ada di python.org/downloads. Buku ini memakai Windows sebagai contoh, tetapi hampir semua program contoh berjalan sama di sistem lain.

Memeriksa pemasangan

Pemasangan berhasil bila perintah ini menampilkan nomor versi:

```
py --version
```

Di komputer penulis, hasilnya:

```
Python 3.14.8
```

Nomor terakhir (misalnya `.8`) boleh berbeda di komputer Anda; yang penting angka depannya `3.14`.

💡 TIPS

Buku ini menguji semua contoh dengan **Python 3.14.8**. Semua skrip di buku ini juga sudah dicoba di Python 3.12.10: hasilnya sama, kecuali pesan error di Python 3.12 menampilkan jalur file lengkap (bukan hanya nama file seperti di buku ini) dan tanda penunjuk di bawah baris error kadang berbeda bentuk.

🔥 BAGIAN TEKNIS (boleh dilewati)

Anaconda adalah paket pemasangan alternatif yang membawa Python sekaligus banyak pustaka sains data. Banyak ilmuwan data memakainya. Anda **tidak perlu** Anaconda untuk membaca Seri B1 dan B2. Penjelasan dan perbandingannya ada di Bab 17.

Mencoba Python langsung: mode interaktif

Python punya “ruang obrol” untuk mencoba satu baris kode dan langsung melihat jawabannya. Namanya **mode interaktif** (disebut juga REPL). Di terminal, ketik:

```
py
```

Muncul tanda `>>>`, artinya Python siap menerima kode. Ketik `2 + 3` lalu Enter:

```
>>> 2 + 3
5
>>> print("Halo")
Halo
```

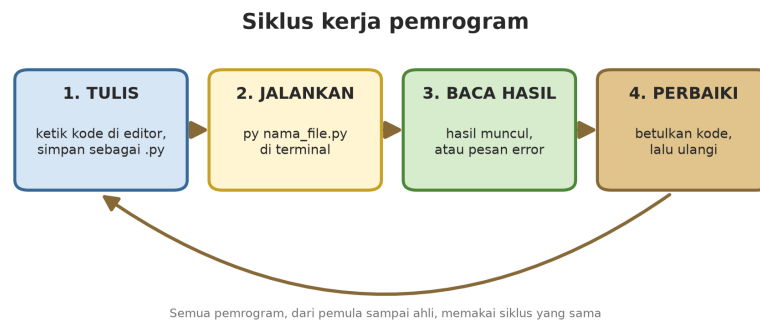
Python menghitung `2 + 3` dan langsung menjawab 5. Untuk keluar, ketik `exit()` lalu Enter.

Mode interaktif cocok untuk percobaan cepat. Tetapi apa pun yang diketik di sana **tidak tersimpan**. Untuk program yang bisa dipakai lagi, kita tulis di sebuah file.

Program pertama Anda

Langkahnya sederhana:

1. **Tulis** kode di editor teks dan simpan sebagai file `.py`.
2. **Jalankan** file itu lewat terminal.
3. **Baca** hasilnya (dan perbaiki bila ada error).



Gambar 0.1: Siklus kerja pemrogram: tulis, jalankan, baca hasil, perbaiki, ulangi.

Untuk mengetik kode, **Notepad** sudah cukup. Bila ingin lebih nyaman (kode berwarna, saran otomatis), banyak pemula memakai **Visual Studio Code**, editor gratis.

⚠ AWAS

Saat menyimpan, pastikan nama file berakhir `.py` dan bukan `.txt`. Di Notepad, ubah “Save as type” menjadi *All files*, lalu ketik nama lengkap seperti `halo.py`. Kalau ekstensi di File Explorer masih tersembunyi (Bab 1), Anda mungkin tanpa sadar membuat `halo.py.txt`.

Buat file bernama `halo.py` di folder `belajar-python` (dari Bab 3), isinya:

```
# Bab 4 - Program pertama: menyapa dunia
print("Halo, Dunia!")
```

Jalankan dari terminal (pastikan Anda berdiri di folder `belajar-python`):

```
py halo.py
```

Hasilnya:

```
Halo, Dunia!
```

Selamat, Anda baru saja menjalankan program pertama. Mari bedah:

- `print(...)` adalah **perintah** yang berarti “tampilkan ini di layar”.
- Teks yang ingin ditampilkan diapit **tanda kutip** `"..."`.
- Teks di dalam kutip tampil persis apa adanya, tanpa tanda kutipnya.

Proyek Bersambung B1: “Halo, Lahan!”

Sepanjang buku ini, kita membangun satu proyek: **Sistem Informasi Hutan, Kebun, Pertanian, Perikanan, dan Peternakan**. Langkah pertamanya sederhana: program yang menyapa dan menampilkan nama serta lokasi sebuah unit usaha.

```
# Bab 4 - Proyek bersambung B1: "Halo, Lahan!"
# Baris yang diawali tanda # adalah komentar: Python mengabaikannya.
```

```
print("Halo, Lahan!")
print("Unit usaha: Kebun Jati Sejahtera")
print("Lokasi: Kabupaten Bogor, Jawa Barat")
```

```
Halo, Lahan!
Unit usaha: Kebun Jati Sejahtera
Lokasi: Kabupaten Bogor, Jawa Barat
```

Perhatikan baris yang diawali tanda **#**. Itu **komentar**: catatan untuk manusia yang diabaikan Python. Komentar membantu Anda (dan rekan Anda) mengingat maksud kode tiga bulan lagi.

INGAT

Python membaca program **dari atas ke bawah**, satu baris demi satu baris. Urutan baris menentukan urutan hasil di layar.

Kalau error: tenang, itu wajar

Pemrogram berpengalaman pun sering melihat error setiap hari. Error bukan tanda Anda gagal; error adalah cara Python memberi tahu **apa yang perlu diperbaiki**. Dua jebakan paling umum bagi pemula:

Jebakan 1: salah ketik nama perintah

```
# Bab 4 - Salah ketik: print ditulis "prnt"
prnt("Halo, Lahan!")
```

```
Traceback (most recent call last):
  File "03_salah_ketik.py", line 2, in <module>
    prnt("Halo, Lahan!")
    ^^^
NameError: name 'prnt' is not defined. Did you mean: 'print'?
```

Baca dari **baris terakhir**: `NameError` berarti Python tidak mengenal nama `prnt`. Dan Python sendiri menebak maksud Anda: *"Did you mean: 'print'?"* (Mungkin maksud Anda `print`?). Perbaiki penulisannya, selesai.

Jebakan 2: tanda kutip penutup hilang

```
# Bab 4 - Tanda kutip penutup hilang
print("Halo, Lahan!)
```

```
File "04_kutip_hilang.py", line 2
    print("Halo, Lahan!)
    ^
SyntaxError: unterminated string literal (detected at line 2)
```

`SyntaxError` berarti **aturan penulisan** dilanggar. Kali ini Python menunjuk teks yang kutip pembukanya tidak punya pasangan. Tambahkan tanda `"` penutup, selesai.

TIPS

Baca pesan error **dari bawah ke atas**: baris terakhir menjelaskan *jenis* masalahnya, baris di atasnya menunjuk *di mana*. Tiga bulan lagi, kebiasaan ini akan menghemat jam-jam frustrasi.

⚠️ AWAS

Jangan memberi nama file Anda sama dengan perintah atau pustaka bawaan Python, misalnya `print.py` atau `random.py`. Python bisa kebingungan memilih mana yang Anda maksud. Pakai nama yang jelas seperti `halo_lahan.py`.

Latihan

1. Pasang Python 3.14, lalu jalankan `py --version`. Catat nomor versi yang muncul.
2. Tulis `halo.py` Anda sendiri, tetapi ganti isinya dengan nama Anda dan nama kota Anda. Jalankan lewat terminal.
3. Ubah `halo_lahan.py` supaya menampilkan **tiga** unit usaha berbeda (misalnya satu kebun, satu kolam, satu kandang), masing-masing dengan lokasinya.
4. **Tantangan:** sengaja hapus satu tanda kurung di salah satu `print`. Baca pesan error yang muncul dari baris terakhir. Apa yang dikatakannya?

Ringkasan Bab

- **Python** = bahasa pemrograman yang mudah dibaca dan serbaguna, gratis.
- Buku ini memakai **Python 3.14**; periksa dengan `py --version`.
- **Mode interaktif** (`py`) untuk mencoba satu baris; **file** `.py` untuk program yang tersimpan.
- Siklus kerja: **tulis** → **jalankan** → **baca hasil** → **perbaiki**.
- `print("...")` menampilkan teks; baris berawalan `#` adalah komentar.
- Python membaca program **dari atas ke bawah**.
- Error adalah petunjuk, bukan hukuman: baca **dari baris terakhir**.

Selanjutnya, di Bab 5 (Seri B2): program Anda sudah bisa dijalankan, tetapi belum bisa *mengingat* apa pun. Kita belajar variabel dan tipe data, supaya program bisa menyimpan nama pohon, bobot sapi, dan jumlah ikan.

Bab 5 — Variabel dan Tipe Data: Memberi Nama pada Segala Sesuatu

Seri B2 — Fondasi Bahasa Python

Setiap nilai di lapangan bisa diberi nama



Gambar 0.1: Gambar pembuka: setiap nilai di lapangan (pohon, ikan, sapi) diberi nama seperti label pada karung.

Di bab ini, Anda akan:

- Memahami apa itu variabel (bocoran: mirip label di karung pakan)
- Mengenal empat tipe data yang paling sering dipakai
- Mencetak kalimat rapi dengan f-string
- Membaca pesan error tanpa panik

Semua skrip bab ini ada di folder `kode/bab05/`. Di versi web, Anda juga bisa [mengunduh semuanya sekaligus](#) (zip).

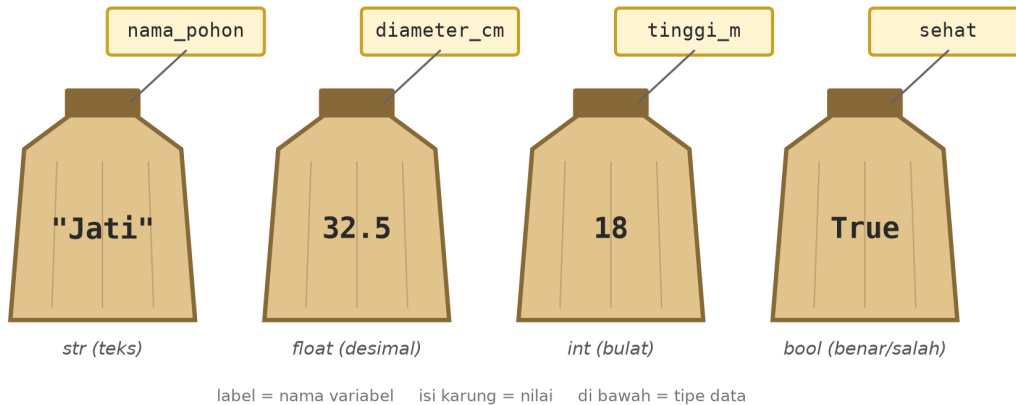
Variabel itu seperti label di karung

Bayangkan gudang pakan di peternakan. Ada puluhan karung yang bentuknya sama persis. Tanpa label, Anda harus membuka setiap karung untuk tahu isinya. Repot, kan?

Maka Anda tempelkan label: **“Konsentrat Sapi”**, **“Pelet Ikan”**, **“Dedak”**. Sekarang cukup baca labelnya.

Variabel di Python bekerja persis seperti itu. Variabel adalah *nama* yang Anda tempelkan pada sebuah *nilai*, supaya nilai itu mudah dipanggil lagi nanti.

Variabel = label pada karung



Gambar 0.2: Variabel seperti label pada karung; nama variabel ada di label, nilainya ada di dalam karung, dan setiap nilai punya tipe data.

```
# Bab 5 - Membuat variabel: memberi "label" pada nilai
nama_pohon = "Jati"
diameter_cm = 32.5
tinggi_m = 18
sehat = True
print(nama_pohon, diameter_cm, tinggi_m, sehat)
```

Hasilnya:

```
Jati 32.5 18 True
```

Tanda = di sini **bukan** "sama dengan" seperti di pelajaran matematika. Bacalah sebagai "**simpan ke dalam**": *simpan "Jati" ke dalam nama_pohon*.

i INGAT

Di Python, = berarti *menyimpan*. Untuk *membandingkan* apakah dua nilai sama, dipakai == (dua tanda sama dengan). Itu dibahas di Bab 6.

Aturan memberi nama (supaya Python tidak ngambek)

Python cukup pemilih soal nama variabel. Ini aturannya:

1. Boleh huruf, angka, dan garis bawah _.
2. **Tidak boleh** diawali angka. `1pohon` ditolak, `pohon1` boleh.
3. **Tidak boleh** ada spasi. Pakai garis bawah: `bobot_sapi`, bukan `bobot sapi`.
4. Huruf besar dan kecil dianggap berbeda. `Bobot` dan `bobot` adalah dua variabel yang berbeda.

💡 TIPS

Beri nama yang menjelaskan isinya, **lengkap dengan satuannya**: `diameter_cm`, `tinggi_m`, `berat_rata_g`. Tiga bulan lagi, Anda (atau rekan Anda) akan berterima kasih karena tidak perlu menebak apakah `d = 32.5` itu sentimeter, inci, atau umur pohon.

Empat tipe data yang akan Anda temui setiap hari

Setiap nilai di Python punya **tipe**, yaitu jenis datanya. Python bisa memberi tahu tipe itu lewat fungsi `type()`:

```
# Bab 5 - Melihat tipe data dengan type()
nama_pohon = "Jati"
diameter_cm = 32.5
tinggi_m = 18
sehat = True
print(type(nama_pohon))
print(type(diameter_cm))
print(type(tinggi_m))
print(type(sehat))
```

```
<class 'str'>
<class 'float'>
<class 'int'>
<class 'bool'>
```

Tipe	Nama lengkap	Isinya	Contoh dari lapangan
str	<i>string</i> (teks)	Huruf, kata, kalimat; selalu diapit tanda kutip	"Jati", "Kolam A", "Sapi Bali"
int	<i>integer</i> (bilangan bulat)	Angka tanpa koma	18 pohon, 1250 ekor ikan
float	bilangan desimal	Angka dengan titik desimal	32.5 cm, 85.4 gram
bool	<i>boolean</i> (benar/salah)	Hanya <code>True</code> atau <code>False</code>	<code>sehat = True</code> , <code>sudah_panen = False</code>

⚠ AWAS

Python memakai **titik** untuk desimal, bukan koma: tulis `32.5`, bukan `32,5`. Kebiasaan menulis koma (cara Indonesia) adalah salah satu jebakan paling umum bagi pemula.

Mencetak kalimat rapi dengan f-string

Menggabungkan teks dan angka paling enak memakai **f-string**: taruh huruf `f` sebelum tanda kutip, lalu masukkan variabel di dalam kurung kurawal `{ }`.

```
# Bab 5 - Mencetak kalimat rapi dengan f-string
jumlah_ikan = 1250
berat_rata_g = 85.4
print(f"Kolam A berisi {jumlah_ikan} ekor, berat rata-rata {berat_rata_g} gram.")
total_kg = jumlah_ikan * berat_rata_g / 1000
print(f"Perkiraan total panen: {total_kg:.1f} kg")
```

```
Kolam A berisi 1250 ekor, berat rata-rata 85.4 gram.
Perkiraan total panen: 106.8 kg
```

Perhatikan `{total_kg:.1f}`. Kode `:.1f` berarti “tampilkan dengan satu angka di belakang koma”. Tanpa itu, Python bisa menampilkan angka panjang yang membuat laporan terlihat berantakan.

Jebakan klasik: angka yang ternyata teks

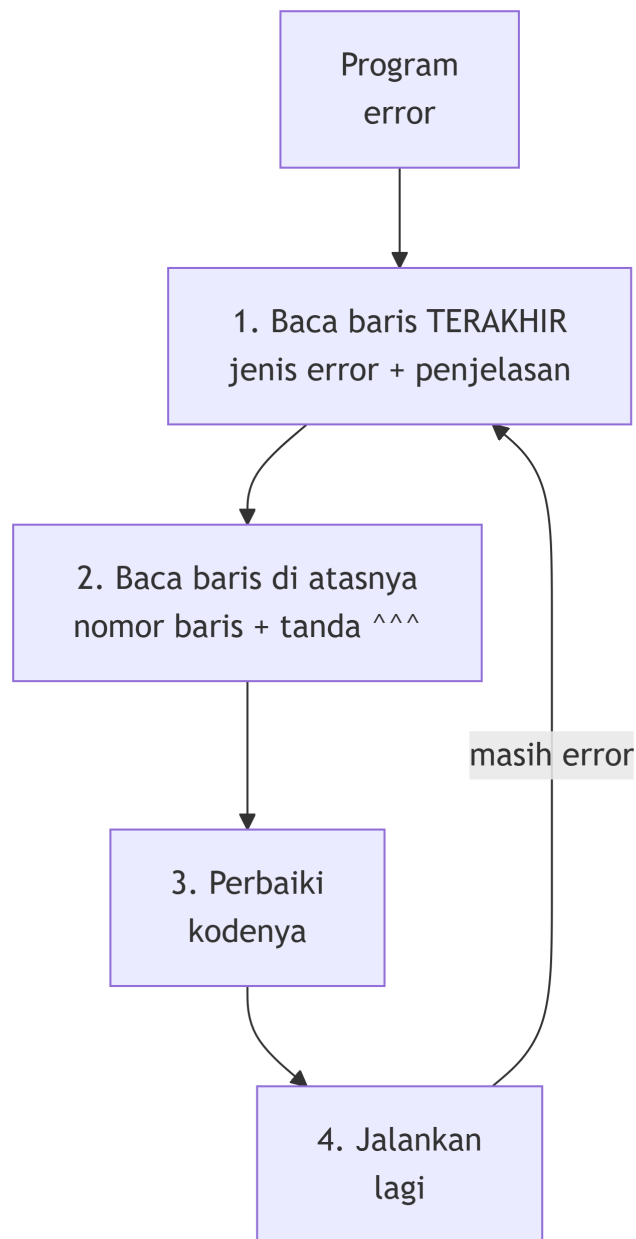
Misalkan Anda meminta pengguna mengetik bobot sapi, lalu ingin menambah 10 kg:

```
# Bab 5 - Jebakan: input() selalu memberi teks (contoh ini SENGAJA error)
bobot = input("Bobot sapi (kg): ")
print(bobot + 10)
```

Anda mengetik 350, lalu... **error**:

```
Bobot sapi (kg): 350
Traceback (most recent call last):
  File "04_input_salah.py", line 3, in <module>
    print(bobot + 10)
          ~~~~~^~~~~
TypeError: can only concatenate str (not "int") to str
```

Tenang, ini bukan komputer Anda yang rusak. Mari baca pesannya pelan-pelan, **dari bawah ke atas**:



Cara membaca pesan error: mulai dari baris terakhir.

1. **Baris terakhir** adalah inti masalahnya: `TypeError`. Artinya ada **tipe** yang tidak cocok. Python hanya bisa menyambung teks (`str`) dengan teks, bukan dengan angka (`int`).
2. **Baris di atasnya** menunjukkan di mana masalahnya: baris 3, tepat di `bobot + 10` (lihat tanda `~~~~^~~~`).

Penyebabnya: `input()` selalu memberikan teks, walaupun yang Anda ketik adalah angka. Jadi `bobot` berisi `"350"` (teks), bukan `350` (angka).

Obatnya, ubah teks itu menjadi angka dengan `float()`:

```
# Bab 5 - Obatnya: ubah teks menjadi angka dengan float()
bobot = float(input("Bobot sapi (kg): "))
print("Target bulan depan:", bobot + 10, "kg")
```

```
Bobot sapi (kg): 350
Target bulan depan: 360.0 kg
```

🔥 BAGIAN TEKNIS (boleh dilewati)

Kenapa tampilnya 360.0, bukan 360? Karena `float()` menghasilkan bilangan desimal, dan bilangan desimal selalu ditampilkan dengan titik. Bila Anda yakin bobotnya selalu bulat, pakai `int()`. Tapi hati-hati: `int("350.5")` akan error, karena teks itu bukan bilangan bulat.

Salah ketik? Python ikut membantu

Python 3.14 cukup baik hati soal salah ketik:

```
# Bab 5 - Salah ketik nama variabel (contoh ini SENGAJA error)
diameter_cm = 32.5
print(diamter_cm)
```

```
Traceback (most recent call last):
  File "06_salah_ketik.py", line 3, in <module>
    print(diamter_cm)
          ^^^^^^^^^
NameError: name 'diamter_cm' is not defined. Did you mean: 'diameter_cm'?
```

`NameError` berarti Python tidak mengenal nama itu. Kalimat terakhirnya bahkan menebak maksud Anda: “Did you mean: ‘diameter_cm’?”, yang artinya “Mungkin maksud Anda diameter_cm?”

💡 TIPS

Biasakan membaca pesan error sampai baris terakhir. Seringkali Python sudah memberi tahu jawabannya.

Proyek Bersambung B2: data lapangan pertama

Sekarang coba gabungkan semuanya. Catat satu data dari tiap sektor:

```
# Bab 5 - Proyek bersambung B2: data lapangan pertama

# Hutan
jenis_pohon = "Meranti"
diameter_pohon_cm = 45.2

# Perikanan
nama_kolam = "Kolam B"
jumlah_ikan = 800

# Peternakan
nama_ternak = "Sapi Limosin"
bobot_ternak_kg = 512.5

print(f"Pohon {jenis_pohon}: diameter {diameter_pohon_cm} cm")
print(f"{nama_kolam}: {jumlah_ikan} ekor ikan")
print(f"{nama_ternak}: {bobot_ternak_kg} kg")
```

```
Pohon Meranti: diameter 45.2 cm
Kolam B: 800 ekor ikan
Sapi Limosin: 512.5 kg
```

Di Bab 9 nanti, data yang terpisah-pisah ini akan kita kumpulkan dalam satu wadah (list dan dict), supaya bisa menampung ratusan pohon, kolam, dan ternak sekaligus.

Latihan

1. Buat variabel untuk satu petak sawah: nama varietas padi (teks), luas dalam hektare (desimal), umur tanaman dalam hari (bulat), dan apakah sudah dipupuk (benar/salah). Cetak keempatnya dengan f-string.
2. Minta pengguna mengetik jumlah telur yang dikumpulkan hari ini, lalu cetak jumlah untuk seminggu (dikali 7). Jangan lupa jebakan `input()`!
3. **Tantangan:** hitung luas lingkaran tajuk pohon berdiameter 6,4 meter (rumus: $3,14 \times r \times r$), dan tampilkan dengan dua angka di belakang koma.

Ringkasan Bab

- **Variabel** = label nama untuk sebuah nilai. = artinya *simpan ke dalam*.
- Empat tipe utama: `str` (teks), `int` (bulat), `float` (desimal), `bool` (benar/salah).
- Desimal memakai **titik**, bukan koma.
- **f-string** (`f"...{variabel}..."`) adalah cara termudah mencetak teks bercampur angka.
- `input()` selalu memberi **teks**. Ubah dulu dengan `int()` atau `float()` sebelum dihitung.
- Baca pesan error **dari baris terakhir**.

Selanjutnya, di Bab 6: menghitung dan membandingkan. Bagaimana Python tahu apakah bobot sapi sudah mencapai target panen?

Bab 6 — Operator dan Input/Output: Berhitung, Membandingkan, dan Ngobrol dengan Program

Seri B2 — Fondasi Bahasa Python

Di bab ini, Anda akan:

- Berhitung dengan Python: tambah, kurang, kali, bagi, pangkat, dan sisa bagi
- Membandingkan nilai dan mendapat jawaban Benar atau Salah
- Menggabungkan syarat dengan `and`, `or`, dan `not`
- Mengatur tampilan `print()` dan menerima ketikan lewat `input()`
- Membuat kalkulator kolam ikan pertama Anda

Semua skrip bab ini ada di folder `code/bab06/`. Di versi web, Anda juga bisa [mengunduh semuanya sekaligus \(zip\)](#).

Di Bab 5, program Anda sudah bisa *mengingat* nilai. Sekarang program belajar tiga hal baru: **menghitung**, **membandingkan**, dan **berbicara dengan pengguna**. Alat untuk dua hal pertama disebut **operator**: tanda khusus seperti `+` dan `>=` yang memerintahkan Python melakukan sesuatu pada nilai.

Berhitung seperti kalkulator

Python bisa semua hitungan dasar, dan lambangnya hampir sama dengan di kalkulator:

Operator	Artinya	Contoh	Hasil
<code>+</code>	tambah	<code>5 + 3</code>	8
<code>-</code>	kurang	<code>5 - 3</code>	2
<code>*</code>	kali (bintang, bukan x)	<code>5 * 3</code>	15
<code>/</code>	bagi	<code>5 / 2</code>	2.5
<code>//</code>	bagi bulat	<code>5 // 2</code>	2
<code>%</code>	sisa bagi	<code>5 % 2</code>	1
<code>**</code>	pangkat	<code>5 ** 2</code>	25

Mari terapkan pada hasil panen ikan:

```
# Bab 6 - Aritmetika: menghitung hasil panen
panen_pagi_kg = 120.5
panen_sore_kg = 98.25

total_kg = panen_pagi_kg + panen_sore_kg
selisih_kg = panen_pagi_kg - panen_sore_kg
harga_per_kg = 15000
pendapatan = total_kg * harga_per_kg

print(f"Total panen: {total_kg} kg")
print(f"Selisih pagi dan sore: {selisih_kg} kg")
print(f"Pendapatan: Rp{pendapatan:.0f}")
```

```
sisi_petak_m = 12
luas_petak_m2 = sisi_petak_m ** 2 # pangkat dua
print(f"Luas petak: {luas_petak_m2} m2")
```

```
Total panen: 218.75 kg
Selisih pagi dan sore: 22.25 kg
Pendapatan: Rp3281250
Luas petak: 144 m2
```

Perhatikan {pendapatan:.0f}: kode :.0f berarti “tampilkan **tanpa** angka di belakang titik”. Tanpa itu, hasilnya tampil sebagai 3281250.0.

TIPS

Gunakan kata `luas_petak_m2`, bukan `luas`. Satuan di dalam nama variabel membuat hitungan Anda mudah diperiksa: kalau Anda tidak sengaja mengalikan meter dengan sentimeter, kesalahannya langsung terlihat.

Tiga cara membagi

Pembagian punya tiga bentuk, dan ketiganya berguna di lapangan. Misalkan 1.250 ekor ikan dibagi rata ke 8 kolam:

```
# Bab 6 - Tiga jenis pembagian: / // %
jumlah_ikan = 1250
jumlah_kolam = 8

print(f"Pembagian biasa (/): {jumlah_ikan / jumlah_kolam}")
print(f"Pembagian bulat (//): {jumlah_ikan // jumlah_kolam}")
print(f"Sisa bagi (%): {jumlah_ikan % jumlah_kolam}")
```

```
Pembagian biasa (/): 156.25
Pembagian bulat (//): 156
Sisa bagi (%): 2
```

- / memberi hasil **desimal** (156,25 ekor per kolam). Cocok untuk rata-rata.
- // memberi hasil **bulat** (156 ekor per kolam). Cocok untuk membagi benda yang tidak bisa dipotong, seperti ikan.
- % memberi **sisa** (2 ekor yang belum kebagian). Ini yang sering dilupakan pemula, padahal sangat berguna.

INGAT

Di Python, / **selalu** menghasilkan bilangan desimal (`float`), bahkan bila hasilnya bulat. `10 / 2` menghasilkan `5.0`, bukan `5`. Untuk hasil bulat, pakai `//`.

Satu hal yang tidak bisa dilakukan: membagi dengan nol. Lihat apa yang terjadi bila jumlah kolam kebetulan 0:

```
# Bab 6 - Salah: membagi dengan nol
jumlah_ikan = 1250
jumlah_kolam = 0
print(jumlah_ikan / jumlah_kolam)
```

```
Traceback (most recent call last):
  File "10_bagi_nol.py", line 4, in <module>
    print(jumlah_ikan / jumlah_kolam)
```

```
~~~~~^~~~~~  
ZeroDivisionError: division by zero
```

`ZeroDivisionError` artinya persis seperti namanya: *pembagian dengan nol*. Dalam program sungguhan, Bab 7 akan mengajarkan cara mengecek `jumlah_kolam` sebelum membagi.

Membandingkan nilai

Operator perbandingan membandingkan dua nilai dan menghasilkan **True** (benar) atau **False** (salah). Ingat tipe `bool` dari Bab 5? Di sinilah ia dipakai.

Operator	Artinya
<code>==</code>	sama dengan
<code>!=</code>	tidak sama dengan
<code>></code>	lebih besar dari
<code>>=</code>	lebih besar atau sama dengan
<code><</code>	lebih kecil dari
<code><=</code>	lebih kecil atau sama dengan

```
# Bab 6 - Operator perbandingan: hasilnya selalu True atau False  
bobot_g = 520  
target_g = 500  
  
print("bobot == target :", bobot_g == target_g)  
print("bobot != target :", bobot_g != target_g)  
print("bobot > target :", bobot_g > target_g)  
print("bobot >= target :", bobot_g >= target_g)  
print("bobot < target :", bobot_g < target_g)  
print("bobot <= target :", bobot_g <= target_g)  
  
# Teks juga bisa dibandingkan, dan huruf besar/kecil dianggap beda  
jenis = "Lele"  
print('jenis == "Lele" :', jenis == "Lele")  
print('jenis == "lele" :', jenis == "lele")
```

```
bobot == target : False  
bobot != target : True  
bobot > target : True  
bobot >= target : True  
bobot < target : False  
bobot <= target : False  
jenis == "Lele" : True  
jenis == "lele" : False
```

Dua catatan penting dari hasil di atas:

1. Teks juga bisa dibandingkan, dan **huruf besar/kecil dianggap berbeda**: `"Lele"` tidak sama dengan `"lele"`.
2. Hasil perbandingan bisa disimpan di variabel, seperti `siap_panen = bobot_g >= 500`.

⚠ AWAS

Ini jebakan paling terkenal: **= dan == berbeda jauh**. Satu tanda sama dengan berarti *simpan*, dua tanda sama dengan berarti *bandingkan*. Python membantu bila Anda tertukar:

```
# Bab 6 - Salah: memakai = (simpan) padahal maksudnya == (bandingkan)
print(500 = 500)
```

```
File "09_salah_sama_dengan.py", line 2
    print(500 = 500)
          ^^^^^
```

```
SyntaxError: expression cannot contain assignment, perhaps you meant "=="?
```

Pesannya jelas: *"ekspresi tidak boleh berisi penugasan, mungkin maksud Anda ==?"* Angka 500 bukan variabel, jadi tidak bisa diisi nilai.

Menggabungkan syarat: and, or, not

Keputusan di lapangan jarang bergantung pada satu syarat. Ikan siap panen bila bobotnya cukup **dan** sehat. Kolam perlu diperiksa bila ikannya masih muda **atau** tidak sehat. Python punya tiga operator logika untuk ini:

Operator	Artinya	Hasil True bila...
and	dan	kedua sisi True
or	atau	salah satu sisi True
not	bukan	sisinya False (dibalik)

```
# Bab 6 - Operator logika: and, or, not
bobot_g = 520
sehat = True
umur_hari = 90

siap_panen = bobot_g >= 500 and sehat
perlu_diperiksa = umur_hari < 30 or not sehat

print("Siap panen      :", siap_panen)
print("Perlu diperiksa :", perlu_diperiksa)
print("Tidak sehat     :", not sehat)
```

```
Siap panen      : True
Perlu diperiksa : False
Tidak sehat     : False
```

Baca `siap_panen` seperti kalimat biasa: *"bobot minimal 500 gram dan sehat"*. Kedua syarat terpenuhi ($520 \geq 500$, dan `sehat` benar), jadi hasilnya `True`. Itulah alasan Python sengaja dibuat mirip bahasa manusia.

Urutan pengerjaan operator

Seperti di pelajaran matematika, operator punya urutan: perkalian dikerjakan sebelum penjumlahan.

