MARS

Editor:
Ahmad Fitriansyah

BUKU AJAR

PEMROGRAMAN KOMPUTER: KONSEP, LOGIKA, DAN IMPLEMENTASI

Penulis:
Dina Fitria Murad | Aswan Supriyadi Sunge
Teguh Prasandy | Maryani | Emny Harna Yossy
Agus Putranto | Suzanna | Ahmad Fitriansyah

BUKU AJAR

PEMROGRAMAN KOMPUTER: KONSEP, LOGIKA, DAN IMPLEMENTASI

Penulis:
Dina Fitria Murad; Aswan Supriyadi Sunge; Teguh Prasandy; Maryani; Emny Harna Yossy; Agus Putranto; Suzanna; Ahmad Fitriansyah

Editor:
Ahmad Fitriansyah

PT. Mustika Sri Rosadi

BUKU AJAR PEMROGRAMAN KOMPUTER: KONSEP, LOGIKA, DAN IMPLEMENTASI

Penulis:
Dina Fitria Murad; Aswan Supriyadi Sunge; Teguh Prasandy; Maryani; Emny Harna Yossy; Agus Putranto; Suzanna; Ahmad Fitriansyah.

Editor: Ahmad Fitriansyah
Layout: Tim PT. Mustika Sri Rosadi
Desain Sampul: Febriansyah
ISBN: 978-634-7535-00-9 (PDF)
Cetakan Pertama: 5 Desember 2025

Diterbitkan oleh Penerbit PT Mustika Sri Rosadi
Alamat Penerbit: Citra Indah City, Bukit Heliconia AG 23/32, Kecamatan Jonggol, Kab. Bogor.
Email: mars.mustikasrirosadi@gmail.com

KATA PENGANTAR

Puji syukur kehadirat Allah SWT, atas rahmat dan karunia-Nya, penulis dapat menyelesaikan buku ajar "Pemrograman Komputer: Konsep, Logika, dan Implementasi". Buku ini disusun dengan tujuan untuk menjadi panduan bagi mahasiswa dan pemula yang ingin mempelajari dasar-dasar pemrograman dari nol.

Pembahasan dirancang secara sistematis, dimulai dari pengenalan Konsep Dasar Pemrograman, Tipe Data dan Variabel, Struktur Kontrol Program, Fungsi dan Prosedur, penggunaan Array dan String, serta Algoritma dan Logika Dasar. Sebagai pijakan paradigma pemrograman modern, diperkenalkan Pemrograman Berorientasi Objek (OOP) dan ditutup dengan implementasi menggunakan bahasa Python.

Penulis menyadari sepenuhnya bahwa buku ini masih jauh dari sempurna. Oleh karena itu, kritik dan saran yang membangun dari para pembaca sangat diharapkan untuk perbaikan di masa mendatang. Semoga buku ajar ini dapat bermanfaat dalam menguasai seni pemrograman komputer.

Bogor, 5 Desember 2025
Penulis

DAFTAR ISI

BAB 1. KONSEP DASAR PEMROGRAMAN

A. Pendahuluan

Konsep dasar pemrograman adalah fondasi penting yang harus dipahami oleh siapa saja yang ingin mulai belajar pemrograman. Pemrograman adalah proses memberikan instruksi secara logis kepada komputer agar dapat menjalankan suatu tugas dengan efisien (Azura Team, 2023). Pemrograman adalah proses merancang, menulis, dan menguji instruksi agar komputer dapat menjalankan tugas tertentu menggunakan bahasa pemrograman sebagai media komunikasi antara manusia dan mesin.

Penguasaan konsep dasar pemrograman terbukti meningkatkan kemampuan berpikir komputasional, pemecahan masalah, dan analisis keterampilan penting di era Revolusi Industri 4.0. Beberapa konsep dasar yang menjadi fondasi pemrograman modern meliputi :

1. Variabel dan Tipe Data: Variabel digunakan untuk menyimpan data sementara dalam program. Tipe data menentukan macam data (seperti angka, teks, dan lain-lain) yang bisa disimpan oleh variabel.
2. Struktur Kontrol: Meliputi percabangan (if, else, switch) dan pengulangan (for, while, do-while) yang mengatur alur eksekusi program.

3. Algoritma: Langkah-langkah atau logika penyelesaian masalah yang dirancang secara sistematis dan digunakan dalam program.
4. Struktur Data: Cara mengorganisasi dan menyimpan data supaya efisien, contohnya array, list, queue, dan stack.
5. Fungsi dan Prosedur: Blok kode yang bisa dipanggil berulang kali untuk melakukan tugas tertentu. Fungsi biasanya menghasilkan nilai, prosedur hanya menjalankan aksi.
6. Pemrograman Berorientasi Objek (OOP): Paradigma yang mendefinisikan program sebagai sekumpulan objek dengan atribut dan method. Konsep inti OOP meliputi enkapsulasi, inheritance, dan polymorphism.

Paradigma dan model pemrograman yang ada saat ini terdiri atas:

1. Prosedural: Menyelesaikan tugas berdasarkan urutan langkah-langkah yang terstruktur dan modular.
2. Modular: Memecah program ke dalam modul-modul atau sub-routine khusus untuk tugas tertentu agar kode lebih mudah dikelola.
3. OOP: Menorganisir kode dalam bentuk objek, meningkatkan modularitas, keamanan, dan kemudahan pengembangan.

