PENERBIT BUKULOKA

# Dasar-Dasar Pemrograman Python

Ir. Agus Siswoyo S.T., M.T.
Ir. Antonius Hendro Noviyanto, S.T., M.T.
Ir. Eko Arianto, S.T., M.T.

# Dasar-Dasar Pemrograman Phyton

**Ir. Agus Siswoyo S.T., M.T.**
**Ir. Antonius Hendro Noviyanto, S.T., M.T.**
**Ir. Eko Arianto, S.T., M.T.**

PT BUKULOKA LITERASI BANGSA

Anggota IKAPI: No. 645/DKI/2024

PENERBIT BUKULOKA

# Dasar-Dasar Pemrograman Phyton


| | |
|---|---|
| Penulis | : Ir. Agus Siswoyo S.T., M.T., Ir. Antonius Hendro Noviyanto, S.T., M.T., dan Ir. Eko Arianto, S.T., M.T. |
| ISBN | : |
| Penyunting Naskah | : Ahmad Fauzy Pratama, S.Pd. |
| Tata Letak | : Ahmad Fauzy Pratama, S.Pd. |
| Desain Sampul | : Fahri Firliansyah |

Penerbit

Penerbit PT Bukuloka Literasi Bangsa

Distributor: PT Yapindo

Kompleks Business Park Kebon Jeruk Blok I No. 21, Jl. Meruya Ilir Raya No. 88, Kelurahan Meruya Utara, Kecamatan Kembangan, Kota Adm. Jakarta Barat, Provinsi DKI Jakarta, Kode Pos: 11620

Email : penerbit.blb@gmail.com

Whatsapp : 0878-3483-2315

Website : bukuloka.com

# KATA PENGANTAR

Puji syukur ke hadirat Tuhan Yang Maha Esa atas rahmat dan karunia-Nya, sehingga buku ajar berjudul *Dasar-Dasar Pemrograman Python* ini dapat tersusun dengan baik dan hadir sebagai panduan awal bagi pembaca umum yang ingin mengenal dan mempelajari bahasa pemrograman Python secara sistematis dan mudah dipahami.

Buku ini disusun untuk memberikan pemahaman menyeluruh tentang konsep dasar pemrograman menggunakan Python, bahasa yang dikenal karena sintaksnya yang sederhana, fleksibel, dan digunakan secara luas di berbagai bidang, mulai dari pengembangan aplikasi hingga analisis data. *Dasar-Dasar Pemrograman Python* ditujukan untuk pembaca umum yang baru memulai perjalanan di dunia coding.

Dengan pendekatan yang praktis dan penjelasan langkah demi langkah, buku ini membahas struktur dasar program, tipe data, perulangan, fungsi, serta pengenalan terhadap pemrograman berorientasi objek. Harapannya, buku ini dapat menjadi fondasi yang kuat untuk mengembangkan keterampilan pemrograman lebih lanjut.

Jakarta, Juli 2025

Tim Penyusun

# DAFTAR ISI

# Bab 1: Pengenalan Pemrograman dan Python

## 1.1 Pengertian Pemrograman

**Pemrograman** adalah proses menulis, menguji, dan memelihara instruksi atau kode yang dapat dijalankan oleh komputer untuk melakukan tugas tertentu. Tujuan dari pemrograman adalah untuk mengembangkan perangkat lunak yang memungkinkan komputer untuk menjalankan berbagai jenis perintah dan fungsi sesuai dengan kebutuhan pengguna atau aplikasi. Dalam pemrograman, instruksi yang ditulis oleh programmer menggunakan **bahasa pemrograman**, yang merupakan seperangkat aturan yang mengatur bagaimana kode harus ditulis agar dapat dipahami dan dijalankan oleh komputer.

Bahasa pemrograman berfungsi sebagai penghubung antara pemrogram (manusia) dan komputer, dengan menyediakan sintaksis dan struktur yang memungkinkan instruksi untuk diterjemahkan dan dieksekusi dengan benar oleh mesin. Bahasa pemrograman dapat bervariasi, mulai dari bahasa tingkat tinggi seperti Python, Java, atau C++, hingga bahasa tingkat rendah yang lebih dekat dengan kode mesin, seperti Assembly.

Proses pemrograman biasanya melibatkan beberapa langkah penting, termasuk:

1. **Penulisan Kode**: Menulis instruksi atau algoritma menggunakan bahasa pemrograman yang sesuai dengan tujuan yang ingin dicapai.
2. **Pengujian**: Menguji kode yang ditulis untuk memastikan bahwa ia berfungsi dengan baik dan tidak ada bug atau kesalahan yang tersembunyi.
3. **Pemeliharaan**: Memperbaiki dan memperbarui kode seiring waktu untuk menambah fitur baru, memperbaiki masalah yang ditemukan, atau meningkatkan kinerja program.

Pemrograman adalah keterampilan dasar yang sangat penting dalam pengembangan perangkat lunak, dan digunakan dalam berbagai aplikasi mulai dari pengembangan aplikasi desktop, web, hingga perangkat keras dan sistem operasi. Sebuah kode yang ditulis dengan baik akan memungkinkan komputer untuk menjalankan perintah dengan efisien dan efektif, menyelesaikan tugas dengan akurat, dan memecahkan masalah yang lebih kompleks.

