Buku Ajar

ALGORITMA DAN PEMROGRAMAN 1



Dr. Yusri Ikhwani, S.Kom., M.Kom.

Buku Ajar

Algoritma dan Pemrograman 1

Penulis :

Dr. Yusri Ikhwani, S.Kom., M.Kom.

Editor :

Antoni Pardede, S.Si., M.Si., Ph.D

Tata Letak:

Abdurrahman Sidik, M.Ds

Desain Sampul:

M. Fikri Ansari, S.Kom

Penerbit :

Universitas Islam Kalimantan Muhammad Arsyad Al-Banjari Banjarmasin

Redaksi :

Jl. Adhyaksa No. 2 Kayutangi

Banjarmasin 70123

Cetakan Pertama : Oktober 2026

ISBN : 978-623-8189-97-7



Hak Cipta dilindungi undang-undang

Dilarang memperbanyak karya tulis ini dalam bentuk dan dengan cara apapun tanpa izin tertulis dari penerbit. Isi diluar tanggung jawab penerbit.

KATA PENGANTAR

Bismillahirrahmanirrahim. Segala puji bagi Allah SWT yang telah memberikan rahmat, kekuatan, dan kemampuan kepada penulis sehingga Buku Ajar Algoritma dan Pemrograman 1 ini dapat diselesaikan.

Buku ini disusun sebagai bahan ajar utama untuk mata kuliah Algoritma dan Pemrograman 1 yang umumnya ditempuh oleh mahasiswa semester pertama program studi Teknik Informatika/Ilmu Komputer, dan Sistem Informasi. Bahasa pemrograman yang dipakai adalah Python 3.10 atau versi yang lebih baru, dipilih karena Python memiliki sintaks yang ramah pemula, ekosistem pustaka yang sangat kaya, dan posisi yang sangat baik di industri serta dunia penelitian.

Empat belas bab buku ini dirancang untuk dipelajari secara berurutan dalam rentang satu semester. Tujuh bab pertama (Bab 1–7) membentuk fondasi pemrograman prosedural, dimulai dari konsep algoritma hingga struktur kontrol perulangan. Bab 8 berfungsi sebagai titik konsolidasi sekaligus persiapan Ujian Tengah Semester. Empat bab berikutnya (Bab 9–12) memperkenalkan struktur data koleksi, fungsi sebagai abstraksi, dan modularisasi proyek. Dua bab terakhir (Bab 13–14) membahas file I/O serta penanganan eksepsi, dan ditutup dengan empat studi kasus proyek mini yang mengintegrasikan seluruh materi.

Setiap bab disusun dengan struktur konsisten: epigraph dan apersepsi sebagai pembuka, peta konsep visual, uraian materi yang dilengkapi contoh kode yang dapat dieksekusi, kotak-kotak khusus (Catatan, Perhatian, Tips Pro, Asah Otak, Sejarah Singkat, Di Balik Layar, Wawasan Industri), pembahasan kesalahan umum, rangkuman, latihan terbimbing dengan pembahasan, latihan mandiri, tugas proyek mini, serta refleksi diri. Pendekatan ini dimaksudkan agar mahasiswa tidak hanya menguasai

sintaks, tetapi juga mengembangkan pola pikir komputasional yang sistematis.

Salah satu hal yang penulis tekankan dalam buku ini adalah pentingnya praktik baik (best practices) sejak awal: penulisan kode yang mengikuti konvensi PEP 8, dokumentasi dengan docstring (PEP 257), penggunaan type hint (PEP 484), modularisasi yang baik, serta penanganan eksepsi yang elegan. Penulis percaya bahwa kebiasaan baik yang dibentuk sejak semester pertama akan menjadi modal berharga ketika mahasiswa memasuki dunia kerja sebagai programmer profesional.

Buku ini tidak akan mungkin tersusun tanpa kontribusi banyak pihak. Penulis mengucapkan terima kasih kepada para kolega di lingkungan kampus yang telah memberikan masukan berharga, mahasiswa-mahasiswa yang telah menjadi peserta uji coba materi, serta keluarga yang selalu memberikan dukungan tak terbatas. Penulis juga mengakui bahwa banyak ide pedagogis dalam buku ini terinspirasi dari karya-karya klasik pemrograman seperti How to Solve It (Polya), The Art of Computer Programming (Knuth), dan The Pragmatic Programmer (Hunt & Thomas).

Akhir kata, semoga buku ini bermanfaat bagi mahasiswa Indonesia yang sedang merintis perjalanan dalam dunia pemrograman, dan menjadi salah satu jembatan kecil yang membantu mereka menjadi pemecah masalah yang andal di masa depan.

Banjarmasin, Oktober 2026

Penulis,

Dr. Yusri Ikhwani, S.Kom., M.Kom.

PETUNJUK PENGGUNAAN BUKU

Untuk Mahasiswa

Bacalah setiap bab secara berurutan. Setiap bab dirancang untuk dipelajari dalam satu pekan kuliah, mencakup pertemuan tatap muka, praktikum, dan studi mandiri. Kerjakan latihan terbimbing setelah membaca uraian materi, lalu lanjutkan ke latihan mandiri untuk mengukur pemahaman Anda. Tugas proyek mini di akhir bab merupakan kesempatan untuk mengintegrasikan konsep yang baru dipelajari ke dalam aplikasi nyata.

Untuk Dosen Pengampu

Bab 1–7 cocok dipakai sebagai materi pra-UTS, dengan Bab 8 sebagai konsolidasi. Bab 9–14 menjadi materi pasca-UTS. Setiap bab mencantumkan Capaian Pembelajaran Bab (CPB) yang dapat dijadikan rujukan untuk perumusan tujuan instruksional. Rubrik penilaian proyek mini dapat disesuaikan dengan kebijakan masing-masing perguruan tinggi.

Tentang Kotak-Kotak Khusus

Tujuh jenis kotak khusus tersebar di seluruh buku. Kotak CATATAN memberikan informasi tambahan penting, PERHATIAN menyoroti jebakan umum, TIPS PRO menyajikan praktik terbaik, ASAH OTAK menantang pembaca berpikir kritis, SEJARAH SINGKAT memperkaya konteks historis, DI BALIK LAYAR mengungkap detail teknis, dan WAWASAN INDUSTRI menghubungkan teori dengan praktik di dunia kerja.

Persyaratan Teknis

Pembaca diharapkan memiliki akses ke komputer dengan Python 3.10 atau versi yang lebih baru terinstal. Editor kode yang direkomendasikan adalah Visual Studio Code (VS Code) dengan ekstensi Python, meskipun editor lain seperti PyCharm atau bahkan IDLE bawaan Python juga dapat dipakai.

Tidak diperlukan koneksi internet untuk menjalankan contoh kode di buku ini.

Kerjakan Setiap Latihan

Pemrograman adalah keterampilan praktis yang tidak dapat dikuasai hanya dengan membaca. Ketiklah setiap contoh kode di komputer Anda, jalankan, perhatikan output, lalu eksperimen dengan modifikasi kecil. Inilah cara paling efektif untuk membangun pemahaman yang dalam dan tahan lama.

DAFTAR ISI

KATA PENGANTAR ..... **iii**

PETUNJUK PENGGUNAAN BUKU ..... **v**

DAFTAR ISI ..... **vii**

BAB 1 Pengantar Algoritma dan Pemrograman **1**

BAB 2 Representasi Algoritma: Flowchart dan Pseudocode **18**

BAB 3 Pengenalan Python dan Lingkungan Pengembangan **41**

BAB 4 Variabel, Tipe Data, dan Operator **63**

BAB 5 Input/Output dan Operasi String **83**

BAB 6 Struktur Kontrol Percabangan **103**

BAB 7 Struktur Kontrol Perulangan **124**

BAB 8 Ujian Tengah Semester: Latihan Komprehensif **145**

BAB 9 List dan Tuple **169**

BAB 10 Dictionary dan Set **192**

BAB 11 Fungsi dan Parameter **211**

BAB 12 Modularisasi dan Scope Variabel **233**

BAB 13 File I/O dan Exception Handling **256**

BAB 14 Studi Kasus Mini-Proyek **276**

DAFTAR PUSTAKA **301**

LAMPIRAN Rencana Pembelajaran Semester (RPS) **305**

BAB 1

PENGANTAR ALGORITMA DAN PEMROGRAMAN

"To understand a program you must become both the machine and the program."
— Alan Perlis, penerima Turing Award pertama (1966)

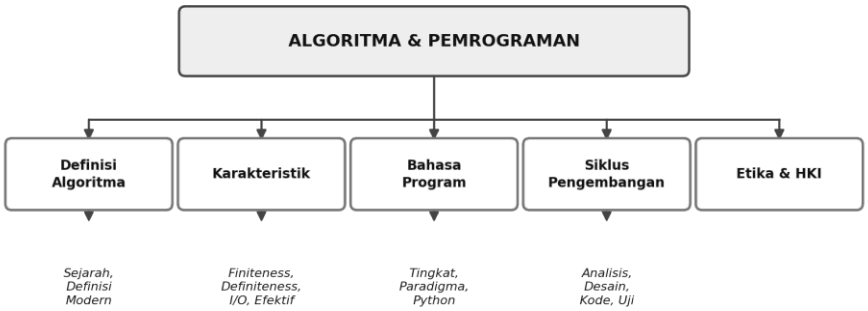
1.0 Capaian Pembelajaran Bab

Setelah mempelajari bab ini, mahasiswa diharapkan mampu mencapai sejumlah luaran pembelajaran sebagai berikut:

Kode	Capaian	Bloom
CPB-1.1	Menjelaskan definisi algoritma dan pemrograman beserta keterkaitannya	C2
CPB-1.2	Mengidentifikasi karakteristik algoritma yang baik	C2
CPB-1.3	Membedakan kategori bahasa pemrograman berdasarkan tingkat dan paradigmanya	C3
CPB-1.4	Menjelaskan tahap-tahap dalam siklus pengembangan program	C2
CPB-1.5	Menganalisis alasan pemilihan Python sebagai bahasa pengantar pembelajaran	C4
CPB-1.6	Menjelaskan prinsip dasar etika pemrograman dan hak kekayaan intelektual	C2

1.1 Peta Konsep

Peta konsep berikut menggambarkan keterkaitan antar topik utama yang akan dibahas dalam Bab 1.



1.2 Apersepsi

Bayangkan suatu pagi yang khas. Alarm berbunyi pada pukul lima. Anda meraih ponsel, menggeser layar untuk mematikannya, lalu beranjak ke dapur untuk menyeduh kopi. Anda mengisi panci dengan air, menyalakan kompor, menunggu air mendidih, menuangkannya ke gelas berisi serbuk kopi, mengaduk, dan duduk menikmati cangkir pertama hari itu. Tidak ada yang istimewa. Namun jika kita perhatikan baik-baik, Anda baru saja mengeksekusi sebuah **algoritma**, yaitu serangkaian langkah berurutan yang terbatas untuk mencapai tujuan tertentu: menyajikan kopi.

Manusia menjalani hidupnya dalam algoritma. Resep masakan adalah algoritma. Petunjuk arah dari titik A ke titik B adalah algoritma. Rangkaian senam pagi adalah algoritma. Bahkan cara kita mengantre di kasir mengikuti algoritma sederhana: datang, ambil nomor, tunggu giliran, bayar, terima struk. Yang membedakan algoritma dalam ilmu komputer dari rutinitas sehari-hari kita bukanlah esensinya, melainkan **siapa yang mengeksekusinya**: di laboratorium komputer, eksekutornya adalah mesin yang tidak dapat menebak, berimprovisasi, atau bertanya ulang.

Inilah inti dari pelajaran kita selama satu semester ke depan. Kita akan belajar bagaimana menerjemahkan keinginan manusia ke dalam langkah-langkah yang dapat dipahami oleh entitas yang sepenuhnya literal. Kita akan belajar

menjadi penerjemah antara dua dunia, dunia ide manusiawi yang sering kabur dan ambigu, dengan dunia mesin yang menuntut presisi absolut.

> **ASAH OTAK 1.1**
>
> Tuliskan algoritma "membuat mi instan" dalam delapan langkah atau kurang. Kemudian berikan kepada teman Anda dan minta ia mengeksekusinya secara harfiah, tanpa improvisasi. Apakah hasilnya sesuai harapan? Manakah langkah yang ternyata ambigu?

1.3 Uraian Materi

1.3.1 Definisi Algoritma: Dari Al-Khwarizmi hingga Era Digital

Kata *algoritma* berakar pada nama seorang ilmuwan muslim abad ke-9, **Muhammad ibn Musa al-Khwarizmi**, yang bekerja di Bait al-Hikmah (Rumah Kebijaksanaan) di Baghdad. Karyanya yang berjudul *Kitab al-Jabr wa al-Muqabala* — yang darinya kata *aljabar* lahir — memperkenalkan prosedur sistematis untuk menyelesaikan persamaan. Ketika karya tersebut diterjemahkan ke bahasa Latin, nama Al-Khwarizmi mengalami latinisasi menjadi *Algoritmi*, dan dari sanalah istilah modern kita berasal.

> **SEJARAH SINGKAT 1.1**
>
> Bait al-Hikmah pada masa Khalifah Al-Ma'mun (813–833 M) adalah pusat penerjemahan dan riset terbesar di dunia. Al-Khwarizmi bekerja berdampingan dengan astronom, matematikawan, dan filsuf dari berbagai bangsa. Banyak istilah ilmiah modern, seperti *algorithm*, *algebra*, dan bahkan beberapa nama bintang seperti *Aldebaran* dan *Altair*, merupakan warisan dari era ini.

Dalam ilmu komputer modern, sebuah algoritma didefinisikan secara formal sebagai berikut:

> **Algoritma** *adalah suatu urutan terbatas dari instruksi yang terdefinisi secara jelas dan tak ambigu, yang ketika dieksekusi*

> *pada masukan tertentu akan menghasilkan keluaran yang dikehendaki dalam waktu yang berhingga.*

Tiga frasa kunci dalam definisi ini patut dicermati. Pertama, **"terbatas"** berarti algoritma harus berakhir; tidak boleh berjalan selamanya. Kedua, **"terdefinisi jelas dan tak ambigu"** berarti setiap langkah harus memiliki satu interpretasi yang pasti. Ketiga, **"keluaran yang dikehendaki"** berarti algoritma memiliki tujuan yang dapat diverifikasi.

Donald Knuth, dalam karyanya yang monumental *The Art of Computer Programming*, menyatakan bahwa pemrograman pada hakikatnya adalah seni *"telling another human being what one wants the computer to do"*. Definisi ini menggeser fokus kita: program bukan sekadar instruksi untuk mesin, melainkan dokumen yang harus dipahami oleh manusia lain (termasuk diri kita sendiri di masa depan).

1.3.2 Karakteristik Algoritma yang Baik

Tidak setiap urutan langkah dapat disebut algoritma. Donald Knuth mengidentifikasi lima karakteristik fundamental yang harus dipenuhi:

- **Finiteness (keterbatasan).** Algoritma harus berakhir setelah sejumlah langkah berhingga. Tidak ada algoritma yang sah jika ia berjalan tanpa akhir. Sebagai contoh, "terus tambahkan satu pada bilangan, lalu berhenti ketika hasilnya negatif" bukan algoritma karena tidak akan pernah berhenti.
- **Definiteness (kepastian).** Setiap langkah harus didefinisikan secara presisi. Instruksi seperti "campurkan secukupnya" boleh dalam resep masakan, tetapi tidak boleh dalam algoritma komputer. Mesin tidak memiliki intuisi.
- **Input (masukan).** Algoritma menerima nol atau lebih masukan yang berasal dari himpunan objek yang terdefinisi.
- **Output (keluaran).** Algoritma menghasilkan setidaknya satu keluaran yang memiliki hubungan tertentu dengan masukan.
- **Effectiveness (keefektifan).** Setiap operasi harus cukup mendasar sehingga dapat dilakukan secara mekanis dalam waktu berhingga. Operasi seperti "temukan bilangan terkecil yang membuktikan hipotesis Riemann" tidak efektif, meskipun terdefinisi.

CATATAN 1.1

Beberapa literatur menambahkan kriteria keenam yaitu **generalitas**, yang berarti algoritma harus dapat diterapkan pada seluruh kelas masukan yang sah, bukan hanya satu kasus spesifik. Algoritma pengurutan yang baik harus bekerja untuk daftar bilangan apa pun, bukan hanya `[3, 1, 2]`.

1.3.3 Hubungan Algoritma, Program, dan Bahasa Pemrograman

Tiga konsep ini sering tertukar di kalangan pemula. Mari kita pisahkan dengan analogi sederhana.

Algoritma adalah resep masakan. Ia adalah ide yang tidak terikat pada bahasa tertentu. Resep nasi goreng tetaplah resep nasi goreng, baik ditulis dalam bahasa Indonesia, Inggris, ataupun Jepang.

Bahasa pemrograman adalah bahasa yang digunakan untuk menuliskan resep tersebut. Python, Java, C++, dan JavaScript adalah bahasa-bahasa berbeda yang dapat mengekspresikan resep yang sama.

Program adalah implementasi konkret dari resep dalam bahasa tertentu. Ia adalah artefak yang siap dieksekusi.

Dengan kata lain: *algoritma dipikirkan, bahasa adalah medium, program adalah hasilnya*.

DI BALIK LAYAR 1.1

Ketika Anda menulis `print("Halo")` dalam Python, mesin tidak benar-benar memahami tulisan tersebut. Yang terjadi adalah Python *interpreter* menerjemahkan kode Anda menjadi *bytecode* (instruksi tingkat menengah), yang kemudian dijalankan oleh Python Virtual Machine. Pada akhirnya, instruksi level mesin dalam bentuk nol dan satulah yang dieksekusi oleh prosesor. Algoritma yang Anda pikirkan menempuh perjalanan panjang sebelum benar-benar "terjadi" di silikon.

1.3.4 Klasifikasi Bahasa Pemrograman

Bahasa pemrograman dapat dikelompokkan berdasarkan **tingkat abstraksinya** dari mesin:

Bahasa Tingkat Rendah (Low-Level Languages). Bahasa yang dekat dengan representasi mesin. Termasuk di dalamnya bahasa mesin (kode biner) dan bahasa rakitan (*assembly*). Bahasa ini sangat cepat dan hemat sumber daya, tetapi sulit dibaca dan rawan kesalahan. Contoh penggunaan: pengembangan *firmware*, sistem tertanam (*embedded systems*), dan optimasi kritis.

Bahasa Tingkat Tinggi (High-Level Languages). Bahasa yang lebih dekat dengan bahasa manusia. Programmer tidak perlu memikirkan alamat memori atau register prosesor. Contoh: Python, Java, JavaScript, C++. Bahasa-bahasa inilah yang umumnya digunakan untuk pengembangan aplikasi modern.

> **WAWASAN INDUSTRI 1.1**
>
> Survei tahunan Stack Overflow Developer Survey secara konsisten menempatkan Python di lima besar bahasa paling banyak digunakan di kalangan pengembang profesional. Python juga merupakan bahasa yang dominan di sektor *data science* dan kecerdasan buatan, sebagaimana tercermin dari banyaknya pustaka komputasi numerik seperti NumPy, Pandas, dan TensorFlow yang berbasis Python. Penguasaan Python di tingkat dasar telah menjadi kompetensi yang banyak diharapkan dari lulusan ilmu komputer.

1.3.5 Paradigma Pemrograman

Paradigma pemrograman adalah *cara berpikir* tentang bagaimana suatu program disusun. Empat paradigma utama yang patut diketahui:

Pemrograman Prosedural memandang program sebagai serangkaian prosedur (atau fungsi) yang dieksekusi secara berurutan. Fokusnya adalah *bagaimana* sesuatu dilakukan, langkah demi langkah. Contoh: C, Pascal.

Pemrograman Berorientasi Objek (OOP) memandang dunia sebagai kumpulan objek yang berinteraksi. Setiap objek memiliki data (atribut) dan perilaku (metode). Contoh: Java, C++, Python.

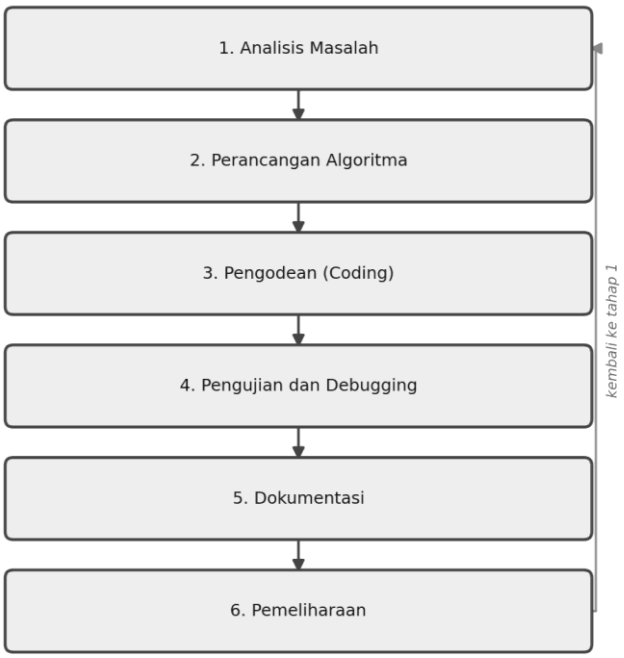
Pemrograman Fungsional memperlakukan komputasi sebagai evaluasi fungsi matematis dan menghindari perubahan keadaan (*state*). Contoh: Haskell, Lisp, dan elemen fungsional dalam Python.

Pemrograman Deklaratif fokus pada *apa* yang diinginkan, bukan *bagaimana* mendapatkannya. Contoh paling umum adalah SQL untuk basis data.

Python adalah bahasa **multi-paradigma**, yang berarti ia mendukung lebih dari satu paradigma. Di Buku 1 ini kita akan fokus pada paradigma prosedural sebagai fondasi; paradigma berorientasi objek akan dieksplorasi di Buku 2.

1.3.6 Siklus Pengembangan Program

Membuat program yang baik bukanlah aktivitas linier "duduk, ketik, selesai". Ia mengikuti siklus terstruktur yang disebut **Program Development Life Cycle (PDLC)**, terdiri atas enam tahap:



Pada tahap **analisis**, kita memahami masalah secara menyeluruh: apa masukannya, apa keluaran yang diharapkan, dan apa kendalanya. Tahap **perancangan** menghasilkan algoritma dalam bentuk flowchart atau pseudocode. Tahap **pengodean** menerjemahkan rancangan ke dalam bahasa

pemrograman. Tahap **pengujian** memverifikasi bahwa program bekerja sesuai harapan dan menemukan kesalahan (*bug*). Tahap **dokumentasi** menghasilkan catatan yang memudahkan pemeliharaan. Terakhir, tahap **pemeliharaan** menangani perbaikan dan pembaruan setelah program digunakan.

> **TIPS PRO 1.1**
>
> Pemula sering menghabiskan 80% waktu di tahap pengodean dan 20% di tahap-tahap lain. Pengembang berpengalaman justru kebalikannya: 60% waktunya untuk analisis dan perancangan, 20% untuk pengodean, dan 20% untuk pengujian serta dokumentasi. Bahasa pemrogramannya boleh berbeda, tetapi prinsip "berpikir dulu, mengetik kemudian" berlaku universal.

1.3.7 Posisi Python dalam Lanskap Modern

Python diciptakan oleh **Guido van Rossum** pada akhir 1980-an dan dirilis pertama kali tahun 1991. Nama "Python" terinspirasi bukan dari ular, melainkan dari grup komedi Inggris *Monty Python's Flying Circus*. Filosofinya tertuang dalam dokumen *The Zen of Python* karya Tim Peters, yang menempatkan keterbacaan kode sebagai nilai tertinggi.

Beberapa alasan mengapa Python ideal sebagai bahasa pertama:

- **Sintaks yang dekat dengan bahasa alami.** Program Python sering kali dapat dibaca seperti bahasa Inggris sederhana.
- **Komunitas dan ekosistem yang masif.** Lebih dari 400.000 paket tersedia di Python Package Index (PyPI).
- **Multi-platform.** Berjalan di Windows, macOS, Linux, bahkan mikrokontroler.
- **Domain aplikasi luas.** Mulai dari web (Django, FastAPI), data science (NumPy, Pandas), kecerdasan buatan (TensorFlow, PyTorch), hingga otomasi.
- **Adopsi industri yang kuat.** Google, Netflix, Instagram, Spotify, dan NASA menggunakan Python dalam sistem produksinya.

SEJARAH SINGKAT 1.2

Pada Desember 1989, Guido van Rossum sedang mencari "proyek hobi" untuk mengisi liburan Natal. Ia memutuskan menulis interpreter untuk bahasa baru yang ia rancang sendiri. Bahasa itu adalah Python. Tiga dekade kemudian, "proyek liburan" tersebut telah menggerakkan rekomendasi film di Netflix, mesin pencari di Google, dan teleskop James Webb di luar angkasa.

1.3.8 Etika Pemrograman dan Hak Kekayaan Intelektual

Kemampuan menulis program adalah kuasa yang signifikan. Sebuah baris kode dapat melindungi data pasien, atau sebaliknya, membocorkannya. Sebuah algoritma dapat mempercepat distribusi bantuan sosial, atau memperkuat diskriminasi yang tidak disengaja. Pemrograman bukanlah aktivitas yang netral nilai.

Beberapa prinsip etika pemrograman yang penting:

Kejujuran intelektual. Berikan atribusi pada kode yang Anda gunakan dari sumber lain. Mengklaim karya orang lain sebagai milik sendiri, baik dalam tugas kuliah maupun pekerjaan profesional, adalah plagiarisme.

Penghormatan terhadap privasi. Data pengguna bukan komoditas bebas. Pelajari prinsip seperti *data minimization* (kumpulkan hanya yang perlu) dan *purpose limitation* (gunakan hanya untuk tujuan yang disepakati).

Akuntabilitas. Program yang Anda tulis akan digunakan oleh manusia lain. Bertanggung jawablah atas kualitas dan dampaknya.

Lisensi perangkat lunak. Setiap perangkat lunak memiliki lisensi yang mengatur penggunaannya. Lisensi *open-source* seperti MIT, Apache 2.0, dan GPL memiliki ketentuan berbeda. Lisensi proprietary tidak mengizinkan distribusi ulang tanpa izin.

PERHATIAN 1.1

Menyalin kode dari Stack Overflow, ChatGPT, atau GitHub tanpa memahami dan tanpa atribusi yang sesuai dapat dikategorikan sebagai pelanggaran akademik. Setiap perguruan tinggi memiliki kebijakan

integritas akademik yang ketat. Selalu konsultasikan dengan dosen pengampu sebelum menggunakan kode dari sumber eksternal dalam tugas Anda.

1.4 Contoh Kode Lengkap

Meskipun bab ini bersifat konseptual, mari kita lihat satu contoh konkret yang mengilustrasikan **algoritma yang sama** ditulis dalam **representasi yang berbeda**. Permasalahannya: hitung luas persegi panjang berdasarkan panjang dan lebar yang dimasukkan pengguna.

Representasi 1 — Narasi

"Mulai. Minta pengguna memasukkan panjang. Minta pengguna memasukkan lebar. Kalikan panjang dengan lebar. Tampilkan hasil sebagai luas. Selesai."

Representasi 2 — Pseudocode (Gaya Wirth)

```
ALGORITMA LuasPersegiPanjang
DEKLARASI
    panjang, lebar, luas : real
DESKRIPSI
    read(panjang)
    read(lebar)
    luas ← panjang × lebar
    write("Luas =", luas)
```

Representasi 3 — Implementasi Python

```
# Program: Menghitung luas persegi panjang
# Penulis: [Nama Mahasiswa]
# Tujuan : Demonstrasi algoritma sederhana dalam Python

# Meminta masukan dari pengguna dan mengonversi ke tipe float
panjang = float(input("Masukkan panjang (cm): "))
lebar = float(input("Masukkan lebar (cm): "))
```

```
# Menghitung luas dengan operasi perkalian
luas = panjang * lebar

# Menampilkan hasil ke layar
print(f"Luas persegi panjang adalah {luas} cm persegi.")
```

Contoh eksekusi:

```
Masukkan panjang (cm): 5
Masukkan lebar (cm): 3
Luas persegi panjang adalah 15.0 cm persegi.
```

Perhatikan bahwa ketiga representasi mengekspresikan **algoritma yang sama**, namun dengan tingkat detail dan formalitas yang berbeda. Narasi paling mudah dipahami manusia awam, pseudocode menjadi jembatan menuju kode, sementara Python adalah implementasi yang dapat dieksekusi mesin.

> **DI BALIK LAYAR 1.2**
>
> Fungsi `input()` selalu mengembalikan nilai bertipe *string* (teks), bukan angka. Itulah sebabnya kita membungkusnya dengan `float()` untuk mengonversinya menjadi bilangan desimal. Jika Anda lupa konversi ini dan mencoba mengalikan dua string, Python akan memprotes. Konsep tipe data akan kita bahas tuntas di Bab 4.

1.5 Kesalahan Umum dan Cara Mengatasinya

Mahasiswa baru sering menghadapi miskonsepsi yang khas pada bab pengantar. Mari kita identifikasi:

Kesalahan 1: Menyamakan algoritma dengan kode.

Miskonsepsi: "Belajar algoritma berarti belajar Python."

Realita: Algoritma adalah ide independen dari bahasa. Algoritma yang sama dapat diimplementasikan dalam Python, Java, atau bahkan tanpa komputer

sama sekali. Latihlah diri Anda merancang algoritma di kertas sebelum menulis kode.

Kesalahan 2: Mengabaikan tahap analisis dan perancangan.

Miskonsepsi: "Langsung ngoding saja, kalau salah tinggal perbaiki."

Realita: Pendekatan ini efektif hanya untuk masalah yang sangat sederhana. Pada masalah yang lebih kompleks, ia menghasilkan kode yang berantakan dan sulit diperbaiki. Berinvestasilah pada perancangan.

Kesalahan 3: Mencari "satu bahasa terbaik".

Miskonsepsi: "Mana bahasa pemrograman paling bagus?"

Realita: Tidak ada bahasa universal yang terbaik. Setiap bahasa memiliki domain idealnya. Python unggul untuk pembelajaran dan analisis data, C untuk sistem operasi, JavaScript untuk web, Rust untuk performa tinggi yang aman. Yang penting adalah menguasai konsep yang transferable antar bahasa.

Kesalahan 4: Menyamakan menghafal sintaks dengan menguasai pemrograman.

Miskonsepsi: "Saya hafal semua perintah Python, jadi saya jago."

Realita: Sintaks dapat dicari di dokumentasi. Yang membedakan programmer pemula dari yang ahli adalah kemampuan memecahkan masalah. Fokuslah pada problem solving.

Kesalahan 5: Takut pada pesan kesalahan (error message).

Miskonsepsi: "Saya panik kalau lihat tulisan merah."

Realita: Pesan kesalahan adalah teman, bukan musuh. Ia memberitahu Anda apa yang salah dan di baris berapa. Belajarlah membaca dan menghargainya.

1.6 Rangkuman

Bab pengantar ini telah meletakkan fondasi konseptual yang akan kita bangun di atasnya selama satu semester ke depan. Beberapa pokok penting:

Algoritma adalah urutan terbatas instruksi yang terdefinisi jelas, menerima masukan, menghasilkan keluaran, dan efektif. Kelima karakteristik ini, yang dirumuskan Knuth, membedakan algoritma dari sekadar daftar langkah biasa.

Algoritma dapat direpresentasikan dalam narasi, pseudocode, atau flowchart, dan ia bersifat *bahasa-agnostik*. Implementasi algoritma dalam bahasa pemrograman tertentu menghasilkan apa yang kita sebut program.

Bahasa pemrograman terbagi atas tingkat rendah dan tinggi, serta dapat dikategorikan menurut paradigmanya. Python yang akan kita gunakan adalah bahasa tingkat tinggi multi-paradigma dengan filosofi keterbacaan. Pengembangan program yang baik mengikuti siklus PDLC, yang menekankan pentingnya tahap analisis dan perancangan sebelum pengodean.

Akhirnya, kekuasaan menulis kode datang bersama tanggung jawab etis: kejujuran intelektual, penghormatan terhadap privasi, akuntabilitas, dan kepatuhan terhadap lisensi adalah bagian tak terpisahkan dari profesi ini.

1.7 Latihan Terbimbing

Soal 1.7.1

Soal: Sebutkan lima karakteristik algoritma menurut Knuth dan berikan contoh pelanggaran untuk masing-masing.

Pembahasan:

1. Finiteness. Pelanggaran: “Tambahkan 1 pada x, ulangi terus” tanpa kondisi berhenti.

2. Definiteness. Pelanggaran: “Aduk secukupnya hingga terlihat bagus” — ambigu.

3. Input. Pelanggaran: prosedur yang seharusnya bergantung pada masukan namun tidak menerimanya sama sekali.

4. Output. Pelanggaran: program yang dijalankan tetapi tidak menghasilkan apa pun yang dapat diverifikasi.

5. Effectiveness. Pelanggaran: “Hitung digit ke-trilyun dari π dalam satu mikrodetik” — tidak praktis.

Soal 1.7.2

Soal: Sebuah resep menyatakan: “Goreng tahu hingga keemasan.” Identifikasi mengapa instruksi ini tidak memenuhi syarat sebagai langkah algoritmik dan tuliskan ulang dalam bentuk algoritmik.

Pembahasan: Instruksi tersebut melanggar prinsip definiteness karena "keemasan" adalah istilah subjektif. Versi algoritmik yang lebih baik: "Goreng tahu dalam minyak suhu 170°C selama 4 menit, balikkan setiap menit."

Soal 1.7.3

Soal: Susunlah algoritma dalam narasi untuk menukar isi dua gelas A dan B yang masing-masing berisi air dan susu.

Pembahasan:

1. Sediakan gelas ketiga (C) yang kosong.

2. Tuang seluruh isi gelas A ke gelas C.

3. Tuang seluruh isi gelas B ke gelas A.

4. Tuang seluruh isi gelas C ke gelas B.

Catatan: variabel sementara C di sini analog dengan variabel bantu dalam pemrograman ketika menukar nilai dua variabel.

Soal 1.7.4

Soal: Identifikasi paradigma yang digunakan dalam pernyataan-pernyataan berikut: (a) "Kelas Mahasiswa memiliki atribut nama dan NIM." (b) "Hitung total nilai dengan menjumlahkan elemen-elemen daftar secara berurutan." (c) "Pilih semua mahasiswa dengan IPK > 3.5."

Pembahasan:

(a) Berorientasi objek — fokus pada entitas dan atributnya.

(b) Prosedural — fokus pada langkah-langkah komputasi.

(c) Deklaratif — fokus pada apa yang diinginkan, bukan bagaimana mendapatkannya (gaya SQL).

Soal 1.7.5

Soal: Mengapa tahap dokumentasi dianggap penting dalam siklus pengembangan program, padahal program sudah berjalan?

Pembahasan: Dokumentasi penting karena: (1) program akan dibaca jauh lebih sering daripada ditulis; (2) tim pengembang dapat berganti; (3) Anda sendiri akan lupa logika program Anda dalam beberapa bulan; (4) dokumentasi

yang baik mengurangi bus factor (risiko jika anggota tim tertentu tidak tersedia); dan (5) dokumentasi adalah bagian dari kontrak antara pengembang dan pengguna.

1.8 Latihan Mandiri

- **(C2)** Jelaskan dengan kata-kata sendiri perbedaan antara algoritma dan program.
- **(C2)** Apa makna filosofis dari penggunaan nama Al-Khwarizmi sebagai akar kata algoritma?
- **(C3)** Berikan contoh masing-masing satu kegiatan sehari-hari yang sesuai dengan paradigma prosedural, OOP, fungsional, dan deklaratif.
- **(C3)** Tuliskan algoritma dalam pseudocode untuk menentukan apakah suatu bilangan bulat genap atau ganjil.
- **(C3)** Sebutkan tiga keunggulan dan dua kelemahan Python dibandingkan dengan bahasa pemrograman tingkat tinggi lainnya yang Anda ketahui.
- **(C4)** Analisislah resep pembuatan teh tarik berikut: "Seduh teh, tarik beberapa kali hingga berbuih, tambahkan susu, sajikan." Identifikasi pelanggaran terhadap karakteristik algoritma menurut Knuth.
- **(C4)** Bandingkan dan kontraskan pendekatan top-down dengan bottom-up dalam perancangan algoritma. Pada kasus seperti apa masing-masing pendekatan lebih efektif?
- **(C3)** Carilah satu lisensi open-source (MIT, Apache 2.0, atau GPL) dan rangkum tiga ketentuan utamanya.
- **(C4)** Studi kasus: sebuah perusahaan rintisan ingin membangun aplikasi mobile, sistem rekomendasi berbasis AI, dan dashboard analitik. Bahasa pemrograman apa yang akan Anda rekomendasikan untuk masing-masing komponen, dan mengapa?
- **(C4)** Diskusikan dilema etis berikut: seorang programmer diminta atasannya untuk membuat fitur yang melacak lokasi pengguna tanpa pemberitahuan eksplisit. Tindakan apa yang harus diambil dan mengapa?

1.9 Tugas Proyek Mini

"Algoritma di Sekitar Kita"

Identifikasilah satu aktivitas yang biasa Anda lakukan setiap hari (misalnya: berangkat kuliah, memesan makanan via aplikasi, mengisi pulsa, menarik uang di ATM). Susunlah laporan pendek berisi:

- Deskripsi naratif aktivitas (1 paragraf).
- Algoritma aktivitas tersebut dalam pseudocode (gaya Wirth).
- Flowchart aktivitas (dapat menggunakan draw.io, Lucidchart, atau gambar tangan kemudian difoto).
- Analisis: identifikasi setiap karakteristik Knuth dalam algoritma Anda dan tunjukkan di langkah mana karakteristik tersebut termanifestasi.
- Refleksi: temukan minimal satu langkah dalam algoritma Anda yang berpotensi ambigu jika dieksekusi oleh mesin, dan usulkan perbaikannya.

Format pengumpulan: dokumen PDF maksimal 4 halaman, font Times New Roman 12, spasi 1,15.

1.10 Refleksi Diri

Centang pernyataan berikut yang sudah Anda kuasai. Jika ada yang belum, kembalilah ke subbab terkait.

☐ Saya dapat menjelaskan definisi algoritma dalam kata-kata saya sendiri.

☐ Saya dapat menyebutkan dan menjelaskan lima karakteristik algoritma menurut Knuth.

☐ Saya memahami perbedaan antara algoritma, bahasa pemrograman, dan program.

☐ Saya dapat membedakan bahasa tingkat rendah dan tingkat tinggi.

☐ Saya memahami empat paradigma pemrograman utama dan dapat memberikan contoh masing-masing.

☐ Saya dapat menyebutkan dan menjelaskan enam tahap dalam PDLC.

☐ Saya memahami mengapa Python dipilih sebagai bahasa pembelajaran.

☐ Saya menyadari aspek etis dalam pemrograman dan implikasinya terhadap praktik akademik saya.

ASAH OTAK 1.2

Jika Anda diberi mesin yang dapat mengeksekusi algoritma apa pun, namun hanya satu kali seumur hidup Anda, algoritma apakah yang akan Anda berikan kepadanya? Apakah algoritma "kebahagiaan umat manusia" memenuhi karakteristik Knuth? Diskusikan dengan teman sekelas.

BAB 2

REPRESENTASI ALGORITMA: FLOWCHART DAN PSEUDOCODE

"Simplicity is the ultimate sophistication."
— Leonardo da Vinci, sebagaimana dikutip dalam tradisi rekayasa perangkat lunak

---

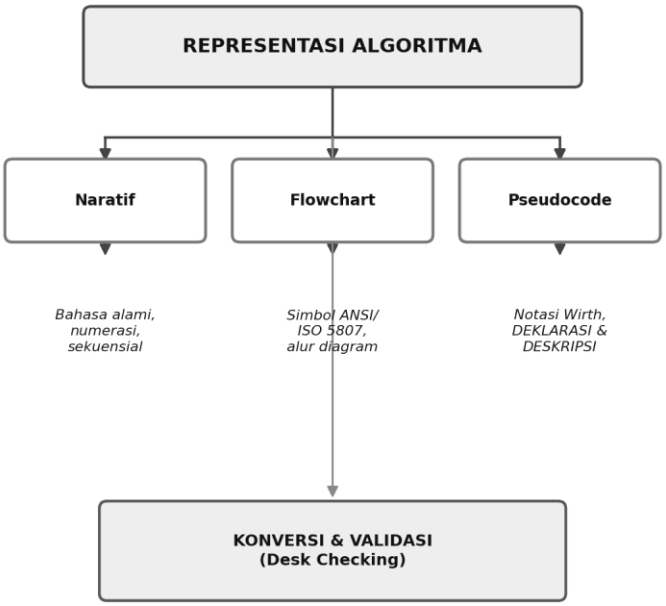
Capaian Pembelajaran Bab

Setelah mempelajari bab ini, mahasiswa diharapkan mampu mencapai sejumlah luaran pembelajaran sebagai berikut:

Kode	Capaian	Bloom
CPB-2.1	Menjelaskan pentingnya notasi algoritmik dan prinsip abstraksi	C2
CPB-2.2	Menerapkan representasi naratif untuk algoritma sederhana	C3
CPB-2.3	Membuat flowchart sesuai standar ANSI/ISO 5807	C3
CPB-2.4	Menyusun pseudocode bergaya Wirth yang disederhanakan	C3
CPB-2.5	Mengonversi algoritma antar tiga representasi (naratif, flowchart, pseudocode)	C4
CPB-2.6	Memvalidasi kebenaran algoritma melalui desk checking	C4

2.1 Peta Konsep

Peta konsep berikut menunjukkan keterkaitan antar topik utama dalam Bab 2, dengan tiga representasi algoritma sebagai inti pembahasan.



2.2 Apersepsi

Sebuah lagu dapat dinikmati lewat berbagai cara. Anda dapat mendengarnya langsung dari penyanyi di panggung, membaca liriknya di buku, atau menelaah partitur not baloknya di atas kertas. Ketiganya merujuk pada karya yang sama, namun dengan tingkat detail dan tujuan yang berbeda. Lirik memudahkan kita bernyanyi bersama; partitur memungkinkan musisi lain memainkan ulang lagu tersebut dengan presisi tinggi; rekaman audio adalah ekspresi hidup dari musik itu sendiri.

Algoritma pun demikian. Sebuah algoritma yang sama dapat dituangkan dalam beberapa bentuk: narasi sederhana untuk komunikasi awam, diagram alur (*flowchart*) untuk visualisasi terstruktur, atau pseudocode untuk jembatan menuju kode. Tugas kita di bab ini bukanlah memilih satu yang "terbaik",

melainkan menguasai ketiganya dan memahami kapan masing-masing paling tepat digunakan.

Mengapa kita perlu lebih dari satu representasi? Jawabannya berhubungan dengan **audiens dan tujuan**. Kepada klien non-teknis, narasi memudahkan diskusi. Kepada rekan satu tim, flowchart membantu menyepakati alur logika. Kepada programmer yang akan mengimplementasi, pseudocode menjadi spesifikasi yang akurat tanpa terikat pada bahasa pemrograman tertentu. Mahasiswa yang menguasai ketiga representasi memiliki perangkat berpikir yang lengkap.

> **ASAH OTAK 2.1**
>
> Pikirkan sebuah aktivitas yang biasa Anda lakukan setiap pagi. Cobalah jelaskan aktivitas tersebut kepada (a) seorang anak berusia lima tahun, (b) seorang teman yang harus menggantikan Anda esok hari, dan (c) seorang robot yang baru pertama kali diaktifkan. Apakah Anda menggunakan kata-kata yang sama untuk ketiganya? Mengapa tidak? Renungkan kaitannya dengan kebutuhan akan tiga representasi algoritma.

2.3 Uraian Materi

2.3.1 Notasi Algoritmik dan Pentingnya Abstraksi

Dalam ilmu komputer, **notasi algoritmik** adalah sistem simbol dan aturan yang digunakan untuk menyatakan algoritma secara jelas dan tidak ambigu, terlepas dari bahasa pemrograman yang akan dipakai. Notasi yang baik memenuhi tiga fungsi utama:

- **Komunikasi**. Memungkinkan ide algoritma disampaikan dari satu pikiran ke pikiran lain dengan akurasi tinggi.
- **Verifikasi**. Memungkinkan kebenaran algoritma diuji secara manual sebelum diimplementasikan.
- **Dokumentasi**. Memberikan rekam jejak pemikiran yang dapat ditinjau ulang di kemudian hari.

Inti dari notasi algoritmik adalah **abstraksi**, yaitu kemampuan untuk fokus pada inti masalah dengan menyembunyikan detail-detail tidak relevan. Ketika

kita menulis “urutkan daftar bilangan secara menaik”, kita melakukan abstraksi tingkat tinggi yang menyembunyikan ribuan kemungkinan implementasi (bubble sort, quick sort, merge sort, dan seterusnya). Pada tahap perancangan, abstraksi adalah teman; ia membebaskan kita dari beban detail yang dapat ditunda.

> **CATATAN 2.1**
>
> Konsep abstraksi adalah salah satu dari empat pilar berpikir komputasional (*computational thinking*) menurut Jeannette Wing, bersama dengan dekomposisi, pengenalan pola, dan rancangan algoritma. Kemampuan berpikir abstrak adalah pembeda utama antara programmer pemula dan ahli, dan ia dapat dilatih secara sengaja.

2.3.2 Representasi Naratif

Representasi naratif adalah bentuk paling sederhana dan paling lama dikenal manusia. Ia menggunakan bahasa alami (Indonesia, Inggris, atau yang lain) untuk menjelaskan langkah-langkah algoritma. Bentuknya umumnya berupa kalimat-kalimat bernomor.

Sebagai contoh, algoritma untuk menghitung luas lingkaran dapat dituliskan dalam narasi seperti berikut:

```
Algoritma Hitung Luas Lingkaran

1. Mulai.
2. Minta pengguna memasukkan nilai jari-jari.
3. Tetapkan nilai pi sebesar 3.14159.
4. Hitung luas dengan mengalikan pi dengan jari-jari kuadrat.
5. Tampilkan hasil luas ke layar.
6. Selesai.
```

Kelebihan representasi naratif:

- Mudah dipahami oleh siapa saja, termasuk yang tidak memiliki latar belakang pemrograman.
- Tidak memerlukan perangkat lunak khusus untuk membuatnya.

- Cocok untuk komunikasi awal dengan klien atau pemangku kepentingan.

Kekurangan representasi naratif:

- Rentan ambigu jika kata-kata yang digunakan tidak presisi.
- Sulit menampilkan alur percabangan dan perulangan yang kompleks.
- Untuk algoritma panjang, narasi menjadi sukar diikuti.

> **TIPS PRO 2.1**
>
> Ketika menulis narasi algoritma, gunakan kalimat aktif dan imperatif ("Hitung", "Bandingkan", "Tampilkan"), hindari kata yang berpotensi ganda makna ("mungkin", "biasanya", "sekitar"), dan satu kalimat sebaiknya menyatakan satu langkah saja. Praktik ini memudahkan konversi narasi ke representasi yang lebih formal.

2.3.3 Flowchart: Simbol Standar ANSI/ISO 5807

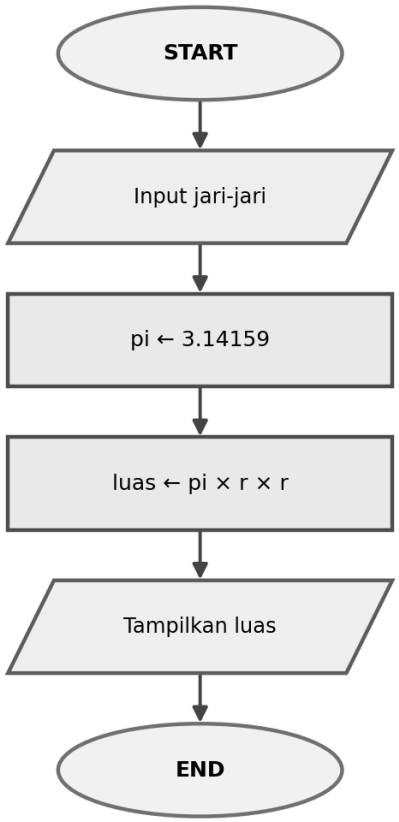
Flowchart (diagram alur) adalah representasi grafis dari algoritma yang menggunakan kotak-kotak terhubung untuk menunjukkan urutan langkah dan alur logika. Simbol-simbol yang digunakan distandardisasi oleh ANSI sejak 1970 dan diadopsi sebagai ISO 5807 pada tahun 1985.

Tujuh simbol utama yang wajib dikenali mahasiswa dirangkum dalam tabel berikut:

Simbol	Nama	Fungsi
(oval)	**Terminator**	Menandai titik awal (START) atau akhir (END) suatu alur.
[persegi panjang]	**Process / Proses**	Menyatakan satu langkah komputasi atau penugasan nilai.
< belah ketupat >	**Decision / Percabangan**	Menyatakan pengambilan keputusan dengan dua atau lebih alur keluar (Ya/Tidak).
/ jajaran genjang /	**Input / Output**	Menyatakan operasi pembacaan masukan atau penampilan keluaran.

Simbol	Nama	Fungsi
( ) bulat kecil	**Connector / Konektor**	Menghubungkan bagian-bagian flowchart yang terpisah pada halaman yang sama.
[[persegi panjang ganda]]	**Predefined Process**	Menyatakan pemanggilan sub-program atau prosedur yang didefinisikan tersendiri.
→ panah	**Flow Line**	Menunjukkan arah alir dari satu simbol ke simbol berikutnya.

Sebagai ilustrasi, flowchart untuk algoritma menghitung luas lingkaran yang telah kita lihat dalam bentuk naratif tampak seperti berikut (digambarkan dengan karakter ASCII; di buku ajar cetak biasanya digambar menggunakan perangkat seperti draw.io, Lucidchart, atau Microsoft Visio):



SEJARAH SINGKAT 2.1

Flowchart pertama kali diperkenalkan dalam ilmu manajemen oleh Frank dan Lillian Gilbreth pada 1921 untuk memodelkan proses kerja industri. Frank Gilbreth adalah perintis studi gerak dan waktu di pabrik, sedangkan Lillian Gilbreth dijuluki "Ibu Manajemen Modern". Pada 1947, Herman Goldstine dan John von Neumann mengadopsi notasi flowchart untuk merancang program komputer, dan sejak itu ia menjadi alat baku dalam ilmu komputer.

2.3.4 Aturan Penyusunan Flowchart yang Baik

Flowchart yang baik tidak hanya benar secara logika, tetapi juga mudah dibaca oleh orang lain. Berikut adalah aturan dan konvensi yang dipatuhi dalam praktik profesional:

- **Arah alir umum dari atas ke bawah dan dari kiri ke kanan.** Ini sesuai dengan kebiasaan membaca pada budaya kebanyakan dan memudahkan navigasi visual.
- **Setiap flowchart memiliki tepat satu simbol START.** Simbol END boleh lebih dari satu jika alur algoritma memang bercabang ke beberapa titik berhenti.
- **Simbol Decision memiliki tepat satu masukan dan minimal dua keluaran.** Setiap keluaran harus diberi label (umumnya "Ya" dan "Tidak", atau benar/salah).
- **Simbol Process memiliki satu masukan dan satu keluaran.** Hindari merangkap beberapa operasi dalam satu kotak proses; pisahkan menjadi beberapa kotak agar lebih jelas.
- **Gunakan konektor untuk diagram yang melebihi satu halaman.** Konektor diberi label berhuruf (A, B, C) yang sama pada titik keluar dan titik masuk.
- **Garis aliran tidak boleh berpotongan.** Jika tak terhindarkan, gunakan jembatan (loop kecil) atau ulang rancangan.
- **Beri label yang ringkas namun bermakna.** Hindari menulis kalimat panjang di dalam kotak; gunakan istilah atau frase pendek.

- **Tata letak harus rapi dan seimbang.** Jarak antar simbol konsisten, ukuran simbol seragam, dan tidak ada area kosong yang tidak perlu.

> **PERHATIAN 2.1**
>
> Kesalahan umum mahasiswa adalah menggambar simbol Decision dengan satu keluaran saja, atau lupa memberi label "Ya"/"Tidak" pada keluarannya. Flowchart semacam ini secara semantik tidak benar; pemeriksa atau rekan kerja akan kesulitan mengikuti alur logikanya. Selalu pastikan kedua cabang keputusan tergambar dengan jelas.

2.3.5 Pseudocode Gaya Wirth

Pseudocode adalah representasi algoritma yang menggabungkan kata-kunci mirip bahasa pemrograman dengan kalimat alami. Tujuannya adalah menjadi jembatan antara pemikiran abstrak dan kode aktual. Tidak ada satu pseudocode standar; tiap institusi atau buku ajar memiliki dialeknya sendiri. Dalam buku ini kita mengadopsi gaya **Niklaus Wirth** yang disederhanakan, yang banyak digunakan dalam kurikulum dasar pemrograman di Indonesia.

Struktur pseudocode gaya Wirth terdiri atas tiga bagian utama:

```
ALGORITMA NamaAlgoritma { judul dan deskripsi singkat }

DEKLARASI
    { deklarasi variabel dan konstanta }

DESKRIPSI
    { langkah-langkah algoritma }
```

Bagian DEKLARASI digunakan untuk menyatakan variabel beserta tipe datanya. Beberapa tipe dasar yang umum:

- `integer` untuk bilangan bulat (contoh: 1, -5, 100)
- `real` untuk bilangan riil/desimal (contoh: 3.14, -0.5, 2.71)
- `char` untuk satu karakter (contoh: 'A', '3')
- `string` untuk rangkaian karakter (contoh: "Halo")
- `boolean` untuk nilai benar/salah (true atau false)

Bagian DESKRIPSI berisi langkah-langkah operasional. Beberapa kata-kunci penting:

```
{ Penugasan nilai }
x ← 5
luas ← panjang * lebar

{ Masukan dan keluaran }
read(panjang)
write("Luas =", luas)

{ Percabangan }
if kondisi then
    aksi-1
else
    aksi-2
endif

{ Perulangan dengan kondisi }
while kondisi do
    aksi
endwhile

{ Perulangan dengan pencacah }
for i ← 1 to n do
    aksi
endfor
```

Mari kita lihat contoh pseudocode lengkap untuk algoritma luas lingkaran yang sebelumnya kita lihat dalam bentuk naratif dan flowchart:

```
ALGORITMA HitungLuasLingkaran
{ Menghitung luas lingkaran berdasarkan jari-jari masukan }

DEKLARASI
    jari_jari, luas : real
    pi : real = 3.14159

DESKRIPSI
    read(jari_jari)
    luas ← pi * jari_jari * jari_jari
    write("Luas lingkaran =", luas)
```

DI BALIK LAYAR 2.1

Pseudocode tidak dieksekusi oleh komputer; ia adalah dokumen perancangan. Namun, karena strukturnya yang mirip kode, mengonversinya menjadi Python, Java, atau C++ menjadi pekerjaan yang nyaris mekanis. Inilah kekuatan pseudocode: ia adalah "spesifikasi yang dapat diterjemahkan". Beberapa alat modern seperti GitHub Copilot bahkan dapat mengubah pseudocode menjadi kode siap pakai secara otomatis.

2.3.6 Konversi Antar Representasi

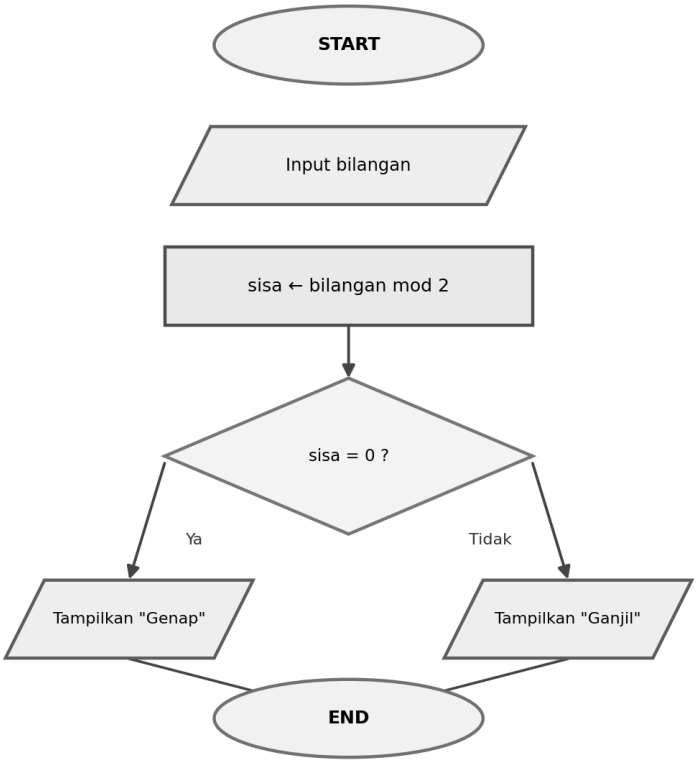
Kemampuan mengonversi antar ketiga representasi adalah keterampilan utama yang akan diuji dalam mata kuliah ini. Mari kita lakukan latihan konversi pada satu kasus konkret: menentukan apakah sebuah bilangan bulat genap atau ganjil.

Bentuk 1: Naratif

```
Algoritma Genap Ganjil

1. Mulai.
2. Minta pengguna memasukkan sebuah bilangan bulat.
3. Bagi bilangan tersebut dengan 2 dan ambil sisa
pembagiannya.
4. Jika sisanya 0, tampilkan "Bilangan genap".
5. Jika sisanya bukan 0, tampilkan "Bilangan ganjil".
6. Selesai.
```

Bentuk 2: Flowchart



Bentuk 3: Pseudocode

```
ALGORITMA GenapGanjil
{ Menentukan apakah bilangan bulat genap atau ganjil }

DEKLARASI
    bilangan, sisa : integer

DESKRIPSI
    read(bilangan)
    sisa ← bilangan mod 2
    if sisa = 0 then
        write("Bilangan genap")
    else
        write("Bilangan ganjil")
    endif
```

Perhatikan bahwa ketiga representasi menyatakan algoritma yang sama, namun **tingkat detailnya bertambah** seiring kita berpindah dari naratif ke pseudocode. Naratif menggunakan "bagi dengan 2 dan ambil sisanya"; flowchart menggunakan `mod` sebagai operator; dan pseudocode mendeklarasikan tipe data eksplisit untuk variabel `bilangan` dan `sisa`.

2.3.7 Studi Kasus: Algoritma Sehari-hari

Untuk memperkuat pemahaman, mari kita amati tiga studi kasus algoritma yang akrab dalam kehidupan sehari-hari.

Studi Kasus 1: Transaksi Tarik Tunai di ATM

```
ALGORITMA TarikTunaiATM
{ Algoritma sederhana penarikan uang di ATM }

DEKLARASI
    pin_input, pin_benar : integer
    jumlah, saldo : integer

DESKRIPSI
    write("Masukkan PIN:")
    read(pin_input)
    if pin_input = pin_benar then
        write("Masukkan jumlah penarikan:")
        read(jumlah)
        if jumlah ≤ saldo then
            saldo ← saldo - jumlah
            write("Ambil uang Anda.")
        else
            write("Saldo tidak mencukupi.")
        endif
    else
        write("PIN salah.")
    endif
```

Studi Kasus 2: Pengaturan Lampu Lalu Lintas Sederhana

```
ALGORITMA LampuLaluLintas
{ Siklus lampu lalu lintas tiga warna }

DEKLARASI
    siklus : integer

DESKRIPSI
    for siklus ← 1 to N do
        nyalakan(merah)   { 30 detik }
        tunggu(30)
        matikan(merah)
        nyalakan(hijau)   { 25 detik }
        tunggu(25)
        matikan(hijau)
        nyalakan(kuning)  { 5 detik }
        tunggu(5)
        matikan(kuning)
    endfor
```

Studi Kasus 3: Membuat Secangkir Kopi Tubruk

```
ALGORITMA BuatKopiTubruk
{ Algoritma penyajian kopi tubruk untuk satu cangkir }

DEKLARASI
    air_panas_siap : boolean
    suhu_air : integer { dalam derajat Celsius }

DESKRIPSI
    isi_panci(air, 250)            { 250 ml air }
    nyalakan(kompor)
    air_panas_siap ← false
    while NOT air_panas_siap do
        ukur(suhu_air)
        if suhu_air ≥ 95 then
            air_panas_siap ← true
        endif
    endwhile
    matikan(kompor)
    masukkan(kopi_bubuk, 2)        { 2 sendok }
```

```
    masukkan(gula, 1)              { sesuai selera }
    tuangkan(air_panas)
    aduk(15)                       { 15 detik }
    write("Kopi siap disajikan.")
```

WAWASAN INDUSTRI 2.1

Di industri perangkat lunak, flowchart dan pseudocode tetap relevan meskipun tools modern seperti UML, BPMN, dan diagram arsitektur cloud telah berkembang pesat. Tim DevOps menggunakan flowchart untuk memodelkan alur *CI/CD pipeline*. Tim AI/ML menggunakan diagram alur untuk merancang *training pipeline*. Bahkan untuk dokumen RFC (*Request for Comments*) protokol internet, pseudocode masih menjadi cara baku menyatakan algoritma. Penguasaan dasar di bab ini adalah modal yang akan terus terpakai sepanjang karier Anda.

2.3.8 Validasi Kebenaran Algoritma melalui Desk Checking

Desk checking, atau yang dalam bahasa Indonesia dapat disebut **telusur manual**, adalah teknik untuk menguji kebenaran algoritma secara mental atau di atas kertas, sebelum algoritma tersebut diimplementasikan menjadi kode. Caranya: kita simulasikan eksekusi algoritma langkah demi langkah, dengan mencatat nilai-nilai variabel dalam sebuah tabel.

Sebagai contoh, mari kita validasi algoritma berikut untuk menjumlahkan bilangan 1 hingga 5:

```
ALGORITMA JumlahLima
DEKLARASI
    i, total : integer
DESKRIPSI
    total ← 0
    for i ← 1 to 5 do
        total ← total + i
    endfor
    write(total)
```

Tabel telusur (*trace table*) untuk algoritma di atas:

Iterasi	i	total (sebelum)	Operasi	total (sesudah)
1	1	0	0 + 1	1
2	2	1	1 + 2	3
3	3	3	3 + 3	6
4	4	6	6 + 4	10
5	5	10	10 + 5	15

Setelah lima iterasi, nilai `total` adalah 15. Kita dapat memverifikasi dengan rumus matematis: 1+2+3+4+5 = 15. Algoritma terbukti benar untuk kasus ini. Tentu, pembuktian benar untuk satu kasus tidak menjamin kebenaran untuk semua kasus; namun *desk checking* membantu menemukan bug logika yang paling jelas tanpa harus menulis kode.

> **TIPS PRO 2.2**
>
> Biasakan diri melakukan desk checking dengan tiga kategori masukan: **kasus normal** (nilai khas yang diharapkan), **kasus batas** (nilai minimum atau maksimum), dan **kasus istimewa** (nol, negatif, masukan kosong). Programmer profesional menyebut pendekatan ini sebagai *boundary testing*, dan ia menyelamatkan ribuan baris kode dari kegagalan di lapangan.