```
# Bab 6 - Urutan pengerjaan operator (seperti pelajaran matematika)
print(2 + 3 * 4)      # perkalian dikerjakan lebih dulu
print((2 + 3) * 4)    # tanda kurung mengubah urutan
print(10 - 2 ** 3)    # pangkat lebih dulu daripada kurang

# Contoh lapangan: biaya pakan 2 kolam, ditambah biaya listrik
```

```
biaya = 50000 + 2 * 30000
print(f"Biaya total: Rp{biaya}")
```

```
14
20
2
Biaya total: Rp110000
```

2 + 3 * 4 menghasilkan 14 (bukan 20) karena 3 * 4 dikerjakan lebih dulu. Dan 10 - 2 ** 3 menghasilkan 2, karena pangkat 2 ** 3 = 8 dikerjakan sebelum pengurangan.

Tangga urutan pengerjaan operator

DIKERJAKAN PALING DULU

1	**	pangkat	2 ** 3
2	*, /, //, %	kali, bagi, bagi bulat, sisa bagi	10 * 2
3	+, -	tambah, kurang	5 + 3
4	==, !=, <, >, <=, >=	perbandingan	bobot >= 500
5	not	kebalikan	not sehat
6	and, lalu, or	logika (and lebih dulu daripada or)	a and b or c

DIKERJAKAN PALING AKHIR

Ragu? Pakai tanda kurung (). Kurung selalu dikerjakan lebih dulu.

Gambar 0.1: Tangga urutan pengerjaan operator (disederhanakan; operator yang belum dibahas tidak dicantumkan).

TIPS

Tidak perlu menghafal tangga itu. **Kalau ragu, pakai tanda kurung.** (2 + 3) * 4 jelas mau mengerjakan penjumlahan dulu, dan orang yang membaca kode Anda tiga bulan lagi tidak perlu menebak.

Kejutan angka desimal

Ini bukan salah Anda, dan bukan salah Python:

```
# Bab 6 - Kejutan angka desimal
print(0.1 + 0.2)
print(0.1 + 0.2 == 0.3)

hasil = round(0.1 + 0.2, 2) # bulatkan ke 2 angka di belakang titik
print(hasil)
print(hasil == 0.3)
```

```
0.30000000000000004
False
0.3
True
```

Ya, $0.1 + 0.2$ **tidak persis** 0.3 di komputer. Obatnya sederhana: bulatkan dulu dengan `round(angka, jumlah_desimal)` sebelum membandingkan.

BAGIAN TEKNIS (boleh dilewati)

Komputer menyimpan angka dalam sistem biner (1 dan 0). Sebagian pecahan desimal yang tampak sederhana bagi kita, seperti 0.1 , tidak bisa ditulis tepat dalam biner, sama seperti $1/3$ tidak bisa ditulis tepat dalam desimal ($0.3333\dots$). Hasilnya selisih yang amat kecil. Ini terjadi di hampir semua bahasa pemrograman, bukan hanya Python.

Berbicara dengan program: `print()` dan `input()`

Anda sudah memakai `print()` sejak Bab 4 dan `input()` di Bab 5. Sekarang kita lihat dua pengaturan `print()` yang berguna, dan `input()` dengan lebih saksama.

```
# Bab 6 - print() dan input() lebih dalam
nama = input("Nama unit usaha: ")
lokasi = input("Lokasi: ")
print("Unit usaha:", nama, "di", lokasi)

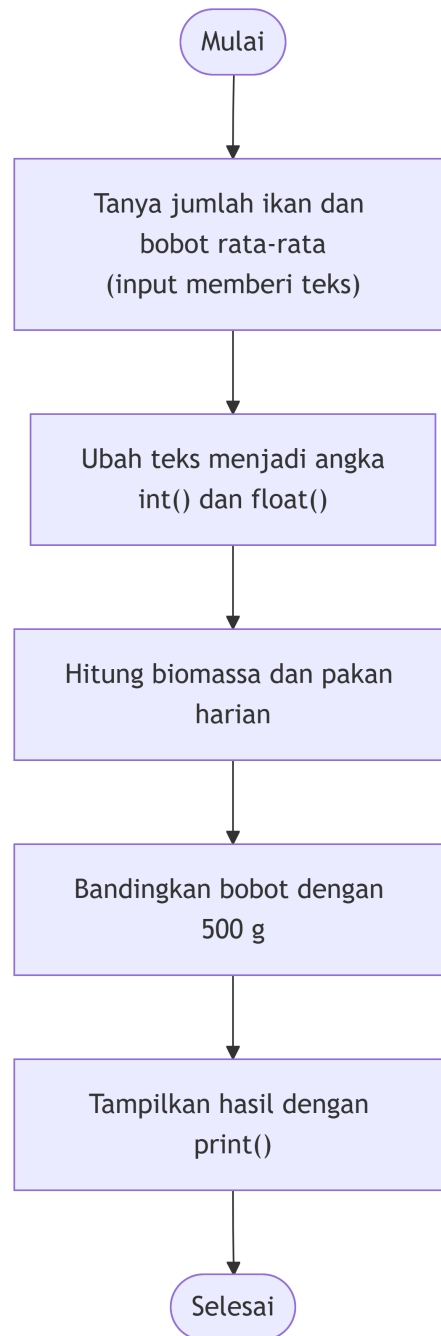
print("Kolam", "A", "aktif", sep="-")      # sep = pemisah antar nilai
print("Memeriksa...", end=" ")            # end = penutup baris
print("selesai")
```

```
Nama unit usaha: Kolam Mina Tani
Lokasi: Subang
Unit usaha: Kolam Mina Tani di Subang
Kolam-A-aktif
Memeriksa... selesai
```

- `input("...")` menampilkan pertanyaan, menunggu Anda mengetik, lalu **memberikan teks** yang diketik.
- Di `print()`, **sep** mengatur pemisah antar nilai (di sini tanda `-`), dan **end** mengatur apa yang menutup baris. Biasanya `end` adalah pindah baris, tetapi di sini kita ganti dengan satu spasi supaya "Memeriksa..." dan "selesai" tampil sebaris.

Proyek Bersambung B2: kalkulator kolam

Sekarang kita gabungkan semua: program yang menanyakan jumlah dan bobot ikan, menghitung biomassa dan kebutuhan pakan harian, lalu menentukan apakah ikan siap panen.



Alur kalkulator kolam: tanya, ubah jadi angka, hitung, tampilkan.

```
# Bab 6 - Proyek bersambung B2: kalkulator kolam
# (angka 3% dan 500 g adalah angka contoh untuk latihan, bukan anjuran budidaya)
jumlah_ikan = int(input("Jumlah ikan (ekor): "))
bobot_rata_g = float(input("Bobot rata-rata (gram): "))

biomassa_kg = jumlah_ikan * bobot_rata_g / 1000
pakan_harian_kg = biomassa_kg * 0.03
siap_panen = bobot_rata_g >= 500
```

```
print(f"Biomassa kolam : {biomassa_kg:.1f} kg")
print(f"Pakan per hari : {pakan_harian_kg:.2f} kg")
print(f"Siap panen      : {siap_panen}")
```

```
Jumlah ikan (ekor): 800
Bobot rata-rata (gram): 540.5
Biomassa kolam : 432.4 kg
Pakan per hari : 12.97 kg
Siap panen      : True
```

Telusuri satu per satu: $800 \text{ ekor} \times 540,5 \text{ gram} = 432.400 \text{ gram} = \mathbf{432,4 \text{ kg}}$ biomassa. Pakan harian 3% dari itu adalah **12,97 kg**. Dan karena $540,5 \geq 500$, kolam **siap panen**.

AWAS

Angka 3% pakan dan 500 gram panen hanyalah **angka contoh untuk latihan**, bukan anjuran budidaya. Kebutuhan sebenarnya bergantung pada jenis ikan, umur, suhu air, dan faktor lain. Tanyakan kepada penyuluh perikanan atau pembudidaya berpengalaman.

Latihan

1. Seorang petani memanen 3 blok kebun: 245,5 kg, 310 kg, dan 198,25 kg. Hitung total panen dan rata-rata per blok, lalu cetak dengan f-string.
2. Sebuah kandang berisi 1.000 butir telur akan dikemas dalam rak berisi 30 butir. Dengan // dan %, hitung berapa rak penuh dan berapa butir telur sisa.
3. Ubah `08_kalkulator_kolam.py` supaya persentase pakan (3%) juga ditanyakan lewat `input()`. Jangan lupa mengubah teks menjadi angka dan membagi 100.
4. **Tantangan:** tulis satu ekspresi yang bernilai `True` bila sebuah pohon **layak ditebang**: diameter minimal 40 cm **dan** sehat, **atau** tinggi lebih dari 25 m. Uji dengan beberapa angka. Hati-hati dengan urutan `and` dan `or`; pakai tanda kurung.

Ringkasan Bab

- **Aritmetika:** `+` `-` `*` `/` `//` `%` `**`. `/` selalu menghasilkan desimal; `//` membagi bulat; `%` memberi sisa.
- **Perbandingan:** `==` `!=` `>` `>=` `<` `<=` menghasilkan `True` atau `False`. `=` **menyimpan**, `==` **membandingkan**.
- **Logika:** `and`, `or`, `not` menggabungkan syarat.
- Operator punya **urutan**; kalau ragu, pakai **tanda kurung**.
- Desimal di komputer bisa sedikit meleset; gunakan `round()` sebelum membandingkan.
- `print()` punya `sep` dan `end`; `input()` selalu memberi **teks**, jadi ubah dengan `int()` atau `float()`.

Selanjutnya, di Bab 7: program Anda sudah bisa menjawab Benar atau Salah. Sekarang kita pakai jawaban itu untuk **mengambil keputusan**: *jika* bobot sudah cukup, *maka* panen; *kalau tidak*, tunggu.

Bab 7 — Kondisional: Mengajari Program Mengambil Keputusan

Seri B2 — Fondasi Bahasa Python

Di bab ini, Anda akan:

- Memakai `if`, `elif`, dan `else` supaya program memilih jalan sendiri
- Memahami **indentasi** (menjorok), aturan penting Python
- Memilih satu dari banyak kemungkinan dengan `match` dan `case`
- Menghindari error pembagian nol yang Anda temui di Bab 6
- Membuat penentu status kolam ikan

Semua skrip bab ini ada di folder `kode/bab07/`. Di versi web, Anda juga bisa [mengunduh semuanya sekaligus](#) (zip).

Di Bab 6, program Anda bisa menjawab *Benar* atau *Salah*. Tetapi jawaban itu belum dipakai untuk apa-apa. Di bab ini, jawaban itu menjadi **keputusan**: *jika* ikan sudah cukup berat, *maka* panen; *kalaupun tidak*, tunggu. Ini persis bahan **pilihan** dari Bab 2.

if: kerjakan hanya bila syaratnya benar

```
# Bab 7 - if: kerjakan hanya bila syaratnya benar
bobot_g = 520

if bobot_g >= 500:
    print("Ikan siap panen")
    print("Catat di buku panen")

print("Pemeriksaan selesai") # tanpa menjorok: selalu dikerjakan
```

```
Ikan siap panen
Catat di buku panen
Pemeriksaan selesai
```

Perhatikan tiga hal:

1. Setelah syarat ada tanda **titik dua** `:`.
2. Baris yang menjadi isi **if** **menjorok** ke dalam (di buku ini 4 spasi). Dua baris `print` yang menjorok hanya dikerjakan bila syaratnya benar.
3. Baris terakhir **tidak menjorok**, sehingga dikerjakan **selalu**, entah syaratnya benar atau salah.

i INGAT

Di Python, **indentasi bukan hiasan**: ia menunjukkan baris mana yang menjadi bagian dari `if`. Bahasa lain memakai kurung kurawal; Python memakai jorokan. Pakai **4 spasi** secara konsisten.

else: jalan kedua

Bagaimana kalau syaratnya salah? Tambahkan `else` (“kalau tidak”):

```
# Bab 7 - if ... else: dua jalan
ph_air = 5.8

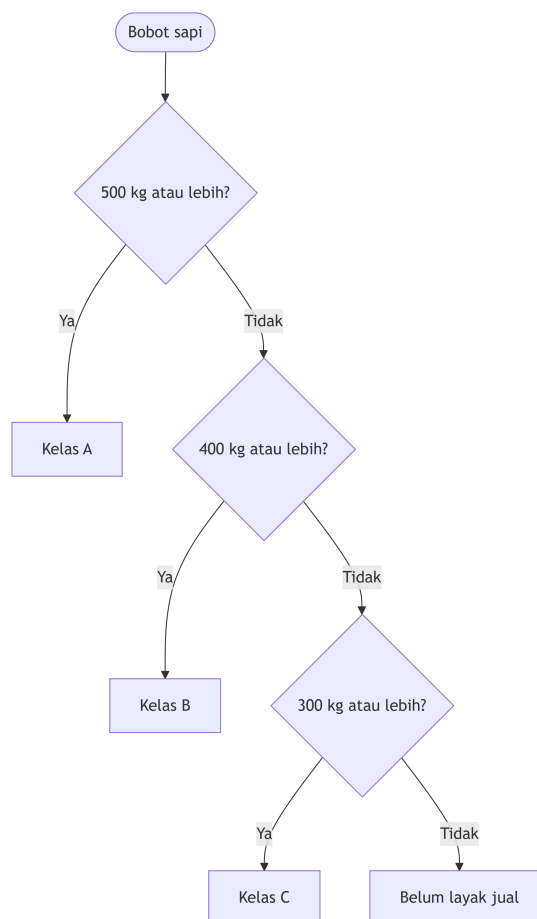
if ph_air < 6:
    print("pH terlalu rendah: tambahkan kapur")
else:
    print("pH aman")
```

pH terlalu rendah: tambahkan kapur

Nilai 5.8 kurang dari 6, jadi jalan `if` dipilih. Hanya **satu** dari dua jalan yang dikerjakan, tidak pernah keduanya.

elif: banyak jalan

Kadang pilihannya lebih dari dua. `elif` (kependekan dari *else if*, “kalau tidak, jika..”) menyambung syarat berikutnya:



Alur memilih kelas sapi: syarat diperiksa dari atas, berhenti di syarat pertama yang benar.

```
# Bab 7 - if ... elif ... else: banyak jalan (angka contoh untuk latihan)
bobot_kg = 450

if bobot_kg >= 500:
    kelas = "Kelas A"
elif bobot_kg >= 400:
    kelas = "Kelas B"
```

```
elif bobot_kg >= 300:
    kelas = "Kelas C"
else:
    kelas = "Belum layak jual"

print(f"Sapi {bobot_kg} kg masuk: {kelas}")
```

Sapi 450 kg masuk: Kelas B

Python memeriksa syarat **dari atas ke bawah**, dan berhenti di syarat pertama yang benar. Karena $450 \geq 400$ tetapi bukan ≥ 500 , hasilnya Kelas B.

⚠️ AWAS

Urutan syarat penting. Kalau Anda menulis `bobot_kg >= 300` lebih dulu, sapi 450 kg akan langsung masuk Kelas C, karena $450 \geq 300$ juga benar. Susun dari syarat yang paling ketat.

💡 TIPS

Angka batas kelas di contoh ini hanya **angka latihan**. Dalam proyek nyata, simpan batas semacam itu dalam variabel bernama jelas (misalnya `batas_kelas_a_kg`) supaya mudah diubah di satu tempat.

Menggabungkan syarat

Operator `and`, `or`, dan `not` dari Bab 6 boleh dipakai langsung di dalam `if`:

```
# Bab 7 - Syarat gabungan dengan and dan or
diameter_cm = 45
tinggi_m = 20
sehat = True

if diameter_cm >= 40 and sehat:
    print("Pohon layak ditebang")
elif tinggi_m > 25 or not sehat:
    print("Pohon perlu diperiksa dulu")
else:
    print("Pohon dibiarkan tumbuh")
```

Pohon layak ditebang

Telusuri: diameter $45 \geq 40$ **dan** pohon sehat, jadi syarat pertama benar dan dua syarat lainnya tidak diperiksa.

Menjaga program dari error

Di Bab 6 Anda melihat `ZeroDivisionError`. Kini kita bisa mencegahnya: periksa dulu sebelum membagi.

```
# Bab 7 - Menghindari ZeroDivisionError dari Bab 6
jumlah_ikan = 1250
jumlah_kolam = 0

if jumlah_kolam == 0:
    print("Jumlah kolam 0: pembagian dibatalkan")
else:
    print(f"Ikan per kolam: {jumlah_ikan // jumlah_kolam}")
```

Jumlah kolam 0: pembagian dibatalkan

Pola “periksa dulu, baru kerjakan” ini akan sering Anda pakai. Bab 12 mengajarkan cara lain yang lebih umum untuk menangani error.

match dan case: memilih satu dari banyak nilai

Bila Anda membandingkan **satu variabel** dengan banyak kemungkinan nilai, rangkaian `elif` menjadi panjang. Python menyediakan `match` dan `case`:

```
# Bab 7 - match / case: memilih satu dari banyak kemungkinan nilai
# (kebutuhan pakan adalah angka contoh untuk latihan, bukan anjuran)
jenis_ternak = "kambing"

match jenis_ternak:
    case "sapi":
        pakan_kg = 25
    case "kambing" | "domba":
        pakan_kg = 4
    case "ayam":
        pakan_kg = 0.12
    case _:
        pakan_kg = None

if pakan_kg is None:
    print("Jenis ternak belum terdaftar")
else:
    print(f"Pakan {jenis_ternak}: {pakan_kg} kg per ekor per hari")
```

```
Pakan kambing: 4 kg per ekor per hari
```

- `match jenis_ternak`: berarti “periksa isi variabel ini”.
- Setiap `case` adalah satu kemungkinan. Tanda `|` berarti **atau** (kambing *atau* domba memakai jalan yang sama).
- `case _`: menangkap **semua sisanya**, seperti `else`.
- `is None` adalah cara membandingkan dengan `None` (nilai “kosong” dari Bab 5).

BAGIAN TEKNIS (boleh dilewati)

`match/case` tersedia mulai Python 3.10. Ia jauh lebih kuat daripada yang ditunjukkan di sini: bisa mencocokkan *bentuk* data (misalnya daftar berisi dua angka). Untuk Seri B2, cukup pakai untuk memilih berdasarkan satu nilai.

Jebakan: lupa menjorok

```
# Bab 7 - Salah: lupa menjorok setelah if
bobot_g = 520
if bobot_g >= 500:
    print("Ikan siap panen")
```

```
File "07_salah_indentasi.py", line 4
    print("Ikan siap panen")
    ^^^^^
```

```
IndentationError: expected an indented block after 'if' statement on line 3
```

`IndentationError` berarti “saya mengharapkan baris yang menjorok setelah `if`”. Pesannya bahkan menyebut `if` di baris berapa yang menuntut jorokan itu. Obatnya: geser baris `print` ke dalam, dengan 4 spasi.

Proyek Bersambung B2: status kolam

Sekarang kalkulator kolam dari Bab 6 kita beri kemampuan memutuskan:

```
# Bab 7 - Proyek bersambung B2: status kolam dari bobot rata-rata
# (batas 500 g dan 400 g adalah angka contoh untuk latihan)
bobot_rata_g = float(input("Bobot rata-rata ikan (gram): "))

if bobot_rata_g >= 500:
    status = "SIAP PANEN"
elif bobot_rata_g >= 400:
    status = "HAMPIR SIAP: periksa lagi minggu depan"
else:
    status = "MASIH TUMBUH"

print(f"Status kolam: {status}")
```

```
Bobot rata-rata ikan (gram): 460
Status kolam: HAMPIR SIAP: periksa lagi minggu depan
```

Dengan bobot rata-rata 460 gram, program menjawab **HAMPIR SIAP**. Coba jalankan dengan angka lain (misalnya 520 dan 300) untuk melihat tiga jalan yang berbeda.

Latihan

1. Tulis program yang menanyakan suhu air kolam (angka desimal) dan mencetak “Terlalu dingin” bila di bawah 25, “Terlalu panas” bila di atas 32, dan “Suhu baik” selainnya. (Batas 25 dan 32 adalah angka latihan.)
2. Ubah `03_elif.py` supaya bobot sapi ditanyakan lewat `input()`. Uji dengan 520, 350, dan 200.
3. Buat program yang menanyakan **umur tanaman dalam hari** dan mencetak fase: “Semai” (0–14), “Vegetatif” (15–45), “Berbunga” (46–70), atau “Panen” (di atas 70). Ini angka rekaan untuk latihan.
4. **Tantangan:** ubah `06_match_case.py` supaya `jenis_ternak` ditanyakan lewat `input()`, dan tambahkan satu jenis ternak baru. Apa yang terjadi bila pengguna mengetik `Sapi` dengan huruf S besar?

Ringkasan Bab

- **if / elif / else** membuat program memilih jalan. Hanya **satu** jalan yang dikerjakan.
- Setelah syarat ada **titik dua**, dan isinya **menjorok** (4 spasi). Indentasi adalah bagian dari tata bahasa Python.
- Python memeriksa syarat **dari atas ke bawah**; susun dari yang paling ketat.
- **and, or, not** menggabungkan syarat.
- **match / case** memilih berdasarkan satu nilai; `case _` menangkap sisanya.
- Periksa dulu sebelum membagi, supaya tidak kena `ZeroDivisionError`.

Selanjutnya, di Bab 8: keputusan sudah bisa diambil. Sekarang kita ajari program **mengulang pekerjaan**: memeriksa semua kolam, menjumlahkan seluruh panen, tanpa menulis ulang kode puluhan kali.

Bab 8 — Perulangan: Menyuruh Program Mengerjakan Hal yang Sama Berkali-kali

Seri B2 — Fondasi Bahasa Python

Di bab ini, Anda akan:

- Memakai `for` untuk mengulang pekerjaan pada setiap isi daftar
- Membuat deretan angka dengan `range()`
- Memakai `while` untuk mengulang selama syarat masih benar
- Menghentikan atau melewati satu putaran dengan `break` dan `continue`
- Membuat rekap panen kebun per blok

Semua skrip bab ini ada di folder `code/bab08/`. Di versi web, Anda juga bisa [mengunduh semuanya sekaligus \(zip\)](#).

Bayangkan memeriksa 40 kolam ikan. Menulis `print` 40 kali jelas melelahkan, dan kalau jumlah kolamnya bertambah menjadi 41, Anda harus mengubah kode lagi. Di sinilah **perulangan** (*loop*) menyelamatkan Anda. Ini bahan ketiga dari Bab 2.

for: untuk setiap isi daftar

```
# Bab 8 - for: mengulang untuk setiap isi daftar
daftar_kolam = ["A", "B", "C"]

for kolam in daftar_kolam:
    print(f"Memeriksa Kolam {kolam}")

print("Semua kolam sudah diperiksa")
```

```
Memeriksa Kolam A
Memeriksa Kolam B
Memeriksa Kolam C
Semua kolam sudah diperiksa
```

Baca seperti kalimat: “*untuk setiap kolam di dalam daftar_kolam, kerjakan yang menjorok.*” Pada putaran pertama kolam berisi “A”, putaran kedua “B”, dan seterusnya. Baris terakhir tidak menjorok, jadi hanya dikerjakan **sekali**, setelah semua putaran selesai.

i INGAT

Aturan indentasi dari Bab 7 berlaku juga di sini: baris yang **menjorok** adalah isi perulangan yang diulang; baris yang tidak menjorok dikerjakan sesudah perulangan selesai.

range: deretan angka

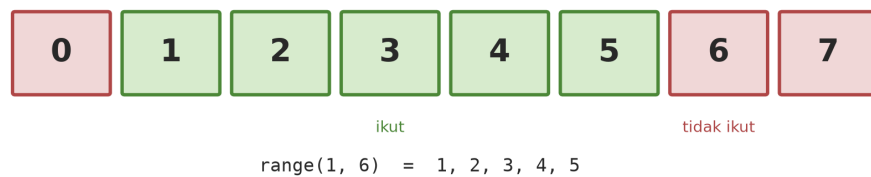
Kalau Anda tidak punya daftar tetapi ingin mengulang sekian kali, pakai `range()`:

```
# Bab 8 - range(): membuat deretan angka
for hari in range(1, 6):          # 1, 2, 3, 4, 5 (angka 6 tidak ikut)
    print(f"Hari ke-{hari}")
```

```
print(list(range(5)))      # mulai dari 0
print(list(range(0, 10, 3))) # loncat 3 angka
```

```
Hari ke-1
Hari ke-2
Hari ke-3
Hari ke-4
Hari ke-5
[0, 1, 2, 3, 4]
[0, 3, 6, 9]
```

range(1, 6): mulai di 1, berhenti SEBELUM 6



Gambar 0.1: `range(1, 6)` berhenti **sebelum** 6, jadi menghasilkan 1, 2, 3, 4, 5.

- `range(5)` menghasilkan 0, 1, 2, 3, 4: **mulai dari 0** dan berhenti sebelum 5.
- `range(1, 6)` menghasilkan 1 sampai 5: angka akhir **tidak ikut**.
- `range(0, 10, 3)` meloncat 3 angka sekali.
- `list(...)` mengubah deretan itu menjadi daftar supaya bisa dicetak.

⚠️ AWAS

Angka akhir di `range()` **tidak ikut**. Ini jebakan klasik: `range(1, 6)` hanya sampai 5, bukan 6. Kalau butuh sampai 6, tulis `range(1, 7)`.

Pola penampung: menjumlahkan lewat perulangan

Salah satu pola paling berguna: siapkan variabel awal bernilai 0, lalu tambahkan isi daftar satu per satu.

```
# Bab 8 - Menjumlahkan lewat perulangan (pola "penampung")
panen_harian_kg = [120.5, 98.25, 143.0]

total_kg = 0
for panen in panen_harian_kg:
    total_kg = total_kg + panen

jumlah_hari = len(panen_harian_kg) # len() = banyaknya isi daftar
print(f"Total panen: {total_kg} kg dalam {jumlah_hari} hari")
print(f"Rata-rata: {total_kg / jumlah_hari:.2f} kg per hari")
```

```
Total panen: 361.75 kg dalam 3 hari
Rata-rata: 120.58 kg per hari
```

Telusuri `total_kg`: mulai 0, jadi 120.5, lalu 218.75, lalu 361.75. `len(daftar)` memberi **banyaknya isi** daftar, di sini 3.

💡 TIPS

Python sebenarnya punya `sum(panen_harian_kg)` yang menjumlahkan dalam satu langkah (Bab 9). Tetapi pola penampung dengan perulangan penting dipahami, karena kerangka yang sama dipakai untuk menghitung, mencari yang terbesar, dan menyaring data.

while: ulangi selama syaratnya benar

`for` cocok bila Anda tahu **berapa kali** atau **daftar**nya. `while` cocok bila Anda hanya tahu **kapan harus berhenti**. Ingat diagram alur memberi pakan di Bab 2? Berikut versi programnya:

```
# Bab 8 - while: mengulang selama syarat masih benar
# (mengikuti diagram alur memberi pakan di Bab 2)
pakan_g = 2000
print(f"Tebar pakan awal: {pakan_g} g")

while pakan_g < 2400:                # ikan masih berebut
    pakan_g = pakan_g + 200
    print(f"Tambah 200 g, total {pakan_g} g")

print("Ikan sudah tenang, selesai")
```

```
Tebar pakan awal: 2000 g
Tambah 200 g, total 2200 g
Tambah 200 g, total 2400 g
Ikan sudah tenang, selesai
```

Putaran pertama: 2000 < 2400, jadi tambah 200 menjadi 2200. Putaran kedua: 2200 < 2400, tambah lagi menjadi 2400. Sekarang 2400 < 2400 **salah**, perulangan berhenti.

⚠️ AWAS

Perulangan tanpa akhir. Kalau Anda lupa mengubah variabel yang diperiksa (misalnya tidak ada baris `pakan_g = pakan_g + 200`), syaratnya tidak pernah menjadi salah dan program berputar selamanya. Jangan panik: tekan **Ctrl + C** di terminal (Bab 3) untuk membatalkannya.

break dan continue

Dua kata penolong untuk mengatur putaran:

- **continue:** lewati sisa putaran ini, langsung ke putaran berikutnya.
- **break:** berhenti total, keluar dari perulangan.