Pentingnya penguasaan dasar konsep pemrograman akan membantu dalam:

1. Memecahkan masalah secara logis dan sistematis
2. Belajar bahasa pemrograman baru lebih mudah
3. Membuat kode yang efisien dan mudah dipelihara
4. Berkomunikasi baik dengan anggota tim pengembangan

Setiap program pada dasarnya mengikuti alur input–proses–output. Dengan memahami konsep dasar pemrograman, seseorang dapat belajar lebih terarah untuk berbagai aplikasi seperti web, mobile, dan AI. Pemrograman bukan hanya menulis kode, tetapi juga memahami bagaimana komputer mengeksekusi instruksi. Banyak pemula kesulitan karena belum memahami hubungan antara kode sumber dan cara kerja komputer, sehingga mempelajari cara kerja komputer menjadi langkah penting sebelum mendalami bahasa pemrograman lebih lanjut.

Semantik bahasa pemrograman merupakan aspek penting dalam ilmu komputer teoretis karena membantu menjelaskan makna dan perilaku program secara formal. (Hutton, 2023) menegaskan bahwa spesifikasi dan semantik yang jelas sejak awal sangat penting untuk memastikan implementasi bahasa pemrograman yang benar. (Aho & Ullman, 2022) menyoroti bahwa cara memodelkan objek semantik misalnya bagaimana memaknai lokasi memori dapat memengaruhi perilaku semantik suatu bahasa. Selain itu, computational thinking (CT) menjadi keterampilan kunci abad ke-21.

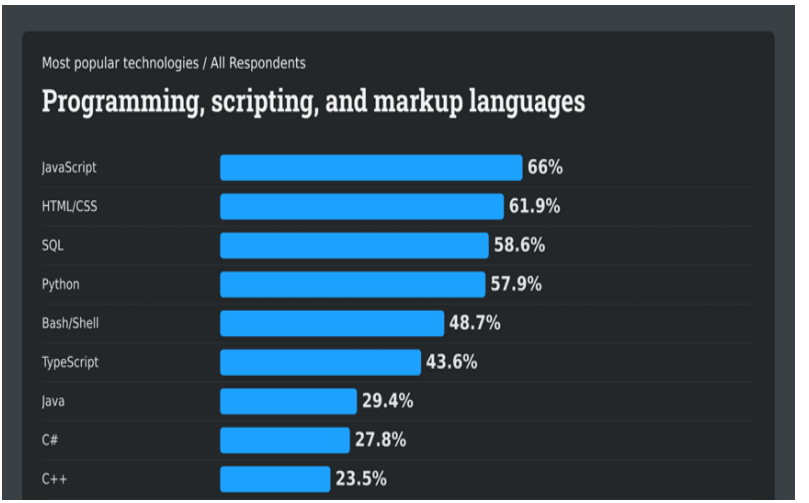
(Kandemir et al., 2020) menekankan bahwa belajar pemrograman tidak cukup dengan memahami sintaksis, tetapi juga harus menguasai kemampuan berpikir komputasional untuk memecahkan masalah secara lebih luas di berbagai bidang.

B. Pengantar Bahasa Pemrograman Populer

Bahasa pemrograman berfungsi sebagai sistem komunikasi khusus yang memungkinkan manusia untuk menginstruksikan komputer secara efektif. (Nagendra Singh et al., 2022) menjelaskan bahwa komputer tidak dapat memahami perintah bahasa manusia normal, sehingga memerlukan pengembangan bahasa pemrograman untuk berkomunikasi dengan komputer dan mengembangkan aplikasi, situs web, dan program perangkat lunak. (Afifa Fathima et al., 2022) mencatat bahwa terdapat lebih dari 100.000 bahasa pemrograman, dengan aturan sintaksis dan semantik yang didefinisikan serupa dengan bahasa alami, meskipun hanya sedikit bahasa yang diadopsi secara luas.

Tipe data merupakan komponen penting dari bahasa pemrograman, berfungsi sebagai klasifikasi yang memberi tahu kompiler atau penerjemah bagaimana pengembang ingin menyisipkan dan memanipulasi data. (Nagendra Singh et al., 2022) mengidentifikasi bahwa tipe data mengorganisasikan kumpulan data yang

disediakan pengguna ke dalam berbagai nilai, yang kemudian dikategorikan ke dalam variabel dan digunakan sesuai dengan berbagai fungsi dan operasi. Evolusi tipe data telah berkembang dari klasifikasi dasar dalam bahasa pemrograman awal seperti Fortran dan COBOL (bilangan bulat, bilangan floating-point, karakter) menjadi sistem tipe yang canggih termasuk pengetikan statis dan dinamis, tipe data yang ditentukan pengguna, inferensi tipe, dan pemrograman generik.



Gambar 1.1 Bahasa pemrograman, skrip, dan markup (sumber: survey.stackoverflow.co)

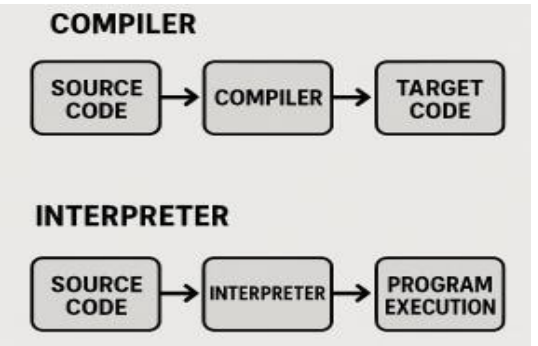
Pada gambar 1.1 Bahasa pemrograman, skrip, dan markup yang diambil berdasarkan survey pengguna website stackoverflow menunjukkan hasil survei mengenai bahasa pemrograman yang ditampilkan hanya diambil 9 dari 42 bahasa pemrograman, skrip, dan markup yang paling banyak digunakan oleh para

pengembang di seluruh dunia. Gambar tersebut juga menunjukan setelah lebih dari satu dekade pertumbuhan yang stabil, adopsi Python telah meningkat secara signifikan. Python mengalami peningkatan sebesar 7 poin persentase dari tahun 2024 ke tahun 2025; hal ini menunjukkan kemampuannya untuk menjadi bahasa pemrograman andalan untuk AI, ilmu data, dan pengembangan back-end.