2.4 Contoh Kode Lengkap

Sebagai penutup pembahasan, mari kita lihat bagaimana algoritma yang sudah dirancang dengan rapi diimplementasikan ke dalam Python. Kita akan menggunakan algoritma genap-ganjil dari Subbab 2.3.6.

```
# Program: Pengecek Genap atau Ganjil
# Penulis: [Nama Mahasiswa]
# Tujuan : Implementasi pseudocode dari Bab 2 ke Python
```

```
# Membaca masukan dari pengguna dan mengonversi ke integer
bilangan = int(input("Masukkan sebuah bilangan bulat: "))

# Menghitung sisa pembagian dengan 2
sisa = bilangan % 2

# Mengambil keputusan berdasarkan sisa
if sisa == 0:
    print("Bilangan genap.")
else:
    print("Bilangan ganjil.")
```

Contoh eksekusi pertama:

```
Masukkan sebuah bilangan bulat: 14
Bilangan genap.
```

Contoh eksekusi kedua:

```
Masukkan sebuah bilangan bulat: -7
Bilangan ganjil.
```

Perhatikan kemiripan langsung antara pseudocode dengan kode Python: kata kunci `if`, `else`, operator modulo `%`, dan struktur masukan/keluaran semuanya mempertahankan logikanya. Inilah sebabnya pseudocode menjadi alat perancangan yang sangat efektif sebelum menulis kode aktual.

> **DI BALIK LAYAR 2.2**
>
> Operator `%` dalam Python disebut *modulo*, yang mengembalikan sisa pembagian. Operasi ini banyak dipakai dalam algoritma: deteksi kelipatan, pengulangan siklik (mod jam, mod hari), hashing, dan kriptografi sederhana. Pemahaman intuitif tentang modulo akan menjadi modal penting di bab-bab selanjutnya.

2.5 Kesalahan Umum dan Cara Mengatasinya

Berikut adalah miskonsepsi dan kesalahan yang paling sering ditemui mahasiswa di bab ini:

Kesalahan 1: Mencampur simbol Process dan Decision.

Miskonsepsi: "Kotak persegi panjang bisa untuk apa saja, termasuk pengambilan keputusan."

Realita: Simbol persegi panjang khusus untuk proses komputasi atau penugasan; pengambilan keputusan wajib menggunakan belah ketupat. Pencampuran simbol membuat flowchart sulit dipahami dan secara semantik salah.

Kesalahan 2: Lupa kondisi berhenti pada perulangan.

Miskonsepsi: "Yang penting ada loop, kondisinya bisa nanti."

Realita: Perulangan tanpa kondisi berhenti yang jelas akan menjadi

infinite loop, melanggar prinsip *finiteness* algoritma. Selalu pastikan ada variabel pencacah atau kondisi logis yang dijamin akan menjadi false pada suatu titik.

Kesalahan 3: Pseudocode terlalu mirip kode.

Miskonsepsi: "Kalau pseudocode-nya pakai sintaks Python lengkap, lebih bagus."

Realita: Pseudocode justru harus bahasa-agnostik. Menulis pseudocode dengan tipe data spesifik bahasa tertentu (seperti `int main()` dari C, atau `public static void` dari Java) menghilangkan tujuannya sebagai notasi netral. Pertahankan abstraksi pada level konseptual.

Kesalahan 4: Mengabaikan kasus batas dalam desk checking.

Miskonsepsi: "Kalau algoritma bekerja untuk satu contoh, berarti benar."

Realita: Bug paling sering muncul pada kasus batas: nilai nol, negatif, kosong, atau maksimum. Selalu uji dengan setidaknya tiga jenis masukan: normal, batas, dan istimewa.

Kesalahan 5: Flowchart yang sukar dibaca karena tata letak buruk.

Miskonsepsi: "Yang penting alurnya benar, tampilan tidak masalah."

Realita: Flowchart adalah alat komunikasi. Tata letak yang berantakan, garis bersilangan, atau ukuran simbol tidak konsisten menghilangkan fungsi utamanya. Investasikan waktu untuk merapikan diagram, terutama yang akan dipresentasikan.

2.6 Rangkuman

Bab 2 telah memperkenalkan tiga representasi utama algoritma. Notasi algoritmik adalah perangkat untuk komunikasi, verifikasi, dan dokumentasi, dengan abstraksi sebagai prinsip dasarnya.

Representasi naratif menggunakan bahasa alami dengan numerasi langkah, cocok untuk komunikasi dengan audiens non-teknis namun rentan ambigu pada algoritma kompleks. Flowchart menggunakan tujuh simbol standar ANSI/ISO 5807, memberikan visualisasi yang kuat untuk alur logika, percabangan, dan perulangan. Aturan penyusunannya menekankan alir atas-bawah, satu titik start, label yang jelas pada decision, dan tata letak yang rapi. Pseudocode gaya Wirth menyediakan struktur ALGORITMA-DEKLARASI-DESKRIPSI dengan kata kunci yang menyerupai kode aktual namun tetap bahasa-agnostik.

Ketiga representasi saling melengkapi: naratif untuk pemahaman awal, flowchart untuk visualisasi, pseudocode untuk persiapan implementasi. Kemampuan mengonversi antar ketiganya dan memvalidasi algoritma melalui *desk checking* adalah keterampilan fundamental yang akan terus digunakan sepanjang mata kuliah ini dan karier Anda sebagai programmer.

2.7 Latihan Terbimbing

Soal 2.7.1

Soal: Sebutkan tujuh simbol flowchart standar ANSI/ISO 5807 dan jelaskan fungsi masing-masing dalam satu kalimat.

Pembahasan:

1. Terminator (oval): menandai titik awal atau akhir alur.

2. Process (persegi panjang): menyatakan langkah komputasi atau penugasan.

3. Decision (belah ketupat): menyatakan pengambilan keputusan dengan dua atau lebih cabang.

4. Input/Output (jajaran genjang): menyatakan operasi masukan atau keluaran.

5. Connector (lingkaran kecil): menghubungkan bagian-bagian terpisah dari flowchart.

6. Predefined Process (persegi panjang ganda): memanggil sub-program atau prosedur yang sudah didefinisikan.

7. Flow Line (panah): menunjukkan arah alir antar simbol.

Soal 2.7.2

Soal: Susunlah pseudocode untuk algoritma menghitung rata-rata tiga bilangan yang dimasukkan pengguna.

Pembahasan:

```
ALGORITMA RataRataTiga
{ Menghitung rata-rata tiga bilangan }

DEKLARASI
    a, b, c, rata : real

DESKRIPSI
    read(a, b, c)
    rata ← (a + b + c) / 3
    write("Rata-rata =", rata)
```

Soal 2.7.3

Soal: Lakukan desk checking terhadap pseudocode berikut untuk masukan n = 4, dan tentukan nilai akhir variabel hasil.

Pembahasan:

Pseudocode: hasil ← 1; for i ← 1 to n do hasil ← hasil * i; endfor

Tabel telusur:

Iterasi 1: i=1, hasil sebelum=1, hasil sesudah=1*1=1

Iterasi 2: i=2, hasil sebelum=1, hasil sesudah=1*2=2

Iterasi 3: i=3, hasil sebelum=2, hasil sesudah=2*3=6

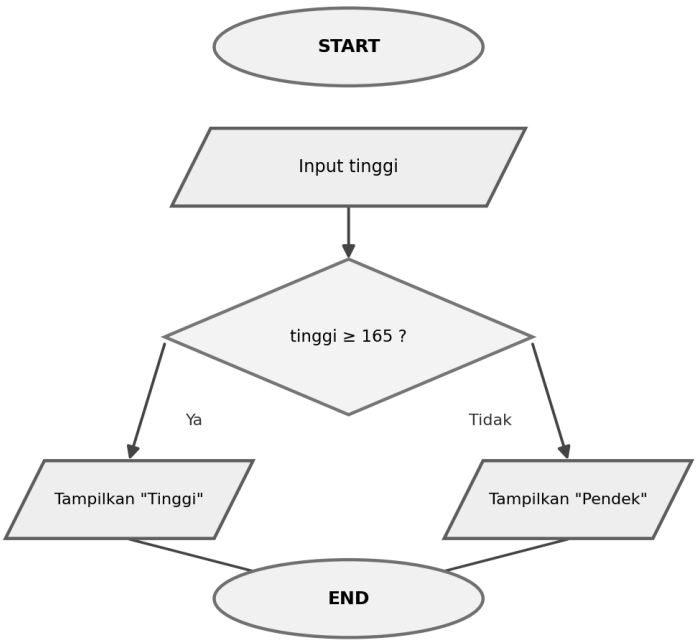
Iterasi 4: i=4, hasil sebelum=6, hasil sesudah=6*4=24

Nilai akhir hasil = 24 (yaitu 4 faktorial, atau 4!).

Soal 2.7.4

Soal: Konversilah narasi berikut ke flowchart (gambarkan dengan karakter ASCII): “Minta tinggi badan pengguna. Jika tinggi ≥ 165 cm tampilkan ‘Tinggi’, sebaliknya tampilkan ‘Pendek’.”

Pembahasan:



Soal 2.7.5

Soal: Jelaskan mengapa pseudocode tidak menggantikan flowchart secara mutlak, meskipun pseudocode lebih dekat ke kode pemrograman.

Pembahasan: Pseudocode dan flowchart memiliki kekuatan komplementer. Pseudocode unggul dalam menampilkan logika algoritma secara linier dan mudah dikonversi ke kode, namun ia kurang efektif menunjukkan alur percabangan yang kompleks secara visual. Flowchart, sebaliknya, sangat kuat sebagai alat komunikasi visual yang mudah dipahami audiens non-teknis dan dengan cepat menunjukkan struktur alur logika secara keseluruhan. Dalam praktik profesional, banyak dokumen perancangan menggabungkan keduanya: flowchart untuk gambaran besar, pseudocode untuk rincian implementasi.

2.8 Latihan Mandiri

- **(C2)** Jelaskan perbedaan mendasar antara representasi naratif, flowchart, dan pseudocode dalam minimal tiga aspek.
- **(C2)** Apa yang dimaksud dengan abstraksi dalam konteks notasi algoritmik? Berikan satu contoh konkret.
- **(C3)** Susunlah algoritma dalam ketiga representasi (naratif, flowchart, pseudocode) untuk menghitung keliling persegi panjang.
- **(C3)** Buatlah pseudocode untuk menentukan bilangan terbesar dari tiga bilangan yang dimasukkan pengguna.
- **(C3)** Gambarkan flowchart untuk algoritma menghitung total belanja dengan diskon: jika total ≥ Rp200.000 berikan diskon 10%, sebaliknya tidak ada diskon.
- **(C3)** Konversilah pseudocode perulangan `for i ← 1 to 10 do write(i*i) endfor` menjadi representasi naratif dan flowchart.
- **(C4)** Lakukan desk checking terhadap algoritma berikut untuk masukan n = 5: "x ← 0; for i ← 1 to n do if i mod 2 = 0 then x ← x + i endfor; write(x)". Jelaskan hasil akhirnya.
- **(C4)** Analisis flowchart yang Anda terima dari rekan kelas (atau buat satu yang sengaja salah). Identifikasi minimal tiga pelanggaran aturan penyusunan flowchart yang baik.
- **(C4)** Bandingkan efektivitas ketiga representasi untuk algoritma yang melibatkan banyak percabangan bersarang (nested if). Mana yang paling jelas dan mengapa?

- **(C4)** Sebuah algoritma menyatakan: "Terima bilangan n. Selama n > 1, jika n genap bagi dengan 2, jika ganjil kalikan 3 lalu tambah 1. Cetak setiap nilai n." (Dikenal sebagai konjektur Collatz.) Buatlah pseudocode-nya, lakukan desk checking untuk n = 7, dan diskusikan apakah algoritma ini memenuhi prinsip finiteness Knuth.

2.9 Tugas Proyek Mini

"Dari Resep ke Algoritma"

Pilihlah satu resep masakan yang Anda kenal (boleh resep keluarga, resep yang Anda temukan di internet, atau resep masakan khas daerah). Tugas Anda adalah "menerjemahkan" resep tersebut menjadi algoritma formal dalam tiga representasi:

- Tulis ulang resep dalam bentuk narasi bernomor yang algoritmik (bebas dari kata-kata ambigu seperti "secukupnya" atau "sampai matang").
- Buat flowchart dari resep tersebut menggunakan perangkat seperti draw.io, Lucidchart, atau Microsoft Visio. Pastikan menggunakan simbol standar dan mengikuti aturan penyusunan yang baik.
- Susun pseudocode gaya Wirth untuk resep tersebut, lengkap dengan bagian DEKLARASI yang menyatakan bahan-bahan beserta takarannya sebagai variabel.
- Lakukan desk checking dengan menelusuri pseudocode Anda dan catat di tabel mana saja kondisi/perulangan terjadi.
- Tulis refleksi singkat: bagian mana dari resep asli yang paling sulit diterjemahkan ke bentuk algoritmik? Mengapa?

Format pengumpulan: dokumen PDF maksimal 6 halaman, font Times New Roman 12, spasi 1,15. Sertakan tangkapan layar atau scan flowchart dengan resolusi yang jelas terbaca.

2.10 Refleksi Diri

Centang pernyataan berikut yang sudah Anda kuasai. Jika ada yang belum, kembalilah ke subbab terkait.

☐ Saya dapat menjelaskan tiga representasi algoritma dan kapan masing-masing digunakan.

☐ Saya hafal tujuh simbol flowchart standar ANSI/ISO 5807 beserta fungsinya.

☐ Saya dapat menyusun flowchart yang mengikuti aturan penyusunan yang baik.

☐ Saya menguasai struktur dan kata kunci pseudocode gaya Wirth.

☐ Saya dapat mengonversi suatu algoritma dari satu representasi ke representasi lain.

☐ Saya memahami konsep dan teknik desk checking dengan tabel telusur.

☐ Saya menyadari pentingnya kasus batas dalam pengujian algoritma.

☐ Saya dapat membedakan pseudocode yang baik dari yang terlalu bahasa-spesifik.

ASAH OTAK 2.2

Jika seseorang dapat dengan sempurna menjelaskan suatu algoritma dalam ketiga representasi, apakah berarti ia sudah "menguasai" algoritma itu? Atau apakah penguasaan sejati baru tercapai ketika ia dapat mengimplementasikannya dalam minimal dua bahasa pemrograman yang berbeda? Diskusikan dengan teman-teman Anda, dan kaitkan dengan pengalaman Anda sendiri dalam mempelajari konsep matematika di sekolah.

BAB 3

PENGENALAN PYTHON DAN LINGKUNGAN PENGEMBANGAN

"Code is read much more often than it is written."
— Guido van Rossum, pencipta Python

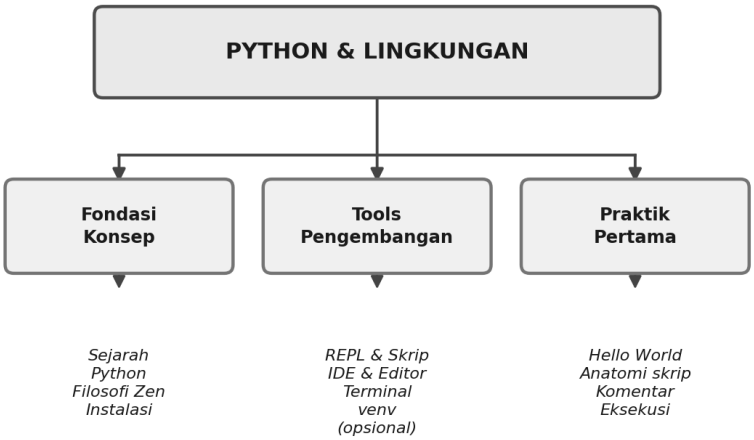
Capaian Pembelajaran Bab

Setelah mempelajari bab ini, mahasiswa diharapkan mampu mencapai sejumlah luaran pembelajaran sebagai berikut:

Kode	Capaian	Bloom
CPB-3.1	Menjelaskan sejarah singkat dan filosofi bahasa Python	C2
CPB-3.2	Menginstal Python pada sistem operasi yang digunakan	C3
CPB-3.3	Membedakan mode interaktif (REPL) dan mode skrip	C2
CPB-3.4	Mengidentifikasi IDE/editor yang sesuai dengan kebutuhan	C3
CPB-3.5	Menulis dan menjalankan program Python pertama (Hello, World!)	C3
CPB-3.6	Mengenali anatomi dasar skrip Python (statement, ekspresi, indentasi)	C2
CPB-3.7	Menerapkan praktik dokumentasi kode menggunakan komentar	C3
CPB-3.8	Menjalankan skrip Python melalui terminal maupun IDE	C3

3.1 Peta Konsep

Peta konsep berikut menggambarkan keterkaitan antar topik dalam Bab 3, mulai dari pengenalan Python hingga eksekusi skrip pertama.



3.2 Apersepsi

Setiap pengrajin memerlukan ruang kerja yang memadai. Seorang tukang kayu memiliki bengkel dengan gergaji, palu, dan meja kerja yang tertata rapi. Seorang koki memiliki dapur dengan kompor, pisau, dan timbangan. Seorang pelukis memiliki studio dengan kanvas, kuas, dan palet warna. Tanpa ruang kerja yang siap, bahkan keahlian yang paling tinggi pun sulit dimanifestasikan.

Pemrograman tidak berbeda. Sebelum kita dapat menulis baris kode pertama, kita membutuhkan sebuah **lingkungan pengembangan** yang berfungsi: interpreter Python yang terpasang, editor untuk menulis kode, dan terminal untuk mengeksekusi program. Bab ini akan memandu Anda menyiapkan ruang kerja digital tersebut, dari nol hingga mampu menjalankan program Python pertama Anda.

Materi bab ini bersifat sangat praktis. Pada akhir bab, Anda akan memiliki: Python yang terpasang pada komputer Anda, satu IDE yang Anda kuasai dasar-dasarnya, dan kemampuan menjalankan skrip pendek baik melalui terminal

maupun antarmuka grafis. Modal inilah yang akan kita pakai untuk seluruh perjalanan satu semester ke depan.

> **ASAH OTAK 3.1**
>
> Sebelum melanjutkan ke pembahasan teknis, renungkan: mengapa para perancang bahasa pemrograman repot-repot membuat sintaks yang "mudah dibaca"? Bukankah komputer akan sama-sama mengeksekusinya, baik sintaksnya mirip bahasa Inggris ataupun mirip simbol matematika? Diskusikan jawaban Anda dengan teman setelah membaca Subbab 3.3.1.

3.3 Uraian Materi

3.3.1 Sejarah dan Filosofi Python

Python dirancang oleh **Guido van Rossum** seorang programmer Belanda yang bekerja di Centrum Wiskunde & Informatica (CWI) di Amsterdam. Pengembangannya dimulai pada akhir 1980-an sebagai proyek pribadi untuk mengisi waktu liburan Natal, dan versi 0.9.0 dirilis ke publik pada Februari 1991. Sejak saat itu, Python berkembang menjadi salah satu bahasa pemrograman paling banyak digunakan di dunia.

Nama "Python" bukan berasal dari ular, melainkan dari grup komedi Inggris *Monty Python's Flying Circus*, yang menjadi favorit Guido. Pilihan nama ini mencerminkan filosofi yang melekat sampai sekarang: Python harus menyenangkan untuk digunakan.

> **SEJARAH SINGKAT 3.1**
>
> Guido van Rossum mendapat julukan *BDFL* (*Benevolent Dictator For Life*) oleh komunitas Python, karena ia berperan sebagai pemegang keputusan akhir dalam evolusi bahasa selama hampir tiga dekade. Pada Juli 2018, ia mengundurkan diri dari posisi BDFL, dan kini Python dikelola oleh *Python Steering Council*, sebuah dewan terpilih yang berisi lima orang anggota inti komunitas.

Filosofi yang menjadi panduan desain Python dirangkum dalam dokumen pendek berjudul *The Zen of Python* yang ditulis oleh **Tim Peters** pada tahun 1999 (kemudian distandardisasi sebagai PEP 20). Dokumen ini terdiri atas 19 aforisme yang dapat ditampilkan dengan menjalankan perintah `import this` di Python. Beberapa aforisme penting:

```
Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Readability counts.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good
idea.
```

Tiga prinsip yang paling sering dikutip adalah: **keindahan, eksplisit, dan keterbacaan**. Inilah yang membedakan Python dari bahasa lain. Python tidak hanya peduli pada apakah kode dapat dijalankan, tetapi juga pada apakah kode tersebut mudah dibaca dan dipahami oleh manusia lain di kemudian hari.

3.3.2 Instalasi Python

Instalasi Python berbeda sedikit untuk setiap sistem operasi. Pada saat buku ini ditulis, versi stabil yang direkomendasikan adalah **Python 3.12 atau lebih baru**. Hindari versi Python 2.x yang sudah tidak didukung sejak Januari 2020.

Instalasi pada Windows

- Kunjungi situs resmi `https://www.python.org/downloads/` dan unduh installer terbaru untuk Windows.
- Jalankan installer. **Centang opsi "Add Python to PATH"** sebelum mengklik Install Now. Langkah ini krusial agar perintah `python` dapat dipanggil dari Command Prompt mana pun.
- Verifikasi instalasi dengan membuka Command Prompt dan menjalankan:

```
python --version
```

Jika instalasi berhasil, Anda akan melihat keluaran seperti `Python 3.12.4` (versi spesifik bergantung pada saat unduhan).

Instalasi pada macOS

macOS sudah memiliki versi Python bawaan, namun seringkali sudah usang. Cara yang direkomendasikan adalah menggunakan **Homebrew**:

```
# Instal Homebrew jika belum ada
/bin/bash -c "$(curl -fsSL
https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"

# Instal Python
brew install python

# Verifikasi
python3 --version
```

Instalasi pada Linux (Ubuntu/Debian)

Sebagian besar distribusi Linux sudah memiliki Python terpasang. Untuk memastikan versi terbaru:

```
sudo apt update
sudo apt install python3 python3-pip

# Verifikasi
python3 --version
pip3 --version
```

> **PERHATIAN 3.1**
>
> Pada macOS dan Linux, perintah `python` sering kali masih mengarah ke Python 2 yang sudah pensiun. Gunakan selalu `python3` dan `pip3` untuk memastikan Anda menggunakan versi yang benar. Pada Windows, baik `python` maupun `python3` biasanya merujuk ke Python 3 jika "Add to PATH" dicentang.

> **TIPS PRO 3.1**
>
> Selain mengunduh dari python.org, alternatif yang sangat populer di kalangan ilmuwan data adalah **Anaconda** (`https://www.anaconda.com`). Anaconda mengemas Python bersama 250+ pustaka data science yang umum dipakai (NumPy, Pandas, Matplotlib, Jupyter, dan lainnya) dalam satu instalasi. Cocok untuk pemula yang ingin langsung praktis tanpa repot menginstal banyak pustaka satu per satu.

3.3.3 Mode Interaktif (REPL) versus Mode Skrip

Python menawarkan dua cara utama untuk mengeksekusi kode: **mode interaktif** dan **mode skrip**. Memahami perbedaan keduanya akan menentukan cara kerja Anda sehari-hari.

Mode Interaktif (REPL)

REPL adalah singkatan dari *Read–Eval–Print Loop*: Python membaca satu baris perintah, mengevaluasinya, mencetak hasilnya, lalu kembali menunggu perintah berikutnya. Untuk masuk ke REPL, cukup ketik `python` (atau `python3`) di terminal:

```
$ python
Python 3.12.4 (main, Jun  6 2024, 18:26:44) [Clang ...]
Type "help", "copyright", "credits" or "license" for more
information.
>>> 2 + 3
5
>>> print("Halo, dunia!")
Halo, dunia!
>>> nama = "Yusri"
>>> print(f"Selamat datang, {nama}!")
Selamat datang, Yusri!
>>> exit()
```

Tanda `>>>` disebut *prompt*, artinya Python siap menerima perintah. REPL sangat cocok untuk: eksperimen kecil, memeriksa sintaks, menguji satu

ekspresi, atau belajar pustaka baru. Namun, kode yang Anda tulis di REPL tidak tersimpan; saat Anda menutup sesi, semuanya hilang.

Mode Skrip

Untuk program yang lebih panjang dan ingin disimpan, kita menggunakan mode skrip. Kode ditulis dalam berkas dengan ekstensi `.py` (misalnya `halo.py`), kemudian dijalankan dengan perintah:

```
python halo.py
```

Mode skrip adalah cara standar untuk semua program serius. Mayoritas waktu Anda akan dihabiskan di mode ini, sementara REPL berperan sebagai "laboratorium mini" untuk uji coba cepat.

> **DI BALIK LAYAR 3.1**
>
> Baik mode interaktif maupun mode skrip dijalankan oleh program yang sama: *interpreter Python* (biasanya CPython, yaitu implementasi standar yang ditulis dalam bahasa C). Interpreter ini membaca kode Anda, mengompilasinya menjadi *bytecode* (instruksi tingkat menengah), lalu menjalankan bytecode tersebut di mesin virtual Python. Itulah sebabnya program Python yang sama dapat berjalan di Windows, macOS, dan Linux tanpa perubahan: interpreter-nyalah yang berbeda, bukan kodenya.

3.3.4 IDE dan Editor: Memilih Ruang Kerja

IDE (*Integrated Development Environment*) adalah aplikasi yang menggabungkan editor kode, terminal, debugger, dan berbagai bantuan dalam satu antarmuka. Tabel berikut membandingkan empat pilihan populer untuk pemula:

IDE/Editor	Keunggulan	Cocok untuk	Catatan
IDLE	Sudah terpasang otomatis bersama	Pemula yang baru pertama	Fitur terbatas; tidak nyaman

IDE/Editor	Keunggulan	Cocok untuk	Catatan
	Python; ringan; sederhana	kali menulis Python	untuk proyek besar
VS Code	Ekstensibel; ringan; gratis; ekosistem ekstensi yang masif	Mayoritas pengembang profesional	Membutuhkan instalasi ekstensi Python
PyCharm	Fitur lengkap; refactoring otomatis; debugger canggih	Proyek berskala menengah hingga besar	Edisi Professional berbayar; cukup berat
Jupyter Notebook	Berbasis sel; mendukung visualisasi inline; markdown	Data science; eksplorasi data; pembelajaran	Kurang cocok untuk aplikasi produksi

Untuk mata kuliah ini, kami merekomendasikan **Visual Studio Code (VS Code)** sebagai editor utama, dengan **Jupyter Notebook** sebagai pelengkap untuk eksplorasi data dan visualisasi. VS Code gratis, ringan, dan menjadi standar de facto di industri.

Persiapan VS Code untuk Python

- Unduh dan instal VS Code dari `https://code.visualstudio.com`
- Buka VS Code, klik ikon Extensions di sidebar kiri (atau tekan Ctrl+Shift+X)
- Cari "Python" dan instal ekstensi resmi dari Microsoft
- Sebagai pelengkap, instal juga "Pylance" untuk auto-completion yang lebih cerdas
- Buka folder kerja Anda dengan File > Open Folder

WAWASAN INDUSTRI 3.1

Survei tahunan Stack Overflow Developer Survey secara konsisten menempatkan VS Code di posisi teratas IDE/editor yang paling banyak digunakan oleh pengembang profesional sejak akhir dekade 2010-an. Visual Studio dan IntelliJ IDEA (basis dari PyCharm) juga selalu

termasuk dalam jajaran teratas. Penguasaan VS Code menjadi modal yang langsung relevan ketika Anda terjun ke dunia kerja, terlepas dari spesialisasi bidang.

3.3.5 Program Pertama: "Hello, World!"

Tradisi pemrograman menempatkan program "Hello, World!" sebagai program pertama yang ditulis dalam suatu bahasa baru. Tradisi ini diperkenalkan oleh **Brian Kernighan** dalam buku *The C Programming Language* (1978). Tujuannya sederhana: memverifikasi bahwa lingkungan pengembangan berfungsi dan menampilkan satu output ke layar.

Dalam Python, program tersebut hanya membutuhkan satu baris:

```
print("Hello, World!")
```

Bandingkan dengan versi dalam Java, yang membutuhkan kerangka berlapis:

```
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello, World!");
    }
}
```

Perbandingan ini bukan untuk menunjukkan bahwa Java lebih buruk — Java memiliki tujuannya sendiri — melainkan untuk mengilustrasikan *biaya sintaktis* yang rendah dari Python. Pemula Python dapat segera fokus pada logika program, bukan pada kerangka boilerplate.

Menjalankan Hello, World!

- Buka editor (misalnya VS Code) dan buat berkas baru bernama `halo.py`
- Ketikkan kode satu baris: `print("Hello, World!")`
- Simpan berkas (Ctrl+S atau Cmd+S)
- Buka terminal terintegrasi (Ctrl+` di VS Code) dan jalankan:

```
python halo.py
```

Output yang Anda lihat:

```
Hello, World!
```

> **CATATAN 3.1**
>
> Selamat! Jika Anda berhasil sampai pada output di atas, Anda telah menyelesaikan ritual peralihan setiap programmer pemula. Tradisi ini telah dilakukan oleh jutaan orang di seluruh dunia sejak akhir 1970-an, dalam puluhan bahasa pemrograman yang berbeda. Anda kini menjadi bagian dari komunitas global tersebut.

3.3.6 Anatomi Skrip Python

Sebelum menulis program yang lebih kompleks, mari kita kenali komponen-komponen dasar yang menyusun sebuah skrip Python.

Statement dan Ekspresi

Statement adalah satu instruksi lengkap yang dapat dieksekusi Python. **Ekspresi** adalah kombinasi nilai, variabel, dan operator yang menghasilkan satu nilai baru. Setiap baris kode dalam Python umumnya adalah satu statement.

```
# Contoh statement: penugasan
x = 10

# Contoh ekspresi: menghasilkan nilai
x + 5             # Ekspresi yang menghasilkan 15
x * 2 - 3         # Ekspresi yang menghasilkan 17

# Statement yang berisi ekspresi
y = x + 5         # Ekspresi (x + 5) dievaluasi, lalu disimpan
ke y
print(x * 2)      # Ekspresi (x * 2) dievaluasi, lalu dicetak
```

Indentasi sebagai Sintaks

Berbeda dari kebanyakan bahasa lain yang menggunakan kurung kurawal { } untuk menandai blok kode, Python menggunakan **indentasi** (lekuk masuk). Spasi atau tab di awal baris bukan sekadar estetika; ia adalah bagian dari sintaks bahasa.

```
# Indentasi yang benar (4 spasi)
if x > 0:
    print("Positif")
    print("Ini bagian dari if")
print("Ini di luar if")
```

Standar PEP 8 (*Style Guide for Python Code*) merekomendasikan penggunaan **4 spasi** per tingkat indentasi. Jangan campur tab dan spasi dalam satu berkas; ini sering menyebabkan `IndentationError` yang menyebalkan.

> **PERHATIAN 3.2**
>
> Indentasi yang tidak konsisten adalah salah satu kesalahan paling sering dialami pemula Python. Pesan `IndentationError: unexpected indent` atau `TabError: inconsistent use of tabs and spaces` hampir selalu disebabkan oleh pencampuran tab dan spasi. Atur editor Anda untuk mengonversi tab menjadi 4 spasi secara otomatis (di VS Code: Settings > Editor: Tab Size = 4, Insert Spaces = on).

Penamaan Identifier

Identifier adalah nama yang Anda berikan kepada variabel, fungsi, atau objek lain. Aturan baku:

- Diawali huruf atau garis bawah, tidak boleh angka
- Hanya berisi huruf, angka, dan garis bawah
- Case-sensitive: nama, Nama, dan NAMA dianggap berbeda
- Tidak boleh menggunakan kata kunci Python (if, for, while, class, def, dan lain-lain)

Konvensi PEP 8 untuk penamaan:

```
nilai_mahasiswa = 85         # snake_case untuk variabel &
fungsi
MAX_SISWA = 40              # SCREAMING_SNAKE untuk konstanta
class Mahasiswa: ...        # PascalCase untuk nama kelas

# Penamaan yang baik vs kurang baik:
nama_lengkap = "Yusri"      # Baik: deskriptif
nl = "Yusri"                # Buruk: terlalu pendek
namaLengkap = "Yusri"       # Buruk: gaya Java, bukan Python
```

3.3.7 Komentar dan Dokumentasi Kode

Komentar adalah teks dalam kode yang diabaikan oleh interpreter, tetapi dibaca oleh manusia. Tujuannya: menjelaskan *mengapa* suatu kode ditulis (bukan *apa* yang dilakukannya — itu sudah terlihat dari kodenya). Python mendukung dua jenis komentar:

Komentar Baris dengan #

```
# Ini komentar satu baris
x = 10  # Komentar di akhir baris kode

# Komentar dapat dipakai untuk:
# 1. Menjelaskan logika yang tidak intuitif
# 2. Menonaktifkan sementara baris kode (debugging)
# 3. Memberi konteks atau referensi sumber
```

Komentar Blok dengan Triple-Quoted String

```
"""
Program: Konverter Suhu
Penulis: Yusri Ikhwani
Tanggal: Mei 2026
Tujuan : Mengubah suhu Celsius menjadi Fahrenheit
"""

celsius = 25
```

```
fahrenheit = (celsius * 9/5) + 32
print(f"{celsius}°C = {fahrenheit}°F")
```

Triple-quoted string (`"""..."""` atau `'''...'''`) sebenarnya bukan komentar dalam arti teknis; ia adalah string literal. Namun ketika tidak ditugaskan ke variabel apa pun, ia berfungsi seperti komentar blok. Ia juga digunakan sebagai *docstring* untuk dokumentasi fungsi (akan dibahas di Bab 11).

> **TIPS PRO 3.2**
>
> Aturan emas komentar: **tulis komentar untuk menjelaskan mengapa, bukan apa**. Komentar buruk: `x = x + 1  # Tambahkan 1 pada x` (sudah jelas dari kode). Komentar baik: `x = x + 1  # Geser indeks ke pengguna berikutnya` (menjelaskan tujuan). Programmer berpengalaman menilai kualitas kode salah satunya dari kualitas komentarnya.

3.3.8 Eksekusi Skrip melalui Terminal dan IDE

Setelah skrip ditulis, ada beberapa cara untuk menjalankannya. Programmer profesional umumnya menguasai keduanya:

Cara 1: Terminal

Cara paling fundamental dan portabel. Buka terminal, pindah ke folder tempat berkas berada, lalu jalankan:

```
# Pindah ke folder tempat berkas berada
cd Documents/PythonLatihan

# Jalankan skrip
python halo.py

# Pada macOS/Linux gunakan python3
python3 halo.py
```

Cara 2: Tombol Run di IDE

Di VS Code, klik tombol play (▶) di kanan atas editor saat sedang membuka berkas `.py`. Di PyCharm, tekan Shift+F10. Di IDLE, tekan F5. Cara ini lebih cepat untuk siklus uji coba, tetapi terkadang menyembunyikan detail yang berguna untuk dipelajari.

Cara 3: Menjadikan Skrip Dapat Dieksekusi Langsung (macOS/Linux)

Pada sistem Unix-like, Anda dapat menambahkan baris *shebang* di awal skrip dan memberinya izin eksekusi:

```
#!/usr/bin/env python3
print("Hello, World!")
```

```
chmod +x halo.py
./halo.py
```

3.3.9 Pengantar Lingkungan Virtual (Pengayaan)

Lingkungan virtual (*virtual environment*) adalah folder terisolasi yang berisi salinan Python dan pustaka-pustaka yang dipasang khusus untuk satu proyek. Konsep ini penting karena proyek yang berbeda sering membutuhkan versi pustaka yang berbeda.

Cara membuat dan menggunakan venv:

```
# Membuat lingkungan virtual bernama 'env'
python -m venv env

# Mengaktifkannya di Windows
env\Scripts\activate

# Mengaktifkannya di macOS/Linux
source env/bin/activate

# Memasang pustaka di dalam venv
pip install requests
```

```
# Keluar dari venv
deactivate
```

> **CATATAN 3.2**
>
> Materi venv bersifat **opsional pengayaan** untuk semester pertama. Anda tidak perlu menggunakannya selama mata kuliah ini, karena seluruh latihan dapat dijalankan di lingkungan Python default. Namun, mengenalnya sejak awal akan memudahkan Anda saat memasuki proyek-proyek nyata di semester yang lebih tinggi.

3.4 Contoh Kode Lengkap

Mari kita rangkum semua konsep bab ini dalam satu skrip Python yang lebih bermakna daripada "Hello, World!". Skrip berikut menyapa pengguna dengan nama mereka dan menampilkan informasi singkat:

```
"""
Program: Penyapa Pengguna
Penulis: Mahasiswa Algoritma & Pemrograman 1
Tujuan : Demonstrasi struktur dasar program Python
         (input, pemrosesan string, dan output berformat)
"""

# Konstanta program
NAMA_INSTITUSI = "Universitas Islam Kalimantan"
VERSI = "1.0"

# Menyapa pengguna
print("=" * 50)
print(f"  Selamat datang di program Python (v{VERSI})")
print("=" * 50)

# Meminta masukan
nama = input("Siapa nama Anda? ")
asal = input("Dari mana asal Anda? ")

# Memproses dan menampilkan hasil
```

```
print()
print(f"Halo, {nama.title()} dari {asal.title()}!")
print(f"Selamat belajar Python di {NAMA_INSTITUSI}.")
print("Semoga Anda menikmati perjalanan pembelajaran ini.")
```

Contoh eksekusi:

```
================================================
  Selamat datang di program Python (v1.0)
================================================
Siapa nama Anda? yusri
Dari mana asal Anda? banjarmasin

Halo, Yusri dari Banjarmasin!
Selamat belajar Python di Universitas Islam Kalimantan.
Semoga Anda menikmati perjalanan pembelajaran ini.
```

Skrip di atas mengilustrasikan beberapa praktik baik sekaligus: docstring di awal berkas, komentar yang menjelaskan struktur, penamaan konstanta dengan SCREAMING_SNAKE_CASE, penggunaan f-string untuk formatting, dan metode `.title()` untuk mengkapitalisasi nama dengan rapi. Banyak konsep di sini akan dibahas tuntas pada bab-bab berikutnya; untuk saat ini cukup memahami bahwa skrip dapat dijalankan dan menghasilkan keluaran yang interaktif.

3.5 Kesalahan Umum dan Cara Mengatasinya

Kesalahan-kesalahan berikut sering ditemui mahasiswa di bab ini. Mengenalinya sejak awal akan menghemat waktu Anda secara signifikan:

Kesalahan 1: Lupa mencentang "Add Python to PATH" di Windows.

Miskonsepsi: "Sudah saya instal, kok perintah python tidak dikenali di Command Prompt?"

Realita: Tanpa opsi "Add to PATH", Windows tidak tahu di mana lokasi executable Python. Solusi: jalankan ulang installer Python, pilih "Modify", lalu centang opsi tersebut. Alternatif: tambahkan path Python ke variabel PATH secara manual melalui System Properties.

Kesalahan 2: Mencampur tab dan spasi untuk indentasi.

Miskonsepsi: "Tab dan spasi terlihat sama di layar, jadi pasti dianggap sama oleh Python."

Realita: Python 3 menganggap pencampuran tab dan spasi sebagai error. Aturlah editor untuk selalu mengkonversi tab menjadi 4 spasi (di VS Code: tombol "Spaces: 4" di pojok kanan bawah; pilih "Convert Indentation to Spaces"). Hindari menyalin kode dari sumber yang bercampur.

Kesalahan 3: Menyimpan berkas tanpa ekstensi .py.

Miskonsepsi: "Saya simpan dengan nama 'halo' saja, kan isinya kode Python."

Realita: Tanpa ekstensi .py, sistem tidak mengenali berkas tersebut sebagai skrip Python. VS Code juga tidak akan memberikan syntax highlighting yang benar. Selalu simpan dengan akhiran .py.

Kesalahan 4: Tanda kutip yang tidak konsisten atau salah jenis.

Miskonsepsi: "Saya pakai kutip miring 'halo' atau "halo" dari Word, harusnya bisa."

Realita: Python hanya menerima kutip lurus ASCII ('...' atau "..."). Kutip melengkung (smart quotes) dari Microsoft Word akan menyebabkan SyntaxError. Selalu tulis kode di editor teks atau IDE, bukan di word processor.

Kesalahan 5: Menggunakan kata kunci Python sebagai nama variabel.

Miskonsepsi: "Kenapa saya tidak boleh menamai variabel 'class' atau 'for'?"

Realita: Nama-nama tersebut adalah kata kunci yang dicadangkan oleh Python untuk konstruksi bahasa. Daftar lengkap dapat dilihat dengan menjalankan import keyword; print(keyword.kwlist) di REPL. Gunakan nama lain seperti 'kelas' atau 'jumlah'.

3.6 Rangkuman

Bab 3 telah memperkenalkan lingkungan pengembangan Python secara menyeluruh. Python lahir dari proyek pribadi Guido van Rossum pada akhir 1980-an dan kini menjadi salah satu bahasa pemrograman paling populer di dunia. Filosofi desainnya, yang dirangkum dalam *The Zen of Python*, menekankan keindahan, eksplisit, dan keterbacaan kode.

Instalasi Python berbeda antar sistem operasi tetapi pada dasarnya sederhana. Anda dapat menjalankan kode Python dalam dua mode: mode interaktif (REPL) untuk eksperimen cepat, dan mode skrip untuk program lengkap. IDE seperti VS Code, PyCharm, dan Jupyter Notebook menawarkan pengalaman pengembangan yang lebih nyaman dibanding editor teks polos.

Setiap skrip Python tersusun dari statement dan ekspresi, dengan indentasi 4 spasi sebagai bagian dari sintaks bahasa. Identifier mengikuti aturan penamaan tertentu, sedangkan komentar dengan tanda # atau triple-quoted string membantu manusia memahami kode. Setelah menulis skrip, Anda dapat menjalankannya melalui terminal, tombol Run di IDE, atau (pada Unix-like) dengan menjadikannya executable.

Dengan menyelesaikan bab ini, Anda telah memiliki lingkungan kerja yang lengkap untuk menulis dan menjalankan program Python. Bab-bab berikutnya akan mengisi lingkungan kerja ini dengan kemampuan-kemampuan pemrograman yang sesungguhnya.

3.7 Latihan Terbimbing

Soal 3.7.1

Soal: Jelaskan perbedaan mode interaktif (REPL) dan mode skrip dalam Python, beserta kapan masing-masing sebaiknya digunakan.

Pembahasan: Mode interaktif (REPL) mengeksekusi satu baris perintah pada satu waktu, hasilnya langsung tampil di layar, dan tidak menyimpan kode setelah sesi berakhir. Mode ini cocok untuk eksperimen kecil, memeriksa sintaks, atau belajar pustaka baru. Mode skrip menjalankan kode dari berkas berekstensi .py secara keseluruhan; kode tersimpan permanen dan dapat dijalankan ulang. Mode ini cocok untuk program serius, tugas kuliah, dan aplikasi yang akan didistribusikan.

Soal 3.7.2

Soal: Tulislah skrip Python pendek yang menampilkan nama, NIM, dan program studi Anda, masing-masing pada baris terpisah. Sertakan komentar yang menjelaskan tujuan program.

Pembahasan:

```
"""
Program: Identitas Diri
Tujuan : Menampilkan identitas mahasiswa
"""

# Cetak identitas
print("Nama          : Yusri Ikhwani")
print("NIM           : 2026010001")
print("Program Studi : Teknik Informatika")
```

Soal 3.7.3

Soal: Apa yang akan terjadi jika kita menjalankan kode berikut, dan mengapa? Tulis pula cara memperbaikinya. Kode: if x > 0: print("positif") print("selesai")

Pembahasan: Kode tersebut akan menghasilkan IndentationError karena print("positif") seharusnya diindentasi sebagai bagian dari blok if. Perbaikannya adalah dengan memberikan indentasi 4 spasi pada baris yang dimaksud menjadi bagian dari if. Jika print("selesai") seharusnya selalu dieksekusi (di luar if), maka ia tidak diindentasi. Versi yang benar: if x > 0: [enter] [4 spasi] print("positif") [enter] print("selesai")

Soal 3.7.4

Soal: Sebutkan tiga aforisme dari The Zen of Python yang menurut Anda paling relevan bagi pemula, dan jelaskan mengapa.

Pembahasan: Tiga aforisme yang sangat relevan adalah: (1) “Readability counts” — mengingatkan pemula bahwa kode akan lebih sering dibaca daripada ditulis, sehingga investasi dalam kejelasan terbayar berlipat. (2) “Explicit is better than implicit” — mendorong pemula menulis kode yang menyatakan maksudnya secara terang-terangan, bukan mengandalkan trik tersembunyi. (3) “Simple is better than complex” — mendorong pemula tidak terjebak menulis kode rumit untuk masalah yang sederhana. Jawaban lain dapat diterima asalkan disertai alasan yang masuk akal.

Soal 3.7.5

Soal: Mengapa rekomendasi PEP 8 menyatakan bahwa indentasi sebaiknya menggunakan 4 spasi, bukan tab? Berikan minimal dua alasan.

Pembahasan: Alasan pertama: spasi terbaca konsisten di editor mana pun, sedangkan lebar tab dapat berbeda-beda (4 karakter di satu editor, 8 di editor lain), sehingga tampilan kode menjadi tidak seragam. Alasan kedua: Python 3 secara aktif menolak pencampuran tab dan spasi dalam satu blok (TabError). Menggunakan spasi sejak awal menghindari risiko pencampuran tersebut. Alasan ketiga (bonus): 4 spasi adalah keseimbangan ideal antara keterbacaan (indentasi terlihat jelas) dan ruang horizontal (tidak terlalu boros).

3.8 Latihan Mandiri

- **(C2)** Jelaskan asal-usul nama "Python" dan kaitannya dengan filosofi bahasa ini.
- **(C2)** Sebutkan minimal tiga aforisme The Zen of Python dan jelaskan maknanya dalam bahasa Anda sendiri.
- **(C3)** Pasang Python pada komputer Anda dan verifikasi dengan menjalankan perintah `python --version`. Sertakan tangkapan layar hasilnya.
- **(C3)** Bandingkan keempat IDE/editor yang dibahas (IDLE, VS Code, PyCharm, Jupyter) dengan membuat tabel berisi kelebihan dan kekurangan masing-masing menurut analisis Anda sendiri.
- **(C3)** Tulislah skrip Python yang mencetak nama Anda 5 kali secara berurutan menggunakan pengulangan tanda bintang (gunakan operator perkalian string *).
- **(C3)** Buatlah skrip Python yang meminta nama, kota asal, dan tahun lahir pengguna, lalu menampilkan kalimat sapaan yang merangkum ketiganya menggunakan f-string.
- **(C4)** Analisis kode berikut dan identifikasi minimal tiga kesalahan: "Name = input('Nama: '); print('Halo 'Name')'". Tulis ulang versi yang benar.
- **(C4)** Diskusikan: jika Python 3 mengabaikan komentar sepenuhnya, mengapa banyak proyek profesional menetapkan aturan ketat tentang gaya penulisan komentar? Apa konsekuensinya jika aturan tersebut dilanggar?

- **(C4)** Bandingkan biaya sintaktis (banyaknya kode kerangka) antara Python dan satu bahasa lain yang Anda kenal (Java, C++, JavaScript). Apa implikasinya bagi pemula?
- **(C4)** Skenario: Anda sedang mempersiapkan laptop untuk seorang mahasiswa pertukaran yang akan belajar Python selama satu semester. Susunlah daftar langkah persiapan yang lengkap, mulai dari instalasi Python hingga pemasangan IDE dan ekstensi pendukung, beserta justifikasi setiap pilihan Anda.

3.9 Tugas Proyek Mini

"Kartu Identitas Digital"

Buatlah skrip Python yang berfungsi sebagai kartu identitas digital Anda. Persyaratan teknis:

- Skrip diberi nama `kartu_identitas.py` dan dimulai dengan docstring berisi nama Anda, NIM, tanggal pembuatan, dan tujuan program.
- Skrip memuat minimal lima konstanta program (gunakan SCREAMING_SNAKE_CASE) untuk: nama lengkap, NIM, program studi, fakultas, dan motto pribadi.
- Skrip menampilkan kartu identitas dalam format yang rapi, menggunakan kombinasi `print()` dan `f-string`. Gunakan kombinasi garis horizontal dengan karakter = atau - untuk pembatas.
- Sertakan minimal tiga komentar baris (#) yang menjelaskan bagian-bagian penting kode.
- Jalankan skrip Anda dan ambil tangkapan layar hasil eksekusi.
- Tulis refleksi singkat (1 paragraf): kendala apa yang Anda alami saat menulis dan menjalankan skrip ini? Bagaimana Anda mengatasinya?

Format pengumpulan: berkas ZIP berisi (a) berkas `kartu_identitas.py`, (b) tangkapan layar hasil eksekusi (PNG/JPG), dan (c) berkas PDF berisi refleksi (maksimal 1 halaman).

3.10 Refleksi Diri

Centang pernyataan berikut yang sudah Anda kuasai. Jika ada yang belum, kembalilah ke subbab terkait.

☐ Saya mengetahui sejarah singkat Python dan kaitannya dengan filosofi The Zen of Python.

☐ Python sudah terpasang di komputer saya dan saya dapat memverifikasi versinya melalui terminal.

☐ Saya memahami perbedaan mode interaktif (REPL) dan mode skrip serta kapan menggunakan masing-masing.

☐ Saya sudah memilih dan memasang minimal satu IDE/editor (direkomendasikan VS Code) lengkap dengan ekstensi Python.

☐ Saya berhasil menulis dan menjalankan program "Hello, World!" pertama saya.

☐ Saya memahami pentingnya indentasi 4 spasi dalam Python dan dapat menghindari pencampuran tab dan spasi.

☐ Saya mengetahui aturan penamaan identifier dan konvensi PEP 8.

☐ Saya dapat menulis komentar baris (#) dan komentar blok (triple-quoted string) yang bermakna.

☐ Saya dapat menjalankan skrip Python melalui terminal dengan perintah python halo.py.

ASAH OTAK 3.2

Kini Anda telah memiliki interpreter Python di komputer Anda. Coba renungkan: program apa yang paling sederhana yang dapat "mengejutkan" Anda? Buka REPL, ketik satu baris perintah yang Anda yakin tahu hasilnya, lalu cek apakah dugaan Anda benar. Jika ya, naikkan kompleksitasnya. Jika tidak, telusuri mengapa Python memberikan hasil yang berbeda. Praktik kecil ini, yang disebut *deliberate practice*, adalah salah satu kebiasaan terbaik yang dapat Anda kembangkan sebagai programmer pemula.

BAB 4

VARIABEL, TIPE DATA, DAN OPERATOR

"Name the things you want to talk about. The world becomes clearer when its parts have names."
— Adaptasi dari Donald Knuth tentang penamaan dalam pemrograman

---

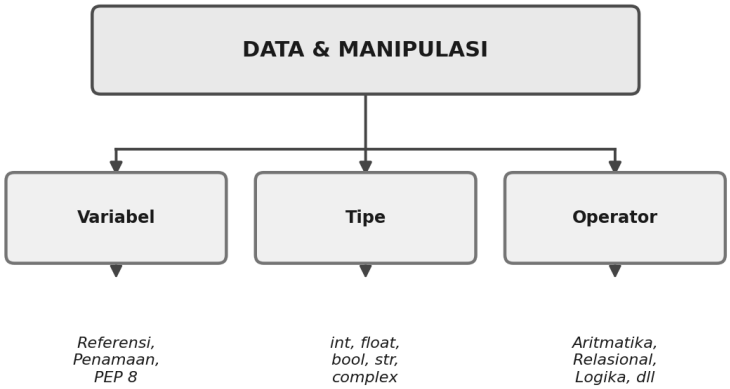
Capaian Pembelajaran Bab

Setelah mempelajari bab ini, mahasiswa diharapkan mampu mencapai sejumlah luaran pembelajaran sebagai berikut:

Kode	Capaian	Bloom
CPB-4.1	Menjelaskan konsep variabel dan model referensi objek di Python	C2
CPB-4.2	Menerapkan aturan penamaan identifier sesuai PEP 8	C3
CPB-4.3	Membedakan tipe data primitif Python (int, float, bool, str, complex)	C3
CPB-4.4	Menerapkan konversi tipe data secara eksplisit dan implisit	C3
CPB-4.5	Menggunakan operator aritmatika, relasional, logika, dan assignment dengan tepat	C3
CPB-4.6	Menganalisis ekspresi kompleks berdasarkan prioritas operator	C4

4.1 Peta Konsep

Peta konsep berikut menggambarkan keterkaitan antara variabel, tipe data, dan operator sebagai tiga pilar dasar pemrograman.



4.2 Apersepsi

Bayangkan Anda sedang mencatat hasil ujian seluruh mahasiswa di kelas. Anda menuliskan nama mereka di sisi kiri kertas, dan nilai mereka di sisi kanan. Setiap nama berfungsi sebagai "wadah" yang menyimpan satu nilai numerik. Ketika Anda ingin mengetahui nilai si Andi, Anda tidak perlu mengingat angkanya secara langsung; cukup ucapkan "Andi", dan nilai yang sesuai akan muncul di benak Anda.

Inilah konsep variabel: **sebuah nama yang merujuk pada suatu nilai**. Dalam pemrograman, variabel adalah jembatan antara pikiran manusia (yang lebih mudah mengingat nama) dengan memori komputer (yang menyimpan angka biner). Variabel membuat program dapat dibaca, dipelihara, dan dikembangkan oleh manusia.

Namun variabel saja tidak cukup. Setiap nilai memiliki **tipe data** yang menentukan operasi apa yang dapat dilakukan padanya. Anda dapat menjumlahkan dua angka, tetapi tidak dapat "menjumlahkan" dua tanggal dengan cara yang sama. Anda dapat menyambung dua kata, tetapi tidak dapat menyambung dua bilangan. Dan untuk melakukan manipulasi ini, kita

membutuhkan **operator**, yaitu simbol yang menyatakan operasi (+, -, *, /, dan seterusnya). Ketiga konsep ini—variabel, tipe data, dan operator—adalah fondasi yang akan dipakai di setiap baris kode yang Anda tulis sepanjang sisa hidup sebagai programmer.

> **ASAH OTAK 4.1**
>
> Coba renungkan: jika Anda hanya boleh menyimpan satu nilai dalam komputer, apakah Anda tetap membutuhkan variabel? Mengapa? Pikirkan hubungan antara konsep variabel dengan kebutuhan untuk "mengingat banyak hal sekaligus".

4.3 Uraian Materi

4.3.1 Konsep Variabel dan Model Memori Python

Dalam bahasa pemrograman seperti C atau Pascal, variabel sering digambarkan sebagai "kotak" yang menyimpan nilai. Anda mendeklarasikan kotak terlebih dahulu, lalu mengisinya. Python tidak bekerja seperti itu. Di Python, variabel adalah **label** (atau *reference*) yang menunjuk ke suatu objek di memori. Nilai-nya tidak "disimpan dalam variabel", melainkan disimpan di lokasi memori tertentu, dan variabel hanyalah nama yang menunjuk ke sana.

```
>>> x = 10        # 'x' adalah label yang menunjuk ke objek 10
>>> y = x         # 'y' kini menunjuk ke objek yang sama
>>> id(x)         # id menampilkan alamat memori objek
140234567890
>>> id(y)
140234567890     # sama persis dengan id(x)
>>> x = 20        # 'x' kini menunjuk ke objek baru (20)
>>> y             # 'y' tetap menunjuk ke 10
10
```

Konsekuensi praktisnya: di Python kita tidak perlu mendeklarasikan tipe variabel sebelum menggunakannya. Cukup beri nilai, dan Python akan mengetahui tipenya secara otomatis. Ini disebut *dynamic typing* (pengetikan dinamis) yang akan dibahas lebih lanjut di Subbab 4.3.6.

DI BALIK LAYAR 4.1

Setiap objek di Python memiliki tiga atribut: identitas (alamat memori, dilihat dengan id()), tipe (dilihat dengan type()), dan nilai. Ketika Anda menulis x = 10, Python: (1) mengalokasikan memori untuk objek bilangan bulat 10, (2) menentukan tipenya sebagai int, (3) menyimpan nilainya, dan (4) membuat label x yang menunjuk ke objek tersebut. Pemahaman model ini akan sangat berguna ketika kita masuk ke topik mutability di bab koleksi data.

4.3.2 Aturan dan Konvensi Penamaan Identifier (PEP 8)

Pada Bab 3 kita telah menyinggung aturan dasar penamaan identifier. Mari kita perdalam dengan konvensi PEP 8 yang menjadi standar industri:

Jenis	Konvensi	Contoh
Variabel & fungsi	snake_case (huruf kecil, dipisah underscore)	nama_mahasiswa, hitung_total, jumlah_siswa
Konstanta	SCREAMING_SNAKE_CASE	MAX_SISWA, PI, NAMA_INSTITUSI
Kelas	PascalCase	Mahasiswa, MataKuliah, KalkulatorIPK
Modul/file	lowercase atau snake_case	utils.py, hitung_nilai.py
Private/internal	diawali underscore tunggal	_helper, _internal_state

Praktik penamaan yang baik:

- **Deskriptif tetapi ringkas.** `jumlah_mahasiswa` lebih baik daripada `jm` atau `jumlah_mahasiswa_yang_terdaftar_di_kelas_ini`.
- **Hindari singkatan ambigu.** Apakah 'tmp' itu 'temperatur' atau 'temporary'? Pilihlah salah satu makna yang jelas.

- **Gunakan bahasa Inggris untuk proyek kolaboratif.** Untuk tugas kuliah bahasa Indonesia tidak masalah, tetapi kebiasaan Inggris akan membantu Anda di proyek profesional.
- **Hindari karakter yang mirip.** Jangan gunakan l (L kecil), O (huruf O besar), atau I (i besar) sebagai nama variabel—mudah tertukar dengan angka 1 dan 0.

> **PERHATIAN 4.1**
>
> Python memiliki sekumpulan kata kunci (reserved words) yang tidak boleh dipakai sebagai nama variabel: if, else, elif, while, for, def, class, return, import, from, as, in, is, not, and, or, lambda, True, False, None, dan lain-lain. Untuk daftar lengkap, jalankan di REPL: import keyword; print(keyword.kwlist).

4.3.3 Tipe Data Numerik: int, float, complex

Python memiliki tiga tipe data numerik utama:

Integer (int)

Bilangan bulat positif, negatif, atau nol. Tidak ada batas atas atau bawah di Python 3 (terbatas hanya oleh memori sistem).

```
a = 10
b = -250
c = 0
d = 1_000_000      # underscore sebagai pemisah ribuan
(kosmetik, tidak mempengaruhi nilai)
e = 0b1010         # biner = 10 desimal
f = 0o17           # oktal = 15 desimal
g = 0xFF           # heksadesimal = 255 desimal

print(type(a))     # <class 'int'>
print(d)           # 1000000
```

Float

Bilangan riil/desimal dengan presisi terbatas (umumnya 64-bit IEEE 754, sekitar 15–17 digit signifikan).

```
pi = 3.14159
suhu = -10.5
jarak = 1.5e3      # notasi ilmiah = 1500.0
kecil = 2.5e-4     # = 0.00025

print(type(pi))    # <class 'float'>
print(jarak)       # 1500.0
```

> **PERHATIAN 4.2**
>
> Float di Python (dan bahasa pemrograman lain pada umumnya) menyimpan bilangan desimal dalam basis 2, sehingga beberapa nilai yang tampak sederhana dalam basis 10 sebenarnya tidak dapat direpresentasikan secara eksak. Contoh klasik: 0.1 + 0.2 di Python menghasilkan 0.30000000000000004, bukan 0.3. Untuk perhitungan keuangan atau presisi tinggi, gunakan modul decimal dari pustaka standar Python.

Complex

Bilangan kompleks dengan bagian riil dan imajiner. Jarang digunakan di pemrograman umum, tetapi penting di komputasi ilmiah.

```
z = 3 + 4j         # 3 adalah real, 4 adalah imajiner (j =
akar dari -1)
print(z.real)      # 3.0
print(z.imag)      # 4.0
print(abs(z))      # 5.0 (besarnya: sqrt(3^2 + 4^2))
```

4.3.4 Tipe Data Boolean dan Logika

Tipe `bool` hanya memiliki dua nilai: `True` dan `False` (dengan huruf besar di awal). Tipe ini fundamental untuk pengambilan keputusan (yang akan dibahas tuntas di Bab 6).

```
lulus = True
absen = False
```

```
# Ekspresi perbandingan menghasilkan boolean
print(10 > 5)       # True
print("a" == "b")   # False

# Di balik layar, True = 1 dan False = 0
print(True + True)  # 2
print(False * 100)  # 0
```

Python juga memiliki konsep *truthy* dan *falsy*: nilai-nilai berikut dianggap False dalam konteks boolean meskipun bukan literal False: `0`, `0.0`, `''` (string kosong), `None`, `[ ]` (list kosong), dan `{ }` (dict kosong). Selain itu semua dianggap True.

```
print(bool(0))      # False
print(bool(""))     # False
print(bool("halo")) # True
print(bool(-3))     # True (negatif tetap truthy karena bukan
nol)
```

4.3.5 Tipe Data String (Pengantar)

String adalah urutan karakter yang ditulis dalam tanda kutip. Bab 5 akan membahasnya secara mendalam; di sini kita hanya kenalan dulu.

```
nama = "Yusri"          # kutip ganda
kota = 'Banjarmasin'    # kutip tunggal (sama saja)
pesan = "Halo, dunia!"

# Operasi dasar string
print(len(nama))         # 5 (banyak karakter)
print(nama + " " + kota) # "Yusri Banjarmasin" (penggabungan)
print("abc" * 3)         # "abcabcabc" (perulangan)

# f-string: cara modern menyisipkan nilai
umur = 21
print(f"{nama} berumur {umur} tahun.")
```

4.3.6 Dynamic Typing dan Duck Typing

Python adalah bahasa dengan **dynamic typing**: tipe data variabel ditentukan saat program berjalan (runtime), bukan saat penulisan kode. Konsekuensinya, satu variabel dapat berganti tipe selama eksekusi program.

```
x = 10              # x adalah int
print(type(x))      # <class 'int'>

x = "sepuluh"       # kini x adalah string
print(type(x))      # <class 'str'>

x = [1, 2, 3]       # kini x adalah list
print(type(x))      # <class 'list'>
```

Konsep terkait yang dikenal di Python adalah **duck typing**: "If it walks like a duck and quacks like a duck, it must be a duck." Maksudnya, Python tidak peduli pada tipe formal suatu objek, asalkan objek itu mendukung operasi yang dibutuhkan. Filosofi ini membuat kode Python sangat fleksibel.

> **WAWASAN INDUSTRI 4.1**
>
> Dynamic typing membuat Python sangat ekspresif dan cepat untuk prototyping, tetapi dapat menjadi sumber bug tersembunyi di proyek besar. Untuk mengatasi ini, Python 3.5+ memperkenalkan type hints (PEP 484), dan pustaka mypy memungkinkan pemeriksaan tipe secara statis sebelum kode dijalankan. Banyak perusahaan teknologi besar seperti Instagram, Dropbox, dan Microsoft mewajibkan type hints pada kode produksi mereka. Kita akan memperkenalkan type hints di Bab 11.

4.3.7 Konversi Tipe Data

Konversi tipe (type casting) terbagi atas dua jenis:

Konversi Eksplisit

Konversi yang dilakukan programmer dengan memanggil fungsi konversi tipe seperti int(), float(), str(), bool(), dan list().

```
# int(): mengonversi ke bilangan bulat
a = int("10")        # 10
b = int(3.7)         # 3 (membulatkan ke bawah)
c = int("-25")       # -25

# float(): mengonversi ke desimal
d = float("3.14")    # 3.14
e = float(5)         # 5.0

# str(): mengonversi ke string
f = str(123)         # "123"
g = str(3.14)        # "3.14"

# bool(): mengonversi ke boolean
h = bool(0)          # False
i = bool("")         # False
j = bool("halo")     # True
```

Konversi Implisit

Konversi otomatis oleh Python ketika operasi melibatkan tipe yang kompatibel.

```
# Python mengonversi int menjadi float dalam operasi campuran
hasil = 5 + 3.0      # int + float menghasilkan float
print(hasil)         # 8.0
print(type(hasil))   # <class 'float'>

# bool diperlakukan sebagai int dalam operasi aritmatika
print(True + 1)      # 2
print(False + 5.5)   # 5.5
```

TIPS PRO 4.1

Konversi int() pada string yang berisi titik desimal akan menyebabkan ValueError. Misalnya int("3.5") akan gagal; Anda harus melakukan float() dulu lalu int(). Pola umum di kode produksi: nilai = int(float(input("Masukkan: "))). Bahkan lebih baik lagi, bungkus dalam try-except untuk menangani masukan tidak valid (akan dibahas di Bab 13).