```
# Bab 8 - continue (lewati) dan break (berhenti)
bobot_timbangan_g = [480, 0, 510, 530]

for bobot in bobot_timbangan_g:
    if bobot == 0:
        print("Timbangan error, dilewati")
        continue
    print(f"Menimbang {bobot} g")
    if bobot >= 500:
        print("Ketemu ikan siap panen, berhenti")
        break
```

```
Menimbang 480 g
Timbangan error, dilewati
```

```
Menimbang 510 g
Ketemu ikan siap panen, berhenti
```

Timbangan membaca 0 (error), jadi `continue` melewatinya. Begitu ketemu ikan 510 g yang memenuhi syarat, `break` menghentikan perulangan: angka 530 tidak pernah diperiksa.

Perulangan di dalam perulangan

```
# Bab 8 - Perulangan di dalam perulangan
for blok in ["A", "B"]:
    for baris in range(1, 3):
        print(f"Blok {blok}, baris {baris}")
```

```
Blok A, baris 1
Blok A, baris 2
Blok B, baris 1
Blok B, baris 2
```

Untuk **setiap** blok, perulangan dalam berjalan **penuh**. Pola ini berguna untuk data berbentuk tabel: blok dan baris, kolam dan hari, petak dan bulan.

Jebakan: range tidak mau desimal

```
# Bab 8 - Salah: range() hanya menerima bilangan bulat
for i in range(2.5):
    print(i)
```

```
Traceback (most recent call last):
  File "07_range_desimal_salah.py", line 2, in <module>
    for i in range(2.5):
                ~~~~~^~~~~
TypeError: 'float' object cannot be interpreted as an integer
```

`TypeError` berarti tipe datanya tidak cocok: `range()` hanya mau bilangan bulat. Bila Anda butuh langkah desimal, pakai `while`.

Proyek Bersambung B2: rekap panen kebun

Sekarang kita gabungkan perulangan, penampung, dan keputusan (Bab 7) untuk merekap panen empat blok kebun:

```
# Bab 8 - Proyek bersambung B2: rekap panen kebun per blok
nama_blok = ["A", "B", "C", "D"]
panen_kg = [245.5, 310.0, 198.5, 276.0]

total_kg = 0
blok_terbaik = nama_blok[0]
panen_terbaik = panen_kg[0]

for i in range(len(nama_blok)):
    print(f"Blok {nama_blok[i]}: {panen_kg[i]} kg")
    total_kg = total_kg + panen_kg[i]
    if panen_kg[i] > panen_terbaik:
        panen_terbaik = panen_kg[i]
        blok_terbaik = nama_blok[i]

print(f"Total: {total_kg} kg")
```

```
print(f"Rata-rata: {total_kg / len(nama_blok)} kg per blok")
print(f"Blok terbaik: {blok_terbaik} ({panen_terbaik} kg)")
```

```
Blok A: 245.5 kg
Blok B: 310.0 kg
Blok C: 198.5 kg
Blok D: 276.0 kg
Total: 1030.0 kg
Rata-rata: 257.5 kg per blok
Blok terbaik: B (310.0 kg)
```

Tiga hal terjadi dalam satu perulangan: mencetak tiap blok, menambah `total_kg`, dan memperbarui “juara” lewat `if`. Setelah perulangan selesai, program mencetak total, rata-rata ($1030,0 \div 4 = 257,5$), dan blok terbaik.

Latihan

1. Cetak tabel perkalian 7 (7×1 sampai 7×10) dengan `for` dan `range()`.
2. Diberikan bobot sapi `[350, 412.5, 389.0, 401.5]`, hitung total dan rata-ratanya dengan pola penampung.
3. Ubah `04_while.py` supaya menambah 150 g tiap putaran, mulai dari 1800 g, berhenti saat mencapai atau melewati 2400 g. Berapa kali ia menambah, dan berapa total akhirnya?
4. **Tantangan:** dari daftar `[480, 0, 510, 530, 0, 495]`, hitung jumlah timbangan yang **valid** (bukan 0) dan rata-rata bobot yang valid, memakai `continue`.

Ringkasan Bab

- **for x in daftar:** mengulang untuk setiap isi; **range(a, b)** membuat deretan angka dan berhenti sebelum b.
- **while syarat:** mengulang selama syarat benar; pastikan ada yang membuatnya akhirnya salah.
- **continue** melewati satu putaran; **break** menghentikan seluruh perulangan.
- **Pola penampung:** mulai dari 0, lalu tambahkan di dalam perulangan.
- Baris yang **menjorok** adalah isi perulangan.
- Bingung dengan perulangan tanpa akhir? **Ctrl + C**.

Selanjutnya, di Bab 9: kita sudah memakai daftar `[...]` tanpa banyak penjelasan. Sekarang kita kenali keluarga lengkapnya: list, tuple, dict, dan set, yaitu wadah untuk data pohon, ikan, dan ternak.

Bab 9 — Struktur Data Bawaan: Wadah untuk Data Pohon, Ikan, dan Ternak

Seri B2 — Fondasi Bahasa Python

Di bab ini, Anda akan:

- Memakai **list** untuk daftar yang bisa diubah
- Mengambil potongan list dengan **slicing**
- Menenal **tuple** (daftar tetap), **dict** (data berlabel), dan **set** (tanpa duplikat)
- Membuat list baru dalam satu baris dengan **comprehension**
- Menyimpan data beberapa pohon sekaligus

Semua skrip bab ini ada di folder `kode/bab09/`. Di versi web, Anda juga bisa [mengunduh semuanya sekaligus](#) ([zip](#)).

Sampai Bab 8, setiap nilai kita simpan dalam satu variabel: `bobot_g = 520`. Tetapi data lapangan jumlahnya banyak: ratusan pohon, puluhan kolam. Membuat ratusan variabel jelas tidak mungkin. Python menyediakan **struktur data**: wadah yang menampung banyak nilai sekaligus. Bayangkan lemari arsip dengan empat jenis laci yang berbeda.

Wadah	Bentuk	Bisa diubah?	Cocok untuk
list	<code>[800, 1250, 640]</code>	ya	daftar berurutan: jumlah ikan tiap kolam
tuple	<code>(-6.6, 106.8)</code>	tidak	data tetap: koordinat
dict	<code>{"jenis": "Jati"}</code>	ya	data berlabel: satu pohon
set	<code>{"lele", "nila"}</code>	ya	kumpulan tanpa duplikat: jenis ikan

list: daftar yang bisa diubah

```
# Bab 9 - list: daftar yang bisa diubah
ikan_per_kolam = [800, 1250, 640]

print(ikan_per_kolam)
print("Kolam pertama:", ikan_per_kolam[0])      # nomor urut mulai dari 0
print("Kolam terakhir:", ikan_per_kolam[-1])     # -1 = yang paling belakang
print("Jumlah kolam:", len(ikan_per_kolam))
print("Total ikan:", sum(ikan_per_kolam))
print("Kolam terpadat:", max(ikan_per_kolam))
```

```
[800, 1250, 640]
Kolam pertama: 800
Kolam terakhir: 640
Jumlah kolam: 3
Total ikan: 2690
Kolam terpadat: 1250
```

Tiga hal penting:

- List ditulis dalam **kurung siku** [], isinya dipisah koma.
- Nomor urut (**indeks**) **mulai dari 0**: isi pertama di [0]. Nomor negatif menghitung dari belakang: [-1] adalah yang terakhir.
- Fungsi bawaan yang berguna: len() (banyaknya), sum() (jumlah), max() (terbesar), min() (terkecil).

Isi list bisa diubah setelah dibuat:

```
# Bab 9 - Mengubah isi list
ikan_per_kolam = [800, 1250, 640]

ikan_per_kolam.append(900)      # tambah di belakang
ikan_per_kolam[0] = 820        # ganti isi nomor 0
ikan_per_kolam.remove(640)     # buang nilai 640
print(ikan_per_kolam)

print(1250 in ikan_per_kolam)  # apakah ada?
print(500 in ikan_per_kolam)

ikan_per_kolam.sort()          # urutkan dari kecil
print(ikan_per_kolam)
```

```
[820, 1250, 900]
True
False
[820, 900, 1250]
```

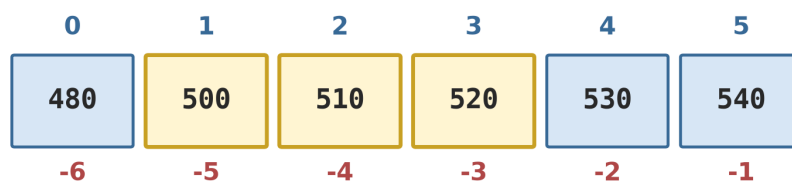
- append(x) menambah di **belakang**; remove(x) membuang nilai pertama yang cocok.
- daftar[0] = 820 mengganti isi nomor 0.
- x in daftar menjawab True atau False: ada tidaknya nilai itu.
- sort() mengurutkan list **di tempat** (list aslinya berubah).

Slicing: mengambil potongan

Rumusnya daftar[mulai:berhenti:langkah]. Seperti range(), angka berhenti **tidak ikut**.

List punya dua cara penomoran

nomor dari depan



nomor dari belakang

bobot_g[1:4] -> [500, 510, 520]

Gambar 0.1: Nomor urut list ada dua: dari depan (mulai 0) dan dari belakang (mulai -1). Potongan [1:4] mengambil nomor 1, 2, 3.

```
# Bab 9 - Slicing: mengambil potongan list [mulai:berhenti:langkah]
bobot_g = [480, 500, 510, 520, 530, 540]

print(bobot_g[1:4])      # nomor 1 sampai 3 (4 tidak ikut)
print(bobot_g[:3])       # tiga pertama
```

```
print(bobot_g[3:])      # dari nomor 3 sampai habis
print(bobot_g[::2])     # loncat 2
print(bobot_g[::-1])    # dibalik
```

```
[500, 510, 520]
[480, 500, 510]
[520, 530, 540]
[480, 510, 530]
[540, 530, 520, 510, 500, 480]
```

Bagian yang dikosongkan berarti “dari awal” atau “sampai akhir”. Trik `[::-1]` membalik seluruh urutan.

tuple: daftar yang dikunci

Tuple mirip list, tetapi ditulis dengan **kurung biasa** dan **tidak bisa diubah** setelah dibuat. Cocok untuk data yang memang tidak boleh berubah, seperti koordinat lokasi.

```
# Bab 9 - tuple: daftar yang tidak boleh diubah (cocok untuk koordinat)
lokasi_kolam = (-6.6, 106.8)      # (lintang, bujur): angka contoh

lintang, bujur = lokasi_kolam     # membongkar isi tuple
print(f"Lintang: {lintang}")
print(f"Bujur: {bujur}")
print(lokasi_kolam[0])
```

```
Lintang: -6.6
Bujur: 106.8
-6.6
```

Baris `lintang, bujur = lokasi_kolam` disebut **unpacking**: membongkar isi tuple ke dalam beberapa variabel sekaligus. Mencoba mengubah isinya akan gagal:

```
# Bab 9 - Salah: mencoba mengubah isi tuple
lokasi_kolam = (-6.6, 106.8)
lokasi_kolam[0] = -6.7
```

```
Traceback (most recent call last):
  File "08_tuple_salah.py", line 3, in <module>
    lokasi_kolam[0] = -6.7
    ~~~~~^~^
TypeError: 'tuple' object does not support item assignment
```

Pesannya jelas: “objek tuple tidak mendukung penugasan item”. Itu justru fungsi tuple: melindungi data dari perubahan tak sengaja.

dict: data berlabel

Bila list diakses dengan **nomor**, dict diakses dengan **label** (disebut *kunci*). Sempurna untuk menggambarkan satu pohon:

```
# Bab 9 - dict: data berlabel (kunci -> nilai)
pohon = {"jenis": "Jati", "diameter_cm": 32.5, "tinggi_m": 18}

print(pohon["jenis"])
pohon["tinggi_m"] = 19      # ubah nilai
pohon["sehat"] = True       # tambah label baru
print(pohon)
print(pohon.get("warna", "belum dicatat")) # aman bila label tidak ada
```

```
for label, isi in pohon.items():
    print(f"{label}: {isi}")
```

```
Jati
{'jenis': 'Jati', 'diameter_cm': 32.5, 'tinggi_m': 19, 'sehat': True}
belum dicatat
jenis: Jati
diameter_cm: 32.5
tinggi_m: 19
sehat: True
```

- Ditulis dalam **kurung kurawal { }**, tiap isi berpasangan "kunci": nilai.
- `pohon["jenis"]` mengambil nilainya; `pohon["sehat"] = True` menambah atau mengubah.
- `.get("warna", "belum dicatat")` aman: bila label tidak ada, ia mengembalikan nilai cadangan.
- `.items()` memberi pasangan label dan isi untuk dipakai di `for`.

Bila label tidak ada dan Anda memakai kurung siku, Python protes:

```
# Bab 9 - Salah: meminta label yang tidak ada di dict
pohon = {"jenis": "Jati", "diameter_cm": 32.5}
print(pohon["warna"])
```

```
Traceback (most recent call last):
  File "09_label_tak_ada.py", line 3, in <module>
    print(pohon["warna"])
    ~~~~~^~~~~~
KeyError: 'warna'
```

`KeyError` berarti “*kunci tidak ditemukan*”, dan Python menyebut nama kunci yang gagal.

set: kumpulan tanpa duplikat

Set otomatis membuang duplikat dan **tidak punya urutan**. Berguna untuk pertanyaan seperti “jenis ikan apa saja yang ada?”

```
# Bab 9 - set: kumpulan tanpa duplikat dan tanpa urutan
ikan_kolam_a = {"lele", "nila", "gurami", "lele"}
ikan_kolam_b = {"nila", "patin"}

print("Banyak jenis di kolam A:", len(ikan_kolam_a)) # lele hanya dihitung sekali
print(sorted(ikan_kolam_a))
print("Ada nila di kolam A?", "nila" in ikan_kolam_a)
print("Ada di kedua kolam:", sorted(ikan_kolam_a & ikan_kolam_b))
print("Ada di salah satu:", sorted(ikan_kolam_a | ikan_kolam_b))
```

```
Banyak jenis di kolam A: 3
['gurami', 'lele', 'nila']
Ada nila di kolam A? True
Ada di kedua kolam: ['nila']
Ada di salah satu: ['gurami', 'lele', 'nila', 'patin']
```

"lele" ditulis dua kali tetapi hanya dihitung sekali. Operator `&` memberi **irisan** (ada di kedua kolam) dan `|` memberi **gabungan**. Kita memakai `sorted()` supaya tampilannya berurutan, karena set sendiri tidak menjamin urutan.

Comprehension: list baru dalam satu baris

```
# Bab 9 - Comprehension: membuat list baru dalam satu baris
bobot_g = [520, 480, 560, 495]
```

```

bobot_kg = [g / 1000 for g in bobot_g]
siap_panen = [g for g in bobot_g if g >= 500]

print(bobot_kg)
print(siap_panen)

```

```

[0.52, 0.48, 0.56, 0.495]
[520, 560]

```

Baca `[g / 1000 for g in bobot_g]` seperti kalimat: “*untuk setiap g di bobot_g, hitung g / 1000.*” Menambahkan `if` di belakang menyaring: `[g for g in bobot_g if g >= 500]` hanya mengambil yang ≥ 500 .

TIPS

Comprehension itu singkatan dari perulangan `for` yang mengisi list. Kalau terasa membingungkan, tulis dulu dengan `for` biasa. Setelah paham, baru dipadatkan.

Proyek Bersambung B2: data pohon

Sekarang gabungkan: **list berisi dict**. Tiap pohon adalah satu dict, dan kebunnya adalah list dari dict-dict itu.

```

# Bab 9 - Proyek bersambung B2: data pohon dalam list berisi dict
data_pohon = [
    {"jenis": "Jati", "diameter_cm": 32.5, "tinggi_m": 18, "sehat": True},
    {"jenis": "Meranti", "diameter_cm": 45.2, "tinggi_m": 27, "sehat": True},
    {"jenis": "Mahoni", "diameter_cm": 28.0, "tinggi_m": 15, "sehat": False},
]

for pohon in data_pohon:
    status = "sehat" if pohon["sehat"] else "perlu diperiksa"
    print(f"{pohon['jenis']}:8} diameter {pohon['diameter_cm']:5.1f} cm, {status}")

diameter = [pohon["diameter_cm"] for pohon in data_pohon]
print(f"Rata-rata diameter: {sum(diameter) / len(diameter):.2f} cm")

sehat_saja = [pohon["jenis"] for pohon in data_pohon if pohon["sehat"]]
print("Pohon sehat:", sehat_saja)

```

```

Jati      diameter 32.5 cm, sehat
Meranti   diameter 45.2 cm, sehat
Mahoni     diameter 28.0 cm, perlu diperiksa
Rata-rata diameter: 35.23 cm
Pohon sehat: ['Jati', 'Meranti']

```

Beberapa hal baru di baris `print`:

- `{pohon['jenis']:8}` berarti “tampilkan dengan lebar 8 karakter”, supaya kolomnya rata. Tanda kutip tunggal dipakai di dalamnya karena f-string-nya sudah dibungkus kutip ganda.
- `{pohon['diameter_cm']:5.1f}` berarti lebar 5, satu angka di belakang titik.
- `"sehat" if pohon["sehat"] else "perlu diperiksa"` adalah **if satu baris**: nilainya “sehat” bila benar, kalau tidak yang satunya.

Rata-rata diameter: $(32,5 + 45,2 + 28,0) \div 3 = 35,23$ **cm**. Pohon Mahoni tidak sehat, jadi tidak masuk daftar “Pohon sehat”.

Latihan

1. Buat list berisi bobot lima ekor sapi (angka bebas). Cetak bobot terberat, bobot teringan, dan rata-ratanya.
2. Dari list `[480, 500, 510, 520, 530, 540]`, ambil dengan slicing: tiga terakhir, dan tiga angka pertama dibalik.
3. Buat dict untuk satu kolam dengan label `nama`, `jumlah_ikan`, `berat_rata_g`, dan `siap_panen` (hasil perbandingan). Cetak semua isinya dengan `for` dan `.items()`.
4. **Tantangan:** tambahkan satu pohon lagi ke `10_proyek_data_pohon.py`, lalu buat list comprehension berisi jenis pohon yang tingginya lebih dari 20 m.

Ringkasan Bab

- **list** `[...]`: berurutan, bisa diubah; indeks mulai dari **0**, negatif dari belakang.
- **Slicing** `[mulai:berhenti:langkah]`; angka berhenti tidak ikut.
- **tuple** `(...)`: tidak bisa diubah; **dict** `{kunci: nilai}`: data berlabel; **set** `{...}`: tanpa duplikat dan tanpa urutan.
- `len()`, `sum()`, `max()`, `min()` bekerja pada daftar.
- **Comprehension** membuat dan menyaring list dalam satu baris.
- **List berisi dict** adalah cara lazim menyimpan banyak catatan lapangan.

Selanjutnya, di Bab 10: kita sudah punya data dan cara mengolahnya. Sekarang kita belajar membungkus pekerjaan berulang menjadi **fungsi**, supaya cukup ditulis sekali dan dipakai berkali-kali.

Bab 10 — Fungsi: Menulis Sekali, Memakai Berkali-kali

Seri B2 — Fondasi Bahasa Python

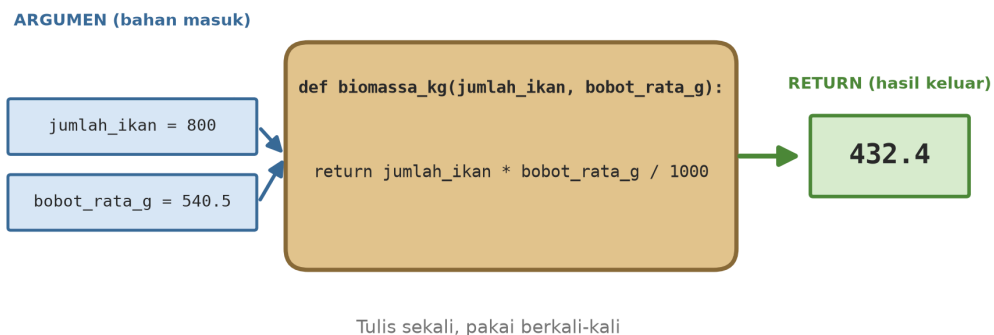
Di bab ini, Anda akan:

- Membuat fungsi sendiri dengan **def**
- Memberi bahan masuk (**argumen**) dan menerima hasil (**return**)
- Memahami **scope**: variabel di dalam dan di luar fungsi
- Mengenal docstring dan lambda
- Menyelesaikan pencatat data lapangan: proyek Seri B2

Semua skrip bab ini ada di folder `kode/bab10/`. Di versi web, Anda juga bisa [mengunduh semuanya sekaligus \(zip\)](#).

Di Bab 6, Anda menghitung biomassa kolam. Bayangkan Anda punya 12 kolam. Menyalin rumusnya 12 kali itu rawan salah, dan bila rumusnya perlu diperbaiki, Anda harus memperbaiki 12 tempat. Solusinya: bungkus rumus itu menjadi **fungsi**, tulis sekali, panggil berkali-kali. Anda sebenarnya sudah memakai fungsi sejak Bab 4: `print()`, `input()`, `len()`, dan `sum()` semuanya fungsi. Sekarang Anda membuatnya sendiri.

Fungsi = mesin kecil: bahan masuk, hasil keluar



Gambar 0.1: Fungsi seperti mesin kecil: bahan masuk (argumen), hasil keluar (return).

def: membuat fungsi

```
# Bab 10 - def: membuat fungsi sendiri  
def sapa_petani():  
    print("Selamat pagi, petani!")  
    print("Semoga panen hari ini melimpah")  
  
sapa_petani()      # memanggil fungsi  
sapa_petani()      # bisa dipanggil berkali-kali
```

```
Selamat pagi, petani!  
Semoga panen hari ini melimpah  
Selamat pagi, petani!  
Semoga panen hari ini melimpah
```

- `def` menandai pembuatan fungsi, diikuti nama dan tanda kurung, lalu **titik dua**.
- Isinya **menjorok** (indentasi dari Bab 7).
- Fungsi baru dikerjakan saat **dipanggil**: `sapa_petani()`. Membuatnya saja belum menjalankan apa-apa.

TIPS

Beri nama fungsi dengan **kata kerja** yang menjelaskan tugasnya: `hitung_biomassa`, `cetak_rekap`. Orang yang membaca kode Anda langsung paham apa yang dikerjakan tanpa membaca isinya.

Argumen dan return

Fungsi menjadi berguna bila menerima **bahan** dan memberi **hasil**:

```
# Bab 10 - Argumen (bahan masuk) dan return (hasil keluar)  
def biomassa_kg(jumlah_ikan, bobot_rata_g):  
    return jumlah_ikan * bobot_rata_g / 1000  
  
total = biomassa_kg(800, 540.5)  
print(f"Biomassa kolam A: {total:.1f} kg")  
  
# Argumen boleh diberi nama supaya lebih jelas  
print(biomassa_kg(jumlah_ikan=1250, bobot_rata_g=300))
```

```
Biomassa kolam A: 432.4 kg  
375.0
```

- Nama di dalam kurung (`jumlah_ikan`, `bobot_rata_g`) disebut **parameter**: tempat bahan diterima. Nilai yang Anda berikan saat memanggil (`800`, `540.5`) disebut **argumen**.
- **return** mengirim hasil kembali ke pemanggil. Hasilnya bisa disimpan di variabel (`total`) atau langsung dipakai.
- Argumen boleh diberi nama (`jumlah_ikan=1250`) supaya jelas, dan urutannya bebas.

Nilai bawaan

Parameter boleh diberi **nilai bawaan** (*default*), dipakai bila pemanggil tidak menyebutkannya:

```
# Bab 10 - Nilai bawaan (default) untuk argumen  
# (persen pakan 3% adalah angka contoh untuk latihan, bukan anjuran)  
def pakan_harian_kg(biomassa, persen=3):  
    return biomassa * persen / 100  
  
print(f"Pakai persen bawaan: {pakan_harian_kg(432.4):.2f} kg")  
print(f"Persen diganti 2.5: {pakan_harian_kg(432.4, persen=2.5):.2f} kg")
```

```
Pakai persen bawaan: 12.97 kg  
Persen diganti 2.5: 10.81 kg
```

Pemanggilan pertama memakai `persen=3` bawaan. Pemanggilan kedua menggantinya menjadi `2.5`. Parameter dengan nilai bawaan harus ditulis **setelah** parameter tanpa nilai bawaan.

Scope: siapa bisa melihat variabel apa

Variabel yang dibuat **di dalam** fungsi bersifat **lokal**: hanya hidup selama fungsi berjalan.

```
# Bab 10 - Scope: variabel di dalam fungsi tidak mengubah yang di luar
suhu_air = 28

def cek_suhu():
    suhu_air = 31          # variabel BARU, hanya hidup di dalam fungsi
    print("Di dalam fungsi:", suhu_air)

cek_suhu()
print("Di luar fungsi:", suhu_air)
```

```
Di dalam fungsi: 31
Di luar fungsi: 28
```

Di dalam fungsi, `suhu_air = 31` membuat variabel **baru** yang kebetulan bernama sama. Variabel `suhu_air` di luar tetap 28. Ini melindungi program: fungsi tidak bisa tanpa sengaja mengubah data di luarnya.

⚠ AWAS

Karena variabel lokal “mati” setelah fungsi selesai, Anda tidak bisa memakainya dari luar:

```
# Bab 10 - Salah: memakai variabel lokal dari luar fungsi
def hitung():
    hasil_lokal = 10

hitung()
print(hasil_lokal)
```

```
Traceback (most recent call last):
  File "08_variabel_lokal_salah.py", line 7, in <module>
    print(hasil_lokal)
    ~~~~~^
NameError: name 'hasil_lokal' is not defined
```

Kalau butuh hasil dari dalam fungsi, **kembalikan dengan `return`**, lalu tangkap di luar.

Mengembalikan beberapa nilai

Satu fungsi bisa mengembalikan lebih dari satu nilai sekaligus (sebenarnya sebuah tuple dari Bab 9), yang bisa langsung dibongkar:

```
# Bab 10 - Satu fungsi bisa mengembalikan beberapa nilai
def ringkas(daftar):
    return min(daftar), max(daftar), sum(daftar) / len(daftar)

bobot_g = [520, 480, 560, 540]
terkecil, terbesar, rata_rata = ringkas(bobot_g)
print(f"Terkecil: {terkecil} g")
print(f"Terbesar: {terbesar} g")
print(f"Rata-rata: {rata_rata:.1f} g")
```

```
Terkecil: 480 g
Terbesar: 560 g
Rata-rata: 525.0 g
```

Docstring dan lambda

Docstring adalah kalimat di dalam tiga tanda kutip pada baris pertama fungsi. Ia menjelaskan tugas fungsi, dan bisa dibaca lewat `__doc__`. **Lambda** adalah fungsi mini satu baris tanpa nama, sering dipakai sesaat, misalnya sebagai aturan pengurutan.

```
# Bab 10 - Docstring (catatan fungsi) dan lambda (fungsi mini)
def luas_petak_m2(panjang_m, lebar_m):
    """Menghitung luas petak persegi panjang dalam meter persegi."""
    return panjang_m * lebar_m

print(luas_petak_m2.__doc__)
print(luas_petak_m2(25, 12))

ikan = [("lele", 520), ("nila", 480), ("gurami", 610)]
urut_bobot = sorted(ikan, key=lambda item: item[1]) #urut menurut bobot
print(urut_bobot)
```

```
Menghitung luas petak persegi panjang dalam meter persegi.
300
[('nila', 480), ('lele', 520), ('gurami', 610)]
```

Pada `sorted(ikan, key=lambda item: item[1])`, `lambda` itu berarti “ambil isi nomor 1 dari tiap pasangan (bobotnya) dan urutkan berdasarkan itu”. Hasilnya dari bobot terkecil ke terbesar.

Jebakan klasik: lupa return

```
# Bab 10 - Jebakan: fungsi tanpa return menghasilkan None
def total_pakan(pagi_kg, sore_kg):
    pagi_kg + sore_kg # dihitung, tapi tidak dikembalikan

hasil = total_pakan(2, 3)
print(hasil)
```

```
None
```

Tidak ada error, tetapi hasilnya **None** (“tidak ada”). Fungsi menghitung `pagi_kg + sore_kg`, lalu membuangnya karena tidak ada `return`. Ini bug yang licin karena Python tidak memprotes.

i INGAT

Fungsi tanpa `return` selalu menghasilkan **None**. Kalau hasil fungsi Anda tiba-tiba **None**, periksa dulu apakah Anda lupa `return`.

Proyek Bersambung B2: pencatat data lapangan

Inilah puncak Seri B2: semua bahan yang Anda pelajari bekerja bersama. Ada **dict** berisi **list** (Bab 9), dua **fungsi** (bab ini), sebuah **perulangan** (Bab 8), dan **f-string** dengan format kolom (Bab 5 dan 9).