Berbagai penelitian tentang pendidikan pemrograman menunjukkan adanya dua pendekatan utama dalam mata kuliah pengantar pemrograman, yaitu pedagogi yang berpusat pada konsep dan pedagogi yang berpusat pada aktivitas (Omer et al., 2021). Pendekatan ini mengatur ulang cara konsep atau aktivitas diajarkan untuk meningkatkan efektivitas pembelajaran. (Eteng et al., 2022) menawarkan model pengajaran yang menggabungkan konsep dasar pemrograman dengan representasi masalah komputasi serta memanfaatkan kompiler seluler, sangat bermanfaat bagi negara dengan keterbatasan infrastruktur. Praktik efektif lainnya meliputi gamifikasi, pembelajaran kolaboratif, umpan balik berbasis alat, dan penggunaan IDE berbasis web. Secara keseluruhan, pendidikan pemrograman yang baik tidak hanya fokus pada sintaksis, tetapi juga pada pengembangan kemampuan berpikir komputasional dan pemecahan masalah.

C. *Compiler* dan *Interpreter*

Perbedaan utama *compiler* dan *interpreter* terletak pada cara eksekusinya: *compiler* menerjemahkan seluruh kode sumber menjadi bahasa mesin sebelum dijalankan, sedangkan *interpreter* mengeksekusi kode baris demi baris tanpa menghasilkan berkas eksekusi. *Compiler* melalui beberapa tahap, mulai dari analisis leksikal, sintaksis, dan semantik, hingga pembuatan kode antara, optimasi, dan pembangkitan kode mesin. Pada tahap awal, scanner memecah kode menjadi token, sementara parser memeriksa apakah token tersebut sesuai aturan gramatikal bahasa pemrograman (Asmitha & Panda, 2024; Januar Batista Ardana et al., 2025; Pandey, 2025).



Gambar 1.2. alur kerja *compiler* dan *interpreter*

Gambar 1.2 tersebut mengilustrasikan perbedaan fundamental antara mekanisme kerja *compiler* dan *interpreter* dalam proses translasi bahasa pemrograman tingkat tinggi. Pada model *compiler*, seluruh kode sumber diproses secara menyeluruh untuk diubah

menjadi target code atau kode mesin sebelum program dijalankan. Pendekatan ini menghasilkan berkas eksekusi mandiri sehingga proses eksekusi berikutnya tidak memerlukan penerjemahan ulang. Sebaliknya, pada model *interpreter*, kode sumber tidak dikompilasi menjadi berkas biner terlebih dahulu, melainkan dibaca, dianalisis, dan dijalankan secara langsung oleh *interpreter* secara bertahap, umumnya baris demi baris. Perbedaan proses ini berimplikasi pada waktu eksekusi, kebutuhan sumber daya, serta karakteristik pengembangan program, di mana *compiler* cenderung menghasilkan performa lebih tinggi, sedangkan *interpreter* menawarkan fleksibilitas dan kemudahan dalam proses debugging.

Interpreter menawarkan model eksekusi yang berbeda, menerjemahkan dan mengeksekusi kode secara bersamaan. Implementasi modern bisa sangat canggih, seperti yang ditunjukkan oleh *interpreter* WebAssembly yang mencapai kinerja yang sebanding dengan kode yang dikompilasi sambil mempertahankan fleksibilitas interpretasi (Titzer, 2022). Beberapa sistem canggih menggunakan teknik interpretasi di tempat yang menghilangkan kebutuhan penulisan ulang kode atau format internal terpisah. Kompiler biasanya unggul dalam kecepatan eksekusi dan efisiensi program jangka panjang karena overhead penerjemahan hanya terjadi sekali selama kompilasi. Kode mesin yang dihasilkan dapat berjalan langsung pada perangkat keras target

tanpa langkah penerjemahan tambahan. Namun, waktu kompilasi dan konsumsi memori dapat memengaruhi startup aplikasi, terutama dalam skenario Just-In-Time (JIT) (Titzer, 2022). *Interpreter*, meskipun umumnya lebih lambat dalam eksekusi karena overhead penerjemahan runtime, memberikan keuntungan signifikan dalam lingkungan pengembangan. *Interpreter* memungkinkan eksekusi kode langsung tanpa penundaan kompilasi, menjadikannya ideal untuk pengujian, debugging, dan pembuatan prototipe cepat.

Pilihan antara kompilasi dan interpretasi bergantung pada persyaratan proyek tertentu, sumber daya yang tersedia, dan tujuan pengembangan perangkat lunak (Januar Batista Ardana et al., 2025). Kompiler lebih disukai untuk:

1. Aplikasi yang sangat penting bagi kinerja dan membutuhkan kecepatan eksekusi maksimum
2. Perangkat lunak produksi yang waktu startup-nya kurang penting dibandingkan kinerja runtime
3. Pemrograman sistem yang memerlukan interaksi perangkat keras langsung.