4.3.8 Operator dalam Python

Operator adalah simbol yang menyatakan operasi pada satu atau lebih operan. Python menyediakan kategori-kategori berikut:

Operator Aritmatika

Operator	Nama	Contoh	Hasil
+	Penjumlahan	10 + 3	13
-	Pengurangan	10 - 3	7
*	Perkalian	10 * 3	30
/	Pembagian (selalu float)	10 / 3	3.333...
//	Pembagian bulat	10 // 3	3
%	Modulo (sisa bagi)	10 % 3	1
**	Pangkat	10 ** 3	1000

Operator Relasional (Perbandingan)

Operator ini menghasilkan nilai boolean (True atau False).

Operator	Makna	Contoh
==	Sama dengan	5 == 5 → True
!=	Tidak sama dengan	5 != 3 → True
>	Lebih dari	5 > 3 → True
<	Kurang dari	5 < 3 → False
>=	Lebih dari atau sama	5 >= 5 → True
<=	Kurang dari atau sama	5 <= 4 → False

Operator Logika

Bekerja pada nilai boolean dan menghasilkan boolean.

```
# 'and': True jika kedua-duanya True
print(True and True)    # True
print(True and False)   # False

# 'or': True jika minimal satu True
print(False or True)    # True
print(False or False)   # False

# 'not': membalik nilai
print(not True)         # False
print(not False)        # True

# Kombinasi praktis
umur = 20
punya_ktp = True
boleh_pilih = umur >= 17 and punya_ktp
print(boleh_pilih)      # True
```

Operator Penugasan (Assignment)

```
x = 10
x += 5      # setara dengan x = x + 5  -> x menjadi 15
x -= 3      # setara dengan x = x - 3  -> x menjadi 12
x *= 2      # setara dengan x = x * 2  -> x menjadi 24
x /= 4      # setara dengan x = x / 4  -> x menjadi 6.0
x //= 2     # setara dengan x = x // 2 -> x menjadi 3.0
x **= 2     # setara dengan x = x ** 2 -> x menjadi 9.0
x %= 4      # setara dengan x = x % 4  -> x menjadi 1.0
```

Operator Identitas dan Keanggotaan

```
# 'is': memeriksa apakah dua variabel menunjuk ke objek yang
SAMA
a = [1, 2, 3]
b = a
c = [1, 2, 3]
```

```
print(a is b)    # True (b dan a label objek yang sama)
print(a is c)    # False (a dan c isinya sama tapi objeknya
beda)
print(a == c)    # True (perbandingan nilai)

# 'in': memeriksa keanggotaan
kata = "Python"
print("P" in kata)      # True
print("z" in kata)      # False
print("x" not in kata)  # True
```

4.3.9 Prioritas dan Asosiativitas Operator

Ketika sebuah ekspresi melibatkan banyak operator, Python mengeksekusi operasi sesuai urutan prioritas. Urutan dari yang tertinggi ke terendah:

Urutan	Operator	Keterangan
1	( )	Tanda kurung dievaluasi pertama
2	**	Pangkat
3	+x, -x, ~x	Unary (tanda positif/negatif)
4	*, /, //, %	Perkalian dan pembagian
5	+, -	Penjumlahan dan pengurangan
6	<, <=, >, >=, ==, !=	Perbandingan
7	not	Logika NOT
8	and	Logika AND
9	or	Logika OR

Contoh evaluasi ekspresi kompleks:

```
>>> 2 + 3 * 4
14         # bukan 20! Perkalian didahulukan: 2 + (3*4) = 2 +
12
```

```
>>> (2 + 3) * 4
20          # tanda kurung memaksa penjumlahan duluan

>>> 2 ** 3 ** 2
512         # pangkat asosiatif kanan: 2 ** (3 ** 2) = 2 ** 9

>>> 10 > 5 and 5 > 3
True        # 10>5 True, 5>3 True, True and True = True

>>> not True or False
False       # not didahulukan: (not True) or False = False or
False
```

TIPS PRO 4.2

Meskipun prioritas operator Python memiliki aturan baku, kode yang mengandalkan prioritas implisit sulit dibaca. Praktik terbaik: gunakan tanda kurung ( ) secara eksplisit untuk memperjelas urutan, terutama dalam ekspresi yang melibatkan campuran operator. Bandingkan: 'a + b * c > d' dengan '(a + (b * c)) > d'. Yang kedua sedikit lebih panjang tetapi jauh lebih jelas. Pembaca kode Anda di masa depan (termasuk diri Anda sendiri) akan berterima kasih.

4.4 Contoh Kode Lengkap

Skrip berikut mengilustrasikan kombinasi variabel, tipe data, dan operator dalam konteks aplikatif: menghitung Indeks Massa Tubuh (BMI) dan menentukan kategorinya.

```
"""
Program: Kalkulator BMI
Tujuan : Menghitung Indeks Massa Tubuh dan kategorinya
         berdasarkan tinggi dan berat badan
"""

# Konstanta batas BMI (berdasarkan kategori WHO)
BMI_KURUS = 18.5
```

```
BMI_NORMAL = 25.0
BMI_GEMUK = 30.0

# Input dari pengguna (konversi eksplisit ke float)
berat = float(input("Berat badan (kg): "))
tinggi_cm = float(input("Tinggi badan (cm): "))

# Konversi cm ke meter (penugasan dengan operator pembagian)
tinggi_m = tinggi_cm / 100

# Hitung BMI (operator pangkat **)
bmi = berat / (tinggi_m ** 2)

# Tentukan kategori menggunakan operator relasional dan logika
kurus = bmi < BMI_KURUS
normal = BMI_KURUS <= bmi < BMI_NORMAL
gemuk = BMI_NORMAL <= bmi < BMI_GEMUK
obesitas = bmi >= BMI_GEMUK

# Tampilkan hasil
print("=" * 40)
print(f"BMI Anda: {bmi:.2f}")
print("=" * 40)
print(f"Kurus     : {kurus}")
print(f"Normal    : {normal}")
print(f"Gemuk     : {gemuk}")
print(f"Obesitas  : {obesitas}")
```

Contoh eksekusi:

```
Berat badan (kg): 65
Tinggi badan (cm): 170
========================================
BMI Anda: 22.49
========================================
Kurus     : False
Normal    : True
Gemuk     : False
Obesitas  : False
```

Perhatikan beberapa hal: penggunaan konstanta dengan SCREAMING_SNAKE_CASE, konversi eksplisit `float()` pada input, operator pangkat ** untuk kuadrat, dan rantai perbandingan `BMI_KURUS <= bmi < BMI_NORMAL` yang merupakan fitur khas Python (di banyak bahasa lain harus ditulis sebagai `(BMI_KURUS <= bmi) and (bmi < BMI_NORMAL)`).

4.5 Kesalahan Umum dan Cara Mengatasinya

Berikut kesalahan-kesalahan yang sering muncul di topik variabel, tipe data, dan operator:

Kesalahan 1: Lupa konversi tipe pada input().

Miskonsepsi: "Saya input umur 20, kenapa hasil 20 + 5 = 205?"

Realita: Fungsi input() selalu mengembalikan string, bukan angka. "20" + "5" = "205" (penggabungan string). Solusi: bungkus dengan int() atau float(). umur = int(input(...)).

Kesalahan 2: Mencampurkan = dan ==.

Miskonsepsi: "Saya tulis if x = 5: kenapa eror?"

Realita: Tanda = adalah penugasan (memberi nilai), == adalah perbandingan (memeriksa kesamaan). Dalam kondisi if, gunakan ==. Python 3 secara eksplisit menolak penggunaan = di dalam if untuk mencegah bug ini.

Kesalahan 3: Mengira / di Python sama dengan / di Java/C.

Miskonsepsi: "Mengapa 10 / 2 hasilnya 5.0, bukan 5?"

Realita: Di Python 3, operator / selalu menghasilkan float, bahkan jika kedua operan bilangan bulat. Jika Anda menginginkan hasil int, gunakan // (pembagian bulat). Ini berbeda dari Python 2 dan banyak bahasa lain.

Kesalahan 4: Lupa bahwa float tidak presisi.

Miskonsepsi: "Kenapa 0.1 + 0.2 != 0.3?"

Realita: Float disimpan dalam basis 2 dan tidak dapat merepresentasikan beberapa nilai desimal secara eksak. Untuk membandingkan float, gunakan toleransi: abs(a - b) < 1e-9. Untuk hitungan finansial, gunakan modul decimal.

Kesalahan 5: Memberi tipe variabel sebagai nama.

Miskonsepsi: “Saya buat variabel namanya int = 5 dan list = [1,2,3].”

Realita: Ini akan menimpa fungsi built-in Python. Setelah list = [1,2,3], Anda tidak dapat memanggil list() lagi. Hindari nama yang sama dengan built-in: int, list, dict, str, type, sum, min, max, dll.

4.6 Rangkuman

Bab 4 telah memperkenalkan tiga pilar fundamental pemrograman: variabel, tipe data, dan operator. Di Python, variabel adalah label yang menunjuk ke objek di memori, bukan kotak yang menyimpan nilai. Aturan penamaannya mengikuti konvensi PEP 8: snake_case untuk variabel dan fungsi, SCREAMING_SNAKE untuk konstanta, dan PascalCase untuk kelas.

Tipe data primitif Python meliputi int (bilangan bulat tanpa batas), float (bilangan desimal dengan presisi terbatas), complex (bilangan kompleks), bool (True/False), dan str (string). Python menerapkan dynamic typing, sehingga variabel dapat berganti tipe selama eksekusi. Konversi tipe dapat dilakukan secara eksplisit dengan fungsi int(), float(), str(), bool() atau secara implisit dalam operasi campuran.

Operator Python terbagi menjadi aritmatika (+, -, *, /, //, %, **), relasional (==, !=, <, >, <=, >=), logika (and, or, not), penugasan (=, +=, -=, dst.), serta identitas dan keanggotaan (is, in). Setiap operator memiliki prioritas yang menentukan urutan evaluasi dalam ekspresi kompleks; gunakan tanda kurung untuk menjaga keterbacaan.

4.7 Latihan Terbimbing

Soal 4.7.1

Soal: Berapakah hasil ekspresi 2 + 3 ** 2 * 4 - 10 / 2? Jelaskan langkah evaluasinya.

Pembahasan:

Berdasarkan prioritas operator: ** dulu, lalu * dan /, baru + dan -.

Langkah 1: 3 ** 2 = 9. Ekspresi menjadi: 2 + 9 * 4 - 10 / 2

Langkah 2: 9 * 4 = 36. Ekspresi menjadi: 2 + 36 - 10 / 2

Langkah 3: 10 / 2 = 5.0. Ekspresi menjadi: 2 + 36 - 5.0

Langkah 4 (kiri ke kanan): 2 + 36 = 38; 38 - 5.0 = 33.0

Hasil akhir: 33.0 (karena ada pembagian /, hasilnya float)

Soal 4.7.2

Soal: Tulis skrip Python yang meminta pengguna memasukkan dua bilangan, lalu menampilkan hasil keenam operasi aritmatika (+ - * / // % **) di antara keduanya.

Pembahasan:

```
a = float(input("Bilangan pertama : "))
b = float(input("Bilangan kedua   : "))

print(f"{a} + {b} = {a + b}")
print(f"{a} - {b} = {a - b}")
print(f"{a} * {b} = {a * b}")
print(f"{a} / {b} = {a / b}")
print(f"{a} // {b} = {a // b}")
print(f"{a} % {b} = {a % b}")
print(f"{a} ** {b} = {a ** b}")
```

Soal 4.7.3

Soal: Jelaskan perbedaan operator is dan == di Python dengan contoh konkret.

Pembahasan: Operator == membandingkan nilai dua objek, sedangkan is membandingkan identitas (alamat memori). Contoh: a = [1,2,3]; b = [1,2,3]. Maka a == b menghasilkan True (nilai sama), tetapi a is b menghasilkan False (objeknya berbeda di memori). Sebaliknya jika c = a, maka c is a menghasilkan True karena c hanya label tambahan ke objek yang sama. Aturan praktis: gunakan == untuk membandingkan nilai (yang umumnya kita inginkan), gunakan is hanya untuk membandingkan dengan None (misal: x is None).

Soal 4.7.4

Soal: Apa output kode berikut, dan mengapa? x = 10; x += 5; x **= 2; x //= 3; print(x)

Pembahasan: Langkah demi langkah: (1) x = 10. (2) x += 5 → x = 15. (3) x **= 2 → x = 15^2 = 225. (4) x //= 3 → x = 225 // 3 = 75. Output: 75.

Soal 4.7.5

Soal: Tulis ekspresi boolean untuk: "mahasiswa lulus jika nilai >= 60 dan absensi >= 80%, atau jika nilai >= 90 tanpa peduli absensi."

Pembahasan:

```
nilai = 75
absensi = 85

lulus = (nilai >= 60 and absensi >= 80) or (nilai >= 90)
print(f"Lulus: {lulus}")
```

4.8 Latihan Mandiri

- **(C2)** Jelaskan dengan kata-kata Anda sendiri mengapa Python disebut menggunakan model referensi (bukan model kotak) untuk variabel.
- **(C2)** Sebutkan minimal lima nilai yang bersifat falsy di Python, dan jelaskan mengapa masing-masing dianggap False dalam konteks boolean.
- **(C3)** Tulislah satu baris kode Python yang mengubah variabel detik (bertipe int) menjadi jam, menit, dan detik. Contoh: 3725 detik → 1 jam, 2 menit, 5 detik.
- **(C3)** Buatlah skrip yang meminta tiga nilai mata kuliah dan menghitung rata-ratanya dengan dua angka di belakang koma.
- **(C3)** Susun ekspresi boolean untuk menentukan apakah suatu tahun adalah tahun kabisat (habis dibagi 4 dan tidak habis dibagi 100, atau habis dibagi 400).
- **(C3)** Tulislah program untuk mengubah suhu dari Celsius ke Fahrenheit dan Kelvin sekaligus.
- **(C4)** Analisis output kode berikut dan jelaskan setiap baris: x = True + False * 5; y = bool(x); z = x // 2; print(x, y, z).
- **(C4)** Bandingkan presisi int dan float di Python untuk operasi 10**50. Apakah keduanya menyimpan hasil dengan presisi sama? Mengapa?

- **(C4)** Diskusikan: mengapa Python 3 mengubah perilaku operator / dari Python 2 (di mana 10/3 = 3) menjadi selalu menghasilkan float (10/3 = 3.333...)? Apa keuntungan dan kerugiannya?
- **(C4)** Studi kasus: Anda sedang menulis aplikasi keuangan untuk menghitung bunga deposito. Mengapa penggunaan tipe float dapat berbahaya, dan alternatif apa yang sebaiknya digunakan?

4.9 Tugas Proyek Mini

"Kalkulator Pengeluaran Harian"

Buatlah skrip Python yang membantu seorang mahasiswa menghitung total pengeluaran harian dan kategorisasinya. Persyaratan teknis:

- Skrip meminta input 5 item pengeluaran (makan, transportasi, fotokopi, internet, lain-lain) dalam satuan rupiah.
- Hitung total pengeluaran dan persentase setiap item terhadap total.
- Berdasarkan total, tampilkan kategori: "Hemat" (< Rp 50.000), "Normal" (Rp 50.000 – Rp 100.000), atau "Boros" (> Rp 100.000).
- Gunakan minimal: konstanta SCREAMING_SNAKE_CASE untuk batas kategori, konversi tipe eksplisit pada input, format f-string untuk output, operator relasional untuk kategorisasi.
- Tampilan output yang rapi dengan pembatas garis.

Format pengumpulan: berkas .py beserta tangkapan layar hasil eksekusi untuk minimal dua skenario uji coba (satu hari hemat, satu hari boros).

4.10 Refleksi Diri

Centang pernyataan berikut yang sudah Anda kuasai:

☐ Saya memahami konsep variabel sebagai referensi ke objek di Python.

☐ Saya dapat menerapkan konvensi penamaan PEP 8 untuk variabel, konstanta, dan kelas.

☐ Saya mengenali kelima tipe data primitif Python beserta karakteristik masing-masing.

☐ Saya memahami perbedaan dynamic typing Python dengan static typing bahasa lain.

☐ Saya dapat melakukan konversi tipe data secara eksplisit dengan int(), float(), str(), bool().

☐ Saya menguasai operator aritmatika, relasional, logika, dan assignment.

☐ Saya dapat memprediksi hasil ekspresi kompleks berdasarkan prioritas operator.

☐ Saya menyadari pentingnya tanda kurung untuk menjaga keterbacaan kode.

ASAH OTAK 4.2

Buka REPL Python dan eksplorasi: jalankan id(1), lalu id(1) lagi. Apakah hasilnya sama? Sekarang coba id(10**5) dan id(10**5) lagi—masih sama? Bagaimana dengan id("a") versus id("a")? Python memiliki optimasi khusus untuk objek kecil yang disebut interning. Telusuri konsep ini di dokumentasi Python dan diskusikan dampaknya pada perbandingan dengan operator is.

BAB 5

INPUT/OUTPUT DAN OPERASI STRING

"Text is the most powerful designed object in computer science."
— Bret Victor, peneliti antarmuka manusia-komputer

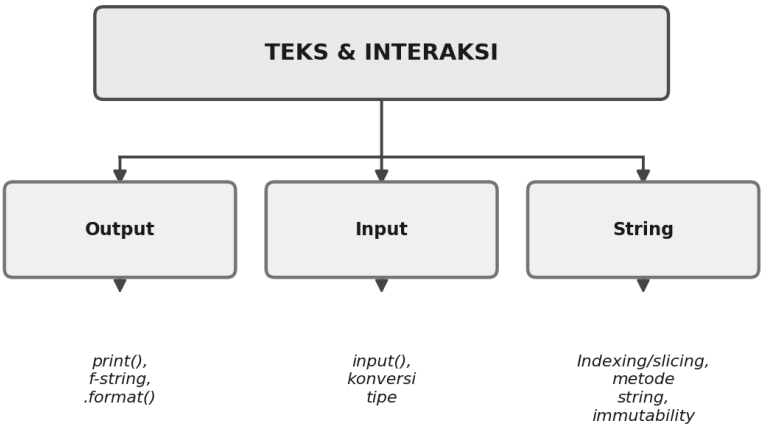
Capaian Pembelajaran Bab

Setelah mempelajari bab ini, mahasiswa diharapkan mampu mencapai sejumlah luaran pembelajaran sebagai berikut:

Kode	Capaian	Bloom
CPB-5.1	Menggunakan fungsi print() dengan parameter sep, end, dan file	C3
CPB-5.2	Memformat keluaran menggunakan f-string, .format(), dan operator %	C3
CPB-5.3	Menerapkan fungsi input() dengan konversi tipe yang tepat	C3
CPB-5.4	Menjelaskan karakteristik string sebagai urutan karakter immutable	C2
CPB-5.5	Menerapkan indexing dan slicing pada string	C3
CPB-5.6	Menggunakan metode string penting untuk manipulasi teks	C3
CPB-5.7	Menerapkan escape sequences, raw string, dan string multi-baris	C3

5.1 Peta Konsep

Peta konsep berikut menunjukkan keterkaitan antara input/output dan operasi string yang menjadi fokus Bab 5.



5.2 Apersepsi

Sebuah program yang tidak berkomunikasi dengan dunia luar bagaikan teleskop yang hanya melihat ke dinding—ia berfungsi, tetapi tanpa makna. Setiap aplikasi yang Anda gunakan setiap hari, mulai dari kalkulator HP hingga ChatGPT, pada hakikatnya melakukan dua hal: **menerima sesuatu dari pengguna (input)** dan **menampilkan sesuatu kepada pengguna (output)**. Di antara keduanya terjadi pemrosesan, namun tanpa input dan output, pemrosesan itu tidak akan pernah dirasakan manusia.

Bab 5 mengajarkan Anda dua keterampilan yang akan dipakai pada setiap program yang Anda tulis sepanjang karier: cara meminta data dari pengguna, dan cara menampilkan hasilnya. Sebagai bonus, kita akan mendalami tipe data **string**, yaitu cara komputer merepresentasikan teks. String adalah tipe data paling sering dipakai dalam aplikasi nyata—lebih sering daripada angka. Setiap kali Anda mengetik di WhatsApp, mencari di Google, atau membuat postingan Instagram, Anda sedang memproses string.

ASAH OTAK 5.1

Mengapa programmer menggunakan istilah "string" (untaian) untuk merepresentasikan teks? Pikirkan asal-usul kata ini dan analogi yang mungkin ada antara teks digital dan untaian (seperti tasbih atau kalung). Hubungkan dengan konsep bahwa string adalah "urutan karakter" yang akan Anda pelajari di subbab 5.3.4.

5.3 Uraian Materi

5.3.1 Fungsi print(): Lebih dari Sekadar Mencetak

Fungsi `print()` sudah kita pakai sejak Bab 3, tetapi ia memiliki kemampuan yang lebih kaya daripada sekadar menampilkan teks. Tanda tangan lengkap fungsi ini adalah:

```
print(*objects, sep=' ', end='\n', file=sys.stdout,
flush=False)
```

Parameter sep menentukan pemisah antar argumen (default: spasi):

```
print("a", "b", "c")               # a b c
print("a", "b", "c", sep="-")      # a-b-c
print("a", "b", "c", sep="")       # abc
print("a", "b", "c", sep=" | ")    # a | b | c
print(2026, 5, 16, sep="-")         # 2026-5-16 (format
tanggal)
```

Parameter end menentukan karakter di akhir output (default: newline \n):

```
# Default: setiap print pada baris baru
print("Halo")
print("Dunia")
# Output:
# Halo
# Dunia
```

```
# Dengan end='', semua dalam satu baris
print("Halo", end="")
print(" Dunia")
# Output: Halo Dunia

# Berguna untuk progres bar atau loading dots
for i in range(5):
    print(".", end="")
# Output: .....
```

> **TIPS PRO 5.1**
>
> Untuk mencetak banyak baris sekaligus, gunakan triple-quoted string daripada banyak panggilan print(). Lebih efisien dan lebih mudah dibaca: print(""" Baris pertama Baris kedua Baris ketiga """)

5.3.2 Formatting String: Tiga Cara Modern

Menyisipkan nilai variabel ke dalam string adalah operasi yang sangat sering dilakukan. Python menawarkan tiga gaya formatting, masing-masing dengan kelebihannya:

Gaya 1: Operator % (Gaya Klasik, Mirip C)

```
nama = "Yusri"
umur = 30
ipk = 3.85

print("Nama: %s, Umur: %d, IPK: %.2f" % (nama, umur, ipk))
# Output: Nama: Yusri, Umur: 30, IPK: 3.85

# %s = string, %d = integer, %f = float, %.2f = float 2
desimal
```

Gaya 2: Metode .format() (Gaya Python 2.6+)

```
print("Nama: {}, Umur: {}, IPK: {:.2f}".format(nama, umur,
ipk))
# Output sama persis dengan gaya %

# Dengan indeks posisi
print("{0} sedang belajar {1}. {0} senang
sekali.".format(nama, "Python"))
# Output: Yusri sedang belajar Python. Yusri senang sekali.

# Dengan nama parameter
print("{nama} berasal dari {kota}".format(nama="Yusri",
kota="Banjarmasin"))
```

Gaya 3: f-string (Python 3.6+, Sangat Direkomendasikan)

```
print(f"Nama: {nama}, Umur: {umur}, IPK: {ipk:.2f}")
# Output sama dengan gaya sebelumnya, tapi jauh lebih ringkas dan
jelas

# Ekspresi langsung di dalam {}
print(f"Tahun depan umur saya {umur + 1}")
print(f"IPK saya {'tinggi' if ipk >= 3.5 else 'biasa'}")

# Format spesifier
harga = 1500000.5
print(f"Harga: Rp{harga:,.2f}")    # Rp1,500,000.50 (koma pemisah
ribuan)
print(f"{ipk:.1%}")                # 385.0% (format persen, tidak
ideal di sini)
print(f"{123:>10}")                # '       123' (rata kanan, lebar
10)
print(f"{123:<10}")                # '123       ' (rata kiri, lebar
10)
print(f"{123:^10}")                # '   123    ' (rata tengah, lebar
10)
print(f"{123:05d}")                # '00123' (padding angka nol)
```

> **CATATAN 5.1**
>
> f-string diperkenalkan di Python 3.6 (Desember 2016) dan dengan cepat menjadi standar de facto karena tiga keunggulannya: lebih

ringkas daripada .format(), lebih jelas karena nama variabel ada di tempatnya, dan lebih cepat secara eksekusi. Sejak Python 3.12, f-string bahkan mendukung ekspresi multibaris dan komentar di dalamnya. Gunakan f-string sebagai default Anda; gaya % dan .format() umumnya hanya muncul di basis kode lama.

5.3.3 Fungsi input() dan Konversi

Fungsi `input()` menunggu pengguna mengetik sesuatu dan menekan Enter, lalu mengembalikan apa yang diketik sebagai string. **Selalu string, bahkan jika pengguna mengetik angka**. Inilah sumber kesalahan paling sering bagi pemula.

```
nama = input("Siapa nama Anda? ")
print(type(nama))           # <class 'str'>

# Kesalahan klasik: melupakan konversi
umur_str = input("Umur: ")  # pengguna ketik "20"
umur_str + 5                # ERROR: cannot concatenate str +
int

# Solusi: konversi eksplisit
umur = int(input("Umur: ")) # ambil input lalu langsung
konversi
tahun_lahir = 2026 - umur
print(f"Anda lahir sekitar tahun {tahun_lahir}")
```

Untuk membaca beberapa nilai dalam satu baris (dipisah spasi), kita dapat menggabungkan input() dengan .split():

```
# Pengguna mengetik: 5 10 15 (tiga angka dipisah spasi)
data = input("Masukkan 3 bilangan dipisah spasi: ")
a, b, c = data.split()      # ['5', '10', '15'] di-unpack ke
a, b, c
a, b, c = int(a), int(b), int(c)

# Cara lebih ringkas (idiomatik Python)
a, b, c = map(int, input("Tiga bilangan: ").split())
```

```
print(f"Jumlah: {a + b + c}")
```

> **PERHATIAN 5.1**
>
> Jika pengguna memasukkan nilai yang tidak dapat dikonversi (misalnya mengetik "abc" ketika diminta angka), Python akan melempar ValueError dan program berhenti. Penanganan kesalahan ini akan dibahas di Bab 13 melalui try-except. Untuk saat ini, asumsikan pengguna memasukkan tipe yang benar.

5.3.4 String sebagai Urutan Karakter

String di Python adalah **urutan karakter** (*sequence of characters*) yang berurutan dan terindeks. Anda dapat membayangkan string seperti deretan kotak yang berisi satu karakter, dengan nomor urut mulai dari 0.

```
  String  :  P    y    t    h    o    n
  Indeks  :  0    1    2    3    4    5    (dari kiri)
  Indeks  : -6   -5   -4   -3   -2   -1    (dari kanan)
```

Beberapa operasi dasar yang dapat dilakukan pada string:

```
s = "Python"

# Panjang string
len(s)              # 6

# Penggabungan (concatenation)
"Halo, " + s        # 'Halo, Python'

# Perulangan
"-" * 10            # '----------'

# Pemeriksaan keanggotaan
"P" in s            # True
"x" in s            # False
"py" in s.lower()   # True
```

```
# Perbandingan (leksikografis, mirip kamus)
"apel" < "jeruk"    # True (a < j)
"abc" == "abc"      # True
```

5.3.5 Indexing dan Slicing

Indexing adalah cara mengakses satu karakter berdasarkan posisinya. **Slicing** adalah cara mengambil bagian (substring) dari string.

Indexing

```
s = "Python"

s[0]      # 'P' (karakter pertama)
s[3]      # 'h' (karakter ke-4)
s[5]      # 'n' (karakter terakhir)
s[-1]     # 'n' (sama, dari belakang)
s[-2]     # 'o' (dua dari belakang)

# Indeks di luar jangkauan -> IndexError
s[6]      # IndexError: string index out of range
s[100]    # IndexError
```

Slicing: [start:stop:step]

```
s = "Pemrograman Python"
#    0123456789012345678

s[0:11]       # 'Pemrograman' (dari 0 sampai sebelum 11)
s[12:18]      # 'Python'
s[:11]        # 'Pemrograman' (dari awal hingga sebelum 11)
s[12:]        # 'Python' (dari 12 hingga akhir)
s[:]          # 'Pemrograman Python' (seluruh string)

# Indeks negatif
s[-6:]        # 'Python' (enam karakter terakhir)
s[:-7]        # 'Pemrograman' (semua kecuali tujuh terakhir)
```

```
# Dengan step
s[::2]        # 'Pmormnpto' (loncat 2)
s[::-1]       # 'nohtyP namargormeP' (kebalikan!)
s[12::1]      # 'Python'
```

TIPS PRO 5.2

Slicing dengan s[::-1] adalah cara paling idiomatik di Python untuk membalik string. Lebih singkat daripada menulis loop manual. Pola ini sering muncul dalam soal palindrom: kata.lower() == kata.lower()[::-1] memeriksa apakah suatu kata adalah palindrom. Hafalkan idiom ini—akan sering Anda butuhkan.

5.3.6 Metode String Penting

Python menyediakan puluhan metode bawaan untuk memanipulasi string. Berikut yang paling sering dipakai:

Metode	Fungsi	Contoh
.upper()	Mengubah ke huruf besar semua	"halo".upper() → "HALO"
.lower()	Mengubah ke huruf kecil semua	"HALO".lower() → "halo"
.title()	Huruf besar di awal tiap kata	"yusri ikhwani".title() → "Yusri Ikhwani"
.capitalize()	Huruf besar hanya di awal string	"halo dunia".capitalize() → "Halo dunia"
.strip()	Menghapus spasi di awal dan akhir	" halo ".strip() → "halo"
.split(sep)	Memecah jadi list berdasar pemisah	"a,b,c".split(",") → ['a','b','c']
.join(list)	Menggabungkan list jadi string	",".join(['a','b','c']) → "a,b,c"

Metode	Fungsi	Contoh
.replace(a, b)	Mengganti substring a dengan b	"halo".replace("a", "o") → "holo"
.find(sub)	Cari indeks sub, -1 jika tak ada	"Python".find("th") → 2
.count(sub)	Hitung kemunculan substring	"abracadabra".count("a") → 5
.startswith(s)	Apakah dimulai dengan s?	"halo".startswith("ha") → True
.endswith(s)	Apakah diakhiri dengan s?	"file.py".endswith(".py") → True

Contoh penggunaan terpadu:

```
kalimat = "  Pemrograman Python itu Menyenangkan!  "

# Rangkaian metode (method chaining)
bersih = kalimat.strip().lower().replace("!", "")
print(bersih)  # 'pemrograman python itu menyenangkan'

# Memecah jadi kata-kata
kata_kata = bersih.split()
print(kata_kata)  # ['pemrograman', 'python', 'itu',
'menyenangkan']
print(len(kata_kata))  # 4 (jumlah kata)

# Mengubah kembali jadi string dengan pemisah berbeda
result = " | ".join(kata_kata)
print(result)  # 'pemrograman | python | itu | menyenangkan'
```

5.3.7 Immutability String

Salah satu karakteristik fundamental string Python adalah **immutability**: *string tidak dapat diubah setelah dibuat.* Anda tidak dapat memodifikasi satu karakter di dalam string.

```
s = "Python"

# Operasi ini akan ERROR
s[0] = "J"          # TypeError: 'str' object does not support
item assignment

# Yang benar: buat string baru
s = "J" + s[1:]     # 'Jython'
# Atau dengan replace
s = s.replace("P", "J")
```

Setiap metode string seperti `.upper()`, `.replace()`, dan `.strip()` selalu mengembalikan string baru, bukan memodifikasi string aslinya:

```
s = "halo"
s.upper()           # mengembalikan 'HALO' tetapi tidak
mengubah s
print(s)            # masih 'halo'

# Kita harus menugaskannya kembali
s = s.upper()
print(s)            # baru sekarang 'HALO'
```

DI BALIK LAYAR 5.1

Mengapa Python membuat string immutable? Tiga alasan utama: (1) Keamanan thread—objek immutable aman digunakan banyak thread tanpa risiko data race. (2) Optimasi memori—Python dapat melakukan interning string, yaitu menyimpan satu salinan untuk string yang sama. (3) Hashing—immutable membuat string dapat menjadi kunci dictionary. Mengubah string yang menjadi kunci akan merusak hash table. Konsep ini akan kita gunakan lagi di Bab 10.

5.3.8 Escape Sequences, Raw String, dan String Multi-baris

Escape Sequences

Karakter-karakter tertentu memerlukan notasi khusus karena mereka adalah "special" dalam string:

Notasi	Arti	Contoh
\n	Newline (baris baru)	"baris1\nbaris2"
\t	Tab (lebar bergantung editor)	"nama\tnilai"
\\	Backslash literal	"C:\\Users\\Yusri"
\'	Kutip tunggal di dalam '...'	'It\'s a book'
\"	Kutip ganda di dalam "..."	"He said \"hi\""
\r	Carriage return	(jarang dipakai)
\0	Karakter null	(untuk interop dengan C)
\uXXXX	Karakter Unicode	"\u00e9" → é

Raw String

Awali string dengan `r` untuk mematikan interpretasi escape sequences. Sangat berguna untuk path file Windows dan ekspresi reguler.

```
# Tanpa raw string, harus double-backslash
path1 = "C:\\Users\\Yusri\\Documents"

# Dengan raw string, lebih bersih
path2 = r"C:\Users\Yusri\Documents"

print(path1 == path2)  # True (keduanya menghasilkan string
yang sama)

# Berguna juga untuk regex
import re
pola = r"\d{3}-\d{4}"  # tanpa raw string harus "\\d{3}-
\\d{4}"
```

String Multi-baris

Gunakan triple quotes ('''...''' atau """...""") untuk string yang membentang banyak baris:

```
pesan = """
Yth. Mahasiswa,

Selamat datang di mata kuliah Algoritma dan Pemrograman.
Silakan persiapkan diri untuk perjalanan satu semester ke
depan.

Salam,
Dosen Pengampu
"""

print(pesan)

# Newline otomatis tercatat
print(len(pesan))  # mengandung karakter \n untuk setiap baris
baru
```

WAWASAN INDUSTRI 5.1

String adalah tipe data nomor satu yang akan Anda olah sepanjang karier sebagai programmer. Aplikasi modern membaca string dari API (format JSON), basis data (SQL queries), berkas konfigurasi (YAML, INI), dan tentu saja input pengguna. Penguasaan metode string Python akan menghemat ribuan baris kode di proyek nyata. Beberapa pustaka populer yang fokus pada pemrosesan string: re (regex), textwrap (formatting), unicodedata (Unicode), dan natural language processing seperti NLTK dan spaCy.

5.4 Contoh Kode Lengkap

Skrip berikut menggabungkan input, formatting output, dan manipulasi string dalam satu aplikasi kecil: validator dan formatter nama lengkap.

```
"""
Program: Validator dan Formatter Nama
Tujuan : Membersihkan, memformat, dan menganalisis nama lengkap
         yang dimasukkan pengguna
"""

# Header program
print("=" * 50)
print("   VALIDATOR DAN FORMATTER NAMA LENGKAP   ")
print("=" * 50)

# Ambil input dan bersihkan spasi berlebih
nama_raw = input("Masukkan nama lengkap Anda: ")
nama_bersih = nama_raw.strip()

# Validasi sederhana: minimal dua kata
kata = nama_bersih.split()
jumlah_kata = len(kata)

if jumlah_kata < 2:
    print("PERINGATAN: Nama lengkap sebaiknya minimal 2 kata.")

# Format dengan title case (huruf besar di awal tiap kata)
nama_format = nama_bersih.title()

# Buat inisial (huruf pertama tiap kata)
inisial = ".".join([k[0].upper() for k in kata]) + "."

# Tampilkan analisis
print()
print(f"{'Nama original':<25}: '{nama_raw}'")
print(f"{'Nama setelah bersih':<25}: '{nama_bersih}'")
print(f"{'Nama format title':<25}: '{nama_format}'")
print(f"{'Jumlah karakter':<25}: {len(nama_bersih)}")
print(f"{'Jumlah kata':<25}: {jumlah_kata}")
print(f"{'Inisial':<25}: {inisial}")
print(f"{'Nama terbalik (per kata)':<25}: {' '.join(kata[::-
1]).title()}")
print("=" * 50)
```

Contoh eksekusi:

```
==================================================
   VALIDATOR DAN FORMATTER NAMA LENGKAP
==================================================
Masukkan nama lengkap Anda:   yusri ikhwani putra

Nama original            : '  yusri ikhwani putra  '
Nama setelah bersih      : 'yusri ikhwani putra'
Nama format title        : 'Yusri Ikhwani Putra'
Jumlah karakter          : 19
Jumlah kata              : 3
Inisial                  : Y.I.P.
Nama terbalik (per kata) : Putra Ikhwani Yusri
==================================================
```

Skrip ini mendemonstrasikan: `.strip()` untuk membersihkan spasi, `.split()` untuk memecah jadi list kata, `.title()` untuk formatting, list comprehension untuk membuat inisial, `[::-1]` untuk membalik urutan, dan format spesifier f-string `{:<25}` untuk perataan kolom.

5.5 Kesalahan Umum dan Cara Mengatasinya

Kesalahan 1: Mengira input() mengembalikan tipe yang sesuai.

Miskonsepsi: "Kenapa input(\"Umur: \") + 5 menghasilkan eror?"

Realita: input() selalu mengembalikan string. Bungkus dengan int() atau float() jika menginginkan angka. umur = int(input(\"Umur: \"))

Kesalahan 2: Lupa bahwa string immutable.

Miskonsepsi: "Saya tulis s[0] = 'A' kok error?"

Realita: String tidak dapat dimodifikasi. Anda harus membuat string baru: s = 'A' + s[1:]. Atau gunakan metode .replace() yang juga mengembalikan string baru.

Kesalahan 3: Memanggil metode string tetapi tidak menugaskan hasilnya.

Miskonsepsi: "Saya panggil s.upper() tapi s tetap huruf kecil."

Realita: Metode string seperti .upper(), .strip(), .replace() mengembalikan string baru, bukan memodifikasi yang asli. Tugaskan hasilnya: s = s.upper().

Kesalahan 4: Salah arah pada slicing.

Miskonsepsi: "Kenapa s[5:2] menghasilkan string kosong?"

Realita: Slicing default melangkah +1 (kiri ke kanan). Untuk mengambil dari kanan ke kiri, gunakan step negatif: s[5:2:-1] atau lebih sering s[::-1] untuk membalik seluruh string.

Kesalahan 5: Bingung antara .find() dan .index().

Miskonsepsi: "Mengapa .index() menghasilkan ValueError tetapi .find() tidak?"

Realita: .find() mengembalikan -1 jika substring tidak ditemukan, sedangkan .index() melempar ValueError. Gunakan .find() untuk pengecekan aman, .index() jika Anda yakin substringnya ada.

5.6 Rangkuman

Bab 5 telah membahas dua topik fundamental: cara program berkomunikasi dengan pengguna, dan cara memanipulasi teks. Fungsi print() memiliki parameter sep, end, dan file yang memberikan kontrol penuh atas tampilan output. Python menyediakan tiga gaya formatting string: operator %, metode .format(), dan f-string (yang direkomendasikan sejak Python 3.6).

Fungsi input() selalu mengembalikan string, sehingga konversi tipe eksplisit dengan int() atau float() seringkali diperlukan. String di Python adalah urutan karakter immutable yang mendukung indexing (s[i]), slicing (s[start:stop:step]), dan berbagai operasi seperti penggabungan dan perulangan.

Metode string yang sering dipakai meliputi .upper(), .lower(), .title(), .strip(), .split(), .join(), .replace(), .find(), .startswith(), dan .endswith(). Karena string immutable, setiap metode mengembalikan string baru. Escape sequences (\n, \t, \\) memungkinkan penyisipan karakter khusus, raw string (r"...") mematikan interpretasinya, dan triple quotes memungkinkan string membentang banyak baris.

5.7 Latihan Terbimbing

Soal 5.7.1

Soal: Apa output dari s = "Pemrograman"; print(s[2:8:2])? Jelaskan langkahnya.

Pembahasan: Slicing [2:8:2] artinya mulai indeks 2, berhenti sebelum indeks 8, melangkah 2. Karakter s adalah P(0) e(1) m(2) r(3) o(4) g(5) r(6) a(7) m(8). Mulai dari indeks 2 ('m'), lalu lompat 2 ke indeks 4 ('o'), lalu indeks 6 ('r'). Output: 'mor'.

Soal 5.7.2

Soal: Tulis skrip yang membalik kalimat per kata. Contoh: \"Saya suka Python\" → \"Python suka Saya\".

Pembahasan:

```
kalimat = input("Masukkan kalimat: ")
kata = kalimat.split()
kata_terbalik = kata[::-1]
hasil = " ".join(kata_terbalik)
print(f"Hasil: {hasil}")

# Versi satu baris (idiomatik):
# print(" ".join(input("Kalimat: ").split()[::-1]))
```

Soal 5.7.3

Soal: Mengapa kode berikut error? nama = "Yusri"; nama[0] = "y". Bagaimana cara memperbaikinya?

Pembahasan: Error karena string immutable: tidak dapat memodifikasi karakter di indeks tertentu. Perbaikan: buat string baru. Beberapa cara: (1) nama = "y" + nama[1:]. (2) nama = nama.replace("Y", "y", 1) (parameter 1 untuk hanya ganti yang pertama). (3) nama = nama.lower() jika ingin seluruh string kecil.

Soal 5.7.4

Soal: Tulis program yang menerima alamat email dari pengguna dan menampilkan: (a) username, (b) domain, (c) panjang username.

Pembahasan:

```
email = input("Masukkan email: ").strip().lower()

if "@" not in email:
    print("Email tidak valid.")
else:
    posisi_at = email.find("@")
    username = email[:posisi_at]
    domain = email[posisi_at + 1:]
    print(f"Username     : {username}")
    print(f"Domain       : {domain}")
    print(f"Panjang user : {len(username)}")

# Alternatif lebih ringkas dengan .split():
# username, domain = email.split("@")
```

Soal 5.7.5

Soal: Apa output dari kode: s = " Hello World "; print(f"[{s.strip().upper().replace('O', '0')}]")?

Pembahasan: Langkah: (1) s.strip() menghapus spasi awal/akhir → 'Hello World'. (2) .upper() → 'HELLO WORLD'. (3) .replace('O', '0') mengganti O dengan 0 → 'HELL0 W0RLD'. (4) f-string membungkus dalam []. Output: [HELL0 W0RLD].

5.8 Latihan Mandiri

- **(C2)** Jelaskan perbedaan antara operator %, metode .format(), dan f-string untuk formatting string.
- **(C2)** Mengapa string di Python disebut immutable? Apa keuntungannya bagi performa dan keamanan?
- **(C3)** Tulislah program yang menampilkan tabel perkalian 1 sampai 10 dengan format kolom yang rapi menggunakan f-string format spesifier.

- **(C3)** Buatlah program yang menghitung jumlah huruf vokal (a, i, u, e, o) dalam sebuah kalimat yang dimasukkan pengguna.
- **(C3)** Tulislah skrip yang menerima nomor HP dengan format \"081234567890\" dan mengubahnya menjadi format \"+62 812-3456-7890\" menggunakan slicing dan f-string.
- **(C3)** Buatlah program palindrom yang menerima sebuah kata dan menentukan apakah kata tersebut sama jika dibaca terbalik (ignore case dan spasi).
- **(C4)** Analisis: mengapa s = s + "x" pada string panjang berkali-kali lebih lambat daripada menggunakan list dan .join() di akhir? Jelaskan dari perspektif immutability.
- **(C4)** Bandingkan ketiga gaya formatting (%, .format(), f-string) dari segi: keterbacaan, kecepatan eksekusi, dan kapan masing-masing tepat digunakan.
- **(C4)** Studi kasus: Anda menerima data nama dari basis data dalam format tidak konsisten: \"YUSRI IKHWANI\", \"yusri ikhwani\", \" Yusri Ikhwani \". Tulis fungsi pembersih yang menormalkan semuanya menjadi format \"Yusri Ikhwani\".
- **(C4)** Diskusikan: mengapa Python menerima baik kutip tunggal ('...') maupun kutip ganda (\"...\") untuk string? Bagaimana konvensi PEP 8 menyikapi pilihan ini?

5.9 Tugas Proyek Mini

"Generator Kartu Ucapan Otomatis"

Buatlah skrip Python yang menghasilkan kartu ucapan personal berdasarkan input pengguna. Persyaratan teknis:

- Skrip meminta input: nama penerima, hubungan (teman/keluarga/dosen), jenis acara (ulang tahun/wisuda/lebaran/idul fitri), dan satu pesan personal.
- Validasi: nama tidak boleh kosong, jenis acara harus salah satu dari pilihan yang valid (case-insensitive).
- Berdasarkan kombinasi hubungan dan acara, hasilkan sapaan pembuka yang sesuai (misalnya: "Sahabat tercinta", "Yang terhormat Bapak/Ibu").
- Format kartu ucapan dalam bingkai ASCII art (gunakan karakter seperti =, |, +, -) dengan minimal lebar 50 karakter.

- Pesan personal di-wrap (dipecah jadi beberapa baris) jika lebih dari 40 karakter per baris—gunakan logika slicing dan split.
- Sertakan tanda tangan otomatis di akhir kartu dengan nama pengirim (input juga).

Format pengumpulan: berkas .py beserta tangkapan layar hasil eksekusi untuk minimal tiga skenario berbeda.

5.10 Refleksi Diri

Centang pernyataan berikut yang sudah Anda kuasai:

☐ Saya dapat menggunakan parameter sep, end, dan file pada fungsi print().

☐ Saya menguasai ketiga gaya formatting (%, .format(), f-string) dan tahu kapan menggunakan masing-masing.

☐ Saya tidak lupa mengonversi hasil input() ketika membutuhkan angka.

☐ Saya memahami konsep string sebagai urutan karakter terindeks dari 0.

☐ Saya dapat melakukan indexing dengan indeks positif maupun negatif.

☐ Saya menguasai slicing dengan notasi [start:stop:step] termasuk step negatif.

☐ Saya hafal minimal 10 metode string penting dan tahu kapan menggunakannya.

☐ Saya memahami konsep immutability dan implikasinya pada metode string.

☐ Saya dapat menggunakan escape sequences, raw string, dan string multi-baris dengan tepat.

> **ASAH OTAK 5.2**
>
> Sebuah teka-teki klasik: tanpa menggunakan .reverse() atau slicing [::-1], bagaimana Anda dapat membalik string menggunakan hanya operasi yang telah Anda pelajari di bab ini? Petunjuk: gunakan list(), pengulangan, dan .join(). Jika berhasil, bandingkan kecepatan eksekusi solusi Anda dengan solusi satu baris [::-1] menggunakan modul timeit. Renungkan: mengapa Python menyediakan banyak cara untuk melakukan hal yang sama?

BAB 6

STRUKTUR KONTROL PERCABANGAN

"Two roads diverged in a wood, and I—I took the one less traveled by."
— Robert Frost, dalam puisi The Road Not Taken (1916)

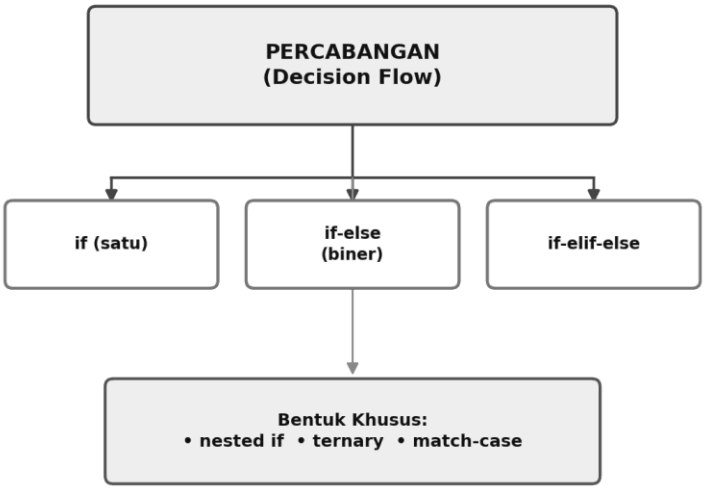
Capaian Pembelajaran Bab

Setelah mempelajari bab ini, mahasiswa diharapkan mampu mencapai sejumlah luaran pembelajaran sebagai berikut:

Kode	Capaian	Bloom
CPB-6.1	Menjelaskan konsep pengambilan keputusan dalam algoritma	C2
CPB-6.2	Menerapkan pernyataan if tunggal untuk kasus pengambilan keputusan sederhana	C3
CPB-6.3	Menerapkan pernyataan if-else untuk percabangan biner	C3
CPB-6.4	Menerapkan pernyataan if-elif-else untuk percabangan multi-cabang	C3
CPB-6.5	Menyusun percabangan bersarang dengan kedalaman yang aman	C3
CPB-6.6	Menggunakan ekspresi kondisional (ternary) untuk kasus yang sesuai	C3
CPB-6.7	Menerapkan pernyataan match-case untuk pencocokan pola sederhana	C3
CPB-6.8	Menganalisis dan memperbaiki pola percabangan yang buruk (code smell)	C4

6.1 Peta Konsep

Peta konsep berikut menggambarkan hirarki bentuk-bentuk percabangan dalam Python, dari yang paling sederhana hingga yang paling ekspresif.



6.2 Apersepsi

Setiap pagi, sejak Anda membuka mata, hidup adalah serangkaian keputusan. Apakah cuaca cerah? Jika ya, kenakan kaus tipis. Jika tidak, ambil jaket. Apakah sudah jam tujuh? Jika ya, percepat sarapan. Jika tidak, masih sempat menyeruput kopi. Apakah hujan? Jika ya, bawa payung. Jika tidak, abaikan. Begitu seterusnya, sepanjang hari, sepanjang hidup.

Hingga Bab 5, program-program yang kita tulis bersifat **sekuensial murni**: setiap baris dieksekusi satu per satu dari atas ke bawah, tanpa kecuali. Program seperti itu tidak dapat membuat keputusan; ia hanya mengikuti instruksi secara harfiah. Tetapi dunia nyata penuh dengan kondisi, dan program yang berguna harus dapat bereaksi terhadap kondisi tersebut.

Struktur kontrol percabangan memberi program kemampuan untuk bertindak berbeda berdasarkan kondisi tertentu. Inilah momen ketika program Anda berubah dari sekadar daftar perintah menjadi entitas yang dapat "berpikir" dalam batas-batas logika yang Anda definisikan. Setelah

menguasai bab ini, Anda telah memiliki separuh dari kekuatan dasar pemrograman; separuh lainnya, perulangan, akan kita pelajari di Bab 7.

> **ASAH OTAK 6.1**
>
> Tuliskan tiga keputusan yang Anda buat hari ini, lengkap dengan kondisi yang mendasarinya. Misalnya: "Jika hujan, saya membawa payung; jika tidak, tidak." Sekarang, perhatikan: berapa banyak dari keputusan itu yang melibatkan hanya satu kondisi (biner: ya/tidak), dan berapa yang melibatkan banyak kondisi sekaligus (multi-cabang)? Renungkan kompleksitas keputusan harian Anda sebelum melanjutkan ke materi formal.

6.3 Uraian Materi

6.3.1 Konsep Pengambilan Keputusan dalam Algoritma

Pengambilan keputusan dalam algoritma selalu mengikuti pola yang sama: ada sebuah **kondisi** yang dapat dievaluasi sebagai True atau False, dan ada satu atau lebih **aksi** yang akan dieksekusi berdasarkan hasil evaluasi tersebut. Secara konseptual:

```
JIKA kondisi BENAR maka:
    Lakukan aksi A
JIKA TIDAK:
    Lakukan aksi B
```

Di Bab 2 kita telah melihat bentuk pseudocode-nya: `if-then-else`. Dalam flowchart, percabangan digambarkan dengan simbol belah ketupat. Sekarang, di Python, struktur ini diterjemahkan menjadi sintaks konkret yang dapat dieksekusi. Kondisi sendiri adalah sebuah *ekspresi boolean* yang menghasilkan True atau False, baik melalui operator perbandingan (==, <, >, dst), operator logika (and, or, not), atau bahkan nilai langsung (truthy/falsy seperti yang dibahas di Bab 4).

> **CATATAN 6.1**
>
> Setiap percabangan secara teori dapat direduksi menjadi serangkaian if-else biner. Itulah sebabnya beberapa bahasa pemrograman fungsional bahkan tidak memiliki konstruksi if-elif-else multi-cabang—mereka mengandalkan rekursi atau pattern matching. Namun, untuk keterbacaan kode, kebanyakan bahasa modern menyediakan berbagai bentuk percabangan yang lebih ekspresif. Python termasuk salah satu yang paling lengkap.

6.3.2 Pernyataan if Tunggal

Bentuk paling sederhana percabangan adalah pernyataan `if` tunggal. Sintaksnya:

```
if kondisi:
    aksi
```

Perhatikan tiga elemen penting: kata kunci `if`, titik dua `:` di akhir baris kondisi, dan *indentasi* (4 spasi) pada baris aksi. Indentasi inilah yang memberi tahu Python bahwa baris tersebut adalah bagian dari blok if.

```
# Contoh sederhana
nilai = 85

if nilai >= 60:
    print("Anda lulus!")

print("Program selesai.")  # ini selalu dieksekusi (di luar
if)
```

Beberapa kondisi yang sering muncul dalam praktik:

```
# Perbandingan numerik
if umur >= 17:
    print("Boleh memilih dalam pemilu.")
```

```
# Perbandingan string
if password == "rahasia123":
    print("Login berhasil.")

# Kombinasi dengan operator logika
if umur >= 17 and punya_ktp:
    print("Boleh memilih.")

# Cek keanggotaan
if huruf in "aiueo":
    print("Huruf vokal.")

# Truthy/falsy
if nama:                    # benar jika string nama tidak
kosong
    print(f"Halo, {nama}!")
```

PERHATIAN 6.1

Jangan pernah menggunakan tanda = (penugasan) di dalam kondisi if. Python 3 akan menolaknya dengan SyntaxError, tetapi kesalahan ini masih sering dialami pemula yang berasal dari pengalaman matematika. Selalu gunakan == untuk perbandingan kesamaan. Aturan praktis: jika Anda membaca kode di kepala Anda sebagai "apakah…?", gunakan ==; jika sebagai "set X menjadi…", gunakan =.

6.3.3 Pernyataan if-else

Ketika kita ingin mengeksekusi aksi alternatif jika kondisi tidak terpenuhi, kita menambahkan blok `else`:

```
if kondisi:
    aksi-jika-benar
else:
    aksi-jika-salah
```

Contoh konkret: program penentuan ganjil-genap yang telah kita lihat di Bab 2, kini ditulis dalam Python:

```
bilangan = int(input("Masukkan bilangan bulat: "))

if bilangan % 2 == 0:
    print(f"{bilangan} adalah bilangan genap.")
else:
    print(f"{bilangan} adalah bilangan ganjil.")
```

Struktur if-else menjamin tepat satu dari dua blok akan dieksekusi. Jika kondisi True, hanya blok if yang berjalan; jika False, hanya blok else yang berjalan. Tidak pernah keduanya, dan tidak pernah tidak satu pun.

Contoh: Diskon Pembelian

```
total = float(input("Total belanja (Rp): "))

if total >= 200000:
    diskon = total * 0.10
    print(f"Selamat! Anda mendapat diskon Rp{diskon:,.0f}")
    total_bayar = total - diskon
else:
    print("Belanja kurang dari Rp200.000, tidak ada diskon.")
    total_bayar = total

print(f"Total bayar: Rp{total_bayar:,.0f}")
```

6.3.4 Pernyataan if-elif-else (Multi-cabang)

Ketika ada lebih dari dua kemungkinan, kita menggunakan `elif` (singkatan dari "else if"). Anda dapat memiliki sebanyak mungkin elif, tetapi hanya satu else di akhir (opsional).

```
if kondisi-1:
    aksi-1
elif kondisi-2:
    aksi-2
elif kondisi-3:
    aksi-3
else:
    aksi-default
```

Python mengevaluasi kondisi dari atas ke bawah. Begitu satu kondisi terpenuhi (True), blok terkait dieksekusi dan seluruh struktur diakhiri—kondisi-kondisi berikutnya tidak diperiksa. Inilah sebabnya **urutan kondisi sangat penting**.

Contoh: Konversi Nilai Angka ke Huruf

```
nilai = float(input("Masukkan nilai (0-100): "))

if nilai >= 85:
    huruf = "A"
elif nilai >= 75:
    huruf = "B"
elif nilai >= 65:
    huruf = "C"
elif nilai >= 55:
    huruf = "D"
else:
    huruf = "E"

print(f"Nilai {nilai} setara dengan huruf {huruf}")
```

Perhatikan: setiap kondisi hanya memeriksa batas bawah, bukan rentang lengkap. Mengapa? Karena Python sudah menjamin bahwa jika kita sampai pada baris `elif nilai >= 75`, berarti kondisi sebelumnya (nilai >= 85) sudah False. Jadi nilai pasti < 85. Kita tidak perlu menulis ulang batas atasnya.

> **TIPS PRO 6.1**
>
> Urutan kondisi dalam if-elif-else berpengaruh pada efisiensi dan kebenaran. Letakkan kondisi yang paling sering terjadi di atas untuk performa terbaik. Letakkan kondisi yang paling spesifik sebelum yang umum, atau Anda akan menulis bug. Contoh bug umum: menulis if x > 0 sebelum if x > 100; kondisi pertama akan selalu mengeksekusi duluan, kondisi kedua tidak akan pernah tercapai. Aturan praktis: dari spesifik ke umum, dari sering ke jarang.

6.3.5 Percabangan Bersarang (Nested if)

Setiap blok if, elif, atau else dapat memuat percabangan lain di dalamnya. Inilah yang disebut *nested if* (if bersarang). Indentasi membuatnya jelas: setiap level bersarang ditambahkan 4 spasi indentasi.

```
umur = int(input("Umur: "))
punya_sim = input("Punya SIM? (ya/tidak): ").lower() == "ya"

if umur >= 17:
    print("Cukup umur untuk mengemudi.")
    if punya_sim:
        print("Silakan mengemudi dengan aman.")
    else:
        print("Buat SIM dulu sebelum mengemudi.")
else:
    print("Belum cukup umur untuk mengemudi.")
```

Walaupun bersarang adalah teknik yang valid, **kedalaman bersarang yang berlebihan adalah tanda bahwa logika perlu dipikir ulang**. Aturan praktis dari pengembang berpengalaman: jangan melewati **tiga level bersarang**. Jika Anda mencapai empat atau lima level, kode menjadi sulit dibaca dan rawan bug. Ada beberapa teknik untuk menghindarinya:

- **Gabungkan kondisi dengan operator logika.** Daripada if A: if B: ...; tulis if A and B: ...
- **Gunakan early return atau guard clauses.** Periksa kondisi negatif lebih dulu dan keluar, sehingga blok utama tidak perlu diindentasi.
- **Pecah menjadi fungsi-fungsi terpisah.** Setiap fungsi menangani satu tingkat logika (akan dibahas di Bab 11).

6.3.6 Ekspresi Kondisional (Ternary)

Python menyediakan bentuk ringkas if-else dalam satu baris, yang disebut **ekspresi kondisional** atau *ternary expression*. Sintaksnya:

```
nilai_jika_benar if kondisi else nilai_jika_salah
```

Berbeda dengan if-else biasa yang merupakan statement (tidak menghasilkan nilai), ternary adalah expression yang dapat ditugaskan ke variabel atau dipakai langsung dalam ekspresi lain.

```
# Cara lama dengan if-else biasa
umur = 20
if umur >= 17:
    status = "dewasa"
else:
    status = "anak-anak"

# Cara ringkas dengan ternary
status = "dewasa" if umur >= 17 else "anak-anak"
print(status)

# Bisa langsung di dalam f-string
print(f"Anda {'dewasa' if umur >= 17 else 'anak-anak'}.")

# Penggunaan praktis: nilai default
nama = input("Nama: ")
sapaan = nama if nama else "Tamu"
print(f"Halo, {sapaan}!")
```

TIPS PRO 6.2

Ternary expression sangat berguna untuk ekspresi sederhana, tetapi jangan menyalahgunakannya. Ternary yang bersarang seperti a if x else (b if y else c) sulit dibaca. Aturan praktis: jika logika sudah memuat lebih dari satu kondisi atau aksi yang kompleks, kembalilah ke if-else biasa. Pythonic code selalu mengutamakan keterbacaan.

6.3.7 Pernyataan match-case (Python 3.10+)

Sejak Python 3.10, tersedia struktur **match-case** yang melakukan pencocokan pola (*pattern matching*). Strukturnya mirip switch-case di bahasa lain, tetapi jauh lebih kuat karena mendukung pencocokan struktural pada tipe data kompleks.

```
match ekspresi:
    case pola_1:
        aksi_1
    case pola_2:
        aksi_2
    case _:                # _ adalah wildcard (default)
        aksi_default
```

Contoh: Konversi Kode Hari ke Nama Hari

```
hari_ke = int(input("Hari ke-? (1-7): "))

match hari_ke:
    case 1:
        nama_hari = "Senin"
    case 2:
        nama_hari = "Selasa"
    case 3:
        nama_hari = "Rabu"
    case 4:
        nama_hari = "Kamis"
    case 5:
        nama_hari = "Jumat"
    case 6 | 7:                  # gabungan pola: | berarti
atau
        nama_hari = "Akhir Pekan"
    case _:
        nama_hari = "Tidak valid"

print(f"Hari ke-{hari_ke} adalah {nama_hari}.")
```

match-case sangat unggul ketika Anda membandingkan satu variabel dengan banyak kemungkinan nilai. Bandingkan kode di atas dengan versi if-elif-else: match-case lebih ringkas dan langsung menyampaikan niat penulis (“saya sedang mencocokkan satu nilai dengan beberapa kemungkinan”).

> **DI BALIK LAYAR 6.1**
>
> Pattern matching di Python tidak sebatas pencocokan nilai literal. Ia dapat mencocokkan struktur data seperti tuple, list, dan bahkan kelas.

> Contoh: case (x, y) cocok untuk tuple dua elemen, dan x serta y akan ter-bind ke nilainya. Kemampuan ini terinspirasi dari bahasa pemrograman fungsional seperti Haskell dan Scala. Untuk semester awal, kita cukup menggunakan match-case untuk pencocokan nilai sederhana; pencocokan struktural akan menjadi materi lanjutan.

