Buku Pegangan Python SMK

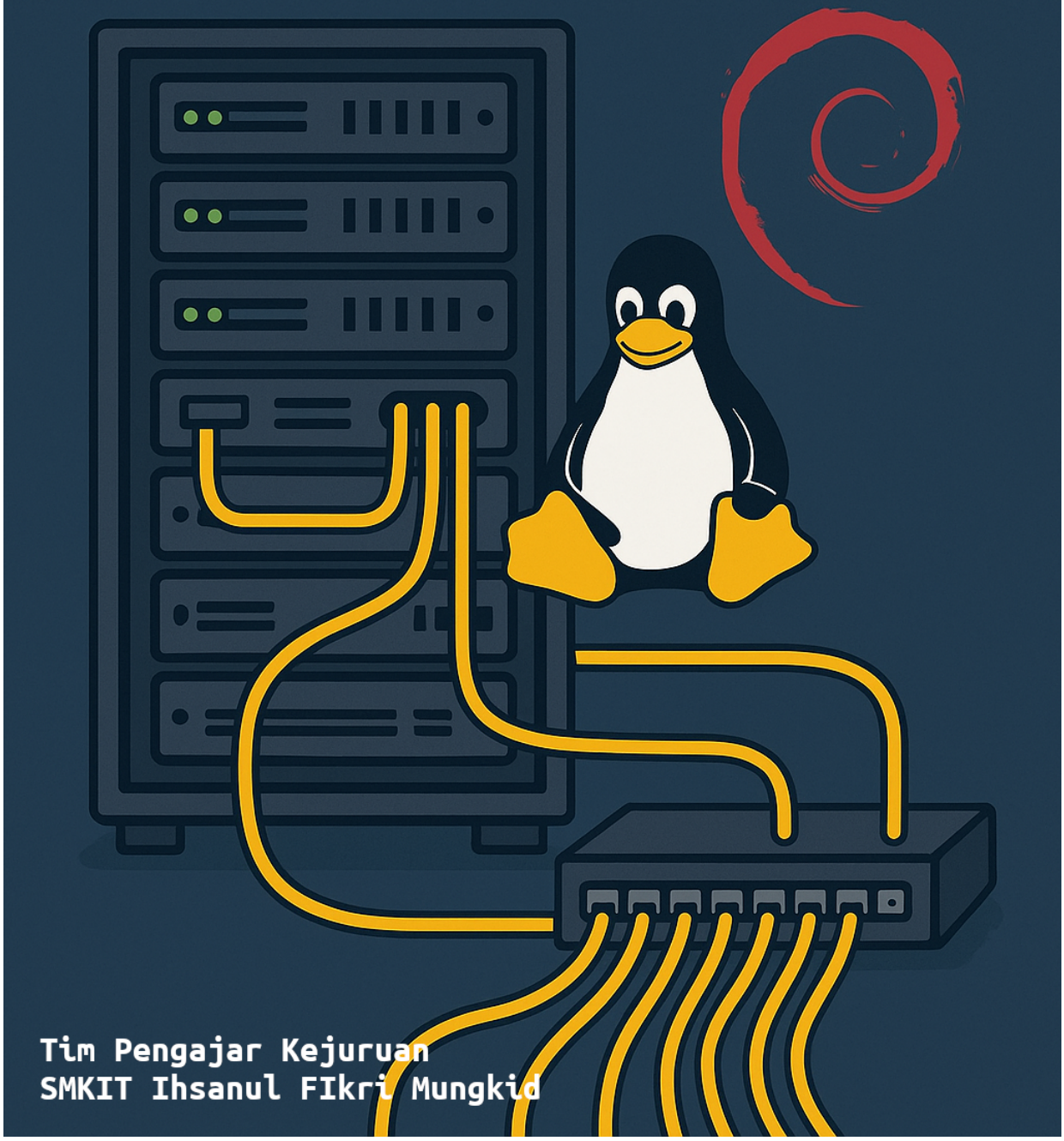
Untuk SMKIT Ihsanul Fikri Mungkid

Saiful Habib

28 Agustus 2025

Buku Pegangan Siswa Mapel Sysadmin Kelas XI

Semester 1



Tim Pengajar Kejuruan
SMKIT Ihsanul FIkri Mungkid

Table of contents

1 Berpikir Komputasional *(Computational Thinking)*

1.1 Definisi Berpikir Komputasional

> *Computational Thinking is the thought processes involved in formulating problems and their solutions so that the solutions are represented in a form that can be effectively carried out by an information-processing agent*

Berpikir komputasional adalah sebuah cara berpikir yang digunakan untuk memahami dan menyelesaikan masalah, baik itu masalah sehari-hari, pelajaran, atau bahkan permainan. Dalam berpikir komputasional, kita belajar bagaimana memecah suatu masalah menjadi bagian-bagian yang lebih kecil agar lebih mudah dipahami dan diselesaikan.

Misalnya, bayangkan kamu ingin membuat kue. Sebelum mulai, kamu perlu tahu bahan-bahannya, langkah-langkah membuat adonannya, memanggangnya, dan akhirnya menyajikannya. Jika kamu langsung mencoba membuat kue tanpa mengikuti langkah-langkah ini, kamu mungkin akan kesulitan atau hasilnya tidak sesuai harapan. Berpikir komputasional membantu kita menguraikan proses ini menjadi tahapan-tahapan yang lebih kecil dan teratur, sehingga kita bisa menyelesaikan tugas dengan lebih efektif dan efisien.

Selain itu, Berpikir komputasional tidak hanya membantu manusia, tetapi juga bisa diterapkan pada komputer. Misalnya, ketika kamu membuat sebuah program komputer untuk menghitung nilai rata-rata kelas, kamu perlu memberi tahu komputer langkah-langkah yang harus diikuti, seperti menjumlahkan semua nilai dan membaginya dengan jumlah siswa. Komputer akan mengikuti instruksi ini untuk memberikan jawaban yang benar.

Intinya, berpikir komputasional adalah cara berpikir yang membantu kita mengubah masalah besar menjadi lebih kecil dan lebih mudah dikelola. Ini sangat berguna tidak hanya untuk belajar pemrograman komputer, tetapi juga untuk kehidupan sehari-hari, seperti menyelesaikan PR, mengatur jadwal belajar, atau merencanakan kegiatan dengan teman-teman.

1.2 Pentingnya *Computational Thinking*

Dilansir dari BBC[1], computational thinking tetap diperlukan meski pekerjaanmu sama sekali tidak berhubungan dengan pemograman komputer. Selain itu, computational thinking adalah skill yang memiliki fungsi hampir di setiap sektor industri. Misal dari produk konsumen, bisnis dan pasar keuangan, energi, pariwisata, atau layanan publik seperti perawatan kesehatan, pendidikan serta hukum dan ketertiban.

Skill ini juga dapat diterapkan pada lini apa pun dari bisnis komersial atau layanan publik. Perencanaan dan peramalan didasarkan pada pola generalisasi atau abstraksi. Contohnya, merancang user story mapping untuk situs e-commerce akan melibatkan kemampuan untuk memecahkan masalah menjadi bagian-bagian kecil menggunakan teknik dekomposisi. Kemudian menyusun urutan langkah-langkah untuk menyelesaikan masalah menggunakan pemikiran algoritmis. Bahkan, jika kamu tidak berada dalam posisi untuk membuat solusi menggunakan bahasa pemrograman dan komputer, kamu dapat memahami dan memikirkan masalah bisnis dengan menggunakan konsep-konsep computational thinking.

1.3 Manfaat Computational Thinking

1.3.1 Menyediakan Langkah Problem Solving yang Efektif

Computational thinking melibatkan pendekatan sistematis dalam memecahkan masalah, yang mencakup empat aspek utama: dekomposisi, pengenalan pola, abstraksi, dan algoritma. Dengan menggunakan langkah-langkah ini, seseorang dapat memecah masalah besar menjadi bagian-bagian yang lebih kecil dan lebih mudah dikelola, mengenali pola yang relevan, dan menyusun solusi berdasarkan informasi yang telah disederhanakan. Ini memungkinkan penyelesaian masalah yang lebih efisien dan terfokus, karena pendekatan ini mengurangi kemungkinan kesalahan dan meningkatkan akurasi dalam menemukan solusi.

[1] sumber: https://id.wikipedia.org/wiki/Algoritma

1.3.2 Melatih Mindset untuk Menjadi Lebih Kreatif

Dalam computational thinking, pemikiran kreatif diperlukan untuk menemukan solusi inovatif. Ketika seseorang menggunakan teknik seperti pengenalan pola dan abstraksi, mereka didorong untuk berpikir "out of the box" atau dari perspektif yang berbeda. Misalnya, menemukan cara baru untuk mengotomatiskan tugas atau mengurangi kompleksitas masalah. Latihan berpikir secara kreatif ini tidak hanya bermanfaat untuk pemrograman atau ilmu komputer, tetapi juga berguna dalam berbagai bidang lain seperti desain, manajemen, dan bahkan seni.

1.3.3 Pola Pikir Menjadi Lebih Logis dan Terstruktur

Computational thinking membantu individu mengembangkan pola pikir yang logis dan terstruktur. Ini berarti mereka terbiasa mengikuti alur berpikir yang jelas dan sistematis, seperti mengidentifikasi masalah, merumuskan hipotesis, mengembangkan solusi, dan mengevaluasi hasil. Proses ini tidak hanya membantu dalam pemecahan masalah teknis, tetapi juga bermanfaat dalam pengambilan keputusan sehari-hari. Pola pikir yang logis dan terstruktur ini memungkinkan seseorang untuk menyusun argumen yang kuat, merencanakan strategi yang efektif, dan mengelola tugas dengan lebih baik.

1.3.4 Mendorong Kita untuk Menjadi Lebih Profesional dan Efisien

Dengan menerapkan computational thinking, kita dapat meningkatkan efisiensi kerja mereka. Misalnya, dengan memahami cara mengotomatisasi tugas-tugas berulang atau memecahkan masalah kompleks dengan lebih cepat, kita dapat menghemat waktu dan sumber daya. Selain itu, kemampuan untuk berpikir secara analitis dan sistematis juga membuat mereka lebih mampu mengantisipasi masalah, mengambil tindakan preventif, dan memberikan solusi yang lebih cepat dan tepat. Hal ini tidak hanya meningkatkan produktivitas individu, tetapi juga meningkatkan profesionalisme dan kualitas kerja secara keseluruhan.

Dengan memahami dan menerapkan computational thinking, seseorang tidak hanya menjadi lebih terampil dalam memecahkan masalah, tetapi juga menjadi lebih siap menghadapi tantangan yang kompleks dalam dunia kerja dan kehidupan sehari-hari.