```
# Bab 10 - Proyek bersambung B2: pencatat data lapangan (rekap per sektor)
def jumlah(angka):
```

```

"""Menjumlahkan semua angka dalam daftar."""
total = 0
for a in angka:
    total = total + a
return total

def rata_rata(angka):
    """Rata-rata daftar angka (daftar tidak boleh kosong)."""
    return jumlah(angka) / len(angka)

data_lapangan = {
    "Pohon (diameter cm)": [32.5, 45.2, 28.0, 51.5],
    "Panen kebun (kg)": [245.5, 310.0, 198.5, 276.0],
    "Ikan (gram)": [520, 480, 560, 540],
    "Ternak (kg)": [350, 412.5, 389.0],
}

for sektor, nilai in data_lapangan.items():
    print(f"{sektor:20} total {jumlah(nilai):8.1f} rata-rata {rata_rata(nilai):7.2f}")

```

Pohon (diameter cm)	total	157.2	rata-rata	39.30
Panen kebun (kg)	total	1030.0	rata-rata	257.50
Ikan (gram)	total	2100.0	rata-rata	525.00
Ternak (kg)	total	1151.5	rata-rata	383.83

Perhatikan bahwa `rata_rata()` memanggil `jumlah()`: fungsi boleh memakai fungsi lain. Menambah sektor baru (misalnya “Telur (butir)”) cukup menambah satu baris di `data_lapangan`, tanpa mengubah fungsi atau perulangan. Itulah kekuatan fungsi.

Latihan

1. Tulis fungsi `luas_lingkaran_m2(jari_jari_m)` yang mengembalikan luas (pakai 3.14 untuk π). Panggil untuk jari-jari 3 dan 5.
2. Tulis fungsi `status_panen(bobot_g)` yang mengembalikan “siap” bila bobot ≥ 500 dan “belum” bila tidak. Pakai di dalam perulangan atas list [480, 520, 505, 495].
3. Ubah `09_proyek_pencatat.py`: tambah sektor “Telur (butir)” dengan tiga angka, dan tambah fungsi `terbesar(angka)` yang dipakai untuk mencetak nilai terbesar tiap sektor. (Jangan pakai `max()`; tulis perulangannya sendiri.)
4. **Tantangan:** tulis fungsi `rata_rata_aman(angka)` yang mengembalikan `None` bila daftarnya kosong, sehingga tidak kena `ZeroDivisionError`. Gunakan `if` dari Bab 7.

Ringkasan Bab

- **def nama(parameter):** membuat fungsi; isi **menjorok**. Fungsi baru berjalan saat **dipanggil**.
- **Argumen** = bahan masuk; **return** = hasil keluar. Tanpa **return**, hasilnya **None**.
- Parameter boleh punya **nilai bawaan**.
- Variabel di dalam fungsi bersifat **lokal** dan tidak terlihat dari luar.
- Fungsi bisa mengembalikan beberapa nilai, memanggil fungsi lain, dan diberi **docstring**.
- **Lambda** = fungsi mini satu baris.

Penutup Seri B2

Selamat! Anda sudah menguasai **seluruh bahan dasar** bahasa Python: variabel dan tipe data, operator, keputusan, perulangan, struktur data, dan fungsi. Dengan itu Anda sudah bisa menulis pencatat data lapangan kecil.

Tetapi ada satu kekurangan besar: **data hilang setiap program ditutup**. Semua angka di `data_lapangan` harus diketik ulang di kode setiap kali. Di Seri B3, program Anda belajar **membaca dan**

menyimpan file: modul dan pustaka (Bab 11), penanganan error (Bab 12), dan file serta format data (Bab 13 dan 14).

Selanjutnya, di Bab 11: modul dan package, cara meminjam alat siap pakai dari gudang Python.

Bab 11 — Modul dan Package: Meminjam Alat dari Gudang Python

Seri B3 — Program Kecil yang Berguna

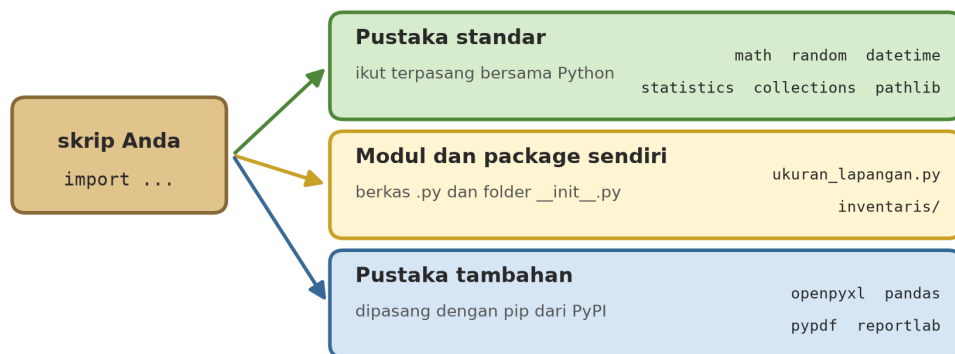
Di bab ini, Anda akan:

- Memakai `import` untuk meminjam alat siap pakai dari **pustaka standar** Python
- Mengenal modul yang paling sering berguna: `math`, `random`, `datetime`, `statistics`, `collections`, `itertools`, `pathlib`
- Membuat **modul** sendiri dan memahami `if __name__ == "__main__"`
- Menyusun **package** kecil dengan `__init__.py`
- Mengenal `pip`, pemasang pustaka tambahan, dan memulai proyek Seri B3

Semua skrip bab ini ada di folder `kode/bab11/`. Di versi web, Anda juga bisa [mengunduh semuanya sekaligus](#) (zip).

Di Bab 10, Anda menulis fungsi sendiri. Tetapi petani tidak membuat cangkul dari besi mentah setiap kali mau mencangkul: ia mengambilnya dari gudang alat. Python punya **gudang alat** yang besar, dan setiap rak di dalamnya disebut **modul**: sebuah berkas Python berisi fungsi dan data siap pakai. Gudang bawaan yang ikut terpasang bersama Python disebut **pustaka standar** (*standard library*). Anda sudah mengintipnya di Bab 4 saat memakai `import`; sekarang kita masuk ke gudangnya.

Gudang modul: semua dipinjam dengan import



Gambar 0.1: Gudang modul: alat bawaan, alat buatan sendiri, dan alat tambahan dari pip, semuanya dipinjam dengan import.

Tiga cara memakai import

```
# Bab 11 - Tiga cara memakai import
import math                                # cara 1: bawa seluruh modul, panggil dengan
awalan math.
from math import sqrt                      # cara 2: ambil satu alat saja
import statistics as stat                  # cara 3: beri nama panggilan (alias)

print("pi          =", math.pi)
print("akar 144    =", sqrt(144))
```

```
print("rata-rata   =", stat.mean([32.5, 45.2, 28.0]))

# Luas tajuk pohon berbentuk lingkaran dengan jari-jari 3,5 m
luas = math.pi * 3.5 ** 2
print(f"Luas tajuk   = {luas:.2f} m2")
```

```
pi           = 3.141592653589793
akar 144     = 12.0
rata-rata    = 35.233333333333334
Luas tajuk   = 38.48 m2
```

- `import math` membawa seluruh modul; alatnya dipanggil dengan awalan: `math.pi`.
- `from math import sqrt` mengambil **satu alat** saja, lalu dipakai tanpa awalan.
- `import statistics as stat` memberi **nama panggilan** (alias) agar lebih pendek.

Awalan itu berguna: `math.pi` langsung memberi tahu bahwa `pi` berasal dari modul `math`, bukan variabel Anda sendiri.

TIPS

Tulis semua `import` di **bagian paling atas** skrip. Pembaca langsung tahu alat apa yang dibutuhkan program.

math dan random

```
# Bab 11 - math dan random (dengan benih supaya hasilnya bisa diulang)
import math
import random

print(math.ceil(4.2), math.floor(4.8), round(4.5), round(5.5))    # pembulatan

random.seed(42)                                                    # benih: urutan "acak" jadi tetap
plot_terpilih = random.choice(["A", "B", "C"])
sampel = random.sample(range(1, 101), 5)                          # 5 nomor pohon berbeda dari 1..100
cuaca = random.randint(24, 33)                                     # bilangan bulat 24..33
print("Plot terpilih:", plot_terpilih)
print("Pohon contoh :", sampel)
print("Suhu tiruan   :", cuaca)

random.seed(42)                                                    # benih yang sama -> hasil yang sama
print("Ulang        :", random.choice(["A", "B", "C"]), random.sample(range(1, 101), 5))
```

```
5 4 4 6
Plot terpilih: C
Pohon contoh : [15, 4, 95, 36, 32]
Suhu tiruan   : 27
Ulang        : C [15, 4, 95, 36, 32]
```

Baris pertama menunjukkan kejutan: `round(4.5)` menghasilkan 4, tetapi `round(5.5)` menghasilkan 6. Python membulatkan angka yang tepat di tengah ke bilangan **genap** terdekat, bukan selalu ke atas. Aturan ini disebut pembulatan bankir; yang perlu Anda ingat, jangan heran bila hasilnya begitu.

Modul `random` menghasilkan angka “acak”. Dengan `random.seed(42)` kita menetapkan **benih**: titik awal urutan acak. Benih sama, urutan sama; lihat baris terakhir yang mengulang hasil C [15, 4, 95, 36, 32]. Inilah cara membuat data tiruan yang bisa diulang persis. Satu catatan jujur: dokumentasi Python hanya menjamin urutan `random.random()` tetap sama untuk benih yang sama di versi Python mendatang, sedangkan `choice` dan `sample` bisa berubah antarversi. Jadi angka di atas pasti sama di komputer Anda bila versi Python-nya sama, tetapi jangan membangun sistem yang bergantung pada angka itu.

Tanggal dengan datetime

```
# Bab 11 - datetime: tanggal tanam, umur, dan perkiraan panen
from datetime import date, timedelta

tanam = date(2026, 3, 1)
panen = tanam + timedelta(days=120)          # padi contoh: 120 hari
umur = date(2026, 5, 15) - tanam             # selisih dua tanggal = timedelta

print("Tanam :", tanam)
print("Panen :", panen.strftime("%d-%m-%Y"))
print("Umur pada 15 Mei 2026:", umur.days, "hari")
print("Dari teks:", date.fromisoformat("2026-07-01"))

nama_hari = ["Senin", "Selasa", "Rabu", "Kamis", "Jumat", "Sabtu", "Minggu"]
print("Hari panen jatuh pada:", nama_hari[panen.weekday()])
```

```
Tanam : 2026-03-01
Panen : 29-06-2026
Umur pada 15 Mei 2026: 75 hari
Dari teks: 2026-07-01
Hari panen jatuh pada: Senin
```

`date(2026, 3, 1)` membuat tanggal. Menambah `timedelta(days=120)` menggeser tanggalnya. Selisih dua tanggal adalah `timedelta`, dan `.days` mengambil jumlah harinya (75 hari dari 1 Maret sampai 15 Mei). `strftime("%d-%m-%Y")` mengatur tampilan tanggal. (Kita sengaja tidak memakai `date.today()` karena hasilnya berubah setiap hari.)

statistics dan collections

```
# Bab 11 - statistics: ringkasan angka tanpa menulis rumus sendiri
import statistics

bobot_g = [520, 480, 560, 540, 480, 505]      # bobot sampel ikan (angka contoh)
print("rata-rata :", round(statistics.mean(bobot_g), 2))
print("median    :", statistics.median(bobot_g))
print("modus     :", statistics.mode(bobot_g))
print("simpangan :", round(statistics.stdev(bobot_g), 2))
print("terkecil/terbesar:", min(bobot_g), max(bobot_g))
```

```
rata-rata : 514.17
median    : 512.5
modus     : 480
simpangan : 32.31
terkecil/terbesar: 480 560
```

Enam bobot ikan itu menghasilkan rata-rata 514,17 g dan median 512,5 g (nilai tengah setelah diurutkan). Modus 480 adalah angka yang paling sering muncul. `stdev` (simpangan baku) mengukur seberapa menyebar angkanya; makin besar, makin tidak seragam ikannya.

```
# Bab 11 - collections: Counter dan defaultdict
from collections import Counter, defaultdict

jenis = ["Jati", "Meranti", "Jati", "Mahoni", "Jati", "Meranti"]
hitung = Counter(jenis)
print(hitung)
print("Terbanyak:", hitung.most_common(1))

per_plot = defaultdict(list)                # kunci baru otomatis diberi list kosong
```

```
for plot, diameter in [("A", 32.5), ("B", 28.0), ("A", 45.2), ("B", 51.5)]:
    per_plot[plot].append(diameter)
print(dict(per_plot))
```

```
Counter({'Jati': 3, 'Meranti': 2, 'Mahoni': 1})
Terbanyak: [('Jati', 3)]
{'A': [32.5, 45.2], 'B': [28.0, 51.5]}
```

Counter menghitung berapa kali tiap nilai muncul: Jati ada 3. `defaultdict(list)` adalah dict yang otomatis membuat list kosong untuk kunci baru, sehingga Anda bisa langsung `append` tanpa memeriksa dulu apakah kuncinya sudah ada.

itertools dan pathlib

```
# Bab 11 - itertools: kombinasi dan kelompok
from itertools import combinations, product, groupby

kolam = ["A", "B", "C"]
print("Pasangan kolam:", list(combinations(kolam, 2)))
print("Kolam x pakan :", list(product(kolam[:2], ["pelet", "cacah"])))

pohon = [("A", "Jati"), ("A", "Meranti"), ("B", "Mahoni"), ("B", "Jati")]
for plot, anggota in groupby(pohon, key=lambda p: p[0]): # data harus sudah berurutan
    print(plot, [jenis for _, jenis in anggota])
```

```
Pasangan kolam: [('A', 'B'), ('A', 'C'), ('B', 'C')]
Kolam x pakan : [('A', 'pelet'), ('A', 'cacah'), ('B', 'pelet'), ('B', 'cacah')]
A ['Jati', 'Meranti']
B ['Mahoni', 'Jati']
```

`combinations` membuat semua pasangan, `product` membuat semua kombinasi dari dua daftar, dan `groupby` mengelompokkan data yang **sudah berurutan**. Perhatikan: `groupby` hanya menyatukan elemen yang berdampingan, jadi urutkan dulu datanya bila belum.

```
# Bab 11 - pathlib: alamat berkas sebagai objek (belum menyentuh disk)
from pathlib import Path

berkas = Path("data") / "pohon.csv" # tanda / menyambung bagian alamat
print(berkas)
print("nama :", berkas.name)
print("akhiran:", berkas.suffix)
print("tanpa akhiran:", berkas.stem)
print("folder :", berkas.parent)
print("ganti akhiran:", berkas.with_suffix(".json"))
print("ada di disk?", berkas.exists())
print("skrip ini bernama:", Path(__file__).name)
```

```
data\pohon.csv
nama : pohon.csv
akhiran: .csv
tanpa akhiran: pohon
folder : data
ganti akhiran: data\pohon.json
ada di disk? False
skrip ini bernama: 07_pathlib_awal.py
```

`pathlib.Path` memperlakukan alamat berkas sebagai **objek**. Tanda `/` menyambung bagian alamat. Pada komputer Windows, pemisahannya tampil sebagai `\`; di macOS atau Linux tampil sebagai `/`. Itulah

keuntungan `Path`: Anda menulis sekali, Python menyesuaikan dengan sistemnya. Di Bab 13 kita akan memakainya untuk membaca dan menulis berkas sungguhan.

Membuat modul sendiri

Modul hanyalah berkas `.py`. Modul kita ada di subfolder `modul/`:

```
# modul/ukuran_lapangan.py (dua fungsi pertamanya)
def luas_bidang_dasar_m2(diameter_cm): ...
def biomassa_kg(jumlah_ikan, bobot_rata_g): ...
```

Isi lengkapnya ada di berkas `kode/bab11/modul/ukuran_lapangan.py`. Skrip berikut memakainya:

```
# Bab 11 - Memakai modul buatan sendiri dari subfolder "modul"
import sys
from pathlib import Path

sys.path.insert(0, str(Path(__file__).parent / "modul")) # beri tahu Python lokasi
modulnya

import ukuran_lapangan as ukuran

print(f"Luas bidang dasar 32,5 cm: {ukuran.luas_bidang_dasar_m2(32.5):.4f} m2")
print(f"Biomassa kolam A      : {ukuran.biomassa_kg(800, 540.5):.1f} kg")
print("Nama modul ini:", ukuran.__name__)
```

```
Luas bidang dasar 32,5 cm: 0.0830 m2
Biomassa kolam A      : 432.4 kg
Nama modul ini: ukuran_lapangan
```

Luas bidang dasar adalah luas penampang batang; untuk diameter 32,5 cm hasilnya 0,0830 m². Python mencari modul di sejumlah folder; kita menambahkan folder `modul/` ke daftarnya lewat `sys.path.insert`. Nama modul (`ukuran_lapangan`) sama dengan nama berkas tanpa `.py`.

if __name__ == "__main__"

Setiap modul punya variabel bawaan `__name__`. Bila berkas **dijalankan langsung**, `__name__` berisi `"__main__"`. Bila berkas **di-import**, `__name__` berisi nama modulnya. Dengan begitu satu berkas bisa punya “uji cepat” yang hanya jalan saat dijalankan langsung:

```
# Bab 11 - if __name__ == "__main__": dijalankan langsung atau di-import
import subprocess
import sys
from pathlib import Path

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

import contoh_nama # di-import: __name__ = nama modul

print("--- sekarang dijalankan langsung ---")
hasil = subprocess.run([sys.executable, str(modul / "contoh_nama.py)],
                       capture_output=True, text=True)
print(hasil.stdout.strip())

print("--- uji cepat di modul ukuran_lapangan (dijalankan langsung) ---")
hasil = subprocess.run([sys.executable, str(modul / "ukuran_lapangan.py)],
                       capture_output=True, text=True)
print(hasil.stdout.strip())
```

```

Di dalam contoh_nama, __name__ = contoh_nama
--- sekarang dijalankan langsung ---
Di dalam contoh_nama, __name__ = __main__
--- uji cepat di modul ukuran_lapangan (dijalankan langsung) ---
Uji cepat: 0.083 432.4

```

Baris pertama muncul saat `contoh_nama` di-import (nama modulnya `contoh_nama`). Setelah itu skrip menjalankan berkas yang sama sebagai program tersendiri, dan kali ini `__name__` berisi `__main__`. Pada `ukuran_lapangan.py`, bagian di bawah `if __name__ == "__main__":` hanya tampil pada pemanggilan kedua; saat di-import di skrip 08, uji cepat itu **tidak** mengganggu.

⚠️ AWAS

Kode di tingkat paling luar modul **dijalankan saat di-import**. Karena itu jangan menaruh perintah panjang atau `print` di sana, kecuali di bawah `if __name__ == "__main__":`.

Package: folder berisi modul

Bila modul makin banyak, kumpulkan dalam **package**: folder yang berisi berkas `__init__.py` dan modul-modul. Package kita bernama `inventaris`:

```

inventaris/
  __init__.py    <- penanda package; boleh memuat daftar alat yang ditampilkan
  data_inti.py   <- data contoh lima sektor
  hitung.py      <- luas_bidang_dasar_m2, biomassa_kg
  ringkas.py     <- ringkas(angka)

```

```

# Bab 11 - Paket kecil: folder "inventaris" berisi __init__.py dan modul-modulnya
import inventaris
from inventaris import biomassa_kg          # tersedia lewat __init__.py
from inventaris.data_inti import ikan       # ambil langsung dari modul di dalam paket

print("Isi paket (__all__):", inventaris.__all__)
print("Modul hitung berasal dari:", inventaris.hitung.__name__)

for kolam in ikan:
    kg = biomassa_kg(kolam["jumlah"], kolam["bobot_rata_g"])
    print(f"Kolam {kolam['kolam']}: {kg:.1f} kg")

```

```

Isi paket (__all__): ['luas_bidang_dasar_m2', 'biomassa_kg', 'ringkas']
Modul hitung berasal dari: inventaris.hitung
Kolam A: 432.4 kg
Kolam B: 375.0 kg
Kolam C: 288.0 kg

```

`__init__.py` mengimpor alat dari modul-modulnya sehingga pemakai cukup menulis `from inventaris import biomassa_kg`. Kolam A berisi 800 ikan seberat 540,5 g: 432,4 kg, persis seperti hasil di Bab 10. Variabel `__all__` mencatat nama-nama yang sengaja ditampilkan paket.

pip: meminjam dari luar gudang

Bila alat yang Anda butuhkan tidak ada di pustaka standar, **pip** memasangnya dari **PyPI**, toko pustaka daring milik komunitas Python. Perintahnya diketik di terminal, bukan di dalam skrip. Perintah berikut hanya contoh dan **tidak dijalankan** saat buku ini dibuat, karena memasang paket mengubah komputer pembaca:

```
python -m pip install openpyxl # memasang pustaka Excel (dipakai di Bab 15)
python -m pip list             # melihat pustaka yang terpasang
python -m pip uninstall openpyxl # mencopot
```

Menulis `python -m pip` (bukan sekadar `pip`) memastikan `pip` yang dipakai berasal dari Python yang sama dengan yang menjalankan skrip Anda. Bila Anda mencoba mengimpor pustaka yang belum terpasang, ini yang terjadi:

```
# Bab 11 - Jebakan: mengimpor modul yang tidak ada
import pustaka_tak_ada

print("baris ini tidak pernah dicapai")
```

```
Traceback (most recent call last):
  File "11_modul_tak_ketemu.py", line 2, in <module>
    import pustaka_tak_ada
ModuleNotFoundError: No module named 'pustaka_tak_ada'
```

Pesan `ModuleNotFoundError` berarti Python tidak menemukan modul bernama itu: bisa karena salah ketik, atau belum dipasang dengan `pip`. Pembahasan lengkap pemasangan dan lingkungan terpisah (`venv`) ada di Bab 17.

BAGIAN TEKNIS (boleh dilewati)

Saat Anda menulis `import x`, Python mencari `x` berurutan di: modul bawaan yang sudah ada, folder tempat skrip dijalankan, lalu daftar folder lain di `sys.path`. Karena folder skrip ada di urutan awal, Anda sebaiknya **tidak menamai berkas sendiri** `random.py` atau `statistics.py`: modul buatan Anda akan menutupi modul asli dan menimbulkan error yang membingungkan.

Proyek Bersambung B3: ringkasan lima sektor

Selamat datang di proyek Seri B3. Di Seri B2, data lapangan tersimpan di dalam kode. Di seri ini kita akan menyimpannya ke file, membuatnya tahan salah, dan mengubahnya menjadi laporan. Langkah pertama: kemas alat-alat kita menjadi package `inventaris` (yang sudah Anda lihat), lalu pakai untuk meringkas data empat tabel dari lima sektor (hutan, kebun, pertanian, perikanan, peternakan). Angka-angkanya fiktif.

```
# Bab 11 - Proyek bersambung B3: ringkasan lima sektor memakai paket sendiri
from inventaris import ringkas
from inventaris.data_inti import pohon, tanaman, ikan, ternak

sektor = {
    "Hutan (diameter cm)": [p["diameter_cm"] for p in pohon],
    "Kebun+sawah (kg)": [t["hasil_kg"] for t in tanaman],
    "Perikanan (g/ekor)": [i["bobot_rata_g"] for i in ikan],
    "Pernakan (kg)": [t["bobot_kg"] for t in ternak],
}

for nama, angka in sektor.items():
    r = ringkas(angka)
    print(f"{nama:22}      n={r['jumlah']}          total={r['total']:8.1f}
rata={r['rata_rata']:7.2f}  median={r['median']}")
```

Hutan (diameter cm)	n=6	total=	234.0	rata=	39.00	median=38.4
Kebun+sawah (kg)	n=4	total=	1030.0	rata=	257.50	median=260.75
Perikanan (g/ekor)	n=3	total=	1320.5	rata=	440.17	median=480.0
Pernakan (kg)	n=4	total=	1190.0	rata=	297.50	median=369.5

Tiap baris dibaca begini: sektor hutan punya 6 pohon, total diameter 234,0 cm, rata-rata 39,00 cm, median 38,4 cm. Kebun dan sawah (padi, jagung, kopi, kakao) menghasilkan 1030,0 kg. Rata-rata bobot ikan per kolam 440,17 g, dan ternak rata-rata 297,50 kg. Perhatikan bagaimana skrip ini pendek: seluruh pekerjaan berat dikerjakan oleh package.

Latihan

1. Pakai `datetime`: hitung tanggal panen untuk kopi yang ditanam 15 Januari 2026 dan dipanen 270 hari kemudian, lalu cetak dengan format `dd-mm-yyyy`.
2. Dengan `random.seed(7)`, pilih 3 nomor kolam berbeda dari 1 sampai 12 memakai `random.sample`. Jalankan dua kali dan pastikan hasilnya sama.
3. Tambahkan fungsi `keliling_lingkaran_m(jari_jari_m)` ke modul/`ukuran_lapangan.py`, lalu panggil dari skrip baru yang meng-import-nya.
4. **Tantangan:** tambahkan modul `inventaris/tanggal.py` berisi fungsi `tanggal_tanam(tanggal_panen, umur_hari)`, ekspor lewat `__init__.py`, lalu pakai untuk mencetak tanggal tanam tiap komoditas di `data_inti.tanaman` bila semuanya dipanen pada 1 Oktober 2026.

Ringkasan Bab

- **Modul** = berkas Python berisi alat; **package** = folder modul dengan `__init__.py`.
- Tiga bentuk impor: `import x`, `from x import y`, `import x as z`.
- Pustaka standar berguna: `math`, `random` (pakai `seed`), `datetime`, `statistics`, `collections`, `itertools`, `pathlib`.
- `if __name__ == "__main__":` memisahkan kode yang hanya jalan saat berkas dijalankan langsung.
- `python -m pip install <nama>` memasang pustaka tambahan; `ModuleNotFoundError` berarti modul tidak ditemukan.
- Proyek B3 dimulai: package `inventaris` meringkas data lima sektor.

Selanjutnya, di Bab 12: penanganan error, cara membuat program tidak langsung runtuh saat ada yang salah.

Bab 12 — Penanganan Error: Program yang Tidak Gampang Runtuh

Seri B3 — Program Kecil yang Berguna

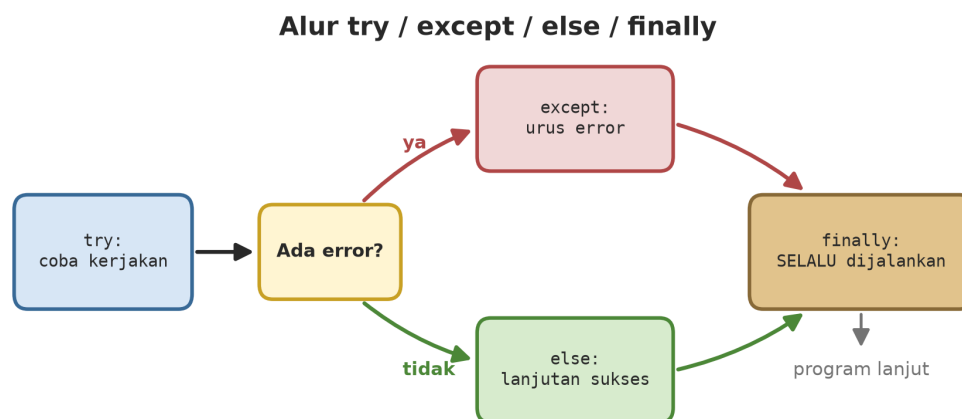
Di bab ini, Anda akan:

- Menangkap error dengan `try`, `except`, `else`, dan `finally`
- Mengenali jenis error yang paling sering muncul
- Memunculkan error sendiri dengan `raise` dan membuat jenis error sendiri
- Membaca **traceback** (jejak error) dari bawah ke atas
- Mencari kutu dengan `print`, `assert`, `logging`, dan mengenal debugger `pdb`

Semua skrip bab ini ada di folder `kode/bab12/`. Di versi web, Anda juga bisa [mengunduh semuanya sekaligus \(zip\)](#).

Bayangkan petugas lapangan mengetik “tiga puluh” padahal komputer menunggu angka 30. Tanpa persiapan, program berhenti mendadak dengan pesan merah yang menakutkan. Di bab-bab sebelumnya, error datang sebagai pelajaran. Sekarang Anda belajar bersiap menghadapinya, seperti petani memasang pagar di tepi kolam: bukan berharap tidak ada yang tersandung, melainkan memastikan tidak ada yang tercebur.

Dalam Python, error disebut **exception** (pengecualian). Saat ia terjadi dan tidak ada yang menangkapnya, program berhenti. Tugas kita: menangkap yang bisa diperbaiki, dan membiarkan yang memang fatal tampil jelas.



Gambar 0.1: Alur try: bagian try dicoba; bila ada error masuk except, bila tidak masuk else; finally selalu dijalankan.

try dan except

```
# Bab 12 - try dan except: menangkap error agar program tidak runtuh
teks_diameter = "tiga puluh" # petugas mengetik kata, bukan angka

try:
    diameter = float(teks_diameter)
    print("Diameter:", diameter)
```

```
except ValueError:
    print("Diameter harus berupa angka, bukan:", teks_diameter)

print("Program tetap berjalan sampai baris terakhir")
```

```
Diameter harus berupa angka, bukan: tiga puluh
Program tetap berjalan sampai baris terakhir
```

Python mencoba isi try. Saat float("tiga puluh") gagal dengan ValueError, loncat ke except ValueError. Sisa isi try dilewati, tetapi program **berlanjut**: baris terakhir tetap tercetak.

⚠️ AWAS

Jangan menulis except: kosong atau menangkap semua dengan except Exception: lalu diam saja. Error nyata akan tersembunyi dan Anda tidak tahu mengapa hasilnya aneh. Tangkap jenis error yang **Anda perkirakan**, dan lakukan sesuatu yang berarti (beri pesan, catat, atau pakai nilai cadangan).

Kenali jenis error

Tiap error punya **jenis** dan **pesan**. Skrip berikut sengaja membuat delapan error umum dan menampilkannya:

```
# Bab 12 - Jenis error yang paling sering muncul
def coba(nama, tindakan):
    try:
        tindakan()
    except Exception as e:
        print(f"{nama:14} -> {type(e).__name__}: {e}")
```

```
daftar = [10, 20, 30]
data = {"jenis": "Jati"}
```

```
coba("ValueError", lambda: int("tiga"))
coba("ZeroDivision", lambda: 10 / 0)
coba("IndexError", lambda: daftar[5])
coba("KeyError", lambda: data["diameter"])
coba("TypeError", lambda: "umur: " + 5)
coba("NameError", lambda: nama_tak_ada)
coba("AttributeError", lambda: daftar.tambah(40))
coba("FileNotFound", lambda: open("tidak_ada.txt"))
```

```
ValueError      -> ValueError: invalid literal for int() with base 10: 'tiga'
ZeroDivision    -> ZeroDivisionError: division by zero
IndexError      -> IndexError: list index out of range
KeyError        -> KeyError: 'diameter'
TypeError       -> TypeError: can only concatenate str (not "int") to str
NameError       -> NameError: name 'nama_tak_ada' is not defined
AttributeError  -> AttributeError: 'list' object has no attribute 'tambah'
FileNotFound    -> FileNotFoundError: [Errno 2] No such file or directory:
'tidak_ada.txt'
```

Jenis	Artinya	Contoh penyebab di lapangan
ValueError	nilai salah untuk tipe itu	mengubah "tiga" jadi angka
ZeroDivisionError	dibagi nol	rata-rata dari daftar kosong

Jenis	Artinya	Contoh penyebab di lapangan
<code>IndexError</code>	nomor urut di luar daftar	mengambil plot ke-6 padahal ada 3
<code>KeyError</code>	kunci dict tidak ada	meminta kolom “diameter” yang tidak ada
<code>TypeError</code>	tipe tidak cocok	menggabung teks dengan angka
<code>NameError</code>	nama variabel belum ada	salah ketik nama variabel
<code>AttributeError</code>	objek tidak punya fitur itu	memanggil <code>.tambah()</code> pada list
<code>FileNotFoundError</code>	berkas tidak ditemukan	salah nama berkas (lihat Bab 13)

Pada skrip itu, `except Exception as e` dipakai **hanya sebagai alat peraga**, supaya semua jenis bisa ditampilkan dengan `type(e).__name__` (nama jenisnya) dan `e` (pesannya).

else dan finally

```
# Bab 12 - try / except / else / finally
def bagi(a, b):
    try:
        hasil = a / b
    except ZeroDivisionError:
        print(" [except] pembagi nol")
        return None
    else:
        print(" [else] tidak ada error")
        return hasil
    finally:
        print(" [finally] selalu dijalankan")

print("bagi(10, 4):")
print(" hasil =", bagi(10, 4))
print("bagi(10, 0):")
print(" hasil =", bagi(10, 0))
```

```
bagi(10, 4):
    [else] tidak ada error
    [finally] selalu dijalankan
    hasil = 2.5
bagi(10, 0):
    [except] pembagi nol
    [finally] selalu dijalankan
    hasil = None
```

- **else** dijalankan **hanya bila tidak ada error** di `try`.
- **finally** dijalankan **selalu**, ada error atau tidak, bahkan saat ada `return`. Cocok untuk “bersih-bersih” seperti menutup berkas.

Perhatikan `bagi(10, 4)`: urutannya `else` lalu `finally`. Pada `bagi(10, 0)`: `except` lalu `finally`.

raise: memunculkan error sendiri

Kadang tidak ada error bawaan, tetapi data jelas tidak masuk akal. pH air hanya mungkin 0 sampai 14. Kita sendiri yang memunculkannya dengan `raise`:

```
# Bab 12 - raise: kita sendiri yang memunculkan error
def cek_ph(ph):
    if not 0 <= ph <= 14:
        raise ValueError(f"pH {ph} di luar rentang 0-14")
    return ph

for nilai in [7.2, 15, -1]:
    try:
        print("pH sah:", cek_ph(nilai))
    except ValueError as e:
        print("Ditolak:", e)
```

```
pH sah: 7.2
Ditolak: pH 15 di luar rentang 0-14
Ditolak: pH -1 di luar rentang 0-14
```

pH 7,2 lolos; 15 dan -1 ditolak dengan pesan yang kita tulis. Pesan error yang jelas menghemat banyak waktu: sebut **apa** yang salah dan **nilainya**.

Jenis error buatan sendiri

Untuk proyek besar, buatlah jenis error sendiri dengan mewarisi `Exception` (**kelas** dan pewarisan dibahas penuh di Bab 16; di sini cukup ikuti polanya):

```
# Bab 12 - Membuat jenis error sendiri
class DataLapanganError(Exception):
    """Induk semua error data lapangan kita."""

class DiameterTidakWajar(DataLapanganError):
    def __init__(self, nilai):
        super().__init__(f"diameter {nilai} cm tidak wajar (harus 1 sampai 300)")
        self.nilai = nilai

def periksa_diameter(cm):
    if not 1 <= cm <= 300:
        raise DiameterTidakWajar(cm)
    return cm

for cm in [32.5, 0.2, 450]:
    try:
        print("OK:", periksa_diameter(cm))
    except DataLapanganError as e:
        # menangkap semua keturunan DataLapanganError
        print(f"{type(e).__name__}: {e} (nilai={e.nilai})")
```

```
OK: 32.5
DiameterTidakWajar: diameter 0.2 cm tidak wajar (harus 1 sampai 300) (nilai=0.2)
DiameterTidakWajar: diameter 450 cm tidak wajar (harus 1 sampai 300) (nilai=450)
```

`DiameterTidakWajar` adalah **turunan** dari `DataLapanganError`, sehingga `except DataLapanganError` menangkap keduanya. Berguna: satu `except` bisa menangani seluruh keluarga error proyek kita, sementara error bawaan lain tetap lewat.

Membaca traceback

Bila error tidak tertangkap, Python mencetak **traceback**: jejak urutan pemanggilan fungsi sampai ke titik error.