6.3.8 Pola dan Anti-Pola Percabangan

Kode yang berfungsi dengan benar belum tentu kode yang baik. Berikut beberapa **anti-pola** (*code smell*) dalam penulisan percabangan yang sebaiknya Anda hindari:

Anti-Pola 1: Indentasi Terlalu Dalam

```
# BURUK: nested 4 level
if user_terdaftar:
    if user_aktif:
        if punya_saldo:
            if jumlah <= saldo:
                proses_transaksi()

# BAIK: gunakan guard clauses dengan return atau
# gabungkan kondisi dengan operator logika
if user_terdaftar and user_aktif and punya_saldo and jumlah <=
saldo:
    proses_transaksi()
```

Anti-Pola 2: Membandingkan Boolean dengan True/False

```
# BURUK: redundan
if lulus == True:
    print("Selamat")

# BAIK: variabel boolean langsung dipakai
if lulus:
    print("Selamat")

# Untuk negasi:
```

```
if not lulus:
    print("Tetap semangat")
```

Anti-Pola 3: Kondisi yang Tidak Pernah Tercapai (Dead Code)

```
# BURUK: kondisi kedua tidak akan pernah dievaluasi
if nilai >= 0:
    print("Tidak negatif")
elif nilai >= 50:           # tidak pernah tercapai!
    print("Lulus")          # karena nilai >= 50 implies nilai
>= 0

# BAIK: urutan dari yang paling spesifik ke yang paling umum
if nilai >= 50:
    print("Lulus")
elif nilai >= 0:
    print("Tidak lulus")
else:
    print("Nilai tidak valid")
```

Anti-Pola 4: Pengulangan Kondisi yang Sama

```
# BURUK: kondisi diperiksa berulang kali
if umur >= 17:
    print("Dewasa")
if umur >= 17:                # diperiksa lagi tanpa perlu
    boleh_pilih = True
if umur >= 17:
    boleh_punya_ktp = True

# BAIK: satu blok if
if umur >= 17:
    print("Dewasa")
    boleh_pilih = True
    boleh_punya_ktp = True
```

WAWASAN INDUSTRI 6.1

Di proyek profesional, kompleksitas percabangan diukur dengan metrik bernama cyclomatic complexity. Setiap if, elif, dan operator logika menambah satu unit kompleksitas. Banyak tim mensyaratkan kompleksitas siklomatik sebuah fungsi tidak melebihi 10. Alat seperti pylint, radon, dan SonarQube otomatis mengukur dan melaporkan ini. Praktik menulis percabangan yang bersih sejak Bab 6 akan membentuk kebiasaan yang berguna sepanjang karier Anda.

6.4 Contoh Kode Lengkap

Skrip berikut menggabungkan semua bentuk percabangan yang telah kita pelajari: *Sistem Penilaian Akademik Otomatis*. Program menerima nilai dan kehadiran, lalu menentukan kelulusan beserta kategorinya.

```
"""
Program: Sistem Penilaian Akademik Otomatis
Tujuan : Menentukan status kelulusan dan kategori prestasi
         mahasiswa berdasarkan nilai dan kehadiran
"""

# Konstanta
BATAS_LULUS = 60
MIN_KEHADIRAN = 75

# Input dari pengguna
print("=" * 50)
print("   SISTEM PENILAIAN AKADEMIK   ")
print("=" * 50)
nama = input("Nama mahasiswa: ").strip().title()
nilai = float(input("Nilai (0-100)  : "))
hadir = float(input("Kehadiran (%)  : "))

# Tahap 1: Validasi input
if not (0 <= nilai <= 100) or not (0 <= hadir <= 100):
    print("\nERROR: Nilai dan kehadiran harus antara 0-100.")
else:
    # Tahap 2: Tentukan kelulusan
    lulus = nilai >= BATAS_LULUS and hadir >= MIN_KEHADIRAN
```

```
    # Tahap 3: Tentukan huruf nilai (multi-cabang)
    if nilai >= 85:
        huruf = "A"
    elif nilai >= 75:
        huruf = "B"
    elif nilai >= 65:
        huruf = "C"
    elif nilai >= 55:
        huruf = "D"
    else:
        huruf = "E"

    # Tahap 4: Tentukan kategori prestasi (match-case)
    match huruf:
        case "A":
            kategori = "Sangat Memuaskan"
        case "B":
            kategori = "Memuaskan"
        case "C":
            kategori = "Cukup"
        case "D" | "E":
            kategori = "Perlu Perbaikan"

    # Tahap 5: Pesan motivasi (ternary)
    pesan = "Pertahankan!" if lulus else "Tetap semangat di
kesempatan berikutnya."

    # Tampilkan hasil
    print()
    print("-" * 50)
    print(f"Nama            : {nama}")
    print(f"Nilai Angka     : {nilai:.1f}")
    print(f"Kehadiran       : {hadir:.1f}%")
    print(f"Nilai Huruf     : {huruf}")
    print(f"Kategori        : {kategori}")
    print(f"Status          : {'LULUS' if lulus else 'TIDAK
LULUS'}")
    print(f"Pesan           : {pesan}")
    print("-" * 50)
```

Contoh eksekusi:

```
==================================================
   SISTEM PENILAIAN AKADEMIK
==================================================
Nama mahasiswa: yusri ikhwani
Nilai (0-100)  : 82
Kehadiran (%)  : 90

--------------------------------------------------
Nama            : Yusri Ikhwani
Nilai Angka     : 82.0
Kehadiran       : 90.0%
Nilai Huruf     : B
Kategori        : Memuaskan
Status          : LULUS
Pesan           : Pertahankan!
--------------------------------------------------
```

Skrip ini mendemonstrasikan: validasi dengan if-else, kondisi gabungan dengan operator logika and, pengelompokan multi-cabang dengan if-elif-else, pencocokan pola dengan match-case (termasuk operator | untuk multi-pola), dan ekspresi ternary di dalam f-string. Inilah cara percabangan dipakai dalam aplikasi nyata.

6.5 Kesalahan Umum dan Cara Mengatasinya

Kesalahan 1: Lupa titik dua (:) di akhir baris if.

Miskonsepsi: "Saya tulis if x > 0 print(x) tapi error."

Realita: Setiap pernyataan if, elif, else, match, dan case harus diakhiri dengan titik dua (:). Tanpa titik dua, Python tidak tahu di mana kondisi berakhir dan blok dimulai. Python akan memberikan SyntaxError yang menunjuk lokasi kesalahan.

Kesalahan 2: Indentasi tidak konsisten.

Miskonsepsi: "Kenapa Python protes padahal logikanya benar?"

Realita: Python sangat ketat tentang indentasi. Semua baris dalam satu blok harus memiliki indentasi yang SAMA PERSIS. Mencampur 2 spasi dan 4 spasi,

atau spasi dan tab, akan menyebabkan IndentationError atau TabError. Atur editor Anda untuk konsisten menggunakan 4 spasi.

Kesalahan 3: Mengira else cocok dengan if mana saja di atasnya.

Miskonsepsi: "Kenapa else saya tidak tereksekusi padahal kondisinya jelas false?"

Realita: else selalu cocok dengan if/elif TERAKHIR yang sejajar dengannya secara indentasi. Dalam nested if, periksa hati-hati: else berpasangan dengan if pada level yang sama, bukan dengan if terjauh.

Kesalahan 4: Urutan kondisi yang salah.

Miskonsepsi: "Kenapa nilai 95 saya dapat huruf B, bukan A?"

Realita: Python mengevaluasi if-elif dari atas ke bawah dan berhenti di kondisi pertama yang true. Jika Anda menulis if nilai >= 75: B elif nilai >= 85: A, maka nilai 95 cocok dengan kondisi pertama (>= 75) dan dapat B. Solusi: letakkan kondisi paling spesifik (paling tinggi) di atas: if nilai >= 85: A elif nilai >= 75: B.

Kesalahan 5: Menggunakan match-case di Python versi lama.

Miskonsepsi: "Kenapa match-case tidak dikenali?"

Realita: match-case baru tersedia di Python 3.10 (Oktober 2021) ke atas. Periksa versi Python Anda dengan python --version. Jika versinya 3.9 atau lebih lama, gunakan if-elif-else sebagai gantinya, atau perbarui Python Anda.

6.6 Rangkuman

Bab 6 telah memperkenalkan struktur kontrol percabangan, kemampuan fundamental yang mengubah program dari sekuensial murni menjadi adaptif terhadap kondisi. Bentuk paling sederhana adalah if tunggal, yang menjalankan blok hanya jika kondisi True. if-else menambahkan jalur alternatif untuk kondisi False, sedangkan if-elif-else menangani percabangan multi-cabang dengan urutan evaluasi dari atas ke bawah.

Percabangan bersarang (nested if) memungkinkan logika berlapis, tetapi kedalaman yang berlebihan adalah tanda kode perlu refactoring. Ekspresi kondisional (ternary) menyediakan bentuk ringkas if-else dalam satu baris yang dapat ditugaskan ke variabel atau dipakai dalam ekspresi. Pernyataan

match-case (Python 3.10+) menawarkan pencocokan pola yang sangat ekspresif untuk membandingkan satu nilai dengan banyak kemungkinan.

Selain sintaks, kita telah membahas anti-pola yang harus dihindari: indentasi terlalu dalam, perbandingan boolean dengan True/False yang redundan, kondisi yang tidak pernah tercapai, dan pengulangan kondisi yang sama. Praktik menulis percabangan yang bersih sejak awal akan membentuk kebiasaan yang berguna sepanjang karier sebagai programmer.

6.7 Latihan Terbimbing

Soal 6.7.1

Soal: Tulis program yang menerima sebuah bilangan dan menampilkan apakah ia positif, negatif, atau nol.

Pembahasan:

```
x = float(input("Bilangan: "))

if x > 0:
    print("Positif")
elif x < 0:
    print("Negatif")
else:
    print("Nol")
```

Soal 6.7.2

Soal: Apa output dari kode berikut, dan mengapa? x = 10; y = 20; print("besar" if x > y else "kecil" if x < y else "sama")

Pembahasan: Ini adalah ternary bersarang. Python mengevaluasi dari kiri: pertama 'x > y' (10 > 20) = False, jadi tidak ambil 'besar'. Lanjut ke 'x < y' (10 < 20) = True, jadi ambil 'kecil'. Output: 'kecil'. Catatan: ternary bersarang seperti ini sulit dibaca; kode if-elif-else biasa akan lebih jelas.

Soal 6.7.3

Soal: Tulis program untuk menentukan apakah suatu tahun adalah tahun kabisat. Aturan: tahun kabisat jika habis dibagi 4 DAN tidak habis dibagi 100, ATAU habis dibagi 400.

Pembahasan:

```
tahun = int(input("Tahun: "))

if (tahun % 4 == 0 and tahun % 100 != 0) or (tahun % 400 ==
0):
    print(f"{tahun} adalah tahun kabisat.")
else:
    print(f"{tahun} bukan tahun kabisat.")

# Contoh: 2024 → kabisat, 1900 → bukan, 2000 → kabisat, 2025 →
bukan
```

Soal 6.7.4

Soal: Identifikasi anti-pola dalam kode berikut dan perbaiki: if x > 0: if x < 100: if x % 2 == 0: print("valid")

Pembahasan:

```
# Anti-pola: nested 3 level yang tidak perlu
# Perbaikan: gabungkan dengan operator and

if x > 0 and x < 100 and x % 2 == 0:
    print("valid")

# Bahkan lebih Pythonic dengan rantai perbandingan:
if 0 < x < 100 and x % 2 == 0:
    print("valid")
```

Soal 6.7.5

Soal: Tulis program konversi nilai huruf ke nilai angka dengan match-case. Contoh: A → 4.0, B → 3.0, C → 2.0, D → 1.0, E → 0.0, lainnya → error.

Pembahasan:

```
huruf = input("Nilai huruf: ").upper()

match huruf:
    case "A":
        angka = 4.0
    case "B":
        angka = 3.0
    case "C":
        angka = 2.0
    case "D":
        angka = 1.0
    case "E":
        angka = 0.0
    case _:
        angka = None

if angka is None:
    print("Nilai huruf tidak valid.")
else:
    print(f"{huruf} = {angka:.1f}")
```

6.8 Latihan Mandiri

- **(C2)** Jelaskan perbedaan antara if-else dan if-elif-else dalam hal eksekusi blok.
- **(C2)** Apa kelebihan match-case dibanding if-elif-else? Pada kondisi seperti apa match-case lebih cocok?
- **(C3)** Tulis program yang menerima 3 bilangan dan menampilkan bilangan terbesar.
- **(C3)** Buatlah kalkulator sederhana yang menerima 2 bilangan dan satu operator (+, -, *, /) lalu menampilkan hasilnya. Tangani pembagian dengan nol.
- **(C3)** Tulis program yang menentukan kategori IMT (Indeks Massa Tubuh): kurus (<18.5), normal (18.5-24.9), gemuk (25-29.9), obesitas (≥30).
- **(C3)** Susun program penentu jenis segitiga (sama sisi, sama kaki, sembarang) berdasarkan tiga panjang sisi.
- **(C4)** Analisis kode berikut dan identifikasi anti-pola: if a == True: x = 1 elif a == False: x = 0. Tulis versi yang lebih Pythonic.

- **(C4)** Bandingkan: kapan sebaiknya menggunakan ekspresi ternary, kapan if-else biasa, dan kapan match-case? Berikan contoh kasus untuk masing-masing.
- **(C4)** Studi kasus: tulis program yang menentukan biaya parkir di pusat perbelanjaan dengan aturan: jam pertama Rp3000, jam selanjutnya Rp2000, maksimum Rp25000 per hari, mobil VIP dapat diskon 50%.
- **(C4)** Diskusikan: dalam praktik profesional, mengapa kompleksitas siklomatik diukur dan dibatasi? Bagaimana cara mengurangi kompleksitas tanpa mengubah perilaku program?

6.9 Tugas Proyek Mini

"Sistem Pemesanan Tiket Bioskop"

Buatlah skrip Python yang mensimulasikan sistem pemesanan tiket bioskop dengan aturan harga sebagai berikut:

- Hari kerja (Senin-Kamis): Rp35.000 (dewasa), Rp25.000 (anak <12 tahun)
- Akhir pekan (Jumat-Minggu): Rp50.000 (dewasa), Rp40.000 (anak <12 tahun)
- Mahasiswa (dengan KTM) mendapat diskon 20% (semua jenis tiket)
- Pemesanan ≥ 5 tiket mendapat diskon tambahan 10% dari subtotal
- Studio premium menambah Rp15.000 per tiket

Persyaratan teknis:

- Skrip meminta input: hari nonton (Senin-Minggu), kategori penonton (dewasa/anak), apakah mahasiswa (y/n), jumlah tiket, studio (regular/premium).
- Gunakan kombinasi if-elif-else, ternary, dan minimal satu match-case.
- Validasi semua input; jika tidak valid, tampilkan pesan kesalahan.
- Tampilkan rincian: subtotal, diskon mahasiswa, diskon kuantitas, total akhir, dalam format yang rapi.
- Sertakan minimal 3 skenario pengujian yang berbeda.

Format pengumpulan: berkas .py beserta tangkapan layar dari 3 skenario pengujian.

6.10 Refleksi Diri

Centang pernyataan berikut yang sudah Anda kuasai:

☐ Saya memahami konsep pengambilan keputusan dalam algoritma.

☐ Saya dapat menulis pernyataan if tunggal dengan sintaks yang benar (titik dua, indentasi).

☐ Saya dapat menggunakan if-else untuk percabangan biner.

☐ Saya dapat menyusun if-elif-else dengan urutan kondisi yang tepat.

☐ Saya menyadari bahaya percabangan bersarang yang terlalu dalam.

☐ Saya dapat menggunakan ekspresi ternary untuk kasus sederhana.

☐ Saya menguasai sintaks dasar match-case dan tahu kapan ia lebih cocok dari if-elif-else.

☐ Saya dapat mengenali anti-pola percabangan dan tahu cara memperbaikinya.

☐ Saya memahami pentingnya keterbacaan kode di atas kepintaran sintaks.

ASAH OTAK 6.2

Eksperimen kecil: tulis fungsi yang sama dalam tiga gaya berbeda (if-elif-else, ternary bersarang, dan match-case), lalu mintalah teman Anda membacanya. Tanyakan mana yang paling mudah dipahami dalam waktu 10 detik. Hasilnya seringkali mengejutkan: kode yang terasa "pintar" bagi penulisnya belum tentu jelas bagi pembaca. Renungkan bagaimana ini membentuk cara Anda menulis kode di masa depan.

BAB 7

STRUKTUR KONTROL PERULANGAN

"Repetition makes reputation, and reputation makes customers."
— Elizabeth Arden — dan dalam pemrograman, repetisi juga membuat program berdaya guna

---

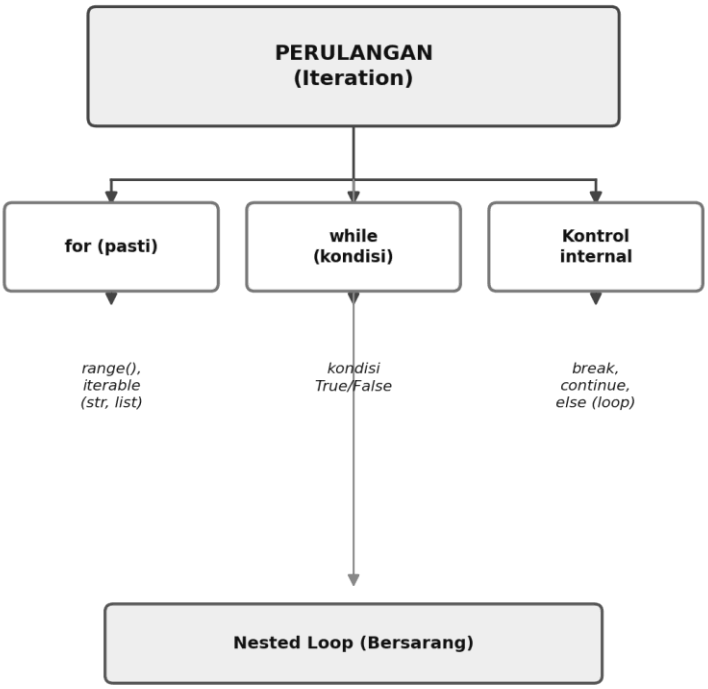
Capaian Pembelajaran Bab

Setelah mempelajari bab ini, mahasiswa diharapkan mampu mencapai sejumlah luaran pembelajaran sebagai berikut:

Kode	Capaian	Bloom
CPB-7.1	Menjelaskan konsep iterasi dan pengulangan terkendali	C2
CPB-7.2	Menerapkan perulangan for dengan fungsi range()	C3
CPB-7.3	Menggunakan perulangan for pada berbagai jenis iterable	C3
CPB-7.4	Menerapkan perulangan while dengan kondisi pemberhentian yang tepat	C3
CPB-7.5	Memilih antara for dan while sesuai konteks masalah	C4
CPB-7.6	Menerapkan pernyataan break, continue, dan else pada loop	C3
CPB-7.7	Menyusun perulangan bersarang untuk kasus dua dimensi	C3
CPB-7.8	Melakukan tracing eksekusi perulangan secara manual untuk verifikasi	C4

7.1 Peta Konsep

Peta konsep berikut menggambarkan keterkaitan antara berbagai bentuk perulangan dan elemen pengendalinya.



7.2 Apersepsi

Pikirkan kegiatan-kegiatan repetitif dalam hidup Anda. Setiap pagi Anda menyikat gigi dengan gerakan yang sama, berulang-ulang. Setiap minggu Anda menerima jadwal kuliah yang memuat hari yang sama. Saat menyetel ulang oven microwave, Anda menekan tombol angka satu per satu hingga angka yang diinginkan tercapai. Jantung Anda berdetak, sekitar 100.000 kali per hari, tanpa pernah lelah. Hidup dipenuhi pengulangan.

Pemrograman pun demikian. Komputer adalah mesin yang sangat baik dalam melakukan satu hal: **mengulang sesuatu dengan cepat dan tanpa salah**. Apa yang dilakukan manusia dalam beberapa menit, komputer melakukannya

dalam mikrodetik. Inilah kekuatan utama komputer atas manusia. Tetapi untuk memanfaatkan kekuatan ini, kita harus tahu cara memberitahu komputer apa yang harus diulang, berapa kali, dan kapan berhenti.

Struktur kontrol perulangan adalah jawabannya. Di bab ini, kita akan belajar dua bentuk perulangan utama di Python: `for` untuk pengulangan dengan jumlah pasti, dan `while` untuk pengulangan berdasarkan kondisi. Bersama dengan percabangan dari Bab 6, perulangan melengkapi fondasi pemrograman prosedural. Setelah bab ini, Anda dapat menulis program yang dapat memproses data dalam jumlah besar, mensimulasikan proses berulang, dan berinteraksi dengan pengguna sepanjang waktu yang dibutuhkan.

ASAH OTAK 7.1

Bayangkan Anda harus mencetak "Halo" ke layar sebanyak 1000 kali. Tanpa menggunakan perulangan, berapa baris kode yang dibutuhkan? Sekarang bayangkan tanpa perulangan, Anda harus mencetak "Halo" sebanyak N kali, di mana N adalah masukan pengguna saat runtime. Apakah ini mungkin? Renungkan: konsep perulangan bukan hanya soal kemudahan, tetapi soal kelayakan.

7.3 Uraian Materi

7.3.1 Konsep Iterasi dan Pengulangan Terkendali

Iterasi adalah istilah teknis untuk satu siklus pengulangan. Jika kita mengulang sesuatu lima kali, kita melakukan lima iterasi. Setiap iterasi mengeksekusi blok kode yang sama (atau hampir sama), seringkali dengan satu nilai yang berubah-ubah (disebut *variabel iterasi*).

Sebuah perulangan yang baik memiliki tiga komponen:

- **Inisialisasi.** Penyiapan kondisi awal, misalnya pencacah dimulai dari nol.
- **Kondisi atau iterabel.** Aturan yang menentukan kapan perulangan berhenti.
- **Pembaruan.** Perubahan yang membawa perulangan menuju kondisi berhenti.

Pada perulangan `for`, ketiga komponen ini terkelola secara otomatis oleh Python. Pada perulangan `while`, programmer-lah yang harus mengelolanya secara manual—dan inilah sumber bug paling umum: *lupa memperbarui kondisi*, yang menyebabkan loop tak berhingga.

> **CATATAN 7.1**
>
> Loop tak berhingga (infinite loop) adalah kondisi ketika program terus mengulang tanpa pernah berhenti, biasanya karena programmer lupa memperbarui variabel kondisi. Gejalanya: program tampak "macet", terminal tidak responsif, dan CPU bekerja 100%. Untuk menghentikannya di terminal, tekan Ctrl+C (Linux/Mac) atau Ctrl+Break (Windows). Jangan panik—ini bagian normal dari proses belajar.

7.3.2 Perulangan for dengan range()

Perulangan `for` digunakan ketika kita tahu (atau dapat menghitung) berapa kali kita ingin mengulang. Bentuk paling umum di Python menggunakan fungsi `range()` yang menghasilkan urutan bilangan.

```
# Sintaks dasar
for variabel in range(n):
    aksi
```

Fungsi range() dapat dipanggil dengan satu, dua, atau tiga argumen:

Pemanggilan	Menghasilkan	Contoh nilai
range(n)	0, 1, 2, ..., n-1	range(5) → 0,1,2,3,4
range(a, b)	a, a+1, ..., b-1	range(2, 7) → 2,3,4,5,6
range(a, b, s)	a, a+s, a+2s, ... hingga sebelum b	range(0, 10, 2) → 0,2,4,6,8
range(b, a, -s)	mundur dengan step negatif	range(5, 0, -1) → 5,4,3,2,1

Contoh penggunaan range() dalam berbagai variasi:

```
# Mencetak angka 1 sampai 5
for i in range(1, 6):
    print(i, end=" ")
# Output: 1 2 3 4 5

# Menjumlahkan 1 hingga 100
total = 0
for i in range(1, 101):
    total += i
print(total)            # 5050

# Bilangan genap dari 0 sampai 20
for i in range(0, 21, 2):
    print(i, end=" ")
# Output: 0 2 4 6 8 10 12 14 16 18 20

# Hitung mundur dari 10 ke 1
for i in range(10, 0, -1):
    print(i, end=" ")
print("\nMeluncur!")
# Output: 10 9 8 7 6 5 4 3 2 1
#         Meluncur!
```

PERHATIAN 7.1

range(a, b) berhenti SEBELUM b, bukan SETELAH b. Ini sumber kesalahan paling umum bagi pemula. Jika Anda ingin mengulang dari 1 sampai 10 (termasuk 10), tulis range(1, 11), bukan range(1, 10). Pola ini disebut half-open interval dan konsisten dengan banyak konvensi pemrograman lainnya.

7.3.3 Perulangan for pada Iterable

Selain range(), perulangan `for` di Python dapat beroperasi pada *iterable* apa pun: string, list, tuple, set, dictionary, file, bahkan generator. Inilah salah satu kekuatan terbesar Python.

Iterasi pada String

```
kata = "Python"

for huruf in kata:
    print(huruf, end="-")
# Output: P-y-t-h-o-n-

# Menghitung vokal
kalimat = "Saya belajar Python"
vokal = 0
for huruf in kalimat.lower():
    if huruf in "aiueo":
        vokal += 1
print(f"Jumlah vokal: {vokal}")
```

Iterasi pada List

```
nilai = [78, 85, 92, 67, 88]

# Cara biasa: nilai langsung
for n in nilai:
    print(n, end=" ")
# Output: 78 85 92 67 88

# Jika butuh indeks juga, gunakan enumerate()
for i, n in enumerate(nilai):
    print(f"Indeks {i}: {n}")

# Output:
# Indeks 0: 78
# Indeks 1: 85
# ...

# Mencari nilai tertinggi
tertinggi = nilai[0]
for n in nilai:
    if n > tertinggi:
        tertinggi = n
print(f"Nilai tertinggi: {tertinggi}")
```

> **TIPS PRO 7.1**
>
> Pola pythonic untuk iterasi dengan indeks adalah enumerate(), bukan for i in range(len(list)). Tulisan for i, item in enumerate(data): lebih ringkas dan lebih ekspresif. Jika Anda butuh memulai indeks dari angka selain 0, gunakan enumerate(data, start=1). List dan koleksi lainnya akan dibahas tuntas di Bab 9; di sini cukup paham bahwa for dapat beriterasi pada apa saja yang "dapat dilalui".

7.3.4 Perulangan while

Perulangan `while` digunakan ketika kita tidak tahu sebelumnya berapa kali kita harus mengulang. Yang kita tahu hanyalah **kondisi** yang harus dipenuhi agar perulangan terus berjalan.

```
# Sintaks dasar
while kondisi:
    aksi
    pembaruan-kondisi
```

Contoh 1: Pencacah Sederhana

```
n = 1
while n <= 5:
    print(n)
    n += 1            # JANGAN LUPA UPDATE!
# Output: 1 2 3 4 5
```

Contoh 2: Input Berulang sampai Valid

```
# Klasik: minta password sampai benar
password_benar = "rahasia123"
percobaan = 0

while True:                                    # loop "sengaja
tak berhingga"
    password = input("Masukkan password: ")
```

```
    percobaan += 1
    if password == password_benar:
        print("Akses diberikan.")
        break                                   # keluar dari
loop
    elif percobaan >= 3:
        print("Akun dikunci.")
        break
    else:
        print("Salah, coba lagi.")
```

Contoh 3: Algoritma Iteratif (Euclidean GCD)

```
# Mencari FPB dua bilangan dengan algoritma Euclidean
a = int(input("Bilangan 1: "))
b = int(input("Bilangan 2: "))

while b != 0:
    a, b = b, a % b           # tuple unpacking: tukar nilai

print(f"FPB = {a}")
```

> **DI BALIK LAYAR 7.1**
>
> Algoritma Euclidean di atas adalah salah satu algoritma tertua yang masih digunakan, dirumuskan oleh matematikawan Yunani Euclid sekitar tahun 300 SM. Ide intinya: FPB(a, b) = FPB(b, a mod b), dengan kasus dasar b = 0 yang menghasilkan a. Jumlah iterasi yang diperlukan terbukti tidak pernah melebihi 5 kali jumlah digit bilangan terkecil (teorema Lamé). Inilah keindahan algoritma: efisiensi yang dapat dianalisis secara matematis.

7.3.5 Memilih antara for dan while

Tabel berikut merangkum kapan menggunakan masing-masing bentuk perulangan:

Gunakan for ketika...	Gunakan while ketika...
Anda tahu berapa kali mengulang (atau dapat menghitungnya)	Jumlah iterasi tidak dapat ditentukan di awal
Anda beriterasi pada elemen koleksi (list, string, dll)	Anda menunggu kondisi tertentu (input pengguna, dst)
Anda bekerja dengan urutan numerik tertentu (1 sampai N)	Anda mengimplementasikan algoritma iteratif (mis. Euclidean, Newton)
Risiko loop tak berhingga rendah karena terbatasi range	Anda siap mengelola kondisi berhenti secara manual

Aturan praktis: **lebih aman pakai for ketika memungkinkan**. Perulangan for dengan range() atau iterable lain memiliki batas yang jelas, sehingga risiko loop tak berhingga rendah. while lebih fleksibel tetapi membutuhkan disiplin tinggi dalam memastikan kondisi pemberhentian dapat dicapai.

7.3.6 Pernyataan break, continue, dan else pada Loop

break: Keluar dari Loop

`break` menghentikan perulangan secara paksa, melewatkan iterasi-iterasi berikutnya.

```
# Mencari bilangan pertama yang habis dibagi 7 dan 11
for i in range(1, 1000):
    if i % 7 == 0 and i % 11 == 0:
        print(f"Ditemukan: {i}")
        break
# Output: Ditemukan: 77
```

continue: Lompati Iterasi Ini

`continue` melompat ke iterasi berikutnya tanpa mengeksekusi sisa baris dalam blok.

```
# Mencetak hanya bilangan ganjil dari 1 sampai 10
```

```
for i in range(1, 11):
    if i % 2 == 0:
        continue          # lompat ke iterasi berikut
    print(i, end=" ")
# Output: 1 3 5 7 9
```

else pada Loop: Eksekusi Saat Selesai Normal

Python memiliki fitur unik: blok `else` setelah loop yang hanya dieksekusi jika loop selesai *tanpa* break. Berguna untuk pola "cari sesuatu, kalau tidak ada...".

```
# Cek apakah bilangan adalah prima
n = int(input("Bilangan: "))

if n < 2:
    print("Bukan prima.")
else:
    for i in range(2, int(n ** 0.5) + 1):
        if n % i == 0:
            print(f"Bukan prima, habis dibagi {i}")
            break
    else:                        # else terkait dengan for,
bukan if!
        print(f"{n} adalah bilangan prima.")
```

WAWASAN INDUSTRI 7.1

Fitur else pada loop adalah ciri khas Python yang jarang ditemukan di bahasa lain. Banyak programmer Python profesional pun tidak menggunakannya karena dianggap membingungkan (else yang tidak terkait dengan if). Sebagai gantinya, mereka mendefinisikan variabel flag (misalnya prima = True/False) di dalam loop. Kedua gaya dapat diterima; gunakan yang paling jelas bagi tim Anda.

7.3.7 Perulangan Bersarang (Nested Loop)

Sama seperti percabangan, perulangan juga dapat bersarang. Setiap iterasi loop luar memicu seluruh siklus loop dalam. Jika loop luar berjalan *m* kali dan loop dalam berjalan *n* kali, total iterasi adalah *m* × *n*.

Contoh: Tabel Perkalian

```
for i in range(1, 6):           # loop luar: baris
    for j in range(1, 6):       # loop dalam: kolom
        print(f"{i*j:4}", end="")
    print()                     # baris baru setelah kolom
selesai

# Output:
#    1   2   3   4   5
#    2   4   6   8  10
#    3   6   9  12  15
#    4   8  12  16  20
#    5  10  15  20  25
```

Contoh: Pola Bintang Segitiga

```
n = 5
for i in range(1, n + 1):
    for j in range(i):
        print("*", end="")
    print()

# Output:
# *
# **
# ***
# ****
# *****
```

PERHATIAN 7.2

Perulangan bersarang dengan loop dalam yang berat secara komputasi dapat memperlambat program secara dramatis. Loop bersarang dua

tingkat memiliki kompleksitas $O(n^2)$, tiga tingkat $O(n^3)$. Untuk n = 1000, loop dua tingkat menjalankan 1 juta iterasi; loop tiga tingkat menjalankan 1 milyar. Selalu pertimbangkan apakah loop bersarang benar-benar diperlukan, atau apakah ada algoritma yang lebih efisien.

7.3.8 Tracing Eksekusi Loop

Untuk memahami atau memverifikasi sebuah loop, teknik *tracing* (yang sudah kita pelajari di Bab 2 untuk desk checking) sangat membantu. Kita simulasikan eksekusi langkah demi langkah dengan tabel yang mencatat nilai variabel.

Sebagai contoh, mari kita telusur kode berikut yang menghitung 5!:

```
hasil = 1
for i in range(1, 6):
    hasil = hasil * i
print(hasil)
```

Tabel telusur:

Iterasi	i	hasil sebelum	Operasi	hasil sesudah
1	1	1	1 × 1	1
2	2	1	1 × 2	2
3	3	2	2 × 3	6
4	4	6	6 × 4	24
5	5	24	24 × 5	120

Hasil akhir adalah 120, yaitu 5 faktorial (5! = 5×4×3×2×1). Tracing seperti ini sangat membantu ketika debugging loop yang menghasilkan output tak terduga. Latihlah kebiasaan ini sejak awal.

7.4 Contoh Kode Lengkap

Skrip berikut menggabungkan semua konsep perulangan yang telah dipelajari dalam sebuah permainan klasik: *Tebak Angka*. Komputer memilih angka rahasia, dan pemain menebak dengan petunjuk "terlalu tinggi" atau "terlalu rendah".

```
"""
Program: Permainan Tebak Angka
Tujuan : Demonstrasi for, while, break, continue, dan nested
loop
"""

import random

# Konstanta permainan
MIN_ANGKA = 1
MAX_ANGKA = 100
MAX_PERCOBAAN = 7

# Header
print("=" * 50)
print("        PERMAINAN TEBAK ANGKA")
print("=" * 50)
print(f"Saya memilih angka antara {MIN_ANGKA} dan
{MAX_ANGKA}.")
print(f"Anda memiliki {MAX_PERCOBAAN} kesempatan menebak.")
print()

# Komputer pilih angka rahasia
rahasia = random.randint(MIN_ANGKA, MAX_ANGKA)

# Loop utama permainan
menang = False
for percobaan in range(1, MAX_PERCOBAAN + 1):
    # Loop dalam: minta input sampai valid
    while True:
        input_str = input(f"Tebakan #{percobaan}: ")
        if input_str.isdigit():
            tebakan = int(input_str)
            if MIN_ANGKA <= tebakan <= MAX_ANGKA:
                break
            else:
```

```
                print(f"  > Harus antara {MIN_ANGKA} dan
{MAX_ANGKA}.")
                continue
        else:
            print("  > Bukan angka valid.")
            continue

    # Bandingkan tebakan dengan rahasia
    if tebakan == rahasia:
        print(f"  >>> TEPAT! Anda menang dalam {percobaan}
percobaan!")
        menang = True
        break
    elif tebakan < rahasia:
        sisa = MAX_PERCOBAAN - percobaan
        print(f"  > Terlalu rendah. Sisa kesempatan: {sisa}")
    else:
        sisa = MAX_PERCOBAAN - percobaan
        print(f"  > Terlalu tinggi. Sisa kesempatan: {sisa}")

# Penutup
print()
print("=" * 50)
if menang:
    print("Selamat! Anda berhasil menebak.")
else:
    print(f"Maaf, kesempatan habis. Angka rahasia adalah
{rahasia}.")
print("=" * 50)
```

Contoh eksekusi:

```
==================================================
        PERMAINAN TEBAK ANGKA
==================================================
Saya memilih angka antara 1 dan 100.
Anda memiliki 7 kesempatan menebak.

Tebakan #1: 50
  > Terlalu tinggi. Sisa kesempatan: 6
Tebakan #2: 25
  > Terlalu rendah. Sisa kesempatan: 5
```

```
Tebakan #3: 37
  > Terlalu tinggi. Sisa kesempatan: 4
Tebakan #4: 31
  >>> TEPAT! Anda menang dalam 4 percobaan!

==================================================
Selamat! Anda berhasil menebak.
==================================================
```

Skrip ini mendemonstrasikan: for dengan range() untuk membatasi jumlah percobaan, while True dengan break sebagai pola "input sampai valid", continue untuk melompati input tidak valid, validasi berlapis dengan if-elif-else di dalam loop, dan flag boolean (`menang`) untuk membedakan kondisi keluar yang berbeda. Pola-pola seperti ini akan sering Anda temui dalam aplikasi nyata.

7.5 Kesalahan Umum dan Cara Mengatasinya

Kesalahan 1: Lupa update kondisi pada while.

Miskonsepsi: "Kenapa loop saya tidak pernah berhenti?"

Realita: Perulangan while membutuhkan pembaruan variabel kondisi di dalam blok. Tanpa pembaruan, kondisi tetap True selamanya. Contoh klasik: while x < 10: print(x). Tanpa baris x += 1, loop tak berhingga. Tekan Ctrl+C untuk menghentikan, lalu tambahkan pembaruan.

Kesalahan 2: Salah memahami range().

Miskonsepsi: "range(1, 10) saya kira mencetak 1 sampai 10."

Realita: range(a, b) menghasilkan a, a+1, ..., b-1 (tidak termasuk b). Jika Anda ingin sampai 10 inklusif, gunakan range(1, 11). Pola half-open ini konsisten dengan slicing string dan list di Python.

Kesalahan 3: Memodifikasi list yang sedang diiterasi.

Miskonsepsi: "Saya menghapus elemen di tengah loop, kok hasilnya aneh?"

Realita: Memodifikasi list yang sedang diiterasi dapat menyebabkan perilaku tidak terduga: ada elemen yang terlewat atau tereksekusi dua kali. Solusi:

iterasi pada salinan dengan list(data) atau bangun list baru dengan list comprehension.

Kesalahan 4: Salah meletakkan break/continue.

Miskonsepsi: "Break di loop dalam menghentikan loop luar saya?"

Realita: break dan continue hanya mempengaruhi loop terdekat yang membungkusnya, bukan loop di luarnya. Jika Anda perlu keluar dari loop bersarang, gunakan variabel flag (atau ekstrak ke fungsi yang menggunakan return).

Kesalahan 5: Else loop dianggap berpasangan dengan if.

Miskonsepsi: "Mengapa else saya tereksekusi padahal kondisi if false?"

Realita: Dalam Python, else dapat berpasangan dengan if, for, atau while berdasarkan indentasi. Periksa indentasi: else yang sejajar dengan for/while adalah else loop, bukan else if. Mungkin ini fitur yang tidak sengaja Anda picu.

7.6 Rangkuman

Bab 7 telah memperkenalkan struktur kontrol perulangan, melengkapi pasangan percabangan-perulangan sebagai fondasi pemrograman prosedural. Perulangan adalah cara komputer mengulangi instruksi dengan efisien, memungkinkan kita memproses data dalam jumlah besar tanpa harus menulis ulang kode.

Python menyediakan dua bentuk utama perulangan. Perulangan `for` ideal ketika jumlah iterasi dapat dihitung di awal, terutama dengan range() untuk urutan numerik atau langsung pada iterable (string, list, tuple). Perulangan `while` ideal ketika perulangan harus berlanjut hingga kondisi tertentu terpenuhi—namun memerlukan disiplin untuk memastikan kondisi pemberhentian dapat tercapai, agar tidak menjadi loop tak berhingga.

Tiga pernyataan kontrol melengkapi struktur perulangan: `break` keluar dari loop, `continue` melompat ke iterasi berikutnya, dan `else` pada loop mengeksekusi blok hanya jika loop selesai tanpa break. Perulangan bersarang memungkinkan pemrosesan dua dimensi (matriks, tabel) namun perlu hati-hati terhadap kompleksitas. Teknik tracing eksekusi membantu memahami dan memverifikasi perulangan yang rumit.

7.7 Latihan Terbimbing

Soal 7.7.1

Soal: Tulis program untuk menghitung jumlah bilangan genap dari 1 sampai 100.

Pembahasan:

```
total = 0
for i in range(2, 101, 2):
    total += i
print(f"Total: {total}")
# Output: 2550

# Alternatif tanpa step:
total = 0
for i in range(1, 101):
    if i % 2 == 0:
        total += i
print(total)
```

Soal 7.7.2

Soal: Lakukan desk checking pada kode: x = 10; while x > 0: x -= 3; print(x). Tentukan output dan nilai akhir x.

Pembahasan:

Iterasi 1: x = 10 > 0 True → x = 10 - 3 = 7, cetak 7

Iterasi 2: x = 7 > 0 True → x = 7 - 3 = 4, cetak 4

Iterasi 3: x = 4 > 0 True → x = 4 - 3 = 1, cetak 1

Iterasi 4: x = 1 > 0 True → x = 1 - 3 = -2, cetak -2

Iterasi 5: x = -2 > 0 False → loop berakhir

Output: 7 4 1 -2, nilai akhir x = -2

Soal 7.7.3

Soal: Tulis program yang menampilkan pola berikut: baris ke-i berisi i bintang. Contoh untuk n=4: *, **, ***, ****.

Pembahasan:

```
n = int(input("n = "))
for i in range(1, n + 1):
    for j in range(i):
        print("*", end="")
    print()

# Cara lebih pythonic dengan operator perkalian string:
for i in range(1, n + 1):
    print("*" * i)
```

Soal 7.7.4

Soal: Buat program yang meminta input angka terus-menerus sampai pengguna mengetik 'selesai'. Tampilkan jumlah semua angka yang valid.

Pembahasan:

```
total = 0
while True:
    inp = input("Angka (atau 'selesai'): ").strip().lower()
    if inp == "selesai":
        break
    if inp.replace(".", "", 1).replace("-", "", 1).isdigit():
        total += float(inp)
    else:
        print("  > input tidak valid, dilewati")
        continue
print(f"Total: {total}")
```

Soal 7.7.5

Soal: Mengapa kode berikut tak berhingga, dan bagaimana memperbaikinya? n = 1; while n != 10: n += 3; print(n)

Pembahasan: Loop tak berhingga karena n tidak pernah persis sama dengan 10. Urutannya: 1, 4, 7, 10, ... seharusnya berhenti di 10. Oh, ternyata 10 tercapai! Tapi tunggu, urutannya adalah 1→4→7→10→13. Berarti kondisi != 10 True ketika n=13, lalu n menjadi 16, dst, tidak pernah persis 10 setelah melewatinya.

Solusi: gunakan operator >= atau >. while n < 10: ... atau gunakan kondisi yang pasti tercapai.

7.8 Latihan Mandiri

- **(C2)** Jelaskan perbedaan fundamental antara perulangan for dan while, dan kapan masing-masing lebih tepat digunakan.
- **(C2)** Apa fungsi break, continue, dan else pada loop? Berikan contoh penggunaan untuk masing-masing.
- **(C3)** Tulis program yang menampilkan bilangan Fibonacci sampai suku ke-n (input pengguna). Contoh n=10: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34.
- **(C3)** Buatlah program yang menampilkan tabel perkalian sebuah bilangan yang dimasukkan pengguna, dari 1 sampai 10.
- **(C3)** Tulis program untuk menghitung mundur dari N sampai 1, kemudian mencetak 'BLAST OFF!'. N dimasukkan pengguna.
- **(C3)** Buat program yang menampilkan semua bilangan prima antara 1 dan 50 menggunakan nested loop.
- **(C4)** Analisis kode berikut: for i in range(3): for j in range(3): if i == j: continue; print(i, j). Tampilkan output dan jelaskan.
- **(C4)** Bandingkan dua pendekatan untuk menghitung 1 + 2 + ... + N: (a) loop biasa dengan O(N) iterasi, (b) rumus n(n+1)/2 dengan O(1). Bahas trade-off kemudahan vs efisiensi.
- **(C4)** Studi kasus: tulis program ATM sederhana yang meminta PIN (maksimal 3 percobaan), kemudian menampilkan menu (1. Saldo, 2. Tarik, 3. Keluar) terus-menerus sampai pengguna pilih Keluar.
- **(C4)** Diskusikan: kapan loop bersarang menjadi masalah performa? Jika sebuah operasi $O(n^2)$ memakan 1 detik untuk n=1000, kira-kira berapa lama untuk n=10000? n=100000?

7.9 Tugas Proyek Mini

"Simulator Pertumbuhan Tabungan"

Buatlah skrip Python yang mensimulasikan pertumbuhan tabungan dengan bunga majemuk. Persyaratan teknis:

- Skrip meminta input: saldo awal (Rp), suku bunga tahunan (%), setoran bulanan (Rp), dan target tabungan (Rp).

- Gunakan loop while untuk menghitung berapa bulan dibutuhkan hingga saldo mencapai target.
- Setiap iterasi mewakili satu bulan: tambahkan setoran bulanan, lalu tambahkan bunga bulanan (bunga tahunan dibagi 12).
- Tampilkan tabel progres untuk setiap 6 bulan: bulan ke-, saldo, total setoran, total bunga.
- Batasi simulasi maksimum 600 bulan (50 tahun). Jika target belum tercapai, beri pesan peringatan.
- Di akhir, tampilkan: total bulan yang dibutuhkan, total uang yang disetor, total bunga yang diperoleh, dan persentase bunga terhadap total saldo akhir.

Format pengumpulan: berkas .py beserta tangkapan layar simulasi untuk minimal 3 skenario berbeda (suku bunga rendah, sedang, tinggi).

7.10 Refleksi Diri

Centang pernyataan berikut yang sudah Anda kuasai:

☐ Saya memahami konsep iterasi dan tiga komponen perulangan (inisialisasi, kondisi, pembaruan).

☐ Saya dapat menggunakan range() dengan satu, dua, atau tiga argumen.

☐ Saya dapat melakukan iterasi pada string, list, dan koleksi lainnya dengan for.

☐ Saya dapat menulis perulangan while dengan kondisi yang dijamin tercapai.

☐ Saya dapat memilih antara for dan while berdasarkan konteks masalah.

☐ Saya menguasai penggunaan break, continue, dan else pada loop.

☐ Saya dapat menyusun perulangan bersarang untuk masalah dua dimensi.

☐ Saya sadar akan implikasi kompleksitas dari loop bersarang.

☐ Saya dapat melakukan tracing eksekusi loop secara manual untuk verifikasi.

☐ Saya tahu cara menghentikan program yang terjebak loop tak berhingga (Ctrl+C).

ASAH OTAK 7.2

Eksperimen: tulis dua versi program untuk menghitung jumlah 1 + 2 + ... + 1.000.000. Versi pertama menggunakan loop, versi kedua menggunakan rumus matematika n*(n+1)/2. Gunakan modul time atau timeit untuk mengukur waktu eksekusi. Berapa lipat versi rumus lebih cepat dari versi loop? Renungkan: kadang algoritma terbaik bukanlah "berpikir seperti komputer", melainkan "berpikir seperti matematikawan".

BAB 8

UJIAN TENGAH SEMESTER LATIHAN KOMPREHENSIF

"An expert is a man who has made all the mistakes which can be made, in a narrow field."
— Niels Bohr, fisikawan peraih Nobel 1922

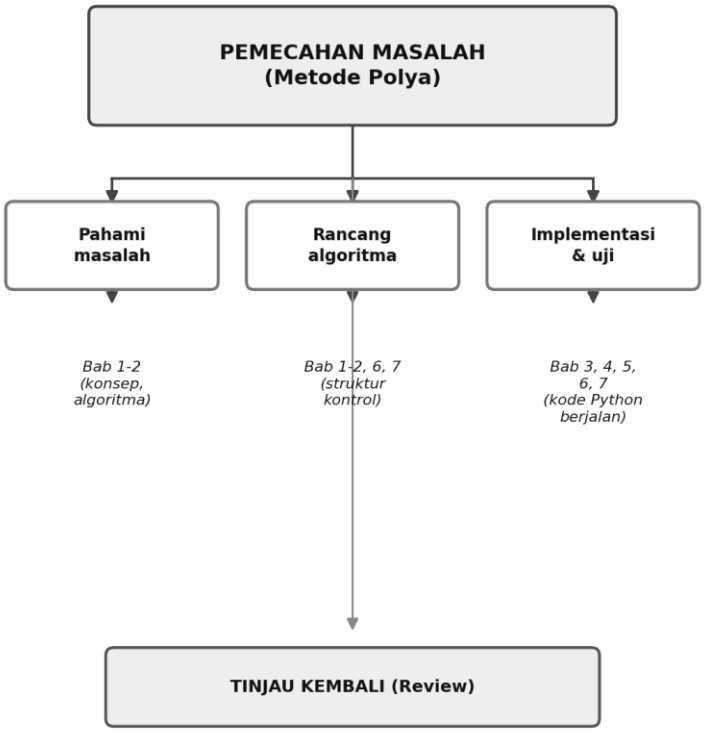
Capaian Pembelajaran Bab

Setelah mempelajari bab ini, mahasiswa diharapkan mampu mencapai sejumlah luaran pembelajaran sebagai berikut:

Kode	Capaian	Bloom
CPB-8.1	Menerapkan metode Polya untuk memecahkan masalah pemrograman	C3
CPB-8.2	Menganalisis masalah dengan mengidentifikasi input, output, dan kendala	C4
CPB-8.3	Merancang algoritma untuk masalah yang melibatkan kombinasi konsep Bab 1–7	C4
CPB-8.4	Mengimplementasikan rancangan algoritma ke dalam kode Python yang berfungsi	C3
CPB-8.5	Menguji program dengan berbagai jenis kasus uji (normal, batas, istimewa)	C4
CPB-8.6	Mengevaluasi dan memperbaiki solusi berdasarkan hasil pengujian	C5

8.1 Peta Konsep

Peta konsep berikut menggambarkan integrasi seluruh materi Bab 1–7 dalam satu kerangka pemecahan masalah.



8.2 Apersepsi

Anda telah menempuh perjalanan setengah semester. Dari mengenal definisi algoritma di Bab 1, Anda kini dapat menulis program Python yang menerima masukan, melakukan komputasi, mengambil keputusan, dan mengulang sesuatu ribuan kali tanpa lelah. Yang dahulu tampak misterius—kalimat-kalimat aneh seperti `for i in range(10):` atau `if x % 2 == 0:`—kini terbaca selancar bahasa ibu Anda.

Bab ini bukanlah bab baru dalam arti memperkenalkan konsep yang belum dikenal. Bab ini adalah **titik konsolidasi**, tempat berbagai serpihan

pengetahuan dari Bab 1 hingga Bab 7 disatukan menjadi kemampuan utuh: **memecahkan masalah dengan pemrograman**. Kita akan mempelajari kerangka berpikir sistematis (metode Polya), menyaksikan demonstrasinya pada masalah aplikatif, lalu berlatih dengan bank soal yang dirancang untuk memetakan kesiapan Anda menghadapi Ujian Tengah Semester.

Pesan terpenting bab ini bukanlah pada sintaks; sintaks telah Anda kuasai. Pesan terpenting adalah pada **cara berpikir**. Programmer yang baik bukanlah yang paling cepat mengetik, melainkan yang paling cermat memahami masalah sebelum mengetik. Programmer pemula yang ingin menjadi profesional perlu menyadari ini sedini mungkin.

ASAH OTAK 8.1

Ambil sebuah masalah pemrograman yang pernah Anda kerjakan di Bab sebelumnya yang Anda anggap sulit pada saat itu. Coba kerjakan kembali sekarang. Apakah masalah itu masih terasa sama sulitnya? Jika lebih mudah, apa yang berubah dalam diri Anda? Renungkan: kemampuan berpikir komputasional adalah keterampilan yang tumbuh bertahap, bukan dimiliki sejak awal.

8.3 Uraian Materi

8.3.1 Strategi Pemecahan Masalah: Metode Polya

Pada tahun 1945, matematikawan Hungaria-Amerika **George Polya** menerbitkan buku *How to Solve It*, sebuah karya kecil yang kemudian menjadi rujukan klasik untuk pemecahan masalah. Polya merumuskan empat tahap universal yang berlaku untuk hampir semua jenis masalah—matematika, sains, dan tentu saja pemrograman:

- **Understand the problem (Pahami masalah).** Apa yang diketahui? Apa yang dicari? Apakah ada kendala? Bisakah saya menyatakan masalah ini dengan kata-kata saya sendiri?
- **Devise a plan (Susun rencana).** Pernahkah saya menemui masalah serupa? Bisakah masalah ini dipecah menjadi sub-masalah yang lebih sederhana? Algoritma apa yang relevan?

- **Carry out the plan (Laksanakan rencana).** Implementasikan rencana langkah demi langkah, dengan teliti memverifikasi setiap langkah.
- **Look back (Tinjau kembali).** Apakah solusinya benar? Apakah ia menjawab pertanyaan yang sesungguhnya? Adakah cara yang lebih baik?

> **SEJARAH SINGKAT 8.1**
>
> Polya menjalani karier panjang sebagai profesor di Stanford University, dan pendekatannya dalam pemecahan masalah memengaruhi banyak generasi ilmuwan komputer—termasuk Donald Knuth, yang secara terbuka menyatakan bahwa karya Polya menjadi salah satu inspirasi penting bagaimana ia berpikir tentang algoritma. Hingga kini, How to Solve It tetap menjadi salah satu buku yang paling banyak direkomendasikan untuk siapa pun yang ingin menjadi pemikir sistematis.

8.3.2 Tahap 1: Analisis Masalah

Tahap ini sering diabaikan oleh pemula yang tergoda langsung mengetik. Padahal di sinilah pondasi solusi terbentuk. Beberapa pertanyaan kunci yang harus dijawab:

- **Apa masukan (input) program?** Tipe data, format, rentang nilai yang valid, batasan.
- **Apa keluaran (output) yang diharapkan?** Format tampilan, tipe data, urutan, presisi.
- **Aturan dan kendala apa yang berlaku?** Misalnya: nilai harus antara 0–100, atau nama tidak boleh kosong.
- **Adakah kasus khusus?** Nol, negatif, kosong, sangat besar.
- **Bagaimana saya akan mengetahui solusi saya benar?** Apa contoh masukan dan keluaran yang dapat saya uji?

Contoh penerapan: misalkan soal berbunyi "Hitung rata-rata nilai mahasiswa yang lulus dari N data masukan, di mana lulus berarti nilai ≥ 60." Analisisnya:

```
INPUT  : N (bilangan bulat positif) dan N nilai (real, 0-100)
OUTPUT : Rata-rata nilai yang lulus (real, 2 desimal) atau
         pesan jika tidak ada yang lulus
ATURAN : Lulus jika nilai >= 60
```

```
          N harus > 0
          Nilai harus dalam rentang [0, 100]
KASUS   : Normal: ada yang lulus (rata-rata = total/jumlah)
          Khusus: tidak ada yang lulus (rata-rata tidak
terdefinisi)
          Batas : N = 1 (hanya satu nilai)
```

8.3.3 Tahap 2: Perancangan Algoritma

Pada tahap ini, kita merancang *logika solusi* sebelum mengetik kode. Tools yang dapat dipakai: narasi, flowchart, pseudocode. Pilihan tergantung kompleksitas masalah.

Untuk masalah rata-rata nilai lulus tadi, pseudocode-nya:

```
ALGORITMA RataRataLulus
DEKLARASI
    n, i : integer
    nilai, total : real
    jumlah_lulus : integer
DESKRIPSI
    read(n)
    total ← 0
    jumlah_lulus ← 0
    for i ← 1 to n do
        read(nilai)
        if nilai >= 60 then
            total ← total + nilai
            jumlah_lulus ← jumlah_lulus + 1
        endif
    endfor
    if jumlah_lulus > 0 then
        write(total / jumlah_lulus)
    else
        write("Tidak ada yang lulus")
    endif
```

TIPS PRO 8.1

Manfaat terbesar pseudocode adalah memaksa Anda memisahkan masalah "algoritma" dari masalah "sintaks Python". Ketika dua hal ini dipikirkan sekaligus, kepala Anda terbebani dan kesalahan bertambah. Dengan pseudocode di tangan, mengubahnya menjadi Python menjadi tugas yang nyaris mekanis.

8.3.4 Tahap 3 dan 4: Implementasi serta Pengujian

Tahap implementasi menerjemahkan pseudocode ke Python. Tahap pengujian memverifikasi bahwa hasil eksekusi sesuai harapan. Keduanya saling berkaitan dalam siklus: *kode → uji → perbaiki → uji lagi*, hingga seluruh kasus uji lolos.

Tiga kategori kasus uji yang wajib dipertimbangkan:

- **Kasus normal.** Masukan tipikal yang diharapkan. Contoh: N=5, nilai-nilai 70, 80, 50, 90, 65.
- **Kasus batas.** Nilai di tepi rentang yang valid. Contoh: N=1 (minimum), nilai = 60 (tepat di batas lulus).
- **Kasus istimewa.** Kondisi yang mungkin menyebabkan kesalahan. Contoh: semua nilai di bawah 60 (tidak ada yang lulus, pembagian nol).

PERHATIAN 8.1

Kesalahan paling fatal di pengujian adalah "program saya bekerja untuk satu contoh, jadi pasti benar". Bug paling sering muncul justru di kasus batas dan istimewa, bukan di kasus normal. Selalu uji minimal tiga kategori kasus sebelum menyatakan program selesai. Di dunia profesional, pendekatan ini disebut boundary testing dan menjadi bagian wajib dari proses pengembangan.

8.3.5 Bank Soal Komprehensif

Berikut adalah 15 soal latihan yang dirancang untuk mencakup seluruh materi Bab 1–7. Pengelompokan berdasarkan topik utama dan tingkat kesulitan.

Kelompok A: Konsep Algoritma (Bab 1–2) — 3 soal

Soal A.1 *(Mudah)*. Jelaskan dengan kata-kata sendiri perbedaan algoritma, pseudocode, dan program. Berikan satu contoh kasus sederhana yang menunjukkan ketiganya.

Soal A.2 *(Sedang)*. Buatlah flowchart untuk algoritma menentukan hari berdasarkan kode angka (1=Senin, 2=Selasa, dst). Sertakan validasi untuk input di luar 1–7.

Soal A.3 *(Sulit)*. Tulis pseudocode gaya Wirth untuk algoritma menentukan apakah suatu kata adalah palindrom (dibaca sama dari depan dan belakang), kemudian lakukan desk checking untuk kata "radar" dan "python".

Kelompok B: Variabel, Tipe Data, Operator (Bab 4) — 3 soal

Soal B.1 *(Mudah)*. Evaluasi nilai akhir variabel x setelah eksekusi: x = 10; x += 5; x **= 2; x //= 11; x %= 7. Tunjukkan langkah demi langkah.

Soal B.2 *(Sedang)*. Tulis ekspresi boolean untuk: "Sebuah karakter c adalah huruf vokal jika c berada di antara 'a' dan 'z' (case-insensitive) DAN c termasuk dalam 'aiueo'." Uji dengan c = 'A', c = 'b', c = '5'.

Soal B.3 *(Sulit)*. Tanpa menjalankan kode, prediksi output dari: print(True + True * 3 - bool("")); print(10 / 3 + 10 // 3 + 10 % 3); print((2 + 3) ** 2 == 25 and not False).

Kelompok C: Input/Output dan String (Bab 5) — 3 soal

Soal C.1 *(Mudah)*. Tulis program yang menerima nama lengkap pengguna, lalu menampilkan: (a) inisial (huruf pertama tiap kata), (b) jumlah karakter (tanpa spasi), (c) versi terbalik.

Soal C.2 *(Sedang)*. Tulis program yang menerima nomor HP dalam format "08xxxxxxxxx" dan mengubahnya ke format "+62 8xx-xxxx-xxxx". Validasi: harus dimulai dengan 0 dan panjang minimal 10 digit.

Soal C.3 *(Sulit)*. Tulis program yang mengenkode pesan dengan Caesar cipher (geser tiap huruf sebanyak n posisi). Contoh: dengan n=3, 'abc' menjadi 'def'. Pertahankan huruf besar/kecil, non-huruf biarkan apa adanya.

Kelompok D: Percabangan (Bab 6) — 3 soal

Soal D.1 *(Mudah)*. Tulis program yang menerima tiga bilangan dan menentukan bilangan terbesar serta terkecil, lalu menghitung selisihnya.

Soal D.2 *(Sedang)*. Tulis program penentu tahun kabisat (gunakan match-case atau if-elif-else). Aturan: habis dibagi 4 dan tidak habis dibagi 100, ATAU habis dibagi 400.

Soal D.3 *(Sulit)*. Tulis program kalkulator tarif taksi dengan aturan: tarif awal Rp10.000 untuk 2 km pertama, Rp4.500/km berikutnya. Tambahan Rp5.000 jika malam (jam 22.00–05.00). Diskon 10% jika jarak > 20 km.

Kelompok E: Perulangan (Bab 7) — 3 soal

Soal E.1 *(Mudah)*. Tulis program yang menampilkan deret 1, 4, 9, 16, 25, ... (kuadrat bilangan asli) sampai suku ke-N.

Soal E.2 *(Sedang)*. Tulis program penghitung faktorial dan kombinasi C(n, r) = n! / (r! × (n−r)!). Validasi: n ≥ r ≥ 0.

Soal E.3 *(Sulit)*. Tulis program yang menampilkan semua bilangan prima antara A dan B (A < B, keduanya bilangan bulat positif). Gunakan algoritma yang efisien (cek hanya hingga akar kuadrat n).

8.3.6 Pembahasan Solusi Terpilih

Berikut pembahasan lengkap dari lima soal terpilih, satu dari tiap kelompok, sebagai panduan menjawab. Solusi soal lainnya akan dibahas di kelas atau menjadi tugas mandiri.