1.4 Komponen *Computational Thinking*

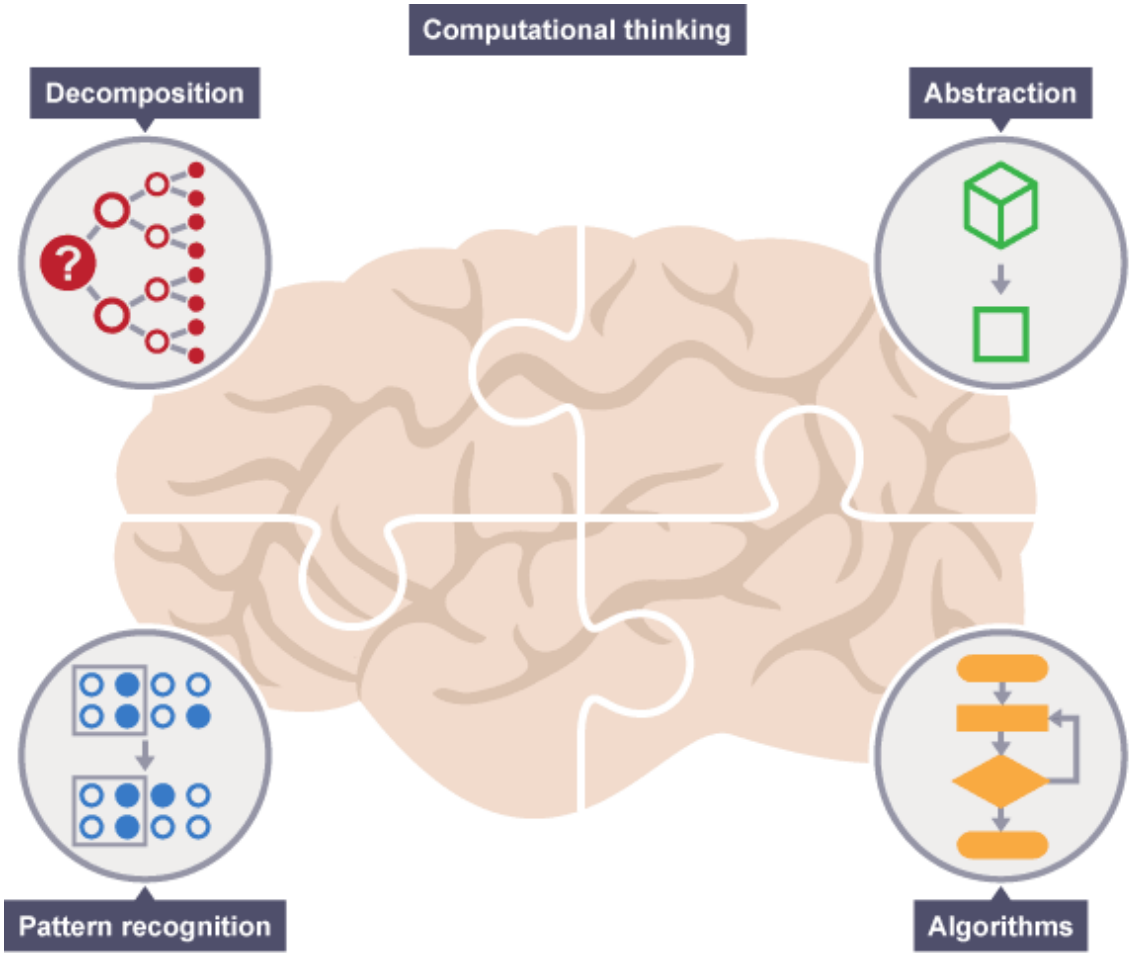
Berpikir komputasional *(Computational Thingking)* terdiri dari empat komponen utama yang membantu kita dalam memecahkan masalah secara lebih terstruktur dan efisien. Keempat komponen ini adalah *dekomposisi*, *pengenalan pola* (*pattern recognition*), *abstraksi*, dan *algoritma*. Dengan memahami dan menerapkan keempat komponen ini, kita dapat menguraikan masalah yang kompleks menjadi lebih sederhana, mengenali pola atau kesamaan dari masalah sebelumnya, menyaring informasi yang tidak penting, serta merancang langkah-langkah logis untuk mencapai solusi. Mari kita bahas lebih dalam tentang masing-masing komponen ini dan bagaimana mereka bekerja bersama untuk menyelesaikan berbagai masalah.

1.4.1 Decomposition (Dekomposisi)

Komponen pertama dalam *Computational Thinking* adalah *dekomposisi*, yaitu memecah masalah kompleks menjadi beberapa bagian kecil dan lebih sederhana. Dengan memecah masalah, kita bisa fokus menyelesaikan satu bagian kecil terlebih dahulu, sehingga masalah besar menjadi lebih mudah diatasi. Misalnya, jika kita diminta membuat sebuah game, langkah pertama yang bisa kita lakukan adalah memecah masalah besar itu menjadi bagian-bagian seperti merancang karakter, membuat alur cerita, menentukan aturan permainan, dan membuat kode untuk interaksi pemain. Dengan memecah masalah menjadi tahapan-tahapan kecil ini, kita dapat menangani setiap bagian satu per satu hingga seluruh game selesai.

1.4.2 Pattern Recognition (Pengenalan Pola)

Tahap kedua adalah *pengenalan pola*. Pada tahap ini, kita mencari pola atau kesamaan yang muncul dari masalah-masalah sebelumnya. Pengenalan pola membantu kita lebih cepat menemukan solusi, karena jika kita menemukan pola yang sama dengan masalah yang sudah pernah kita selesaikan, kita bisa menggunakan pendekatan atau solusi yang sama. Misalnya, saat kamu belajar matematika, sering kali kamu akan menemukan soal yang mirip dengan soal-soal sebelumnya. Dengan mengenali pola penyelesaian



Gambar 1: 4_pilar

dari soal-soal yang pernah kamu kerjakan, kamu bisa lebih mudah menyelesaikan soal baru yang memiliki pola serupa.

1.4.3 Abstraction (Abstraksi)

Tahap ketiga adalah *abstraksi*, yaitu menyaring informasi yang penting dan mengabaikan informasi yang tidak relevan. Di sini, kita berusaha melihat inti dari masalah tanpa terlalu terpengaruh oleh detail yang tidak diperlukan. Sebagai contoh, saat mendesain poster tentang pentingnya membuang sampah pada tempatnya, kita tidak perlu membuat gambar orang membuang sampah secara detail untuk menyampaikan pesan. Cukup dengan ilustrasi sederhana, kita dapat mengurangi usaha pembuatan dan membantu pembaca lebih fokus pada pesan utama yang ingin disampaikan.

1.4.4 Algorithm (Algoritma)

Tahap terakhir adalah membuat *algoritma*, yaitu serangkaian langkah atau instruksi yang jelas untuk menyelesaikan masalah secara efektif. Algoritma seperti resep memasak yang memberi tahu kita langkah-langkah yang harus diambil secara berurutan. Contoh sederhana adalah ketika kamu ingin menghitung rata-rata nilai kelas. Algoritma yang digunakan bisa berupa langkah-langkah seperti: menjumlahkan semua nilai siswa, kemudian membagi hasilnya dengan jumlah siswa. Dengan algoritma yang baik, masalah bisa diselesaikan dengan cara yang efisien dan tidak memakan waktu lama.

1.5 Langkah Menerapkan Berpikir Komputasional

Berpikir komputasional dilakukan secara bertahap dan berulang-ulang (iteratif) melalui tiga langkah utama: *Spesifikasi Masalah* (*Problem Specification*), *Ekspresi Algoritmik* (*Algorithmic Expression*), dan *Implementasi Solusi dan Evaluasi* (*Solution Implementation & Evaluation*). Setiap tahap ini berperan penting dalam menyelesaikan masalah secara terstruktur.

1.5.1 Spesifikasi Masalah**

Pada tahap ini, kita menganalisis masalah secara menyeluruh. Kita memecah masalah besar menjadi bagian-bagian kecil yang lebih mudah diatasi (*dekomposisi*), mengidentifikasi informasi yang penting dan membuang yang tidak relevan (*abstraksi*), serta mencari pola atau kesamaan dari masalah serupa yang pernah kita hadapi (*pengenalan pola*). Selain itu, kita juga menetapkan kriteria yang menunjukkan bahwa masalah tersebut sudah selesai dengan baik.

Contoh: Kamu diminta membuat aplikasi sederhana untuk menghitung rata-rata nilai siswa. Pada tahap ini, kamu akan memecah masalah dengan menentukan input yang diperlukan (nilai siswa), proses penghitungan (menjumlahkan nilai dan membaginya dengan jumlah siswa), dan output yang diinginkan (rata-rata nilai). Di sini, kamu juga harus menetapkan apa yang dianggap sukses—misalnya, aplikasi harus menghitung rata-rata dengan benar tanpa error.

1.5.2 Ekspresi Algoritmik**

Setelah kita memahami masalah dengan jelas, tahap selanjutnya adalah membuat langkah-langkah atau instruksi yang harus diikuti untuk menyelesaikan masalah tersebut. Langkah-langkah ini bisa kita tuliskan dalam bentuk kalimat terstruktur, *pseudo code* (kode semu yang menyerupai bahasa pemrograman), atau menggunakan *flowchart* (bagan alur). Semua ini bertujuan agar solusi dapat dipahami dan diimplementasikan dengan mudah.

Contoh: Untuk masalah menghitung rata-rata nilai siswa, langkah-langkah yang bisa kamu buat misalnya:

1. Ambil input jumlah siswa.
2. Ambil input nilai setiap siswa.
3. Jumlahkan semua nilai.
4. Bagikan hasil penjumlahan dengan jumlah siswa.
5. Tampilkan hasil rata-rata.

Langkah-langkah tersebut bisa diubah menjadi *pseudo code* atau digambarkan dalam bentuk *flowchart* agar lebih mudah dipahami dan diimplementasikan.

1.5.3 Implementasi Solusi dan Evaluasi**

Tahap terakhir adalah mengimplementasikan langkah-langkah yang sudah kita buat. Ini berarti kita menjalankan solusi yang sudah direncanakan dan kemudian mengevaluasi apakah hasilnya sesuai dengan harapan. Jika ada masalah atau kesalahan, kita kembali ke langkah sebelumnya dan memperbaiki algoritma. Setelah evaluasi, solusi tersebut bisa diterapkan pada masalah serupa di masa depan.

Contoh: Setelah membuat algoritma untuk menghitung rata-rata nilai siswa, kamu bisa menuliskannya menjadi sebuah program sederhana menggunakan bahasa pemrograman seperti Python atau C++. Setelah program dijalankan, kamu bisa memeriksa apakah hasil rata-rata yang dihasilkan sesuai dengan nilai-nilai yang dimasukkan. Jika program menghasilkan nilai yang salah, kamu bisa mengecek kembali langkah-langkah penyelesaianmu dan memperbaiki kesalahan. Setelah itu, program yang sudah benar bisa kamu gunakan lagi untuk menghitung rata-rata nilai pada kelas lain atau data yang serupa.

Dengan mengikuti ketiga langkah ini, kamu bisa menyelesaikan berbagai masalah secara terstruktur, mulai dari menganalisis masalah hingga membuat solusi yang bisa digunakan secara efektif.[2] [3]

[2] sumber: https://bee.telkomuniversity.ac.id/pengertian-flowchart-fungsi-jenis-simbol-dan-contohnya/
[3] Sumber 2: https://towardsdatascience.com/computational-thinking-defined-7806ffc70f5e

2 Algoritma Pemrograman

2.1 Pengenalan Algoritma Pemrograman

2.1.1 Sejarah Algoritma

Istilah "algoritma" berasal dari nama seorang ilmuwan Persia bernama Abu Ja'far Mohammed Ibn Musa al-Khowarizmi, yang hidup pada abad ke-9. Ia menulis sebuah buku berjudul *al-mukhta□ar fī□isāb al-jabr wal-muqābala*, yang dalam bahasa Inggris dikenal sebagai *The Compendious Book on Calculation by Completion and Balancing*. Karya ini mengandung dasar-dasar ilmu aljabar dan dianggap sebagai salah satu buku matematika paling berpengaruh di dunia. Dari namanya, muncullah kata "algoritma" yang kini digunakan dalam dunia teknologi untuk menggambarkan urutan langkah yang logis untuk menyelesaikan masalah. [4]

2.1.2 Definisi Algoritma

Secara umum, algoritma adalah serangkaian langkah atau instruksi yang dirancang untuk menyelesaikan masalah tertentu. Langkah-langkah ini harus jelas, terstruktur, dan dapat diterapkan secara berulang pada situasi atau data yang berbeda untuk mendapatkan hasil yang diinginkan. Dalam kehidupan sehari-hari, kita sering menggunakan algoritma, meskipun mungkin tidak kita sadari. Misalnya, resep masakan adalah algoritma—langkah-langkah sistematis untuk menghasilkan hidangan.