```
# Bab 12 - Membaca traceback: error tiga lapis fungsi
def rata_rata(angka):
    return sum(angka) / len(angka)

def ringkas_plot(nama, diameter):
    rata = rata_rata(diameter)
    print(f"Plot {nama}: rata-rata {rata:.1f} cm")

def laporan():
    ringkas_plot("A", [32.5, 45.2])
    ringkas_plot("B", [])          # plot B belum ada datanya

laporan()
```

```
Plot A: rata-rata 38.9 cm
Traceback (most recent call last):
  File "06_baca_traceback.py", line 16, in <module>
    laporan()
    ~~~~~^
  File "06_baca_traceback.py", line 13, in laporan
    ringkas_plot("B", [])          # plot B belum ada datanya
    ~~~~~^~~~~~
  File "06_baca_traceback.py", line 7, in ringkas_plot
    rata = rata_rata(diameter)
  File "06_baca_traceback.py", line 3, in rata_rata
    return sum(angka) / len(angka)
           ~~~~~^~~~~~
ZeroDivisionError: division by zero
```

Bacalah dari bawah ke atas:

1. **Baris terakhir** adalah inti: `ZeroDivisionError: division by zero`, yaitu jenis dan pesan error.
2. Baris `File ... in rata_rata` tepat di atasnya menunjukkan **tempat kejadian**: pembagian `sum(angka) / len(angka)` dengan `len(angka)` bernilai 0.
3. Ke atas lagi, `ringkas_plot` yang memanggilnya, lalu `laporan`, lalu `<module>` (kode tingkat paling luar). Itulah rantai pemanggilan.
4. Tanda `~~~~^` di bawah baris kode menunjuk bagian ekspresi yang bermasalah.
5. Sebab sebenarnya ada di data: plot B daftarnya kosong. Plot A tercetak sebelum error, jadi jejaknya juga memberi tahu **sampai mana** program sempat jalan.

💡 TIPS

Nomor baris pada traceback dihitung dari baris pertama berkas, **termasuk baris komentar di paling atas**. Karena itu nomor yang tampil di atas cocok persis dengan baris pada kode di buku dan pada berkas unduhan: hitung baris komentar pembuka sebagai baris 1.

Mencari kutu: print, assert, logging

Kesalahan program disebut **bug** (kutu). Mencarinya disebut **debugging**. Tiga alat dasar:

1. **print debugging**: cetak isi variabel di tiap langkah. Pakai `repr()` agar tanda kutip dan spasi tersembunyi tampak.

```
# Bab 12 - Debugging dengan print: lihat isi variabel di tiap langkah
def bobot_total_kg(kolam):
    total = 0
    for k in kolam:
```

```

        bobot_kg = k["jumlah"] * k["bobot_g"] / 1000
        print("    [debug]", repr(k["nama"]), "->", bobot_kg)      # repr menampilkan
tanda kutip
        total += bobot_kg
    return total

kolam = [
    {"nama": "A", "jumlah": 800, "bobot_g": 540.5},
    {"nama": "B", "jumlah": 1250, "bobot_g": 300.0},
]
print("Total:", bobot_total_kg(kolam), "kg")

```

```

[debug] 'A' -> 432.4
[debug] 'B' -> 375.0
Total: 807.4 kg

```

Kolam A 432,4 kg dan kolam B 375,0 kg, jumlahnya 807,4 kg. Dengan baris [debug] Anda melihat tiap suku penjumlahannya. Hapus baris itu setelah bug ketemu.

2. **assert**: pagar keamanan yang menyatakan “ini pasti benar”. Bila salah, program berhenti dengan `AssertionError`.

```

# Bab 12 - assert: pagar keamanan "ini PASTI benar" (versi yang lolos)
def persen_hidup(awal, hidup):
    assert awal > 0, "jumlah awal harus lebih dari 0"
    assert 0 <= hidup <= awal, "ikan hidup tidak boleh lebih banyak dari awal"
    return hidup / awal * 100

print(f"{persen_hidup(1000, 940):.1f}%")

```

```

94.0%

```

```

# Bab 12 - assert yang gagal: asumsi kita ternyata keliru
def persen_hidup(awal, hidup):
    assert awal > 0, "jumlah awal harus lebih dari 0"
    assert 0 <= hidup <= awal, "ikan hidup tidak boleh lebih banyak dari awal"
    return hidup / awal * 100

print(f"{persen_hidup(1000, 1040):.1f}%")

```

```

Traceback (most recent call last):
  File "09_assert_gagal.py", line 8, in <module>
    print(f"{persen_hidup(1000, 1040):.1f}%")
    ~~~~~^~~~~~
  File "09_assert_gagal.py", line 4, in persen_hidup
    assert 0 <= hidup <= awal, "ikan hidup tidak boleh lebih banyak dari awal"
    ~~~~~^~~~~~
AssertionError: ikan hidup tidak boleh lebih banyak dari awal

```

Ikan hidup (1040) lebih banyak daripada ikan awal (1000), tidak masuk akal, maka `assert` kedua menghentikan program dengan pesan yang kita tulis. Gunakan `assert` untuk menangkap **kesalahan programmer**, bukan kesalahan input pengguna.

⚠️ AWAS

Python dapat dijalankan dengan opsi `-0` yang **membuang semua `assert`**. Kami mencobanya pada skrip di atas: dengan `python -0`, hasilnya tercetak 104.0% tanpa error. Jadi jangan memakai `assert` sebagai satu-satunya pemeriksa input dari pengguna; pakai `if` dan `raise`.

3. **logging**: print yang lebih rapi, dengan **tingkat** kepentingan (DEBUG, INFO, WARNING, ERROR, CRITICAL) yang bisa disaring.

```
# Bab 12 - logging: catatan kerja program dengan tingkat kepentingan
import logging
import sys

logging.basicConfig(stream=sys.stdout, level=logging.DEBUG,
                    format="%(levelname)-8s %(message)s")

logging.debug("memeriksa 3 baris data")
logging.info("data kolam A dibaca")
logging.warning("pH kolam B = 9.8, terlalu tinggi")
logging.error("data kolam C tidak ditemukan")

logging.getLogger().setLevel(logging.WARNING)      # sekarang hanya WARNING ke atas
logging.info("pesan ini tidak muncul")
logging.warning("pesan ini muncul")
```

```
DEBUG    memeriksa 3 baris data
INFO     data kolam A dibaca
WARNING  pH kolam B = 9.8, terlalu tinggi
ERROR    data kolam C tidak ditemukan
WARNING  pesan ini muncul
```

Setelah tingkat dinaikkan ke WARNING, pesan `info` tidak muncul lagi, tetapi `warning` tetap. Dengan `logging`, Anda tidak perlu menghapus pesan debug saat program selesai; cukup naikan tingkatnya. (Pembahasan logging lebih jauh ada di Bab 19.)

Alat keempat adalah **debugger** `pdb` bawaan Python: fungsi `breakpoint()` menghentikan program di satu baris dan membuka prompt (`Pdb`) tempat Anda bisa mengetik `p variabel` (lihat isi), `n` (baris berikut), `c` (lanjut), dan `q` (keluar).

```
# Bab 12 - Debugger interaktif pdb: berhenti di satu baris dan mengintip isi variabel
def rata_rata(angka):
    total = sum(angka)
    breakpoint()          # program berhenti di sini dan membuka prompt (Pdb)
    return total / len(angka)

print(rata_rata([32.5, 45.2, 28.0]))
```

Kode ini **tidak dijalankan** saat buku dibuat karena `pdb` menunggu ketikan pengguna, jadi keluarannya tidak ditampilkan. Coba sendiri di terminal: jalankan berkasnya, lalu ketik `p total` dan `c`.

Proyek Bersambung B3: tahan terhadap input salah

Dua skrip proyek. Pertama, **meminta input sampai benar**. Masukan tiruan (disediakan berkas `.input`) berisi lima jawaban: `abc`, `-5`, baris kosong, `450`, dan akhirnya `32,5`.

```
# Bab 12 - Proyek B3: meminta angka sampai benar (tahan terhadap input salah)
def minta_diameter():
    while True:
        teks = input("Diameter pohon (cm): ")
        try:
```

```

        nilai = float(teks.replace(",", "."))      # terima "32,5" dan "32.5"
    except ValueError:
        print(" Bukan angka, coba lagi.")
        continue
    if not 1 <= nilai <= 300:
        print(" Harus antara 1 dan 300 cm, coba lagi.")
        continue
    return nilai

hasil = minta_diameter()
print("Diameter tercatat:", hasil)

```

```

Diameter pohon (cm): abc
    Bukan angka, coba lagi.
Diameter pohon (cm): -5
    Harus antara 1 dan 300 cm, coba lagi.
Diameter pohon (cm):
    Bukan angka, coba lagi.
Diameter pohon (cm): 450
    Harus antara 1 dan 300 cm, coba lagi.
Diameter pohon (cm): 32,5
Diameter tercatat: 32.5

```

Empat jawaban pertama ditolak dengan alasan berbeda (bukan angka, di luar rentang), dan jawaban 32,5 diterima setelah koma diganti titik. Pola `while True + try/except + continue + return` adalah resep standar “ulangi sampai benar”.

Kedua, **membersihkan baris data campur aduk** yang kelak datang dari berkas (Bab 13 dan 14). Ada enam baris; tiga di antaranya rusak:

```

# Bab 12 - Proyek B3: membersihkan baris data yang campur aduk tanpa runtuh
import logging
import sys

logging.basicConfig(stream=sys.stdout, level=logging.WARNING, format="%(levelname)s:
%(message)s")

class BarisSalah(Exception):
    pass

def baca_baris(teks):
    """Mengubah 'plot,jenis,diameter' menjadi dict, atau munculkan BarisSalah."""
    bagian = [b.strip() for b in teks.split(",")]
    if len(bagian) != 3:
        raise BarisSalah(f"harus 3 kolom, ada {len(bagian)}: {teks!r}")
    plot, jenis, diameter = bagian
    try:
        diameter = float(diameter.replace(",", "."))
    except ValueError:
        raise BarisSalah(f"diameter bukan angka: {diameter!r}") from None
    if not 1 <= diameter <= 300:
        raise BarisSalah(f"diameter di luar 1-300: {diameter}")
    return {"plot": plot, "jenis": jenis, "diameter_cm": diameter}

mentah = ["A,Jati,32.5", "A,Meranti,45.2", "B,Mahoni,dua puluh", "B,Sengon",
"C,Jati,-3", "C,Jati,36.0"]
bersih, ditolak = [], 0

```

```

for baris in mentah:
    try:
        bersih.append(baca_baris(baris))
    except BarisSalah as e:
        logging.warning("baris ditolak: %s", e)
        ditolak += 1

print(f"Diterima {len(bersih)} baris, ditolak {ditolak}")
for b in bersih:
    print(b)

```

```

WARNING: baris ditolak: diameter bukan angka: 'dua puluh'
WARNING: baris ditolak: harus 3 kolom, ada 2: 'B,Sengon'
WARNING: baris ditolak: diameter di luar 1-300: -3.0
Diterima 3 baris, ditolak 3
{'plot': 'A', 'jenis': 'Jati', 'diameter_cm': 32.5}
{'plot': 'A', 'jenis': 'Meranti', 'diameter_cm': 45.2}
{'plot': 'C', 'jenis': 'Jati', 'diameter_cm': 36.0}

```

Alurnya: `baca_baris` memunculkan `BarisSalah` untuk tiap jenis kerusakan; perulangan menangkapnya, **mencatat peringatan** dengan `logging`, dan lanjut ke baris berikutnya. Hasil akhir: 3 baris diterima, 3 ditolak, dan program tidak pernah runtuh. Bagian `from None` menyembunyikan error asli (`ValueError`) supaya pesan kita yang tampil; tanpa itu, traceback akan memuat dua error yang membingungkan pembaca.

Latihan

1. Tulis fungsi `ke_angka(teks)` yang mengembalikan `float` dari teks, atau `None` bila teksnya bukan angka. Uji dengan "12,5", "abc", dan "".
2. Ubah `13_proyek_tahan_salah.py` supaya diameter di atas 150 cm tidak ditolak, tetapi hanya menghasilkan `logging.warning("pohon sangat besar")`.
3. Buat `assert` pada fungsi `rata_rata(angka)` yang memastikan daftar tidak kosong. Bandingkan pesan errornya dengan `ZeroDivisionError` di traceback tadi.
4. **Tantangan:** buat kelas error `PhTidakWajar(DataLapanganError)` dan fungsi `periksa_kolam(daftar)` yang mengumpulkan **semua** kolom bermasalah (bukan berhenti di yang pertama), lalu mencetak ringkasan.

Ringkasan Bab

- **try / except** menangkap error; **else** jalan bila tidak ada error; **finally** selalu jalan.
- Tangkap jenis error yang spesifik; hindari `except:` kosong.
- **raise** memunculkan error sendiri; jenis error baru dibuat dengan mewarisi `Exception`.
- **Traceback** dibaca dari bawah: jenis dan pesan, tempat kejadian, lalu rantai pemanggilan.
- Alat debugging: `print` (dengan `repr`), `assert` (bisa dimatikan dengan `-O`), `logging`, dan `pdb/breakpoint()`.
- Proyek B3: program meminta input sampai benar dan menolak baris data rusak tanpa runtuh.

Selanjutnya, di Bab 13: file dan I/O dasar, cara program menyimpan dan membaca data agar tidak hilang saat ditutup.

Bab 13 — File dan I/O Dasar: Data yang Tidak Hilang Saat Program Ditutup

Seri B3 — Program Kecil yang Berguna

Di bab ini, Anda akan:

- Menulis dan membaca berkas teks dengan `open()` dan `with`
- Memilih mode yang tepat: `"r"`, `"w"`, `"a"`, `"x"`
- Mengelola berkas dan folder dengan `pathlib`
- Memahami **encoding** (cara teks disimpan sebagai byte) dan mengapa harus UTF-8
- Menghadapi berkas yang tidak ada, dan menyimpan data pohon ke berkas lalu membacanya kembali

Semua skrip bab ini ada di folder `kode/bab13/`. Di versi web, Anda juga bisa [mengunduh semuanya sekaligus \(zip\)](#). Skrip yang membuat berkas menuliskannya ke subfolder `_tmp/` di samping skrip, jadi folder Anda tidak berantakan.

Sejauh ini, semua data hidup di **memori** komputer, seperti catatan di papan tulis: begitu program ditutup, tulisannya terhapus. Petugas lapangan yang baik mencatat di **buku**, yaitu **berkas** (*file*) di disk, yang tetap ada walau komputer dimatikan. Bab ini mengajarkan program Anda menulis dan membaca buku itu. Istilah umumnya **I/O** (*input/output*): data masuk dari berkas, data keluar ke berkas.

Program dan berkas di disk



`"r"` baca (bawaan)

`"a"` tambah di akhir

`"w"` tulis baru, isi lama hilang

`"x"` buat baru, gagal bila sudah ada

`with open(...) as f: -> otomatis ditutup`

Gambar 0.1: Program menulis ke berkas di disk (write) dan membacanya kembali kemudian (read). Mode: `r` = baca, `w` = tulis baru, `a` = tambah, `x` = buat baru saja.

Menulis berkas teks

```
# Bab 13 - Menulis berkas teks dengan open() dan with
from pathlib import Path

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

with open(berkas, "w", encoding="utf-8") as f:      # "w" = tulis (isi lama dihapus)
```

```
f.write("Catatan lapangan 1 Oktober\n")
f.write("Plot A: 12 pohon jati\n")
f.write("Plot B: 9 pohon meranti\n")

print("Berkas ditulis:", berkas.name)
print("Ukuran:", berkas.stat().st_size, "byte")
```

```
Berkas ditulis: catatan.txt
Ukuran: 76 byte
```

- `open(berkas, "w", encoding="utf-8")` membuka berkas untuk **menulis** ("w"). Bila berkas sudah ada, isi lamanya **dihapus**.
- `with ... as f:` membuka berkas dan **menutupnya otomatis** saat blok selesai. Anda akan memahami pentingnya di bawah.
- `f.write(...)` menulis teks. Ia **tidak** menambah baris baru sendiri, jadi Anda menulis `\n` di ujung tiap baris.

Mengapa ukurannya 76 byte, padahal hitungan huruf hanya 73? Pada Windows, setiap `\n` ditulis sebagai dua byte (`\r\n`), dan ada tiga baris, jadi $73 + 3 = 76$. Di macOS atau Linux hasilnya 73. Jangan heran bila angka ukuran berkas berbeda antarkomputer.

Membaca berkas teks

```
# Bab 13 - Empat cara membaca berkas teks
from pathlib import Path

tmp = Path(__file__).parent / "_tmp"
tmp.mkdir(exist_ok=True)
berkas = tmp / "catatan.txt"
berkas.write_text("Catatan lapangan 1 Oktober\nPlot A: 12 pohon jati\nPlot B: 9 pohon
meranti\n",
                  encoding="utf-8")

with open(berkas, "r", encoding="utf-8") as f:
    semua = f.read()
    print(repr(semua))
    # 1. seluruh isi sebagai satu teks

with open(berkas, encoding="utf-8") as f:
    print(repr(f.readline()))
    # 2. satu baris (dengan \n di ujung)

with open(berkas, encoding="utf-8") as f:
    daftar = f.readlines()
    print(len(daftar), "baris")
    # 3. semua baris sebagai list

with open(berkas, encoding="utf-8") as f:
    for nomor, baris in enumerate(f, start=1):
        print(nomor, baris.rstrip())
    # 4. baris demi baris (hemat memori)
```

```
'Catatan lapangan 1 Oktober\nPlot A: 12 pohon jati\nPlot B: 9 pohon meranti\n'
'Catatan lapangan 1 Oktober\n'
3 baris
1 Catatan lapangan 1 Oktober
2 Plot A: 12 pohon jati
3 Plot B: 9 pohon meranti
```

Empat cara membaca, dari yang paling kasar ke yang paling hemat:

1. `read()`: seluruh isi sebagai **satu teks**. Perhatikan `\n` di sela baris (`repr` menampakkannya).
2. `readline()`: **satu baris**, lengkap dengan `\n` di ujung. Dipanggil lagi, ia memberi baris berikutnya.
3. `readlines()`: **semua baris sebagai list**.

4. **Perulangan for langsung atas berkas:** satu baris per putaran, tanpa memuat semuanya ke memori. Ini pilihan terbaik untuk berkas besar. `rstrip()` membuang `\n` di ujungnya, dan `enumerate(f, start=1)` memberi nomor baris.

Mode: w, a, x

```
# Bab 13 - Mode "w" menimpa, mode "a" menambah, mode "x" menolak menimpa
from pathlib import Path

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

with open(berkas, "w", encoding="utf-8") as f:
    f.write("Panen 1: 245.5 kg\n")
with open(berkas, "a", encoding="utf-8") as f:      # tambah di akhir
    f.write("Panen 2: 310.0 kg\n")
print(berkas.read_text(encoding="utf-8"))

with open(berkas, "w", encoding="utf-8") as f:      # menimpa: isi lama hilang
    f.write("Panen 3: 198.5 kg\n")
print(berkas.read_text(encoding="utf-8"))

try:
    with open(berkas, "x", encoding="utf-8") as f:  # "x": gagal bila sudah ada
        f.write("tidak jadi")
except FileExistsError:
    print("Mode x menolak: berkas sudah ada")
```

```
Panen 1: 245.5 kg
Panen 2: 310.0 kg

Panen 3: 198.5 kg

Mode x menolak: berkas sudah ada
```

Mode	Artinya
"r"	baca (bawaan); error bila berkas tidak ada
"w"	tulis; menimpa isi lama
"a"	tambah di akhir; isi lama aman
"x"	buat baru; gagal bila berkas sudah ada

Lihat hasilnya: setelah mode "a", ada dua panen. Setelah mode "w", panen 1 dan 2 **hilang**, tinggal panen 3. Mode "x" melindungi berkas penting dari tertimpa tanpa sengaja (ia memunculkan `FileExistsError`, yang kita tangkap dengan `try` dari Bab 12).

⚠️ AWAS

Mode "w" menghapus isi lama **tanpa bertanya**. Bila Anda ingin menambahkan catatan harian, pakai "a". Salah memilih satu huruf bisa menghilangkan data sehari-hari.

Mengapa with

```
# Bab 13 - Mengapa with: berkas ditutup otomatis, bahkan bila terjadi error
from pathlib import Path
```

```

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

f = open(berkas, "w", encoding="utf-8")      # cara lama: harus menutup sendiri
f.write("halo\n")
print("Sebelum close, tertutup?", f.closed)
f.close()
print("Sesudah close, tertutup?", f.closed)

try:
    with open(berkas, "w", encoding="utf-8") as g:
        g.write("halo lagi\n")
        raise RuntimeError("ada masalah di tengah menulis")
except RuntimeError as e:
    print("Error tertangkap:", e)
print("Berkas tetap tertutup rapi?", g.closed)

```

```

Sebelum close, tertutup? False
Sesudah close, tertutup? True
Error tertangkap: ada masalah di tengah menulis
Berkas tetap tertutup rapi? True

```

Berkas yang sudah dibuka harus **ditutup** agar isinya benar-benar tersimpan dan tidak mengunci berkas bagi program lain. Cara lama: `open()` lalu `f.close()` sendiri; bila di tengah terjadi error, `close()` bisa terlewat. Dengan `with`, penutupan dijamin: pada contoh, error `RuntimeError` meledak di tengah menulis, tetapi `g.closed` tetap `True`.

i INGAT

Biasakan **selalu** `with open(...)`. Anda hampir tidak pernah perlu menulis `close()` sendiri.

pathlib untuk berkas dan folder

Di Bab 11 Anda sudah berkenalan dengan `Path`. Sekarang ia bekerja pada disk sungguhan:

```

# Bab 13 - pathlib untuk berkas dan folder sungguhan
from pathlib import Path

tmp = Path(__file__).parent / "_tmp" / "lapangan"
tmp.mkdir(parents=True, exist_ok=True)      # buat folder bertingkat; tidak error bila
sudah ada

for nama in ["plot_a.txt", "plot_b.txt", "ikan.csv"]:
    (tmp / nama).write_text(f"isi {nama}\n", encoding="utf-8")

print("Ada folder?", tmp.is_dir())
print("Semua berkas :", sorted(p.name for p in tmp.iterdir()))
print("Hanya .txt    :", sorted(p.name for p in tmp.glob("*.txt")))
print("Ukuran plot_a:", (tmp / "plot_a.txt").stat().st_size, "byte")

lama = tmp / "ikan.csv"
baru = lama.rename(tmp / "ikan_2026.csv")
print("Diganti nama :", baru.name)
baru.unlink()                                # hapus satu berkas (di folder _tmp
milik kita)
print("Tersisa      :", sorted(p.name for p in tmp.iterdir()))

```

```

Ada folder? True
Semua berkas : ['ikan.csv', 'plot_a.txt', 'plot_b.txt']
Hanya .txt   : ['plot_a.txt', 'plot_b.txt']
Ukuran plot_a: 16 byte
Diganti nama : ikan_2026.csv
Tersisa      : ['plot_a.txt', 'plot_b.txt']

```

- `mkdir(parents=True, exist_ok=True)` membuat folder bertingkat dan tidak protes bila sudah ada.
- `write_text()` dan `read_text()` adalah jalan pintas untuk menulis atau membaca seluruh isi berkas kecil, tanpa `with`.
- `iterdir()` mendaftar isi folder; `glob("*.txt")` memilih berkas berpola. Kita menyusunnya dengan `sorted` karena urutan dari disk tidak dijamin.
- `rename()` mengganti nama, `unlink()` menghapus satu berkas, `stat().st_size` memberi ukuran dalam byte.

Pakai `unlink()` dengan hati-hati: berkas yang dihapus dengan cara ini **tidak masuk keranjang sampah**. Dalam contoh, kita hanya menghapus berkas buatan kita sendiri di folder `_tmp`.

Encoding: teks yang berubah jadi simbol aneh

Disk hanya menyimpan **byte** (angka 0 sampai 255), bukan huruf. **Encoding** adalah aturan yang menerjemahkan huruf menjadi byte dan sebaliknya. Aturan paling dianjurkan adalah **UTF-8**, yang mampu menulis huruf apa pun, termasuk °, ±, dan aksara non-Latin.

```

# Bab 13 - Teks disimpan sebagai byte; pakai UTF-8 dan tulis secara eksplisit
from pathlib import Path

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

teks = "Suhu air 28 °C, Pontianak, hasil ± 5%"
berkas.write_text(teks, encoding="utf-8")

mentah = berkas.read_bytes()          # isi berkas apa adanya: deretan byte
print("Jumlah karakter:", len(teks))
print("Jumlah byte      :", len(mentah))
print("Byte lambang derajat:", "°".encode("utf-8"))
print("Dibaca balik    :", berkas.read_text(encoding="utf-8"))

```

```

Jumlah karakter: 37
Jumlah byte      : 39
Byte lambang derajat: b'\xc2\xb0'
Dibaca balik    : Suhu air 28 °C, Pontianak, hasil ± 5%

```

Teks itu 37 karakter tetapi 39 byte: lambang ° dan ± masing-masing memakai **dua byte** dalam UTF-8 (`b'\xc2\xb0'` untuk °). Itu sebabnya “jumlah karakter” dan “jumlah byte” bisa berbeda.

Sekarang yang terjadi bila berkas yang ditulis dengan satu aturan dibaca dengan aturan lain:

```

# Bab 13 - Encoding SALAH: ditulis dengan satu aturan, dibaca dengan aturan lain
from pathlib import Path

tmp = Path(__file__).parent / "_tmp"
tmp.mkdir(exist_ok=True)
berkas = tmp / "suhu_utf8.txt"
berkas.write_text("Suhu air 28 °C, hasil ± 5%", encoding="utf-8")

# cp1252 = aturan bawaan banyak komputer Windows berbahasa Barat
print("Dibaca sebagai cp1252:", berkas.read_text(encoding="cp1252"))

```

```
try:
    berkas.read_text(encoding="ascii")          # ascii hanya mengenal 128 lambang
except UnicodeDecodeError as e:
    print("Dibaca sebagai ascii -> UnicodeDecodeError:", e.reason)

print("Dibaca sebagai utf-8 :", berkas.read_text(encoding="utf-8"))
```

```
Dibaca sebagai cp1252: Suhu air 28 Â°C, hasil Â± 5%
Dibaca sebagai ascii -> UnicodeDecodeError: ordinal not in range(128)
Dibaca sebagai utf-8 : Suhu air 28 °C, hasil ± 5%
```

Hasil Â°C dan Â± itulah **mojibake**: dua byte UTF-8 dibaca sebagai dua huruf terpisah oleh aturan cp1252. Pada aturan ascii, Python malah menolak dengan UnicodeDecodeError. Hanya pembacaan UTF-8 yang mengembalikan teks asli.

⚠️ AWAS

Bila Anda **tidak menyebut** `encoding=`, Python memakai aturan bawaan sistem. Di komputer penulis buku ini (Windows), bawaannya cp1252; kami mengeceknya langsung dengan `locale.getencoding()`. Akibatnya berkas yang ditulis di satu komputer bisa rusak saat dibaca di komputer lain. Mulai Python 3.15 (dirilis 9 Oktober 2026), UTF-8 menjadi bawaan: kami membuktikannya dengan membuka berkas tanpa `encoding=` di Python 3.15.0, hasilnya utf-8, dan mematikannya dengan `PYTHONUTF8=0` mengembalikan cp1252 (lihat Bab 47). Buku ini berpatokan ke Python 3.14, yang masih memakai aturan sistem, jadi **tulislah `encoding="utf-8"` selalu**, supaya kode Anda benar di semua versi dan komputer.

Berkas yang tidak ada

```
# Bab 13 - Jebakan: membuka berkas yang tidak ada
with open("data_pohon_tidak_ada.txt", encoding="utf-8") as f:
    print(f.read())
```

```
Traceback (most recent call last):
  File "08_berkas_tak_ada.py", line 2, in <module>
    with open("data_pohon_tidak_ada.txt", encoding="utf-8") as f:
        ~~~~^~~~~~
FileNotFoundError: [Errno 2] No such file or directory: 'data_pohon_tidak_ada.txt'
```

FileNotFoundError adalah error paling umum dalam urusan berkas: salah ketik nama, salah folder, atau memang belum dibuat. Dua cara menghadapinya dengan sopan:

```
# Bab 13 - Menghadapi berkas yang tidak ada dengan sopan
from pathlib import Path

def baca_atau_cadangan(berkas, cadangan="(belum ada data)"):
    try:
        return Path(berkas).read_text(encoding="utf-8")
    except FileNotFoundError:
        print(f" [info] {Path(berkas).name} tidak ditemukan, memakai cadangan")
        return cadangan

print(baca_atau_cadangan("data_pohon_tidak_ada.txt"))

# Cara kedua: periksa dulu dengan exists()
alamat = Path("data_pohon_tidak_ada.txt")
if alamat.exists():
```

```

print(alamat.read_text(encoding="utf-8"))
else:
    print("Cek exists(): berkas belum ada, lewati")

```

```

[info] data_pohon_tidak_ada.txt tidak ditemukan, memakai cadangan
(belum ada data)
Cek exists(): berkas belum ada, lewati

```

Cara pertama memakai `try/except` (disarankan, karena berkas bisa saja lenyap di antara “dicek” dan “dibuka”). Cara kedua memakai `exists()`: lebih mudah dibaca, tetapi hanya memeriksa keadaan saat itu.

Proyek Bersambung B3: menyimpan dan memuat data pohon

Sekarang kita gabungkan Bab 12 dan 13. Skrip menyimpan enam pohon ke `pohon.txt` (satu baris satu pohon, kolom dipisah `|`) dan **sengaja menyisipkan satu baris rusak**. Lalu fungsi `muat_pohon` membacanya kembali dengan tahan salah:

```

# Bab 13 - Proyek B3: data pohon disimpan ke berkas teks dan dibaca kembali
from pathlib import Path

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

pohon = [
    ("A", "Jati", 32.5), ("A", "Meranti", 45.2), ("B", "Mahoni", 28.0),
    ("B", "Sengon", 51.5), ("C", "Jati", 36.0), ("C", "Meranti", 40.8),
]

# 1. Simpan: satu pohon satu baris, kolom dipisah tanda |
with open(berkas, "w", encoding="utf-8") as f:
    for plot, jenis, diameter in pohon:
        f.write(f"{plot}|{jenis}|{diameter}\n")
    f.write("C|Jati|rusak\n")          # satu baris rusak, sengaja disisipkan