Solusi Soal A.3 — Pseudocode Palindrom

```
ALGORITMA Palindrom
{ Menentukan apakah kata input adalah palindrom }

DEKLARASI
    kata : string
    i, n : integer
    palindrom : boolean
```

```
DESKRIPSI
    read(kata)
    n ← panjang(kata)
    palindrom ← true
    i ← 0
    while i < n / 2 and palindrom = true do
        if kata[i] ≠ kata[n - 1 - i] then
            palindrom ← false
        endif
        i ← i + 1
    endwhile
    if palindrom = true then
        write("Palindrom")
    else
        write("Bukan palindrom")
    endif
```

Desk checking untuk kata "radar" (n=5):

```
Iterasi 1: i=0, kata[0]='r', kata[4]='r' → sama, palindrom
tetap true
Iterasi 2: i=1, kata[1]='a', kata[3]='a' → sama, palindrom
tetap true
Iterasi 3: i=2 < 2.5? Ya, tapi cek: kata[2]='d', kata[2]='d' →
sama
Iterasi 4: i=3 < 2.5? Tidak. Loop berakhir.
Output: "Palindrom"
```

Solusi Soal B.3 — Prediksi Output

```
# Baris 1: True + True * 3 - bool("")
# Prioritas: bool("") = False = 0
# True * 3 = 1 * 3 = 3
# True + 3 - 0 = 1 + 3 - 0 = 4
# Output: 4

# Baris 2: 10 / 3 + 10 // 3 + 10 % 3
# 10/3 = 3.333..., 10//3 = 3, 10%3 = 1
# Total: 3.333... + 3 + 1 = 7.333...
# Output: 7.333333333333333
```

```
# Baris 3: (2 + 3) ** 2 == 25 and not False
# 5 ** 2 = 25, 25 == 25 = True, not False = True
# True and True = True
# Output: True
```

Solusi Soal C.2 — Format Nomor HP

```
nomor = input("Nomor HP: ").strip().replace(" ",
"").replace("-", "")

# Validasi
if not nomor.startswith("0") or len(nomor) < 10:
    print("Format nomor tidak valid.")
elif not nomor.isdigit():
    print("Nomor harus hanya berisi angka.")
else:
    # Hapus 0 di depan, ganti dengan +62 8
    inti = nomor[1:]   # buang '0' di depan
    # Bagi menjadi: 3 digit pertama, 4 digit tengah, sisanya
    bagian1 = inti[:3]
    bagian2 = inti[3:7]
    bagian3 = inti[7:]
    print(f"+62 {bagian1}-{bagian2}-{bagian3}")

# Contoh: input '081234567890' → output '+62 812-3456-7890'
```

Solusi Soal D.3 — Kalkulator Tarif Taksi

```
TARIF_AWAL = 10000
TARIF_PER_KM = 4500
JARAK_AWAL = 2
TAMBAHAN_MALAM = 5000
DISKON_JAUH = 0.10
JARAK_DISKON = 20

jarak = float(input("Jarak (km): "))
jam = int(input("Jam keberangkatan (0-23): "))

# Hitung tarif dasar
```

```
if jarak <= JARAK_AWAL:
    tarif = TARIF_AWAL
else:
    tarif = TARIF_AWAL + (jarak - JARAK_AWAL) * TARIF_PER_KM

# Tambahan malam (22:00 - 05:00)
if jam >= 22 or jam < 5:
    tarif += TAMBAHAN_MALAM

# Diskon untuk jarak jauh
if jarak > JARAK_DISKON:
    tarif *= (1 - DISKON_JAUH)

print(f"Total tarif: Rp{tarif:,.0f}")
```

Solusi Soal E.3 — Bilangan Prima dalam Rentang

```
a = int(input("Batas bawah: "))
b = int(input("Batas atas: "))

if a > b:
    a, b = b, a   # tukar agar a <= b

print(f"Bilangan prima antara {a} dan {b}:")
for n in range(max(2, a), b + 1):
    # Cek apakah n prima
    prima = True
    for i in range(2, int(n ** 0.5) + 1):
        if n % i == 0:
            prima = False
            break
    if prima:
        print(n, end=" ")
print()

# Optimasi: cek hanya hingga sqrt(n) memberikan kompleksitas
O(N*sqrt(N))
# Untuk N = 1000, ini jauh lebih cepat daripada O(N²)
```

8.3.7 Kisi-Kisi UTS dan Tips Menjawab

Ujian Tengah Semester umumnya menguji empat aspek kompetensi berikut. Persentase dapat bervariasi tergantung kebijakan dosen pengampu:

Aspek	Cakupan	Bobot
Konsep Algoritmik	Definisi, karakteristik, representasi (Bab 1-2)	15%
Sintaks & Tipe Data	Variabel, operator, tipe data, I/O (Bab 3-5)	25%
Logika Percabangan	if-else, match-case, nested (Bab 6)	25%
Logika Perulangan	for, while, nested loop (Bab 7)	25%
Pemecahan Masalah	Studi kasus integratif (Bab 1-7)	10%

Tips menjawab soal pemrograman:

- **Baca soal dua kali.** Pertama untuk gambaran umum, kedua untuk detail. Banyak kesalahan terjadi karena terburu-buru pada bacaan pertama.
- **Tulis pseudocode sebelum kode.** Bahkan pada ujian terbatas waktu, 3 menit untuk pseudocode menghemat 15 menit debugging.
- **Mulai dari kasus paling sederhana.** Jika soal meminta N input, coba dulu untuk N=1 atau N=2 di kepala Anda.
- **Beri nama variabel yang bermakna.** jumlah_lulus jauh lebih baik daripada x. Pemeriksa juga menilai keterbacaan.
- **Selalu sertakan komentar pada bagian kompleks.** Jika logika Anda terlihat rumit, satu kalimat komentar bisa menyelamatkan.
- **Sisakan waktu untuk review.** 10 menit terakhir gunakan untuk membaca ulang dan menelusur eksekusi mental satu kasus uji.

> **WAWASAN INDUSTRI 8.1**
>
> Banyak perusahaan teknologi besar (Google, Meta, Microsoft, Amazon) menggunakan format wawancara coding yang sangat mirip dengan UTS pemrograman: kandidat diberi masalah, kemudian harus

menyelesaikannya dengan menulis kode di papan tulis atau editor sambil menjelaskan pemikiran. Tools seperti LeetCode, HackerRank, dan Codeforces khusus untuk berlatih jenis soal ini. Kebiasaan baik yang Anda bangun saat UTS akan terbawa hingga sesi wawancara kerja nanti.

8.4 Contoh Kode Lengkap

Sebagai demonstrasi penerapan metode Polya secara penuh, mari kita selesaikan satu studi kasus dari awal hingga akhir.

Studi Kasus: Analisis Statistik Nilai Mahasiswa

Soal: Buatlah program yang menerima jumlah mahasiswa (N), kemudian N nilai (rentang 0–100). Program harus menampilkan: total, rata-rata, nilai tertinggi, nilai terendah, jumlah mahasiswa lulus (nilai ≥ 60), dan persentase kelulusan.

Tahap 1: Analisis

```
INPUT  : N (integer, > 0)
         N nilai (real, dalam [0, 100])
OUTPUT : total, rata-rata (2 desimal),
         max, min,
         jumlah_lulus, persen_lulus (1 desimal)
ATURAN : Lulus jika nilai >= 60
         N harus positif
         Nilai harus dalam rentang yang valid
KASUS  : Normal (campuran lulus dan tidak)
         Batas (N=1)
         Istimewa (semua lulus / tidak ada yang lulus)
```

Tahap 2: Rancangan (Pseudocode)

```
ALGORITMA AnalisisStatistik
DEKLARASI
    n, i, jumlah_lulus : integer
```

```
    nilai, total, rata, persen : real
    maks, mins : real
DESKRIPSI
    read(n)
    total ← 0
    jumlah_lulus ← 0
    for i ← 1 to n do
        read(nilai)
        total ← total + nilai
        if i = 1 then
            maks ← nilai
            mins ← nilai
        else
            if nilai > maks then maks ← nilai endif
            if nilai < mins then mins ← nilai endif
        endif
        if nilai >= 60 then jumlah_lulus ← jumlah_lulus + 1
endif
    endfor
    rata ← total / n
    persen ← (jumlah_lulus / n) * 100
    { tampilkan semua statistik }
```

Tahap 3: Implementasi Python

```
"""
Program: Analisis Statistik Nilai Mahasiswa
Tujuan : Menampilkan total, rata-rata, max, min, dan
persentase
         kelulusan dari N nilai yang dimasukkan pengguna
"""

BATAS_LULUS = 60

# Input N
n = int(input("Jumlah mahasiswa: "))

if n <= 0:
    print("ERROR: jumlah harus positif.")
else:
    # Inisialisasi
    total = 0
```

```
jumlah_lulus = 0
maks = float("-inf")    # nilai paling kecil yang mungkin
mins = float("inf")     # nilai paling besar yang mungkin

# Loop pembacaan nilai
for i in range(1, n + 1):
    nilai = float(input(f"Nilai mahasiswa ke-{i}: "))

    # Validasi rentang
    if nilai < 0 or nilai > 100:
        print("  > nilai di luar rentang, dianggap 0")
        nilai = 0

    # Akumulasi statistik
    total += nilai
    if nilai > maks:
        maks = nilai
    if nilai < mins:
        mins = nilai
    if nilai >= BATAS_LULUS:
        jumlah_lulus += 1

# Hitung statistik turunan
rata = total / n
persen = (jumlah_lulus / n) * 100

# Tampilkan hasil
print()
print("=" * 50)
print("       ANALISIS STATISTIK NILAI")
print("=" * 50)
print(f"Jumlah mahasiswa  : {n}")
print(f"Total nilai       : {total:.2f}")
print(f"Rata-rata         : {rata:.2f}")
print(f"Nilai tertinggi   : {maks:.2f}")
print(f"Nilai terendah    : {mins:.2f}")
print(f"Jumlah lulus      : {jumlah_lulus} dari {n}")
print(f"Persentase lulus  : {persen:.1f}%")
print("=" * 50)
```

Tahap 4: Pengujian

Kasus normal: N=5, nilai 70, 85, 55, 90, 60. Diharapkan: total=360, rata=72, maks=90, mins=55, lulus=4, persen=80%.

Kasus batas: N=1, nilai 75. Diharapkan: semua statistik = 75, lulus=1, persen=100%.

Kasus istimewa: N=3, nilai 30, 40, 50. Diharapkan: lulus=0, persen=0%.

Kasus invalid: N=0 atau N=-5. Diharapkan: pesan error, program berhenti.

> **DI BALIK LAYAR 8.1**
>
> Penggunaan float("inf") dan float("-inf") sebagai nilai awal untuk min dan max adalah teknik klasik dalam pemrograman. Mereka adalah nilai positif dan negatif tak hingga yang valid dalam IEEE 754. Bilangan apa pun pasti < float("inf") dan > float("-inf"), sehingga iterasi pertama akan selalu menggantikannya. Alternatifnya adalah menggunakan nilai dari iterasi pertama, seperti pada pseudocode di atas. Keduanya valid.

8.5 Kesalahan Umum saat Ujian

Kesalahan 1: Langsung mengetik tanpa memahami soal.

Miskonsepsi: "Saya hanya punya 2 jam, tidak ada waktu untuk pseudocode."

Realita: Justru karena waktu terbatas, perencanaan menjadi penting. 5 menit pseudocode menghemat 20 menit debugging. Hampir semua mahasiswa yang gagal UTS bukan karena tidak tahu sintaks, melainkan karena salah memahami soal.

Kesalahan 2: Menguji hanya satu kasus contoh.

Miskonsepsi: "Contoh di soal sudah benar, berarti program saya beres."

Realita: Contoh di soal biasanya kasus normal. Bug paling sering muncul di kasus batas (N=0, N=1, masukan negatif) dan istimewa (semua sama, semua berbeda, dst). Selalu uji minimal tiga kategori.

Kesalahan 3: Menulis kode panjang yang tidak terstruktur.

Miskonsepsi: "Yang penting jalan, kerapihan urusan nanti."

Realita: Pemeriksa membaca puluhan kertas; kode yang berantakan cenderung dapat nilai lebih rendah meskipun benar. Indentasi konsisten, penamaan variabel bermakna, dan komentar singkat memudahkan pemeriksa dan meningkatkan kepercayaan terhadap solusi Anda.

Kesalahan 4: Lupa output sesuai format yang diminta.

Miskonsepsi: "Hasil saya benar, tinggal ubah formatnya nanti."

Realita: Output yang tidak sesuai format (kurang satuan, salah jumlah desimal, urutan tertukar) sering dianggap salah secara penuh oleh pemeriksa otomatis. Selalu cek instruksi format output, dan pastikan f-string Anda sesuai.

Kesalahan 5: Stress saat menemui bug.

Miskonsepsi: "Kalau error, berarti saya tidak bisa."

Realita: Bug adalah bagian normal dari pemrograman, bahkan untuk profesional. Tarik napas, baca pesan error dengan teliti (ia memberitahu Anda apa yang salah dan di baris berapa), telusur eksekusi secara manual, dan jangan menyerah. Mental yang tenang menyelesaikan masalah lebih cepat daripada panik.

8.6 Rangkuman

Bab 8 berfungsi sebagai titik konsolidasi materi Bab 1–7. Metode Polya menyediakan kerangka empat tahap untuk memecahkan masalah pemrograman: pahami, rancang, laksanakan, tinjau. Tahap analisis melibatkan identifikasi input, output, aturan, kasus, dan kriteria kebenaran. Tahap perancangan menghasilkan pseudocode atau flowchart sebagai jembatan menuju kode. Tahap implementasi menerjemahkan rancangan ke Python, dan tahap pengujian memverifikasi dengan kasus normal, batas, dan istimewa.

Bank soal pada bab ini terdiri atas 15 soal yang mencakup seluruh aspek kompetensi: konsep algoritmik, variabel dan tipe data, input/output dan string, percabangan, serta perulangan. Lima soal di antaranya telah dibahas tuntas sebagai panduan. Tips menjawab UTS mencakup membaca soal dua kali,

menulis pseudocode lebih dahulu, memilih nama variabel bermakna, dan menyisakan waktu untuk review.

Setelah Bab 8, mahasiswa siap memasuki separuh kedua semester yang akan memperkenalkan struktur data koleksi (List, Tuple, Dictionary, Set), fungsi sebagai abstraksi, modularisasi, dan penanganan berkas. Materi tersebut akan dibangun di atas fondasi yang telah Anda kuasai sekarang.

8.7 Latihan Terbimbing

Lima soal berikut dirancang sebagai latihan komprehensif yang dipandu langkah demi langkah, untuk membantu Anda melatih metode Polya pada masalah pemrograman.

Soal 8.7.1

Soal: Tulis program yang menerima sebuah string dan menghitung berapa banyak vokal dan konsonan di dalamnya (huruf non-alfabet diabaikan).

Pembahasan:

```
kalimat = input("Masukkan kalimat: ").lower()
vokal = 0
konsonan = 0

for huruf in kalimat:
    if huruf.isalpha():
        if huruf in "aiueo":
            vokal += 1
        else:
            konsonan += 1

print(f"Vokal    : {vokal}")
print(f"Konsonan : {konsonan}")
print(f"Total    : {vokal + konsonan} huruf")
```

Soal 8.7.2

Soal: Tulis program konverter satuan suhu (Celsius, Fahrenheit, Kelvin) yang menerima nilai dan satuan asal, lalu menampilkan konversi ke dua satuan lainnya.

Pembahasan:

```
nilai = float(input("Nilai suhu: "))
satuan = input("Satuan (C/F/K): ").upper()

match satuan:
    case "C":
        c = nilai
        f = (c * 9/5) + 32
        k = c + 273.15
    case "F":
        f = nilai
        c = (f - 32) * 5/9
        k = c + 273.15
    case "K":
        k = nilai
        c = k - 273.15
        f = (c * 9/5) + 32
    case _:
        print("Satuan tidak valid.")
        c = f = k = None

if c is not None:
    print(f"Celsius    : {c:.2f}°C")
    print(f"Fahrenheit : {f:.2f}°F")
    print(f"Kelvin     : {k:.2f} K")
```

Soal 8.7.3

Soal: Tulis program yang mensimulasikan pengundian arisan untuk N peserta. Setiap putaran, satu nomor diundi secara acak dan dihilangkan dari undian sampai semua peserta dapat giliran.

Pembahasan:

```
import random

n = int(input("Jumlah peserta: "))
peserta = list(range(1, n + 1))     # peserta nomor 1 sampai n
putaran = 1

print(f"\nMengundi arisan untuk {n} peserta...\n")
```

```
while peserta:
    terpilih = random.choice(peserta)
    print(f"Putaran {putaran}: peserta nomor {terpilih}")
    peserta.remove(terpilih)
    putaran += 1

print("\nArisan selesai. Semua peserta telah mendapat
giliran.")
```

Soal 8.7.4

Soal: Tulis program yang menampilkan piramida bintang dengan tinggi N. Contoh untuk N=4: `   *  *** ***** *******`

Pembahasan:

```
n = int(input("Tinggi piramida: "))

for i in range(1, n + 1):
    # Spasi: n - i di kiri
    spasi = " " * (n - i)
    # Bintang: 2*i - 1 di tengah
    bintang = "*" * (2 * i - 1)
    print(spasi + bintang)

# Untuk N=4:
#    *
#   ***
#  *****
# *******
```

Soal 8.7.5

Soal: Tulis program penghitung skor permainan: input berupa skor satu per satu (positif untuk poin, negatif untuk penalti, 0 untuk berhenti). Tampilkan total skor, jumlah poin, jumlah penalti, dan skor tertinggi yang pernah dicatat.

Pembahasan:

```
total = 0
jumlah_poin = 0
jumlah_penalti = 0
skor_tertinggi = float("-inf")

print("Masukkan skor (0 untuk berhenti):")
while True:
    skor = float(input("> "))
    if skor == 0:
        break

    total += skor
    if skor > skor_tertinggi:
        skor_tertinggi = skor

    if skor > 0:
        jumlah_poin += 1
    else:
        jumlah_penalti += 1

print(f"\n--- HASIL ---")
if jumlah_poin + jumlah_penalti == 0:
    print("Tidak ada skor dicatat.")
else:
    print(f"Total skor       : {total}")
    print(f"Jumlah poin      : {jumlah_poin}")
    print(f"Jumlah penalti   : {jumlah_penalti}")
    print(f"Skor tertinggi   : {skor_tertinggi}")
```

8.8 Latihan Mandiri (Simulasi UTS)

Sepuluh soal berikut adalah simulasi soal UTS. Kerjakan dalam waktu maksimal 120 menit, tanpa membuka catatan atau internet, sebagai latihan kondisi ujian. Setiap soal bernilai sama; lulus jika menyelesaikan minimal 6 soal dengan benar.

- **(C3)** Tulis program yang menerima jari-jari sebuah lingkaran dan menampilkan keliling serta luasnya (gunakan π = 3.14159).

- **(C3)** Tulis program konverter rupiah ke dollar (1 USD = Rp 16.000) dan sebaliknya. Pengguna memilih arah konversi.
- **(C3)** Tulis program yang menerima nilai dan menampilkan kategorinya: A (>=85), B (75-84), C (65-74), D (55-64), E (<55). Validasi rentang 0-100.
- **(C3)** Tulis program yang menerima sebuah kata dan memeriksa apakah ia palindrom (case-insensitive).
- **(C3)** Tulis program yang menampilkan tabel perkalian dari 1 sampai N (input pengguna) dalam format yang rapi.
- **(C4)** Tulis program yang menerima 3 sisi segitiga dan menentukan jenisnya (sama sisi, sama kaki, sembarang) atau bahwa ketiga sisi tidak dapat membentuk segitiga.
- **(C4)** Tulis program yang mensimulasikan kalkulator sederhana dengan menu (1. Tambah, 2. Kurang, 3. Kali, 4. Bagi, 5. Keluar) yang berulang sampai pengguna pilih Keluar. Tangani pembagian dengan nol.
- **(C4)** Tulis program yang menghitung gaji karyawan: gaji pokok ditambah bonus berdasarkan jam lembur. Aturan: 1–10 jam = Rp50.000/jam, 11–20 jam = Rp75.000/jam, >20 jam = Rp100.000/jam.
- **(C4)** Tulis program yang menerima N kata dan menampilkan kata terpanjang serta panjangnya. Jika ada beberapa yang sama panjangnya, tampilkan kata pertama yang ditemui.
- **(C4)** Tulis program penghitung deret Fibonacci yang menampilkan suku ke-1 sampai suku ke-N, di mana N adalah masukan pengguna.

8.9 Tugas Proyek Mini

"Sistem Pengelolaan Nilai Mahasiswa Mini"

Sebagai konsolidasi sebelum UTS, kerjakan proyek terintegrasi berikut yang menggabungkan semua materi Bab 1–7.

Spesifikasi:

- Program memulai dengan menu utama: (1) Tambah nilai, (2) Tampilkan statistik, (3) Cari nilai tertentu, (4) Keluar.
- Untuk "Tambah nilai": minta NIM, nama, dan 3 nilai (UTS, UAS, Tugas) dengan bobot 30%, 40%, 30%. Hitung nilai akhir dan tampilkan.
- Untuk "Statistik": tampilkan jumlah data yang sudah dimasukkan, rata-rata nilai akhir, dan persentase kelulusan (lulus jika nilai akhir ≥ 60).
- Untuk "Cari nilai": minta NIM, lalu tampilkan rincian mahasiswa tersebut jika ditemukan.

- Validasi semua input: NIM minimal 8 digit, nama minimal 3 karakter, nilai 0–100.
- Tampilan menu menggunakan ASCII art yang rapi, dengan pembatas garis horizontal.

Catatan: Untuk menyimpan banyak data, Anda boleh menggunakan list dan dictionary, meskipun keduanya baru dibahas tuntas di Bab 9–10. Konsep dasarnya: `data_mahasiswa = []` sebagai list, dan setiap data adalah dictionary `{"nim":..., "nama":..., "nilai_akhir":...}`. Lakukan riset mandiri untuk pendekatan ini—ini adalah latihan belajar mandiri yang berharga.

Format pengumpulan: berkas .py beserta dokumen PDF berisi (a) analisis masalah dengan metode Polya, (b) pseudocode setiap fungsi/menu, (c) tangkapan layar pengujian minimal 5 skenario, (d) refleksi pengalaman pengerjaan.

8.10 Refleksi Diri

Centang pernyataan berikut yang sudah Anda kuasai sebelum mengikuti UTS:

☐ Saya memahami dan dapat menerapkan empat tahap metode Polya.

☐ Saya dapat menganalisis masalah dengan mengidentifikasi input, output, aturan, dan kasus uji.

☐ Saya dapat menulis pseudocode dari soal naratif sebelum mengetik kode.

☐ Saya mampu mengimplementasikan algoritma sederhana dari Bab 1-7 dalam Python.

☐ Saya menguasai sintaks dan semantik percabangan (if-elif-else, match-case).

☐ Saya menguasai sintaks dan semantik perulangan (for, while, break, continue).

☐ Saya dapat melakukan tracing eksekusi secara manual untuk verifikasi.

☐ Saya tahu cara menguji program dengan kasus normal, batas, dan istimewa.

☐ Saya tetap tenang ketika menghadapi pesan error dan dapat membacanya.

☐ Saya percaya diri menghadapi UTS dan tahu apa yang harus dikuasai.

ASAH OTAK 8.2

Sebelum UTS, ambil tiga proyek mini yang telah Anda kerjakan dari Bab 1–7. Buka lagi kode tersebut, baca dengan teliti, dan tanyakan: apakah saya masih dapat menjelaskan setiap baris? Jika ada bagian yang tidak Anda ingat alasannya, di situlah celah pemahaman Anda berada. Tutup celah itu sebelum hari ujian. Programmer yang kuat adalah yang dapat menjelaskan kodenya sendiri—bukan hanya yang dapat menulisnya.

BAB 9

LIST DAN TUPLE

"Data dominates. If you've chosen the right data structures and organized things well, the algorithms will almost always be self-evident."

— Rob Pike, salah satu perancang bahasa Go

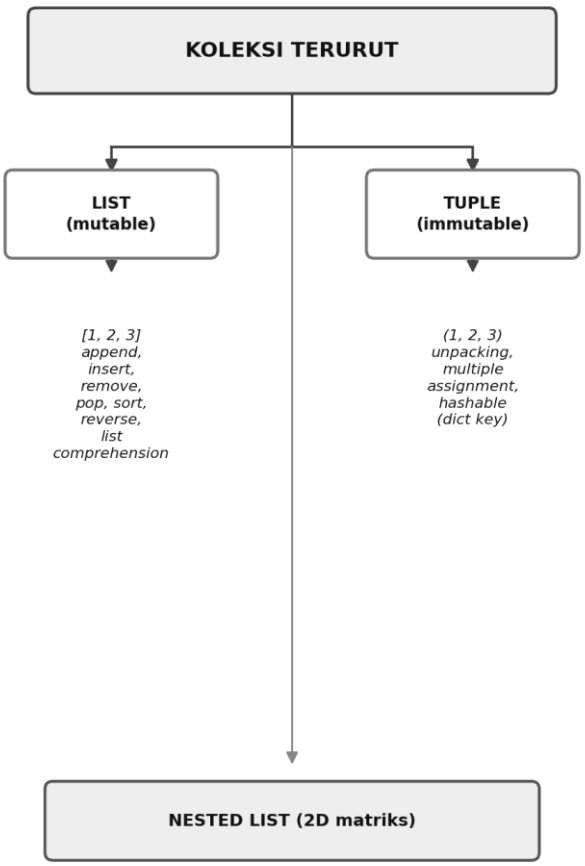
Capaian Pembelajaran Bab

Setelah mempelajari bab ini, mahasiswa diharapkan mampu mencapai sejumlah luaran pembelajaran sebagai berikut:

Kode	Capaian	Bloom
CPB-9.1	Menjelaskan konsep koleksi data terurut dan kegunaannya	C2
CPB-9.2	Membuat, mengakses, dan memodifikasi list dengan operasi dasar	C3
CPB-9.3	Menerapkan metode-metode list untuk manipulasi data	C3
CPB-9.4	Menyusun list comprehension untuk transformasi data ringkas	C3
CPB-9.5	Membedakan karakteristik dan penggunaan tuple versus list	C2
CPB-9.6	Menerapkan tuple unpacking dan multiple assignment	C3
CPB-9.7	Menganalisis implikasi mutability dan immutability pada memori	C4
CPB-9.8	Menggunakan list bersarang untuk merepresentasikan data dua dimensi	C3

9.1 Peta Konsep

Peta konsep berikut menggambarkan keterkaitan dua koleksi terurut Python: list yang mutable dan tuple yang immutable.



9.2 Apersepsi

Hingga Bab 8, ketika kita perlu menyimpan banyak data, kita menggunakan banyak variabel terpisah: `nilai1, nilai2, nilai3, ...` atau `mahasiswa_satu, mahasiswa_dua, ...`. Pendekatan ini cepat menjadi tidak praktis. Bagaimana jika kita harus menyimpan nilai 30 mahasiswa di kelas? 100 mahasiswa di angkatan? 1.000.000 transaksi di e-commerce? Membuat sejuta variabel jelas tidak masuk akal.

Inilah mengapa setiap bahasa pemrograman menyediakan **struktur data koleksi**: cara menyimpan banyak nilai dalam satu wadah. Python memiliki empat koleksi bawaan utama, dan kita akan mempelajari yang pertama dan paling sering dipakai di bab ini: **list** dan **tuple**. Setelah menguasai keduanya, Anda akan terkejut betapa banyak program yang sebelumnya tampak rumit menjadi sangat sederhana.

List dan tuple bukan sekadar fitur baru—ia adalah *lompatan kuantum* dalam kemampuan Anda menulis program. Bersama dengan dictionary dan set di Bab 10, struktur data ini membuka pintu menuju aplikasi nyata yang memproses data dunia: daftar mahasiswa, riwayat transaksi, koleksi gambar, dataset penelitian.

> **ASAH OTAK 9.1**
>
> Tanpa menggunakan list atau struktur data koleksi apa pun, bisakah Anda menulis program yang menyimpan dan menampilkan 10 nilai mahasiswa? Ya, bisa—dengan 10 variabel terpisah. Sekarang: bagaimana jika program harus menyimpan N nilai di mana N baru diketahui saat runtime? Apakah masih mungkin tanpa koleksi? Renungkan keterbatasan ini sebelum melanjutkan ke materi formal.

9.3 Uraian Materi

9.3.1 Konsep Koleksi Data Terurut

Sebuah *koleksi* adalah objek yang menyimpan banyak nilai. Python memiliki beberapa jenis koleksi yang berbeda dalam tiga dimensi:

- **Terurut atau tidak?** Apakah urutan elemen penting dan dipertahankan?
- **Bisa diubah (mutable) atau tetap (immutable)?** Apakah elemen dapat ditambah, dihapus, atau dimodifikasi setelah pembuatan?
- **Boleh duplikat atau unik?** Apakah elemen yang sama boleh muncul lebih dari satu kali?

Empat jenis koleksi bawaan Python diringkas dalam tabel berikut:

Tipe	Terurut	Mutable	Duplikat	Sintaks
list	Ya	Ya	Boleh	[1, 2, 3]
tuple	Ya	Tidak	Boleh	(1, 2, 3)
set	Tidak	Ya	Tidak	{1, 2, 3}
dict	Ya (3.7+)	Ya	Kunci unik	{"a": 1}

Di bab ini kita fokus pada dua yang pertama (`list` dan `tuple`). Bab 10 akan membahas `dict` dan `set`.

9.3.2 Pembuatan, Akses, dan Modifikasi List

List adalah koleksi paling fleksibel di Python: terurut, mutable, dan boleh berisi tipe data apa pun (bahkan campuran).

Pembuatan List

```
# List kosong
kosong = []
kosong2 = list()                          # cara alternatif

# List dengan elemen homogen
nilai = [78, 85, 92, 67, 88]
nama = ["Andi", "Budi", "Cici"]

# List dengan elemen heterogen (boleh, tapi jarang)
campur = [1, "halo", 3.14, True, [1, 2]]

# List dari range
angka = list(range(1, 11))                #
[1,2,3,4,5,6,7,8,9,10]

# List dari string
huruf = list("Python")                    #
['P','y','t','h','o','n']

# List dengan elemen berulang
nol_lima = [0] * 5                        # [0, 0, 0, 0, 0]
```

Akses Elemen

List menggunakan indexing dan slicing yang sama seperti string yang sudah kita pelajari di Bab 5.

```
nilai = [78, 85, 92, 67, 88]
#        0   1   2   3   4    (indeks positif)
#       -5  -4  -3  -2  -1    (indeks negatif)

nilai[0]        # 78  (elemen pertama)
nilai[-1]       # 88  (elemen terakhir)
nilai[1:4]      # [85, 92, 67]  (slicing)
nilai[::2]      # [78, 92, 88]  (loncat 2)
nilai[::-1]     # [88, 67, 92, 85, 78]  (terbalik)
len(nilai)      # 5  (panjang)
```

Modifikasi (Ini Berbeda dari String!)

Karena list *mutable,* kita dapat memodifikasi elemennya tanpa membuat list baru. Ini perbedaan fundamental dengan string.

```
nilai = [78, 85, 92, 67, 88]

# Modifikasi satu elemen
nilai[0] = 80
print(nilai)              # [80, 85, 92, 67, 88]

# Modifikasi rentang (slice assignment)
nilai[1:3] = [70, 75]
print(nilai)              # [80, 70, 75, 67, 88]

# Slice assignment bisa mengubah ukuran
nilai[1:3] = [100]        # ganti 2 elemen jadi 1 elemen
print(nilai)              # [80, 100, 67, 88]

# Penggabungan (concatenation) menghasilkan list BARU
gabung = [1, 2] + [3, 4]  # [1, 2, 3, 4]

# Pengulangan
ulang = [0] * 3           # [0, 0, 0]
```

CATATAN 9.1

Iterasi pada list bekerja sama dengan iterasi pada string yang sudah kita pelajari: for elemen in list: ...; atau dengan enumerate() untuk mendapatkan indeks. Operator in juga bekerja untuk mengecek keanggotaan: if 85 in nilai: ...

9.3.3 Metode-Metode Penting List

List memiliki banyak metode bawaan. Tabel berikut merangkum yang paling sering dipakai:

Metode	Fungsi	Contoh
.append(x)	Menambah x di akhir list	[1,2].append(3) → [1,2,3]
.extend(iter)	Menambah elemen-elemen dari iterable	[1,2].extend([3,4]) → [1,2,3,4]
.insert(i, x)	Menyisipkan x di indeks i	[1,3].insert(1, 2) → [1,2,3]
.remove(x)	Menghapus kemunculan pertama x	[1,2,3].remove(2) → [1,3]
.pop([i])	Mengeluarkan dan mengembalikan elemen	[1,2,3].pop() → 3
.clear()	Menghapus semua elemen	[1,2,3].clear() → []
.index(x)	Indeks kemunculan pertama x	[1,2,3].index(2) → 1
.count(x)	Jumlah kemunculan x	[1,2,2,3].count(2) → 2
.sort()	Mengurutkan in-place (ascending)	[3,1,2].sort() → [1,2,3]
.reverse()	Membalik urutan in-place	[1,2,3].reverse() → [3,2,1]
.copy()	Membuat salinan (shallow copy)	[1,2,3].copy() → [1,2,3]

Demonstrasi dalam aksi:

```
nilai = [78, 85, 92, 67]

# Menambah elemen
nilai.append(88)              # [78, 85, 92, 67, 88]
nilai.insert(0, 100)          # [100, 78, 85, 92, 67, 88]

# Menghapus elemen
nilai.remove(78)              # [100, 85, 92, 67, 88]
terakhir = nilai.pop()        # ambil yang terakhir: 88
kedua = nilai.pop(1)          # ambil indeks 1: 85

# Pengurutan dan pembalikan
nilai.sort()                  # urut menaik in-place
nilai.sort(reverse=True)      # urut menurun
nilai.reverse()               # balik urutan

# Pencarian
if 92 in nilai:
    print(f"Indeks 92: {nilai.index(92)}")
```

PERHATIAN 9.1

Metode .sort() dan .reverse() memodifikasi list AS-IS (in-place) dan mengembalikan None. Jangan menulis: nilai = nilai.sort()! Ini akan membuat nilai menjadi None, dan list aslinya hilang. Cukup tulis: nilai.sort(). Jika Anda ingin mengurutkan tanpa mengubah list asli, gunakan fungsi sorted(nilai) yang mengembalikan list baru.

9.3.4 List Comprehension: Idiom Pythonic

List comprehension adalah cara ringkas untuk membuat list baru berdasarkan iterasi atas iterable lain, dengan opsi filter dan transformasi. Sintaksnya:

```
[ekspresi for elemen in iterable if kondisi]
```

Bandingkan dua cara membuat list bilangan kuadrat dari 1 sampai 10:

```
# Cara lama (4 baris)
kuadrat = []
for i in range(1, 11):
    kuadrat.append(i ** 2)

# Dengan list comprehension (1 baris)
kuadrat = [i ** 2 for i in range(1, 11)]
# [1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

Dengan filter (klausa if):

```
# Kuadrat dari bilangan genap saja
kuadrat_genap = [i ** 2 for i in range(1, 11) if i % 2 == 0]
# [4, 16, 36, 64, 100]

# Mengubah list nilai menjadi huruf
nilai = [85, 72, 91, 55, 68]
huruf = ["A" if n >= 85 else "B" if n >= 75 else "C" if n >=
65 else "E" for n in nilai]
# ['A', 'C', 'A', 'E', 'C']

# Menyaring nama yang berawalan huruf besar
kata = ["halo", "Dunia", "python", "AI"]
kapital = [k for k in kata if k[0].isupper()]
# ['Dunia', 'AI']
```

TIPS PRO 9.1

List comprehension yang baik singkat dan jelas. Jika sebuah comprehension Anda harus dibaca beberapa kali untuk dipahami, kembali ke loop biasa. Aturan praktis: jika comprehension Anda lebih panjang dari 80 karakter atau memiliki lebih dari satu if/for, ia sudah terlalu kompleks. Pythonic bukan tentang menulis kode terpendek, tetapi yang paling jelas.

9.3.5 Tuple: Karakteristik dan Penggunaan

Tuple adalah saudara list yang *tidak dapat dimodifikasi* setelah pembuatan. Ia ditulis dengan tanda kurung biasa ( ) dan koma sebagai pemisah.

```
# Pembuatan tuple
kosong = ()                  # tuple kosong
satu = (5,)                  # PERHATIAN: koma wajib untuk
tuple 1 elemen
banyak = (1, 2, 3, 4, 5)
campur = ("Yusri", 30, 3.85)

# Bahkan tanpa kurung pun masih tuple
tanpa_kurung = 1, 2, 3       # juga tuple

# Akses sama seperti list
banyak[0]                    # 1
banyak[-1]                   # 5
banyak[1:3]                  # (2, 3)
len(banyak)                  # 5

# Tetapi MODIFIKASI dilarang
banyak[0] = 100              # TypeError!
banyak.append(6)             # AttributeError!
```

Mengapa menggunakan tuple jika list lebih fleksibel?

- **Performa.** Tuple sedikit lebih cepat untuk diakses dan menempati memori lebih sedikit.
- **Keamanan data.** Untuk data yang seharusnya tidak berubah (koordinat, tanggal, konfigurasi), tuple mencegah modifikasi tidak sengaja.
- **Dapat dijadikan kunci dictionary.** Tuple hashable (jika elemennya juga hashable), list tidak.
- **Pengembalian banyak nilai dari fungsi.** return x, y secara internal mengembalikan tuple.

> **DI BALIK LAYAR 9.1**
>
> Tuple satu elemen sintaks (5) bukan tuple, melainkan ekspresi dengan tanda kurung. Untuk membuat tuple satu elemen, Anda HARUS menambahkan koma: (5,). Ini sering menjebak pemula. Demonstrasi:

> type((5)) menghasilkan int, sedangkan type((5,)) menghasilkan tuple. Detail kecil ini adalah salah satu "wart" Python yang Guido sendiri akui sebagai trade-off antara konsistensi sintaks dan ekspresi matematis.

9.3.6 Tuple Unpacking dan Multiple Assignment

Tuple unpacking adalah fitur yang membuat Python sangat ekspresif: Anda dapat "membongkar" tuple ke beberapa variabel dalam satu pernyataan.

```
# Unpacking dasar
koordinat = (3, 5)
x, y = koordinat
print(x, y)                  # 3 5

# Multiple assignment (kasus khusus unpacking)
a, b, c = 1, 2, 3
print(a, b, c)               # 1 2 3

# Swap (tukar) nilai variabel—idiom klasik Python
a, b = b, a                  # tidak perlu variabel sementara!

# Unpacking dengan list/range
x, y, z = [10, 20, 30]
p, q, r = range(3)           # p=0, q=1, r=2

# Unpacking dengan * (catch-all)
pertama, *sisanya = [1, 2, 3, 4, 5]
# pertama = 1, sisanya = [2, 3, 4, 5]

kepala, *tengah, ekor = [1, 2, 3, 4, 5]
# kepala = 1, tengah = [2, 3, 4], ekor = 5

# Mengembalikan banyak nilai dari fungsi
def min_maks(angka):
    return min(angka), max(angka)     # secara teknis return
tuple

lo, hi = min_maks([3, 1, 4, 1, 5, 9])
print(lo, hi)                # 1 9
```

Iterasi atas Tuple dalam List

```
# List berisi tuple—pola umum untuk data "berstruktur"
mahasiswa = [
    ("Andi", 85, "A"),
    ("Budi", 72, "B"),
    ("Cici", 91, "A")
]

# Iterasi dengan unpacking langsung
for nama, nilai, huruf in mahasiswa:
    print(f"{nama}: {nilai} ({huruf})")

# Output:
# Andi: 85 (A)
# Budi: 72 (B)
# Cici: 91 (A)
```

9.3.7 Mutability dan Implikasi pada Memori

Di Bab 4 kita telah menyinggung bahwa variabel di Python adalah *label* yang menunjuk ke objek di memori. Konsep ini menjadi sangat penting saat bekerja dengan list, karena list *mutable* dapat memberikan kejutan yang membingungkan.

Jebakan Aliasing

```
# Penugasan list TIDAK menyalin list, hanya membuat label baru
a = [1, 2, 3]
b = a                       # b dan a menunjuk ke list yang
SAMA

b.append(4)                 # modifikasi melalui b
print(a)                    # [1, 2, 3, 4] -- a juga berubah!

# Untuk membuat salinan independen
c = a.copy()                # atau list(a) atau a[:]
c.append(5)
print(a)                    # [1, 2, 3, 4] -- a tidak berubah
print(c)                    # [1, 2, 3, 4, 5]
```

Shallow vs Deep Copy

.copy() membuat *shallow copy*: list luar disalin, tetapi list dalam masih dishare. Untuk salinan independen sepenuhnya, gunakan modul copy:

```
import copy

matriks_asli = [[1, 2], [3, 4]]
shallow = matriks_asli.copy()           # list luar baru, list
dalam dishare
deep = copy.deepcopy(matriks_asli)      # sepenuhnya
independen

shallow[0][0] = 99                      # modifikasi list
dalam
print(matriks_asli)                     # [[99, 2], [3, 4]] --
terpengaruh!

deep[0][0] = 0
print(matriks_asli)                     # [[99, 2], [3, 4]] --
tidak berubah
```

PERHATIAN 9.2

Jebakan aliasing adalah salah satu sumber bug terbesar di Python. Gejala: Anda memodifikasi variabel A, dan variabel B yang "tidak ada hubungannya" juga berubah. Aturan praktis: selalu sadar apakah Anda ingin berbagi list yang sama (jarang) atau bekerja pada salinan (sering). Saat ragu, gunakan .copy() atau deepcopy().

9.3.8 List Bersarang: Representasi 2D

List dapat berisi list lain, menciptakan struktur multi-dimensi. Bentuk paling umum adalah *list dua dimensi* (matriks), yang sering dipakai untuk papan permainan, tabel data, dan komputasi numerik.

```
# Matriks 3x3
matriks = [
    [1, 2, 3],
```

```
    [4, 5, 6],
    [7, 8, 9]
]

# Akses elemen: [baris][kolom]
matriks[0][0]               # 1 (kiri atas)
matriks[2][2]               # 9 (kanan bawah)
matriks[1][2]               # 6 (baris 2, kolom 3)

# Modifikasi
matriks[0][0] = 100

# Iterasi berlapis
for baris in matriks:
    for elemen in baris:
        print(elemen, end=" ")
    print()

# Pembuatan matriks dengan list comprehension
# Matriks 5x5 berisi nol
zero = [[0 for _ in range(5)] for _ in range(5)]

# Matriks identitas 3x3
identitas = [[1 if i == j else 0 for j in range(3)] for i in
range(3)]
```

PERHATIAN 9.3

Jangan gunakan [[0]*n]*m untuk membuat matriks! Ini menciptakan m list yang merupakan referensi ke list yang sama, sehingga modifikasi satu baris akan mempengaruhi semua baris. Gunakan list comprehension: [[0]*n for _ in range(m)] yang membuat m list yang independen. Bug yang sangat sering dialami pemula.

WAWASAN INDUSTRI 9.1

Untuk komputasi numerik serius (jutaan elemen, operasi matriks kompleks), Python list yang berbasis objek umum cenderung lambat. Solusi standar adalah pustaka NumPy yang menggunakan array

berbasis C dengan tipe data homogen, sehingga secara signifikan lebih cepat untuk operasi vektor dan matriks. Hampir semua data science, machine learning, dan komputasi ilmiah di Python menggunakan NumPy array, bukan list bawaan. NumPy akan diperkenalkan singkat di akhir buku ini dan menjadi materi utama di mata kuliah Algoritma dan Pemrograman 2.

9.4 Contoh Kode Lengkap

Skrip berikut menggabungkan semua konsep bab ini dalam aplikasi nyata: *Analisis Nilai Kelas*. Program menerima data N mahasiswa berupa nama dan nilai, lalu menampilkan analisis komprehensif.

```
"""
Program: Analisis Nilai Kelas
Tujuan : Demonstrasi list, tuple, list comprehension, dan
sorting
         dalam konteks analisis data mahasiswa
"""

BATAS_LULUS = 60

# Kumpulkan data
print("=" * 50)
print("       ANALISIS NILAI KELAS")
print("=" * 50)
n = int(input("Jumlah mahasiswa: "))

# List berisi tuple (nama, nilai)—pola umum untuk data
terstruktur
data = []
for i in range(1, n + 1):
    print(f"\nMahasiswa ke-{i}:")
    nama = input("  Nama  : ").strip().title()
    nilai = float(input("  Nilai : "))
    data.append((nama, nilai))

# === ANALISIS DENGAN LIST COMPREHENSION ===
```

```
# Ekstrak hanya nilai—ini contoh transformasi list
semua_nilai = [n for _, n in data]

# Filter mahasiswa yang lulus
lulus = [(nama, n) for nama, n in data if n >= BATAS_LULUS]

# Filter yang tidak lulus
tidak_lulus = [(nama, n) for nama, n in data if n <
BATAS_LULUS]

# === STATISTIK DASAR ===

total = sum(semua_nilai)
rata = total / n
maks = max(semua_nilai)
mins = min(semua_nilai)

# Mahasiswa dengan nilai tertinggi (boleh lebih dari satu)
juara = [nama for nama, ni in data if ni == maks]

# Persentase kelulusan
persen_lulus = (len(lulus) / n) * 100

# === PENGURUTAN ===

# Urutkan berdasarkan nilai (descending) -- gunakan sorted,
bukan .sort()
data_urut = sorted(data, key=lambda x: x[1], reverse=True)

# === TAMPILKAN HASIL ===

print()
print("=" * 50)
print("            HASIL ANALISIS")
print("=" * 50)
print(f"Total mahasiswa  : {n}")
print(f"Rata-rata kelas  : {rata:.2f}")
print(f"Nilai tertinggi  : {maks:.2f}")
print(f"Nilai terendah   : {mins:.2f}")
print(f"Juara kelas      : {', '.join(juara)}")
print(f"Lulus            : {len(lulus)}
({persen_lulus:.1f}%)")
```

```
print(f"Tidak lulus      : {len(tidak_lulus)} ({100 -
persen_lulus:.1f}%)")

print("\n--- RANKING ---")
for peringkat, (nama, nilai) in enumerate(data_urut, start=1):
    print(f"{peringkat:2}. {nama:<20} {nilai:>6.2f}")

if tidak_lulus:
    print("\n--- PERLU PERHATIAN ---")
    for nama, nilai in tidak_lulus:
        kurang = BATAS_LULUS - nilai
        print(f"   {nama}: kurang {kurang:.2f} dari batas
lulus")
```

Contoh eksekusi (input 4 mahasiswa):

```
==================================================
       ANALISIS NILAI KELAS
==================================================
Jumlah mahasiswa: 4

Mahasiswa ke-1:
  Nama  : andi
  Nilai : 85
...

==================================================
            HASIL ANALISIS
==================================================
Total mahasiswa  : 4
Rata-rata kelas  : 72.50
Nilai tertinggi  : 92.00
Nilai terendah   : 55.00
Juara kelas      : Cici
Lulus            : 3 (75.0%)
Tidak lulus      : 1 (25.0%)

--- RANKING ---
 1. Cici                 92.00
 2. Andi                 85.00
 3. Budi                 58.00
 4. Eko                  55.00
```

Skrip ini menggabungkan: list of tuples untuk data terstruktur, list comprehension untuk filtering dan transformasi, fungsi-fungsi agregat (sum, max, min), pengurutan dengan sorted() dan parameter key (akan dibahas tuntas di Bab 11), serta enumerate() untuk iterasi dengan indeks.

9.5 Kesalahan Umum dan Cara Mengatasinya

Kesalahan 1: Mengira .sort() mengembalikan list baru.

Miskonsepsi: "Saya tulis hasil = nilai.sort() tapi hasil = None."

Realita: Metode .sort() memodifikasi list AS-IS dan mengembalikan None. Jika Anda ingin mengurutkan tanpa mengubah list asli, gunakan fungsi sorted(nilai) yang mengembalikan list baru. Ini berlaku juga untuk .reverse() vs reversed().

Kesalahan 2: Memodifikasi list yang sedang diiterasi.

Miskonsepsi: "Saya hapus elemen di tengah for loop, kok ada yang terlewat?"

Realita: Memodifikasi list yang sedang diiterasi menyebabkan perilaku tidak terduga: indeks bergeser tetapi loop tetap melanjutkan ke indeks berikutnya. Solusi: iterasi pada salinan dengan for x in list_data.copy(): atau bangun list baru dengan list comprehension.

Kesalahan 3: Tuple satu elemen tanpa koma.

Miskonsepsi: "Kenapa type((5)) bukan tuple?"

Realita: (5) adalah ekspresi bilangan dengan tanda kurung. Untuk tuple satu elemen, koma WAJIB: (5,). Atau bahkan tanpa kurung: x = 5, yang juga membuat tuple satu elemen.

Kesalahan 4: Aliasing tanpa sadar.

Miskonsepsi: "Saya bikin b = a, lalu modifikasi b, kok a ikut berubah?"

Realita: b = a tidak menyalin list, hanya membuat label baru ke objek yang sama. Untuk salinan independen: gunakan b = a.copy() atau b = a[:] atau b = list(a). Untuk list bersarang, gunakan copy.deepcopy().

Kesalahan 5: Membuat matriks dengan operator perkalian.

Miskonsepsi: "matriks = [[0]*3]*3 memberikan matriks 3x3, tapi modifikasinya aneh."

Realita: [[0]*3]*3 membuat 3 referensi ke list yang sama, sehingga modifikasi baris pertama mempengaruhi semua baris. Gunakan list comprehension: [[0]*3 for _ in range(3)] yang menjamin tiap baris adalah list independen.

9.6 Rangkuman

Bab 9 telah memperkenalkan dua koleksi terurut fundamental di Python: list dan tuple. List adalah koleksi mutable yang dapat dimodifikasi setelah pembuatan, dengan banyak metode bawaan untuk menambah (append, insert, extend), menghapus (remove, pop, clear), dan mengubah (sort, reverse). Tuple adalah versi immutable list, ditandai dengan tanda kurung biasa dan tidak dapat dimodifikasi setelah pembuatan.

List comprehension menyediakan cara ringkas dan pythonic untuk membuat list baru dari iterable, dengan opsi filtering dan transformasi. Tuple unpacking dan multiple assignment memungkinkan kode yang ekspresif, termasuk idiom klasik swap variabel `a, b = b, a` tanpa variabel sementara.

Konsep mutability membawa implikasi penting pada memori: penugasan list tidak menyalin, hanya membuat label baru ke objek yang sama (aliasing). Untuk salinan independen, gunakan .copy() atau copy.deepcopy() untuk list bersarang. List bersarang berfungsi sebagai representasi data dua dimensi (matriks, papan permainan, tabel), dengan list comprehension sebagai cara aman untuk pembuatannya.

9.7 Latihan Terbimbing

Soal 9.7.1

Soal: Tulis program yang menerima N bilangan dari pengguna, menyimpannya dalam list, lalu menampilkan rata-rata dan median (nilai tengah setelah diurutkan).

Pembahasan:

```
n = int(input("Berapa bilangan? "))
data = []

for i in range(n):
    data.append(float(input(f"Bilangan ke-{i+1}: ")))

data.sort()
rata = sum(data) / n

if n % 2 == 1:
    median = data[n // 2]
else:
    median = (data[n // 2 - 1] + data[n // 2]) / 2

print(f"Rata-rata : {rata:.2f}")
print(f"Median    : {median:.2f}")
```

Soal 9.7.2

Soal: Dengan list comprehension, buatlah: (a) list bilangan kelipatan 3 dari 1-50, (b) list panjang setiap kata dari kalimat "saya belajar python".

Pembahasan:

```
# Bagian (a)
kelipatan3 = [i for i in range(1, 51) if i % 3 == 0]
print(kelipatan3)
# [3, 6, 9, ..., 48]

# Bagian (b)
kalimat = "saya belajar python"
panjang = [len(k) for k in kalimat.split()]
print(panjang)
# [4, 7, 6]
```

Soal 9.7.3

Soal: Apa output kode berikut? Jelaskan: a = [1, 2, 3]; b = a; c = a.copy(); a.append(4); print(a, b, c)

Pembahasan: Output: [1, 2, 3, 4] [1, 2, 3, 4] [1, 2, 3]. Penjelasan: b = a membuat b sebagai label tambahan ke list yang sama (aliasing), sehingga modifikasi melalui a juga mempengaruhi b. c = a.copy() membuat list baru yang independen, sehingga c tidak terpengaruh oleh modifikasi pada a.

Soal 9.7.4

Soal: Tulis program yang menerima list nama, lalu menampilkan: (a) nama paling panjang, (b) jumlah nama yang dimulai dengan huruf vokal, (c) list nama dalam urutan abjad.

Pembahasan:

```
nama = input("Nama dipisah koma: ").split(",")
nama = [n.strip().title() for n in nama]

# (a) Nama paling panjang
terpanjang = max(nama, key=len)
print(f"Terpanjang : {terpanjang}")

# (b) Nama yang dimulai dengan vokal
vokal = [n for n in nama if n[0].lower() in "aiueo"]
print(f"Diawali vokal : {len(vokal)} nama -> {vokal}")

# (c) Urutkan abjad
urut = sorted(nama)
print(f"Urutan abjad : {urut}")
```

Soal 9.7.5

Soal: Buatlah matriks 4x4 berisi tabel perkalian (matriks[i][j] = i*j) menggunakan list comprehension bersarang.

Pembahasan:

```
# Matriks 4x4 tabel perkalian
tabel = [[(i + 1) * (j + 1) for j in range(4)] for i in
range(4)]

# Tampilkan dalam format rapi
for baris in tabel:
```

```
    for nilai in baris:
        print(f"{nilai:4}", end="")
    print()

# Output:
#    1   2   3   4
#    2   4   6   8
#    3   6   9  12
#    4   8  12  16
```

9.8 Latihan Mandiri

- **(C2)** Jelaskan perbedaan fundamental antara list dan tuple. Berikan dua skenario untuk masing-masing yang menjelaskan kapan tipe tersebut lebih cocok dipakai.
- **(C2)** Apa yang dimaksud dengan aliasing? Mengapa hal ini sering menjadi sumber bug bagi pemula?
- **(C3)** Tulis program yang menerima list bilangan dan menampilkan: terbesar kedua, terkecil kedua. Catatan: jangan gunakan fungsi bawaan max/min secara berulang; pikirkan algoritmanya.
- **(C3)** Dengan list comprehension dalam satu baris, buatlah list berisi karakter unik dari kalimat "saya suka python sekali" (case-insensitive, abaikan spasi).
- **(C3)** Tulis program yang membalik list TANPA menggunakan .reverse(), reversed(), atau slicing [::-1]. Gunakan loop manual.
- **(C3)** Tulis program yang menerima dua list bilangan dan menghasilkan list baru berisi: (a) gabungan tanpa duplikat, (b) elemen yang ada di keduanya (irisan).
- **(C4)** Analisis: a = [1, 2, [3, 4]]; b = a.copy(); b[2].append(5). Apakah a juga berubah? Jelaskan mengapa, dan bagaimana cara mendapatkan salinan yang sepenuhnya independen.
- **(C4)** Bandingkan tiga cara mengubah list ke string: (a) loop manual dengan concatenation, (b) loop manual dengan .append() lalu .join(), (c) langsung dengan .join() pada list comprehension. Bahas kompleksitas dan kebersihan kode.
- **(C4)** Studi kasus: Anda mengelola data peserta seminar berupa list of tuples (nama, NIM, hadir). Tulis program untuk: (a) hitung jumlah yang

hadir, (b) cari peserta dengan NIM tertentu, (c) urutkan berdasarkan nama abjad.

- **(C4)** Tulis program transpose matriks: matriks 3x4 menjadi 4x3 (baris jadi kolom, kolom jadi baris). Gunakan list comprehension bersarang.

9.9 Tugas Proyek Mini

"Sistem Manajemen Inventaris Mini"

Bangun sistem manajemen inventaris toko kecil dengan persyaratan teknis berikut:

- Setiap produk direpresentasikan sebagai tuple: (kode, nama, harga, stok).
- Inventaris adalah list of tuples, dimulai dengan minimal 5 produk awal yang sudah ditentukan.
- Menu utama dengan opsi: (1) Tampilkan inventaris, (2) Tambah produk, (3) Update stok, (4) Cari produk, (5) Laporan, (6) Keluar.
- Untuk "Tampilkan": tampilkan tabel rapi semua produk dengan kolom kode, nama, harga (format Rp), stok, dan total nilai (harga × stok).
- Untuk "Tambah": minta input dan tambahkan ke inventaris dengan .append().
- Untuk "Update stok": cari produk berdasarkan kode, lalu update stok. Karena tuple immutable, ganti tuple lama dengan tuple baru di indeks yang sama.
- Untuk "Cari": tampilkan produk yang cocok dengan kata kunci (case-insensitive, di kode atau nama).
- Untuk "Laporan": gunakan list comprehension untuk menghitung dan menampilkan: total nilai inventaris, produk dengan stok terendah, produk termahal, jumlah produk dengan stok < 10.

Format pengumpulan: berkas .py dan tangkapan layar pengujian minimal 3 skenario.

9.10 Refleksi Diri

Centang pernyataan berikut yang sudah Anda kuasai:

☐ Saya memahami konsep koleksi data dan tiga dimensinya (terurut, mutable, duplikat).

☐ Saya dapat membuat, mengakses, dan memodifikasi list dengan operasi dasar.

☐ Saya menguasai metode-metode list penting (append, insert, remove, pop, sort).

☐ Saya dapat menulis list comprehension untuk transformasi dan filtering data.

☐ Saya memahami perbedaan list dan tuple beserta alasan keberadaan keduanya.

☐ Saya menguasai tuple unpacking dan multiple assignment (termasuk idiom swap).

☐ Saya menyadari jebakan aliasing dan tahu cara membuat salinan independen.

☐ Saya memahami perbedaan shallow copy dan deep copy.

☐ Saya dapat menggunakan list bersarang untuk merepresentasikan data 2D.

☐ Saya tahu mengapa [[0]*n]*m berbahaya untuk membuat matriks.

ASAH OTAK 9.2

Eksperimen: di REPL, jalankan a = [1, 2, 3]; b = a; b += [4]; print(a, b). Apa yang terjadi? Lalu coba a = [1, 2, 3]; b = a; b = b + [4]; print(a, b). Mengapa hasilnya berbeda? Petunjuk: operator += pada list memodifikasi in-place, sedangkan b = b + [4] menciptakan list baru. Renungkan konsekuensinya pada kode Anda di masa depan.

BAB 10

DICTIONARY DAN SET

"A dictionary is just a key-value store. But like all simple ideas, it changes everything."
— Adaptasi atas observasi Bjarne Stroustrup tentang struktur data

---

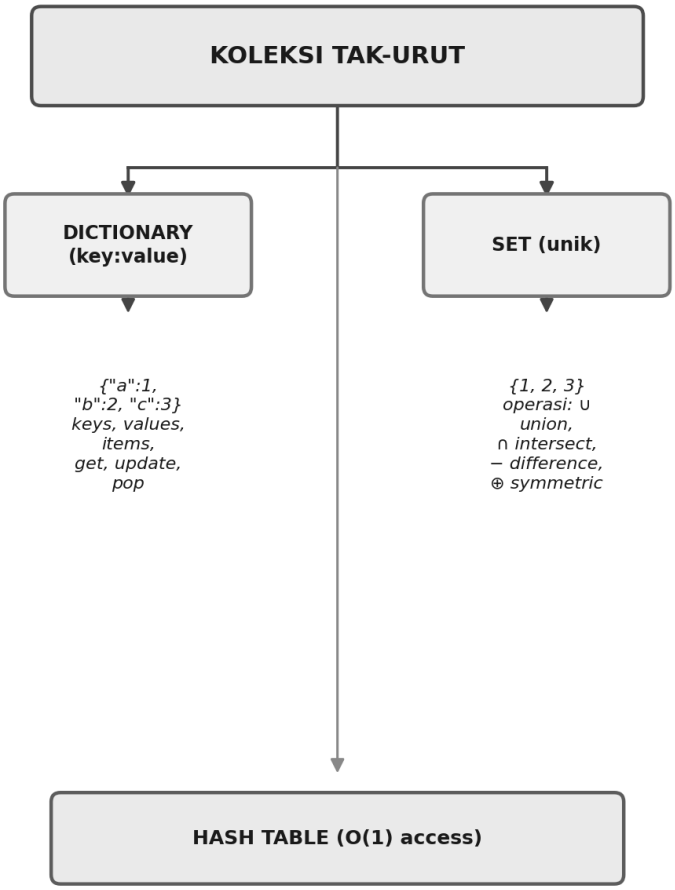
Capaian Pembelajaran Bab

Setelah mempelajari bab ini, mahasiswa diharapkan mampu mencapai sejumlah luaran pembelajaran sebagai berikut:

Kode	Capaian	Bloom
CPB-10.1	Menjelaskan konsep pasangan kunci-nilai dan tabel hash secara intuitif	C2
CPB-10.2	Membuat, mengakses, dan memodifikasi dictionary dengan operasi dasar	C3
CPB-10.3	Menerapkan metode-metode dictionary untuk manipulasi data	C3
CPB-10.4	Melakukan iterasi pada dictionary dengan berbagai cara	C3
CPB-10.5	Menyusun dictionary comprehension untuk transformasi data	C3
CPB-10.6	Menggunakan set untuk menyimpan elemen unik	C3
CPB-10.7	Menerapkan operasi himpunan (union, irisan, selisih)	C3
CPB-10.8	Memilih struktur data yang tepat antara list, tuple, dict, dan set	C4

10.1 Peta Konsep

Peta konsep berikut menggambarkan dua koleksi tak-terurut Python: dictionary (kunci-nilai) dan set (himpunan unik).



10.2 Apersepsi

Bayangkan Anda memiliki buku telepon. Untuk menemukan nomor seseorang, Anda tidak membaca setiap halaman dari awal sampai akhir—itu akan butuh jam. Sebagai gantinya, Anda membuka halaman berdasarkan huruf depan nama, lalu mencari secara cepat di halaman itu. Dalam satu menit, nomor

ditemukan. Ini adalah esensi dari **akses berbasis kunci**: kita memberikan satu informasi (nama) dan mendapatkan informasi terkait (nomor telepon) secara langsung.

Di Bab 9 kita mempelajari list, di mana setiap elemen diakses melalui *indeks numerik*. Tetapi banyak data dunia nyata tidak terorganisir berdasarkan posisi—ia terorganisir berdasarkan *kunci yang bermakna*: nama → nomor telepon, NIM → nilai, kode pos → kota. Untuk kasus seperti ini, **dictionary** adalah struktur data idealnya.

Sementara itu, **set** menyelesaikan masalah berbeda: bagaimana menyimpan koleksi elemen *unik* (tanpa duplikat) dan melakukan operasi matematis seperti gabungan dan irisan? Bersama dictionary, set melengkapi empat koleksi bawaan Python, dan setelah bab ini Anda memiliki perangkat lengkap untuk memodelkan hampir semua jenis data.

> **ASAH OTAK 10.1**
>
> Dengan list, untuk mencari satu nilai di antara satu juta elemen, komputer harus memeriksa rata-rata setengah juta elemen. Dengan dictionary, komputer dapat menemukan nilai berdasarkan kunci hampir instan, terlepas dari ukuran dictionary. Bagaimana ini bisa terjadi? Renungkan dulu sebelum membaca Subbab 10.3.1 yang membahas konsep hash table.

10.3 Uraian Materi

10.3.1 Konsep Pasangan Kunci-Nilai dan Tabel Hash

Dictionary adalah koleksi pasangan *kunci-nilai* (*key-value*). Setiap kunci memetakan ke satu nilai, dan kunci harus unik dalam satu dictionary. Sintaks: `{kunci: nilai, kunci: nilai, ...}`.