2.1.3 Algoritma dalam Pemrograman

Dalam konteks pemrograman, algoritma lebih spesifik sebagai urutan langkah yang digunakan untuk menyelesaikan masalah matematika atau logika dengan bantuan komputer. Pemrograman adalah proses menuliskan algoritma tersebut dalam bahasa pemrograman agar komputer dapat menjalankannya. Sebagai contoh, sebuah algoritma bisa digunakan untuk menyelesaikan masalah seperti pengurutan angka dari yang terkecil hingga yang terbesar, menghitung rata-rata nilai, atau bahkan memprediksi cuaca berdasarkan data historis.

2.1.4 Sifat-Sifat Algoritma

Agar algoritma bisa digunakan secara efektif, menurut **Donald E.Knuth** ia harus memenuhi beberapa kriteria penting yaitu:

- **Terdefinisi dengan baik *(Definiteness)***: Setiap langkah dalam algoritma harus jelas dan tidak ambigu, sehingga siapa pun yang membaca algoritma tersebut akan memahami apa yang harus dilakukan.
- **Terbatas *(Finiteness)***: Algoritma harus berakhir setelah sejumlah langkah tertentu. Algoritma yang tidak memiliki titik akhir akan terus berjalan tanpa pernah menyelesaikan masalah.
- **Efisien**: Algoritma harus disusun sedemikian rupa agar dapat menyelesaikan masalah dalam waktu dan sumber daya yang efisien.
- **Input dan Output**: Algoritma harus menerima input (data awal) dan menghasilkan output (hasil dari langkah-langkah pemrosesan).

2.2 Penulisan Algoritma Pemrograman

Algoritma pemrograman dapat dituliskan dengan tiga macam notasi yaitu *Kalimat Terstruktur*, *Pseudo Code* dan *Flowchart*

Berikut elaborasi untuk poin tentang *Kalimat Terstruktur* dalam algoritma pemrograman:

2.2.1 Kalimat Terstruktur

Kalimat terstruktur adalah cara sederhana untuk mendeskripsikan langkah-langkah sebuah algoritma dengan menggunakan bahasa yang mudah dipahami. Notasi ini menggunakan kalimat biasa yang diatur secara jelas dan berurutan. Tidak ada aturan baku yang harus diikuti, tetapi penting untuk menyusun setiap langkah dengan rapi dan logis agar mudah dimengerti. Kalimat terstruktur sering digunakan ketika

[4] sumber: https://id.wikipedia.org/wiki/Algoritma

algoritma masih dalam tahap awal perancangan atau ketika kita mencoba untuk menjelaskan cara kerja algoritma kepada seseorang yang tidak terlalu familiar dengan pemrograman.

Kelebihan:

- **Mudah dipahami**: Karena menggunakan bahasa sehari-hari, kalimat terstruktur dapat dimengerti oleh siapa saja, bahkan oleh orang yang belum mengerti bahasa pemrograman.
- **Fleksibel**: Tidak ada aturan ketat, sehingga kita bisa menulisnya dengan gaya yang paling nyaman.

Kekurangan:

- **Tidak standar**: Karena tidak ada aturan khusus, setiap orang mungkin akan menulisnya dengan cara yang berbeda, sehingga bisa sulit untuk diubah menjadi kode pemrograman.
- **Sulit dikonversi menjadi kode**: Notasi ini lebih mudah dipahami manusia, namun terkadang sulit diterjemahkan langsung menjadi kode program karena tidak mengikuti format tertentu seperti *pseudo code* atau *flowchart*.

Contoh Penggunaan: Misalnya, jika kita ingin membuat algoritma sederhana untuk menambahkan dua angka, kita bisa menuliskannya dengan kalimat terstruktur sebagai berikut:

```
Algoritma penjumlahan
1. Masukkan angka pertama (angka1)
2. Masukkan angka kedua (angka2)
3. Jumlahkan angka1 dan angka2 dan simpan hasilnya
4. Tampilkan hasil penjumlahan
```

Pada contoh ini, setiap langkah diuraikan secara jelas dan berurutan, sehingga pembaca dapat mengikuti prosesnya dengan mudah. Misalnya, pada langkah pertama dan kedua, kita meminta pengguna memasukkan dua angka. Kemudian, kita menjumlahkan kedua angka tersebut dan menyimpan hasilnya. Langkah terakhir adalah menampilkan hasil penjumlahan.

Aplikasi dalam Pemrograman: Walaupun *kalimat terstruktur* cocok digunakan untuk algoritma sederhana, jika kita ingin mengubah algoritma ini menjadi kode program, mungkin akan sedikit sulit karena kalimat-kalimat tersebut harus diterjemahkan terlebih dahulu ke dalam sintaksis bahasa pemrograman. Misalnya, dalam Python, algoritma penjumlahan tersebut dapat diimplementasikan seperti berikut:

```python
# Algoritma penjumlahan dalam Python
angka1 = int(input("Masukkan angka pertama: "))
angka2 = int(input("Masukkan angka kedua: "))
hasil = angka1 + angka2
print("Hasil penjumlahan:", hasil)
```

Seperti yang terlihat, setiap kalimat terstruktur diterjemahkan menjadi instruksi yang bisa dijalankan oleh komputer. Meskipun kalimat terstruktur sangat membantu dalam tahap awal perancangan algoritma, untuk menjalankan algoritma tersebut dalam komputer, kita tetap perlu mengubahnya menjadi kode yang sesuai dengan aturan bahasa pemrograman yang digunakan.

Jadi, *kalimat terstruktur* adalah titik awal yang baik dalam merancang algoritma karena menawarkan fleksibilitas dan kemudahan pemahaman, tetapi harus diperhalus lebih lanjut agar bisa diimplementasikan dalam kode program.

Berikut elaborasi mengenai *Pseudo Code* dan penjelasan lebih dalam tentang penerapannya dalam pemrograman:

2.2.2 Pseudo Code

Definisi: *Pseudo Code* adalah cara penulisan algoritma yang menyerupai bahasa pemrograman, namun tidak mengikuti aturan ketat dari bahasa pemrograman apapun. Dengan menulis algoritma dalam bentuk *pseudo code*, kita dapat menjembatani antara rancangan logika algoritma dan penerjemahan algoritma tersebut ke dalam kode program yang sesungguhnya. Tujuan utama dari *pseudo code* adalah untuk memudahkan proses konversi dari langkah-langkah logis menjadi sintaks pemrograman.

Pseudo code banyak menggunakan elemen-elemen umum dalam pemrograman seperti perintah *if-else*, *while*, *for*, dan lain-lain, yang biasanya diambil dari berbagai bahasa pemrograman. Meskipun tidak dapat dijalankan oleh komputer, *pseudo code* menawarkan kejelasan dan struktur yang mendekati bahasa pemrograman sehingga programmer lebih mudah memahami dan menerjemahkannya ke dalam kode sebenarnya.

Ciri-Ciri:

- Menggunakan bahasa yang mirip dengan bahasa pemrograman, tetapi tetap dalam bentuk yang bisa dipahami oleh manusia.
- Tidak mengikuti aturan sintaksis ketat seperti dalam bahasa pemrograman, sehingga lebih fleksibel dan mudah dimodifikasi.
- Terdiri dari dua bagian penting: **Deklarasi** (untuk mendeklarasikan variabel dan tipe data) dan **Deskripsi** (alur langkah-langkah atau instruksi logika algoritma).

Bagian-Bagian Pseudo Code:

1. **Deklarasi**: Pada bagian ini, kita mendefinisikan variabel yang digunakan dalam algoritma beserta tipe datanya. Ini serupa dengan bagian deklarasi variabel dalam bahasa pemrograman, di mana kita memberi tahu komputer jenis data apa yang akan digunakan.
2. **Deskripsi**: Di sini, kita menuliskan langkah-langkah algoritma, menggunakan elemen-elemen logika seperti operasi matematika, kondisi (if-else), perulangan (loops), dan sebagainya. Deskripsi ini akan diubah menjadi kode program sesungguhnya ketika algoritma sudah siap diimplementasikan.

Kelebihan Pseudo Code:

- **Lebih dekat dengan bahasa pemrograman**: *Pseudo code* lebih mudah dikonversi ke dalam kode program yang sebenarnya karena bentuknya sudah menyerupai sintaks pemrograman.
- **Fleksibilitas**: Walaupun menyerupai bahasa pemrograman, *pseudo code* tetap lebih bebas dalam struktur dan tidak terikat aturan ketat, sehingga bisa ditulis dengan berbagai gaya tanpa harus memperhatikan kesalahan sintaksis.
- **Lebih mudah untuk dipahami oleh programmer**: Karena menggunakan istilah yang dekat dengan bahasa pemrograman, *pseudo code* sangat membantu programmer untuk membayangkan bagaimana algoritma akan diimplementasikan dalam kode.

Kekurangan Pseudo Code:

- **Tidak bisa dieksekusi langsung**: Meskipun mendekati bahasa pemrograman, *pseudo code* tidak bisa dijalankan oleh komputer dan harus dikonversi terlebih dahulu ke dalam bahasa pemrograman yang sebenarnya.
- **Variabilitas penulisan**: Berbeda dengan bahasa pemrograman yang memiliki aturan sintaks baku, *pseudo code* bisa ditulis dengan gaya yang berbeda oleh setiap orang, sehingga kadang bisa membingungkan jika standar penulisan tidak konsisten.

Contoh: Berikut adalah contoh *pseudo code* sederhana untuk algoritma penjumlahan dua angka:

```
Algoritma penjumlahan
Deklarasi:
  a, b, hasil: integer

Deskripsi:
  input a
  input b
  hasil <- a + b
  print hasil
```

Penjelasan:

- Pada bagian **Deklarasi**, kita mendeklarasikan variabel `a`, `b`, dan `hasil` sebagai tipe data integer (bilangan bulat).
- Pada bagian **Deskripsi**, kita menuliskan alur langkah-langkah algoritma. Pertama, kita meminta input dari pengguna untuk variabel `a` dan `b`. Lalu, kita menjumlahkan kedua variabel tersebut dan menyimpan hasilnya ke variabel `hasil`. Terakhir, kita menampilkan hasil penjumlahan.

Aplikasi dalam Pemrograman: *Pseudo code* sangat berguna dalam merancang algoritma sebelum langsung menuliskannya dalam bahasa pemrograman. Misalnya, ketika mengembangkan program yang lebih kompleks seperti pengurutan data atau pencarian informasi, menuliskannya terlebih dahulu dalam *pseudo code* dapat membantu programmer untuk mengatur alur logika sebelum berurusan dengan detail teknis dari bahasa pemrograman yang digunakan.

Dengan menuliskan algoritma dalam bentuk *pseudo code*, kita dapat membuat rancangan yang mudah dipahami dan diimplementasikan oleh tim, tanpa harus khawatir tentang kesalahan sintaks atau aturan

spesifik dari bahasa pemrograman.