# 2. Baca kembali dengan tahan salah (pelajaran Bab 12)
def muat_pohon(alamat):
    hasil, ditolak = [], 0
    try:
        with open(alamat, encoding="utf-8") as f:
            for baris in f:
                try:
                    plot, jenis, diameter = baris.strip().split("|")
                    hasil.append((plot, jenis, float(diameter)))
                except ValueError:
                    ditolak += 1
    except FileNotFoundError:
        print("Berkas belum ada, mulai dengan data kosong")
    return hasil, ditolak

data, ditolak = muat_pohon(berkas)
print(f"Terbaca {len(data)} pohon, {ditolak} baris ditolak")
print("Rata-rata diameter:", round(sum(d for _, _, d in data) / len(data), 2), "cm")
print("Data pertama:", data[0])
print("Berkas yang tidak ada:", muat_pohon(tmp / "tidak_ada.txt"))

```

```
Terbaca 6 pohon, 1 baris ditolak
Rata-rata diameter: 39.0 cm
Data pertama: ('A', 'Jati', 32.5)
Berkas belum ada, mulai dengan data kosong
Berkas yang tidak ada: ([], 0)
```

Enam pohon terbaca, baris C|Jati|rusak ditolak (diameternya bukan angka), rata-rata diameter 39,0 cm (cocok dengan hasil Bab 11). Berkas yang tidak ada menghasilkan data kosong dengan pesan ramah, bukan error. Tetapi format | buatan sendiri ini rapuh: bagaimana bila nama jenis mengandung |? Di Bab 14 kita pakai format standar (CSV dan JSON) yang sudah memecahkan masalah itu.

Latihan

1. Tulis skrip yang menulis angka 1 sampai 10 (satu per baris) ke `angka.txt`, lalu membacanya kembali dan mencetak jumlahnya.
2. Jalankan skrip pada latihan 1 dua kali dengan mode `"a"` (bukan `"w"`). Berapa baris isi berkasnya, dan mengapa?
3. Ubah `07_encoding_salah.py`: tulis dengan `encoding="utf-16"`, lalu baca dengan `"utf-8"`. Apa yang terjadi? Tangkap error-nya dengan `try`.
4. **Tantangan:** tulis fungsi `catat_panen(berkas, tanggal, kg)` yang menambah satu baris `tanggal|kg` ke berkas (mode `"a"`), dan fungsi `total_panen(berkas)` yang menjumlahkan kolom kg sambil melewati baris rusak.

Ringkasan Bab

- **Berkas** menyimpan data di disk; program membacanya (`"r"`) dan menulisnya (`"w"`, `"a"`, `"x"`).
- **Selalu** `with open(..., encoding="utf-8") as f::` berkas tertutup otomatis.
- Membaca: `read()`, `readline()`, `readlines()`, dan perulangan `for` atas berkas (hemat memori).
- `pathlib.Path`: `mkdir`, `read_text`, `write_text`, `iterdir`, `glob`, `rename`, `unlink`, `exists`.
- **Encoding** salah menghasilkan simbol aneh atau `UnicodeDecodeError`; tulis `encoding="utf-8"` selalu.
- Berkas yang tidak ada: tangkap `FileNotFoundError`.

Selanjutnya, di Bab 14: format data standar (CSV dan JSON) agar data kita rapi dan bisa dibaca program lain.

Bab 14 — Format Data: CSV, JSON, dan Hati-hati dengan Pickle

Seri B3 — Program Kecil yang Berguna

Di bab ini, Anda akan:

- Menyimpan dan membaca tabel dengan modul `csv`
- Menyimpan dan membaca data bersarang dengan modul `json`
- Mengerti **serialisasi**, dan tahu mengapa nilai dari CSV selalu berupa teks
- Memahami bahaya `pickle` dan membuktikannya secara aman
- Menyimpan data kelima sektor ke CSV dan JSON, lalu memuatnya kembali dengan tahan salah

Semua skrip bab ini ada di folder `kode/bab14/`. Di versi web, Anda juga bisa [mengunduh semuanya sekaligus \(zip\)](#). Berkas hasil ditulis ke subfolder `_tmp/`.

Di Bab 13, kita menyimpan data pohon dengan format buatan sendiri (`A|Jati|32.5`) dan sudah curiga formatnya rapuh. Bayangkan dua petugas yang saling mengirim catatan: bila masing-masing punya cara menulis sendiri, tidak ada yang bisa membaca catatan lainnya. Maka dunia sepakat pada beberapa **format standar**. Mengubah data di memori menjadi teks (atau byte) yang bisa disimpan disebut **serialisasi**; mengembalikannya disebut **deserialisasi**. Bab ini membahas tiga format: CSV, JSON, dan pickle.

Data yang sama dalam dua format

CSV: tabel datar	JSON: data bersarang
<pre>kolam,jenis,jumlah,ph_air A,lele,800,7.2 B,nila,1250,6.9</pre>	<pre>{"kolam": "A", "jenis": "lele", "jumlah": 800, "ph_air": 7.2, "sampel_g": [520, 480]}</pre>
semua nilai terbaca sebagai teks	angka, true/false, null, list kembali bertipe

Gambar 0.1: CSV menyimpan data sebagai tabel datar; JSON menyimpan data bersarang dengan tanda kurung. Keduanya teks biasa.

CSV: tabel sebagai teks

CSV (*comma-separated values*) adalah tabel yang tiap barisnya satu catatan dan tiap kolomnya dipisah koma. Berkasnya bisa dibuka Excel. Kita menulis data kolam dari Bab 11:

```
# Bab 14 - Menulis CSV: tabel sebagai teks berkolom koma
import csv
from pathlib import Path

from data_inti import ikan

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

```

berkas = tmp / "ikan.csv"

kolom = ["kolam", "jenis", "jumlah", "bobot_rata_g", "ph_air"]
with open(berkas, "w", newline="", encoding="utf-8") as f:      # newline="" wajib
    untuk csv
        penulis = csv.DictWriter(f, fieldnames=kolom)
        penulis.writeheader()                                  # baris judul kolom
        penulis.writerows(ikan)                                # semua baris sekaligus

print(berkas.read_text(encoding="utf-8"))

```

```

kolam,jenis,jumlah,bobot_rata_g,ph_air
A,lele,800,540.5,7.2
B,nila,1250,300.0,6.9
C,lele,600,480.0,7.0

```

- DictWriter menulis daftar dict; fieldnames menentukan **urutan kolom**.
- writeheader() menulis baris judul, writerows() menulis semua baris.
- newline="" pada open() **wajib** untuk modul csv; dokumentasi Python menyebutkan berkas CSV harus dibuka dengan newline='' agar akhir baris ditangani dengan benar.

Data ini data tiruan (angka contoh).

Membaca CSV: semuanya teks

```

# Bab 14 - Membaca CSV: semua nilai datang sebagai TEKS
import csv
from pathlib import Path

tmp = Path(__file__).parent / "_tmp"
tmp.mkdir(exist_ok=True)
berkas = tmp / "ikan_baca.csv"
berkas.write_text("kolam,jenis,jumlah,bobot_rata_g\nA,lele,800,540.5\nB,nila,1250,300.0\n",
                  encoding="utf-8")

with open(berkas, newline="", encoding="utf-8") as f:
    baris = list(csv.DictReader(f))          # tiap baris menjadi dict berkunci
    judul kolom

print(baris[0])
print("Tipe jumlah:", type(baris[0]["jumlah"]).__name__)
print("Salah  :", baris[0]["jumlah"] + baris[1]["jumlah"])    # teks digabung,
                                                                bukan dijumlah

for b in baris:
    b["jumlah"] = int(b["jumlah"])            # ubah tipe sendiri
    b["bobot_rata_g"] = float(b["bobot_rata_g"])
print("Benar  :", baris[0]["jumlah"] + baris[1]["jumlah"])

{'kolam': 'A', 'jenis': 'lele', 'jumlah': '800', 'bobot_rata_g': '540.5'}
Tipe jumlah: str
Salah  : 8001250
Benar  : 2050

```

DictReader membuat dict per baris dengan kunci dari baris judul. Perhatikan: 'jumlah': '800' diapit tanda kutip, artinya **teks**, bukan angka. Dokumentasi csv menegaskan tidak ada konversi tipe otomatis. Akibatnya "800" + "1250" menghasilkan 8001250 (teks digabung), bukan 2050. **Anda sendiri yang harus mengubah tipenya** dengan int() dan float().

⚠️ AWAS

Nilai `True` dan `False` di CSV juga teks. `bool("False")` menghasilkan `True` (karena teks tak kosong dianggap benar)! Cocokkan teksnya secara eksplisit, seperti pada proyek di bawah.

Mengapa bukan `split(",")`

```
# Bab 14 - Koma di dalam isi: alasan memakai modul csv, bukan split(",")
import csv
import io

mentah = 'plot,catatan\nA,"dekat sungai, tanah gambut"\nB,"disebut ""hutan keramat""\noleh warga"\n'
print("Isi mentah:")
print(mentah)

print("Cara naif split(','):")
for baris in mentah.splitlines()[1:]:
    print("  ", baris.split(","))

print("Memakai csv.reader:")
for baris in csv.reader(io.StringIO(mentah)):
    print("  ", baris)
```

```
Isi mentah:
plot,catatan
A,"dekat sungai, tanah gambut"
B,"disebut ""hutan keramat"" oleh warga"

Cara naif split(','):
['A', 'dekat sungai', ' tanah gambut']
['B', 'disebut ""hutan keramat"" oleh warga']
Memakai csv.reader:
['plot', 'catatan']
['A', 'dekat sungai, tanah gambut']
['B', 'disebut "hutan keramat" oleh warga']
```

Catatan “dekat sungai, tanah gambut” memuat koma. CSV membungkus isi seperti itu dengan tanda kutip, dan tanda kutip di dalam isi digandakan (“”). `split(",")` buta terhadap aturan itu dan memotong catatan menjadi tiga potong yang salah; `csv.reader` menanganinya dengan benar. Inilah pelajaran dari Bab 13: format buatan sendiri akan menabrak kasus yang tidak Anda duga, sedangkan modul standar sudah memikirkannya.

JSON: data bersarang

CSV hanya cocok untuk tabel datar. Untuk data bersarang (misalnya dict berisi list), pakai **JSON** (*JavaScript Object Notation*), format teks yang dipakai hampir semua layanan internet.

```
# Bab 14 - JSON: data bersarang sebagai teks
import json

kolam = {"kolam": "A", "jenis": "lele", "jumlah": 800, "ph_air": 7.2,
        "aktif": True, "catatan": None, "sampel_g": [520, 480, 560]}

teks = json.dumps(kolam)                                # dict -> teks JSON
print(teks)
print(json.dumps(kolam, indent=2, ensure_ascii=False)[:90], "...")
```

```

kembali = json.loads(teks)                # teks JSON -> dict
print(kembali == kolam, type(kembali).__name__)

print(json.dumps({"suhu": "28 °C"}))      # bawaan: huruf khusus
di-escape
print(json.dumps({"suhu": "28 °C"}, ensure_ascii=False)) # tampil apa adanya
print(json.dumps((1, 2)), json.dumps({1: "a"})) # tuple jadi list, kunci
angka jadi teks

```

```

{"kolam": "A", "jenis": "lele", "jumlah": 800, "ph_air": 7.2, "aktif": true, "catatan":
null, "sampel_g": [520, 480, 560]}
{
  "kolam": "A",
  "jenis": "lele",
  "jumlah": 800,
  "ph_air": 7.2,
  "aktif": true,
  ...
True dict
{"suhu": "28 \u00b0C"}
{"suhu": "28 °C"}
[1, 2] {"1": "a"}

```

- `json.dumps()` mengubah objek Python menjadi **teks** JSON (`s = string`); `json.loads()` kebalikannya.
- Padanan tipe: `True` menjadi `true`, `None` menjadi `null`, `dict` menjadi objek `{...}`, `list` menjadi larik `[...]`. Teks JSON memakai **tanda kutip ganda**.
- `indent=2` merapikan tampilan agar enak dibaca manusia.
- `ensure_ascii=False` menampilkan ° apa adanya; tanpa itu, JSON menulis ° (tetap sah, hanya kurang enak dibaca).
- Dua kejutan di baris terakhir: tuple `(1, 2)` kembali sebagai **list**, dan kunci angka `1` menjadi teks `"1"`. JSON tidak punya tuple, dan kuncinya selalu teks.

JSON ke berkas

```

# Bab 14 - JSON ke dan dari berkas, dengan data bersarang lima sektor
import json
from pathlib import Path

from data_inti import pohon, tanaman, ikan, ternak

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

semua = {"pohon": pohon, "tanaman": tanaman, "ikan": ikan, "ternak": ternak}

with open(berkas, "w", encoding="utf-8") as f:
    json.dump(semua, f, indent=2, ensure_ascii=False)

with open(berkas, encoding="utf-8") as f:
    kembali = json.load(f)

print("Sama persis?", kembali == semua)
print("Bagian:", list(kembali))
print("Pohon pertama:", kembali["pohon"][0])
print("Tipe 'sehat':", type(kembali["pohon"][2]["sehat"]).__name__)
print("Ukuran berkas:", berkas.stat().st_size, "byte")

```

```
Sama persis? True
Bagian: ['pohon', 'tanaman', 'ikan', 'ternak']
Pohon pertama: {'jenis': 'Jati', 'diameter_cm': 32.5, 'tinggi_m': 18.0, 'sehat': True, 'plot': 'A'}
Tipe 'sehat': bool
Ukuran berkas: 2200 byte
```

Berbeda dengan CSV, JSON **mengembalikan tipe**: `sehat` kembali sebagai `bool`, angka kembali sebagai `int` atau `float`, dan seluruh struktur (empat tabel dalam satu dict) kembali **persis sama** dengan aslinya. (Ukuran berkas dihitung pada Windows, dan bisa sedikit berbeda di komputer lain karena akhir baris.) Untuk menyimpan satu set data utuh, JSON biasanya lebih nyaman; untuk tabel besar yang akan dibuka di Excel, CSV lebih cocok.

Dua masalah JSON

```
# Bab 14 - Dua masalah JSON yang umum: teks rusak dan tipe yang tidak dikenal
import json
from datetime import date

rusak = '{"kolam": "A", "jumlah": 800,}'          # koma berlebih di akhir: JSON tidak
mengizinkan
try:
    json.loads(rusak)
except json.JSONDecodeError as e:
    print("JSON rusak:", type(e).__name__, "pada baris", e.lineno)

catatan = {"kolam": "A", "tanggal_tebat": date(2026, 3, 1)}
try:
    json.dumps(catatan)
except TypeError as e:
    print("Tidak bisa:", e)

print("Dengan default=str:", json.dumps(catatan, default=str))
print("Cara tetap: simpan tanggal sebagai teks ISO ->", date(2026, 3, 1).isoformat())
```

```
JSON rusak: JSONDecodeError pada baris 1
Tidak bisa: Object of type date is not JSON serializable
Dengan default=str: {"kolam": "A", "tanggal_tebat": "2026-03-01"}
Cara tetap: simpan tanggal sebagai teks ISO -> 2026-03-01
```

1. **Teks rusak**: koma berlebih di akhir membuat `JSONDecodeError`. Ini sering terjadi bila JSON ditulis atau disunting tangan. Pesan lengkapnya menyebut baris dan kolom; kami hanya menampilkan nama error dan nomor barisnya karena kalimat pesan berbeda antarversi Python.
2. **Tipe tak dikenal**: JSON tidak mengenal tanggal, jadi `json.dumps` melempar `TypeError`. Jalan keluar cepat: `default=str`. Kebiasaan yang lebih baik: simpan tanggal sebagai teks ISO (`2026-03-01`) dan ubah kembali dengan `date.fromisoformat()` (Bab 11).

Pickle: praktis, tetapi berbahaya

Pickle menyimpan hampir **apa saja** dari Python (objek buatan sendiri, fungsi, set) ke dalam byte. Praktis, tetapi dokumentasi Python menyatakan terang-terangan bahwa modul **pickle tidak aman** dan hanya boleh dipakai untuk data yang Anda percayai, karena data pickle yang dirancang jahat dapat **menjalankan kode apa saja** saat dibuka. Mari buktikan, **dengan cara yang aman**: kami membuat pickle yang hanya menjalankan `print`, tidak ada perintah yang berbahaya.

```
# Bab 14 - pickle: praktis untuk data sendiri, BERBAHAYA untuk data orang lain
import pickle
import pickletools
```

```

class Pembawa:
    """Objek yang sengaja membuat pickle MENJALANKAN fungsi saat dibuka."""
    def __reduce__(self):
        # Di serangan nyata, ini bisa berupa perintah merusak. Di sini hanya print.
        return (print, ("!!! kode ini berjalan hanya karena pickle.loads() dipanggil",))

aman = {"kolam": "A", "jumlah": 800}
paket_aman = pickle.dumps(aman)
print("Pickle data sendiri dibuka:", pickle.loads(paket_aman))

paket_berisiko = pickle.dumps(Pembawa(), protocol=4) # protokol dipatok agar keluaran
sama di semua versi
print("Sebelum loads: belum ada apa-apa yang berjalan")
pickle.loads(paket_berisiko) # <- inilah bahayanya
print("Setelah loads: print di atas dijalankan oleh pickle, bukan oleh kita")

print("--- melihat isi pickle TANPA membukanya (pickletools) ---")
pickletools.dis(paket_berisiko)

```

```

Pickle data sendiri dibuka: {'kolam': 'A', 'jumlah': 800}
Sebelum loads: belum ada apa-apa yang berjalan
!!! kode ini berjalan hanya karena pickle.loads() dipanggil
Setelah loads: print di atas dijalankan oleh pickle, bukan oleh kita
--- melihat isi pickle TANPA membukanya (pickletools) ---
  0: \x80 PROTO      4
  2: \x95 FRAME      88
 11: \x8c SHORT_BINUNICODE 'builtins'
 21: \x94 MEMOIZE    (as 0)
 22: \x8c SHORT_BINUNICODE 'print'
 29: \x94 MEMOIZE    (as 1)
 30: \x93 STACK_GLOBAL
 31: \x94 MEMOIZE    (as 2)
 32: \x8c SHORT_BINUNICODE '!!! kode ini berjalan hanya karena pickle.loads()
dipanggil'
 93: \x94 MEMOIZE    (as 3)
 94: \x85 TUPLE1
 95: \x94 MEMOIZE    (as 4)
 96: R      REDUCE
 97: \x94 MEMOIZE    (as 5)
 98: .      STOP
highest protocol among opcodes = 4

```

Telusuri hasilnya:

1. Pickle data sendiri (dict kolam) terbuka biasa.
2. Kita membuat pickle dari objek `Pembawa` yang `__reduce__`-nya berkata “tolong panggil fungsi ini saat pickle dibuka”.
3. Di antara dua `print` kita, muncul baris `!!! kode ini berjalan ... Kita tidak memanggil print itu; pickle.loads() yang memanggilnya`. Bila isinya perintah penghapus berkas, hasilnya sama: berjalan tanpa izin.
4. `pickletools.dis()` membaca pickle **tanpa menjalankannya** dan memperlihatkan dengan jelas bahwa pickle itu memanggil `builtins.print` (`STACK_GLOBAL` lalu `REDUCE`). Dokumentasi Python juga menyarankan `python -m pickletools` sebagai cara yang lebih aman daripada membuka pickle untuk memeriksanya.

⚠️ AWAS

Jangan pernah membuka (`pickle.load` atau `pickle.loads`) berkas pickle yang berasal dari orang lain, dari internet, atau dari surel, dan jangan menerimanya dari pengguna aplikasi web. Untuk data dari luar, pakai **JSON** atau CSV. Pickle hanya untuk data yang **Anda sendiri** tulis dan simpan di tempat yang aman.

Proyek Bersambung B3: simpan dan muat lima sektor

Data lima sektor (hutan, kebun, pertanian, perikanan, peternakan; kebun dan pertanian berbagi tabel tanaman) kini disimpan ke CSV dan JSON. Perkakasnya ada di modul `alat_data.py`: `simpan_csv`, `muat_csv` (dengan **skema** yang menyebut pengubah tipe tiap kolom), `simpan_json`, dan `muat_json`.

```
# Bab 14 - Proyek B3: data kelima sektor disimpan ke CSV dan JSON, dibaca kembali,
dibandingkan
from pathlib import Path

from alat_data import simpan_csv, muat_csv, simpan_json, muat_json
from data_inti import pohon, tanaman, ikan, ternak

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

tabel = {"pohon": pohon, "tanaman": tanaman, "ikan": ikan, "ternak": ternak}
bool_teks = lambda s: {"True": True, "False": False}[s] # KeyError bila bukan
True/False
skema = {
    "pohon": {"jenis": str, "diameter_cm": float, "tinggi_m": float, "sehat": bool_teks,
    "plot": str},
    "tanaman": {"komoditas": str, "blok": str, "umur_hari": int, "hasil_kg": float},
    "ikan": {"kolam": str, "jenis": str, "jumlah": int, "bobot_rata_g": float, "ph_air":
float},
    "ternak": {"kandang": str, "jenis": str, "umur_bulan": int, "bobot_kg": float},
}

for nama, baris in tabel.items():
    simpan_csv(tmp / f"{nama}.csv", baris)
simpan_json(tmp / "inventaris.json", tabel)

for nama in tabel:
    dibaca, ditolak = muat_csv(tmp / f"{nama}.csv", skema[nama])
    sama = dibaca == tabel[nama]
    print(f"{nama:8} CSV: {len(dibaca)} baris, ditolak {ditolak}, sama dengan asli:
{sama}")

print("JSON utuh sama dengan asli:", muat_json(tmp / "inventaris.json") == tabel)
print("Isi pohon.csv (3 baris pertama):")
for baris in (tmp / "pohon.csv").read_text(encoding="utf-8").splitlines()[1:4]:
    print(baris)
```

```
pohon    CSV: 6 baris, ditolak 0, sama dengan asli: True
tanaman  CSV: 4 baris, ditolak 0, sama dengan asli: True
ikan     CSV: 3 baris, ditolak 0, sama dengan asli: True
ternak   CSV: 4 baris, ditolak 0, sama dengan asli: True
JSON utuh sama dengan asli: True
Isi pohon.csv (3 baris pertama):
jenis,diameter_cm,tinggi_m,sehat,plot
```

```
Jati,32.5,18.0,True,A
Meranti,45.2,27.5,True,A
```

Hasil sama dengan asli: True untuk keempat tabel membuktikan bahwa data yang kembali dari CSV (setelah dikonversi sesuai skema) **identik** dengan yang disimpan, termasuk kolom **sehat** yang berupa bool. JSON juga identik. Sekarang uji ketahanannya:

```
# Bab 14 - Proyek B3: memuat CSV yang sudah dikotori petugas, dan berkas yang hilang
atau rusak
import logging
import sys
from pathlib import Path

from alat_data import simpan_csv, muat_csv, muat_json
from data_inti import tanaman

logging.basicConfig(stream=sys.stdout, level=logging.WARNING, format="%(levelname)s:
%(message)s")

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

simpan_csv(berkas, tanaman)
with open(berkas, "a", newline="", encoding="utf-8") as f:      # ulah petugas lapangan
    f.write("padi,E,seratus,250.0\n")                          # umur bukan angka
    f.write("jagung,F,95\n")                                   # kolom kurang

skema = {"komoditas": str, "blok": str, "umur_hari": int, "hasil_kg": float}
baris, ditolak = muat_csv(berkas, skema)
print(f"Diterima {len(baris)} baris, ditolak {ditolak}")
print("Total panen sah:", sum(b["hasil_kg"] for b in baris), "kg")

print("Berkas hilang:", muat_csv(tmp / "tidak_ada.csv", skema))

(tmp / "rusak.json").write_text('{"pohon": [1, 2,', encoding="utf-8")
print("JSON rusak -> cadangan:", muat_json(tmp / "rusak.json", cadangan={}))
```

```
WARNING: baris 6 di tanaman_kotor.csv ditolak (ValueError)
WARNING: baris 7 di tanaman_kotor.csv ditolak (TypeError)
Diterima 4 baris, ditolak 2
Total panen sah: 1030.0 kg
WARNING: berkas tidak_ada.csv belum ada
Berkas hilang: ([], 0)
WARNING: berkas rusak.json bukan JSON yang sah
JSON rusak -> cadangan: {}
```

Petugas menambahkan dua baris rusak: **seratus** tidak bisa menjadi `int` (`ValueError`), dan baris dengan kolom kurang membuat `DictReader` mengisi `None`, yang gagal diubah (`TypeError`). Keduanya **ditolak dengan peringatan**, empat baris sah tetap terbaca (total 1030,0 kg, cocok dengan Bab 11). Berkas hilang menghasilkan daftar kosong, dan JSON rusak menghasilkan nilai cadangan `{}`. Tidak ada satu pun yang meruntuhkan program: itulah arti “tahan terhadap input salah”.

Latihan

1. Tulis data ternak ke `ternak.csv`, baca kembali dengan `DictReader`, dan cetak total `bobot_kg` (jangan lupa mengubah tipenya).
2. Simpan dict `{"nama": "Kolam 01", "suhu": "28 °C"}` ke JSON dengan dan tanpa `ensure_ascii=False`. Bandingkan isi berkasnya.
3. Ubah `08_proyek_simpan.py` agar `tinggi_m` pohon dibulatkan ke bilangan bulat saat dimuat (ubah skemanya, bukan datanya).

4. **Tantangan:** tulis fungsi `bandingkan_csv_json(nama)` yang memuat tabel `nama` dari CSV dan dari JSON, lalu melaporkan apakah keduanya sama. Apa yang terjadi bila ada nilai `True` yang disimpan lewat CSV tanpa pengubah `bool_teks`?

Ringkasan Bab

- **Serialisasi** = mengubah data di memori menjadi teks/byte untuk disimpan; **deserialisasi** = sebaliknya.
- **CSV**: tabel datar; buka dengan `newline=""`; semua nilai datang sebagai **teks**, ubah tipenya sendiri.
- Jangan memotong CSV dengan `split(",")`; pakai modul `csv`.
- **JSON**: data bersarang; mengembalikan `bool`, angka, `None`, list, dict; tidak mengenal tuple dan tanggal.
- **Pickle tidak aman**: membukanya bisa menjalankan kode. Jangan buka pickle dari sumber yang tidak Anda percayai.
- Proyek B3: lima sektor tersimpan di CSV/JSON dan dimuat kembali dengan skema, tahan baris rusak, berkas hilang, dan JSON rusak.

Selanjutnya, di Bab 15: otomasi dokumen kantor, mengubah data tadi menjadi laporan Excel, Word, dan PDF.

Bab 15 — Otomasi Dokumen Kantor: Dari Data Menjadi Laporan Excel dan PDF

Seri B3 — Program Kecil yang Berguna

Di bab ini, Anda akan:

- Membuat dan membaca berkas **Excel** dengan `openpyxl` dan `pandas`
- Membuat dan membaca berkas **Word** dengan `python-docx`
- Membuat **PDF** dengan `reportlab`, membaca dan menggabungkannya dengan `pypdf`, dan mengambil tabelnya dengan `pdfplumber`
- Menyusun objek **email** dengan lampiran, dan mengujinya ke server SMTP lokal tiruan (tanpa mengirim email sungguhan)
- Menyelesaikan proyek Seri B3: rekap data lapangan menjadi laporan Excel dan PDF

Semua skrip bab ini ada di folder `kode/bab15/`. Di versi web, Anda juga bisa [mengunduh semuanya sekaligus \(zip\)](#). Berkas hasil ditulis ke subfolder `_tmp/`.

Setiap akhir pekan, seseorang di kantor lapangan menyalin angka dari tabel ke Excel, memformatnya, menjadikannya PDF, lalu menempelkannya ke email. Pekerjaan itu membosankan, mudah salah, dan memakan waktu yang sama setiap minggu. Python bisa melakukannya dalam hitungan detik. Itulah **otomasi dokumen kantor**: program yang membuat dokumen, bukan manusia yang mengetik ulang.

Dari data lapangan menjadi laporan otomatis



Gambar 0.1: Alur proyek bab ini: CSV lapangan dimuat dengan tahan salah, diringkas, lalu keluar sebagai Excel, PDF, dan lampiran email.

Memasang pustaka

Bab ini memakai pustaka di luar pustaka standar (Bab 11). Perintah pemasangannya (dikitik di terminal) hanya contoh dan **tidak dijalankan** saat buku ini dibuat, karena memasang paket mengubah komputer pembaca:

```
python -m pip install openpyxl pandas python-docx pypdf reportlab pdfplumber aiosmtpd
```

Pustaka	Kegunaan di bab ini
<code>openpyxl</code>	membaca dan menulis berkas Excel <code>.xlsx</code> (lembar, gaya, rumus, grafik)
<code>pandas</code>	mengolah tabel, lalu membaca dan menulis Excel dengan satu baris
<code>python-docx</code>	membuat dan membaca berkas Word <code>.docx</code>
<code>reportlab</code>	membuat PDF dari nol
<code>pypdf</code>	membaca teks PDF, serta menggabung dan memisah halaman
<code>pdfplumber</code>	mengambil tabel dari PDF
<code>aiosmtpd</code>	server SMTP tiruan untuk menguji pengiriman email secara lokal

Semuanya kami jalankan di komputer penulis (Python 3.14) sebelum ditulis di sini; keluaran di buku adalah hasil asli. Semua pustaka ini tidak membutuhkan Microsoft Office; berkas yang dihasilkan bisa dibuka di Excel atau LibreOffice.

Excel dengan openpyxl

```
# Bab 15 - Excel dengan openpyxl: membuat buku kerja, judul tebal, angka, dan rumus
from pathlib import Path

from openpyxl import Workbook
from openpyxl.styles import Font, PatternFill

from data_inti import tanaman

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

buku = Workbook()
lembar = buku.active
lembar.title = "Panen"

lembar.append(["Komoditas", "Blok", "Umur (hari)", "Hasil (kg)"]) # baris judul
for t in tanaman:
    lembar.append([t["komoditas"], t["blok"], t["umur_hari"], t["hasil_kg"]])

for sel in lembar[1]: # gaya baris judul
    sel.font = Font(bold=True)
    sel.fill = PatternFill("solid", start_color="D7ECCF")
lembar.column_dimensions["A"].width = 14

akhir = lembar.max_row
lembar[f"C{akhir + 1}"] = "Total"
lembar[f"D{akhir + 1}"] = f"=SUM(D2:D{akhir})" # rumus Excel
sebagai teks
lembar[f"D{akhir + 1}"].number_format = "#,##0.0"

buku.save(tmp / "panen.xlsx")
print("Baris terisi:", lembar.max_row, "| kolom:", lembar.max_column)
print("Isi sel D6 :", lembar["D6"].value)
```

```
Baris terisi: 6 | kolom: 4
Isi sel D6 : =SUM(D2:D5)
```

- `Workbook()` membuat buku kerja baru; `.active` adalah lembar pertamanya.
- `lembar.append(...)` menambah satu baris di bawah baris terakhir.
- `Font(bold=True)` dan `PatternFill(...)` memberi gaya pada baris judul.
- Rumus Excel ditulis sebagai teks yang diawali `=`. Baris data ada di 2 sampai 5, jadi total ditaruh di baris 6, sel D6.