Interpreter optimal untuk:

1. Lingkungan pengembangan yang membutuhkan iterasi dan pengujian yang cepat
2. Lingkungan pendidikan yang membutuhkan umpan balik langsung untuk meningkatkan pembelajaran

3. Aplikasi yang membutuhkan modifikasi kode dinamis selama runtime
4. Tugas skrip dan otomatisasi yang fleksibilitasnya lebih diutamakan daripada masalah kinerja

Sistem perangkat lunak kontemporer sering kali menggunakan pendekatan hibrida, yang menggabungkan teknik kompilasi dan interpretasi. Mesin virtual dapat memiliki beberapa mesin eksekusi, termasuk *interpreter* bytecode dan penerjemah kode mesin dinamis (kompiler JIT), yang memerlukan validasi bahwa mesin-mesin yang berbeda ini mempertahankan kesetaraan fungsional (Polito et al., 2022).

Memahami perbedaan mendasar antara kompiler dan *interpreter* sangat penting untuk membuat keputusan yang tepat dalam pengembangan perangkat lunak. Sementara kompiler menawarkan kinerja runtime yang superior melalui penerjemahan pra-eksekusi, *interpreter* memberikan fleksibilitas dan kemudahan pengembangan yang tak tertandingi. Pilihan optimal bergantung pada keseimbangan persyaratan kinerja, kecepatan pengembangan, kendala sumber daya, dan karakteristik spesifik domain aplikasi target.

D. Latihan Soal

1. Konsep dasar pemrograman mencakup aktivitas berikut, kecuali…
   a. Merancang instruksi

b. Menulis instruksi
c. Menguji instruksi
d. Menginstal perangkat keras

2. Variabel dalam pemrograman berfungsi untuk…
   a. Mengatur struktur kontrol
   b. Menyimpan data sementara
   c. Menampilkan output
   d. Mempercepat proses kompilasi

3. Proses percabangan seperti if dan switch termasuk dalam…
   a. Struktur kontrol
   b. Struktur data
   c. Algoritma berulang
   d. Komunikasi data

4. Urutan langkah sistematis untuk menyelesaikan masalah disebut…
   a. Tipe data
   b. Fungsi
   c. Algoritma
   d. Kompiler

5. Paradigma pemrograman yang memecah program menjadi modul-modul disebut…
   a. OOP
   b. Modular
   c. Struktural
   d. Rekursif

6. Perbedaan utama antara compiler dan interpreter adalah…
   a. Compiler bekerja lebih lambat dari interpreter
   b. Compiler menerjemahkan seluruh kode sekaligus
   c. Interpreter menghasilkan file executable

d. Interpreter tidak dapat menjalankan kode langsung

7. Contoh penggunaan interpreter adalah ketika menjalankan program…
   a. C
   b. C++
   c. Python
   d. Assembly

8. Dalam OOP, konsep untuk membungkus data dan fungsi dalam satu unit disebut…
   a. Polymorphism
   b. Encapsulation
   c. Inheritance
   d. Compilation

9. Bahasa pemrograman yang paling banyak digunakan menurut survei StackOverflow 2025 adalah…
   a. Python
   b. C++
   c. JavaScript
   d. PHP

10. Pada proses kompilasi, tahap yang bertugas memecah kode menjadi token adalah…
    a. Parser
    b. Scanner (Lexer)
    c. Optimizer
    d. Code Generator

BAB 2. TIPE DATA DAN VARIABEL

A. Pendahuluan

Data merupakan representasi fakta atau kejadian dalam bentuk angka, teks, atau simbol yang mendukung proses komputasi dan pengambilan keputusan setelah diolah dengan benar (Sanders, 2016). Sebaliknya tipe data berfungsi mengelompokkan dan membatasi data agar sistem dapat menyimpan, memproses, serta menafsirkan informasi secara tepat, sekaligus mencegah kesalahan (Shankar, 1978). Pemahaman yang baik tentang data dan tipe data membantu pengembang memilih struktur dan metode pengolahan yang efisien untuk meningkatkan kinerja sistem dan kualitas hasil komputasi, khususnya pada skala besar (Smith & Jones, 2023).

Tipe data berperan penting dalam pemrograman karena menentukan cara penyimpanan, representasi, dan pemrosesan informasi oleh komputer. Melalui pengelompokan data seperti angka, karakter, atau teks, sistem dapat mengatur memori dan mencegah kesalahan tipe. Pemilihan tipe data yang tepat mendukung validitas logika program, efisiensi komputasi, serta keandalan system (Telkom, 2025). Selain itu, konsep ini menjadi dasar bagi pembentukan struktur data dan abstraksi yang kompleks, yang

memengaruhi skalabilitas dan kemudahan pemeliharaan perangkat lunak (Asri, Palit, & Noertjahyana, 2024).

B. Tipe Data

1. Tipe Data Integer.

Tipe data ini digunakan untuk merepresentasikan bilangan bulat, baik positif, negatif, maupun nol, tanpa nilai desimal. Tipe data ini berperan dalam operasi aritmetika seperti penjumlahan, pengurangan, perkalian, dan pembagian bilangan bulat. Ukuran dan rentang nilai integer bergantung pada bahasa pemrograman serta arsitektur sistem yang digunakan. Pemahaman integer yang baik membantu pengembang mengelola memori dan memilih algoritma yang efisien dalam pengolahan data numerik (Dicoding Intern, 2020).