```
# Dictionary sederhana
mahasiswa = {
    "nama": "Yusri Ikhwani",
    "nim": "2026010001",
    "prodi": "Informatika",
```

```
    "ipk": 3.85
}

# Akses berdasarkan kunci
print(mahasiswa["nama"])     # Yusri Ikhwani
print(mahasiswa["ipk"])      # 3.85
```

Di balik kemampuan akses cepat ini terdapat struktur data internal bernama **hash table**. Ketika Anda menulis `mahasiswa["nama"]`, Python menjalankan fungsi hash pada string "nama" untuk mendapatkan angka, lalu menggunakan angka itu sebagai “alamat” untuk langsung ke lokasi yang menyimpan "Yusri Ikhwani". Inilah sebabnya akses dictionary memiliki kompleksitas O(1)—cepat terlepas dari ukuran data.

> **DI BALIK LAYAR 10.1**
>
> Tidak semua objek dapat menjadi kunci dictionary. Hanya objek yang HASHABLE: int, float, str, tuple (yang elemennya juga hashable), dan beberapa lainnya. List, set, dan dictionary tidak hashable karena mereka mutable—jika dipakai sebagai kunci, modifikasi akan merusak struktur hash. Aturan praktis: jika sebuah objek dapat dimodifikasi setelah pembuatan, ia tidak dapat menjadi kunci dictionary.

10.3.2 Pembuatan, Akses, dan Modifikasi Dictionary

Pembuatan

```
# Dictionary kosong
kosong = {}
kosong2 = dict()                # cara alternatif

# Dengan literal
harga = {"apel": 5000, "jeruk": 8000, "mangga": 15000}

# Dari list of tuples
pasangan = [("a", 1), ("b", 2), ("c", 3)]
huruf_angka = dict(pasangan)    # {'a': 1, 'b': 2, 'c': 3}
```

```
# Dengan argumen keyword
config = dict(host="localhost", port=8080, debug=True)

# Dari dua list dengan zip()
kunci = ["nama", "umur", "kota"]
nilai = ["Yusri", 30, "Banjarmasin"]
data = dict(zip(kunci, nilai))
```

Akses dan Modifikasi

```
harga = {"apel": 5000, "jeruk": 8000}

# Akses
harga["apel"]                    # 5000
harga["durian"]                  # KeyError!
harga.get("durian")              # None (aman, tidak error)
harga.get("durian", 0)           # 0 (default jika tidak ada)

# Pengecekan kunci
"apel" in harga                  # True
"durian" in harga                # False

# Menambah atau memodifikasi
harga["mangga"] = 15000          # tambah
harga["apel"] = 6000             # update

# Menghapus
del harga["jeruk"]               # hapus pasangan
nilai = harga.pop("apel")        # hapus dan kembalikan nilai

# Panjang
len(harga)                       # jumlah pasangan kunci-nilai
```

PERHATIAN 10.1

Akses dengan tanda kurung siku (dict["kunci"]) akan melempar KeyError jika kunci tidak ada. Untuk akses yang aman, gunakan metode .get(): dict.get("kunci") mengembalikan None jika tidak ada, dan

dict.get("kunci", default) mengembalikan nilai default. Pola ini sangat umum di kode produksi untuk menghindari crash.

10.3.3 Metode-Metode Dictionary

Metode	Fungsi	Contoh
.keys()	View semua kunci	d.keys() → dict_keys([...])
.values()	View semua nilai	d.values() → dict_values([...])
.items()	View pasangan (key, value)	d.items() → dict_items([...])
.get(k, default)	Akses aman dengan nilai default	d.get("x", 0)
.update(d2)	Gabungkan dengan dictionary lain	d.update({"x": 1})
.pop(k)	Hapus dan kembalikan nilai	v = d.pop("x")
.popitem()	Hapus pasangan terakhir	d.popitem()
.setdefault(k, v)	Get nilai, set v jika belum ada	d.setdefault("x", 0)
.clear()	Hapus semua pasangan	d.clear()
.copy()	Shallow copy	d2 = d.copy()

Demonstrasi metode-metode yang sering dipakai:

```
nilai = {"Andi": 85, "Budi": 72, "Cici": 91}

# View kunci, nilai, item
list(nilai.keys())                # ['Andi', 'Budi', 'Cici']
list(nilai.values())              # [85, 72, 91]
list(nilai.items())               # [('Andi', 85), ('Budi',
72), ('Cici', 91)]
```

```
# Penggabungan
tambahan = {"Dedi": 78, "Cici": 95}    # Cici ada di kedua dict
nilai.update(tambahan)            # Cici di-update jadi 95
print(nilai)
# {'Andi': 85, 'Budi': 72, 'Cici': 95, 'Dedi': 78}

# Akses aman dengan default
ipk = nilai.get("Eko", 0)         # 0 (Eko tidak ada)
```

10.3.4 Iterasi pada Dictionary

Ada tiga cara iterasi pada dictionary, masing-masing dengan kegunaan berbeda:

```
harga = {"apel": 5000, "jeruk": 8000, "mangga": 15000}

# Cara 1: iterasi atas kunci (default)
for buah in harga:
    print(buah)
# Output: apel jeruk mangga

# Cara 2: iterasi atas nilai
for h in harga.values():
    print(h)
# Output: 5000 8000 15000

# Cara 3: iterasi atas pasangan (PALING UMUM)
for buah, h in harga.items():
    print(f"{buah}: Rp{h}")
# Output:
# apel: Rp5000
# jeruk: Rp8000
# mangga: Rp15000
```

> **TIPS PRO 10.1**
>
> Sejak Python 3.7, dictionary mempertahankan urutan penyisipan (insertion order). Artinya ketika Anda iterasi, urutan akan sama dengan urutan Anda menambahkan kunci-nilai. Ini berbeda dari versi Python

yang lebih lama. Anda dapat memanfaatkan fakta ini untuk membuat dictionary yang "terurut" tanpa perlu modul OrderedDict yang sekarang sudah redundan.

10.3.5 Dictionary Comprehension

Mirip dengan list comprehension di Bab 9, Python juga mendukung *dictionary comprehension* untuk membuat dict baru dari iterable lain.

```
# Sintaks: {kunci: nilai for elemen in iterable if kondisi}

# Kuadrat dari 1 sampai 5
kuadrat = {i: i**2 for i in range(1, 6)}
# {1: 1, 2: 4, 3: 9, 4: 16, 5: 25}

# Konversi celsius ke fahrenheit
celsius = [0, 10, 20, 30, 40]
fahrenheit = {c: c * 9/5 + 32 for c in celsius}
# {0: 32.0, 10: 50.0, 20: 68.0, 30: 86.0, 40: 104.0}

# Filter dengan if
nilai = {"Andi": 85, "Budi": 50, "Cici": 91, "Dedi": 68}
lulus = {nama: n for nama, n in nilai.items() if n >= 60}
# {'Andi': 85, 'Cici': 91, 'Dedi': 68}

# Membalik kunci dan nilai
asli = {"a": 1, "b": 2, "c": 3}
terbalik = {v: k for k, v in asli.items()}
# {1: 'a', 2: 'b', 3: 'c'}
```

10.3.6 Set: Konsep Himpunan

Set adalah koleksi tak-terurut dari elemen *unik*. Set berguna untuk: menghilangkan duplikat, pengecekan keanggotaan yang cepat, dan operasi matematis himpunan.

```
# Pembuatan set
kosong = set()                      # WAJIB pakai set() bukan
{}
                                    # {} adalah dict kosong
warna = {"merah", "hijau", "biru"}
angka = {1, 2, 3, 4, 5}

# Dari iterable lain (menghilangkan duplikat)
data = [1, 2, 2, 3, 3, 3, 4]
unik = set(data)                    # {1, 2, 3, 4}

# Operasi dasar
warna.add("kuning")                 # tambah elemen
warna.remove("merah")               # hapus (error jika tidak
ada)
warna.discard("ungu")               # hapus aman (tidak error)
warna.pop()                         # hapus elemen sembarang

# Pengecekan
"merah" in warna                    # True/False (O(1) cepat)
len(warna)                          # jumlah elemen

# Iterasi (urutan tidak dijamin)
for w in warna:
    print(w)
```

10.3.7 Operasi Himpunan

Set mendukung operasi himpunan matematis dengan dua sintaks: operator atau metode.

Operasi	Operator	Metode	Makna
Union	A | B	A.union(B)	Gabungan A dan B
Intersection	A & B	A.intersection(B)	Elemen di A dan B
Difference	A - B	A.difference(B)	Di A tapi tidak di B

Operasi	Operator	Metode	Makna
Symmetric diff	A ^ B	A.symmetric_difference(B)	Di A atau B saja
Subset	A <= B	A.issubset(B)	A subset dari B?
Superset	A >= B	A.issuperset(B)	A superset dari B?

```
A = {1, 2, 3, 4, 5}
B = {4, 5, 6, 7, 8}

A | B           # {1, 2, 3, 4, 5, 6, 7, 8}  -- gabungan
A & B           # {4, 5}                     -- irisan
A - B           # {1, 2, 3}                  -- selisih A\B
B - A           # {6, 7, 8}                  -- selisih B\A
A ^ B           # {1, 2, 3, 6, 7, 8}         -- selisih
simetris

# Contoh aplikatif: mata kuliah yang diambil dua mahasiswa
andi = {"Algoritma", "Kalkulus", "Fisika", "Bahasa"}
budi = {"Algoritma", "Statistika", "Bahasa", "Kimia"}

sama = andi & budi              # mata kuliah yang sama-sama
diambil
khusus_andi = andi - budi       # khusus Andi
semua = andi | budi             # gabungan keduanya
```

10.3.8 frozenset: Set yang Immutable

Layaknya tuple adalah versi immutable dari list, **frozenset** adalah versi immutable dari set. Karena hashable, frozenset dapat menjadi kunci dictionary atau elemen set lain.

```
fs = frozenset([1, 2, 3])
fs.add(4)                       # AttributeError!

# Penggunaan praktis: kunci dictionary yang berupa kumpulan
```

```
skill_set = {
    frozenset({"Python", "SQL"}): "Data Engineer",
    frozenset({"HTML", "CSS", "JS"}): "Frontend",
    frozenset({"Python", "ML", "Statistics"}): "Data Scientist"
}

kemampuan = frozenset({"Python", "SQL"})
print(skill_set[kemampuan])      # Data Engineer
```

WAWASAN INDUSTRI 10.1

Dictionary adalah struktur data paling banyak dipakai di Python, bahkan oleh Python itu sendiri: namespace modul, atribut kelas, variabel lokal—semuanya disimpan dalam dictionary di balik layar. JSON (JavaScript Object Notation), format pertukaran data dominan di web modern, langsung memetakan ke dictionary Python. API REST yang Anda konsumsi dari layanan online seperti GitHub, Twitter, atau bank biasanya mengembalikan JSON yang kemudian Anda olah sebagai dictionary.

10.4 Contoh Kode Lengkap

Skrip berikut adalah *Analisis Frekuensi Kata*: menerima sebuah kalimat dan menghitung frekuensi setiap kata, lalu menampilkan statistik dengan dictionary dan set.

```
"""
Program: Analisis Frekuensi Kata
Tujuan : Demonstrasi dictionary dan set untuk analisis teks
"""

kalimat = input("Masukkan kalimat (boleh panjang): ").lower()

# Bersihkan tanda baca sederhana
for tanda in ".,!?;:":
    kalimat = kalimat.replace(tanda, "")

# Pecah jadi list kata
kata_list = kalimat.split()
```

```
# === ANALISIS DENGAN SET ===
kata_unik = set(kata_list)
kata_pendek = {k for k in kata_unik if len(k) <= 3}
kata_panjang = {k for k in kata_unik if len(k) > 7}

# === ANALISIS DENGAN DICTIONARY ===
# Frekuensi kemunculan setiap kata
frekuensi = {}
for kata in kata_list:
    frekuensi[kata] = frekuensi.get(kata, 0) + 1

# Versi dengan dict comprehension
# (alternatif: hanya kata yang muncul lebih dari sekali)
berulang = {k: v for k, v in frekuensi.items() if v > 1}

# === STATISTIK ===
total_kata = len(kata_list)
total_unik = len(kata_unik)
tingkat_keragaman = (total_unik / total_kata) * 100 if
total_kata else 0

# Kata paling sering muncul (gunakan max dengan key)
if frekuensi:
    paling_sering = max(frekuensi, key=frekuensi.get)
    jumlah_sering = frekuensi[paling_sering]

# === OUTPUT ===
print()
print("=" * 50)
print("        ANALISIS FREKUENSI KATA")
print("=" * 50)
print(f"Total kata        : {total_kata}")
print(f"Kata unik         : {total_unik}")
print(f"Tingkat keragaman : {tingkat_keragaman:.1f}%")
print(f"Kata pendek (≤3) : {len(kata_pendek)}")
print(f"Kata panjang (>7) : {len(kata_panjang)}")

if frekuensi:
    print(f"\nPaling sering     : '{paling_sering}'
({jumlah_sering}x)")

if berulang:
    print("\n--- Kata yang berulang ---")
```

```
    # Urutkan berdasarkan frekuensi tertinggi
    for kata, jumlah in sorted(berulang.items(), key=lambda x:
-x[1]):
        print(f"  {kata}: {jumlah}x")
```

Contoh eksekusi:

```
Masukkan kalimat: Saya suka belajar Python. Python sangat
menyenangkan
dan Python juga berguna untuk belajar data.

==================================================
        ANALISIS FREKUENSI KATA
==================================================
Total kata        : 13
Kata unik         : 11
Tingkat keragaman : 84.6%
Kata pendek (≤3) : 4
Kata panjang (>7) : 2

Paling sering     : 'python' (3x)

--- Kata yang berulang ---
  python: 3x
  belajar: 2x
```

10.5 Kesalahan Umum dan Cara Mengatasinya

Kesalahan 1: KeyError pada akses dict.

Miskonsepsi: "Saya tulis nilai = data[\"x\"] dan dapat error."

Realita: Jika kunci tidak ada, akses dengan tanda kurung siku melempar KeyError. Gunakan .get(kunci) untuk akses aman yang mengembalikan None, atau .get(kunci, default) untuk nilai default. Atau cek dulu dengan if kunci in dict:

Kesalahan 2: Menggunakan list sebagai kunci dict.

Miskonsepsi: "Saya bikin dict dengan kunci list, kok error?"

Realita: Kunci dictionary harus hashable. List mutable sehingga tidak hashable. Solusi: gunakan tuple sebagai kunci. d = {("a", "b"): 1} valid, tetapi d = {["a", "b"]: 1} error.

Kesalahan 3: {} dianggap set kosong.

Miskonsepsi: "Saya tulis s = {} sebagai set kosong tapi type(s) menunjukkan dict."

Realita: Sintaks {} di Python adalah dict kosong, bukan set kosong. Untuk set kosong: gunakan s = set(). {1, 2, 3} memang set, tapi {} adalah dict.

Kesalahan 4: Mengandalkan urutan set.

Miskonsepsi: "Set saya kadang menampilkan {1, 2, 3}, kadang {3, 1, 2}."

Realita: Set tidak menjamin urutan. Jika urutan penting, jangan pakai set. Gunakan list atau dict (yang mempertahankan urutan sejak Python 3.7). Set hanya menjamin keunikan elemen dan akses cepat.

Kesalahan 5: Memodifikasi dict saat iterasi.

Miskonsepsi: "Saya hapus kunci di tengah loop dan kena RuntimeError."

Realita: Python tidak mengizinkan modifikasi dict yang sedang diiterasi. Solusi: iterasi pada list(dict.keys()) untuk membuat salinan kunci, atau bangun dict baru dengan dict comprehension.

10.6 Rangkuman

Bab 10 melengkapi pengenalan empat koleksi bawaan Python dengan dua koleksi tak-terurut: dictionary dan set. Dictionary menyimpan pasangan kunci-nilai, memungkinkan akses cepat berbasis kunci (O(1) berkat hash table internal). Kuncinya harus hashable (immutable), nilainya boleh apa saja. Dictionary mendukung berbagai metode (.get, .keys, .values, .items, .update, .pop) dan dictionary comprehension untuk pembuatan ringkas.

Set adalah koleksi elemen unik yang ideal untuk menghilangkan duplikat dan pengecekan keanggotaan cepat. Operasi himpunan matematis didukung lengkap: union (|), intersection (&), difference (-), dan symmetric difference (^). Frozenset adalah versi immutable yang dapat menjadi kunci dictionary atau elemen set lain.

Pemilihan struktur data yang tepat adalah keterampilan kunci: gunakan list untuk urutan yang dapat berulang, tuple untuk data tetap, dictionary untuk pemetaan kunci-nilai, dan set untuk koleksi unik. Bab ini juga mempersiapkan Anda untuk format data JSON yang dominan di industri modern.

10.7 Latihan Terbimbing

Soal 10.7.1

Soal: Tulis program yang menerima N data mahasiswa (nim dan nilai), menyimpannya dalam dictionary (nim sebagai kunci), lalu menampilkan mahasiswa dengan nilai tertinggi.

Pembahasan:

```
n = int(input("Jumlah mahasiswa: "))
data = {}

for i in range(n):
    nim = input(f"NIM ke-{i+1}: ")
    nilai = float(input(f"Nilai: "))
    data[nim] = nilai

# Cari NIM dengan nilai tertinggi
nim_tertinggi = max(data, key=data.get)
print(f"\nNIM tertinggi: {nim_tertinggi} dengan nilai
{data[nim_tertinggi]}")
```

Soal 10.7.2

Soal: Dengan dict comprehension, buatlah dictionary yang memetakan setiap huruf alfabet kecil ke posisinya dalam alfabet (a=1, b=2, ..., z=26).

Pembahasan:

```
alfabet = {chr(ord('a') + i): i + 1 for i in range(26)}
print(alfabet)
# {'a': 1, 'b': 2, 'c': 3, ..., 'z': 26}

# Verifikasi: posisi 'y' adalah 25
print(alfabet['y'])  # 25
```

Soal 10.7.3

Soal: Tulis program yang menerima dua daftar mata kuliah (dipisah koma), lalu tampilkan: (a) yang diambil keduanya, (b) khusus daftar pertama, (c) total mata kuliah berbeda.

Pembahasan:

```
mk1 = set(input("Daftar 1: ").split(","))
mk2 = set(input("Daftar 2: ").split(","))

# Bersihkan whitespace
mk1 = {m.strip() for m in mk1}
mk2 = {m.strip() for m in mk2}

sama = mk1 & mk2
khusus1 = mk1 - mk2
semua = mk1 | mk2

print(f"Diambil keduanya : {sama}")
print(f"Khusus daftar 1  : {khusus1}")
print(f"Total berbeda    : {len(semua)} mata kuliah")
```

Soal 10.7.4

Soal: Apa output kode: d = {"a":1,"b":2}; d.setdefault("a",10); d.setdefault("c",30); print(d)?

Pembahasan: Output: {'a': 1, 'b': 2, 'c': 30}. Penjelasan: .setdefault(k, v) hanya menambahkan v jika k belum ada. Untuk 'a' yang sudah ada dengan nilai 1, setdefault tidak mengubahnya. Untuk 'c' yang belum ada, setdefault menambahkannya dengan nilai 30.

Soal 10.7.5

Soal: Tulis program penghitung frekuensi karakter dalam sebuah kata (case-insensitive, abaikan spasi). Tampilkan hasil terurut berdasarkan frekuensi tertinggi.

Pembahasan:

```
teks = input("Teks: ").lower().replace(" ", "")

# Hitung frekuensi
frek = {}
for char in teks:
    frek[char] = frek.get(char, 0) + 1

# Urutkan dari yang terbanyak
terurut = sorted(frek.items(), key=lambda x: -x[1])

print("\nFrekuensi karakter:")
for char, jumlah in terurut:
    print(f"  '{char}': {jumlah}x")
```

10.8 Latihan Mandiri

- **(C2)** Jelaskan kapan menggunakan list, tuple, dict, dan set. Berikan satu skenario aplikatif untuk masing-masing.
- **(C2)** Apa yang dimaksud objek hashable? Mengapa list tidak hashable padahal tuple bisa?
- **(C3)** Tulis program yang menerima kalimat dan menampilkan jumlah kata unik (gunakan set).
- **(C3)** Buat dictionary yang memetakan nama negara ke ibukotanya untuk minimal 10 negara. Buatlah program yang memungkinkan pengguna mencari ibukota berdasarkan nama negara.
- **(C3)** Tulis program yang menerima dua list bilangan dan menggunakan set untuk menampilkan: gabungan, irisan, dan elemen yang hanya ada di salah satu list.
- **(C3)** Dengan dict comprehension, buat dictionary yang memetakan bilangan 1-20 ke statusnya: "genap" atau "ganjil".
- **(C4)** Analisis kode: d = {"a": [1,2,3]}; e = d.copy(); e["a"].append(4); print(d). Apa output dan mengapa? Bagaimana memperbaikinya?
- **(C4)** Bandingkan kecepatan operasi 'in' pada list vs set untuk pencarian 1 elemen di antara 100.000 data. Gunakan timeit untuk pengukuran.

- **(C4)** Studi kasus: tulis program manajemen perpustakaan dengan dict (buku -> list peminjam) dan set (anggota terdaftar). Sediakan menu untuk meminjam, mengembalikan, dan melihat status.
- **(C4)** Diskusikan: mengapa Python 3.7+ menjamin urutan insertion pada dict, sedangkan versi sebelumnya tidak? Apa implikasi praktisnya pada migrasi kode?

10.9 Tugas Proyek Mini

"Sistem Manajemen Kontak Sederhana"

Bangun aplikasi manajemen kontak dengan persyaratan:

- Kontak disimpan dalam dictionary: kunci adalah nama (lowercase), nilai adalah dictionary nested berisi nomor_hp, email, kategori.
- Kategori adalah set yang berisi minimal: "keluarga", "teman", "kerja", "dosen", "lainnya".
- Menu: (1) Tambah kontak, (2) Cari kontak, (3) Update kontak, (4) Hapus kontak, (5) Tampilkan per kategori, (6) Statistik, (7) Keluar.
- Validasi: nama tidak boleh duplikat, kategori harus ada di set yang valid, nomor HP minimal 10 digit.
- Untuk Statistik: jumlah total kontak, jumlah per kategori (gunakan dict), kategori dengan kontak terbanyak.
- Untuk pencarian: dukung pencarian parsial (substring) dan case-insensitive.

Format pengumpulan: berkas .py beserta tangkapan layar pengujian 3 skenario.

10.10 Refleksi Diri

Centang pernyataan berikut yang sudah Anda kuasai:

☐ Saya memahami konsep pasangan kunci-nilai dan kegunaannya.

☐ Saya tahu apa itu hash table dan mengapa akses dict cepat O(1).

☐ Saya dapat membuat, mengakses, memodifikasi, dan menghapus elemen dict.

☐ Saya menguasai metode .get(), .keys(), .values(), .items() pada dict.

☐ Saya dapat melakukan iterasi pada dict dengan tiga cara berbeda.

☐ Saya dapat menulis dictionary comprehension dengan filter.

☐ Saya memahami konsep set dan kapan menggunakannya.

☐ Saya menguasai keempat operasi himpunan (|, &, -, ^).

☐ Saya tahu perbedaan set dan frozenset, serta kapan menggunakan masing-masing.

☐ Saya dapat memilih struktur data yang tepat antara list, tuple, dict, set.

ASAH OTAK 10.2

Eksperimen: buat list 100.000 angka, lalu cari apakah angka 99999 ada di dalamnya menggunakan 'in'. Sekarang konversi list itu ke set dan lakukan pencarian yang sama. Gunakan modul time untuk mengukur. Anda akan menyaksikan perbedaan signifikan: pencarian di set jauh lebih cepat. Renungkan mengapa: list harus dicek satu per satu (O(n)), set langsung menghitung hash (O(1)).

BAB 11

FUNGSI DAN PARAMETER

"Functions are the building blocks of readable, maintainable, and reusable code."
— Brian Kernighan, salah satu pencipta bahasa C

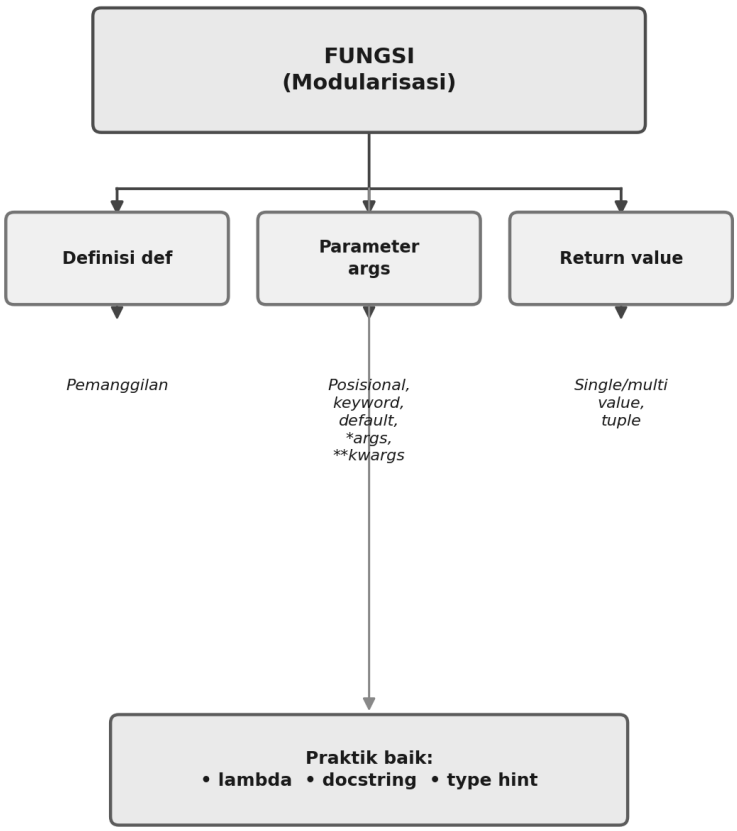
Capaian Pembelajaran Bab

Setelah mempelajari bab ini, mahasiswa diharapkan mampu mencapai sejumlah luaran pembelajaran sebagai berikut:

Kode	Capaian	Bloom
CPB-11.1	Menjelaskan konsep modularisasi dan prinsip DRY (Don't Repeat Yourself)	C2
CPB-11.2	Mendefinisikan, memanggil, dan mengembalikan nilai dari fungsi	C3
CPB-11.3	Menggunakan parameter posisional dan keyword argument	C3
CPB-11.4	Menerapkan nilai default parameter dan menghindari mutable default pitfall	C3
CPB-11.5	Menggunakan *args dan **kwargs untuk fungsi dengan parameter variabel	C3
CPB-11.6	Menulis dan menggunakan fungsi lambda untuk kasus sederhana	C3
CPB-11.7	Menerapkan docstring (PEP 257) dan type hint (PEP 484) sebagai praktik profesional	C3
CPB-11.8	Melakukan refaktorisasi kode panjang menjadi fungsi-fungsi yang lebih kecil	C4

11.1 Peta Konsep

Peta konsep berikut menggambarkan komponen-komponen fungsi sebagai unit modularisasi kode.



11.2 Apersepsi

Bayangkan Anda harus menulis program yang menghitung luas lingkaran berkali-kali, dalam puluhan bagian berbeda. Tanpa fungsi, Anda akan menyalin-tempel formula `luas = 3.14159 * r * r` puluhan kali. Bayangkan jika kemudian ditemukan bahwa konstanta yang lebih akurat adalah 3.14159265—Anda harus mengubahnya di puluhan tempat. Bayangkan jika ada perubahan kebijakan, atau penambahan validasi: bencana.

Fungsi menyelesaikan masalah ini dengan elegan. Tulis logika sekali, beri nama, lalu gunakan berulang kali. Jika ada perubahan, ubah di satu tempat—semua pemanggilannya otomatis menggunakan versi baru. Inilah inti dari prinsip *DRY (Don't Repeat Yourself)*, salah satu pilar utama rekayasa perangkat lunak modern.

Tetapi fungsi lebih dari sekadar alat pengulangan. Ia adalah **unit abstraksi**, yang memungkinkan kita berpikir pada level yang lebih tinggi. Ketika kita memanggil `print("halo")`, kita tidak peduli bagaimana karakter ditampilkan di layar—itu detail yang sudah disembunyikan oleh fungsi. Demikian pula, kita dapat membuat fungsi-fungsi sendiri yang menyembunyikan kompleksitas dari sisa program. Inilah kekuatan terbesar fungsi: ia membuat program besar dapat dipikirkan oleh otak manusia yang terbatas.

> **ASAH OTAK 11.1**
>
> Tanpa fungsi, satu program yang berisi 10.000 baris kode adalah satu blok raksasa yang harus dipikirkan sekaligus. Dengan fungsi, program yang sama bisa dipecah menjadi 100 fungsi dengan rata-rata 100 baris masing-masing. Setiap fungsi dapat dipahami secara terpisah. Renungkan: berapa banyak hal yang dapat Anda pikirkan sekaligus? Penelitian psikologi kognitif menyebut angka 7±2. Bagaimana ini menjelaskan mengapa fungsi adalah keharusan, bukan kemewahan?

11.3 Uraian Materi

11.3.1 Konsep Modularisasi dan Prinsip DRY

Modularisasi adalah praktik memecah program besar menjadi bagian-bagian kecil yang independen. Setiap bagian (modul) memiliki tugas spesifik dan dapat dipahami secara terpisah. Fungsi adalah unit modularisasi paling dasar di Python.

Empat manfaat utama modularisasi:

- **Reusability.** Tulis sekali, pakai berkali-kali. Logika yang dipakai di banyak tempat hanya didefinisikan di satu lokasi.

- **Maintainability.** Perubahan di satu fungsi otomatis berlaku di semua pemanggilan. Tidak perlu mengejar baris-baris yang sama tersebar di banyak tempat.
- **Testability.** Fungsi kecil dapat diuji secara terisolasi. Bug menjadi lebih mudah diisolasi dan diperbaiki.
- **Readability.** Pembaca kode tidak perlu memahami semua detail sekaligus; ia bisa berfokus pada satu fungsi pada satu waktu.

> **CATATAN 11.1**
>
> Prinsip DRY pertama kali dirumuskan oleh Andy Hunt dan Dave Thomas dalam buku The Pragmatic Programmer (1999): "Every piece of knowledge must have a single, unambiguous, authoritative representation within a system." Pelanggaran prinsip DRY adalah salah satu sumber bug terbesar di proyek besar. Jika satu fakta dinyatakan di banyak tempat, cepat atau lambat versi-versi tersebut akan tidak sinkron.

11.3.2 Definisi, Pemanggilan, dan Return Value

Sintaks dasar fungsi di Python:

```
def nama_fungsi(parameter1, parameter2, ...):
    """Docstring (opsional tapi sangat direkomendasikan)."""
    # body fungsi
    return nilai     # opsional
```

Contoh sederhana:

```
def luas_lingkaran(jari_jari):
    """Menghitung luas lingkaran berdasarkan jari-jari."""
    pi = 3.14159
    return pi * jari_jari ** 2

# Pemanggilan
hasil = luas_lingkaran(5)
print(hasil)                # 78.53975

# Pemanggilan langsung tanpa simpan
print(luas_lingkaran(10))   # 314.159
```

Fungsi Tanpa Return

Fungsi yang tidak memiliki return secara eksplisit akan mengembalikan None. Fungsi seperti ini biasanya digunakan untuk efek samping (side effect) seperti mencetak ke layar.

```
def sapa(nama):
    print(f"Halo, {nama}!")

hasil = sapa("Yusri")     # mencetak "Halo, Yusri!"
print(hasil)              # None
```

Fungsi Mengembalikan Banyak Nilai

Python dapat mengembalikan beberapa nilai sekaligus (sebenarnya tuple), yang sangat praktis dengan unpacking dari Bab 9.

```
def statistik(angka):
    """Mengembalikan jumlah, rata-rata, min, dan max."""
    total = sum(angka)
    rata = total / len(angka)
    return total, rata, min(angka), max(angka)

# Unpacking saat pemanggilan
t, r, mn, mx = statistik([78, 85, 92, 67, 88])
print(f"Total: {t}, Rata: {r:.2f}, Min: {mn}, Max: {mx}")
```

11.3.3 Parameter Posisional dan Keyword Argument

Saat memanggil fungsi, Anda dapat memberikan argumen dengan dua cara: *posisional* (berdasarkan urutan) atau *keyword* (berdasarkan nama).

```
def luas_persegi_panjang(panjang, lebar):
    return panjang * lebar

# Posisional: urutan menentukan
luas_persegi_panjang(5, 3)         # 15 (panjang=5, lebar=3)
```

```
luas_persegi_panjang(3, 5)         # 15 tapi maknanya beda!

# Keyword: nama parameter eksplisit
luas_persegi_panjang(panjang=5, lebar=3)
luas_persegi_panjang(lebar=3, panjang=5)   # urutan boleh
berbeda

# Campuran: posisional dulu, baru keyword
luas_persegi_panjang(5, lebar=3)

# Ini ERROR: keyword sebelum posisional
# luas_persegi_panjang(panjang=5, 3)
```

TIPS PRO 11.1

Untuk fungsi dengan banyak parameter (apalagi yang bertipe sama), gunakan keyword argument untuk memperjelas. Bandingkan: kirim_email("yusri@email.com", "halo", True, False, 3) dengan kirim_email(to="yusri@email.com", subject="halo", html=True, attach=False, retry=3). Versi kedua jauh lebih mudah dibaca tanpa harus melihat definisi fungsi. Ini disebut self-documenting code.

11.3.4 Nilai Default Parameter

Parameter dapat memiliki nilai default yang dipakai jika argumen tidak diberikan.

```
def sapa(nama, salam="Halo"):
    return f"{salam}, {nama}!"

print(sapa("Yusri"))              # Halo, Yusri!
print(sapa("Yusri", "Selamat pagi"))  # Selamat pagi, Yusri!
print(sapa("Yusri", salam="Hai")) # Hai, Yusri!
```

Aturan penting: parameter dengan default harus diletakkan SETELAH parameter tanpa default.

```
# BENAR
def f(a, b, c=10): ...

# SALAH (SyntaxError)
def f(a, b=5, c): ...
```

PERHATIAN 11.1

MUTABLE DEFAULT PITFALL—salah satu jebakan paling terkenal di Python. Jangan pernah menggunakan list, dict, atau objek mutable lainnya sebagai nilai default! SALAH: def tambah_item(item, list=[]): list akan DIBAGI di antara semua pemanggilan! BENAR: def tambah_item(item, list=None): if list is None: list = [] Pola list=None ini adalah idiom standar yang HARUS Anda kuasai.

11.3.5 *args dan **kwargs

Bagaimana jika kita ingin membuat fungsi yang menerima jumlah argumen yang tidak diketahui sebelumnya? Python menyediakan dua mekanisme: `*args` untuk argumen posisional variabel, dan `**kwargs` untuk argumen keyword variabel.

*args: Tuple Argumen Posisional

```
def total(*angka):
    """Menjumlahkan sebanyak apapun argumen."""
    print(f"Tipe args: {type(angka)}")   # <class 'tuple'>
    return sum(angka)

print(total(1, 2, 3))                    # 6
print(total(1, 2, 3, 4, 5, 6))           # 21
print(total())                           # 0

# Membuka list/tuple menjadi argumen dengan *
data = [10, 20, 30]
print(total(*data))                      # 60
```

***kwargs: Dictionary Argumen Keyword*

```
def info(**data):
    """Menerima sebanyak apapun keyword argument."""
    print(f"Tipe kwargs: {type(data)}")   # <class 'dict'>
    for kunci, nilai in data.items():
        print(f"{kunci}: {nilai}")

info(nama="Yusri", umur=30, kota="Banjarmasin")
# Output:
# nama: Yusri
# umur: 30
# kota: Banjarmasin

# Membuka dict menjadi keyword arguments dengan **
data = {"nama": "Andi", "umur": 25}
info(**data)
```

Kombinasi Lengkap

```
def fungsi_lengkap(a, b, *args, default=10, **kwargs):
    print(f"a={a}, b={b}")
    print(f"args={args}")
    print(f"default={default}")
    print(f"kwargs={kwargs}")

fungsi_lengkap(1, 2, 3, 4, 5, default=20, x=100, y=200)
# Output:
# a=1, b=2
# args=(3, 4, 5)
# default=20
# kwargs={'x': 100, 'y': 200}
```

11.3.6 Lambda: Fungsi Anonim

Lambda adalah cara ringkas mendefinisikan fungsi pendek dalam satu baris. Berguna untuk fungsi sederhana yang dipakai sekali, terutama sebagai argumen ke fungsi lain seperti `sorted()`, `map()`, dan `filter()`.

```
# Sintaks: lambda parameter: ekspresi

# Fungsi biasa
def kuadrat(x):
    return x ** 2

# Setara dengan lambda
kuadrat = lambda x: x ** 2
print(kuadrat(5))                  # 25

# Penggunaan paling umum: sebagai argumen sort
mahasiswa = [("Andi", 85), ("Budi", 72), ("Cici", 91)]

# Urutkan berdasarkan nilai (elemen ke-2 tiap tuple)
urut = sorted(mahasiswa, key=lambda m: m[1])
# [('Budi', 72), ('Andi', 85), ('Cici', 91)]

# Urutkan menurun
urut_desc = sorted(mahasiswa, key=lambda m: -m[1])

# map: terapkan fungsi ke setiap elemen
kuadrat_list = list(map(lambda x: x**2, [1, 2, 3, 4]))
# [1, 4, 9, 16]

# filter: ambil hanya yang memenuhi kondisi
genap = list(filter(lambda x: x % 2 == 0, range(10)))
# [0, 2, 4, 6, 8]
```

CATATAN 11.2

Lambda di Python sengaja dibatasi hanya satu ekspresi (tidak boleh ada statement seperti if-else multiline, loop, atau assignment). Ini adalah keputusan desain Guido van Rossum: jika logika lambda Anda terlalu kompleks, gunakan fungsi biasa dengan def. Aturan praktis: jika lambda Anda lebih panjang dari 80 karakter atau tidak muat di satu baris yang jelas, kembalilah ke def.

11.3.7 Docstring dan Type Hint: Praktik Profesional

Docstring (PEP 257)

Docstring adalah string yang ditulis sebagai baris pertama di dalam fungsi untuk mendokumentasikan tujuan, parameter, dan return value-nya. Dapat diakses melalui atribut __doc__ atau fungsi help().

```
def hitung_bmi(berat, tinggi_cm):
    """Menghitung Indeks Massa Tubuh.

    Args:
        berat (float): Berat badan dalam kilogram.
        tinggi_cm (float): Tinggi badan dalam sentimeter.

    Returns:
        float: Nilai BMI.

    Contoh:
        >>> hitung_bmi(65, 170)
        22.49
    """
    tinggi_m = tinggi_cm / 100
    return berat / (tinggi_m ** 2)

# Mengakses docstring
print(hitung_bmi.__doc__)
help(hitung_bmi)
```

Type Hint (PEP 484)

Sejak Python 3.5, kita dapat menyertakan *type hint* untuk parameter dan return value. Type hint tidak mengubah perilaku Python (Python tetap dynamic typing), tetapi sangat membantu pembaca kode dan alat seperti mypy dan IDE.

```
def hitung_bmi(berat: float, tinggi_cm: float) -> float:
    """Menghitung Indeks Massa Tubuh."""
    tinggi_m = tinggi_cm / 100
    return berat / (tinggi_m ** 2)

# Type hint kompleks
```

```
from typing import List, Dict, Optional

def rata_rata(nilai: List[float]) -> float:
    return sum(nilai) / len(nilai)

def cari_mahasiswa(nim: str, data: Dict[str, str]) ->
Optional[str]:
    return data.get(nim)
```

WAWASAN INDUSTRI 11.1

Type hint awalnya kontroversial di komunitas Python—"kenapa Python harus jadi mirip Java?" Tetapi sejak adopsi luas di perusahaan besar (Instagram, Dropbox, Microsoft, dll), type hint kini menjadi standar de facto untuk kode Python profesional. Tools seperti mypy dan Pylance dapat menangkap banyak bug sebelum kode dijalankan, dan IDE memberikan autocomplete yang jauh lebih baik dengan type hint. Pembiasaan menulis type hint sejak awal akan sangat berharga di karier Anda.

11.3.8 Studi Kasus: Refaktorisasi Kode Panjang

Salah satu kemampuan paling penting yang dibawa fungsi adalah *refaktorisasi*: menyederhanakan kode yang sudah ada menjadi struktur yang lebih bersih tanpa mengubah perilakunya. Mari kita lihat contoh.

Sebelum: Kode Monolitik

```
# Versi tanpa fungsi (sulit dibaca dan dipertahankan)
print("=== KALKULATOR NILAI ===")
uts = float(input("UTS: "))
uas = float(input("UAS: "))
tugas = float(input("Tugas: "))
akhir = 0.3 * uts + 0.4 * uas + 0.3 * tugas
if akhir >= 85:
    huruf = "A"
elif akhir >= 75:
    huruf = "B"
```

```
elif akhir >= 65:
    huruf = "C"
elif akhir >= 55:
    huruf = "D"
else:
    huruf = "E"
lulus = akhir >= 60
print(f"Akhir: {akhir}")
print(f"Huruf: {huruf}")
print(f"Status: {'LULUS' if lulus else 'TIDAK LULUS'}")
```

Sesudah: Kode Termodularisasi

```
def baca_nilai_komponen() -> tuple[float, float, float]:
    """Membaca UTS, UAS, dan Tugas dari pengguna."""
    uts = float(input("UTS: "))
    uas = float(input("UAS: "))
    tugas = float(input("Tugas: "))
    return uts, uas, tugas

def hitung_nilai_akhir(uts: float, uas: float, tugas: float) -
> float:
    """Menghitung nilai akhir dengan bobot 30-40-30."""
    return 0.3 * uts + 0.4 * uas + 0.3 * tugas

def konversi_huruf(nilai: float) -> str:
    """Mengonversi nilai angka ke huruf A-E."""
    if nilai >= 85: return "A"
    if nilai >= 75: return "B"
    if nilai >= 65: return "C"
    if nilai >= 55: return "D"
    return "E"

def tampilkan_hasil(nilai: float, huruf: str, lulus: bool) ->
None:
    """Mencetak hasil akhir."""
    print(f"Akhir: {nilai:.2f}")
    print(f"Huruf: {huruf}")
    print(f"Status: {'LULUS' if lulus else 'TIDAK LULUS'}")

# Program utama jauh lebih jelas
print("=== KALKULATOR NILAI ===")
```

```
uts, uas, tugas = baca_nilai_komponen()
nilai_akhir = hitung_nilai_akhir(uts, uas, tugas)
huruf = konversi_huruf(nilai_akhir)
lulus = nilai_akhir >= 60
tampilkan_hasil(nilai_akhir, huruf, lulus)
```

Versi kedua lebih panjang dari segi baris total, tetapi jauh lebih *modular*, *testable*, dan *maintainable*. Setiap fungsi memiliki satu tanggung jawab (single responsibility principle), dapat diuji terpisah, dan dapat digunakan ulang di program lain. Inilah pola pikir yang membedakan programmer pemula dari profesional.

11.4 Contoh Kode Lengkap

Skrip berikut adalah *Sistem Pengelolaan Daftar Belanja* yang menerapkan modularisasi penuh dengan berbagai fitur fungsi: default parameter, *args, type hint, dan docstring.

```
"""
Program: Sistem Daftar Belanja
Tujuan : Demonstrasi modularisasi lengkap dengan berbagai
jenis parameter
"""

def tambah_item(daftar: dict, nama: str,
                harga: float, jumlah: int = 1) -> dict:
    """Menambah item ke daftar belanja.

    Args:
        daftar: dictionary daftar belanja.
        nama: nama item.
        harga: harga satuan.
        jumlah: kuantitas (default 1).

    Returns:
        dictionary yang diperbarui.
    """
    daftar[nama.lower()] = {"harga": harga, "jumlah": jumlah}
    return daftar
```

```
def hapus_item(daftar: dict, nama: str) -> bool:
    """Menghapus item; mengembalikan True jika berhasil."""
    nama = nama.lower()
    if nama in daftar:
        del daftar[nama]
        return True
    return False

def hitung_subtotal(harga: float, jumlah: int) -> float:
    """Menghitung subtotal satu item."""
    return harga * jumlah

def hitung_total(daftar: dict, *, diskon: float = 0.0) ->
float:
    """Menghitung total dengan opsi diskon (keyword-only)."""
    subtotal = sum(
        hitung_subtotal(item["harga"], item["jumlah"])
        for item in daftar.values()
    )
    return subtotal * (1 - diskon)

def tampilkan_daftar(daftar: dict, judul: str = "DAFTAR
BELANJA") -> None:
    """Mencetak daftar dalam format tabel."""
    print("=" * 50)
    print(f"  {judul}")
    print("=" * 50)
    if not daftar:
        print("  (kosong)")
        return
    print(f"
{'Item':<20}{'Harga':>10}{'Qty':>5}{'Subtotal':>12}")
    print("-" * 50)
    for nama, info in daftar.items():
        sub = hitung_subtotal(info["harga"], info["jumlah"])
        print(f"  {nama.title():<20}{info['harga']:>10,.0f}"
              f"{info['jumlah']:>5}{sub:>12,.0f}")
    print("-" * 50)

def kebutuhan_dasar() -> list:
    """Mengembalikan list item kebutuhan dasar yang umum."""
    return ["beras", "telur", "minyak goreng", "gula",
"garam"]
```

```
# === Program Utama ===
belanja = {}

# Tambah beberapa item
tambah_item(belanja, "Beras", 12000, 5)
tambah_item(belanja, "Telur", 28000, 2)
tambah_item(belanja, "Minyak", 18000)        # jumlah default
1
tambah_item(belanja, "Gula", 14000, 3)

tampilkan_daftar(belanja)

# Hitung total dengan dan tanpa diskon
total_normal = hitung_total(belanja)
total_diskon = hitung_total(belanja, diskon=0.10)

print(f"\nTotal normal      : Rp {total_normal:,.0f}")
print(f"Total dengan 10% diskon : Rp {total_diskon:,.0f}")
print(f"Anda hemat        : Rp {total_normal -
total_diskon:,.0f}")

# Cek item kebutuhan dasar yang belum ada
yang_belum = [k for k in kebutuhan_dasar() if k.lower() not in
belanja]
if yang_belum:
    print(f"\nBelum ada: {', '.join(yang_belum)}")
```

Skrip ini menunjukkan: type hint pada parameter dan return, docstring dengan format Google style, parameter default (`jumlah: int = 1`), keyword-only parameter dengan `*,` sebelum diskon, dan komposisi fungsi (`hitung_total` memanggil `hitung_subtotal`).

11.5 Kesalahan Umum dan Cara Mengatasinya

Kesalahan 1: Mutable default argument.

Miskonsepsi: "Saya bikin def f(items=[]) dan tiap pemanggilan items-nya kebawa."

Realita: Default mutable di-evaluasi SEKALI saat definisi fungsi, lalu di-share di semua pemanggilan. Gunakan idiom: def f(items=None): if items is None: items

= []. Pola ini adalah salah satu hal pertama yang dikoreksi reviewer kode profesional.

Kesalahan 2: Lupa return statement.

Miskonsepsi: "Fungsi saya tidak mengembalikan nilai, kok dapat None?"

Realita: Fungsi tanpa return eksplisit selalu mengembalikan None. Pastikan ada baris 'return nilai' di akhir fungsi yang seharusnya mengembalikan sesuatu. Atau periksa: apakah return Anda ada di dalam blok if yang mungkin tidak tereksekusi?

Kesalahan 3: Tidak memahami pass-by-reference untuk objek mutable.

Miskonsepsi: "Saya kirim list ke fungsi, list-nya dimodifikasi, kok list di luar juga berubah?"

Realita: Python tidak pass-by-value atau pass-by-reference dalam arti tradisional, tetapi pass-by-object-reference. Untuk objek mutable (list, dict), fungsi dapat memodifikasi objek asli. Jika Anda ingin fungsi tidak memodifikasi, kirim salinan dengan list.copy() atau buat objek baru di dalam fungsi.

Kesalahan 4: Memberi nilai default ke parameter di tengah.

Miskonsepsi: "Saya bikin def f(a=5, b, c=10) dan dapat SyntaxError."

Realita: Parameter dengan default harus berada di akhir, setelah semua parameter tanpa default. Aturan ini menghindari ambiguitas saat memanggil dengan argumen posisional.

Kesalahan 5: Lambda terlalu kompleks.

Miskonsepsi: "Saya bikin lambda 4 baris dengan ternary bersarang dan tidak bisa dibaca."

Realita: Lambda dimaksudkan untuk fungsi sangat sederhana satu baris. Jika lambda Anda sulit dibaca atau melebihi 80 karakter, gantilah dengan fungsi biasa def. Keterbacaan selalu lebih penting daripada keringkasan.

11.6 Rangkuman

Bab 11 telah memperkenalkan fungsi sebagai unit modularisasi paling fundamental di Python. Fungsi didefinisikan dengan keyword def, dapat menerima parameter posisional dan keyword, mengembalikan nilai tunggal atau ganda, dan mendukung nilai default untuk parameter opsional. Prinsip DRY (Don't Repeat Yourself) menjadikan fungsi alat utama untuk menulis kode yang reusable, maintainable, testable, dan readable.

Mekanisme *args dan **kwargs memungkinkan fungsi menerima jumlah argumen variabel: *args mengumpulkan argumen posisional sebagai tuple, **kwargs mengumpulkan argumen keyword sebagai dictionary. Operator * dan ** juga dapat digunakan saat pemanggilan untuk membuka iterable atau dictionary. Fungsi lambda menyediakan cara ringkas untuk fungsi pendek, ideal sebagai argumen ke fungsi seperti sorted(), map(), dan filter().

Praktik profesional mencakup penggunaan docstring (PEP 257) untuk dokumentasi dan type hint (PEP 484) untuk kejelasan tipe. Mutable default argument adalah jebakan klasik yang harus dihindari dengan idiom None default. Refaktorisasi kode monolitik menjadi fungsi-fungsi kecil dengan tanggung jawab tunggal adalah keterampilan kunci yang membedakan pemula dari profesional.

11.7 Latihan Terbimbing

Soal 11.7.1

Soal: Tulis fungsi konverter_suhu(nilai, dari, ke) yang mengonversi suhu antar tiga satuan (C, F, K). Sertakan docstring dan type hint.

Pembahasan:

```
def konverter_suhu(nilai: float, dari: str, ke: str) -> float:
    """Konversi suhu antar satuan C, F, K.

    Args:
        nilai: nilai suhu.
        dari: satuan asal ('C', 'F', atau 'K').
        ke: satuan tujuan.
```

```
    Returns:
        nilai dalam satuan tujuan.
    """
    # Konversi semua ke Celsius dulu
    if dari == "F":
        c = (nilai - 32) * 5 / 9
    elif dari == "K":
        c = nilai - 273.15
    else:
        c = nilai

    # Lalu ke satuan tujuan
    if ke == "F":
        return c * 9 / 5 + 32
    elif ke == "K":
        return c + 273.15
    return c

print(konverter_suhu(100, "C", "F"))    # 212.0
print(konverter_suhu(32, "F", "C"))     # 0.0
```

Soal 11.7.2

Soal: Apa output kode berikut, dan mengapa? def tambah(item, list=[]): list.append(item); return list. print(tambah(1)); print(tambah(2)); print(tambah(3)).

Pembahasan: Output: [1] [1, 2] [1, 2, 3]. Ini adalah jebakan mutable default. List [] di-evaluasi sekali saat fungsi didefinisikan, lalu di-share di semua pemanggilan. Perbaikan: def tambah(item, list=None): if list is None: list = []; list.append(item); return list.

Soal 11.7.3

Soal: Tulis fungsi yang menerima jumlah parameter variabel berupa nilai mahasiswa, dan mengembalikan rata-rata serta deviasi standar.

Pembahasan:

```
def statistik(*nilai: float) -> tuple[float, float]:
    """Hitung rata-rata dan deviasi standar."""
```

```
    if not nilai:
        return 0.0, 0.0
    rata = sum(nilai) / len(nilai)
    variansi = sum((n - rata) ** 2 for n in nilai) /
len(nilai)
    deviasi = variansi ** 0.5
    return rata, deviasi

r, d = statistik(78, 85, 92, 67, 88)
print(f"Rata-rata: {r:.2f}, Deviasi: {d:.2f}")
```

Soal 11.7.4

Soal: Gunakan lambda dan sorted() untuk mengurutkan list of tuples [("Andi", 85), ("Budi", 72), ("Cici", 91)] berdasarkan: (a) nama abjad, (b) nilai descending, (c) panjang nama.

Pembahasan:

```
data = [("Andi", 85), ("Budi", 72), ("Cici", 91)]

# (a) nama abjad
a = sorted(data, key=lambda x: x[0])

# (b) nilai descending
b = sorted(data, key=lambda x: x[1], reverse=True)

# (c) panjang nama
c = sorted(data, key=lambda x: len(x[0]))

print(a)
print(b)
print(c)
```

Soal 11.7.5

Soal: Refaktorisasi: pecah kode berikut menjadi minimal 3 fungsi dengan tanggung jawab masing-masing. Kode asli menghitung diskon dan total pajak.

Pembahasan:

```
# Versi termodularisasi
def hitung_diskon(harga: float, persen: float) -> float:
    """Menghitung jumlah diskon dalam rupiah."""
    return harga * persen / 100

def hitung_pajak(harga: float, tarif: float = 11.0) -> float:
    """Menghitung PPN (default 11%)."""
    return harga * tarif / 100

def total_belanja(harga: float, diskon_persen: float = 0,
                  pajak_persen: float = 11.0) -> float:
    """Menghitung total akhir setelah diskon dan pajak."""
    diskon = hitung_diskon(harga, diskon_persen)
    setelah_diskon = harga - diskon
    pajak = hitung_pajak(setelah_diskon, pajak_persen)
    return setelah_diskon + pajak

# Pemakaian
print(total_belanja(100000, 10))          # diskon 10%, PPN
11%
print(total_belanja(100000, 0))           # tanpa diskon
print(total_belanja(100000, 20, 5))       # diskon 20%, PPN
5%
```

11.8 Latihan Mandiri

- **(C2)** Jelaskan prinsip DRY dan empat manfaat modularisasi dengan fungsi.
- **(C2)** Apa perbedaan parameter posisional, keyword, dan default? Berikan contoh untuk masing-masing.
- **(C3)** Tulis fungsi adalah_prima(n) yang mengembalikan True jika n bilangan prima, False jika tidak.
- **(C3)** Tulis fungsi balik_string(s) yang mengembalikan string terbalik TANPA menggunakan slicing [::-1].
- **(C3)** Buat fungsi statistik(*nilai, presisi=2) yang mengembalikan tuple (min, max, rata-rata) dengan presisi default 2 desimal.
- **(C3)** Tulis fungsi info_mahasiswa(**data) yang menerima keyword arguments tentang mahasiswa dan menampilkannya dalam format yang rapi.

- **(C4)** Analisis kode: def f(x, y=x): return x + y. Apakah ini valid? Mengapa? Jika tidak, bagaimana cara mendapatkan perilaku yang diinginkan?
- **(C4)** Refaktorisasi: ambil proyek mini Bab 8 (Sistem Pengelolaan Nilai) dan ubah menjadi struktur modular dengan minimal 5 fungsi terpisah.
- **(C4)** Studi kasus: tulis fungsi kirim_email() dengan parameter: penerima (wajib), subjek (wajib), isi (wajib), cc (default None), bcc (default None), html (default False), attachment (default None). Beri docstring dan type hint lengkap.
- **(C4)** Diskusikan: kapan lambda lebih cocok daripada def, dan sebaliknya? Berikan dua contoh untuk masing-masing kasus.

11.9 Tugas Proyek Mini

"Library Kalkulator Matematika Pribadi"

Bangun pustaka fungsi matematika yang dapat dipakai berulang. Spesifikasi:

- Buat berkas `math_utils.py` yang berisi minimal 8 fungsi terkait matematika dasar.
- Setiap fungsi WAJIB memiliki: docstring lengkap (deskripsi, Args, Returns, Contoh), type hint pada parameter dan return value, dan validasi input sederhana.
- Fungsi yang harus ada (minimal): faktorial(n), fpb(a, b), kpk(a, b), adalah_prima(n), fibonacci(n), kuadrat_terkecil(*angka), rata_rata_terbobot(nilai, bobot), konversi_suhu(nilai, dari, ke).
- Buat berkas terpisah `main.py` yang mengimpor pustaka tadi dan mendemonstrasikan setiap fungsi dengan minimal 2 kasus uji per fungsi.
- Sertakan satu fungsi yang menggunakan `*args`, satu yang menggunakan `**kwargs`, dan satu yang menggunakan lambda di dalamnya.

Format pengumpulan: berkas ZIP berisi math_utils.py dan main.py beserta tangkapan layar hasil eksekusi dan dokumentasi minimal 1 halaman.

11.10 Refleksi Diri

Centang pernyataan berikut yang sudah Anda kuasai:

☐ Saya memahami prinsip DRY dan empat manfaat modularisasi.

☐ Saya dapat mendefinisikan fungsi dengan return value tunggal maupun ganda.

☐ Saya dapat membedakan parameter posisional, keyword, dan default.

☐ Saya menyadari mutable default pitfall dan tahu idiom None default.

☐ Saya menguasai *args dan **kwargs untuk fungsi dengan parameter variabel.

☐ Saya dapat menulis lambda untuk kasus sederhana dan tahu kapan kembali ke def.

☐ Saya menulis docstring untuk setiap fungsi yang saya buat.

☐ Saya mulai menggunakan type hint untuk kode yang lebih jelas.

☐ Saya dapat melakukan refaktorisasi kode panjang menjadi fungsi-fungsi kecil.

☐ Saya menerapkan single responsibility principle pada fungsi-fungsi saya.

ASAH OTAK 11.2

Eksperimen: ambil salah satu proyek mini Anda dari Bab 1-10. Hitung berapa banyak baris kode (LoC) yang ada di program utama (di luar fungsi). Sekarang refaktorisasi menjadi minimal 4 fungsi. Hitung lagi LoC di program utama. Apakah berkurang? Apakah kode menjadi lebih jelas? Salah satu metrik kualitas kode adalah 'main function length'—semakin pendek semakin baik. Programmer profesional menargetkan main function hanya 5-10 baris.

BAB 12

MODULARISASI DAN SCOPE VARIABEL

"Namespaces are one honking great idea — let's do more of those!"
— The Zen of Python, aforisme penutup oleh Tim Peters

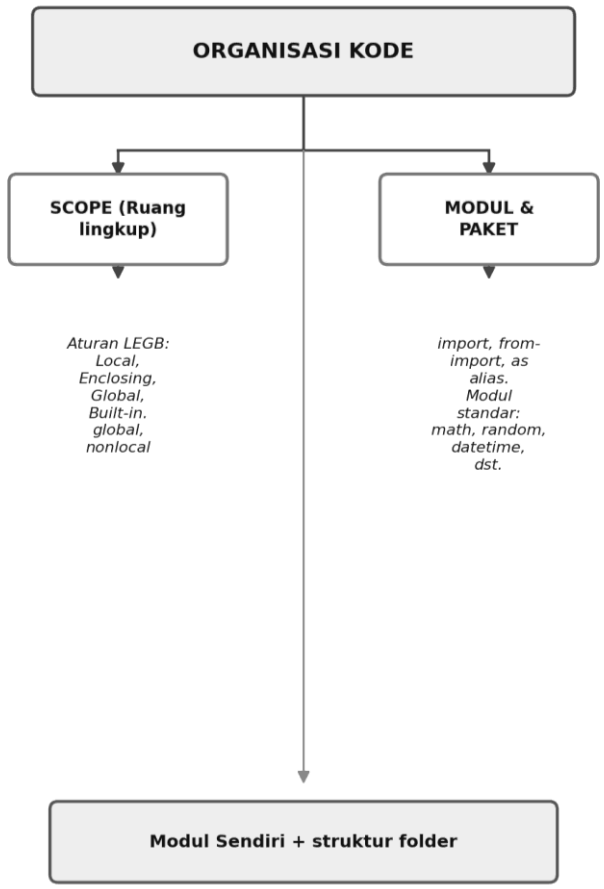
Capaian Pembelajaran Bab

Setelah mempelajari bab ini, mahasiswa diharapkan mampu mencapai sejumlah luaran pembelajaran sebagai berikut:

Kode	Capaian	Bloom
CPB-12.1	Menjelaskan aturan LEGB untuk pencarian variabel	C2
CPB-12.2	Membedakan variabel lokal dan global serta kapan menggunakannya	C3
CPB-12.3	Menerapkan keyword global dan nonlocal dengan tepat	C3
CPB-12.4	Menjelaskan konsep modul dan paket dalam Python	C2
CPB-12.5	Menggunakan pernyataan import dengan berbagai variasi sintaksnya	C3
CPB-12.6	Memanfaatkan modul-modul standar Python (math, random, datetime, statistics)	C3
CPB-12.7	Membuat modul sendiri dan menggunakan if __name__ == "__main__"	C3
CPB-12.8	Merancang struktur folder proyek Python yang terorganisir	C3

12.1 Peta Konsep

Peta konsep berikut menggambarkan dua pilar pengorganisasian kode: scope variabel dan modularisasi melalui modul.



12.2 Apersepsi

Di Bab 11 kita memperkenalkan fungsi sebagai unit modularisasi paling dasar. Tetapi ketika program tumbuh besar, satu berkas yang berisi puluhan fungsi pun menjadi sulit dikelola. Bayangkan Anda membangun aplikasi e-commerce: ada logika pembayaran, manajemen produk, autentikasi, pengiriman, laporan. Memuat semuanya dalam satu berkas berarti gulir tanpa akhir.

Modul adalah jawaban Python untuk modularisasi pada level yang lebih tinggi: bukan sekadar memecah kode menjadi fungsi, melainkan memecah aplikasi menjadi berkas-berkas terpisah, masing-masing dengan tanggung jawab tersendiri. Di bab ini Anda akan belajar bagaimana Python menemukan dan memuat modul, bagaimana membuat modul sendiri, dan bagaimana mengorganisasi proyek besar.

Sekaligus, kita akan mendalami konsep **scope variabel**: bagaimana Python menentukan variabel mana yang Anda maksud ketika ada nama yang sama di beberapa tempat. Pemahaman scope adalah fondasi yang membedakan programmer pemula dari yang mahir, dan banyak bug halus berakar pada kesalahpahaman scope. Setelah bab ini, Anda akan memiliki kosakata teknis untuk mendiskusikan kode dengan presisi profesional.

> **ASAH OTAK 12.1**
>
> Jika Anda menulis x = 10 di awal program, lalu di dalam sebuah fungsi menulis x = 20, mengapa kode di luar fungsi tetap melihat x = 10? Bagaimana Python tahu "x mana" yang Anda maksud di setiap tempat? Renungkan pertanyaan ini sebelum membaca bagian Aturan LEGB.

12.3 Uraian Materi

12.3.1 Aturan LEGB: Pencarian Variabel

Ketika Python menemui nama variabel dalam kode, ia mencari nilainya berdasarkan empat tingkatan scope, dengan urutan: **L→E→G→B**. Begitu nama ditemukan di satu tingkat, pencarian berhenti.

Huruf	Tingkat	Penjelasan
L	**Local**	Variabel yang didefinisikan di dalam fungsi saat ini. Diperiksa pertama.
E	**Enclosing**	Variabel di fungsi pembungkus (untuk nested functions). Diperiksa jika tidak ada di Local.
G	**Global**	Variabel di level modul (di luar semua fungsi). Diperiksa jika tidak ada di L atau E.

Huruf	Tingkat	Penjelasan
B	**Built-in**	Nama bawaan Python: print, len, range, dll. Diperiksa terakhir.

```
# Contoh keempat scope
x = "global"                # Global scope

def luar():
    x = "enclosing"         # Enclosing scope (untuk fungsi
dalam)

    def dalam():
        x = "local"         # Local scope
        print(x)            # mencetak 'local'

    dalam()
    print(x)                # mencetak 'enclosing'

luar()
print(x)                    # mencetak 'global'
print(len)                  # built-in: <built-in function len>
```

CATATAN 12.1

Akronim LEGB merupakan mnemonic yang populer di komunitas Python untuk mengingat empat tingkat scope, dan banyak dijelaskan dalam literatur Python populer seperti Learning Python karya Mark Lutz. Aturan ini tidak hanya teoretis—ia menjelaskan banyak "magic" di Python, seperti mengapa fungsi bawaan seperti print() selalu tersedia (Built-in selalu ada di akhir pencarian).

12.3.2 Variabel Lokal vs Global

Secara default, variabel yang ditugaskan di dalam fungsi adalah *lokal*. Mereka lahir saat fungsi dipanggil dan mati saat fungsi selesai. Variabel yang ditugaskan di level modul (di luar fungsi apa pun) adalah *global*, hidup selama program berjalan.