2.2.3 Diagram Alur *(Flowchart)*

Flowchart adalah alat visual yang digunakan untuk merepresentasikan alur kerja atau proses dalam bentuk diagram. Dalam dunia pemrograman dan sistem, flowchart digunakan untuk merencanakan, menganalisis, dan memahami langkah-langkah yang diperlukan dalam menyelesaikan suatu tugas atau masalah.[5]

2.2.3.1 Model dan Jenis Flowchart Flowchart atau bagan alur merupakan diagram yang menggunakan simbol-simbol grafis untuk merepresentasikan urutan langkah, proses, atau aktivitas dalam menyelesaikan suatu masalah. Ada dua model utama flowchart yang umum digunakan, yaitu:

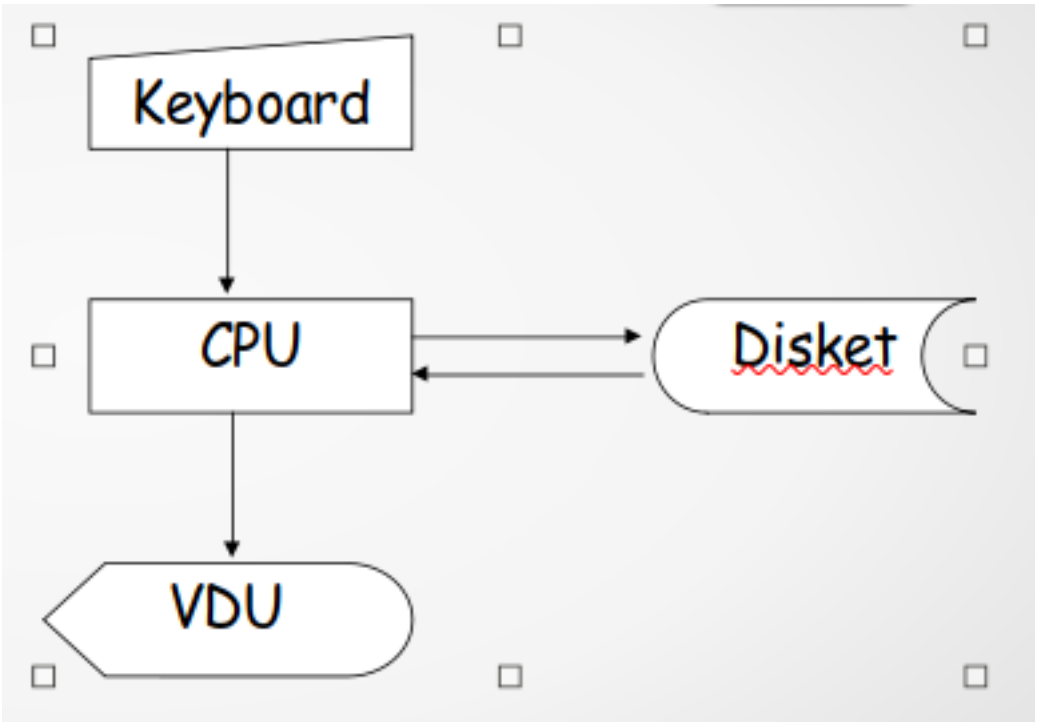
1. **System Flowchart**
2. **Program Flowchart**

2.2.3.1.1 System Flowchart (Bagan Alur Sistem) **System flowchart** adalah jenis flowchart yang digunakan untuk menggambarkan keseluruhan sistem peralatan komputer yang terlibat dalam proses pengolahan data. Bagan ini menunjukkan hubungan antara satu perangkat dengan perangkat lainnya dalam suatu sistem informasi.

Penjelasan lebih lanjut:

- Tidak dimaksudkan untuk menjelaskan urutan langkah-langkah logis dalam memecahkan suatu masalah.
- Lebih berfokus pada gambaran fisik atau infrastruktur sistem, seperti perangkat keras (hardware), perangkat lunak (software), media penyimpanan data, dan jalur komunikasi data.
- Digunakan untuk memvisualisasikan prosedur atau proses kerja dalam sistem yang dibentuk, termasuk alur data antar bagian atau antar perangkat.
- Berguna dalam tahap analisis dan perancangan sistem untuk membantu pemahaman struktur teknologi informasi dalam organisasi.

Contoh: Sistem komputer memiliki keyboard, monitor, sistem proses dan sistem penyimpanan. Hubungan antar sistem dan jalur perpindahan data dapat dituls dengan system flowchart seperti dibawah.



Gambar 2: system image

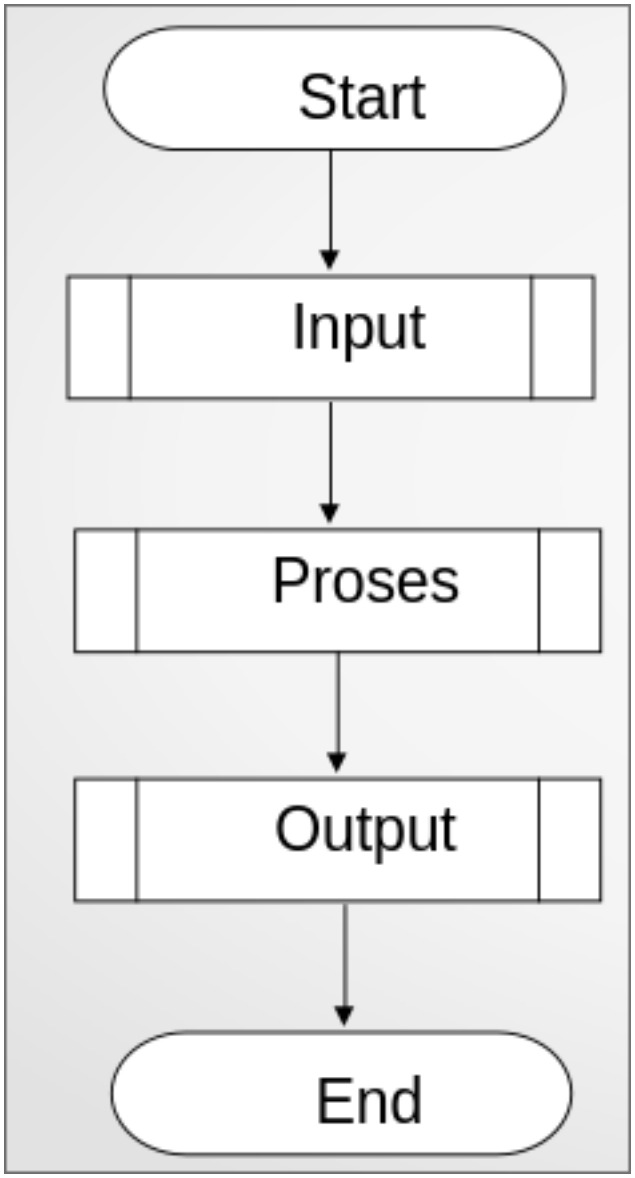
[5] sumber: https://bee.telkomuniversity.ac.id/pengertian-flowchart-fungsi-jenis-simbol-dan-contohnya/

2.2.3.1.2 Program Flowchart (Bagan Alur Program) **Program flowchart** adalah jenis flowchart yang digunakan untuk menggambarkan secara visual urutan logika atau alur dari suatu program komputer. Bagan ini sangat penting dalam dunia pemrograman karena membantu programmer dalam merancang solusi sebelum menulis kode.

Ada dua metode utama dalam membuat program flowchart, yaitu:

Conceptual Flowchart

- Merupakan bagan alur yang menggambarkan alur pemecahan masalah secara umum atau konseptual.
- Tidak terlalu detail dalam hal langkah-langkah teknis, tetapi memberikan gambaran besar tentang cara suatu masalah akan diselesaikan melalui program.
- Digunakan pada tahap awal perencanaan program untuk menyusun ide dan logika dasar.
- Contoh:



Gambar 3: conceptual flowchart

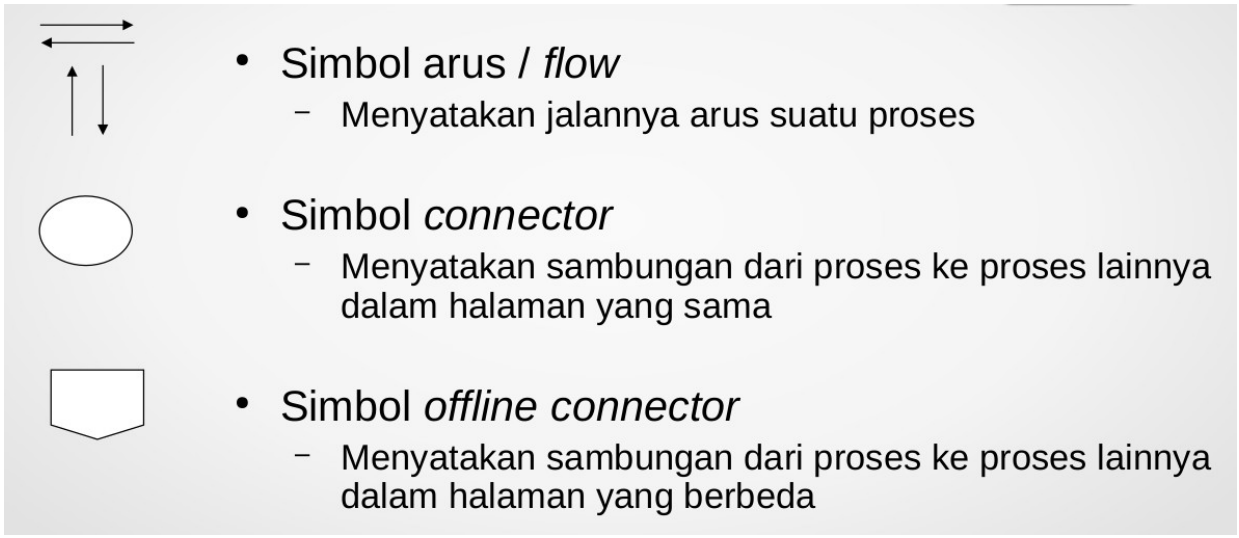
Detail Flowchart

- Merupakan bagan alur yang menggambarkan proses pemecahan masalah secara lengkap dan rinci.
- Semua langkah, kondisi, dan operasi ditampilkan secara spesifik sesuai dengan alur program yang akan dibuat.
- Sangat berguna untuk meminimalkan kesalahan logika saat penulisan kode dan mempermudah proses debugging.

Dengan kedua jenis flowchart tersebut, seorang programmer dapat lebih mudah memahami serta menerjemahkan algoritma menjadi program yang dapat dijalankan oleh komputer.

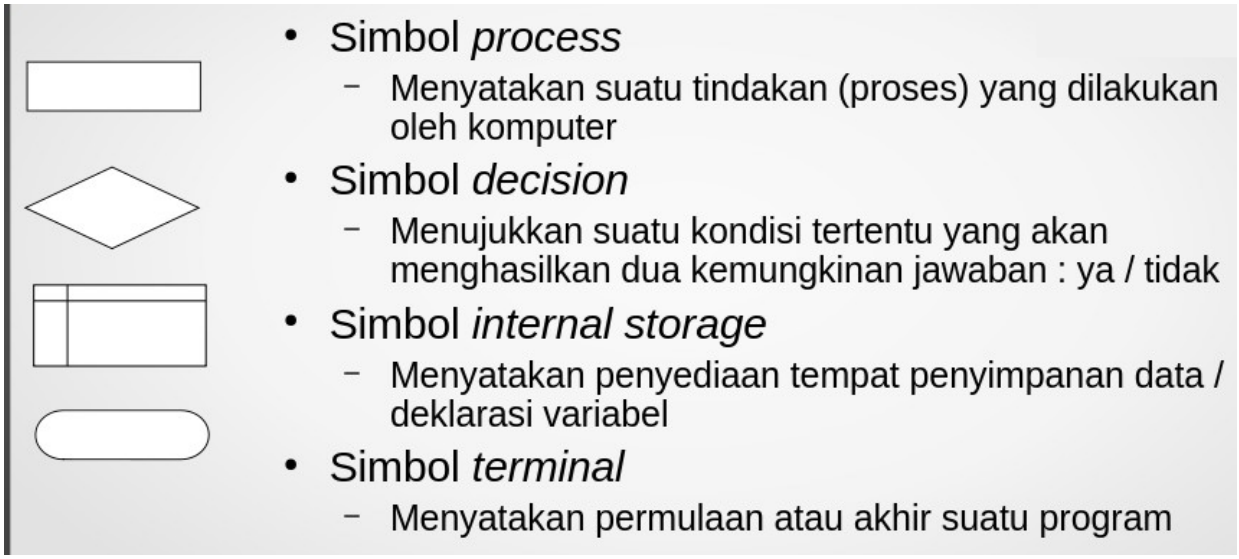
2.2.3.2 Simbol-simbol dalam Flowchart

2.2.3.2.1 Flow direction symbols Digunakan untuk menghubungkan simbol satu dengan yang lain, disebut juga connecting line Macamnya:



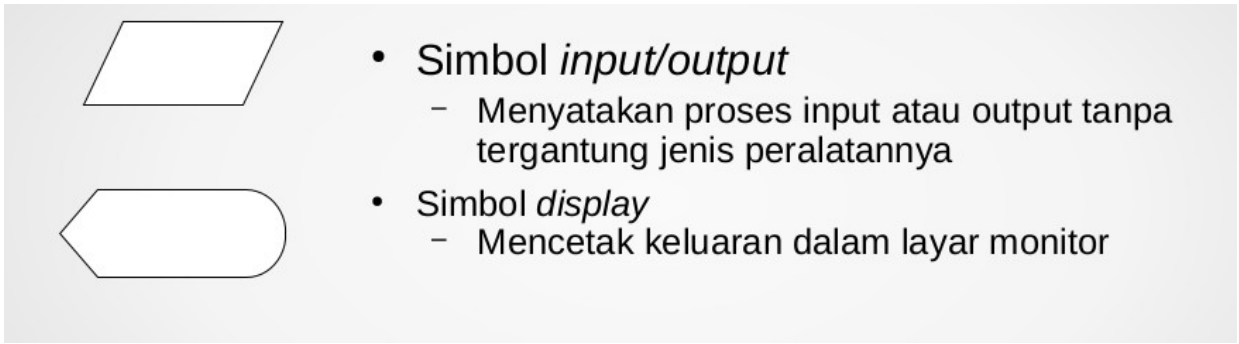
Gambar 4: simbol_arus

2.2.3.2.2 Processing symbols Menunjukan jenis operasi pengolahan dalam suatu proses / prosedur Macamnya:



Gambar 5: simbol_proses

2.2.3.2.3 Input / Output symbols Menunjukkan jenis input atau output dari suatu sistem Macamnya:



Gambar 6: simbol_io

3 Bahasa Pemrograman

3.1 Cara Kerja Komputer

Komputer pada dasarnya adalah sebuah mesin elektronik yang hanya dapat menjalankan operasi-operasi sederhana, seperti penjumlahan, pengurangan, perbandingan, atau pemindahan data. Namun, dengan kecepatan pemrosesan yang sangat tinggi, operasi sederhana ini dapat dirangkai menjadi tugas-tugas yang kompleks.

Komputer tidak memiliki kecerdasan alami. Tanpa instruksi dari manusia, komputer hanyalah perangkat keras yang tidak dapat melakukan apa pun. Oleh karena itu, diperlukan **program** untuk mengarahkan komputer melakukan tugas tertentu. Program inilah yang menjadikan komputer bermanfaat, baik untuk perhitungan matematika, pengolahan teks, pengendalian robot, hingga menjalankan aplikasi sehari-hari.

3.2 Bahasa Natural vs Bahasa Pemrograman

Sebelum membahas bahasa pemrograman, mari kita pahami terlebih dahulu apa itu bahasa.

Bahasa adalah sarana untuk mengekspresikan atau menyampaikan pemikiran, ide, maupun instruksi. Manusia menggunakan bahasa alami (natural language) seperti Bahasa Indonesia atau Bahasa Inggris untuk berkomunikasi.

Komputer pun memiliki "bahasanya" sendiri. Bahasa komputer paling dasar disebut **bahasa mesin (machine language)**, yaitu kumpulan instruksi biner berupa angka 0 dan 1.

Bahasa mesin sangat sulit dipahami manusia karena terlalu sederhana dan tidak menyerupai bahasa alami. Komputer dapat diibaratkan seperti hewan sirkus: ia hanya bisa melakukan apa yang diperintahkan, tanpa inisiatif. Semua kemampuan yang terlihat "pintar" pada komputer sebenarnya adalah hasil dari instruksi yang dibuat oleh manusia.

Kumpulan instruksi yang dapat dijalankan oleh sebuah komputer disebut **Instruction Set** (daftar instruksi).

3.3 Komponen Bahasa

Setiap bahasa, baik bahasa manusia maupun bahasa pemrograman, memiliki komponen penting yang serupa:

Komponen	Penjelasan
Alphabet	Simbol-simbol dasar yang digunakan untuk membentuk bahasa. Dalam bahasa pemrograman, alphabet bisa berupa huruf, angka, maupun simbol khusus.
Lexis	Kumpulan kata atau istilah dalam sebuah bahasa. Dalam pemrograman, ini berupa kata kunci (keyword) dan identifier.
Syntax	Aturan penyusunan simbol atau kata sehingga membentuk pernyataan yang valid.
Semantics	Aturan yang menentukan makna dari susunan kata atau instruksi yang dibuat.

Contohnya, dalam Python kita menuliskan:

```
print("Halo Dunia")
```

- **Alphabet**: huruf `p`, `r`, `i`, `n`, `t`, tanda kurung, dan tanda kutip.
- **Lexis**: kata `print` adalah keyword bawaan Python.
- **Syntax**: penulisan fungsi harus menggunakan tanda kurung.
- **Semantics**: instruksi ini bermakna "tampilkan teks Halo Dunia ke layar".

3.4 Evolusi Bahasa Pemrograman

Awalnya, manusia berinteraksi langsung dengan komputer menggunakan **bahasa mesin**. Namun, bahasa ini sulit dipahami dan rawan kesalahan.

Untuk mempermudah, dibuatlah **bahasa rakitan (assembly language)** yang menggunakan singkatan (mnemonic) seperti `MOV`, `ADD`, atau `JMP` sebagai pengganti kode biner.

Meski assembly lebih mudah dipahami daripada bahasa mesin, tetap saja terlalu teknis. Maka kemudian berkembanglah **bahasa pemrograman tingkat tinggi** yang lebih dekat dengan bahasa manusia, seperti C, Java, Python, atau PHP.

Kode yang ditulis oleh manusia dalam bahasa pemrograman disebut **source code**, dan disimpan dalam sebuah **source file** (misalnya: `program.py`, `main.cpp`). Source code ini tidak bisa langsung dijalankan komputer, tetapi harus diterjemahkan terlebih dahulu ke dalam bahasa mesin.

3.5 Dari Bahasa Pemrograman ke Bahasa Mesin

Agar source code dapat dieksekusi oleh komputer, ia harus diterjemahkan ke bahasa mesin. Proses ini dapat dilakukan dengan dua pendekatan utama:

1. **Kompilasi (Compilation)** Source code diterjemahkan sekaligus menjadi file program yang bisa dijalankan secara langsung.
2. **Interpretasi (Interpretation)** Source code dibaca dan dijalankan baris demi baris setiap kali program dijalankan.

Ada pula pendekatan **Hybrid** yang menggabungkan keduanya, seperti Java dan C#. Dalam bahasa ini, source code dikompilasi terlebih dahulu menjadi *bytecode*, lalu dijalankan oleh mesin virtual (misalnya Java Virtual Machine). Hal ini memungkinkan portabilitas yang lebih tinggi.

3.5.1 Compilation

Dalam metode kompilasi:

- Source code diterjemahkan sekali, menghasilkan file **executable** (misalnya `program.exe` di Windows atau `program` di Linux).
- File executable ini dapat dijalankan berkali-kali tanpa perlu menerjemahkan ulang.
- Jika source code diubah, maka harus dikompilasi ulang.
- Program penerjemah yang digunakan disebut **compiler**.

Contoh bahasa yang menggunakan kompilasi:

- C
- C++
- Rust
- Go
- Pascal

3.5.2 Interpretation

Dalam metode interpretasi:

- Kode program tidak diterjemahkan sekaligus, melainkan baris demi baris saat dijalankan.
- Proses ini dilakukan oleh **interpreter**.
- Program yang menggunakan interpretasi biasanya harus didistribusikan bersama interpreter.

Contoh bahasa berbasis interpretasi:

- Python
- JavaScript
- PHP
- Ruby

3.5.3 Perbandingan Compilation dan Interpretation

Aspek	Compilation	Interpretation
Kecepatan Eksekusi	Lebih cepat karena sudah menjadi file biner.	Lebih lambat karena harus diterjemahkan saat runtime.
Distribusi	Hanya file hasil kompilasi yang perlu dibagikan.	Harus menyertakan interpreter di komputer pengguna.
Portabilitas	Harus dikompilasi ulang untuk sistem operasi atau arsitektur yang berbeda.	Cukup membutuhkan interpreter yang tersedia di berbagai sistem.
Keamanan Source Code	Source code tidak langsung terlihat.	Source code biasanya terbuka bagi pengguna.

3.6 Tingkatan Bahasa Pemrograman

Bahasa pemrograman dapat diklasifikasikan menjadi tiga tingkatan:

1. **Bahasa Tingkat Rendah (Low-Level Language)**
   - Sangat dekat dengan mesin.
   - Biasanya berupa bahasa rakitan (assembly).
   - Contoh instruksi: `MOV`, `SUB`, `JMP`.
   - Digunakan untuk pemrograman sistem, driver perangkat keras, atau optimasi performa.
2. **Bahasa Tingkat Menengah (Middle-Level Language)**
   - Menggabungkan konsep bahasa rendah dan tinggi.
   - C adalah contoh klasik, karena dapat digunakan untuk mengakses hardware sekaligus menulis program aplikasi.
3. **Bahasa Tingkat Tinggi (High-Level Language)**
   - Mendekati bahasa manusia dan lebih mudah dipahami.
   - Banyak menyediakan abstraksi sehingga programmer tidak perlu memikirkan detail mesin.
   - Contoh: Python, Java, C++, PHP, Kotlin, JavaScript.

3.7 Penutup

Bahasa pemrograman merupakan jembatan antara manusia dan komputer. Tanpa bahasa pemrograman, komputer tidak akan bisa digunakan untuk menyelesaikan tugas-tugas kompleks yang saat ini sudah menjadi bagian penting dalam kehidupan sehari-hari.

Pemahaman tentang sejarah, evolusi, serta tingkatan bahasa pemrograman membantu kita menyadari bahwa perkembangan teknologi tidak terjadi seketika, melainkan melalui proses panjang. Dari kode biner sederhana, kita kini memiliki bahasa pemrograman tingkat tinggi yang memudahkan siapa pun belajar menjadi seorang programmer.

4 Bahasa Pemrograman Python

4.1 Apa itu Python?

Python adalah bahasa pemrograman interpretatif yang bersifat **multiguna (general-purpose programming language)**. Artinya, Python dapat digunakan untuk berbagai macam kebutuhan, mulai dari pengembangan aplikasi desktop, aplikasi web, hingga pengolahan data dan kecerdasan buatan.

Python memiliki filosofi utama: **fokus pada kejelasan kode**. Bahasa ini dirancang agar mudah dipelajari, mudah dibaca, dan mudah digunakan kembali. Karena sifatnya yang sederhana namun kuat, Python menjadi salah satu bahasa pemrograman yang paling populer di dunia saat ini.

Python pertama kali diciptakan oleh **Guido van Rossum** pada akhir tahun 1980-an di Centrum Wiskunde & Informatica (CWI), Belanda. Nama "Python" bukan berasal dari ular, melainkan terinspirasi dari acara komedi Inggris **"Monty Python's Flying Circus"**.