2. Tipe Data Float.

Tipe data ini merepresentasikan bilangan real dengan komponen desimal untuk memungkinkan perhitungan numerik yang memerlukan presisi. Tipe ini digunakan dalam berbagai operasi seperti pengukuran ilmiah, statistik, dan keuangan. Representasinya umumnya mengikuti standar IEEE 754 yang menentukan format penyimpanan nilai floating-point dengan rentang luas, meskipun presisinya terbatas. Karena itu, penggunaan float perlu dilakukan dengan hati-hati pada

perhitungan yang membutuhkan ketelitian tinggi (Dicoding Intern, 2020).

3. Tipe Data Character (Char).

Tipe character merupakan merepresentasikan satu karakter tunggal, seperti huruf, angka, atau simbol, yang disimpan dalam kode numerik seperti ASCII atau Unicode. Char menjadi dasar pengolahan teks, memungkinkan manipulasi elemen tekstual secara efisien. Dengan dukungan Unicode, tipe ini dapat menampung berbagai simbol internasional dan emoji. Penggunaan char penting dalam pemrosesan string, validasi input, dan pengkodean data karena karakter merupakan komponen dasar pembentuk teks dalam pemrograman (Dicoding Intern, 2020).

4. Tipe Data String

Tipe string merepresentasikan kumpulan karakter yang tersusun berurutan dan diperlakukan sebagai satu kesatuan, biasanya digunakan untuk menyimpan teks seperti kata, kalimat, atau simbol. String memiliki panjang bervariasi dan mendukung operasi manipulasi seperti penggabungan, pencarian, pemotongan, dan penggantian. Karena perannya yang penting dalam pengolahan teks, antarmuka pengguna, dan komunikasi data, pengelolaan string yang efisien menjadi kunci dalam pengembangan perangkat lunak modern (Dicoding Intern, 2020).

C. Variabel

Variabel merupakan penanda simbolik dalam program yang digunakan untuk menyimpan nilai di dalam memori selama eksekusi, sehingga data dapat diakses dan dimodifikasi sesuai kebutuhan. Setiap variabel memiliki nama dan tipe data yang menentukan bagaimana nilai tersebut direpresentasikan dan diolah oleh komputer. Konsep variabel menjadi dasar dalam pemrograman karena memungkinkan pengelolaan data secara dinamis untuk mendukung logika dan alur kerja program (PSOU, 2025).

Penamaan variabel harus mengikuti aturan yang ada dengan tujuan memastikan identitas variabel dapat dikenali dengan jelas oleh programmer maupun sistem, sehingga penulisan kode menjadi lebih terstruktur dan mudah dipahami. Secara umum, nama variabel harus diawali huruf atau garis bawah, tidak boleh diawali angka, serta tidak boleh mengandung spasi maupun simbol khusus selain garis bawah. Nama variabel tidak boleh menggunakan kata kunci yang telah ditetapkan bahasa pemrograman, serta idealnya merepresentasikan fungsi atau nilai yang disimpan untuk meningkatkan keterbacaan kode. Konvensi seperti penggunaan gaya penulisan camelCase atau snake_case juga sering diterapkan untuk konsistensi dalam proyek pemrograman. (Dicoding, 2020).

Deklarasi suatu Variabel merupakan proses mendefinisikan nama dan tipe data suatu variabel agar

sistem dapat menyediakan ruang memori yang sesuai, sedangkan inisialisasi merupakan pemberian nilai awal terhadap variabel tersebut sebelum digunakan dalam proses komputasi. Deklarasi memastikan program mengetahui bagaimana data akan direpresentasikan dan operasi apa yang dapat dilakukan, sementara inisialisasi mencegah terjadinya kesalahan akibat penggunaan nilai yang belum ditentukan. Kedua proses ini menjadi langkah dasar dalam pemrograman karena menjamin variabel siap digunakan dalam perhitungan, penyimpanan sementara, maupun pengolahan data lainnya. (Dicoding, 2020)

Menampilkan nilai suatu Variabel merupakan proses output data ke layar atau media tampilan lain agar pengguna dapat melihat informasi yang disimpan program. Proses ini umumnya dilakukan dengan menggunakan perintah atau fungsi bawaan bahasa pemrograman, misalnya print() dalam Python atau printf() dalam C, yang akan membaca nilai variabel di memori dan menampilkannya dalam format tertentu. Tahap ini penting untuk memverifikasi hasil perhitungan, menelusuri kesalahan program, serta meningkatkan interaksi antara pengguna dan sistem. Dengan kemampuan menampilkan nilai variabel, programmer dapat mengevaluasi alur logika program secara langsung dan memastikan bahwa variabel mengandung nilai yang sesuai dengan tujuan pemrosesan. (Dicoding, 2020)

Ruang lingkup (scope) Variabel mengacu pada batasan area di dalam program tempat sebuah variabel dapat diakses, sehingga menentukan keterlihatan (visibility) dan masa hidupnya selama eksekusi. Scope umumnya dibagi menjadi variabel lokal dan global; variabel lokal hanya dapat digunakan dalam blok atau fungsi tempat ia dideklarasikan, sedangkan variabel global dapat diakses dari seluruh bagian program. Pemahaman mengenai scope penting untuk mencegah konflik penamaan, mengurangi ketergantungan antarbagian program, serta meningkatkan modularitas dan keamanan data. Dengan mengelola scope secara tepat, programmer dapat memastikan bahwa variabel hanya tersedia ketika dibutuhkan dan tidak dapat dimanipulasi secara tidak sengaja oleh bagian program lain. (Dicoding, 2020).