```
# Variabel global
pi = 3.14159
nama_program = "KalkulatorLuas"

def luas_lingkaran(r):
    # 'r' adalah lokal, 'pi' adalah global (read-only di sini)
    return pi * r ** 2

def main():
    # 'hasil' adalah lokal
    hasil = luas_lingkaran(5)
    print(f"{nama_program}: {hasil}")

main()
# print(hasil)              # ERROR: hasil tidak ada di luar
main()
```

PERHATIAN 12.1

Penggunaan variabel global yang berlebihan adalah praktik buruk. Variabel global membuat fungsi sulit diuji (perilaku bergantung pada keadaan luar) dan rentan bug karena dapat dimodifikasi dari mana saja. Aturan praktis: gunakan global hanya untuk konstanta (SCREAMING_SNAKE_CASE) atau konfigurasi yang dibaca-saja. Untuk data yang berubah, lewatkan sebagai parameter dan return nilai baru.

12.3.3 Keyword global dan nonlocal

Jika Anda ingin memodifikasi variabel di scope yang lebih luar dari dalam fungsi, Anda harus menyatakannya dengan keyword khusus. Jika tidak, Python akan menganggap penugasan sebagai variabel lokal baru.

Keyword global

```
counter = 0                     # global

def tambah():
    global counter              # "counter di sini adalah yang
global"
    counter += 1
```

```
tambah()
tambah()
print(counter)                  # 2

# TANPA 'global', kode berikut akan ERROR:
def tambah_salah():
    counter += 1                # UnboundLocalError!
    # Python kira counter adalah variabel lokal yang belum
diinisialisasi
```

Keyword nonlocal

`nonlocal` digunakan dalam nested function untuk merujuk ke variabel di scope enclosing (bukan global).

```
def luar():
    pesan = "halo"

    def dalam():
        nonlocal pesan          # rujuk ke 'pesan' di luar()
        pesan = "hai"

    print(f"Sebelum: {pesan}")  # 'halo'
    dalam()
    print(f"Sesudah: {pesan}")  # 'hai'

luar()
```

TIPS PRO 12.1

Banyak programmer Python berpengalaman sangat hemat menggunakan global dan nonlocal. Sebagai aturan praktis, jika Anda mendapati diri ingin menggunakan keduanya, tanyakan dulu: apakah ada cara yang lebih bersih dengan parameter dan return value? Biasanya jawabannya YA. State mutation di scope luar membuat aliran data dalam program tidak jelas—sumber bug yang paling sukar dideteksi.

12.3.4 Modul dan Paket

Modul adalah berkas Python (`.py`) yang berisi definisi-definisi (fungsi, kelas, variabel) yang dapat digunakan oleh berkas Python lain. **Paket** adalah kumpulan modul yang diorganisir dalam folder.

Tiga sumber modul di Python:

- **Modul standar (built-in).** Sudah tersedia tanpa instalasi tambahan: math, random, datetime, os, sys, json, dll. Ratusan jumlahnya.
- **Pustaka pihak ketiga (third-party).** Diinstal dengan pip: requests, numpy, pandas, flask, dll. Ratusan ribu jumlahnya di PyPI.
- **Modul buatan sendiri.** Berkas .py yang Anda buat dan letakkan di folder proyek Anda.

12.3.5 Pernyataan import dan Variasinya

Python menyediakan beberapa cara mengimpor modul:

```
# 1. Impor seluruh modul
import math
math.sqrt(16)                       # 4.0
math.pi                             # 3.14159...

# 2. Impor dengan alias (untuk modul dengan nama panjang)
import math as m
m.sqrt(25)                          # 5.0

# 3. Impor anggota spesifik
from math import sqrt, pi
sqrt(16)                            # 4.0 (tanpa prefix math.)
pi                                  # langsung dipakai

# 4. Impor anggota dengan alias
from math import sqrt as akar
akar(36)                            # 6.0

# 5. Impor SEMUA (HINDARI di kode produksi)
from math import *
sqrt(49)                            # 7.0
# Bahaya: nama modul mengotori namespace Anda
```

PERHATIAN 12.2

Hindari from modul import * di kode serius. Pola ini mengimpor SEMUA nama dari modul ke namespace Anda, sehingga pembaca kode tidak tahu dari mana fungsi-fungsi berasal. Selain itu, dapat menimpa variabel yang sudah ada. Pengecualian: di REPL untuk eksperimen cepat, atau pada __init__.py paket. Selebihnya: impor secara eksplisit.

12.3.6 Modul Standar Penting

Modul math

```
import math

math.pi                                 # 3.141592653589793
math.e                                  # 2.718281828459045
math.sqrt(2)                            # 1.4142135623730951
math.pow(2, 10)                         # 1024.0 (sama dengan 2 **
10)
math.floor(3.7)                         # 3 (pembulatan ke bawah)
math.ceil(3.2)                          # 4 (pembulatan ke atas)
math.factorial(5)                       # 120
math.log(100, 10)                       # 2.0 (log basis 10)
math.sin(math.pi / 2)                   # 1.0 (radian)
math.degrees(math.pi)                   # 180.0
math.gcd(12, 18)                        # 6 (FPB)
```

Modul random

```
import random

random.random()                         # 0.0 - 1.0
random.uniform(1.0, 10.0)               # float di rentang
tertentu
random.randint(1, 100)                  # integer 1-100 inklusif
random.randrange(0, 10, 2)              # bilangan genap 0-8

buah = ["apel", "jeruk", "mangga"]
random.choice(buah)                     # 1 elemen acak
```

```
random.sample(buah, 2)              # 2 elemen acak, tanpa
duplikat
random.shuffle(buah)                # acak in-place

# Reproducible random untuk testing
random.seed(42)                     # bibit tetap
```

Modul datetime

```
from datetime import datetime, date, timedelta

sekarang = datetime.now()
print(sekarang)                     # 2026-05-16
15:30:45.123456

hari_ini = date.today()             # 2026-05-16
ultah = date(2000, 5, 15)
umur_hari = (hari_ini - ultah).days

# Operasi waktu
besok = sekarang + timedelta(days=1)
minggu_depan = sekarang + timedelta(weeks=1)

# Format string
sekarang.strftime("%Y-%m-%d %H:%M") # '2026-05-16 15:30'
sekarang.strftime("%d %B %Y")       # '16 May 2026'
```

Modul statistics

```
import statistics as st

data = [78, 85, 92, 67, 88, 75, 81]

st.mean(data)                       # 80.857... (rata-rata)
st.median(data)                     # 81 (median)
st.mode([1, 1, 2, 3, 3, 3, 4])      # 3 (modus)
st.stdev(data)                      # 8.40... (deviasi
standar)
st.variance(data)                   # 70.81... (variansi)
```

WAWASAN INDUSTRI 12.1

Filosofi Python adalah "batteries included"—pustaka standar yang sangat lengkap, mencakup lebih dari 200 modul untuk hampir setiap kebutuhan: matematika, regex, jaringan, parsing JSON/XML/CSV, kompresi, kriptografi, GUI, dan banyak lagi. Sebelum mencari pustaka pihak ketiga, selalu cek dulu apakah ada modul standar yang dapat menyelesaikan masalah Anda. Dokumentasi resmi di docs.python.org adalah salah satu sumber paling berharga yang harus Anda bookmark.

12.3.7 Membuat Modul Sendiri

Setiap berkas `.py` adalah modul. Untuk mengimpor modul sendiri, letakkan berkas tersebut di folder yang sama dengan program utama Anda.

Contoh: Membuat Modul math_utils.py

```
# === berkas: math_utils.py ===

"""Modul utilitas matematika dasar."""

PI = 3.14159265358979

def luas_lingkaran(r: float) -> float:
    """Luas lingkaran berdasarkan jari-jari."""
    return PI * r ** 2

def luas_persegi(s: float) -> float:
    """Luas persegi berdasarkan sisi."""
    return s ** 2

def factorial(n: int) -> int:
    """Faktorial bilangan bulat non-negatif."""
    if n < 0:
        raise ValueError("Faktorial tak terdefinisi untuk
negatif")
    if n == 0:
        return 1
    return n * factorial(n - 1)
```

```
# === berkas: main.py (di folder yang sama) ===

import math_utils

print(math_utils.luas_lingkaran(5))
print(math_utils.PI)

# Atau impor anggota spesifik
from math_utils import luas_persegi, factorial
print(luas_persegi(4))
print(factorial(5))
```

Pola if __name__ == "__main__"

Setiap modul memiliki variabel khusus `__name__`. Jika modul dijalankan langsung, `__name__` bernilai **"__main__"**. Jika diimpor, `__name__` bernilai nama modul. Ini memungkinkan kode yang hanya dijalankan saat berkas dijalankan langsung, bukan saat diimpor.

```
# === math_utils.py ===

def luas_lingkaran(r):
    return 3.14159 * r ** 2

# Kode di bawah hanya dijalankan jika file ini dieksekusi
langsung
if __name__ == "__main__":
    # Tes/demo modul
    print("Demo math_utils:")
    print(f"Luas r=5: {luas_lingkaran(5)}")
    print(f"Luas r=10: {luas_lingkaran(10)}")

# Jika dijalankan dengan: python math_utils.py
# → mencetak demo

# Jika diimpor dari main.py:
# import math_utils
# → demo TIDAK dicetak, fungsinya tetap dapat dipakai
```

TIPS PRO 12.2

Pola if __name__ == "__main__" adalah idiom Python yang harus Anda kuasai. Manfaatnya: (1) Modul dapat berfungsi sebagai pustaka (saat diimpor) maupun sebagai program (saat dijalankan langsung). (2) Demo dan tes inline tidak mengganggu impor. (3) Memudahkan unit testing dengan pytest. Letakkan blok ini di akhir berkas modul Anda.

12.3.8 Struktur Proyek Python Sederhana

Saat proyek tumbuh, organisasi folder menjadi penting. Berikut struktur yang umum dipakai untuk proyek pemula menengah:

```
proyek_saya/
│
├── README.md                    # dokumentasi proyek
├── requirements.txt             # daftar pustaka pihak
ketiga
├── .gitignore                   # berkas yang diabaikan git
│
├── main.py                      # titik masuk program
│
├── utils/                       # paket utility
│   ├── __init__.py              # menandai folder sbg paket
│   ├── math_utils.py
│   ├── string_utils.py
│   └── date_utils.py
│
├── data/                        # data dan konfigurasi
│   ├── config.json
│   └── mahasiswa.csv
│
└── tests/                       # unit tests
    ├── __init__.py
    └── test_math_utils.py
```

Berkas `__init__.py` menandai sebuah folder sebagai *paket Python*, sehingga modul-modul di dalamnya dapat diimpor. Sejak Python 3.3, file ini dapat kosong dan tidak selalu wajib (namespace packages), tetapi konvensinya tetap menyertakannya.

```
# Mengimpor dari paket
from utils.math_utils import luas_lingkaran
from utils.string_utils import balik_string

# Atau impor seluruh paket
import utils.math_utils as mu
mu.luas_lingkaran(5)
```

> **DI BALIK LAYAR 12.1**
>
> Saat Anda mengimpor modul, Python mencarinya dalam urutan: (1) modul yang sudah pernah diimpor (cache di sys.modules), (2) modul built-in di interpreter, (3) berkas di sys.path. sys.path adalah list folder yang dicari, mulai dari folder berkas yang dijalankan, lalu folder dalam variabel environment PYTHONPATH, lalu folder instalasi Python. Cetak sys.path di REPL Anda untuk melihat—pengetahuan ini akan menyelamatkan Anda dari banyak masalah impor di masa depan.

12.4 Contoh Kode Lengkap

Berikut adalah proyek kecil yang mendemonstrasikan modularisasi penuh: pembagian kode menjadi beberapa modul, penggunaan modul standar, dan pola `if __name__ == "__main__"`.

Berkas 1: utils/finance.py

```
"""Modul utilitas finansial."""

PPN = 0.11

def hitung_diskon(harga: float, persen: float) -> float:
    """Hitung nominal diskon."""
    return harga * persen / 100

def hitung_pajak(harga: float, tarif: float = PPN) -> float:
    """Hitung pajak dari harga (default PPN 11%)."""
```

```
    return harga * tarif

def total_akhir(harga: float, diskon_persen: float = 0) ->
float:
    """Total setelah diskon dan pajak."""
    diskon = hitung_diskon(harga, diskon_persen)
    setelah_diskon = harga - diskon
    pajak = hitung_pajak(setelah_diskon)
    return setelah_diskon + pajak

if __name__ == "__main__":
    print(f"Total 100k tanpa diskon:
{total_akhir(100000):,.0f}")
    print(f"Total 100k diskon 20%:   {total_akhir(100000,
20):,.0f}")
```

Berkas 2: utils/laporan.py

```
"""Modul pembuat laporan dengan tanggal."""

from datetime import datetime

def cetak_header(judul: str, lebar: int = 50) -> None:
    """Cetak header laporan dengan tanggal."""
    print("=" * lebar)
    print(f"  {judul.upper()}")
    print(f"  Dicetak: {datetime.now().strftime('%d %B %Y,
%H:%M')}")
    print("=" * lebar)

def cetak_baris(label: str, nilai: float, lebar: int = 50) ->
None:
    """Cetak baris label-nilai yang rata."""
    print(f"  {label:<25}{'Rp ' + format(nilai, ',.0f'):>20}")

def cetak_footer(lebar: int = 50) -> None:
    """Cetak footer laporan."""
    print("=" * lebar)
```

Berkas 3: main.py

```
"""Program utama: laporan transaksi sederhana."""

from utils.finance import hitung_diskon, hitung_pajak,
total_akhir
from utils.laporan import cetak_header, cetak_baris,
cetak_footer
import random

def buat_transaksi_acak(n: int = 5) -> list:
    """Buat n transaksi acak untuk demo."""
    items = ["Buku", "Pulpen", "Tas", "Sepatu", "Topi", "Jam"]
    return [
        (random.choice(items), random.randint(50, 500) * 1000)
        for _ in range(n)
    ]

def main():
    random.seed(42)              # untuk hasil yang konsisten
    transaksi = buat_transaksi_acak(5)

    cetak_header("Laporan Transaksi")

    subtotal = 0
    for nama, harga in transaksi:
        cetak_baris(nama, harga)
        subtotal += harga

    cetak_baris("-" * 25, 0)     # garis pemisah
    cetak_baris("Subtotal", subtotal)
    cetak_baris("Diskon (10%)", -hitung_diskon(subtotal, 10))
    cetak_baris("PPN", hitung_pajak(subtotal * 0.9))
    cetak_baris("TOTAL", total_akhir(subtotal, 10))
    cetak_footer()

if __name__ == "__main__":
    main()
```

Proyek ini menunjukkan: pemisahan tanggung jawab (finance vs laporan), pembuatan paket dengan folder utils/, impor selektif dengan from-import, penggunaan modul standar (datetime, random), serta pola main() yang

dilindungi `if __name__ == “__main__”`. Inilah anatomi proyek Python profesional dalam skala kecil.

12.5 Kesalahan Umum dan Cara Mengatasinya

Kesalahan 1: UnboundLocalError saat memodifikasi global.

Miskonsepsi: “Saya akses counter di dalam fungsi dan dapat UnboundLocalError.”

Realita: Jika Python melihat penugasan ke variabel di dalam fungsi, ia menganggap variabel itu LOKAL untuk seluruh fungsi, bahkan sebelum penugasan. Solusi: deklarasikan global counter di awal fungsi sebelum dipakai.

Kesalahan 2: Lingkaran impor (circular import).

Miskonsepsi: “Saya bikin a.py impor b, b.py impor a, lalu dapat ImportError.”

Realita: Saat dua modul saling impor, Python bingung mana yang harus dimuat dulu. Solusi: pikirkan ulang desain. Biasanya artinya ada fungsi yang seharusnya di modul ketiga (utilitas bersama), atau Anda perlu impor di dalam fungsi (bukan di top-level).

Kesalahan 3: ModuleNotFoundError padahal modul ada.

Miskonsepsi: “Modul saya ada di folder, kenapa Python tidak menemukannya?”

Realita: Python mencari modul di sys.path. Folder berkas yang dijalankan otomatis ditambahkan, tetapi jika Anda menjalankan dari folder lain, modul tidak ditemukan. Pastikan menjalankan dari folder yang benar, atau gunakan struktur paket dengan __init__.py.

Kesalahan 4: Menggunakan from modul import *.

Miskonsepsi: “Saya pakai from math import * supaya tidak perlu tulis math. terus.”

Realita: Ini mengotori namespace Anda dan menyembunyikan asal-usul fungsi. Sebaiknya import math (lebih jelas) atau from math import sqrt, pi (impor spesifik). Bayangkan pembaca kode Anda yang melihat sqrt(16) tanpa tahu dari mana sqrt berasal.

Kesalahan 5: Lupa pola if __name__ == "__main__".

Miskonsepsi: "Kenapa demo modul saya berjalan saat saya import dari main.py?"

Realita: Tanpa if __name__ == "__main__", semua kode di level modul akan dieksekusi saat diimpor, termasuk demo. Pisahkan: definisi fungsi/variabel di top-level (akan diimpor), demo di dalam if __name__ block (hanya saat dijalankan langsung).

12.6 Rangkuman

Bab 12 telah membahas dua aspek penting pengorganisasian kode di Python. Aturan LEGB (Local, Enclosing, Global, Built-in) menjelaskan bagaimana Python menemukan variabel: dimulai dari scope terdekat dan meluas keluar. Variabel yang ditugaskan di dalam fungsi adalah lokal secara default; untuk memodifikasi variabel di scope lebih luar, gunakan keyword global atau nonlocal—meskipun keduanya sebaiknya dihindari demi kebersihan kode.

Modul adalah berkas Python yang dapat diimpor oleh berkas lain. Python menyediakan ratusan modul standar (math, random, datetime, statistics, dan banyak lagi) yang menyelesaikan kebutuhan umum tanpa perlu instalasi tambahan. Pernyataan import memiliki beberapa variasi: import biasa, import as alias, from-import selektif, dan from-import-* (yang sebaiknya dihindari). Anda dapat membuat modul sendiri dengan membuat berkas .py dan mengimpornya, dan paket dengan mengorganisasi modul-modul dalam folder dengan __init__.py.

Pola `if __name__ == "__main__"` memungkinkan modul berfungsi sebagai pustaka maupun program. Struktur folder yang baik (utils/, data/, tests/) memudahkan pemeliharaan proyek besar. Penguasaan modul dan scope adalah pembeda penting antara programmer pemula dan profesional, dan menjadi fondasi untuk masuk ke topik-topik lanjutan seperti pemrograman berorientasi objek di Algoritma dan Pemrograman 2.

12.7 Latihan Terbimbing

Soal 12.7.1

Soal: Apa output kode berikut, dan mengapa? x = 10; def f(): print(x); x = 20; print(x). f(); print(x).

Pembahasan: Output: UnboundLocalError. Penjelasan: karena ada x = 20 di dalam f(), Python menganggap x sebagai variabel lokal di SELURUH fungsi f, termasuk pada baris print(x) pertama. Saat baris itu dieksekusi, x belum ditugaskan, sehingga error. Perbaikan: tambahkan global x di awal fungsi, atau gunakan nama berbeda untuk variabel lokal.

Soal 12.7.2

Soal: Tulis program yang mensimulasikan pelemparan dadu N kali dan menampilkan frekuensi setiap angka, menggunakan modul random dan collections.

Pembahasan:

```
import random

n = int(input("Berapa kali lempar dadu? "))

# Lempar dadu n kali
hasil = [random.randint(1, 6) for _ in range(n)]

# Hitung frekuensi
frekuensi = {}
for nilai in hasil:
    frekuensi[nilai] = frekuensi.get(nilai, 0) + 1

# Tampilkan dengan format grafik batang sederhana
for angka in range(1, 7):
    bar = "" * frekuensi.get(angka, 0)
    print(f"{angka}: {bar} ({frekuensi.get(angka, 0)})")
```

Soal 12.7.3

Soal: Buatlah modul date_utils.py yang berisi 3 fungsi: hari_dalam_minggu(tanggal), umur_dalam_hari(tanggal_lahir), dan tambah_hari(tanggal, n). Gunakan datetime.

Pembahasan:

```
# === date_utils.py ===
from datetime import date, timedelta

HARI = ["Senin", "Selasa", "Rabu", "Kamis",
        "Jumat", "Sabtu", "Minggu"]

def hari_dalam_minggu(tahun: int, bulan: int, hari: int) -> str:
    """Mengembalikan nama hari dalam minggu."""
    return HARI[date(tahun, bulan, hari).weekday()]

def umur_dalam_hari(tahun: int, bulan: int, hari: int) -> int:
    """Mengembalikan umur dalam hari dari tanggal lahir hingga
hari ini."""
    lahir = date(tahun, bulan, hari)
    return (date.today() - lahir).days

def tambah_hari(tahun: int, bulan: int, hari: int, n: int) ->
date:
    """Menambahkan n hari ke tanggal yang diberikan."""
    return date(tahun, bulan, hari) + timedelta(days=n)

if __name__ == "__main__":
    print(hari_dalam_minggu(2026, 5, 16))    # 'Sabtu'
    print(umur_dalam_hari(2005, 1, 1))
    print(tambah_hari(2026, 5, 16, 100))
```

Soal 12.7.4

Soal: Jelaskan perbedaan import math, from math import sqrt, dan from math import *. Berikan pro dan kontra masing-masing.

Pembahasan: 1. import math: mengimpor seluruh modul, akses dengan prefix math.sqrt(). Pro: jelas asal fungsi. Kontra: tulisan lebih panjang. 2. from math

import sqrt: hanya mengimpor sqrt, dipanggil sqrt() saja. Pro: lebih ringkas, eksplisit memilih yang dibutuhkan. Kontra: kehilangan informasi asal. 3. from math import *: mengimpor SEMUA. Pro: lebih ringkas. Kontra: mengotori namespace, dapat menimpa variabel lokal, asal fungsi tidak jelas. Untuk kode produksi, gunakan opsi 1 atau 2; hindari opsi 3.

Soal 12.7.5

Soal: Tulis modul calculator.py dengan 4 fungsi (tambah, kurang, kali, bagi) dan pola if __name__ == "__main__" yang menjalankan demo masing-masing fungsi.

Pembahasan:

```
# === calculator.py ===
"""Modul kalkulator dasar."""

def tambah(a: float, b: float) -> float:
    return a + b

def kurang(a: float, b: float) -> float:
    return a - b

def kali(a: float, b: float) -> float:
    return a * b

def bagi(a: float, b: float) -> float:
    if b == 0:
        raise ZeroDivisionError("Tidak boleh dibagi nol")
    return a / b

if __name__ == "__main__":
    print("Demo calculator:")
    print(f"5 + 3 = {tambah(5, 3)}")
    print(f"5 - 3 = {kurang(5, 3)}")
    print(f"5 * 3 = {kali(5, 3)}")
    print(f"5 / 3 = {bagi(5, 3):.2f}")
```

12.8 Latihan Mandiri

- **(C2)** Jelaskan aturan LEGB dan berikan contoh kasus di mana setiap level berperan.

- **(C2)** Apa perbedaan keyword global dan nonlocal? Berikan contoh kasus untuk masing-masing.
- **(C3)** Tulis program yang menggunakan modul random untuk membuat kuis matematika sederhana: 5 soal penjumlahan dengan angka acak 1-100.
- **(C3)** Gunakan modul datetime untuk membuat program yang menghitung umur seseorang dalam tahun, bulan, dan hari, berdasarkan tanggal lahir yang dimasukkan.
- **(C3)** Buat modul string_utils.py dengan 5 fungsi: balik_string, hitung_vokal, kapitalisasi_judul, hitung_kata, ringkas_teks (potong di 100 karakter).
- **(C3)** Tulis program yang membaca daftar nilai dari pengguna, lalu menggunakan modul statistics untuk menghitung mean, median, mode, dan standard deviation.
- **(C4)** Analisis kode: counter = 0; def inc(): counter = counter + 1; inc(). Mengapa ini error? Tulis dua perbaikan yang berbeda.
- **(C4)** Refaktorisasi proyek mini Bab 11 menjadi struktur multi-file dengan minimal: main.py, utils/calculator.py, dan utils/__init__.py.
- **(C4)** Diskusikan: kapan menggunakan global variabel masih diperbolehkan? Berikan tiga skenario yang valid dan jelaskan mengapa.
- **(C4)** Studi kasus: Anda diminta merancang struktur folder untuk aplikasi manajemen perpustakaan kecil dengan fitur: pencatatan buku, peminjaman, pengembalian, laporan. Gambar struktur folder lengkap dengan nama-nama file.

12.9 Tugas Proyek Mini

"Sistem Manajemen Tugas Multi-Modul"

Bangun aplikasi manajemen tugas (to-do list) dengan struktur folder profesional. Persyaratan:

- Struktur folder minimum: `todo_app/` dengan `main.py`, `models/`, `utils/`, dan `tests/`.
- Modul `models/task.py`: berisi fungsi-fungsi terkait pembuatan dan manipulasi tugas (representasi dict).
- Modul `utils/storage.py`: fungsi untuk menyimpan dan memuat tugas (bisa di-mock dulu dengan list global atau JSON dummy).
- Modul `utils/display.py`: fungsi untuk menampilkan tugas dalam berbagai format (list, tabel, ringkas).

- `main.py` menggunakan menu interaktif: tambah tugas, tampilkan semua, tandai selesai, hapus, filter (selesai/belum), keluar.
- Setiap modul WAJIB memiliki docstring di awal, dan pola if __name__ == "__main__" di akhir yang berisi demo modul itu sendiri.
- Gunakan minimal 2 modul standar Python (misalnya datetime untuk timestamp dan random untuk ID, atau yang lainnya).

Format pengumpulan: berkas ZIP berisi seluruh struktur folder, tangkapan layar pengujian 3 skenario, dan dokumen PDF berisi diagram struktur folder serta penjelasan tanggung jawab tiap modul.

12.10 Refleksi Diri

Centang pernyataan berikut yang sudah Anda kuasai:

☐ Saya memahami aturan LEGB untuk pencarian variabel.

☐ Saya dapat membedakan variabel lokal dan global di Python.

☐ Saya tahu kapan menggunakan global dan nonlocal, dan kapan menghindarinya.

☐ Saya memahami perbedaan modul, paket, dan pustaka.

☐ Saya menguasai berbagai variasi pernyataan import.

☐ Saya tahu cara menggunakan modul standar (math, random, datetime, statistics).

☐ Saya dapat membuat modul sendiri dan mengimpornya dari berkas lain.

☐ Saya menerapkan pola if __name__ == "__main__" dalam modul saya.

☐ Saya dapat merancang struktur folder proyek Python yang terorganisir.

☐ Saya tahu mengapa from modul import * sebaiknya dihindari.

ASAH OTAK 12.2

Eksperimen mandiri: jelajahi sys.path di REPL Anda dengan import sys; print(sys.path). Daftarkan semua folder yang muncul, lalu coba telusuri—folder apa saja itu? Mana yang berisi modul standar Python? Mana yang berisi pustaka pihak ketiga yang Anda instal? Mana yang merupakan

folder kerja Anda? Pemahaman ini akan membuat Anda tidak pernah lagi bingung tentang ModuleNotFoundError di masa depan.

BAB 13

FILE I/O DAN EXCEPTION HANDLING

"Errors should never pass silently. Unless explicitly silenced."
— The Zen of Python, prinsip ke-11 oleh Tim Peters

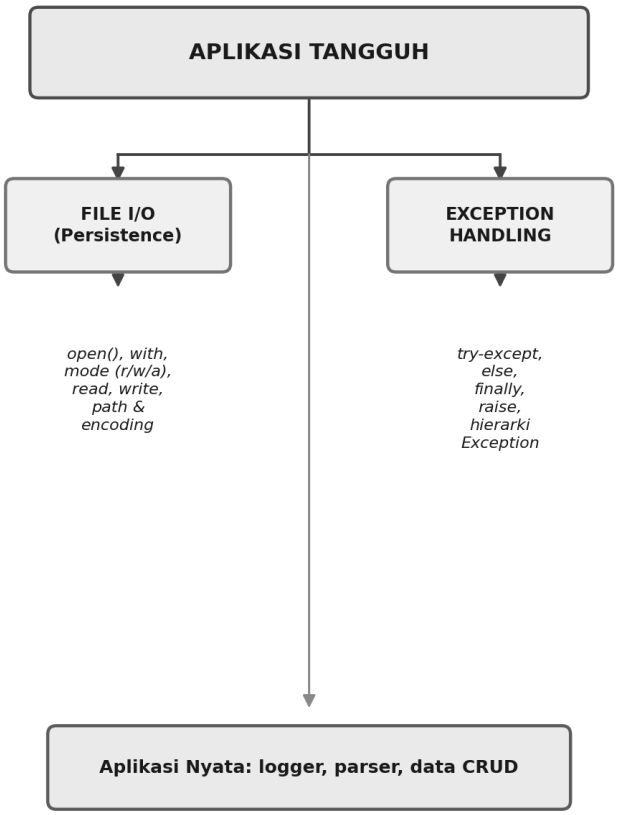
Capaian Pembelajaran Bab

Setelah mempelajari bab ini, mahasiswa diharapkan mampu mencapai sejumlah luaran pembelajaran sebagai berikut:

Kode	Capaian	Bloom
CPB-13.1	Menjelaskan konsep berkas pada sistem operasi (path, encoding, mode)	C2
CPB-13.2	Menggunakan fungsi open() untuk membuka berkas dengan mode yang tepat	C3
CPB-13.3	Menerapkan pernyataan with sebagai context manager	C3
CPB-13.4	Membaca berkas teks dengan berbagai metode (read, readline, readlines, iterasi)	C3
CPB-13.5	Menulis dan menambahkan data ke berkas teks	C3
CPB-13.6	Menerapkan try-except untuk menangani eksepsi	C3
CPB-13.7	Menggunakan klausa else, finally, dan pernyataan raise	C3
CPB-13.8	Mengintegrasikan file I/O dan exception handling dalam aplikasi nyata	C4

13.1 Peta Konsep

Peta konsep berikut menggambarkan keterkaitan antara file I/O dan exception handling sebagai dua keterampilan untuk membangun aplikasi yang tangguh.



13.2 Apersepsi

Sejauh ini, semua program yang Anda tulis bersifat *ephemeral*: data hidup hanya selama program berjalan, dan menghilang sesaat setelah program berakhir. Mahasiswa yang Anda daftarkan di proyek mini Bab 8 lenyap saat program ditutup. Daftar belanja di Bab 11 hilang saat Python interpreter ditutup. Program-program ini, meskipun fungsional, tidak benar-benar *berguna* di dunia nyata.

Aplikasi nyata membutuhkan **persistence**—kemampuan menyimpan data sehingga tersedia kembali setelah program dihidupkan ulang, bahkan setelah

komputer dimatikan. Cara paling sederhana untuk persistence adalah dengan menulis data ke berkas, dan membacanya kembali saat dibutuhkan. Inilah yang akan kita pelajari di paruh pertama bab ini.

Paruh kedua membahas **exception handling**, kemampuan program untuk merespons dengan anggun ketika sesuatu yang tidak diharapkan terjadi: berkas tidak ada, input pengguna salah format, koneksi internet putus, atau pembagian dengan nol. Program tanpa exception handling akan *crash* saat menemui kondisi tak terduga; program dengan exception handling dapat memberikan pesan kesalahan yang jelas atau melakukan tindakan pemulihan. Inilah pembeda penting antara prototype dan aplikasi production.

ASAH OTAK 13.1

Bayangkan aplikasi WhatsApp tiba-tiba tertutup karena pesan terakhir Anda berisi karakter aneh yang tidak diharapkan oleh programmer. Itulah yang terjadi pada program tanpa exception handling. Sebutkan tiga jenis input "yang tidak diharapkan" yang dapat menyebabkan program crash, dan pikirkan bagaimana exception handling dapat menyelamatkannya.

13.3 Uraian Materi

13.3.1 Konsep Berkas pada Sistem Operasi

Sebuah **berkas (file)** adalah urutan byte yang disimpan secara permanen pada media penyimpanan (hard disk, SSD, USB). Sistem operasi menyediakan abstraksi yang memungkinkan program membaca dan menulis berkas tanpa harus tahu detail teknis penyimpanan fisik.

Path: Alamat Berkas

```
# Absolute path (lengkap dari root):
# Windows: C:\Users\Yusri\Documents\data.txt
# Linux/Mac: /home/yusri/documents/data.txt

# Relative path (relatif terhadap folder kerja):
```

```
# data.txt           (di folder yang sama dengan skrip)
# data/nilai.txt     (di subfolder data/)
# ../config.json     (satu level di atas)
```

Encoding: Cara Karakter Direpresentasikan

Komputer menyimpan teks sebagai byte. *Encoding* menentukan bagaimana karakter dipetakan ke byte. Standar modern adalah **UTF-8**, yang dapat merepresentasikan semua karakter Unicode (termasuk huruf Indonesia, emoji, aksara Arab, dll).

```
# Contoh perbedaan encoding
# Karakter 'é' (e dengan aksen):
# UTF-8       : 2 byte (0xC3 0xA9)
# Windows-1252: 1 byte (0xE9)
# ASCII       : TIDAK BISA (di luar 128 karakter dasar)

# Best practice: selalu eksplisit menentukan encoding
open("data.txt", "r", encoding="utf-8")
```

> **PERHATIAN 13.1**
>
> Jika Anda lupa menentukan encoding, Python akan memakai default sistem operasi (sering kali berbeda antara Windows, Mac, dan Linux). Konsekuensi: berkas yang dibuat di satu komputer mungkin tidak terbaca di komputer lain, atau karakter khusus seperti “é”, “°”, atau aksara non-Latin akan rusak. Selalu sertakan encoding=“utf-8” di kode produksi.

13.3.2 Membuka dan Menutup Berkas: open() dan close()

Fungsi `open()` menjadi titik masuk untuk operasi berkas. Setelah dipakai, berkas harus *ditutup* dengan `close()` agar sumber daya sistem operasi dilepaskan.

```
# Cara klasik (tidak direkomendasikan)
f = open("data.txt", "r", encoding="utf-8")
isi = f.read()
print(isi)
f.close()                       # WAJIB! Tapi mudah dilupakan

# Masalah: jika ada error di antara open dan close,
# close() tidak akan tereksekusi. File tetap terbuka,
# kemungkinan corrupt, dan memori bocor.

# Cara modern dengan with: aman, tidak perlu close manual
with open("data.txt", "r", encoding="utf-8") as f:
    isi = f.read()
    print(isi)
# Setelah blok with, file otomatis ditutup, BAHKAN jika error
```

13.3.3 Context Manager: Pernyataan with

Pernyataan **with** adalah *context manager* yang menjamin sumber daya (di sini, berkas) dibersihkan dengan benar setelah selesai dipakai, bahkan jika terjadi error. Ini adalah pola standar Python modern untuk file I/O dan sumber daya lain (koneksi database, lock, dll).

```
# Pola dasar with
with open("data.txt", "r", encoding="utf-8") as f:
    # 'f' hanya valid di dalam blok ini
    isi = f.read()
# Setelah keluar blok, f.close() otomatis dipanggil

# Membuka beberapa file sekaligus
with open("input.txt", "r") as f_in, open("output.txt", "w")
as f_out:
    data = f_in.read()
    f_out.write(data.upper())
# Keduanya akan otomatis ditutup
```

TIPS PRO 13.1

Sejak kira-kira tahun 2010, with statement adalah CARA STANDAR membuka berkas di Python. Kode profesional modern hampir tidak pernah menggunakan open()/close() manual. Latihlah refleks: setiap kali Anda mengetik ‘open(‘, otomatis pikirkan ‘with‘. Pola ini juga dipakai dengan banyak objek lain (koneksi database, kunci threading, dll), sehingga investasinya berlipat.

13.3.4 Mode Akses Berkas

Parameter kedua open() menentukan mode akses:

Mode	Nama	Perilaku
r	read (default)	Baca berkas. Error jika tidak ada.
w	write	Tulis. Berkas baru jika belum ada; HAPUS ISI jika sudah ada!
a	append	Tambah di akhir. Berkas baru jika belum ada.
x	exclusive create	Buat baru. Error jika sudah ada.
r+	read+write	Baca dan tulis. Error jika tidak ada.
rb / wb	binary mode	Untuk gambar, audio, dll. Bukan teks.
rt / wt	text mode	Eksplisit teks (default sebenarnya).

PERHATIAN 13.2

Mode “w” adalah JEBAKAN! Ia akan menghapus seluruh isi berkas yang sudah ada SEBELUM menulis. Jika Anda ingin menambah data tanpa menghapus, gunakan mode “a” (append). Banyak data hilang permanen karena programmer menggunakan “w” padahal yang dimaksud adalah “a”. Selalu pikirkan dua kali sebelum menggunakan mode “w”.

13.3.5 Membaca Berkas Teks

Python menyediakan beberapa cara membaca berkas, masing-masing untuk konteks berbeda:

```
# === Cara 1: read() - seluruh isi sebagai satu string ===
with open("data.txt", "r", encoding="utf-8") as f:
    isi = f.read()
    print(isi)
# Cocok untuk berkas kecil. Bahaya untuk berkas besar
(kehabisan RAM).

# === Cara 2: readline() - satu baris pada satu waktu ===
with open("data.txt", "r", encoding="utf-8") as f:
    baris_pertama = f.readline()
    baris_kedua = f.readline()
# Cocok jika hanya butuh sedikit baris dari awal berkas.

# === Cara 3: readlines() - list of strings ===
with open("data.txt", "r", encoding="utf-8") as f:
    baris_list = f.readlines()
    for i, baris in enumerate(baris_list, 1):
        print(f"{i}: {baris.strip()}")
# Memuat seluruh berkas ke memori. Hindari untuk berkas besar.

# === Cara 4 (TERBAIK): iterasi langsung ===
with open("data.txt", "r", encoding="utf-8") as f:
    for baris in f:                  # iterasi satu baris pada
satu waktu
        print(baris.strip())
# Efisien memori, dapat menangani berkas berukuran gigabyte.
```

> **TIPS PRO 13.2**
>
> Untuk berkas teks besar (log file, dataset, dll), HINDARI .read() dan .readlines(). Keduanya memuat seluruh berkas ke RAM. Gunakan iterasi langsung (for baris in f) yang membaca satu baris pada satu waktu. Inilah cara membaca berkas log puluhan gigabyte tanpa kehabisan memori—pola standar di engineering data.

13.3.6 Menulis dan Menambahkan ke Berkas

```
# === Menulis baru (mode "w") ===
with open("hasil.txt", "w", encoding="utf-8") as f:
    f.write("Halo, dunia!\n")
    f.write("Baris kedua.\n")
    # \n untuk baris baru—write tidak otomatis menambahkan

# === Menulis banyak baris dari list ===
baris = ["Andi\n", "Budi\n", "Cici\n"]
with open("nama.txt", "w", encoding="utf-8") as f:
    f.writelines(baris)

# === Menambahkan (mode "a") ===
with open("log.txt", "a", encoding="utf-8") as f:
    f.write("2026-05-16 10:30: Aktivitas baru\n")
# Tidak menghapus isi sebelumnya, hanya menambahkan di akhir.

# === Menulis data terstruktur ===
nilai_mhs = [("Andi", 85), ("Budi", 72), ("Cici", 91)]
with open("nilai.csv", "w", encoding="utf-8") as f:
    f.write("Nama,Nilai\n")            # header
    for nama, nilai in nilai_mhs:
        f.write(f"{nama},{nilai}\n")
```

13.3.7 Eksepsi dan Pernyataan try-except

Eksepsi (exception) adalah kejadian tak terduga yang menginterupsi alur normal program. Tanpa penanganan, eksepsi menyebabkan program crash. Dengan **try-except**, kita dapat menangkap dan merespons eksepsi dengan elegan.

```
# Sintaks dasar
try:
    # kode yang mungkin menimbulkan eksepsi
    angka = int(input("Masukkan angka: "))
    hasil = 100 / angka
    print(f"100 / {angka} = {hasil}")
except ValueError:
    print("Input bukan angka valid.")
except ZeroDivisionError:
```

```
    print("Tidak bisa dibagi nol.")
except Exception as e:
    print(f"Eksepsi tak terduga: {e}")
```

Beberapa Eksepsi Bawaan yang Sering Muncul

Eksepsi	Kapan muncul
ValueError	Tipe benar tapi nilai tidak valid, mis. int("abc")
TypeError	Operasi pada tipe yang tidak kompatibel, mis. "5" + 5
ZeroDivisionError	Pembagian dengan nol
FileNotFoundError	Berkas tidak ditemukan saat dibuka mode "r"
PermissionError	Tidak ada izin akses berkas atau folder
IndexError	Akses indeks di luar rentang list
KeyError	Akses kunci yang tidak ada di dictionary
AttributeError	Akses atribut yang tidak dimiliki objek
KeyboardInterrupt	Pengguna menekan Ctrl+C

PERHATIAN 13.3

JANGAN tangkap Exception terlalu luas (“except:” atau “except Exception:”) jika Anda tidak benar-benar tahu apa yang akan terjadi. Eksepsi yang tertangkap tanpa penanganan yang tepat menyembunyikan bug dan membuat debugging mimpi buruk. Aturan praktis: TANGKAP YANG SPESIFIK. Hanya tangkap eksepsi yang Anda tahu cara menanganinya. Untuk yang lain, biarkan program crash dengan pesan jelas—itu lebih baik daripada perilaku misterius.

13.3.8 Klausa else, finally, dan Pernyataan raise

Klausa else

Blok else pada try-except dieksekusi HANYA JIKA tidak ada eksepsi yang terjadi. Berguna untuk memisahkan kode yang bergantung pada sukses-nya try.

```
try:
    angka = int(input("Angka: "))
except ValueError:
    print("Input tidak valid.")
else:
    # hanya berjalan jika int() sukses
    print(f"Angka kuadrat: {angka ** 2}")
```

Klausa finally

Blok finally SELALU dieksekusi, baik ada eksepsi maupun tidak. Berguna untuk pembersihan sumber daya (meskipun with sudah menangani ini untuk file).

```
try:
    f = open("data.txt")
    data = f.read()
    angka = int(data)
except FileNotFoundError:
    print("File tidak ada.")
except ValueError:
    print("Isi file bukan angka.")
finally:
    print("Selesai (apapun hasilnya).")
    # Cleanup, log, dll yang harus selalu dilakukan
```

Pernyataan raise

raise digunakan untuk MENIMBULKAN eksepsi secara sengaja. Berguna untuk validasi input atau menandai kondisi error di fungsi yang Anda buat.

```
def umur_valid(umur: int) -> int:
    """Validasi umur; raise ValueError jika tidak valid."""
```

```
    if not isinstance(umur, int):
        raise TypeError("Umur harus integer.")
    if umur < 0:
        raise ValueError("Umur tidak boleh negatif.")
    if umur > 150:
        raise ValueError("Umur tidak realistis.")
    return umur

try:
    u = umur_valid(-5)
except ValueError as e:
    print(f"Error: {e}")
```

DI BALIK LAYAR 13.1

Eksepsi di Python diorganisir dalam hierarki kelas. BaseException adalah induk segalanya; Exception adalah induk dari eksepsi normal yang biasa Anda tangani. ValueError, TypeError, dll adalah turunan dari Exception. Hierarki ini memungkinkan except Exception menangkap semua eksepsi normal, sedangkan except ValueError hanya yang spesifik. Pemahaman hierarki ini penting untuk menangkap dengan presisi—kita akan mendalaminya di mata kuliah OOP.

13.4 Contoh Kode Lengkap

Skrip berikut adalah *Pencatat Log Aktivitas* yang mendemonstrasikan integrasi penuh file I/O dan exception handling: menulis log dengan timestamp, membaca dan menampilkan log, validasi input, dan penanganan kesalahan.

```
"""
Program: Pencatat Log Aktivitas
Tujuan : Demonstrasi file I/O + exception handling
"""

from datetime import datetime

LOG_FILE = "aktivitas.log"
```

```
def catat_aktivitas(pesan: str, level: str = "INFO") -> bool:
    """Mencatat satu entri log dengan timestamp."""
    if not pesan.strip():
        raise ValueError("Pesan tidak boleh kosong.")

    waktu = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
    entri = f"[{waktu}] [{level}] {pesan}\n"

    try:
        with open(LOG_FILE, "a", encoding="utf-8") as f:
            f.write(entri)
        return True
    except PermissionError:
        print("ERROR: Tidak ada izin menulis ke log.")
        return False
    except IOError as e:
        print(f"ERROR: Gagal menulis log: {e}")
        return False

def tampilkan_log(n: int = 10) -> None:
    """Tampilkan n entri log terakhir."""
    try:
        with open(LOG_FILE, "r", encoding="utf-8") as f:
            baris = f.readlines()
    except FileNotFoundError:
        print("Log kosong (file belum ada).")
        return

    if not baris:
        print("Log kosong.")
        return

    # Ambil n terakhir
    terakhir = baris[-n:] if len(baris) >= n else baris
    print(f"\n--- {len(terakhir)} aktivitas terakhir ---")
    for b in terakhir:
        print(b.strip())

def hitung_log() -> dict:
    """Hitung jumlah entri per level."""
    counts = {"INFO": 0, "WARNING": 0, "ERROR": 0}
    try:
        with open(LOG_FILE, "r", encoding="utf-8") as f:
```

```
            for baris in f:
                for level in counts:
                    if f"[{level}]" in baris:
                        counts[level] += 1
                        break
    except FileNotFoundError:
        pass            # file belum ada—semuanya tetap 0
    return counts

def menu():
    """Menu interaktif."""
    while True:
        print("\n" + "=" * 40)
        print("   PENCATAT LOG AKTIVITAS")
        print("=" * 40)
        print("1. Catat aktivitas baru")
        print("2. Lihat 10 log terakhir")
        print("3. Lihat statistik level")
        print("4. Keluar")

        try:
            pilih = input("Pilihan: ").strip()
        except KeyboardInterrupt:
            print("\nDibatalkan.")
            break

        if pilih == "1":
            pesan = input("Pesan log: ")
            level = input("Level (INFO/WARNING/ERROR):
").upper() or "INFO"
            try:
                if catat_aktivitas(pesan, level):
                    print("✓ Tercatat.")
            except ValueError as e:
                print(f"✗ {e}")
        elif pilih == "2":
            tampilkan_log()
        elif pilih == "3":
            stats = hitung_log()
            print("\n--- Statistik ---")
            for level, jumlah in stats.items():
                print(f"{level:10}: {jumlah}")
        elif pilih == "4":
            print("Selamat tinggal.")
```

```
            break
        else:
            print("Pilihan tidak valid.")

if __name__ == "__main__":
    menu()
```

Skrip ini menggabungkan: `with` untuk file I/O aman, `try-except` untuk PermissionError dan FileNotFoundError, `raise` untuk validasi input, mode `"a"` untuk append tanpa menghapus, `KeyboardInterrupt` untuk handling Ctrl+C, dan pola modular dengan `if __name__ == "__main__"`.

13.5 Kesalahan Umum dan Cara Mengatasinya

Kesalahan 1: Lupa close() pada cara klasik.

Miskonsepsi: "Saya pakai f = open(…) dan tidak pakai close(). Programnya jalan kok."

Realita: Tanpa close(), file tetap terbuka di sistem operasi sampai garbage collector Python menutupnya—tidak dijamin. Untuk file write, perubahan mungkin tidak ter-flush ke disk. Solusi: SELALU gunakan with statement.

Kesalahan 2: Mode \"w\" menghapus file.

Miskonsepsi: "Saya buka file dengan 'w' untuk menambahkan data, tapi isi lama hilang."

Realita: Mode 'w' menghapus seluruh isi file sebelum menulis. Gunakan mode 'a' (append) untuk menambah di akhir, atau 'r+' untuk membaca dan menulis.

Kesalahan 3: Tidak menentukan encoding.

Miskonsepsi: "File saya tampil aneh di komputer lain, ada karakter rusak."

Realita: Tanpa encoding= eksplisit, Python pakai default OS yang berbeda-beda. Selalu sertakan encoding="utf-8" agar konsisten antar sistem.

Kesalahan 4: Menangkap eksepsi terlalu luas.

Miskonsepsi: "Saya pakai except: untuk tangkap semua, supaya tidak crash."

Realita: Ini menyembunyikan bug. Jika ada KeyError atau AttributeError dari logika yang salah, Anda tidak akan tahu—programnya tampak 'jalan' tapi hasilnya salah. Tangkap eksepsi spesifik yang Anda tahu cara menanganinya.

Kesalahan 5: Membaca file besar dengan .read().

Miskonsepsi: "Saya baca file 5GB dengan .read() dan komputer hang."

Realita: .read() memuat seluruh file ke RAM. Untuk file besar, gunakan iterasi langsung: for baris in f: …. Ini efisien memori dan dapat menangani file berukuran terabyte.

13.6 Rangkuman

Bab 13 telah memperkenalkan dua keterampilan kunci untuk membangun aplikasi yang tangguh: file I/O dan exception handling. File I/O memungkinkan program menyimpan dan memuat data secara permanen, mengubahnya dari prototype menjadi aplikasi yang berguna. Fungsi open() membuka berkas dengan mode tertentu (r, w, a, x), dan pernyataan with menjamin penutupan otomatis yang aman. Selalu sertakan encoding="utf-8" untuk konsistensi lintas sistem.

Untuk membaca berkas, gunakan iterasi langsung (for baris in f) yang efisien memori, kecuali jika berkas pasti kecil. Untuk menulis, mode "w" menggantikan seluruh isi sedangkan "a" menambahkan di akhir—pilihan yang salah dapat menyebabkan kehilangan data permanen.

Exception handling dengan try-except memungkinkan program merespons kesalahan tanpa crash. Klausa else dieksekusi jika tidak ada eksepsi, finally selalu dieksekusi, dan raise menimbulkan eksepsi sendiri untuk validasi. Aturan praktis: tangkap eksepsi spesifik (ValueError, FileNotFoundError, dll), bukan Exception umum. Penggabungan file I/O dan exception handling menjadi fondasi untuk aplikasi production yang andal.

13.7 Latihan Terbimbing

Soal 13.7.1

Soal: Tulis program yang membaca file teks dan menampilkan: jumlah baris, jumlah kata, dan jumlah karakter (tidak termasuk newline).

Pembahasan:

```
nama_file = input("Nama file: ")

try:
    with open(nama_file, "r", encoding="utf-8") as f:
        isi = f.read()
    baris = isi.count("\n") + (1 if isi and not
isi.endswith("\n") else 0)
    kata = len(isi.split())
    karakter = len(isi) - isi.count("\n")
    print(f"Baris    : {baris}")
    print(f"Kata     : {kata}")
    print(f"Karakter : {karakter}")
except FileNotFoundError:
    print(f"File '{nama_file}' tidak ditemukan.")
```

Soal 13.7.2

Soal: Apa output dan urutan eksekusi kode berikut? try: print("A"); x = int("abc"); print("B") except ValueError: print("C") else: print("D") finally: print("E")

Pembahasan: Output: A C E. Penjelasan: (1) print("A") berjalan. (2) int("abc") melempar ValueError. (3) Karena ada eksepsi, print("B") DI-SKIP, langsung ke except. (4) print("C") berjalan. (5) Karena ada eksepsi yang tertangkap, else BLOCK DI-SKIP (else hanya jika tidak ada eksepsi). (6) finally SELALU berjalan: print("E").

Soal 13.7.3

Soal: Tulis program yang menyalin isi satu file ke file lain. Tangani semua kemungkinan error.

Pembahasan:

```
sumber = input("File sumber: ")
tujuan = input("File tujuan: ")

try:
    with open(sumber, "r", encoding="utf-8") as src, \\
```

```
         open(tujuan, "w", encoding="utf-8") as dst:
        for baris in src:
            dst.write(baris)
    print(f"✓ Berhasil menyalin {sumber} → {tujuan}")
except FileNotFoundError:
    print(f"File sumber '{sumber}' tidak ditemukan.")
except PermissionError:
    print("Tidak ada izin baca/tulis.")
except IOError as e:
    print(f"Error I/O: {e}")
```

Soal 13.7.4

Soal: Tulis fungsi validasi_email(email) yang menggunakan raise untuk eksepsi: kosong, tidak ada @, tidak ada titik setelah @. Sertakan tes.

Pembahasan:

```
def validasi_email(email: str) -> str:
    """Validasi email; raise ValueError jika tidak valid."""
    email = email.strip()
    if not email:
        raise ValueError("Email tidak boleh kosong.")
    if "@" not in email:
        raise ValueError("Email harus mengandung @.")
    bagian = email.split("@")
    if len(bagian) != 2 or "." not in bagian[1]:
        raise ValueError("Domain email tidak valid.")
    return email.lower()

# Test
for inp in ["", "yusri", "yusri@example.com", "a@b"]:
    try:
        print(f"'{inp}' → OK: {validasi_email(inp)}")
    except ValueError as e:
        print(f"'{inp}' → INVALID: {e}")
```

Soal 13.7.5

Soal: Tulis program yang menambah satu baris ke file 'catatan.txt' setiap kali dipanggil, dengan timestamp. Pastikan file tidak hilang jika sudah ada.

Pembahasan:

```
from datetime import datetime

catatan = input("Catatan baru: ")

if not catatan.strip():
    print("Catatan kosong, dibatalkan.")
else:
    waktu = datetime.now().strftime("%Y-%m-%d %H:%M")
    try:
        # Mode "a" PENTING: tidak menghapus isi lama
        with open("catatan.txt", "a", encoding="utf-8") as f:
            f.write(f"[{waktu}] {catatan}\n")
        print("✓ Catatan tersimpan.")
    except PermissionError:
        print("Tidak ada izin menulis ke file.")
```

13.8 Latihan Mandiri

- **(C2)** Jelaskan perbedaan mode r, w, a, dan r+. Pada kasus seperti apa masing-masing dipakai?
- **(C2)** Apa keuntungan pernyataan with dibanding open()/close() manual? Berikan minimal tiga alasan.
- **(C3)** Tulis program yang membaca file CSV sederhana (header + data) dan menampilkan dalam format tabel rapi.
- **(C3)** Buat program yang menyimpan daftar tugas (to-do list) ke file. Setiap kali program dijalankan, daftar lama dimuat, dan pengguna dapat menambah tugas baru.
- **(C3)** Tulis fungsi cari_kata_dalam_file(kata, nama_file) yang mengembalikan list baris yang mengandung kata tersebut (case-insensitive).
- **(C3)** Buatlah program yang mensimulasikan login: baca file users.txt yang berisi 'username:password' per baris. Pengguna memasukkan kredensial dan diberi tahu apakah login berhasil.
- **(C4)** Analisis kode: try: open("a.txt"); except FileNotFoundError: print("x"); open("a.txt") tetap akan menimbulkan error tidak tertangkap. Mengapa? Bagaimana memperbaiki?

- **(C4)** Bandingkan tiga cara membaca file: .read(), .readlines(), dan iterasi langsung. Pada kasus apa masing-masing optimal?
- **(C4)** Studi kasus: tulis program konverter format. Input: file dengan format 'nama,nilai' per baris. Output: file baru dengan format 'nama:nilai' (titik dua sebagai pemisah). Tangani semua kemungkinan error.
- **(C4)** Diskusikan: mengapa Python merancang exception handling sebagai mekanisme utama untuk error handling (bukan return code seperti di C)? Apa kelebihan dan kekurangannya?

13.9 Tugas Proyek Mini

"Manajer Catatan Harian dengan Persistence"

Bangun aplikasi catatan harian yang menyimpan data ke file dan dapat dimuat ulang. Persyaratan:

- Setiap catatan berisi: ID otomatis, timestamp, kategori, dan isi. Disimpan dalam format CSV: ID,timestamp,kategori,isi.
- File data: `catatan.csv` (otomatis dibuat jika belum ada).
- Menu: (1) Tambah catatan, (2) Lihat semua, (3) Cari berdasarkan kata kunci, (4) Filter per kategori, (5) Hapus catatan, (6) Statistik, (7) Keluar.
- Validasi: isi tidak boleh kosong (raise ValueError), kategori harus salah satu dari list valid.
- Setiap perubahan langsung disimpan ke file (write-through), sehingga jika program crash, data tidak hilang.
- Tangani semua kemungkinan eksepsi dengan pesan yang jelas: FileNotFoundError, PermissionError, ValueError, KeyboardInterrupt (Ctrl+C → simpan dan keluar dengan elegan).
- Sertakan logging: aktivitas penting (tambah, hapus) dicatat ke `aktivitas.log` dengan timestamp.

Format pengumpulan: berkas .py beserta contoh file catatan.csv dan aktivitas.log setelah uji coba, dan dokumentasi minimal 1 halaman tentang struktur file dan penanganan error.

13.10 Refleksi Diri

Centang pernyataan berikut yang sudah Anda kuasai:

☐ Saya memahami konsep berkas pada sistem operasi (path, encoding).

☐ Saya selalu menggunakan with statement untuk file I/O.

☐ Saya tahu perbedaan mode r, w, a, dan kapan menggunakan masing-masing.

☐ Saya dapat membaca file dengan iterasi langsung (efisien memori).

☐ Saya dapat menulis dan menambahkan ke file dengan format yang sesuai.

☐ Saya selalu menyertakan encoding="utf-8" di kode produksi.

☐ Saya dapat menulis try-except untuk menangkap eksepsi spesifik.

☐ Saya memahami fungsi else dan finally pada try block.

☐ Saya dapat menggunakan raise untuk validasi di fungsi sendiri.

☐ Saya tidak menangkap Exception terlalu luas, hanya yang spesifik dan terkelola.

ASAH OTAK 13.2

Eksperimen mandiri: buat file dummy 1 GB dengan loop yang menulis string panjang berulang kali. Lalu, baca file tersebut dengan dua cara: (a) sekali baca dengan .read(), (b) iterasi langsung. Bandingkan penggunaan RAM dengan task manager. Anda akan melihat perbedaan dramatis. Inilah alasan mengapa engineering data menggunakan streaming, bukan loading penuh—satu konsep yang akan terus relevan sepanjang karier Anda.

BAB 14

STUDI KASUS MINI-PROYEK

"Talk is cheap. Show me the code."
— Linus Torvalds, pencipta Linux

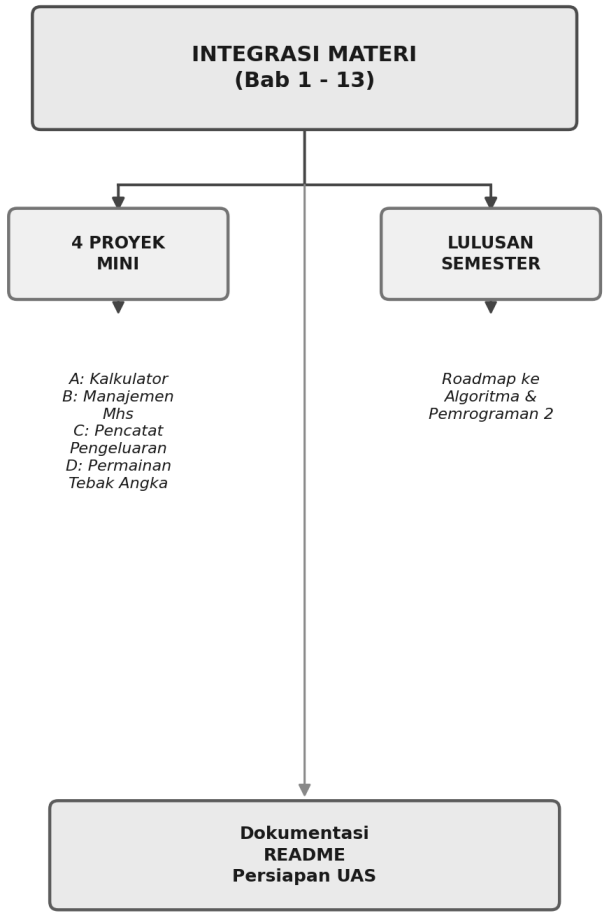
Capaian Pembelajaran Bab

Setelah mempelajari bab ini, mahasiswa diharapkan mampu mencapai sejumlah luaran pembelajaran sebagai berikut:

Kode	Capaian	Bloom
CPB-14.1	Menerapkan metodologi pengerjaan proyek mini secara sistematis	C3
CPB-14.2	Mengintegrasikan seluruh konsep Bab 1–13 dalam aplikasi end-to-end	C4
CPB-14.3	Merancang aplikasi dengan struktur modular dan persistence data	C4
CPB-14.4	Mendokumentasikan proyek dengan README dan komentar profesional	C3
CPB-14.5	Mengevaluasi kualitas kode sendiri berdasarkan best practices	C5
CPB-14.6	Mempersiapkan diri menuju mata kuliah Algoritma dan Pemrograman 2	C2

14.1 Peta Konsep

Peta konsep berikut menggambarkan empat proyek mini sebagai titik puncak integrasi semua materi semester.



14.2 Apersepsi

Empat belas pekan telah berlalu sejak Anda pertama kali membuka buku ini. Pada Bab 1, sebagian dari Anda mungkin belum pernah menulis satu pun baris kode. Kini, di Bab 14, Anda mampu merancang aplikasi yang menerima input pengguna, memprosesnya melalui logika kompleks, menyimpan data secara permanen, menangani kesalahan dengan elegan, dan menyajikan hasilnya dalam format yang rapi. Perjalanan yang luar biasa.

Tetapi pengetahuan terpencar adalah pengetahuan yang lemah. Mahasiswa yang menguasai semua konsep secara terpisah, namun tidak pernah menyatukannya, akan kesulitan ketika menghadapi masalah dunia nyata yang tidak datang dalam kotak-kotak rapi seperti soal latihan. Bab terakhir ini ada untuk satu tujuan: **menyatukan semua serpihan menjadi keterampilan utuh**.

Anda akan diajak merancang empat proyek mini yang berbeda dalam domain, tetapi sama dalam tuntutan: penggunaan modular fungsi, manajemen state dengan struktur data koleksi, persistence dengan file I/O, dan ketangguhan dengan exception handling. Setelah menyelesaikan bab ini, Anda tidak hanya "mengetahui" Python; Anda akan *menjadi* seseorang yang dapat menggunakan Python untuk memecahkan masalah.

> **ASAH OTAK 14.1**
>
> Sebelum melanjutkan, renungkan: Apa satu masalah dalam hidup Anda saat ini yang dapat dibantu oleh program komputer? Bisa sangat sederhana—mencatat pengeluaran, menghitung biaya kos, mengatur jadwal tugas. Jangan menulis apa pun, cukup pikirkan. Pada akhir bab ini, Anda akan memiliki perangkat untuk membangun solusinya.

14.3 Uraian Materi

14.3.1 Metodologi Pengerjaan Proyek Mini

Sebelum menyelam ke kode, pahami dulu metodologi yang akan dipakai pada setiap proyek. Metodologi ini adalah perluasan dari metode Polya di Bab 8:

- **Spesifikasi (1 jam).** Tulis dalam bahasa alami: apa yang dilakukan program, siapa penggunanya, fitur utamanya, dan batasannya. Jangan sentuh kode.
- **Perancangan struktur data (30 menit).** Tentukan bagaimana data akan direpresentasikan: list of dicts? Dict of lists? Bagaimana di-persist ke file? Format apa (txt, csv, json)?

- **Perancangan modul/fungsi (30 menit).** Daftar fungsi apa saja yang diperlukan, beserta signature-nya. Bagi proyek ke beberapa modul jika sudah besar.
- **Implementasi inkremental (60-90 menit).** Bangun satu fitur dulu sampai jalan, uji, baru pindah ke fitur berikutnya. JANGAN tulis semuanya sekaligus.
- **Testing menyeluruh (30 menit).** Uji setiap fitur dengan kasus normal, batas, dan istimewa. Catat bug yang ditemukan.
- **Dokumentasi (30 menit).** Tulis README.md dengan deskripsi, cara pakai, fitur, dan contoh output. Sertakan komentar di kode pada bagian yang kompleks.

> **TIPS PRO 14.1**
>
> Pemula sering melompati tahap 1-3 dan langsung ke implementasi. Ini menyebabkan kode yang berantakan, fitur tidak konsisten, dan banyak refactoring di tengah jalan. Programmer profesional melakukan kebalikannya: menghabiskan 30-40% waktu di perancangan, sehingga implementasi lebih cepat dan kualitasnya lebih tinggi. Aturan klasik: "Weeks of programming can save hours of planning."

14.3.2 Proyek A: Kalkulator Aritmatika dengan Riwayat

Spesifikasi: Bangun kalkulator command-line yang melakukan empat operasi dasar (+, -, *, /), menyimpan riwayat operasi terakhir, dan dapat menampilkan ulang serta menyimpan riwayat ke file.

Konsep yang Dipakai

- Fungsi untuk operasi aritmatika (Bab 11)
- Exception handling untuk pembagian nol dan input tidak valid (Bab 13)
- List untuk menyimpan riwayat operasi (Bab 9)
- File I/O untuk persistence riwayat (Bab 13)
- Loop while dengan menu interaktif (Bab 7)

Kode Implementasi