Rilis resmi pertama Python terjadi pada tahun **1991**. Sejak itu Python terus berkembang dengan komunitas yang sangat aktif. Saat ini, versi terbaru Python yang stabil adalah **Python 3.x** (per Agustus 2025, versi stabil terbaru adalah **Python 3.13**). Versi Python 2 sudah **tidak lagi didukung (end of life)** sejak Januari 2020.

4.2 Tujuan dan Filosofi Python

Pada tahun 1999, Guido van Rossum merumuskan tujuan utama Python sebagai berikut:

1. Python harus **mudah dan intuitif**, tetapi tetap **powerful**.
2. Python harus **open source**, sehingga dapat digunakan dan dikembangkan oleh siapa pun.
3. Kode Python harus mudah dibaca dan dipahami, layaknya bahasa Inggris sederhana.
4. Python harus cocok untuk menyelesaikan **tugas sehari-hari**.

Prinsip dasar ini kemudian dituangkan dalam filosofi Python yang dikenal dengan **The Zen of Python**, yang dapat dilihat langsung di interpreter Python dengan mengetik:

```
import this
```

4.3 Penggunaan Python

Python digunakan secara luas di berbagai bidang, antara lain:

- **Aplikasi Desktop (GUI):** Menggunakan pustaka seperti Tkinter, PyQt, atau Kivy.
- **Pengembangan Web:** Framework populer seperti **Flask** dan **Django** memungkinkan pembuatan aplikasi web modern dengan cepat.
- **Pengolahan Data & Analisis Ilmiah:** Pustaka seperti **NumPy**, **Pandas**, dan **Matplotlib** menjadikan Python bahasa utama dalam data science.
- **Kecerdasan Buatan (Artificial Intelligence) & Machine Learning:** Dengan pustaka seperti **TensorFlow**, **PyTorch**, dan **scikit-learn**, Python mendominasi bidang AI.
- **Internet of Things (IoT):** Python dapat dijalankan pada perangkat kecil seperti **Raspberry Pi** atau **MicroPython** pada mikrokontroler.
- **Automasi & Scripting:** Python banyak digunakan untuk menulis skrip otomatisasi di sistem operasi, server, maupun jaringan.

4.4 Kelebihan dan Kekurangan Python

Kelebihan Python:

- **Mudah dipelajari:** Sintaks Python sederhana dan mirip bahasa manusia.
- **Mudah diajarkan:** Sangat cocok sebagai bahasa pertama untuk pemula.
- **Mudah digunakan:** Banyak pustaka bawaan dan eksternal yang siap pakai.
- **Mudah dipahami:** Kode lebih ringkas, lebih sedikit tanda baca dibanding C/C++ atau Java.
- **Mudah didapatkan & diinstal:** Python gratis, tersedia di berbagai platform (Windows, macOS, Linux).

- **Komunitas besar:** Dukungan dokumentasi dan forum sangat banyak.

Kekurangan Python:

- **Lebih lambat dibanding bahasa kompilasi** (misalnya C atau Rust), karena merupakan bahasa interpretatif.
- **Kurang cocok untuk aplikasi yang membutuhkan performa sangat tinggi** (misalnya game engine atau sistem real-time kritis).
- **Manajemen memori otomatis (garbage collection)** kadang membuat performa tidak stabil untuk tugas tertentu.

4.5 Python 2 vs Python 3

Sebelum 2008, Python yang banyak digunakan adalah Python 2. Namun, pada tahun 2008, Python 3 dirilis dengan banyak perubahan besar yang tidak sepenuhnya kompatibel dengan Python 2.

Sejak **Januari 2020**, Python 2 **tidak lagi didukung**. Semua pengembangan Python berfokus pada **Python 3**.

Dalam pembelajaran dan praktikum pemrograman dasar, **Python 3** selalu digunakan karena:

- Mendapatkan dukungan penuh.
- Lebih aman.
- Lebih banyak pustaka modern yang kompatibel.

4.6 Persiapan Menggunakan Python

Untuk menggunakan Python, kita perlu menginstal **Python Runtime**.

- **Windows:** Python dapat diunduh dari python.org dan dipasang langsung.
- **Linux & macOS:** Umumnya Python sudah tersedia bawaan, namun bisa diperbarui melalui package manager.

Setelah instalasi, Python dapat dijalankan melalui:

- **IDLE (Integrated Development and Learning Environment)** bawaan Python.
- **Terminal/Command Prompt** dengan mengetik:

```
python
```

atau

```
python3
```

4.7 Contoh Program Sederhana

Berikut contoh program Python untuk menerima input nama dan menampilkan sapaan:

```
nama = input("Masukkan nama: ")
print("Halo", nama)
```

Penjelasan kode:

- `input()` digunakan untuk membaca input dari pengguna.
- `print()` digunakan untuk menampilkan output ke layar.

Jika pengguna mengetik **Budi**, maka hasilnya:

```
Masukkan nama: Budi
Halo Budi
```

5 Dasar Pemrograman Python

5.1 Pendahuluan

Python adalah bahasa pemrograman tingkat tinggi yang mudah dipelajari dan banyak digunakan untuk berbagai keperluan seperti pengembangan web, analisis data, kecerdasan buatan, dan automasi. Bab ini akan membahas konsep-konsep dasar pemrograman Python yang perlu dikuasai oleh pemula.

5.2 Statement (Pernyataan)

Statement adalah instruksi atau perintah yang diberikan kepada Python untuk melakukan suatu tindakan. Setiap statement biasanya ditulis dalam satu baris kode.

```
print("ini adalah statement")
nama = "Saiful"
print(nama)
```

Penjelasan:

- Baris pertama menggunakan fungsi `print()` untuk menampilkan teks "ini adalah statement"
- Baris kedua membuat variabel `nama` dan menyimpan nilai "Saiful" di dalamnya
- Baris ketiga menampilkan nilai yang disimpan dalam variabel `nama`

Output :

```
ini adalah statement
Saiful
```

Setiap statement dalam Python dieksekusi secara berurutan dari atas ke bawah. Python membaca dan menjalankan kode baris demi baris, sehingga urutan penulisan kode sangat penting.

5.3 Indentasi dan Scope

Indentasi (penulisan menjorok ke dalam) dalam Python bukan hanya masalah estetika, tetapi memiliki makna penting yang menentukan **scope** (ruang lingkup) suatu blok kode.

```
print("Halo")
for i in range(3):
    print(i)  # bagian dari loop for

x = 5
if x < 4:
    print("kecil")  # bagian dari if
    print(x)        # masih bagian dari if
print("selesai")    # sudah di luar blok if
```

Penjelasan:

- Statement yang sejajar dengan margin kiri (tanpa indentasi) adalah main scope
- Statement yang memiliki indentasi (biasanya 4 spasi) adalah bagian dari blok kode di atasnya
- Loop `for` dan percabangan `if` memerlukan indentasi untuk menandai blok kode yang menjadi bagiannya

Output :

```
Halo
0
1
2
selesai
```

Python menggunakan indentasi untuk mengetahui kode mana yang termasuk dalam blok tertentu. Ini berbeda dengan bahasa pemrograman lain yang menggunakan kurung kurawal `{}` untuk menandai blok kode. Konsistensi dalam penggunaan indentasi sangat penting agar kode dapat berjalan dengan benar.

5.4 Komentar

Komentar adalah teks dalam kode yang tidak dieksekusi oleh Python. Komentar digunakan untuk memberikan penjelasan, dokumentasi, atau menonaktifkan sementara bagian kode.

```
# komentar adalah teks yang tidak akan dibaca oleh python
print("setelah ini komentar")  # komentar bisa ditulis setelah statement
```

Penjelasan:

- Komentar diawali dengan tanda `#`
- Teks setelah tanda `#` tidak akan diproses oleh Python
- Komentar dapat ditulis di baris tersendiri atau setelah statement

Komentar sangat penting untuk membuat kode yang mudah dipahami, baik oleh diri sendiri maupun orang lain. Dengan komentar yang baik, kita dapat menjelaskan logika atau tujuan dari suatu bagian kode.

5.5 Fungsi Dasar: print()

Fungsi `print()` digunakan untuk menampilkan output ke terminal atau konsol. Fungsi ini sangat penting untuk debugging dan menampilkan informasi kepada pengguna.

5.5.1 Penggunaan Dasar print()

```
print("Halo Dunia")
x = "Bumi"
print("Selamat datang di", x)
```

Output :

```
Halo Dunia
Selamat datang di Bumi
```

5.5.2 Mengubah Karakter Akhir (end)

Secara default, `print()` akan menambahkan newline (`\n`) di akhir output, tetapi kita dapat mengubahnya dengan parameter `end` .

```
print("Perkenalkan, nama saya", end=" ")
print("Saiful Habib")

print("Kab. Magelang", end="__--__")
print("Jawa", end="-->><<--")
print("Tengah")
```

Output :

```
Perkenalkan, nama saya Saiful Habib
Kab. Magelang__--__Jawa-->><<--Tengah
```

5.5.3 Mengubah Pemisah Nilai (sep)

Secara default, `print()` memisahkan multiple values dengan spasi, tetapi kita dapat mengubahnya dengan parameter `sep` .

```
a = 3
b = "kali"
c = "3 sama dengan"
print(a, b, c, 9)  # default separator

print(a, b, c, 9, sep=")]>-<[(")  # custom separator
```

Output :

```
3 kali 3 sama dengan 9
3)]>-<[(kali)]>-<[(3 sama dengan)]>-<[(9
```

Fungsi `print()` sangat fleksibel dan dapat disesuaikan dengan kebutuhan output yang diinginkan.

5.6 Fungsi Dasar: input()

Fungsi `input()` digunakan untuk menerima masukan dari pengguna melalui keyboard. Input tersebut kemudian dapat diproses lebih lanjut dalam program.

```
nama = input()  # input tanpa petunjuk
print("halo", nama)

kelas = input("masukkan kelas: ")  # input dengan petunjuk
print("kamu kelas", kelas)
```

Contoh interaksi dan output:

```
(users mengetik: Ihsan)
halo Ihsan
masukkan kelas: 10 TJTK1
kamu kelas 10 TJKT1
```

Fungsi `input()` selalu mengembalikan nilai dalam bentuk string, meskipun pengguna memasukkan angka. Jika membutuhkan tipe data lain, kita perlu mengkonversinya.

5.7 Variabel dan Aturan Penamaan

Variabel adalah tempat penyimpanan data dalam memori yang memiliki nama dan nilai. Nama variabel digunakan untuk mengakses nilai yang disimpan.

```
a = 1
b = 2
c = a + b
print(a, "+", b, "=", c)
```

Output :

```
1 + 2 = 3
```

5.7.1 Aturan Penamaan Variabel

Python memiliki aturan tertentu dalam penamaan variabel:

1. **Hanya boleh mengandung huruf, angka, dan underscore (_)**

   ```
   nama = "ihsan"
   Nama2 = "upin"
   nama_3 = "ipin"
   # $nama4 = "fizi"  # Error: simbol $ tidak diperbolehkan
   ```

2. **Tidak boleh diawali dengan angka**

   ```
   3serangkai = "Douwes Dekker"  # Error: tidak boleh diawali angka
   ```

3. **Tidak boleh mengandung spasi**

   ```
   smk it = "Ihsanul Fikri"  # Error: spasi tidak diperbolehkan
   ```

4. **Tidak boleh menggunakan kata kunci Python (reserved words)**

   ```
   for = 3  # Error: for adalah kata kunci Python
   ```

Python adalah case-sensitive, artinya variabel `Nama` dan `nama` dianggap sebagai dua variabel yang berbeda.

5.8 Tipe Data

Setiap nilai dalam Python memiliki tipe data tertentu. Pemahaman tentang tipe data penting untuk operasi dan pengolahan data yang benar dalam program.