D. Konstanta

Merupakan sebuah nilai yang ditetapkan pada saat deklarasi dan tidak dapat diubah selama program berjalan, sehingga memberikan stabilitas dalam pengolahan data dan mencegah modifikasi yang tidak diinginkan. Berbeda dari variabel yang nilainya dapat diperbarui, konstanta digunakan untuk menyimpan data tetap seperti nilai matematika (misalnya π), batas tertentu, atau konfigurasi yang tidak boleh berubah. Penggunaan konstanta dapat meningkatkan keterbacaan kode, mengurangi potensi kesalahan, serta

mempermudah pemeliharaan karena perubahan hanya perlu dilakukan pada satu definisi jika diperlukan. Dalam banyak bahasa pemrograman, konstanta dideklarasikan dengan kata kunci khusus seperti const atau final. (Kumparan, 2024) (Prasatya, 2025).

E. Latihan Soal

1. Tipe data yang digunakan untuk menyimpan bilangan bulat tanpa koma adalah ...
   a. Float
   b. Integer
   c. Boolean
   d. Char

2. Tipe data yang paling tepat untuk menyimpan nilai seperti 3.14 adalah ...
   a. Integer
   b. Character
   c. Float
   d. Boolean

3. Tipe data yang digunakan untuk menyimpan satu simbol, misalnya 'A', adalah ...
   a. Integer
   b. String
   c. Character
   d. Float

4. Tipe data yang dapat menyimpan sekumpulan karakter disebut ...
   a. Integer
   b. String
   c. Character
   d. Boolean

5. Pernyataan yang benar mengenai variabel adalah ...
   a. Nilainya tidak dapat diubah

b. Digunakan untuk menyimpan data yang hanya berupa angka.
c. Didefinisikan dan tidak dapat digunakan ulang
d. Nilainya dapat berubah selama program berjalan

6. Istilah untuk penentuan nama dan tipe data variabel agar dapat digunakan dalam program disebut ...
   a. Kompilasi
   b. Eksekusi
   c. Inisialisasi
   d. Deklarasi

7. Memberikan nilai awal pada variable disebut
   a. Kompilasi
   b. Eksekusi
   c. Inisialisasi
   d. Deklarasi

8. Ruang lingkup variabel (variable scope) berhubungan dengan ...
   a. Ukuran memori
   b. Lokasi variable diakses
   c. Cara variable dibaca
   d. Jenis operator

9. Konstanta berbeda dari variable karena....
   a. Tidak memerlukan deklarasi
   b. Hanya digunakan sekali
   c. Nilainya tetap selama program berjalan
   d. Tidak menggunakan deklarasi

10. Kata kunci yang biasanya digunakan untuk mendefinisikan konstanta dalam beberapa bahasa pemrograman adalah ...
    a. Const
    b. For
    c. read
    d. var

BAB 3. STRUKTUR KONTROL PROGRAM

A. Pendahuluan

Dalam dunia pemrograman, sebuah aplikasi tidak dapat langsung berjalan hanya dengan menuliskan kode begitu saja. Seorang programmer perlu membaginya ke dalam beberapa bagian logika, seperti struktur kontrol, pengambilan keputusan (logika), dan pengulangan (iterasi). Pembagian ini bertujuan agar alur program menjadi lebih teratur dan dapat berfungsi sesuai kebutuhan pengguna.

Pada bagian struktur kontrol program, alur pemrograman harus diatur dengan jelas mulai dari proses awal hingga akhir. Dengan pengaturan alur yang baik, aplikasi dapat berjalan sesuai urutan proses yang diinginkan dan selaras dengan tujuan serta proses yang mendasarinya.

Sementara itu, bagian logika merupakan komponen paling penting dalam suatu program. Bagian ini berfungsi sebagai "otak" yang menentukan arah pengambilan keputusan dari keseluruhan sistem. Misalnya, pada aplikasi mitigasi tingkat keparahan COVID-19 (Prasandy, 2023), logika fuzzy menggunakan struktur if–else untuk menentukan tingkat keparahan kasus berdasarkan data masukan tertentu.

Bagian terakhir adalah pengulangan (iterasi). Struktur ini sangat penting, terutama saat program perlu menampilkan data atau membuat visualisasi. Jika tidak diatur dengan benar, perulangan dapat berjalan tanpa batas sehingga menampilkan seluruh data secara terus-menerus. Akibatnya, program bisa tidak terkendali, tampilan tabel atau grafik menjadi rusak, bahkan menyebabkan server berhenti (hung).

B. Struktur Urutan (*Sequence*)

Struktur urutan berarti program dijalankan baris demi baris dari atas ke bawah. Setiap perintah akan dieksekusi sesuai urutan penulisannya.

Source Code

```
print("Langkah 1: Menyalakan kompor")
print("Langkah 2: Memasak air")
print("Langkah 3: Menyeduh kopi")
```

Output

```
Langkah 1: Menyalakan kompor
Langkah 2: Memasak air
Langkah 3: Menyeduh kopi.
```

C. Struktur Seleksi (Selection / Percabangan)

Struktur seleksi digunakan ketika program harus memilih satu dari beberapa kemungkinan keputusan

berdasarkan suatu kondisi. Python menggunakan kata kunci:

if → untuk memeriksa kondisi.
elif → untuk menambahkan kondisi tambahan.
else → untuk kondisi alternatif terakhir.