Pada kolom D6, Python hanya **menyimpan teks rumusnya**. Siapa yang menghitung? Excel atau LibreOffice, saat berkas dibuka. Kami membuktikannya dengan mengubah berkas itu ke CSV lewat LibreOffice: baris terakhirnya berisi total 1030 (sama dengan hasil Bab 11).

```
# Bab 15 - Membaca Excel: nilai rumus hanya ada bila Excel pernah menghitungnya
from pathlib import Path

from openpyxl import Workbook, load_workbook

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

buku = Workbook()
ws = buku.active
ws.append(["blok", "hasil_kg"])
for baris in [{"A", 245.5}, {"B", 310.0}, {"C", 198.5}]:
    ws.append(baris)
ws["B5"] = "=SUM(B2:B4)"
buku.save(berkas)

dibuka = load_workbook(berkas) # bawaan: rumus dibaca sebagai teks
print("Daftar lembar:", dibuka.sheetnames)
for baris in dibuka.active.iter_rows(min_row=2, values_only=True):
    print(baris)

print("Rumus      :", dibuka.active["B5"].value)
nilai = load_workbook(berkas, data_only=True) # minta HASIL hitungan terakhir
print("Hasilnya   :", nilai.active["B5"].value) # None: file ini belum pernah
dibuka Excel
```

```
Daftar lembar: ['Sheet']
('A', 245.5)
('B', 310)
('C', 198.5)
(None, '=SUM(B2:B4)')
Rumus      : =SUM(B2:B4)
Hasilnya   : None
```

Bila dibuka biasa, sel B5 berisi **teks rumusnya**. Bila dibuka dengan `data_only=True`, ia memberi hasil hitungan terakhir yang disimpan Excel; dokumentasi `openpyxl` menyatakan “nilai yang tersimpan saat Excel terakhir membaca lembar”. Karena berkas ini baru dibuat oleh Python dan belum pernah dibuka di Excel, hasilnya `None`. Hal kecil lain yang perlu Anda tahu: `310.0` dibaca kembali sebagai `310` (bilangan bulat), sebab Excel tidak membedakan keduanya.

⚠️ AWAS

`openpyxl` **tidak menghitung rumus**. Bila yang Anda perlukan adalah angkanya (untuk dipakai program lain atau dimasukkan ke PDF), hitung di Python dan tulis angkanya. Tulis rumus hanya bila pembaca di Excel perlu melihat atau mengubah rumusnya.

Grafik di dalam Excel

```
# Bab 15 - Lembar rekap dengan diagram batang bawaan Excel (openpyxl.chart)
import zipfile
from pathlib import Path

from openpyxl import Workbook
from openpyxl.chart import BarChart, Reference

from data_inti import tanaman, ikan, ternak

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

total_panen = sum(t["hasil_kg"] for t in tanaman)
total_ikan = sum(i["jumlah"] * i["bobot_rata_g"] / 1000 for i in ikan)
total_ternak = sum(t["bobot_kg"] for t in ternak)

buku = Workbook()
ws = buku.active
ws.title = "Rekap"
ws.append(["Sektor", "Total (kg)"])
ws.append(["Panen kebun dan sawah", total_panen])
ws.append(["Biomassa ikan", round(total_ikan, 1)])
ws.append(["Bobot ternak", total_ternak])

grafik = BarChart()
grafik.title = "Total per sektor"
grafik.y_axis.scaling.min = 0 # sumbu mulai dari nol agar jujur
grafik.add_data(Reference(ws, min_col=2, min_row=1, max_row=4), titles_from_data=True)
grafik.set_categories(Reference(ws, min_col=1, min_row=2, max_row=4))
ws.add_chart(grafik, "D2")
buku.save(berkas)

with zipfile.ZipFile(berkas) as z: # .xlsx sebenarnya zip berisi XML
    print("Berkas grafik di dalam xlsx:", [n for n in z.namelist() if "chart" in n
    and n.endswith(".xml")])
    print("Jumlah grafik di lembar:", len(ws._charts))
    print("Isi tabel:", [tuple(b) for b in ws.iter_rows(min_row=2, values_only=True)])
```

```
Berkas grafik di dalam xlsx: ['xl/charts/chart1.xml']
Jumlah grafik di lembar: 1
Isi tabel: [('Panen kebun dan sawah', 1030.0), ('Biomassa ikan', 1095.4), ('Bobot
ternak', 1190.0)]
```

Grafik batang dibuat dari `BarChart` dan dua `Reference` (rentang data dan rentang label). `scaling.min = 0` memaksa sumbu mulai dari nol: tanpa itu, Excel bisa memilih sumbu mulai dari angka 950 sehingga batang tampak berbeda jauh padahal selisihnya kecil. Berkas `.xlsx` sebenarnya **zip berisi XML**; keluaran menunjukkan `xl/charts/chart1.xml` ada di dalamnya.

Excel dengan pandas

Untuk **tabel**, `pandas` lebih ringkas. Satu `DataFrame` adalah tabel di memori:

```
# Bab 15 - pandas: CSV ke ringkasan ke Excel (beberapa lembar) dan kembali
from pathlib import Path

import pandas as pd
```

```

from data_inti import pohon, ikan

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

df_pohon = pd.DataFrame(pohon)
df_ikan = pd.DataFrame(ikan)
df_ikan["biomassa_kg"] = df_ikan["jumlah"] * df_ikan["bobot_rata_g"] / 1000

rekap_plot = df_pohon.groupby("plot")["diameter_cm"].agg(["count", "mean"]).round(2)
print(rekap_plot.to_string())

berkas = tmp / "pandas_rekap.xlsx"
with pd.ExcelWriter(berkas, engine="openpyxl") as penulis:          # satu berkas,
    banyak lembar
        df_pohon.to_excel(penulis, sheet_name="Pohon", index=False)
        df_ikan.to_excel(penulis, sheet_name="Ikan", index=False)
        rekap_plot.to_excel(penulis, sheet_name="Rekap plot")

kembali = pd.read_excel(berkas, sheet_name=None)                  # None = semua
lembar sekaligus
print("Lembar:", list(kembali))
print("Biomassa total ikan:", kembali["Ikan"]["biomassa_kg"].sum(), "kg")
print("Tipe kolom sehat   :", kembali["Pohon"]["sehat"].dtype)

```

```

      count  mean
plot
A          2  38.85
B          2  39.75
C          2  38.40
Lembar: ['Pohon', 'Ikan', 'Rekap plot']
Biomassa total ikan: 1095.4 kg
Tipe kolom sehat   : bool

```

- `DataFrame(pohon)` mengubah list dict menjadi tabel.
- `groupby("plot")` mengelompokkan per plot, `agg(["count", "mean"])` menghitung jumlah dan rata-rata. Plot A: 2 pohon, rata-rata 38,85 cm.
- `pd.ExcelWriter` menulis beberapa lembar dalam satu berkas; `read_excel(sheet_name=None)` membaca semua lembar sekaligus sebagai dict.
- Kolom `sehat` kembali bertipe `bool`, dan biomassa total 1095,4 kg (432,4 + 375,0 + 288,0). (Pandas dibahas penuh di Seri I2.)

Word dengan python-docx

```

# Bab 15 - Word dengan python-docx: judul, paragraf, tabel, lalu dibaca kembali
from pathlib import Path

from docx import Document

from data_inti import ikan

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

dok = Document()
dok.add_heading("Laporan Kolam (data contoh)", level=1)
dok.add_paragraph("Rekap biomassa tiga kolam pada awal Oktober. Angka fiktif untuk

```

```

latihan.")

tabel = dok.add_table(rows=1, cols=3)
tabel.style = "Table Grid"
for sel, judul in zip(tabel.rows[0].cells, ["Kolam", "Jenis", "Biomassa (kg)"]):
    sel.text = judul
for i in ikan:
    baris = tabel.add_row().cells
    baris[0].text = i["kolam"]
    baris[1].text = i["jenis"]
    baris[2].text = f"{i['jumlah'] * i['bobot_rata_g'] / 1000:.1f}"
dok.save(berkas)

dibaca = Document(berkas) # buka lagi dan periksa
print("Paragraf:", [p.text for p in dibaca.paragraphs])
print("Jumlah tabel:", len(dibaca.tables), "| baris:", len(dibaca.tables[0].rows))
for baris in dibaca.tables[0].rows:
    print([sel.text for sel in baris.cells])

```

```

Paragraf: ['Laporan Kolam (data contoh)', 'Rekap biomassa tiga kolam pada awal Oktober.
Angka fiktif untuk latihan.']
Jumlah tabel: 1 | baris: 4
['Kolam', 'Jenis', 'Biomassa (kg)']
['A', 'lele', '432.4']
['B', 'nila', '375.0']
['C', 'lele', '288.0']

```

`Document()` membuat dokumen kosong; `add_heading`, `add_paragraph`, dan `add_table` menambah isinya. Setelah disimpan, kita **membukanya lagi** dan membaca paragraf serta sel tabelnya: itulah buktinya berkas tidak kosong. Gaya "Table Grid" (tabel bergaris) ada di templat bawaan python-docx; kami mengujinya langsung.

PDF: membuat dengan reportlab

PDF dirancang untuk **dibaca**, bukan disunting. Untuk membuatnya dari Python, `reportlab` menyediakan "platypus": Anda menyusun daftar elemen (judul, paragraf, tabel), lalu `build`. Pembuat dasarnya ada di modul `laporan_pdf.py` (lihat folder `kode/bab15/`).

```

# Bab 15 - PDF dengan reportlab: membuat laporan satu halaman
from pathlib import Path

from laporan_pdf import buat_pdf
from data_inti import ternak

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

tabel = [{"Kandang", "Jenis", "Umur (bulan)", "Bobot (kg)"}]
for t in ternak:
    tabel.append([t["kandang"], t["jenis"], str(t["umur_bulan"]),
f"{t['bobot_kg']:.1f}"])
buat_pdf(berkas, "Laporan Ternak", tabel, "Bobot ternak per kandang (angka contoh).")

print("PDF dibuat:", berkas.name, "| awal berkas:", berkas.read_bytes()[:8])

```

```
PDF dibuat: ternak.pdf | awal berkas: b'%PDF-1.4'
```

Awal berkas `%PDF-1.4` adalah tanda pengenal format PDF.

PDF: membaca dan mengambil tabel

```
# Bab 15 - Membaca PDF: teks dengan pypdf, tabel dengan pdfplumber
from pathlib import Path

import pdfplumber
from pypdf import PdfReader

from laporan_pdf import buat_pdf
from data_inti import ternak

tmp = Path(__file__).parent / "_tmp"
tmp.mkdir(exist_ok=True)
berkas = tmp / "ternak_baca.pdf"
tabel = [{"Kandang", "Jenis", "Umur (bulan)", "Bobot (kg)"}]
tabel += [{"kandang", t["jenis"], str(t["umur_bulan"]), f"{t['bobot_kg']:.1f}"} for
t in ternak]
buat_pdf(berkas, "Laporan Ternak", tabel, "Bobot ternak per kandang (angka contoh).")

pembaca = PdfReader(berkas)
print("Jumlah halaman:", len(pembaca.pages))
print("Judul metadata:", pembaca.metadata.title)
print("--- teks (pypdf) ---")
baris_teks = pembaca.pages[0].extract_text().splitlines()
print(len(baris_teks), "baris; 7 pertama:", baris_teks[:7])

print("--- tabel (pdfplumber) ---")
with pdfplumber.open(berkas) as pdf:
    for baris in pdf.pages[0].extract_tables()[0]:
        print(baris)
```

```
Jumlah halaman: 1
Judul metadata: Laporan Ternak
--- teks (pypdf) ---
22 baris; 7 pertama: ['Laporan Ternak', 'Bobot ternak per kandang (angka contoh).',
'Kandang', 'Jenis', 'Umur (bulan)', 'Bobot (kg)', 'K1']
--- tabel (pdfplumber) ---
['Kandang', 'Jenis', 'Umur (bulan)', 'Bobot (kg)']
['K1', 'sapi', '36', '350.0']
['K1', 'sapi', '30', '412.5']
['K2', 'sapi', '42', '389.0']
['K3', 'kambing', '14', '38.5']
```

Dua alat, dua hasil berbeda:

- **pypdf** mengambil **teks**, tetapi struktur tabelnya hilang: tiap sel menjadi satu baris (22 baris, dan sel-sel tampil berurutan, Kandang, Jenis, Umur (bulan), ...).
- **pdfplumber** mengenali garis tabel dan mengembalikan tiap baris sebagai list sel: lima baris rapi.

Catatan jujur dari dokumentasi pdfplumber: ia bekerja paling baik pada PDF **buatan mesin** (seperti PDF yang kita buat), bukan PDF hasil **pindai** (foto kertas). PDF pindai butuh OCR, yang tidak dibahas di buku ini. Untuk PDF tanpa garis tabel, hasil `extract_tables()` bisa kurang tepat; selalu periksa.

PDF: menggabung dan memisah

```
# Bab 15 - Menggabung dan memisah PDF dengan pypdf
from pathlib import Path

from pypdf import PdfReader, PdfWriter
```

```

from laporan_pdf import buat_pdf

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

buat_pdf(tmp / "bagian_hutan.pdf", "Bagian Hutan", [{"Plot", "Pohon"}, [{"A", "2"}, [{"B", "2"}]]
buat_pdf(tmp / "bagian_kolam.pdf", "Bagian Kolam", [{"Kolam", "kg"}, [{"A", "432.4"}, [{"B", "375.0"}]]

penulis = PdfWriter()
for nama in ["bagian_hutan.pdf", "bagian_kolam.pdf"]:
    penulis.append(str(tmp / nama)) # tambahkan semua halaman berkas itu
penulis.add_metadata({"Title": "Laporan Gabungan"})
with open(tmp / "gabungan.pdf", "wb") as f:
    penulis.write(f)

gabungan = PdfReader(tmp / "gabungan.pdf")
print("Halaman gabungan:", len(gabungan.pages))
for no, halaman in enumerate(gabungan.pages, start=1):
    print(f"  halaman {no}:", halaman.extract_text().splitlines()[0])

satu = PdfWriter() # memisah: ambil halaman ke-2 saja
satu.add_page(gabungan.pages[1])
with open(tmp / "halaman2.pdf", "wb") as f:
    satu.write(f)
print("Halaman hasil pisah:", len(PdfReader(tmp / "halaman2.pdf").pages))

```

```

Halaman gabungan: 2
  halaman 1: Bagian Hutan
  halaman 2: Bagian Kolam
Halaman hasil pisah: 1

```

`PdfWriter.append()` menambahkan seluruh halaman berkas lain; dua PDF satu halaman menjadi satu PDF dua halaman. Memisah dilakukan dengan memilih halaman (`pages[1]` adalah halaman kedua, karena penomoran dari 0) dan menulisnya sebagai berkas baru.

Email: membuat pesan

Python punya modul `email.message` untuk **menyusun** surel, termasuk lampiran:

```

# Bab 15 - Membuat objek email dengan email.message (TIDAK dikirim ke mana pun)
from email.message import EmailMessage
from pathlib import Path

from laporan_pdf import buat_pdf

tmp = Path(__file__).parent / "_tmp"
tmp.mkdir(exist_ok=True)
lampiran = tmp / "rekap.pdf"
buat_pdf(lampiran, "Rekap Mingguan", [{"Sektor", "Total"}, [{"Panen (kg)", "1030.0"}]]

pesan = EmailMessage()
pesan["Subject"] = "Rekap data lapangan mingguan"
pesan["From"] = "pelaporan@contoh.example" # alamat fiktif (domain .example
dicadangkan)
pesan["To"] = "kepala-unit@contoh.example"
pesan.set_content("Selamat pagi,\n\nTerlampir rekap data lapangan minggu ini.\n\nSalam,\nTim Lapangan")
pesan.add_attachment(lampiran.read_bytes(), maintype="application", subtype="pdf",

```

```

        filename="rekap_mingguan.pdf")

print("Subjek      :", pesan["Subject"])
print("Dari/Ke     :", pesan["From"], "->", pesan["To"])
print("Multipart?", pesan.is_multipart())
print("Isi         teks                                :",
pesan.get_body(preferencelist=("plain",)).get_content().splitlines()[0])
for lamp in pesan.iter_attachments():
    print("Lampiran :", lamp.get_filename(), lamp.get_content_type())
print("Surel ini hanya objek di memori; belum ada yang dikirim.")

```

```

Subjek      : Rekap data lapangan mingguan
Dari/Ke     : pelaporan@contoh.example -> kepala-unit@contoh.example
Multipart?  True
Isi teks    : Selamat pagi,
Lampiran    : rekap_mingguan.pdf application/pdf
Surel ini hanya objek di memori; belum ada yang dikirim.

```

Pesan ini adalah **objek di memori**; belum ada yang dikirim. Alamatnya fiktif, memakai domain `.example` yang memang dicadangkan untuk contoh. Pesan ber-lampiran otomatis menjadi *multipart* (True).

AWAS

Buku ini **tidak mengirim email sungguhan** dan kami minta Anda berhati-hati: email otomatis yang salah alamat atau berulang tak terkendali bisa membocorkan data atau membanjiri kotak surat orang. Selalu uji dulu ke server tiruan, dan jangan menulis kata sandi di kode (ambil dari variabel lingkungan).

Email: uji ke server lokal

Cara aman menguji pengiriman: jalankan **server SMTP tiruan di komputer sendiri**, dan kirim ke sana. Kami memakai `aiosmtpd`; dokumentasinya menyatakan `Controller` menjalankan server SMTP di thread terpisah sehingga cocok untuk menyiapkan server dalam pengujian.

```

# Bab 15 - Mengirim email ke server SMTP LOKAL tiruan (hanya di komputer ini, tidak
ke internet)
import smtplib
import socket
from email.message import EmailMessage

from aiosmtpd.controller import Controller

class Penampung:
    """Penangan server tiruan: menyimpan setiap surat yang masuk."""
    def __init__(self):
        self.surat = []

    async def handle_DATA(self, server, session, envelope):
        self.surat.append(envelope)
        return "250 Pesan diterima (server tiruan)"

def port_bebas():
    """Meminta sistem operasi memilih satu port kosong di komputer ini."""
    with socket.socket() as s:
        s.bind(("127.0.0.1", 0))
        return s.getsockname()[1]

```

```

penampung = Penampung()
port = port_bebas()
server = Controller(penampung, hostname="127.0.0.1", port=port) # hanya menerima dari
komputer ini
server.start()
try:
    pesan = EmailMessage()
    pesan["Subject"] = "Uji kirim lokal"
    pesan["From"] = "pelaporan@contoh.example"
    pesan["To"] = "kepala-unit@contoh.example"
    pesan.set_content("Ini hanya uji ke server tiruan di komputer sendiri.")

    with smtplib.SMTP("127.0.0.1", port) as klien:
        klien.send_message(pesan)
finally:
    server.stop()

masuk = penampung.surat[0]
print("Surat diterima server tiruan:", len(penampung.surat))
print("Pengirim :", masuk.mail_from)
print("Penerima :", masuk.rcpt_tos)
print("Ada subjek uji?", b"Subject: Uji kirim lokal" in masuk.content)

```

```

Surat diterima server tiruan: 1
Pengirim : pelaporan@contoh.example
Penerima : ['kepala-unit@contoh.example']
Ada subjek uji? True

```

Server hanya mendengarkan di 127.0.0.1 (alamat “komputer ini sendiri”), jadi suratnya tidak pernah keluar ke internet. Server tiruan menerima satu surat, dan pengirim, penerima, dan subjek cocok dengan yang kita kirim. Untuk mengirim sungguhan, Anda memakai `smtplib.SMTP_SSL` dengan alamat server, nama pengguna, dan kata sandi dari penyedia surel Anda. Bentuknya seperti ini (**tidak dijalankan** di sini karena akan mengirim email sungguhan):

```

import os, smtplib, ssl
with smtplib.SMTP_SSL(os.environ["SMTP_HOST"], 465,
context=ssl.create_default_context()) as s:
    s.login(os.environ["SMTP_USER"], os.environ["SMTP_PASSWORD"]) # nilainya:
[disamarkan]
    s.send_message(pesan)

```

Proyek Bersambung B3: rekap lapangan menjadi laporan

Kini seluruh Seri B3 bekerja bersama. Skrip berikut: (1) menyimpan data lima sektor ke CSV, dan **sengaja menambah satu baris rusak** pada data ikan, (2) memuatnya kembali dengan skema (Bab 14), (3) menghitung rekap per sektor, (4) membuat laporan **Excel** (lembar Rekap + grafik + satu lembar per tabel), (5) membuat laporan **PDF**, dan (6) menyusun email dengan dua lampiran (tidak dikirim).

```

# Bab 15 - Proyek B3: dari CSV lapangan menjadi laporan Excel dan PDF, plus surel
(tidak dikirim)
import logging
import sys
from email.message import EmailMessage
from pathlib import Path

from openpyxl import Workbook
from openpyxl.chart import BarChart, Reference
from openpyxl.styles import Font, PatternFill

```

```

from alat_data import simpan_csv, muat_csv
from data_inti import pohon, tanaman, ikan, ternak
from laporan_pdf import buat_pdf

logging.basicConfig(stream=sys.stdout, level=logging.WARNING, format="%(levelname)s:
%(message)s")
kerja = Path(__file__).parent / "_tmp" / "proyek"
kerja.mkdir(parents=True, exist_ok=True)

# 1. Simpan data mentah ke CSV; petugas menambah satu baris rusak di data ikan
tabel = {"pohon": pohon, "tanaman": tanaman, "ikan": ikan, "ternak": ternak}
for nama, baris in tabel.items():
    simpan_csv(kerja / f"{nama}.csv", baris)
with open(kerja / "ikan.csv", "a", newline="", encoding="utf-8") as f:
    f.write("D,nila,banyak,350.0,7.4\n")

# 2. Muat kembali dengan skema (baris rusak ditolak)
bool_teks = lambda s: {"True": True, "False": False}[s]
skema = {
    "pohon": {"jenis": str, "diameter_cm": float, "tinggi_m": float, "sehat": bool_teks,
    "plot": str},
    "tanaman": {"komoditas": str, "blok": str, "umur_hari": int, "hasil_kg": float},
    "ikan": {"kolam": str, "jenis": str, "jumlah": int, "bobot_rata_g": float, "ph_air":
float},
    "ternak": {"kandang": str, "jenis": str, "umur_bulan": int, "bobot_kg": float},
}
data, ditolak = {}, 0
for nama in tabel:
    data[nama], tolak = muat_csv(kerja / f"{nama}.csv", skema[nama])
    ditolak += tolak

# 3. Hitung rekap per sektor
biomassa = sum(i["jumlah"] * i["bobot_rata_g"] / 1000 for i in data["ikan"])
rekap = [
    ["Sektor", "Catatan", "Ukuran", "Nilai"],
    ["Hutan", len(data["pohon"]), "Rata-rata diameter (cm)",
    round(sum(p["diameter_cm"] for p in data["pohon"]) / len(data["pohon"]), 1)],
    ["Hutan", len(data["pohon"]), "Pohon sehat", sum(p["sehat"] for p in
data["pohon"])],
    ["Kebun dan sawah", len(data["tanaman"]), "Total panen (kg)", sum(t["hasil_kg"]
for t in data["tanaman"])],
    ["Perikanan", len(data["ikan"]), "Biomassa (kg)", round(biomassa, 1)],
    ["Peternakan", len(data["ternak"]), "Total bobot (kg)", sum(t["bobot_kg"] for t
in data["ternak"])],
]

# 4. Laporan Excel: lembar Rekap + grafik + satu lembar per tabel
buku = Workbook()
ws = buku.active
ws.title = "Rekap"
for baris in rekap:
    ws.append(baris)
ws.append([])
ws.append([f"Baris data ditolak karena rusak: {ditolak}"])
for sel in ws[1]:
    sel.font = Font(bold=True)
    sel.fill = PatternFill("solid", start_color="D7ECCF")
ws.column_dimensions["A"].width = 18
ws.column_dimensions["C"].width = 26

```

```

grafik = BarChart()
grafik.title = "Nilai per ukuran (kg)"
grafik.y_axis.scaling.min = 0
grafik.add_data(Reference(ws, min_col=4, min_row=4, max_row=6),
titles_from_data=False)
grafik.set_categories(Reference(ws, min_col=3, min_row=4, max_row=6))
ws.add_chart(grafik, "F2")
for nama, baris in data.items():
    lembar = buku.create_sheet(nama.capitalize())
    lembar.append(list(baris[0]))
    for b in baris:
        lembar.append(list(b.values()))
buku.save(kerja / "laporan.xlsx")

# 5. Laporan PDF
isi_pdf = [rekap[0]] + [[s, str(n), u, str(v)] for s, n, u, v in recap[1:]]
buat_pdf(kerja / "laporan.pdf", "Rekap Inventarisasi Lapangan", isi_pdf,
        f"Angka contoh. Baris data ditolak karena rusak: {ditolak}.")

# 6. Surel dengan dua lampiran (hanya dibuat, tidak dikirim)
pesan = EmailMessage()
pesan["Subject"] = "Rekap inventarisasi lapangan"
pesan["From"] = "pelaporan@contoh.example"
pesan["To"] = "kepala-unit@contoh.example"
pesan.set_content("Terlampir rekap inventarisasi dalam Excel dan PDF.")
pesan.add_attachment((kerja / "laporan.xlsx").read_bytes(), maintype="application",
        subtype="vnd.openxmlformats-officedocument.spreadsheetml.sheet",
        filename="laporan.xlsx")
pesan.add_attachment((kerja / "laporan.pdf").read_bytes(), maintype="application",
        subtype="pdf", filename="laporan.pdf")

print(f"Baris ditolak: {ditolak}")
for baris in recap[1:]:
    print(f"{baris[0]:16} {baris[1]:>2} {baris[2]:24} {baris[3]}")
print("Berkas laporan:", sorted(p.name for p in kerja.iterdir() if p.suffix in (".xlsx",
    ".pdf")))
print("Lampiran surel:", [a.get_filename() for a in pesan.iter_attachments()], "(belum
    dikirim)")

```

```

WARNING: baris 5 di ikan.csv ditolak (ValueError)
Baris ditolak: 1
Hutan          6 Rata-rata diameter (cm)  39.0
Hutan          6 Pohon sehat              5
Kebun dan sawah 4 Total panen (kg)          1030.0
Perikanan      3 Biomassa (kg)           1095.4
Pernakan       4 Total bobot (kg)         1190.0
Berkas laporan: ['laporan.pdf', 'laporan.xlsx']
Lampiran surel: ['laporan.xlsx', 'laporan.pdf'] (belum dikirim)

```

Telusuri hasilnya:

- Baris rusak D, nila, banyak, ... ditolak (jumlahnya banyak, bukan angka), dan laporan mencatat **baris ditolak: 1**. Sisa tiga kolam tetap diolah: biomassa 1095,4 kg.
- Rekap per sektor cocok dengan Bab 11: panen 1030,0 kg, bobot ternak 1190,0 kg, rata-rata diameter 39,0 cm; pohon sehat 5 dari 6.
- Berkas laporan.xlsx dan laporan.pdf terbentuk, dan email berlampiran keduanya siap, belum dikirim. Kami juga membuka laporan.xlsx dengan LibreOffice (diubah ke PDF) untuk memastikan berkasnya sah dan grafiknya muncul.

Sebuah program otomatis tidak boleh dipercaya begitu saja. Skrip berikutnya **membaca kembali** laporan yang dihasilkan dan membandingkan Excel dengan PDF:

```
# Bab 15 - Proyek B3: memeriksa laporan yang dihasilkan dengan membacanya kembali
(bukti, bukan asumsi)
import subprocess
import sys
from pathlib import Path

import pdfplumber
from openpyxl import load_workbook
from pypdf import PdfReader

folder = Path(__file__).parent
kerja = folder / "_tmp" / "proyek"
subprocess.run([sys.executable, str(folder / "11_proyek_laporan.py")],
capture_output=True, check=True)

buku = load_workbook(kerja / "laporan.xlsx")
print("Lembar Excel:", buku.sheetnames)
rekap = list(buku["Rekap"].iter_rows(min_row=2, max_row=6, values_only=True))
total_excel = {baris[2]: baris[3] for baris in rekap}
print("Excel - total panen :", total_excel["Total panen (kg)"])
print("Excel - biomassa      :", total_excel["Biomassa (kg)"])
print("Excel - baris ikan   :", buku["Ikan"].max_row - 1, "(tanpa baris judul)")

pembaca = PdfReader(kerja / "laporan.pdf")
print("PDF - halaman       :", len(pembaca.pages))
with pdfplumber.open(kerja / "laporan.pdf") as pdf:
    tabel = pdf.pages[0].extract_tables()[0]
    total_pdf = {baris[2]: baris[3] for baris in tabel[1:]}
    print("PDF - total panen :", total_pdf["Total panen (kg)"])
    print("PDF - biomassa      :", total_pdf["Biomassa (kg)"])

# Excel menyimpan 1030.0 sebagai 1030, jadi bandingkan sebagai angka, bukan teks
cocok = all(float(total_excel[k]) == float(total_pdf[k]) for k in ("Total panen (kg)",
"Biomassa (kg)"))
print("Excel dan PDF konsisten:", cocok)
```

```
Lembar Excel: ['Rekap', 'Pohon', 'Tanaman', 'Ikan', 'Ternak']
Excel - total panen : 1030
Excel - biomassa      : 1095.4
Excel - baris ikan   : 3 (tanpa baris judul)
PDF - halaman       : 1
PDF - total panen   : 1030.0
PDF - biomassa      : 1095.4
Excel dan PDF konsisten: True
```

Excel memiliki lima lembar, tabel ikan berisi 3 baris data, dan angka total di Excel dan PDF sama. Perhatikan satu pelajaran kecil: Excel menyimpan total panen sebagai **1030**, sedangkan PDF menulis teks **1030.0**. Membandingkan keduanya sebagai **teks** akan melaporkan “berbeda”, padahal angkanya sama; kami membandingkannya sebagai **angka** (`float`). Pemeriksaan otomatis seperti ini membuat Anda berani menjadwalkan laporan berjalan sendiri.

Latihan

1. Ubah `01_excel_tulis.py`: tambahkan kolom “Hasil per 100 hari (kg)” yang diisi Python (bukan rumus), yaitu `hasil_kg / umur_hari * 100`, dibulatkan dua desimal.
2. Pakai `pandas` untuk membaca `ikan.csv` hasil Bab 14 dan menulis `ikan.xlsx` dengan satu lembar bernama “Ikan”. Buka dengan `read_excel` dan cetak rata-rata `bobot_rata_g`.
3. Ubah `11_proyek_laporan.py` agar laporan PDF-nya juga memuat tabel ringkasan **per plot** hutan (jumlah pohon dan rata-rata diameter).