5.8.1 Tipe Data Dasar

Python memiliki beberapa tipe data dasar yang penting untuk dipahami:

1. **Integer (bilangan bulat)**:

```
a = 4  # integer (bilangan bulat)
```

   - Digunakan untuk menyimpan bilangan bulat positif/negatif
   - Contoh: 0, 10, -5, 1000
   - Tidak memiliki batasan ukuran (kecuali memori sistem)

2. **Float (bilangan pecahan)**:

```
b = 4.4  # float (bilangan pecahan)
```

   - Digunakan untuk menyimpan bilangan real/pecahan
   - Contoh: 3.14, -0.001, 2.0
   - Memiliki presisi sekitar 15-17 digit desimal

3. **String (teks)**:

```
c = "4"  # string (teks)
```

   - Digunakan untuk menyimpan data teks
   - Diapit tanda kutip (bisa tunggal ' atau ganda ")
   - Contoh: "hello", '123', "Python"
   - Meskipun berisi angka, jika diapit kutip maka tipe datanya string

4. **Boolean (True/False)**:

```
d = (4 == 4)  # boolean (True/False)
```

   - Hanya memiliki dua nilai: True atau False
   - Sering digunakan dalam percabangan dan perulangan
   - Hasil dari operasi perbandingan (==, !=, >, <, dll)

Untuk mengecek tipe data suatu variabel, kita bisa menggunakan fungsi `type()`:

```
print("variable a bertipe", type(a), "dengan nilai", a)
print("variable b bertipe", type(b), "dengan nilai", b)
print("variable c bertipe", type(c), "dengan nilai", c)
print("variable d bertipe", type(d), "dengan nilai", d)
```

Output :

```
variable a bertipe <class 'int'> dengan nilai 4
variable b bertipe <class 'float'> dengan nilai 4.4
variable c bertipe <class 'str'> dengan nilai 4
variable d bertipe <class 'bool'> dengan nilai True
```

Fungsi `type()` mengembalikan objek tipe data dari variabel yang diperiksa. Dalam Python, semua tipe data sebenarnya adalah objek (kelas), sehingga outputnya berupa `<class 'tipe_data'>`.

Output :

```
variable a bertipe <class 'int'> dengan nilai 4
variable b bertipe <class 'float'> dengan nilai 4.4
variable c bertipe <class 'str'> dengan nilai 4
variable d bertipe <class 'bool'> dengan nilai True
```

5.8.2 Konversi Tipe Data

Kadang kita perlu mengubah tipe data dari satu bentuk ke bentuk lain, seperti mengubah string menjadi integer:

```
# Mengubah input string menjadi integer
a = int(input("masukkan a: "))
b = int(input("masukkan b: "))
c = a + b
print(a, "+", b, "=", c)
```

Contoh interaksi dan output:

```
masukkan a: 5
masukkan b: 3
5 + 3 = 8
```

Fungsi konversi tipe data yang umum digunakan:

- `int()` : mengubah menjadi integer
- `float()` : mengubah menjadi float
- `str()` : mengubah menjadi string
- `bool()` : mengubah menjadi boolean

5.9 Kesimpulan

Dalam bab ini, kita telah mempelajari dasar-dasar pemrograman Python meliputi statement, indentasi, komentar, fungsi dasar `print()` dan `input()`, variabel, dan tipe data. Konsep-konsep ini merupakan fondasi penting untuk mempelajari topik-topik yang lebih advanced dalam Python.

Pemahaman yang kuat tentang dasar-dasar ini akan memudahkan kita dalam mempelajari struktur kontrol, fungsi, modul, dan konsep pemrograman lainnya yang akan dibahas dalam bab-bab selanjutnya.

5.10 Latihan

1. Buat program yang meminta nama dan usia pengguna, kemudian tampilkan pesan sambutan yang menyertakan informasi tersebut
2. Buat program kalkulator sederhana yang meminta dua angka dan menampilkan hasil penjumlahan, pengurangan, perkalian, dan pembagian
3. Coba berbagai variasi parameter `sep` dan `end` pada fungsi `print()` untuk memahami fungsinya dengan lebih baik

Dengan berlatih secara konsisten, kemampuan pemrograman Python Anda akan semakin terasah dan berkembang.

6 Operator pada Python

6.1 Pendahuluan

Dalam pemrograman Python, operator merupakan simbol-simbol khusus yang digunakan untuk melakukan operasi tertentu pada nilai dan variabel. Memahami berbagai jenis operator merupakan hal fundamental dalam belajar pemrograman, karena operator memungkinkan kita melakukan manipulasi data, perbandingan, dan operasi logika yang menjadi dasar pembuatan program yang kompleks.

Pada bab ini, kita akan membahas empat jenis operator utama dalam Python: operator aritmatika, operator perbandingan, operator logika, dan operator gabungan. Setiap jenis operator memiliki fungsi dan kegunaan spesifik yang akan kita pelajari dengan detail.

6.2 Operator Aritmatika

Operator aritmatika adalah operator yang digunakan untuk melakukan operasi matematika dasar seperti penjumlahan, pengurangan, perkalian, dan pembagian. Operator-operator ini bekerja pada nilai numerik (integer dan float) dan menghasilkan hasil sesuai operasi yang dilakukan.

Berikut adalah contoh implementasi operator aritmatika dalam Python:

```
# Meminta input dari pengguna
a = int(input("a: "))
b = int(input("b: "))

# Operasi aritmatika dasar
c = a + b  # Penjumlahan
d = a - b  # Pengurangan
e = a * b  # Perkalian (menggunakan * bukan x)
f = a / b  # Pembagian biasa (hasilnya float)

# Menampilkan hasil operasi
print(a, "+", b, "=", c)
print(a, "-", b, "=", d)
print(a, "*", b, "=", e)
print(a, "/", b, "=", f)
print("tipe dari f:", type(f))  # Menunjukkan tipe data hasil pembagian

# Pembagian bulat (floor division)
g = a // b  # Membulatkan hasil pembagian ke bawah
print(a, "//", b, "=", g)
print("tipe dari g:", type(g))

# Modulus (sisa hasil bagi)
h = a % b  # Menghasilkan sisa dari pembagian
print(a, "%", b, "=", h)

# Pemangkatan
i = a ** b  # Memangkatkan a dengan b
print(a, "**", b, "=", i)
```

Interaksi dan Output:

```
(user memasukkan nilai 7 dan 3)
a: 7
b: 3
7 + 3 = 10
7 - 3 = 4
7 * 3 = 21
7 / 3 = 2.3333333333333335
tipe dari f: <class 'float'>
7 // 3 = 2
tipe dari g: <class 'int'>
```

```
7 % 3 = 1
7 ** 3 = 343
```

Penjelasan Detail:

Operator aritmatika adalah dasar dari semua operasi matematika dalam pemrograman. Python menyediakan berbagai operator aritmatika yang fungsinya mirip dengan matematika biasa, tetapi dengan beberapa penyesuaian sintaks.

- **Penjumlahan (+)**: Menambahkan dua nilai. Contoh: `5 + 3` menghasilkan `8`
- **Pengurangan (-)**: Mengurangi nilai kedua dari nilai pertama. Contoh: `5 - 3` menghasilkan `2`
- **Perkalian (*)**: Mengalikan dua nilai. Dalam pemrograman, simbol yang digunakan adalah asterisk (*) bukan huruf x. Contoh: `5 * 3` menghasilkan `15`
- **Pembagian (/)**: Membagi nilai pertama dengan nilai kedua. Hasilnya selalu bertipe float meskipun kedua operand adalah integer. Contoh: `5 / 2` menghasilkan `2.5`

Python juga menyediakan operator aritmatika khusus:

- **Pembagian Bulat (//)**: Juga disebut floor division, operator ini membagi dua bilangan dan membulatkan hasilnya ke bilangan bulat terdekat yang lebih kecil. Contoh: `5 // 2` menghasilkan `2` (bukan 2.5)
- **Modulus (%)**: Menghasilkan sisa dari pembagian dua bilangan. Operator ini sangat berguna untuk menentukan apakah sebuah bilangan genap atau ganjil, atau untuk operasi siklus. Contoh: `5 % 2` menghasilkan `1`
- **Pemangkatan (**)**: Memangkatkan bilangan pertama dengan bilangan kedua. Contoh: `2 ** 3` menghasilkan `8` (2 pangkat 3)

Penting untuk memahami tipe data hasil dari setiap operasi. Operasi pembagian biasa (/) selalu menghasilkan nilai float, bahkan jika hasilnya adalah bilangan bulat. Sementara operasi pembagian bulat (//) akan menghasilkan integer jika kedua operand adalah integer, tetapi bisa menghasilkan float jika salah satu operand adalah float.

6.3 Operator Perbandingan

Operator perbandingan digunakan untuk membandingkan dua nilai dan menghasilkan nilai boolean (True atau False). Operator ini sangat penting dalam pengambilan keputusan dalam program, seperti dalam struktur percabangan (if-else) dan perulangan.

```
# Operator perbandingan menghasilkan nilai boolean (True/False)
a = 3 == 3  # Sama dengan: True jika kedua nilai sama
b = 4 < 2   # Kurang dari: True jika nilai pertama kurang dari nilai kedua
c = 5 != 7  # Tidak sama dengan: True jika kedua nilai berbeda
d = 9 >= 10 # Lebih besar atau sama dengan: True jika nilai pertama lebih besar atau sama dengan nilai kedua

print(a, b, c, d)  # Menampilkan hasil perbandingan

# Perbedaan antara < dan <=
e = 7 < 7   # Kurang dari: False karena 7 tidak kurang dari 7
f = 7 <= 7  # Kurang dari atau sama dengan: True karena 7 sama dengan 7

print(e, f)  # Menampilkan hasil perbandingan
```

Output:

```
True False True False
False True
```

Penjelasan Detail:

Operator perbandingan memungkinkan kita membuat keputusan dalam program dengan membandingkan nilai-nilai. Hasil dari operasi perbandingan selalu berupa boolean (True atau False), yang kemudian dapat digunakan untuk mengontrol alur program.

Berikut adalah operator perbandingan dalam Python:

- **Sama dengan (==)**: Mengecek apakah dua nilai sama. Contoh: `5 == 5` menghasilkan `True`, `5 == 3` menghasilkan `False`
- **Tidak sama dengan (!=)**: Mengecek apakah dua nilai berbeda. Contoh: `5 != 3` menghasilkan `True`, `5 != 5` menghasilkan `False`
- **Lebih besar dari (>)**: Mengecek apakah nilai pertama lebih besar dari nilai kedua. Contoh: `5 > 3` menghasilkan `True`, `3 > 5` menghasilkan `False`
- **Lebih kecil dari (<)**: Mengecek apakah nilai pertama lebih kecil dari nilai kedua. Contoh: `3 < 5` menghasilkan `True`, `5 < 3` menghasilkan `False`
- **Lebih besar atau sama dengan (>=)**: Mengecek apakah nilai pertama lebih besar atau sama dengan nilai kedua. Contoh: `5 >= 5` menghasilkan `True`, `5 >= 3` menghasilkan `True`, `3 >= 5` menghasilkan `False`
- **Lebih kecil atau sama dengan (<=)**: Mengecek apakah nilai pertama lebih kecil atau sama dengan nilai kedua. Contoh: `3 <= 3` menghasilkan `True`, `3 <= 5` menghasilkan `True`, `5 <= 3` menghasilkan `False`

Perhatikan perbedaan antara operator `=` dan `==`. Operator `=` adalah operator penugasan (assignment) yang digunakan untuk memberikan nilai ke variabel, sementara `==` adalah operator perbandingan yang digunakan untuk mengecek kesamaan dua nilai.