Contoh sederhana

```
nilai = 80
if nilai >= 90:
    print("Nilai Anda A")
elif nilai >= 75:
    print("Nilai Anda B")
else:
    print("Nilai Anda C")
```

Output

```
Nilai Anda B
```

Catatan

Gunakan tanda : di akhir baris if, elif, dan else.

Gunakan indentasi (spasi/tab) untuk menunjukkan blok kode di dalam kondisi.

Gunakan operator perbandingan seperti ==, !=, <, >, <=, >=

Contoh Menggunakan Operator Logika

```
umur = 20
berpenghasilan = True
if umur >= 18 and berpenghasilan:
    print("Anda sudah dewasa dan mandiri.")
```

Output

```
Anda sudah dewasa dan mandiri.
```

Operator yang biasa digunakan dalam phyton dapat dilihat pada tabel 3.1

Tabel 3.1. Operator Logika di Phyton

Operator	Arti
and	kedua kondisi benar
or	salah satu kondisi benar
not	membalikan nilai kondisi

Bentuk Percabangan If

Contoh

```
total_belanja = 250000
if total_belanja >= 500000:
    diskon = 0.2
    print("Anda mendapatkan diskon
          20%")
elif total_belanja >= 200000:
    diskon = 0.1
    print("Anda mendapatkan diskon
          10%")
else:
    diskon = 0
    print("Maaf, belum ada diskon untuk
          pembelian ini.")
total_bayar = total_belanja -
(total_belanja * diskon)
print("Total yang harus dibayar:",
total_bayar)
```

Output

```
Anda mendapatkan diskon 10%
Total yang harus dibayar: 225000.0
```

Bentuk Percabangan Bertingkat

Kadang kita perlu membuat keputusan di dalam keputusan lain. Hal ini disebut percabangan bertingkat (*nested if*).

```
umur = 25
status = "pelajar"
if umur >= 18:
    if status == "pelajar":
        print("Anda mendapat diskon
              pelajar 20%")
    else:
        print("Anda mendapat diskon umum
              10%")
else:
    print("Maaf, Anda belum memenuhi
          syarat diskon.")
```

Output

Anda mendapat diskon pelajar 20%

Catatan:

Walaupun nested if berguna, sebaiknya digunakan dengan hati-hati agar kode tidak terlalu panjang dan sulit dibaca. Untuk kondisi kompleks, lebih baik menggunakan struktur data atau fungsi terpisah.

Struktur seleksi memberikan kemampuan bagi program untuk “berpikir” dan menyesuaikan tindakan

berdasarkan kondisi. Dengan memahami cara kerja if–elif–else, programmer dapat membuat sistem yang: Merespons input pengguna, Mengambil keputusan otomatis, dan Mengendalikan alur proses secara dinamis.

Konsep ini menjadi dasar bagi banyak fitur dalam aplikasi nyata, seperti login sistem, validasi data, hingga sistem rekomendasi.

Contoh: Logika Fuzzy Mitigasi Tingkat Keparahan COVID-19 (Prasandy, 2023)

Struktur seleksi juga dapat digunakan untuk sistem pengambilan keputusan kompleks berbasis logika fuzzy, seperti penelitian Prasandy (2023) mengenai mitigasi tingkat keparahan COVID-19. Dalam sistem tersebut, beberapa parameter kesehatan pasien digunakan untuk menentukan kategori tingkat keparahan:

a. Suhu tubuh (body temperature)
b. Saturasi oksigen (blood oxygen saturation / SpO_2)
c. Detak jantung (heart rate)
d. Tekanan darah (blood pressure)
e. Tingkat batuk (cough)

Setiap parameter dikategorikan menjadi tiga tingkat: Normal, Middle, dan High. Kombinasi dari lima parameter tersebut menghasilkan prediksi tingkat keparahan pasien: Good, Middle, atau Bad. Ilustrasi Aturan Logika Fuzzy (Contoh dari Prasandy, 2023). Beberapa contoh aturan keputusan:

a. IF suhu tubuh High, SpO_2 Low, detak jantung High, tekanan darah High, dan batuk High → prediksi = Bad
b. IF semua parameter Middle → prediksi = Middle
c. IF suhu tubuh Normal, SpO_2 High, dan parameter lain Normal/Low → prediksi = Good

Daftar lengkap berisi 20 aturan keputusan sebagaimana diteliti dalam model Prasandy, 2023. Implementasi struktur seleksi tersebut di Python:

```
body_temp = "High"
oxygen = "Low"
heart_rate = "High"
blood_pressure = "High"
cough = "High"
if body_temp == "High" and oxygen ==
             "Low" and heart_rate ==
             "High" and blood_pressure
             == "High" and cough ==
             "High":
    prediction = "Bad"
elif body_temp == "Normal" and oxygen
            == "High" and heart_rate
            ==      "Normal"      and
            blood_pressure == "Normal"
            and cough == "Low":
    prediction = "Good"
elif body_temp == "Middle" and oxygen
            == "Middle" and heart_rate
            ==      "Middle"      and
            blood_pressure == "Middle"
            and cough == "Middle":
    prediction = "Middle"
else:
    prediction = "Undetermined"
print("Prediksi    tingkat    keparahan
COVID-19:", prediction)
```

Output

```
Prediksi tingkat keparahan COVID-19: Bad
```

Program di atas menggunakan beberapa kondisi yang saling terhubung untuk menentukan kategori prediksi. Kombinasi lima variabel kesehatan pasien diolah menggunakan serangkaian perintah if–elif–else sehingga menghasilkan prediksi akhir.