4. **Tantangan:** buat skrip yang menggabungkan `laporan.pdf` dari proyek dengan satu PDF sampel buatan Anda sendiri (satu halaman judul) menjadi `laporan_lengkap.pdf`, lalu periksa dengan `pypdf` bahwa halaman pertama adalah sampel.

Ringkasan Bab

- **Excel:** `openpyxl` (lembar, gaya, rumus, grafik) dan `pandas` (tabel); `openpyxl` tidak menghitung rumus.
- **Word:** `python-docx` untuk judul, paragraf, dan tabel.
- **PDF:** `reportlab` membuat; `pypdf` membaca teks, menggabung, dan memisah; `pdfplumber` mengambil tabel dari PDF buatan mesin.
- **Email:** `email.message.EmailMessage` menyusun pesan; uji ke server SMTP lokal tiruan; jangan menulis kata sandi di kode.
- Laporan otomatis perlu **diperiksa otomatis**: baca kembali hasilnya dan bandingkan.
- Proyek B3 lengkap: data lima sektor tersimpan, tahan input salah, dan menjadi laporan Excel/PDF.

Penutup Seri B3

Anda sudah menyelesaikan Seri B3. Program Anda kini bisa **meminjam alat** dari pustaka (Bab 11), **tidak runtuh** saat ada yang salah (Bab 12), **menyimpan dan membaca berkas** (Bab 13), **memakai format standar CSV dan JSON** (Bab 14), dan **membuat laporan Excel dan PDF otomatis** (Bab 15). Anda juga sudah punya satu proyek utuh: data hutan, kebun, pertanian, perikanan, dan peternakan yang tersimpan, tahan salah, dan menjadi laporan.

Selamat: Tingkat Basic (Jilid 1) tamat!

Dengan bab ini, **Tingkat Basic selesai**. Dalam 15 bab, Anda berangkat dari “apa itu komputer” sampai menulis program yang berguna di dunia kerja: variabel, keputusan, perulangan, struktur data, fungsi, modul, penanganan error, berkas, format data, dan otomasi dokumen. Itu fondasi yang sungguh-sungguh, dan banyak pekerjaan kantor sehari-hari sudah bisa Anda otomasi dengannya.

Tetapi lihat sendiri: program proyek kita makin besar, fungsinya makin banyak, dan mulai **berantakan**: data dan perkakas tercampur, ada salinan berkas `data_inti.py` di beberapa folder bab. **Tingkat Intermediate dimulai dengan cara menata kode** supaya tetap rapi dan aman diubah.

Selanjutnya, di Bab 16: OOP (pemrograman berorientasi objek): kelas, objek, pewarisan, dan cara menyusun pohon, kebun, kolam, dan kandang sebagai objek yang punya perilaku sendiri.

Lampiran A — Pemasangan di Windows, Linux, dan macOS

Di bab ini, Anda akan:

- Memasang Python 3.14 di Windows, Linux, atau macOS (atau memakai `uv` di ketiganya)
- Membuat dan mengaktifkan lingkungan virtual (**venv**) di tiap sistem
- Memeriksa bahwa pemasangan Anda sehat, memakai skrip kecil
- Memasang alat pendukung buku: VS Code dan Git
- Mengenali masalah pemasangan yang paling umum dan cara mengatasinya

Semua skrip lampiran ini ada di folder `kode/bab71/`. Di versi web, Anda juga bisa [mengunduh semuanya sekaligus](#) (zip).

Bab 4 sudah menuntun Anda memasang Python di Windows. Lampiran ini melengkapinya: ia mengumpulkan **semua sistem operasi** di satu tempat, ditambah cara membuat lingkungan virtual, alat pendukung, dan daftar “kalau begini, coba begitu”. Anggap ini seperti buku petunjuk di laci bengkel: tidak dibaca dari awal sampai akhir, tetapi dibuka bila ada baut yang macet.

Kejujuran dulu: apa yang diuji, apa yang tidak

Buku ini ditulis dan dijalankan di **satu komputer Windows 11**. Karena itu:

Bagian	Status	Dasarnya
Windows (Python install manager, <code>py</code> , <code>venv</code> , <code>uv</code>)	Diuji di komputer penulis	Perintah dijalankan; keluarannya ditampilkan
Linux (paket distro, <code>pyenv</code> , <code>uv</code>)	Belum diuji di Linux	Dokumentasi resmi, dibaca 2026-10-10
macOS (pemasang <code>python.org</code> , Homebrew, <code>uv</code>)	Belum diuji di macOS	Dokumentasi resmi, dibaca 2026-10-10

Setiap perintah Linux dan macOS di bawah ini ditandai asalnya. Bila sebuah perintah tidak saya temukan di dokumentasi resmi, saya tulis **belum diverifikasi** dan tidak saya sajikan sebagai kepastian. Nomor versi di komputer Anda boleh berbeda dari yang tertera di sini.

Terminal: tempat mengetik perintah

Semua perintah di lampiran ini diketik di **terminal** (dijelaskan di Bab 3):

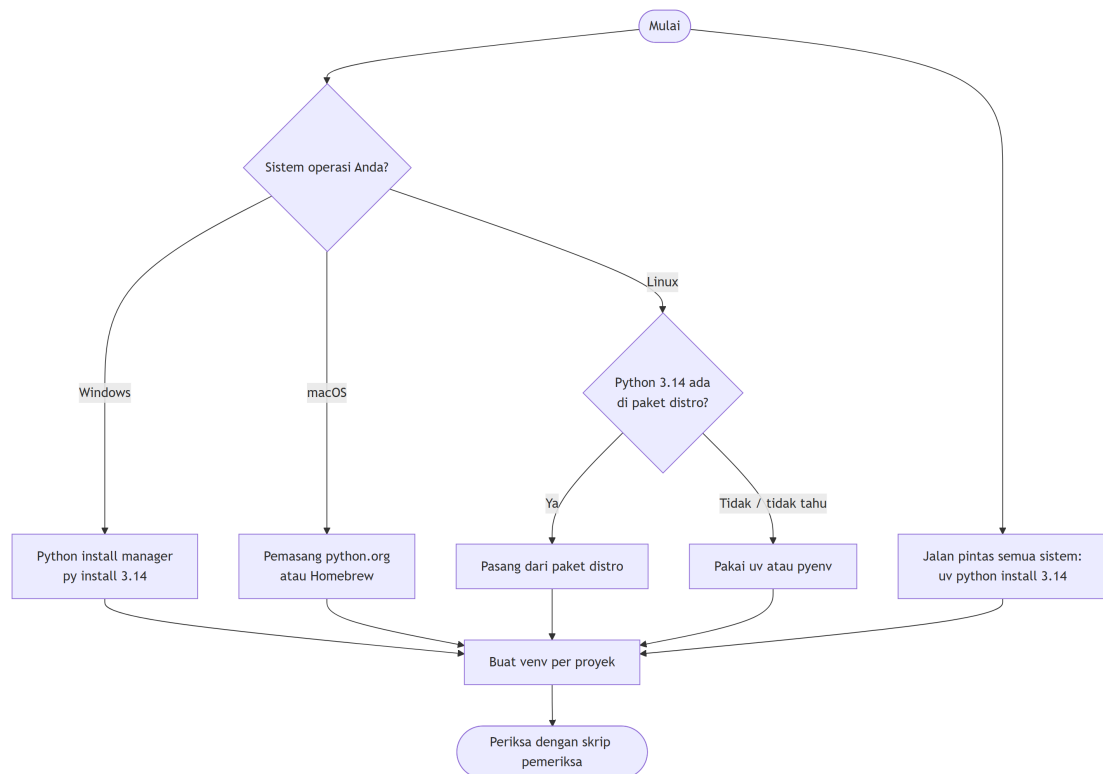
- **Windows:** buka *PowerShell* (cari “PowerShell” di menu Start).
- **macOS:** buka aplikasi *Terminal* (di Launchpad, folder Other).
- **Linux:** buka aplikasi *Terminal* dari menu aplikasi Anda.

Tanda `$` di awal baris contoh Linux dan macOS hanyalah tanda “siap menerima perintah” (*prompt*). Jangan ikut mengetiknya.

i INGAT

Python punya dua “sebutan” yang membingungkan pemula: `python` dan `python3`. Di Windows dengan Python install manager, `python` dan `py` sama-sama bisa dipakai. Di Linux dan macOS, biasanya mengetik **`python3`**, karena `python` kadang tidak ada atau menunjuk ke versi lain.

Peta keputusan: cara memasang yang mana?



Memilih cara memasang Python: jalur utama per sistem, dan uv sebagai jalan pintas di semuanya. Telusuri dari oval **Mulai**. Hampir semua jalur bermuara di langkah yang sama: **membuat venv per proyek** (bagian di bawah), lalu memeriksa hasilnya.

Windows

Python install manager

Dokumentasi resmi Python 3.14 untuk Windows (docs.python.org/3.14/using/windows.html, dibaca 2026-10-10) menyebut **Python install manager** sebagai cara yang disarankan. Pemasang lama berbentuk satu berkas besar dinyatakan usang sejak 3.14 dan tidak akan dibuat lagi untuk Python 3.16 ke atas. Ada tiga cara memasang pengelolanya:

1. **Microsoft Store**: cari aplikasi Python install manager lalu klik Install.
2. **Berkas MSIX dari python.org**: unduh dari python.org/downloads, klik dua kali, pilih Install. Paket Store dan paket python.org identik, tetapi **tidak bisa dipasang bersamaan**.
3. **MSI** (untuk keperluan khusus, misalnya Windows Server 2019): tanpa antarmuka dan memasang untuk seluruh komputer. Pemula tidak perlu ini.

Setelah pengelola terpasang, pasang Pythonnya:

```
py install 3.14
```

Perintah pengelola lain yang tercantum di dokumentasi yang sama:

Perintah	Fungsi
<code>py list</code>	Menampilkan runtime Python yang terpasang
<code>py list --online 3.14</code>	Menampilkan yang bisa dipasang
<code>py install 3.14</code>	Memasang Python 3.14
<code>py uninstall 3.14</code>	Mencopot Python 3.14

Perintah	Fungsi
<code>py -V:3.14 skrip.py</code>	Menjalankan skrip dengan Python 3.14 (opsi ini harus ditulis lebih dulu)
<code>py install --refresh</code>	Membuat ulang pintasan Start menu dan alias

Hasil di komputer penulis

Di komputer penulis, perintah `pymanager list` (nama lengkap pengelola, lihat kotak AWAS di bawah) menghasilkan ini pada 2026-10-10. Teks di bawah **disalin dari keluaran asli**, bukan dibuat saat buku dibangun:

```
Tag      Name      Managed By  Version  Alias
-- No runtimes. Use 'py install <version>' to install one. --

* These runtimes were found, but cannot be updated or uninstalled. *
3.14 * Python 3.14 (64-bit) PythonCore 3.14.8
3.12 Python 3.12 (64-bit) PythonCore 3.12.10
```

Artinya: pengelola sendiri belum memasang Python apa pun, tetapi ia **menemukan** dua Python yang dipasang lewat cara lama (3.14.8 dan 3.12.10). Bintang * menandai versi bawaan. Keduanya tetap bisa dipakai lewat `py`.

⚠️ AWAS

Di komputer penulis ada **tiga** berkas bernama `py.exe` sekaligus: satu di `C:\WINDOWS`, satu di folder Python lama, satu lagi dari install manager. Ketika penulis mengetik `py list`, yang menjawab adalah `py.exe` **lama**, yang mengira `list` adalah nama skrip dan menjawab “*can't open file ... list*”. Sedangkan `pymanager list` selalu memanggil pengelola yang baru. Dokumentasi resmi mencatat gejala yang sama: peluncur lama bisa mengambil prioritas, dan obatnya adalah mencopot “Python launcher” dari **Installed apps**. Bila perintah `py install` atau `py list` Anda tidak dikenal, gunakan `pymanager` sebagai gantinya.

Memeriksa versi

```
py --version
```

Di komputer penulis: `Python 3.14.8`. Perintah ini bekerja baik dengan peluncur lama maupun baru. Dokumentasi pengelola tidak membahas `py --version` secara khusus; pembuktiannya hanya dari uji penulis di atas.

Folder PATH

Dokumentasi menyebut penambahan `%LocalAppData%\Python\bin` ke **PATH** (daftar folder tempat Windows mencari program) sebagai pilihan, bukan kewajiban. Biarkan juga `%UserProfile%\AppData\Local\Microsoft\WindowsApps` tetap ada di PATH, sebab menghapusnya dapat merusak pintasan.

Pemasangan lewat skrip (opsional)

Untuk pemasangan otomatis, dokumentasi resmi menyarankan WinGet:

```
winget install 9NQ7512CXL7T -e --accept-package-agreements --disable-interactivity
py install --configure -y
```

Kode ini **tidak dijalankan** saat buku dibuat karena mengubah komputer penulis, jadi keluarannya tidak ditampilkan; perintahnya disalin dari dokumentasi resmi di atas.

Linux

Status: belum diuji di Linux. Isi bagian ini bersumber dari dokumentasi resmi (docs.python.org/3.14/using/unix.html, dibaca 2026-10-10).

Python dari paket distro

Dokumentasi resmi menyatakan Python sudah terpasang bawaan di sebagian besar distribusi Linux dan tersedia sebagai paket di distribusi lainnya. Dokumen itu **tidak** memberi perintah pemasangan Python itu sendiri (ia menyerahkannya pada panduan tiap distro). Periksa dulu apa yang Anda punya:

```
$ python3 --version
```

Versinya mungkin bukan 3.14: itu tergantung distribusi dan rilisnya, dan saya tidak dapat memastikannya untuk distro Anda. Buku ini dijalankan dengan Python 3.14.8; hampir semua skrip sudah dicoba pula di 3.12.10 dengan hasil yang sama (lihat kotak TIPS di Bab 4), tetapi Bab 47 membahas fitur yang hanya ada di versi tertentu.

Dokumentasi resmi memuat perintah untuk **IDLE** (editor bawaan Python) yang kadang tidak ikut terpasang:

```
$ sudo apt update && sudo apt install idle          # Debian dan Ubuntu
$ sudo dnf install python3-idle                     # Fedora, RHEL, CentOS
$ sudo zypper install python3-idle                  # SUSE dan openSUSE
$ sudo apk add python3-idle                         # Alpine
```

i INGAT

Di Linux, `python3 -m venv` dan `pip` kadang disediakan sebagai paket **terpisah** oleh distribusinya. Panduan resmi pengemasan Python menulis: “*On Linux, you may have to install an additional package such as `python3-pip`*”. Nama paket persis untuk `venv` di distro Anda (di Debian dan Ubuntu kemungkinan `python3-venv`) **belum saya verifikasi ke dokumentasi distro**; bila `python3 -m venv` mengeluh soal `ensurepip`, cari paket `venv` milik distro Anda.

pyenv: memasang banyak versi Python

pyenv memasang beberapa versi Python berdampingan dengan mengompilasinya. Menurut github.com/pyenv/pyenv (dibaca 2026-10-10):

```
$ curl -fsSL https://pyenv.run | bash
$ pyenv install -l          # daftar versi yang bisa dipasang
$ pyenv install 3.14        # awalan versi dipetakan ke rilis terbaru yang cocok
$ pyenv global 3.14
```

Sebelum `pyenv install`, **pasang dulu pustaka pembangun** (*build dependencies*); daftarnya ada di wiki `pyenv`, bagian “Suggested build environment”. `Pyenv` juga butuh penyiapan shell (baris `eval "$(pyenv init - bash)”` di `~/.bashrc`, atau padanannya untuk `zsh` dan `fish`; lihat README-nya). Karena `pyenv` mengompilasi Python, pemasangan bisa makan waktu beberapa menit.

Catatan: README `pyenv` menyatakan `pyenv` **tidak mendukung Windows** di luar WSL (Windows Subsystem for Linux); bagi Windows ia merekomendasikan fork bernama `pyenv-win`. Buku ini tidak membahas `pyenv-win`.

Membangun dari kode sumber

Dokumentasi resmi memberi urutan `./configure`, `make`, `make install`, tetapi **menyarankan `make altinstall`** supaya berkas `python3` bawaan sistem tidak tertimpa. Ini jalur tingkat lanjut; pemula sebaiknya memakai `uv` atau `pyenv`.

macOS

Status: belum diuji di macOS. Sumber: docs.python.org/3.14/using/mac.html dan docs.brew.sh/Language-Runtimes-and-Packages, dibaca 2026-10-10.

Pemasang dari python.org

Unduh pemasang dari python.org/downloads dan klik dua kali. Menurut dokumentasi resmi, pemasang ini berjenis *universal2* (berjalan di Mac Apple Silicon maupun Intel) dan memerlukan akun macOS dengan hak **Administrator**. Hasilnya: folder `Python 3.14` di *Applications* (berisi IDLE dan Python Launcher), kerangka di `/Library/Frameworks/Python.framework`, dan tautan di `/usr/local/bin/`.

Setelah itu, **klik dua kali Install Certificates.command** di folder `/Applications/Python 3.14/`. Ini memasang sertifikat keamanan agar Python bisa mengunduh dari internet. Tanda berhasilnya: muncul tulisan *Successfully installed certifi dan update complete*.

Jalankan Python dengan `python3` (atau `python3.14`):

```
$ python3 --version
```

⚠️ AWAS

macOS membawa `/usr/bin/python3` bawaan Apple. Dokumentasi resmi menyebutnya “biasanya lebih tua dan tidak lengkap”, dibuat untuk alat pengembang Apple, dan **jangan diubah atau dihapus**. Setelah memasang dari `python.org`, `python3` Anda semestinya sudah menunjuk ke yang baru; bila ragu, periksa dengan `which python3`.

Homebrew

Homebrew adalah pengelola paket untuk macOS (brew.sh). Dokumentasi Python menyebutnya sebagai distribusi alternatif yang tidak dipelihara tim inti Python. Dokumentasi Homebrew menyatakan paket Python bawaannya memasang `python3` dan `pip3`, dan menyarankan memasang **dependensi proyek di venv**, bukan di lingkungan dasar Homebrew (yang ditandai “externally managed” menurut PEP 668). Dokumentasi Homebrew yang saya baca **tidak** mencantumkan perintah `brew install python` secara harfiah (ia menyuruh memakai `brew info python` untuk melihat versi terkini); panduan VS Code menulis `brew install python3`. Karena itu, periksa dulu dengan `brew info python`.

uv: jalan pintas di semua sistem

uv (dikenalkan di Bab 17) bisa memasang Python sekaligus mengelola proyek. Menurut docs.astral.sh/uv (dibaca 2026-10-10), memasang uv:

```
# macOS dan Linux
$ curl -LsSf https://astral.sh/uv/install.sh | sh

# Windows (PowerShell)
powershell -ExecutionPolicy ByPass -c "irm https://astral.sh/uv/install.ps1 | iex"

# alternatif: Homebrew (macOS) atau WinGet (Windows)
$ brew install uv
winget install --id=astral-sh.uv -e
```

Lalu Python-nya (dari [panduan uv](#)):

```
uv python install 3.14    # pasang Python 3.14
uv python list            # lihat yang tersedia dan terpasang
uv venv                   # buat venv; pasang Python lebih dulu bila belum ada
```

Menurut panduan itu, uv memakai Python yang sudah ada di sistem bila cocok dan, secara bawaan, mengunduh sendiri Python yang dibutuhkan. Di komputer penulis, `uv --version` menjawab `0.11.28`; uv sudah terpasang sebelumnya, jadi pemasangan uv sendiri **tidak diuji ulang**.

BAGIAN TEKNIS (boleh dilewati)

Perintah `curl ... | sh` mengunduh sebuah skrip lalu langsung menjalankannya. Resmi dan lazim, tetapi Anda sebaiknya terbiasa **membaca skripnya dulu** (buka alamatnya di browser) atau memakai pengelola paket (`brew`, `winget`). Uji dulu di komputer yang bukan komputer produksi.

Lingkungan virtual (venv) di tiap sistem

Bab 17 menjelaskan *mengapa* tiap proyek perlu venv. Ini ringkasan *bagaimana* di tiap sistem, dari [panduan resmi pengemasan](#) dan docs.python.org/3.14/library/venv.html (dibaca 2026-10-10):

Langkah	Windows (PowerShell)	Linux dan macOS (bash/zsh)
Buat	<code>py -m venv .venv</code>	<code>python3 -m venv .venv</code>
Aktifkan	<code>.venv\Scripts\Activate.ps1</code>	<code>source .venv/bin/activate</code>
Di <code>cmd.exe</code>	<code>.venv\Scripts\activate.bat</code>	(tidak berlaku)
Periksa	<code>where python</code>	<code>which python</code>
Periksa pip	<code>py -m pip --version</code>	<code>python3 -m pip --version</code>
Nonaktifkan	<code>deactivate</code>	<code>deactivate</code>

Setelah aktif, nama folder venv muncul di awal prompt, dan `python` menunjuk ke Python di dalam venv.

TIPS

Anda **tidak wajib mengaktifkan** venv. Dokumentasi resmi mengatakan Anda cukup memanggil Python di dalamnya dengan jalur lengkap, misalnya `.venv\Scripts\python.exe skrip.py` di Windows atau `.venv/bin/python skrip.py` di Linux dan macOS. Begitulah proyek buku ini dijalankan.

Uji nyata: membuat venv dari skrip

Python bisa membuat venv lewat modul `venv` (yang sama dipakai perintah `-m venv`). Skrip ini membuatnya di folder `_tmp/`, lalu memeriksa isinya:

```
# Lampiran A - Membuat lingkungan virtual (venv) dari Python
import shutil
import subprocess
import sys
import venv
from pathlib import Path

folder = Path(__file__).parent / "_tmp" / "venv_contoh"
if folder.exists():
    shutil.rmtree(folder) # hanya folder contoh milik skrip ini

venv.create(folder, with_pip=True)
print("Lingkungan dibuat:", folder.exists())

# Di Windows program ada di Scripts/, di Linux dan macOS di bin/
sub = "Scripts" if sys.platform == "win32" else "bin"
isi = sorted(p.name for p in (folder / sub).iterdir())
print("Isi folder", sub + ":", ", ".join(isi[:6]), "...")

python_venv = folder / sub / ("python.exe" if sys.platform == "win32" else "python")
for argumen in (["--version"], ["-m", "pip", "--version"]):
```

```

        hasil = subprocess.run([str(python_venv), *argumen], capture_output=True,
text=True)
        teks = hasil.stdout.strip()
        label = "python " + " ".join(argumen[:2])
        print(f"{label:<18}->", teks.split(" from ")[0]) # buang jalur

```

```

Lingkungan dibuat: True
Isi folder Scripts:  Activate.ps1,  activate,  activate.bat,  activate.fish,
deactivate.bat, pip.exe ...
python --version -> Python 3.14.8
python -m pip -> pip 26.2.1

```

Baris pertama membuktikan venv terbentuk. Baris kedua menunjukkan isi folder `Scripts` di Windows: ada `activate`, `Activate.ps1`, `activate.bat`, dan seterusnya (di Linux dan macOS skrip aktivasinya ada di `bin/`). Dua baris terakhir memanggil Python dan pip **di dalam venv** secara langsung, tanpa aktivasi. Di venv segar ini pip sudah ikut terpasang. Nomor versi pip akan berubah seiring waktu; yang penting ia menjawab.

⚠️ AWAS

Di Windows PowerShell, `Activate.ps1` kadang ditolak dengan pesan tentang *execution policy* (kebijakan menjalankan skrip). Dokumentasi resmi menyebut obatnya: `Set-ExecutionPolicy -ExecutionPolicy RemoteSigned -Scope CurrentUser`. Itu **mengubah pengaturan keamanan akun Anda**, jadi pertimbangkan jalan lain: pakai `cmd.exe` dengan `activate.bat`, atau tidak mengaktifkan venv sama sekali (lihat TIPS di atas). Kebijakan di komputer penulis tidak diuji untuk kasus ini.

Memeriksa lingkungan Anda

Jalankan skrip pemeriksa ini di komputer Anda. Ia mencatat versi Python, sistem operasi, apakah modul yang dibutuhkan ada, dan apakah beberapa perintah ditemukan di PATH:

```

# Lampiran A - Memeriksa lingkungan Python di komputer Anda
import platform
import shutil
import sys

print(f"{'Versi Python':<16}: {platform.python_version()}")
print(f"{'Sistem operasi':<16}: {platform.system()}")
print(f"{'Arsitektur':<16}: {platform.machine()}")
print(f"{'Seri 3.14?':<16}: {'ya' if sys.version_info[:2] == (3, 14) else 'tidak'}")

# Modul bawaan yang dibutuhkan untuk lingkungan virtual
for modul in ("venv", "ensurepip", "tkinter", "sqlite3"):
    try:
        __import__(modul)
        status = "ada"
    except ImportError:
        status = "TIDAK ada"
    print(f"{'modul ' + modul:<16}: {status}")

# Perintah yang dicari di PATH (hasilnya berbeda di tiap komputer)
for perintah in ("python", "py", "uv", "git", "code"):
    status = "ditemukan" if shutil.which(perintah) else "tidak ditemukan"
    print(f"{'perintah ' + perintah:<16}: {status}")

```

```

Versi Python      : 3.14.8
Sistem operasi    : Windows

```

```

Arsitektur      : AMD64
Seri 3.14?      : ya
modul venv      : ada
modul ensurepip : ada
modul tkinter  : ada
modul sqlite3   : ada
perintah python : ditemukan
perintah py     : ditemukan
perintah uv     : ditemukan
perintah git    : ditemukan
perintah code   : ditemukan

```

Ini keluaran **komputer penulis**; keluaran Anda akan berbeda pada baris sistem operasi, arsitektur, dan perintah yang ditemukan. Cara membacanya:

- **Versi Python** dan **Seri 3.14?** memberi tahu apakah Anda memakai versi acuan buku.
- **modul venv** / **ensurepip** harus **ada**. Bila **TIDAK ada** di Linux, lihat kotak INGGAT di bagian Linux.
- **modul tkinter** hanya dibutuhkan untuk contoh antarmuka jendela di Bab 42; **sqlite3** untuk Bab 24.
- **perintah ... ditemukan** berarti ada di PATH. **tidak ditemukan** bukan kesalahan: itu hanya berarti perintah itu belum terpasang atau belum ada di PATH.

Alat lain yang dipakai buku ini

Alat	Untuk apa	Cara memasang	Wajib?
VS Code	Editor kode	code.visualstudio.com	Tidak (Notepad cukup untuk pemula)
Git	Pelacak versi (Bab 20)	git-scm.com/downloads	Mulai Bab 20
Quarto	Menyusun buku ini	-	Tidak perlu untuk membaca atau mengikuti buku

Panduan VS Code (dibaca 2026-10-10) menyebut: pasang ekstensi Python lalu pilih interpreter lewat Command Palette dengan perintah **Python: Select Interpreter**. Untuk Windows ia memeriksa pemasangan dengan `py -3 --version`. Di komputer penulis, VS Code dan Git terpasang (`code` dan `git` ditemukan di skrip pemeriksa; `git --version` menjawab `git version 2.55.0.windows.3`), tetapi proses **memasang**nya tidak diuji ulang.

Alat berat yang hanya dibutuhkan bab tertentu ada petunjuknya di bab itu: PostgreSQL dan PostGIS (Bab 24 dan 40), QGIS (Bab 40), Redis (Bab 67), Docker (Bab 48), serta driver NVIDIA dan CUDA (Bab 55). Pasang **hanya saat Anda sampai di bab itu**.

Masalah umum dan solusinya

Gejala	Kemungkinan penyebab	Yang bisa dicoba
<code>py</code> tidak dikenal (Windows)	Install manager belum terpasang, atau alias dimatikan	Pasang pengelola (lihat bagian Windows); buka terminal baru setelah memasang
<code>py list</code> / <code>py install</code> dijawab “can’t open file”	<code>py.exe</code> lama mengambil prioritas	Pakai <code>pymanager list</code> ; atau copot “Python launcher” lama dari Installed apps (kata dokumentasi resmi)
<code>python</code> membuka Store (Windows)	Microsoft Alias Store aktif tanpa Python terpasang	Pasang Python dulu; perintah <code>py</code> tetap bisa dipakai

Gejala	Kemungkinan penyebab	Yang bisa dicoba
<code>python</code> tidak ditemukan (Linux/macOS)	Hanya ada <code>python3</code>	Ketik <code>python3</code> ; di venv aktif, <code>python</code> tersedia
<code>pip</code> memberi <code>externally-managed-environment</code>	Python milik distro/Homebrew melarang pemasangan global (PEP 668)	Buat venv , lalu pasang di situ; jangan memakai <code>--break-system-packages</code>
<code>python3 -m venv</code> gagal soal <code>ensurepip</code> (Linux)	Paket <code>venv</code> distro belum terpasang	Cari paket <code>venv</code> distro Anda (belum diverifikasi nama paketnya)
<code>Activate.ps1</code> ditolak	Kebijakan eksekusi PowerShell	Pakai <code>activate.bat</code> di <code>cmd</code> , atau panggil <code>python.exe</code> venv langsung
Pustaka gagal dipasang karena tidak ada kompilator	Tidak ada paket jadi (<i>wheel</i>) untuk Python/OS Anda	Coba versi Python lebih lama, <code>uv</code> , atau <code>conda</code> (Bab 17 dan Bab 38)
Sertifikat SSL gagal (macOS <code>python.org</code>)	Lupa menjalankan <code>Install Certificates.command</code>	Jalankan berkasnya (lihat bagian macOS)
Skrip “tidak menemukan modul” padahal sudah dipasang	Dipasang di venv/Python yang berbeda dari yang menjalankan skrip	Jalankan <code>python -m pip install ...</code> dengan Python yang sama; periksa <code>which python / where python</code>
Venv rusak setelah folder dipindah	Venv tidak bisa dipindah atau disalin	Buat ulang venv (kata dokumentasi resmi) dan pasang ulang dari <code>requirements.txt</code>

Baris tentang `py list`, `externally-managed-environment`, `Activate.ps1`, venv yang dipindah, dan sertifikat SSL bersumber dari dokumentasi resmi yang dikutip di lampiran ini. Baris lainnya adalah pengalaman umum yang tidak saya uji di komputer penulis (label: **kemungkinan**); nama paket venv di Linux bahkan **belum diverifikasi**.

i INGAT

Bila bingung, **kembali ke dasar**: (1) `py --version` atau `python3 --version`, (2) buat venv di folder proyek, (3) panggil Python di dalam venv dengan jalur lengkap. Tiga langkah itu menyelesaikan sebagian besar masalah pemasangan.

Selanjutnya: Lampiran B merangkum perintah penting dalam lembar contekan. Bila Anda mencari urutan belajar, lompat ke Lampiran E.

Belajar **Python** dari nol, langkah demi langkah

- Mengenal komputer, terminal, dan cara berpikir program
- Variabel, kondisional, perulangan, dan fungsi
- Berkas, format data, dan otomasi dokumen kantor
- Contoh dari hutan, kebun, sawah, kolam, dan kandang

Jilid 1 menemani Anda dari komputer yang masih kosong sampai bisa menulis program kecil yang benar-benar jalan. Setiap bab memakai data tiruan dan latihan yang bisa dijalankan langsung.

Disusun oleh

Badar Mubarak Yogaswara