6.4 Operator Logika

Operator logika digunakan untuk menggabungkan atau memanipulasi nilai boolean (True dan False). Operator ini sangat penting untuk membuat kondisi yang lebih kompleks dalam struktur kendali program.

```
# Operator logika bekerja dengan nilai boolean
a = True and True   # AND: True jika kedua nilai True
b = True and False  # AND: False jika salah satu False
c = True or False   # OR: True jika salah satu True
d = False or False  # OR: False jika kedua nilai False

# Menampilkan hasil dengan separator khusus
print(a, b, c, d, sep='__--__')

# Operator NOT untuk membalik nilai boolean
e = not True  # NOT: Membalik nilai True menjadi False
print(e)
```

Output:

```
True__--__False__--__True__--__False
False
```

Penjelasan Detail:

Operator logika memungkinkan kita menggabungkan beberapa kondisi menjadi satu ekspresi yang lebih kompleks. Terdapat tiga operator logika utama dalam Python:

- **AND**: Operator ini menghasilkan True hanya jika kedua operand bernilai True. Jika salah satu atau kedua operand False, hasilnya akan False. Bayangkan seperti syarat yang sangat ketat dimana semua kondisi harus terpenuhi.

A	B	A AND B
True	True	True
True	False	False
False	True	False
False	False	False

- **OR**: Operator ini menghasilkan True jika setidaknya satu operand bernilai True. Hasilnya hanya False jika kedua operand False. Bayangkan seperti syarat yang longgar dimana hanya perlu satu kondisi yang terpenuhi.

A	B	A OR B
True	True	True
True	False	True
False	True	True
False	False	False

- **NOT**: Operator ini membalik nilai boolean dari operandnya. Jika operand True, hasilnya False, dan sebaliknya. Operator ini hanya bekerja pada satu operand (unary operator).

A	NOT A
True	False
False	True

Operator logika sering digunakan dalam struktur percabangan untuk membuat kondisi yang kompleks. Misalnya: `if age >= 18 and has_id:` akan menghasilkan True hanya jika kedua kondisi terpenuhi (usia minimal 18 tahun DAN memiliki kartu identitas).

6.5 Operator Gabungan

Operator gabungan (atau compound operators) adalah operator yang menggabungkan operasi aritmatika atau operasi lainnya dengan operasi penugasan. Operator ini memberikan cara yang lebih singkat untuk melakukan operasi dan penugasan nilai pada variabel yang sama.

```
# Operator gabungan menggabungkan operasi dengan penugasan
a = 0
a += 1  # Sama dengan: a = a + 1
print(a)  # Menampilkan hasil

b = 2
b -= 1  # Sama dengan: b = b - 1
print(b)  # Menampilkan hasil
```

Output:

```
1
1
```

Penjelasan Detail:

Operator gabungan menyederhanakan ekspresi yang melibatkan operasi pada variabel dan penugasan kembali hasilnya ke variabel yang sama. Daripada menulis `a = a + 1`, kita dapat menulis `a += 1` yang lebih ringkas dan mudah dibaca.

Berikut adalah berbagai operator gabungan dalam Python:

- **+= (Penugasan Penjumlahan)**: Menambahkan nilai ke variabel dan menugaskan hasilnya kembali ke variabel tersebut. Contoh: `a += 5` setara dengan `a = a + 5`
- **-= (Penugasan Pengurangan)**: Mengurangi nilai variabel dan menugaskan hasilnya kembali. Contoh: `a -= 3` setara dengan `a = a - 3`
- ***= (Penugasan Perkalian)**: Mengalikan variabel dengan nilai dan menugaskan hasilnya kembali. Contoh: `a *= 2` setara dengan `a = a * 2`
- **/= (Penugasan Pembagian)**: Membagi variabel dengan nilai dan menugaskan hasilnya kembali. Contoh: `a /= 4` setara dengan `a = a / 4`

- **//= (Penugasan Pembagian Bulat)**: Membagi bulat variabel dengan nilai dan menugaskan hasilnya kembali. Contoh: `a //= 2` setara dengan `a = a // 2`
- **%= (Penugasan Modulus)**: Melakukan operasi modulus pada variabel dan menugaskan hasilnya kembali. Contoh: `a %= 3` setara dengan `a = a % 3`
- ****= (Penugasan Pemangkatan)**: Memangkatkan variabel dengan nilai dan menugaskan hasilnya kembali. Contoh: `a **= 2` setara dengan `a = a ** 2`

Operator gabungan tidak hanya membuat kode lebih ringkas, tetapi juga dapat meningkatkan performa dalam beberapa kasus, karena variabel hanya diakses sekali saja.

6.6 Kesimpulan

Operator adalah komponen fundamental dalam Python yang memungkinkan kita melakukan manipulasi data, perbandingan, dan operasi logika. Dalam bab ini, kita telah mempelajari empat jenis operator utama:

1. **Operator aritmatika** untuk melakukan operasi matematika dasar
2. **Operator perbandingan** untuk membandingkan nilai dan menghasilkan boolean
3. **Operator logika** untuk bekerja dengan nilai boolean dan membuat kondisi kompleks
4. **Operator gabungan** untuk menyederhanakan operasi dan penugasan nilai

Pemahaman yang solid tentang berbagai jenis operator ini sangat penting untuk membangun program Python yang efektif dan efisien. Dengan menguasai operator-operator ini, kita telah melangkah lebih jauh dalam perjalanan belajar pemrograman Python.

Pada bab-bab selanjutnya, kita akan melihat bagaimana operator-operator ini digunakan dalam konteks yang lebih praktis, seperti dalam struktur kontrol percabangan dan perulangan, yang merupakan fondasi dari pembuatan program yang dinamis dan interaktif.

6.7 Tugas

Buatlah Program untuk:

- Mengkonversi suhu dari Celcius ke Fahrenheit! (input: suhu)
- Menhitung Luas Sisi dari sebuah Silinder (input: jari-jari dan tinggi)
- Menghitung Volume dari Bola (input: jari2) rumus: `4/3 pi * r * pangkat 3`
- Menghitung Energi Kinetik dari suatu benda (input: massa(m) dan kecepatan(v), rumus: e = setengah m kali v kuadrat)

7 String di Python

7.1 Apa itu String?

String adalah tipe data yang digunakan untuk menyimpan teks. String ditulis dengan tanda kutip tunggal `'...'` atau ganda `"..."`.

Contoh:

```
teks1 = "Halo"
teks2 = 'Python'
print(teks1, teks2)
```

Output:

```
Halo Python
```

7.2 Operasi Dasar pada String

7.2.1 Menggabungkan String (Concatenation)

Dua string bisa digabung dengan operator `+`.

```
nama = "Andi"
sapaan = "Halo " + nama
print(sapaan)
```

Output:

```
Halo Andi
```

7.2.2 Mengulang String

String bisa dikalikan dengan integer (`*`).

```
print("Hi! " * 3)
```

Output:

```
Hi! Hi! Hi!
```

7.3 String sebagai Array (Indexing & Slicing)

String di Python bisa diakses seperti array, yaitu berdasarkan **indeks** (dimulai dari 0).

7.3.1 Indexing

```
teks = "Python"
print(teks[0])   # P
print(teks[3])   # h
```

Output:

```
P
h
```

7.3.2 Slicing

Mengambil bagian tertentu dari string.

```
teks = "Informatika"
print(teks[0:5])   # Infor
print(teks[5:])    # matika
print(teks[:4])    # Info
```

Output:

```
Infor
matika
Info
```

7.4 Fungsi Penting untuk String

7.4.1 `len()`

Menghitung panjang string.

```
teks = "Python"
print(len(teks))   # 6
```

Output:

```
6
```

7.4.2 Ubah ke huruf besar / kecil

- `upper()` -> jadi huruf besar semua
- `lower()` -> jadi huruf kecil semua
- `title()` -> huruf pertama tiap kata besar

```
teks = "belajar python"
print(teks.upper())   # BELAJAR PYTHON
print(teks.lower())   # belajar python
print(teks.title())   # Belajar Python
```

Output:

```
BELAJAR PYTHON
belajar python
Belajar Python
```

7.4.3 Menghapus spasi di awal/akhir

- `strip()` -> hapus spasi di kiri dan kanan
- `lstrip()` -> hapus spasi kiri
- `rstrip()` -> hapus spasi kanan

```
teks = "   Halo Dunia   "
print(teks.strip())   # "Halo Dunia"
```

Output:

```
Halo Dunia
```

7.4.4 Memecah string (`split()`)

Digunakan untuk memecah string berdasarkan pemisah (default: spasi).

```
kalimat = "Saya suka Python"
kata = kalimat.split()
print(kata)   # ['Saya', 'suka', 'Python']
```

Output:

```
['Saya', 'suka', 'Python']
```

Dengan pemisah lain:

```
data = "apel,jeruk,mangga"
buah = data.split(",")
print(buah)   # ['apel', 'jeruk', 'mangga']
```

Output:

```
['apel', 'jeruk', 'mangga']
```

7.4.5 Menggabungkan list jadi string (`join()`)

Kebalikan dari `split()` .

```
kata = ["Belajar", "Python", "Menyenangkan"]
kalimat = " ".join(kata)
print(kalimat)
```

Output:

```
Belajar Python Menyenangkan
```

7.4.6 Mengganti teks (`replace()`)

```
teks = "Saya suka Java"
print(teks.replace("Java", "Python"))
```

Output:

```
Saya suka Python
```

7.4.7 Mengecek isi string

- `startswith()` -> cek apakah string diawali teks tertentu
- `endswith()` -> cek apakah string diakhiri teks tertentu
- `isdigit()` -> cek apakah isinya angka
- `isalpha()` -> cek apakah isinya huruf

```
teks = "12345"
print(teks.isdigit())   # True
print(teks.isalpha())   # False
```

Output:

```
True
False
```

7.5 Format String

Kadang kita ingin menampilkan teks yang berisi variabel. Gunakan `f-string` (format string).

```
nama = "Budi"
umur = 16
print(f"Halo, nama saya {nama}, umur saya {umur} tahun.")
```

Output:

```
Halo, nama saya Budi, umur saya 16 tahun.
```

7.6 Multi-line String

String pada umumnya terdiri dari 1 baris saja, akan tetapi saat butuh strng multibars, string bisa dibuat dengan tiga tanda kutip (`'''...'''` atau `"""..."""`).

Contoh:

```
teks = """Ini adalah
string dengan
beberapa baris."""
print(teks)
```

Output:

```
Ini adalah
string dengan
beberapa baris.
```

7.7 Escape Character pada String

Kadang kita perlu menampilkan karakter khusus dalam string, seperti tanda kutip, baris baru, atau tab. Untuk itu, gunakan **escape character** dengan tanda backslash (`\`).

7.7.1 Macam-macam Escape Character

Escape	Fungsi
`\'`	Menampilkan tanda `'`
`\"`	Menampilkan tanda `"`
`\n`	Baris baru
`\t`	Tab
`\\`	Garis miring terbalik (`\`)

7.7.2 Contoh Penggunaan

Menampilkan tanda kutip di dalam string:

```
print('Abdullah ibnu Mas\'ud')
print("Dia berkata: \"Python itu mudah!\"")
```

Output:

```
Abdullah ibnu Mas'ud
Dia berkata: "Python itu mudah!"
```

Membuat baris baru dengan `\n` :

```
print('baris_satu\nbaris_dua\nbaris_tiga')
```

Output:

```
baris_satu
baris_dua
baris_tiga
```

Membuat tabel sederhana dengan tab (`\t`):

```
print('|\tno\t|\tNama\t|\tKelas\t|')
print('|\t1\t|\tAdin\t|\tTKJ1\t|')
print('|\t2\t|\tJack\t|\tTKJ1\t|')
```

Output:

```
|   no  | Nama | Kelas |
|   1   | Adin | TKJ1  |
|   2   | Jack | TKJ1  |
```