Pendekatan ini menunjukkan bahwa struktur seleksi dapat digunakan tidak hanya untuk keputusan sederhana, tetapi juga untuk sistem berbasis aturan yang kompleks, seperti logika fuzzy, kecerdasan buatan, dan sistem pakar di bidang medis.

Dari contoh tersebut dapat disimpulkan bahwa Struktur if–elif–else memungkinkan program menyesuaikan perilaku berdasarkan kondisi tertentu. Kombinasi operator logika (and, or, not) memperluas kemampuan pengambilan keputusan. Prinsip dasar pengambilan keputusan ini menjadi fondasi bagi sistem cerdas di berbagai bidang, mulai dari ekonomi hingga kesehatan.

Pemahaman yang baik terhadap struktur seleksi akan memudahkan programmer untuk membangun program yang adaptif, efisien, dan cerdas.

D. Percabangan Lainnya menggunakan *Match-Case*

Python memperkenalkan sintaks baru bernama *match–case*, yang berfungsi mirip *switch–case* di bahasa lain. Namun, versi Python ini jauh lebih fleksibel karena mendukung *pattern matching* (pencocokan pola).

```
hari = 3
match hari:
    case 1:
        print("Senin")
    case 2:
        print("Selasa")
    case 3:
        print("Rabu")
    case 4:
        print("Kamis")
    case _:
        print("Hari tidak dikenal")
```

Output

```
Rabu
```

Penjelasan:

match → memeriksa nilai variabel.

case → berfungsi seperti case pada switch.

Perbandingan If-Elif-Else dengan Match-Case dapat dilihat pada tabel 3.2 berikut

Tabel 3.2 Perbandingan If-Elif-Else dengan Match-Case

Aspek	If–Elif–Else	Match–Case (Python ≥3.10)
Versi tersedia	Semua versi Python	Mulai Python 3.10
Tujuan	Percabangan logika umum	Pencocokan pola dan pengganti switch
Kelebihan	Mudah dipahami, fleksibel	Lebih ringkas, rapi, bisa memeriksa struktur data
Kekurangan	Bisa panjang dan berulang	Hanya tersedia di Python terbaru

E. Struktur Iterasi (Perulangan / Loop)

Dalam pemrograman, sering kali kita perlu melakukan suatu proses berulang kali tanpa menuliskannya berulang-ulang secara manual. Contohnya, menampilkan daftar siswa, menghitung rata-rata nilai dari banyak data, atau memproses ribuan transaksi.

Agar lebih efisien, Python menyediakan struktur iterasi (loop), yaitu perintah yang memungkinkan pengulangan blok kode secara otomatis hingga kondisi tertentu terpenuhi. Terdapat dua jenis utama perulangan di Python:

1. for loop → digunakan saat jumlah perulangan sudah diketahui,
2. while loop → digunakan saat jumlah perulangan belum pasti dan bergantung pada kondisi tertentu.

Perulangan for

Struktur for digunakan ketika kita ingin menjalankan kode berulang kali dengan jumlah yang sudah diketahui, biasanya dengan bantuan fungsi range() atau ketika menelusuri elemen-elemen dalam koleksi data seperti list, tuple, atau string.

Contoh : Perulangan dengan range()

```
for i in range(5):
print("Perulangan ke-", i)
```

Output

```
Perulangan ke- 0
Perulangan ke- 1
Perulangan ke- 2
Perulangan ke- 3
Perulangan ke- 4
```

Fungsi range() menghasilkan deretan angka:

a. range(n) → menghasilkan 0 sampai n–1
b. range(a, b) → menghasilkan angka mulai dari a hingga b–1
c. range(a, b, c) → menghasilkan angka dari a hingga b–1 dengan loncatan c

Contoh Lainnya

```
for i in range(2, 11, 2):
    print(i)
```

Output

```
2
4
6
8
10
```

Perulangan for tidak hanya untuk angka, tapi juga bisa digunakan untuk menelusuri isi dari list atau string.

Contoh

```
buah = ["apel", "jeruk", "mangga"]
for item in buah:
    print("Saya suka", item)
```

Output

```
Saya suka apel
Saya suka jeruk
Saya suka mangga
```

Dengan cara ini, Python memudahkan kita untuk memproses seluruh data dalam koleksi tanpa harus tahu panjangnya.

Perulangan while

Struktur while digunakan saat jumlah perulangan belum diketahui dan bergantung pada kondisi logika tertentu. Program akan terus mengulangi blok perintah selama kondisi bernilai True.

Contoh : Menghitung Nilai

```
hitung = 1
while hitung <= 5:
    print("Hitung:", hitung)
    hitung += 1
```

Output

```
Hitung: 1
Hitung: 2
Hitung: 3
Hitung: 4
Hitung: 5
```

Pastikan kondisi di dalam while suatu saat menjadi False, jika tidak, perulangan akan berjalan tanpa henti (infinite loop) dan menyebabkan program berhenti merespons.

Mengontrol Jalannya Perulangan

Sama seperti Bahasa pemrograman lainnya Python menyediakan beberapa perintah khusus untuk mengatur perilaku loop, dapat dilihat pada tabel 3.3 berikut