```
"""
Proyek A: Kalkulator dengan Riwayat
"""

from datetime import datetime

FILE_RIWAYAT = "riwayat_kalkulator.txt"

# === Fungsi operasi ===
def tambah(a: float, b: float) -> float: return a + b
def kurang(a: float, b: float) -> float: return a - b
def kali(a: float, b: float) -> float: return a * b

def bagi(a: float, b: float) -> float:
    if b == 0:
        raise ZeroDivisionError("Tidak boleh dibagi nol")
    return a / b

OPERASI = {
    "+": ("Tambah", tambah),
    "-": ("Kurang", kurang),
    "*": ("Kali",   kali),
    "/": ("Bagi",   bagi)
}

def catat_riwayat(riwayat: list, a: float, op: str, b: float,
                  hasil: float) -> None:
    """Tambahkan satu entri ke riwayat dengan timestamp."""
    waktu = datetime.now().strftime("%H:%M:%S")
    entri = f"[{waktu}] {a} {op} {b} = {hasil}"
    riwayat.append(entri)

def simpan_riwayat(riwayat: list) -> None:
    """Tulis riwayat ke file."""
    with open(FILE_RIWAYAT, "a", encoding="utf-8") as f:
        for r in riwayat:
            f.write(r + "\n")

def tampilkan_riwayat(riwayat: list) -> None:
    """Cetak semua riwayat sesi ini."""
    if not riwayat:
        print("Belum ada operasi.")
```

```
        return
    print("\n--- Riwayat Sesi Ini ---")
    for r in riwayat:
        print(r)

def input_angka(prompt: str) -> float:
    """Input dengan validasi ulang sampai valid."""
    while True:
        try:
            return float(input(prompt))
        except ValueError:
            print("  > Bukan angka valid, coba lagi.")

def main():
    riwayat = []
    print("=" * 40)
    print("   KALKULATOR ARITMATIKA")
    print("=" * 40)

    while True:
        print("\nOperasi: + - * /  |  Lainnya: r=riwayat,
q=keluar")
        pilih = input("Pilih: ").strip().lower()

        if pilih == "q":
            print("Selamat tinggal!")
            simpan_riwayat(riwayat)
            break
        elif pilih == "r":
            tampilkan_riwayat(riwayat)
        elif pilih in OPERASI:
            nama_op, fungsi = OPERASI[pilih]
            a = input_angka("  Angka 1: ")
            b = input_angka("  Angka 2: ")
            try:
                hasil = fungsi(a, b)
                print(f"  Hasil  : {hasil}")
                catat_riwayat(riwayat, a, pilih, b, hasil)
            except ZeroDivisionError as e:
                print(f"  Error  : {e}")
        else:
            print("Pilihan tidak valid.")
 if __name__ == "__main__":
    main()
```

14.3.3 Proyek B: Manajemen Daftar Mahasiswa (CRUD)

Spesifikasi: Aplikasi pengelolaan data mahasiswa dengan operasi CRUD (Create, Read, Update, Delete). Data disimpan ke file CSV dan dimuat ulang setiap program dimulai.

Konsep yang Dipakai

- List of dicts untuk data mahasiswa (Bab 9 dan 10)
- File I/O CSV untuk persistence (Bab 13)
- Fungsi modular untuk setiap operasi CRUD (Bab 11)
- Exception handling untuk file dan input (Bab 13)

Kode Implementasi

```
"""
Proyek B: Manajemen Daftar Mahasiswa (CRUD)
Data tersimpan dalam mahasiswa.csv
"""

FILE = "mahasiswa.csv"
HEADER = "NIM,Nama,Jurusan,IPK"

def muat_data() -> list:
    """Muat data dari file CSV. Mengembalikan list of
dicts."""
    try:
        with open(FILE, "r", encoding="utf-8") as f:
            baris = f.readlines()
    except FileNotFoundError:
        return []      # file belum ada—mulai dari kosong

    data = []
    for b in baris[1:]:                # skip header
        parts = b.strip().split(",")
        if len(parts) == 4:
            data.append({
                "nim": parts[0],
                "nama": parts[1],
                "jurusan": parts[2],
                "ipk": float(parts[3])
```

```
            })
    return data

def simpan_data(data: list) -> None:
    """Tulis data ke file CSV."""
    with open(FILE, "w", encoding="utf-8") as f:
        f.write(HEADER + "\n")
        for m in data:

f.write(f"{m['nim']},{m['nama']},{m['jurusan']},{m['ipk']}\n")

def cari_indeks(data: list, nim: str) -> int:
    """Cari indeks mahasiswa berdasarkan NIM. -1 jika tidak
ada."""
    for i, m in enumerate(data):
        if m["nim"] == nim:
            return i
    return -1

def tambah(data: list) -> None:
    nim = input("NIM: ").strip()
    if cari_indeks(data, nim) != -1:
        print("NIM sudah terdaftar.")
        return
    try:
        nama = input("Nama: ").strip().title()
        jurusan = input("Jurusan: ").strip().title()
        ipk = float(input("IPK: "))
        if not (0 <= ipk <= 4):
            raise ValueError("IPK harus 0–4.")
        data.append({"nim": nim, "nama": nama,
                     "jurusan": jurusan, "ipk": ipk})
        simpan_data(data)
        print("✓ Mahasiswa ditambahkan.")
    except ValueError as e:
        print(f"Input tidak valid: {e}")

def tampil_semua(data: list) -> None:
    if not data:
        print("Belum ada data.")
        return

print(f"\n{'NIM':<12}{'Nama':<25}{'Jurusan':<20}{'IPK':>6}")
    print("-" * 63)
```

```
    for m in sorted(data, key=lambda x: x["nim"]):

print(f"{m['nim']:<12}{m['nama']:<25}{m['jurusan']:<20}"
              f"{m['ipk']:>6.2f}")

def update(data: list) -> None:
    nim = input("NIM yang akan diupdate: ").strip()
    idx = cari_indeks(data, nim)
    if idx == -1:
        print("NIM tidak ditemukan.")
        return
    print(f"Data saat ini: {data[idx]}")
    field = input("Field (nama/jurusan/ipk):
").strip().lower()
    if field not in ("nama", "jurusan", "ipk"):
        print("Field tidak valid.")
        return
    baru = input(f"{field.title()} baru: ").strip()
    try:
        if field == "ipk":
            data[idx][field] = float(baru)
        else:
            data[idx][field] = baru.title()
        simpan_data(data)
        print("✓ Data diperbarui.")
    except ValueError:
        print("Input tidak valid.")

def hapus(data: list) -> None:
    nim = input("NIM yang akan dihapus: ").strip()
    idx = cari_indeks(data, nim)
    if idx == -1:
        print("NIM tidak ditemukan.")
        return
    konfirmasi = input(f"Hapus {data[idx]['nama']}? (y/n): ")
    if konfirmasi.lower() == "y":
        del data[idx]
        simpan_data(data)
        print("✓ Mahasiswa dihapus.")

def main():
    data = muat_data()
    while True:
        print("\n=== MANAJEMEN MAHASISWA ===")
```

```
        print("1. Tambah  2. Tampil  3. Update  4. Hapus  5.
Keluar")
        pilih = input("Pilih: ").strip()
        if pilih == "1": tambah(data)
        elif pilih == "2": tampil_semua(data)
        elif pilih == "3": update(data)
        elif pilih == "4": hapus(data)
        elif pilih == "5": break
        else: print("Pilihan tidak valid.")

if __name__ == "__main__":
    main()
```

14.3.4 Proyek C: Pencatat Pengeluaran Harian

Spesifikasi: Aplikasi pencatat pengeluaran pribadi dengan kategorisasi, laporan harian/mingguan/bulanan, dan analisis pola pengeluaran. Cocok untuk mahasiswa yang ingin mengelola keuangan.

Konsep yang Dipakai

- Dict untuk struktur data per transaksi (Bab 10)
- Modul datetime untuk timestamp dan filtering periode (Bab 12)
- Set untuk daftar kategori unik (Bab 10)
- List comprehension untuk filter dan agregasi (Bab 9)
- File I/O CSV untuk persistence (Bab 13)

Inti Kode (versi singkat untuk dipelajari)

```
"""
Proyek C: Pencatat Pengeluaran
"""

from datetime import datetime, timedelta

FILE = "pengeluaran.csv"
KATEGORI_VALID = {"Makanan", "Transport", "Pendidikan",
                  "Hiburan", "Kesehatan", "Lainnya"}

def parse_tanggal(s: str) -> datetime:
```

```
    """Parse format YYYY-MM-DD."""
    return datetime.strptime(s, "%Y-%m-%d")

def muat_data() -> list:
    try:
        with open(FILE, "r", encoding="utf-8") as f:
            baris = f.readlines()[1:]   # skip header
    except FileNotFoundError:
        return []
    return [{
        "tanggal": parse_tanggal(b.split(",")[0]),
        "kategori": b.split(",")[1],
        "jumlah": float(b.split(",")[2]),
        "keterangan": b.split(",")[3].strip()
    } for b in baris if len(b.split(",")) == 4]

def simpan_data(data: list) -> None:
    with open(FILE, "w", encoding="utf-8") as f:
        f.write("Tanggal,Kategori,Jumlah,Keterangan\n")
        for d in data:
            tgl = d["tanggal"].strftime("%Y-%m-%d")

f.write(f"{tgl},{d['kategori']},{d['jumlah']},{d['keterangan']}\n")

def tambah(data: list) -> None:
    try:
        kategori = input(f"Kategori {KATEGORI_VALID}:
").strip().title()
        if kategori not in KATEGORI_VALID:
            raise ValueError("Kategori tidak valid.")
        jumlah = float(input("Jumlah (Rp): "))
        if jumlah <= 0:
            raise ValueError("Jumlah harus positif.")
        keterangan = input("Keterangan: ").strip()
        data.append({
            "tanggal": datetime.now(),
            "kategori": kategori,
            "jumlah": jumlah,
            "keterangan": keterangan
        })
        simpan_data(data)
        print("✓ Tercatat.")
    except ValueError as e:
        print(f"Error: {e}")
```

```
def laporan_periode(data: list, hari: int) -> None:
    """Laporan pengeluaran N hari terakhir."""
    batas = datetime.now() - timedelta(days=hari)
    dalam_periode = [d for d in data if d["tanggal"] >= batas]

    if not dalam_periode:
        print(f"Tidak ada pengeluaran {hari} hari terakhir.")
        return

    total = sum(d["jumlah"] for d in dalam_periode)
    print(f"\n--- {hari} hari terakhir ---")
    print(f"Total: Rp {total:,.0f}")

    # Per kategori
    per_kategori = {}
    for d in dalam_periode:
        per_kategori[d["kategori"]] =
per_kategori.get(d["kategori"], 0) + d["jumlah"]

    print("\nPer kategori:")
    for k, v in sorted(per_kategori.items(), key=lambda x: -x[1]):
        persen = (v / total) * 100
        print(f"  {k:<15} Rp {v:>10,.0f} ({persen:.1f}%)")

def main():
    data = muat_data()
    while True:
        print("\n=== PENCATAT PENGELUARAN ===")
        print("1. Tambah  2. Laporan 7 hari  3. Laporan 30 hari  4.
Keluar")
        pilih = input("Pilih: ").strip()
        if pilih == "1": tambah(data)
        elif pilih == "2": laporan_periode(data, 7)
        elif pilih == "3": laporan_periode(data, 30)
        elif pilih == "4": break

if __name__ == "__main__":
    main()
```

14.3.5 Proyek D: Permainan Tebak Angka dengan Leaderboard

Spesifikasi: Game tebak angka yang lebih kompleks dari versi di Bab 7: ada level kesulitan, sistem skor, dan leaderboard yang disimpan ke file.

Konsep yang Dipakai

- Modul random untuk angka acak (Bab 12)
- Match-case untuk pilihan level (Bab 6)
- List of tuples untuk leaderboard yang dapat di-sort (Bab 9)
- File I/O untuk persistence skor (Bab 13)
- Fungsi untuk modularisasi (Bab 11)

Kode Implementasi

```
"""
Proyek D: Tebak Angka dengan Leaderboard
"""

import random
from datetime import datetime

FILE_SKOR = "leaderboard.csv"

def konfigurasi_level(level: str) -> tuple:
    """Mengembalikan (min, max, percobaan_maksimum, bobot_skor)."""
    match level:
        case "mudah":   return (1, 50,  10, 100)
        case "sedang":  return (1, 100, 8,  200)
        case "sulit":   return (1, 500, 7,  400)
        case _:         return (1, 100, 8,  200)

def hitung_skor(percobaan: int, maks: int, bobot: int) -> int:
    """Skor berdasarkan sisa percobaan."""
    sisa = maks - percobaan + 1
    return max(0, sisa * bobot // maks)

def muat_leaderboard() -> list:
    try:
        with open(FILE_SKOR, "r", encoding="utf-8") as f:
            return [(b.split(",")[0],
                     int(b.split(",")[1]),
                     b.split(",")[2],
```

```
                    b.split(",")[3].strip())
                   for b in f.readlines()[1:] if b.strip()]
    except FileNotFoundError:
        return []

def simpan_leaderboard(board: list) -> None:
    with open(FILE_SKOR, "w", encoding="utf-8") as f:
        f.write("Nama,Skor,Level,Tanggal\n")
        for nama, skor, level, tgl in board:
            f.write(f"{nama},{skor},{level},{tgl}\n")

def tampil_leaderboard(board: list, n: int = 10) -> None:
    if not board:
        print("Leaderboard kosong.")
        return
    board_urut = sorted(board, key=lambda x: -x[1])[:n]
    print(f"\n--- TOP {min(n, len(board_urut))} ---")

print(f"{'No':<4}{'Nama':<15}{'Skor':>7}{'Level':>10}{'Tanggal':>13}")
    for i, (nama, skor, level, tgl) in enumerate(board_urut, 1):
        print(f"{i:<4}{nama:<15}{skor:>7}{level:>10}{tgl:>13}")

def main_game(level: str) -> int:
    """Satu sesi permainan. Mengembalikan skor."""
    mn, mx, percobaan_maks, bobot = konfigurasi_level(level)
    rahasia = random.randint(mn, mx)
    print(f"\nTebak angka antara {mn}-{mx}, max {percobaan_maks}
percobaan.")

    for percobaan in range(1, percobaan_maks + 1):
        try:
            tebak = int(input(f"Tebakan #{percobaan}: "))
        except ValueError:
            print("  > Harus angka.")
            continue

        if tebak == rahasia:
            skor = hitung_skor(percobaan, percobaan_maks, bobot)
            print(f"✓ BENAR! Skor: {skor}")
            return skor
        elif tebak < rahasia:
            print("  Terlalu kecil.")
        else:
            print("  Terlalu besar.")
```

```
    print(f"✗ Habis percobaan. Angka rahasia: {rahasia}")
    return 0

def main():
    board = muat_leaderboard()
    while True:
        print("\n=== TEBAK ANGKA ===")
        print("1. Main  2. Leaderboard  3. Keluar")
        pilih = input("Pilih: ").strip()

        if pilih == "1":
            level = input("Level (mudah/sedang/sulit):
").strip().lower()
            if level not in ("mudah", "sedang", "sulit"):
                level = "sedang"
            skor = main_game(level)
            if skor > 0:
                nama = input("Nama Anda: ").strip()[:14]
                tgl = datetime.now().strftime("%Y-%m-%d")
                board.append((nama, skor, level, tgl))
                simpan_leaderboard(board)
        elif pilih == "2":
            tampil_leaderboard(board)
        elif pilih == "3":
            break

if __name__ == "__main__":
    main()
```

14.3.6 Dokumentasi Proyek: README dan Komentar

Proyek tanpa dokumentasi adalah proyek yang akan *terlupakan*—bahkan oleh pembuatnya sendiri. Setiap proyek profesional memiliki minimal satu berkas dokumentasi: README.md. Berikut template yang dapat Anda pakai:

```
# Nama Proyek

Deskripsi singkat (1-2 kalimat) tentang apa yang dilakukan
program.
```

```
## Fitur
- Fitur 1
- Fitur 2
- Fitur 3

## Persyaratan
- Python 3.10 atau lebih baru
- (jika ada pustaka pihak ketiga, daftarkan di sini)

## Cara Penggunaan
```
python nama_program.py
```

## Struktur File
- nama_program.py - program utama
- data.csv - file penyimpanan data (otomatis dibuat)

## Contoh Output
```
(salin tangkapan output di sini)
```

## Penulis
Yusri Ikhwani - Mahasiswa Informatika

## Lisensi
Tugas mata kuliah Algoritma dan Pemrograman 1
```

Praktik komentar di kode:

- **Setiap fungsi memiliki docstring.** Wajib menjelaskan tujuan, parameter, dan return value.
- **Bagian kompleks diberi komentar 1-2 baris.** Khusus jika algoritma tidak intuitif.
- **TODO untuk fitur yang belum selesai.** Pola: # TODO: implementasi validasi tanggal.
- **Hindari komentar yang menjelaskan APA.** Fokus pada MENGAPA. Komentar baik: # urut menurun karena prioritas tertinggi di atas.

14.3.7 Roadmap ke Mata Kuliah Algoritma dan Pemrograman 2

Selamat—Anda telah menyelesaikan separuh perjalanan! Mata kuliah Algoritma dan Pemrograman 2 akan membawa Anda lebih jauh dengan topik-topik berikut:

Topik	Apa yang akan Anda pelajari
Analisis Algoritma	Big-O notation: cara mengukur efisiensi algoritma secara matematis.
Rekursi	Fungsi yang memanggil dirinya sendiri—elegan tetapi penuh jebakan.
Searching & Sorting	Algoritma fundamental seperti binary search, merge sort, quick sort.
OOP	Pemrograman Berorientasi Objek: class, inheritance, polymorphism, encapsulation.
Struktur Data Lanjut	Stack, Queue, Linked List, Tree, Graph—semua dari nol.
Testing	Unit testing dengan pytest. Test-driven development.
NumPy & Pandas	Pustaka komputasi numerik dan analisis data—pintu masuk ke data science.

WAWASAN INDUSTRI 14.1

Apa yang akan Anda pelajari di Algoritma dan Pemrograman 2 adalah materi yang langsung relevan dengan dunia kerja. Big-O notation dan OOP adalah dua topik yang HAMPIR PASTI muncul di wawancara kerja perusahaan teknologi besar. NumPy dan Pandas adalah "batu loncatan" ke data science dan machine learning, dua bidang dengan pertumbuhan tercepat di industri. Jangan tunggu mata kuliah dimulai—eksplorasi mandiri di liburan akan memberi Anda keunggulan signifikan.

14.4 Kesalahan Umum dalam Proyek Mini

Kesalahan 1: Langsung coding tanpa spesifikasi tertulis.

Miskonsepsi: "Saya tahu mau bikin apa, langsung saja kerjakan."

Realita: Tanpa spesifikasi tertulis, fitur sering bertambah/berkurang di tengah jalan, struktur kode berantakan, dan deadline molor. Investasikan 30 menit untuk spesifikasi sebelum sentuh kode—akan menghemat berjam-jam debugging.

Kesalahan 2: Menulis semua kode sekaligus.

Miskonsepsi: "Kalau saya ketik semuanya dulu baru jalankan, lebih cepat."

Realita: Sebaliknya. Implementasi inkremental (satu fitur jadi, uji, lanjut) mendeteksi bug lebih awal dan jauh lebih mudah diperbaiki. Bug yang muncul setelah 500 baris kode jauh lebih sulit daripada bug setelah 50 baris.

Kesalahan 3: Tidak menggunakan version control.

Miskonsepsi: "Saya hanya simpan satu salinan kode di laptop."

Realita: Kehilangan kode karena hard drive rusak atau file ter-overwrite adalah bencana umum. Gunakan git (dengan GitHub) sejak awal. Bahkan untuk tugas kuliah, version control memungkinkan Anda kembali ke versi sebelumnya jika perubahan baru ternyata salah.

Kesalahan 4: Lupa exception handling pada file I/O.

Miskonsepsi: "File pasti adalah saat saya jalankan; tidak perlu try-except."

Realita: Pengguna akan menjalankan program Anda di komputer yang berbeda, dengan kondisi yang berbeda. File mungkin hilang, izin tidak cukup, atau format korup. SELALU bungkus operasi file dengan try-except yang sesuai.

Kesalahan 5: Tidak menulis README.

Miskonsepsi: "Kode saya sudah cukup jelas, tidak perlu README."

Realita: README adalah hal pertama yang dibaca dosen, atasan, atau kolaborator masa depan. Tanpa README, mereka tidak tahu cara menjalankan program, fitur apa yang ada, atau apa yang sebenarnya program lakukan. README profesional adalah pembeda antara 'kode mahasiswa' dan 'kode siap pakai'.

14.5 Rangkuman dan Refleksi Semester

Bab 14 adalah penutup, sekaligus penyatuan, dari semua materi semester pertama Algoritma dan Pemrograman. Empat proyek mini (kalkulator dengan riwayat, manajemen mahasiswa CRUD, pencatat pengeluaran, dan tebak angka dengan leaderboard) mendemonstrasikan bagaimana konsep-konsep yang dipelajari secara terpisah—variabel, percabangan, perulangan, struktur data, fungsi, modul, file I/O, exception handling—disatukan menjadi aplikasi yang berguna.

Metodologi pengerjaan proyek mencakup enam tahap: spesifikasi, perancangan struktur data, perancangan modul, implementasi inkremental, testing, dan dokumentasi. Setiap tahap memiliki output yang spesifik dan terukur. Dokumentasi proyek dengan README dan komentar profesional adalah pembeda penting antara prototype dan aplikasi siap pakai.

Yang telah Anda capai dalam satu semester:

- Memahami konsep algoritma dan representasinya
- Menguasai sintaks Python untuk pemrograman prosedural lengkap
- Mampu menulis program yang membaca, memproses, dan menyimpan data
- Membangun aplikasi multi-modul dengan struktur folder profesional
- Menerapkan praktik baik: PEP 8, docstring, type hint, exception handling

Yang akan dieksplorasi di semester berikutnya: rekursi, kompleksitas algoritma, struktur data lanjut, OOP, dan komputasi ilmiah dengan NumPy/Pandas. Bersiaplah untuk perjalanan yang lebih dalam.

14.6 Latihan Terbimbing

Lima soal berikut adalah analisis dan modifikasi pada keempat proyek mini, untuk memperdalam pemahaman Anda.

Soal 14.6.1

Soal: Pada Proyek A (Kalkulator), simpan_riwayat() ditulis dengan mode 'a'. Apa konsekuensinya, dan bagaimana memperbaikinya jika kita ingin file riwayat hanya berisi sesi terbaru?

Pembahasan: Dengan mode 'a', riwayat semua sesi akan terakumulasi tanpa pernah dihapus, sehingga file dapat tumbuh besar. Untuk hanya menyimpan sesi terbaru, gunakan mode 'w' yang menggantikan isi. Untuk skenario produksi yang lebih bagus, simpan riwayat dengan batas (misalnya 100 baris terakhir) atau pisahkan per tanggal.

Soal 14.6.2

Soal: Pada Proyek B (Manajemen Mahasiswa), fungsi muat_data() menggunakan split(',') sederhana. Apa risiko jika nama mahasiswa mengandung koma (misalnya 'Smith, John')?

Pembahasan: Split sederhana akan memecah nama menjadi dua bagian yang salah, menggeser kolom berikutnya. Hasilnya: data parsing rusak. Solusi profesional: gunakan modul csv bawaan Python yang menangani escaping dan kutip dengan benar. Atau, pilih pemisah yang tidak mungkin muncul di data (seperti '|' atau ';').

Soal 14.6.3

Soal: Pada Proyek C (Pengeluaran), bagaimana cara menambahkan fitur 'edit pengeluaran' tanpa merusak struktur yang ada?

Pembahasan:

Langkah-langkah refactoring:

1. Tambah ID unik (atau timestamp string) ke setiap entri agar dapat di-referensi.

2. Buat fungsi cari_indeks_pengeluaran(data, id) seperti pada Proyek B.

3. Buat fungsi edit(data) yang: tampilkan beberapa entri terakhir, minta ID, minta field yang diedit, validasi input baru, update, panggil simpan_data().

4. Tambahkan ke menu utama sebagai opsi baru.

Soal 14.6.4

Soal: Pada Proyek D (Tebak Angka), bagaimana cara menambahkan fitur 'hari kemarin terbaik' yang menampilkan skor tertinggi pada hari sebelumnya?

Pembahasan:

Strategi: filter leaderboard berdasarkan tanggal kemarin.

1. import datetime untuk dapat date.today() dan timedelta.

2. kemarin = (date.today() - timedelta(days=1)).strftime('%Y-%m-%d').

3. Filter dengan list comprehension: [e for e in board if e[3] == kemarin].

4. Jika ada, tampilkan yang tertinggi (max dengan key=lambda).

5. Tambahkan ke menu sebagai opsi baru.

Soal 14.6.5

Soal: Sebutkan minimal 5 perbaikan/penambahan yang akan Anda lakukan pada salah satu proyek mini jika dilanjutkan sebagai proyek serius.

Pembahasan:

Saran perbaikan untuk Proyek B (Manajemen Mahasiswa):

1. Gunakan modul csv resmi Python untuk parsing aman.

2. Tambahkan validasi NIM (format dan panjang).

3. Backup otomatis ke .bak sebelum overwrite.

4. Tambahkan fitur cari (substring nama/jurusan).

5. Tambahkan unit tests dengan pytest.

6. Pisahkan logika ke modul (models/, utils/) seperti Bab 12.

7. Tambahkan type hint dan docstring lengkap.

8. Logging aktivitas penting (tambah, hapus, update) ke file log.

14.7 Latihan Mandiri (Persiapan UAS)

Sepuluh soal berikut adalah simulasi soal UAS yang menguji integrasi semua materi semester.

- **(C3)** Modifikasi Proyek A: tambahkan operasi pangkat (**) dan akar kuadrat. Sertakan exception handling untuk akar bilangan negatif.
- **(C3)** Modifikasi Proyek B: tambahkan fitur cari mahasiswa berdasarkan substring nama (case-insensitive).
- **(C3)** Modifikasi Proyek C: tambahkan fitur ekspor laporan ke file .txt yang dapat dicetak.
- **(C3)** Modifikasi Proyek D: tambahkan fitur 'tutorial' yang menjelaskan aturan permainan kepada pemain baru.
- **(C4)** Pilih SATU proyek mini, lalu lakukan refaktorisasi penuh: pecah ke beberapa modul (minimal main.py + 2 modul lain), tambahkan type hint di semua fungsi, dan tulis README.
- **(C4)** Rancang proyek mini ke-5 sendiri: aplikasi manajemen perpustakaan mini. Tuliskan: (a) spesifikasi 1 paragraf, (b) struktur data yang dipakai, (c) daftar minimal 5 fungsi, (d) alur menu utama.
- **(C4)** Bandingkan keempat proyek dari sisi: kompleksitas data, jumlah fitur, jenis exception yang ditangani, dan tingkat kesulitan. Buat tabel perbandingan.
- **(C4)** Identifikasi 3 fitur Python yang TIDAK dipakai di proyek-proyek mini tetapi dapat memperbaikinya (misalnya dictionary comprehension untuk laporan, match-case untuk menu, dll). Untuk masing-masing, beri contoh penerapan konkret.
- **(C4)** Studi kasus: dosen meminta proyek akhir 'aplikasi yang membantu mahasiswa lainnya'. Pilih topik (kalkulator IPK, pencatat tugas, dll), tulis spesifikasi lengkap dan rancangan struktur file.
- **(C4)** Refleksi diri: dari 13 bab yang telah dipelajari, sebutkan: (a) bab yang paling mudah, mengapa? (b) bab yang paling sulit, mengapa? (c) bab yang paling berguna untuk karier Anda, mengapa?

14.8 Tugas Proyek Akhir Mata Kuliah

"Aplikasi Terintegrasi Pilihan Anda"

Sebagai proyek penutup mata kuliah, kerjakan SATU proyek mini berskala lebih besar dengan tema bebas. Persyaratan minimum:

- **Domain.** Pilih satu topik yang Anda minati: pendidikan, keuangan, kesehatan, hiburan, sosial, atau lainnya. Pastikan dapat dikerjakan oleh satu orang dalam 1-2 minggu.

- **Fitur.** Minimal 6 fitur dengan menu interaktif. Salah satunya wajib: pencarian/filter, dan satu lagi: laporan/statistik.
- **Struktur.** Multi-modul: main.py + minimal 2 modul terpisah di folder utils/ atau setara. Sertakan `__init__.py` jika menggunakan paket.
- **Data persistence.** Data disimpan ke file (CSV, txt, atau JSON) dan dimuat ulang saat program dimulai.
- **Exception handling.** Tangani minimal 4 jenis eksepsi berbeda yang relevan dengan domain Anda.
- **Praktik baik.** Semua fungsi memiliki docstring dan type hint. Konvensi penamaan PEP 8.
- **Dokumentasi.** README.md lengkap dengan: deskripsi, fitur, cara pakai, struktur file, dan contoh output.
- **Demo.** Rekam video demo singkat (3-5 menit) yang menunjukkan semua fitur berjalan. Boleh upload ke YouTube unlisted atau Google Drive.

Format pengumpulan: berkas ZIP berisi seluruh struktur folder proyek beserta README.md, link video demo, dan laporan PDF 3-5 halaman yang berisi: (a) spesifikasi proyek, (b) struktur data dan modul yang dipakai, (c) tantangan yang dihadapi dan solusinya, (d) refleksi pembelajaran dari proyek ini.

> **CATATAN 14.1**
>
> Proyek akhir ini bukan hanya tugas akademik—ia akan menjadi salah satu portofolio pertama Anda sebagai programmer. Beberapa mahasiswa terdahulu meletakkan proyek serupa di GitHub mereka, mencantumkannya di CV, dan mendapat magang di perusahaan teknologi berkat itu. Investasikan effort yang baik—ini adalah jejak digital pertama Anda di komunitas programmer.

14.9 Refleksi Akhir Semester

Centang pernyataan berikut yang sudah Anda kuasai sepanjang semester:

☐ Saya dapat merancang algoritma untuk masalah baru sebelum menulis kode.

☐ Saya menulis kode Python yang mengikuti konvensi PEP 8.

☐ Saya menguasai struktur kontrol (percabangan dan perulangan).

☐ Saya dapat memilih struktur data yang tepat untuk masalah yang dihadapi.

☐ Saya menulis fungsi modular dengan docstring dan type hint.

☐ Saya dapat mengorganisasi proyek besar dengan multi-modul.

☐ Saya menerapkan file I/O dan exception handling untuk aplikasi tangguh.

☐ Saya tahu cara menulis README dan dokumentasi kode.

☐ Saya menyelesaikan minimal 3 proyek mini selama semester.

☐ Saya siap melanjutkan ke Algoritma dan Pemrograman 2.

ASAH OTAK 14.2

Pertanyaan penutup: 14 minggu yang lalu, sebelum membuka buku ini, apa yang Anda kira programmer kerjakan? Bagaimana persepsi itu berubah setelah satu semester? Apakah Anda merasa lebih, sama, atau kurang tertarik pada bidang ini dibanding sebelumnya? Tulis refleksi Anda, ini akan menjadi catatan berharga yang Anda baca kembali bertahun-tahun ke depan, ketika Anda telah menjadi programmer yang jauh lebih mahir dari hari ini.

Pesan Penutup dari Penulis

Halaman ini adalah halaman terakhir dari buku ini, tetapi bukan halaman terakhir dari perjalanan pemrograman Anda, ia justru adalah halaman pertama. Apa yang kita pelajari bersama selama 14 minggu hanyalah fondasi. Di atasnya akan dibangun struktur yang jauh lebih tinggi: pemrograman berorientasi objek, struktur data lanjut, algoritma kompleks, framework web, kecerdasan buatan, dan banyak lagi.

Yang lebih penting daripada konten teknis adalah **cara berpikir** yang telah Anda kembangkan: kemampuan memecah masalah, ketelitian terhadap detail, kesabaran menghadapi error, kebiasaan mendokumentasikan pekerjaan, dan rasa ingin tahu yang menggerakkan eksplorasi. Keterampilan-keterampilan ini berlaku bukan hanya untuk pemrograman, tetapi untuk hampir setiap bidang dalam hidup.

Terima kasih telah menempuh perjalanan ini bersama. Saya berharap Anda terus bertumbuh sebagai programmer, sebagai akademisi, dan terutama sebagai manusia. Selamat melanjutkan ke bab-bab berikutnya dalam hidup Anda.

— Dr. Yusri Ikhwani, S.Kom., M.Kom.

DAFTAR PUSTAKA

Daftar pustaka berikut memuat rujukan utama yang menjadi sandaran penyusunan buku ajar ini, baik berupa buku teks klasik dan kontemporer, dokumen standar internasional, spesifikasi bahasa pemrograman Python (PEP), publikasi ilmiah, maupun sumber daring resmi. Rujukan diurutkan berdasarkan abjad nama penulis pertama, dan dikelompokkan menurut jenis sumber untuk memudahkan penelusuran.

A. Buku Teks dan Referensi Akademik

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2022). Introduction to Algorithms (4th ed.). Cambridge, MA: The MIT Press.

Downey, A. B. (2024). Think Python: How to Think Like a Computer Scientist (3rd ed.). Needham, MA: Green Tea Press.

Hunt, A., & Thomas, D. (2019). The Pragmatic Programmer: Your Journey to Mastery (20th Anniversary ed.). Boston, MA: Addison-Wesley Professional.

Knuth, D. E. (1997). The Art of Computer Programming, Volume 1: Fundamental Algorithms (3rd ed.). Boston, MA: Addison-Wesley.

Knuth, D. E. (1998). The Art of Computer Programming, Volume 3: Sorting and Searching (2nd ed.). Boston, MA: Addison-Wesley.

Liang, Y. D. (2022). Introduction to Programming Using Python: An Inter-Disciplinary Approach (2nd ed.). Boston, MA: Pearson.

Liem, I. (2007). Diktat Dasar Pemrograman. Bandung: Penerbit ITB.

Lutz, M. (2013). Learning Python (5th ed.). Sebastopol, CA: O'Reilly Media.

Matthes, E. (2023). Python Crash Course: A Hands-On, Project-Based Introduction to Programming (3rd ed.). San Francisco, CA: No Starch Press.

Munir, R. (2021). Algoritma dan Pemrograman dalam Bahasa Pascal, C, dan C++ (Edisi 6). Bandung: Penerbit Informatika.

Polya, G. (1945). How to Solve It: A New Aspect of Mathematical Method. Princeton, NJ: Princeton University Press.

Sedgewick, R., & Wayne, K. (2016). Computer Science: An Interdisciplinary Approach. Boston, MA: Addison-Wesley Professional.

Wirth, N. (1976). Algorithms + Data Structures = Programs. Englewood Cliffs, NJ: Prentice-Hall.

Zelle, J. M. (2017). Python Programming: An Introduction to Computer Science (3rd ed.). Wilsonville, OR: Franklin, Beedle & Associates.

B. Jurnal dan Prosiding Ilmiah

Gilbreth, F. B., & Gilbreth, L. M. (1921). Process Charts: First Steps in Finding the One Best Way to Do Work. In Transactions of the American Society of Mechanical Engineers (Vol. 43, pp. 1029–1050). New York: ASME.

Goldstine, H. H., & von Neumann, J. (1947). Planning and Coding of Problems for an Electronic Computing Instrument. Princeton, NJ: Institute for Advanced Study, Princeton University.

Hill, R. K. (2016). What an Algorithm Is. Philosophy & Technology, 29(1), 35–59. https://doi.org/10.1007/s13347-014-0184-5

McCabe, T. J. (1976). A Complexity Measure. IEEE Transactions on Software Engineering, SE-2(4), 308–320. https://doi.org/10.1109/TSE.1976.233837

Wing, J. M. (2006). Computational Thinking. Communications of the ACM, 49(3), 33–35. https://doi.org/10.1145/1118178.1118215

C. Dokumen Standar dan Spesifikasi

American National Standards Institute (1970). ANSI X3.5-1970: Flowchart Symbols and Their Usage in Information Processing. New York: ANSI.

International Organization for Standardization (1985). ISO 5807:1985: Information Processing — Documentation Symbols and Conventions for Data, Program and System Flowcharts, Program Network Charts and System Resources Charts. Geneva: ISO.

Peters, T. (2004). PEP 20 — The Zen of Python. Python Software Foundation. Diakses dari https://peps.python.org/pep-0020/

Python Software Foundation (2024). The Python Language Reference. Diakses dari https://docs.python.org/3/reference/

Python Software Foundation (2024). The Python Standard Library. Diakses dari https://docs.python.org/3/library/

van Rossum, G., & Coghlan, N. (2013). PEP 257 — Docstring Conventions. Python Software Foundation. Diakses dari https://peps.python.org/pep-0257/

van Rossum, G., Lehtosalo, J., & Langa, L. (2014). PEP 484 — Type Hints. Python Software Foundation. Diakses dari https://peps.python.org/pep-0484/

van Rossum, G., Warsaw, B., & Coghlan, N. (2001). PEP 8 — Style Guide for Python Code. Python Software Foundation. Diakses dari https://peps.python.org/pep-0008/

D. Sumber Daring dan Kode Etik Profesional

Association for Computing Machinery (2018). ACM Code of Ethics and Professional Conduct. New York: ACM. Diakses dari https://www.acm.org/code-of-ethics

IEEE Computer Society & ACM Joint Task Force (2014). Software Engineering Code of Ethics and Professional Practice. Piscataway, NJ: IEEE. Diakses dari https://www.computer.org/education/code-of-ethics

Stack Overflow (2024). Stack Overflow Annual Developer Survey. Diakses dari https://survey.stackoverflow.co/

TIOBE Software BV (2024). TIOBE Programming Community Index. Diakses dari https://www.tiobe.com/tiobe-index/

Visual Studio Code Documentation (2024). Microsoft Corporation. Diakses dari https://code.visualstudio.com/docs

E. Karya Terkait yang Berpengaruh dalam Pengembangan Buku Ini

Beazley, D. M., & Jones, B. K. (2013). Python Cookbook (3rd ed.). Sebastopol, CA: O'Reilly Media.

Brown, P. C., Roediger III, H. L., & McDaniel, M. A. (2014). Make It Stick: The Science of Successful Learning. Cambridge, MA: Belknap Press of Harvard University Press.

Hofstadter, D. R. (1979). Gödel, Escher, Bach: An Eternal Golden Braid. New York: Basic Books.

McConnell, S. (2004). Code Complete: A Practical Handbook of Software Construction (2nd ed.). Redmond, WA: Microsoft Press.

Papert, S. (1980). Mindstorms: Children, Computers, and Powerful Ideas. New York: Basic Books.

Petzold, C. (1999). Code: The Hidden Language of Computer Hardware and Software. Redmond, WA: Microsoft Press.

Sipser, M. (2012). Introduction to the Theory of Computation (3rd ed.). Boston, MA: Cengage Learning.

Skiena, S. S. (2020). The Algorithm Design Manual (3rd ed.). London: Springer.

LAMPIRAN

RENCANA PEMBELAJARAN SEMESTER (RPS)

	UNIVERSITAS ISLAM KALIMANTAN MUHAMMAD ARSYAD AL BANJARI BANJARMASIN **FAKULTAS TEKNOLOGI INFORMASI** **PROGRAM STUDI TEKNIK INFORMATIKA (S1)** **RENCANA PEMBELAJARAN SEMESTER (RPS)**

RENCANA PEMBELAJARAN SEMESTER — TIF1103: Algoritma dan Pemrograman 1

Nama Mata Kuliah	Algoritma dan Pemrograman 1	**Kode Mata Kuliah**	TIF1103
Rumpun Mata Kuliah	Mata Kuliah Keahlian Berkarya (MKB) — Inti Keilmuan	**Bobot (SKS)**	3 SKS (T-P: 2-1)
Semester	I (Satu) / Ganjil	**Kurikulum**	Kurikulum OBE 2024 — KKNI Level 6
Nama (English)	*Algorithms and Programming 1*	**Tanggal Revisi**	TA 2026/2027
Mata Kuliah Syarat	–	**Program Studi**	Teknik Informatika (S1)

Pengembang RPS	Koordinator RMK / KBK	Ketua Program Studi
Dr. Yusri Ikhwani, S.Kom., M.Kom. (Dosen Pengembang)	Dr. Yusri Ikhwani, S.Kom., M.Kom. (Koordinator RMK)	Dr. Wagino, S.Kom., M.Kom. (Ketua Prodi Teknik Informatika)

A. Deskripsi Mata Kuliah

Mata kuliah ini merupakan fondasi keterampilan pemrograman yang membekali mahasiswa dengan pola pikir komputasional dan kemampuan menyusun algoritma serta mengimplementasikannya menggunakan bahasa pemrograman Python (versi 3.10 atau lebih baru). Cakupan materi meliputi konsep algoritma dan representasinya (flowchart dan pseudocode), lingkungan pengembangan Python, variabel dan tipe data, operator, input/output dan operasi string, struktur kontrol percabangan dan perulangan, struktur data koleksi (list, tuple, dictionary, set), fungsi dan modularisasi, penanganan berkas (file I/O), penanganan eksepsi, serta integrasi seluruh konsep melalui proyek mini. Pembelajaran menekankan praktik langsung dan penerapan praktik baik (PEP 8, docstring, type hint) sejak dini.

This course is the foundation of programming skills, equipping students with computational thinking and the ability to design algorithms and implement them using the Python programming language (version 3.10 or later). Topics include algorithm concepts and their representation (flowchart and pseudocode), the Python development environment, variables and data types, operators, input/output and string operations, selection and iteration control structures, collection data structures (list, tuple, dictionary, set), functions and modularisation, file I/O, exception handling, and the integration of all concepts through a mini-project.

Learning emphasises hands-on practice and the early adoption of best practices (PEP 8, docstrings, type hints).

B. Capaian Pembelajaran

Capaian Pembelajaran Lulusan (CPL) yang Dibebankan pada Mata Kuliah

Kode	Capaian Pembelajaran Lulusan (CPL) yang Dibebankan
S9	Menunjukkan sikap bertanggung jawab atas pekerjaan di bidang keahliannya secara mandiri serta menjunjung etika dan integritas akademik.
P3	Menguasai konsep teoritis algoritma, struktur data, dan pemrograman sebagai dasar pengembangan perangkat lunak.
KU1	Mampu menerapkan pemikiran logis, kritis, sistematis, dan inovatif dalam penyelesaian masalah menggunakan pendekatan komputasional.
KU5	Mampu mengambil keputusan secara tepat dalam konteks penyelesaian masalah berdasarkan analisis logis dan pengujian.
KK2	Mampu merancang, mengimplementasikan, dan menguji program komputer untuk menyelesaikan permasalahan menggunakan bahasa pemrograman prosedural.

Capaian Pembelajaran Mata Kuliah (CPMK)

Kode CPMK	CPL	Deskripsi CPMK
CPMK-1	P3, KU1	Mampu menjelaskan konsep algoritma, merepresentasikannya dalam flowchart dan pseudocode, serta menyiapkan dan menggunakan lingkungan pemrograman Python.
CPMK-2	P3, KK2	Mampu menerapkan variabel, tipe data, operator, operasi input/output, dan operasi string dalam program Python.
CPMK-3	KU1, KK2	Mampu merancang dan mengimplementasikan struktur kontrol percabangan dan perulangan untuk menyelesaikan masalah.
CPMK-4	P3, KK2	Mampu memanfaatkan struktur data koleksi (list, tuple, dictionary, set) untuk mengelola data.
CPMK-5	KU1, KK2	Mampu merancang fungsi, menerapkan modularisasi, dan mengelola scope variabel untuk menyusun kode yang terstruktur dan dapat digunakan kembali.
CPMK-6	S9, KU5, KK2	Mampu menerapkan penanganan berkas dan eksepsi, serta membangun aplikasi sederhana melalui proyek mini secara bertanggung jawab.

C. Pemetaan CPMK terhadap Sub-CPMK

CPMK	Sub-CPMK	Deskripsi Sub-CPMK (Kemampuan Akhir Tiap Tahapan Belajar)
CPMK-1	Sub-CPMK 1.1	Menjelaskan definisi, karakteristik, dan siklus pengembangan algoritma serta program.
CPMK-1	Sub-CPMK 1.2	Merepresentasikan algoritma dalam bentuk flowchart (ANSI/ISO 5807) dan pseudocode.
CPMK-1	Sub-CPMK 1.3	Menyiapkan lingkungan Python dan menjalankan program pertama (mode REPL dan skrip).
CPMK-2	Sub-CPMK 2.1	Menggunakan variabel, tipe data numerik/boolean/string, operator, dan konversi tipe.
CPMK-2	Sub-CPMK 2.2	Menerapkan input/output, formatting string, dan metode string.

CPMK	Sub-CPMK	Deskripsi Sub-CPMK (Kemampuan Akhir Tiap Tahapan Belajar)
CPMK-3	Sub-CPMK 3.1	Menerapkan percabangan if, if-else, if-elif-else, ekspresi ternary, dan match-case.
CPMK-3	Sub-CPMK 3.2	Menerapkan perulangan for dan while beserta break, continue, dan else pada loop.
CPMK-4	Sub-CPMK 4.1	Mengoperasikan list dan tuple, termasuk list comprehension dan list bersarang.
CPMK-4	Sub-CPMK 4.2	Mengoperasikan dictionary dan set beserta operasi himpunannya.
CPMK-5	Sub-CPMK 5.1	Mendefinisikan fungsi, parameter, return value, *args/**kwargs, dan lambda.
CPMK-5	Sub-CPMK 5.2	Menerapkan aturan scope (LEGB), modul, paket, dan struktur proyek Python.
CPMK-6	Sub-CPMK 6.1	Menerapkan file I/O dengan context manager dan penanganan eksepsi try-except.
CPMK-6	Sub-CPMK 6.2	Membangun, menguji, dan mendokumentasikan aplikasi melalui proyek mini.

D. Bobot Penilaian per CPMK (%)

CPMK	Partisipasi & Kuis (10%)	Tugas & Praktikum (25%)	Proyek Mini (15%)	UTS (20%)	UAS (30%)	Total
CPMK-1	2	5	–	5	–	**12**
CPMK-2	2	4	–	5	–	**11**
CPMK-3	2	4	–	10	–	**16**
CPMK-4	1	5	–	–	8	**14**
CPMK-5	1	4	–	–	10	**15**
CPMK-6	2	3	15	–	12	**32**
Total	**10**	**25**	**15**	**20**	**30**	**100**

Nilai akhir mata kuliah merupakan akumulasi skor seluruh CPMK sesuai bobotnya (total 100%).

E. Pustaka, Media, dan Ketentuan Kelulusan

Pustaka Utama	[1] Ikhwani, Y. (2026). Buku Ajar Algoritma dan Pemrograman 1 (Edisi Pertama, berbasis Python). Banjarmasin: FTI UNISKA MAB. [2] Ikhwani, Y. (2026). Buku Praktik Algoritma dan Pemrograman 1. Banjarmasin: FTI UNISKA MAB.
Pustaka Pendukung	[3] Matthes, E. (2023). Python Crash Course (3rd ed.). No Starch Press. [4] Downey, A. B. (2015). Think Python: How to Think Like a Computer Scientist (2nd ed.). O'Reilly. [5] Lutz, M. (2013). Learning Python (5th ed.). O'Reilly. [6] Dokumentasi resmi Python 3 (docs.python.org) serta PEP 8, PEP 257, dan PEP 484.
Perangkat Lunak (Software)	Python 3.10+; Visual Studio Code + ekstensi Python (alternatif: PyCharm/IDLE); Git.
Perangkat Keras (Hardware)	Komputer/laptop; laboratorium komputer; projector; akses internet.

Dosen Pengampu	Dr. Yusri Ikhwani, S.Kom., M.Kom. (Koordinator) Putra, S.Kom., M.Kom. Hafidh, S.Kom., M.Kom.
Mata Kuliah Syarat	–
Ambang Batas Kelulusan per CPMK	Skor minimal 55,01 (skala 0–100) untuk setiap CPMK.
Ambang Batas Kelulusan Mata Kuliah	85,00% mahasiswa mencapai ambang batas seluruh CPMK
Konversi Nilai	A 80–100 | B+ 75–79 | B 70–74 | C+ 65–69 | C 60–64 | D 45–59 | E < 45

F. Rencana Pembelajaran Mingguan (16 Minggu)

Mg	Id CPMK	Kemampuan Akhir (Sub-CPMK)	Indikator	Materi / Bahan Kajian	Metode Pembelajaran	Bentuk Penilaian	Bobot (%)
1	CPMK-1	Pengantar Algoritma & Pemrograman	Menjelaskan definisi, karakteristik algoritma, dan hubungan algoritma–program–bahasa.	Definisi algoritma (Al-Khwarizmi–era digital) Karakteristik algoritma yang baik Paradigma & siklus pengembangan program Etika & HKI dalam pemrograman	Kuliah interaktif, diskusi.	Partisipasi & Kuis	–
2	CPMK-1	Representasi Algoritma	Merepresentasikan algoritma dengan flowchart dan pseudocode serta melakukan desk checking.	Notasi algoritmik & abstraksi Flowchart (simbol ANSI/ISO 5807) Pseudocode gaya Wirth Konversi antar representasi & desk checking	Kuliah, latihan terbimbing.	Tugas & Praktikum	–
3	CPMK-1	Pengenalan Python & Lingkungan	Menyiapkan lingkungan Python dan menjalankan program pertama.	Sejarah & filosofi Python Instalasi Python & IDE (VS Code) Mode REPL vs skrip Anatomi skrip, komentar, & dokumentasi	Praktikum di laboratorium.	Tugas & Praktikum	–
4	CPMK-2	Variabel, Tipe Data, & Operator	Menerapkan variabel, tipe data, operator, dan konversi tipe.	Model referensi objek & PEP 8 Tipe numerik, boolean, string (pengantar) Operator & prioritasnya Konversi tipe data	Kuliah, praktikum.	Tugas & Praktikum	–
5	CPMK-2	Input/Output & Operasi String	Menerapkan I/O, formatting, dan metode string.	Fungsi print() (sep, end, file) Formatting string (f-string, format, %) Fungsi input() & konversi String sebagai urutan, metode & immutability	Kuliah, praktikum.	Tugas & Praktikum	–

Mg	Id CPMK	Kemampuan Akhir (Sub-CPMK)	Indikator	Materi / Bahan Kajian	Metode Pembelajaran	Bentuk Penilaian	Bobot (%)
6	CPMK-3	Struktur Kontrol Percabangan	Mengimplementasikan percabangan untuk pengambilan keputusan.	if, if-else, if-elif-else Ekspresi ternary match-case (Python 3.10+) Pola & anti-pola percabangan	Kuliah, praktikum.	Tugas & Praktikum	–
7	CPMK-3	Struktur Kontrol Perulangan	Mengimplementasikan perulangan for/while dan kontrol loop.	for dengan range() & iterable while break, continue, else pada loop Tracing eksekusi loop	Kuliah, praktikum.	Tugas & Praktikum	–
8	CPMK-1, CPMK-2, CPMK-3	**UJIAN TENGAH SEMESTER**	Menyelesaikan masalah komprehensif materi minggu 1–7 dengan metode Polya.	Metode Polya (analisis–rancang–implementasi–uji) Bank soal komprehensif materi Bab 1–7	Ujian praktik/tertulis.	Tes (UTS)	**20**
9	CPMK-4	List dan Tuple	Mengoperasikan list dan tuple untuk data terurut.	Konsep koleksi terurut Pembuatan, akses, modifikasi list & metodenya List comprehension & tuple unpacking List bersarang (2D)	Kuliah, praktikum.	Tugas & Praktikum	–
10	CPMK-4	Dictionary dan Set	Mengoperasikan dictionary dan set beserta operasi himpunan.	Pasangan kunci-nilai & tabel hash Metode & iterasi dictionary Dictionary comprehension Set, operasi himpunan, frozenset	Kuliah, praktikum.	Tugas & Praktikum	–
11	CPMK-5	Fungsi dan Parameter	Merancang fungsi dan mengelola parameter.	Modularisasi & prinsip DRY Parameter posisional/keyword & nilai default *args dan **kwargs Lambda & refaktorisasi kode	Kuliah, praktikum.	Tugas & Praktikum	–
12	CPMK-5	Modularisasi dan Scope	Menerapkan scope, modul, dan struktur proyek.	Aturan LEGB, global & nonlocal Modul, paket, & import Modul standar penting Struktur proyek Python sederhana	Kuliah, praktikum.	Tugas & Praktikum	–
13	CPMK-6	File I/O dan Exception Handling	Menerapkan operasi berkas dan penanganan eksepsi.	Konsep berkas (path, encoding, mode) open()/close() & context manager (with) Mode akses berkas Eksepsi & try-except-else-finally	Kuliah, praktikum.	Tugas & Praktikum	–

Mg	Id CPMK	Kemampuan Akhir (Sub-CPMK)	Indikator	Materi / Bahan Kajian	Metode Pembelajaran	Bentuk Penilaian	Bobot (%)
14	CPMK-6	Studi Kasus Mini-Proyek	Merancang dan membangun aplikasi mini yang mengintegrasikan seluruh materi.	Metodologi pengerjaan proyek Proyek: kalkulator riwayat / CRUD mahasiswa / pencatat pengeluaran Dokumentasi (README & komentar)	Praktikum berbasis proyek, bimbingan.	Proyek Mini	**15**
15	CPMK-6	Presentasi & Evaluasi Proyek	Mempresentasikan, menguji, dan mengevaluasi proyek mini.	Presentasi & demo proyek Pengujian & debugging Peer review & perbaikan	Seminar kelas, peer review.	Presentasi	–
16	CPMK-4, CPMK-5, CPMK-6	**UJIAN AKHIR SEMESTER**	Menyelesaikan soal komprehensif materi minggu 9–15 dan menyerahkan proyek final.	Materi Bab 9–14 Portofolio proyek mini	Ujian praktik/tertulis + portofolio.	Tes + Portofolio (UAS)	**30**

Catatan: Mg = Minggu ke-. Bobot 3 SKS (2-1) setara 2 jam tatap muka teori dan 1 sesi praktikum per minggu, ditambah kegiatan terstruktur dan mandiri sesuai SN-Dikti.

G. Rubrik Penilaian CPMK

CPMK-1 — kemampuan menjelaskan konsep algoritma dan menyiapkan lingkungan pemrograman Python

Rentang Skor	Kategori	Deskripsi Kinerja
80 – 100	Istimewa (A)	Mampu menjelaskan konsep algoritma dan menyiapkan lingkungan pemrograman Python secara akurat, mandiri, dan melebihi ekspektasi.
75 – 79	Baik Sekali (B+)	Mampu menjelaskan konsep algoritma dan menyiapkan lingkungan pemrograman Python secara tepat, lengkap, dan sistematis.
70 – 74	Baik (B)	Mampu menjelaskan konsep algoritma dan menyiapkan lingkungan pemrograman Python dengan baik sesuai kriteria yang ditetapkan.
65 – 69	Cukup Baik (C+)	Cukup mampu menjelaskan konsep algoritma dan menyiapkan lingkungan pemrograman Python, namun belum konsisten pada beberapa aspek.
60 – 64	Cukup (C)	Cukup mampu menjelaskan konsep algoritma dan menyiapkan lingkungan pemrograman Python pada tingkat dasar dengan bimbingan.
45 – 59	Kurang (D)	Kurang mampu menjelaskan konsep algoritma dan menyiapkan lingkungan pemrograman Python; banyak aspek belum terpenuhi.
< 45	Sangat Kurang (E)	Tidak mampu menjelaskan konsep algoritma dan menyiapkan lingkungan pemrograman Python.

CPMK-2 — kemampuan menerapkan variabel, tipe data, operator, dan operasi I/O serta string

Rentang Skor	Kategori	Deskripsi Kinerja
80 – 100	Istimewa (A)	Mampu menerapkan variabel, tipe data, operator, dan operasi I/O serta string secara akurat, mandiri, dan melebihi ekspektasi.
75 – 79	Baik Sekali (B+)	Mampu menerapkan variabel, tipe data, operator, dan operasi I/O serta string secara tepat, lengkap, dan sistematis.
70 – 74	Baik (B)	Mampu menerapkan variabel, tipe data, operator, dan operasi I/O serta string dengan baik sesuai kriteria yang ditetapkan.
65 – 69	Cukup Baik (C+)	Cukup mampu menerapkan variabel, tipe data, operator, dan operasi I/O serta string, namun belum konsisten pada beberapa aspek.

Rentang Skor	Kategori	Deskripsi Kinerja
60 – 64	Cukup (C)	Cukup mampu menerapkan variabel, tipe data, operator, dan operasi I/O serta string pada tingkat dasar dengan bimbingan.
45 – 59	Kurang (D)	Kurang mampu menerapkan variabel, tipe data, operator, dan operasi I/O serta string; banyak aspek belum terpenuhi.
< 45	Sangat Kurang (E)	Tidak mampu menerapkan variabel, tipe data, operator, dan operasi I/O serta string.

CPMK-3 — kemampuan merancang dan mengimplementasikan struktur kontrol percabangan dan perulangan

Rentang Skor	Kategori	Deskripsi Kinerja
80 – 100	Istimewa (A)	Mampu merancang dan mengimplementasikan struktur kontrol percabangan dan perulangan secara akurat, mandiri, dan melebihi ekspektasi.
75 – 79	Baik Sekali (B+)	Mampu merancang dan mengimplementasikan struktur kontrol percabangan dan perulangan secara tepat, lengkap, dan sistematis.
70 – 74	Baik (B)	Mampu merancang dan mengimplementasikan struktur kontrol percabangan dan perulangan dengan baik sesuai kriteria yang ditetapkan.
65 – 69	Cukup Baik (C+)	Cukup mampu merancang dan mengimplementasikan struktur kontrol percabangan dan perulangan, namun belum konsisten pada beberapa aspek.
60 – 64	Cukup (C)	Cukup mampu merancang dan mengimplementasikan struktur kontrol percabangan dan perulangan pada tingkat dasar dengan bimbingan.
45 – 59	Kurang (D)	Kurang mampu merancang dan mengimplementasikan struktur kontrol percabangan dan perulangan; banyak aspek belum terpenuhi.
< 45	Sangat Kurang (E)	Tidak mampu merancang dan mengimplementasikan struktur kontrol percabangan dan perulangan.

CPMK-4 — kemampuan memanfaatkan struktur data koleksi untuk mengelola data

Rentang Skor	Kategori	Deskripsi Kinerja
80 – 100	Istimewa (A)	Mampu memanfaatkan struktur data koleksi untuk mengelola data secara akurat, mandiri, dan melebihi ekspektasi.
75 – 79	Baik Sekali (B+)	Mampu memanfaatkan struktur data koleksi untuk mengelola data secara tepat, lengkap, dan sistematis.
70 – 74	Baik (B)	Mampu memanfaatkan struktur data koleksi untuk mengelola data dengan baik sesuai kriteria yang ditetapkan.
65 – 69	Cukup Baik (C+)	Cukup mampu memanfaatkan struktur data koleksi untuk mengelola data, namun belum konsisten pada beberapa aspek.
60 – 64	Cukup (C)	Cukup mampu memanfaatkan struktur data koleksi untuk mengelola data pada tingkat dasar dengan bimbingan.
45 – 59	Kurang (D)	Kurang mampu memanfaatkan struktur data koleksi untuk mengelola data; banyak aspek belum terpenuhi.
< 45	Sangat Kurang (E)	Tidak mampu memanfaatkan struktur data koleksi untuk mengelola data.

CPMK-5 — kemampuan merancang fungsi dan menerapkan modularisasi kode

Rentang Skor	Kategori	Deskripsi Kinerja
80 – 100	Istimewa (A)	Mampu merancang fungsi dan menerapkan modularisasi kode secara akurat, mandiri, dan melebihi ekspektasi.
75 – 79	Baik Sekali (B+)	Mampu merancang fungsi dan menerapkan modularisasi kode secara tepat, lengkap, dan sistematis.
70 – 74	Baik (B)	Mampu merancang fungsi dan menerapkan modularisasi kode dengan baik sesuai kriteria yang ditetapkan.
65 – 69	Cukup Baik (C+)	Cukup mampu merancang fungsi dan menerapkan modularisasi kode, namun belum konsisten pada beberapa aspek.

Rentang Skor	Kategori	Deskripsi Kinerja
60 – 64	Cukup (C)	Cukup mampu merancang fungsi dan menerapkan modularisasi kode pada tingkat dasar dengan bimbingan.
45 – 59	Kurang (D)	Kurang mampu merancang fungsi dan menerapkan modularisasi kode; banyak aspek belum terpenuhi.
< 45	Sangat Kurang (E)	Tidak mampu merancang fungsi dan menerapkan modularisasi kode.

CPMK-6 — kemampuan menerapkan file I/O dan eksepsi serta membangun aplikasi melalui proyek mini

Rentang Skor	Kategori	Deskripsi Kinerja
80 – 100	Istimewa (A)	Mampu menerapkan file I/O dan eksepsi serta membangun aplikasi melalui proyek mini secara akurat, mandiri, dan melebihi ekspektasi.
75 – 79	Baik Sekali (B+)	Mampu menerapkan file I/O dan eksepsi serta membangun aplikasi melalui proyek mini secara tepat, lengkap, dan sistematis.
70 – 74	Baik (B)	Mampu menerapkan file I/O dan eksepsi serta membangun aplikasi melalui proyek mini dengan baik sesuai kriteria yang ditetapkan.
65 – 69	Cukup Baik (C+)	Cukup mampu menerapkan file I/O dan eksepsi serta membangun aplikasi melalui proyek mini, namun belum konsisten pada beberapa aspek.
60 – 64	Cukup (C)	Cukup mampu menerapkan file I/O dan eksepsi serta membangun aplikasi melalui proyek mini pada tingkat dasar dengan bimbingan.
45 – 59	Kurang (D)	Kurang mampu menerapkan file I/O dan eksepsi serta membangun aplikasi melalui proyek mini; banyak aspek belum terpenuhi.
< 45	Sangat Kurang (E)	Tidak mampu menerapkan file I/O dan eksepsi serta membangun aplikasi melalui proyek mini.

Penerbit :
Universitas Islam Kalimantan
Muhammad Arsyad Al -Banjary

Alamat :
Gedung A UPT Publikasi dan Pengelolaan Jurnal
Universitas Islam Kalimantan
Muhammad Arsyad Al-Banjary

Jl. Adhyaksa No. 2 Kayutangi
Banjarmasin, Kalimantan Selatan
Telepon : 0511 - 3304352
FAX : 0511

ISBN 978-623-8189-97-7