## 1.2 Sejarah Singkat Python

**Python** adalah bahasa pemrograman tingkat tinggi yang dirancang untuk **kemudahan penggunaan**, dengan sintaks yang jelas dan mudah dibaca. Python pertama kali dikembangkan oleh **Guido van Rossum** dan dirilis pada **tahun 1991**. Tujuan utama dari pembuatan Python adalah untuk menciptakan bahasa yang memiliki sintaks sederhana, tetapi tetap kuat dan fleksibel, memungkinkan

pengembang untuk menulis kode yang **lebih sedikit** namun **lebih efisien**.

Nama "Python" sendiri diambil dari acara televisi komedi Inggris **Monty Python's Flying Circus**, yang menunjukkan bahwa pengembang bahasa ini ingin menciptakan sesuatu yang menyenangkan dan tidak terlalu kaku. Python dirancang untuk **mendukung berbagai paradigma pemrograman**, termasuk **pemrograman berorientasi objek**, **imperatif**, dan **fungsional**.

Sejak pertama kali diperkenalkan, Python telah berkembang pesat dan digunakan di berbagai bidang, mulai dari **pengembangan web**, **analisis data**, **kecerdasan buatan**, **machine learning**, hingga **automasi sistem**. Keberhasilannya dapat dilihat dari **komunitas pengguna** yang sangat besar, serta berbagai **pustaka dan alat** yang tersedia, yang membuatnya sangat populer di kalangan pengembang perangkat lunak dari berbagai latar belakang.

Python terus berkembang dengan pembaruan dan peningkatan fitur yang dilakukan oleh **Python Software Foundation**. Saat ini, Python dianggap sebagai salah satu bahasa pemrograman paling **populer dan serbaguna** di dunia, dengan banyak aplikasi di berbagai industri teknologi dan riset.

## 1.3 Keunggulan Python

Python adalah bahasa pemrograman yang sangat populer karena keunggulannya yang membuatnya sangat disukai oleh pengembang di seluruh dunia. Beberapa alasan utama mengapa

Python begitu diminati dan digunakan secara luas di berbagai bidang adalah sebagai berikut:

### 1.3.1 Sintaks yang Sederhana dan Mudah Dibaca

Salah satu keunggulan terbesar dari Python adalah sintaksisnya yang sederhana dan mudah dibaca. Python dirancang dengan tujuan untuk membuat kode yang lebih jelas dan mudah dipahami, bahkan oleh pemula. Struktur kode Python menyerupai bahasa alami, sehingga pengembang dapat menulis kode yang bersih dan mudah dipelihara. Ini sangat membantu dalam mengurangi kesalahan pemrograman, mempermudah debugging, dan meningkatkan produktivitas pengembang. Sintaks yang mudah ini juga mempercepat proses pembelajaran bagi mereka yang baru memulai karier di dunia pemrograman.

### 1.3.2 Bersifat Open-Source dan Memiliki Komunitas yang Besar

Python bersifat open-source, yang berarti siapa saja dapat mengakses, memodifikasi, dan mendistribusikan kode sumbernya tanpa biaya. Ini memungkinkan siapa saja untuk menggunakannya secara bebas, menjadikannya lebih terjangkau dan lebih mudah diakses di berbagai industri dan tingkat pendidikan. Selain itu, komunitas Python yang besar dan aktif memberikan banyak dukungan bagi pengguna baru dan berpengalaman. Komunitas ini menyediakan berbagai forum diskusi, tutorial, pustaka (library), dan tools yang sangat membantu untuk pengembangan perangkat lunak. Dukungan komunitas yang luas memudahkan pengembang untuk menemukan solusi atas masalah mereka, serta berbagi pengetahuan dan pengalaman.

### 1.3.3 Mendukung Berbagai Paradigma Pemrograman

Python mendukung berbagai paradigma pemrograman, termasuk:

- Pemrograman prosedural: Di mana program ditulis dalam bentuk urutan langkah-langkah prosedural.
- Pemrograman fungsional: Di mana fungsi digunakan sebagai elemen dasar, dan fungsi tersebut dapat digunakan secara independen.
- Pemrograman berorientasi objek (OOP): Di mana objek dan kelas digunakan untuk membangun struktur perangkat lunak yang lebih modular dan terorganisir.

Kemampuan untuk bekerja dalam berbagai paradigma pemrograman ini membuat Python sangat fleksibel dan dapat diterapkan pada berbagai jenis proyek, dari yang sederhana hingga kompleks. Ini memungkinkan pengembang untuk memilih pendekatan yang paling sesuai dengan masalah yang sedang dipecahkan.

### 1.3.4 Memiliki Banyak Pustaka (Library) yang Mendukung Berbagai Bidang

Python memiliki banyak pustaka (library) yang siap pakai yang mendukung pengembangan aplikasi di berbagai bidang, seperti:

- Data Science: Python menyediakan pustaka seperti NumPy, Pandas, Matplotlib, dan SciPy yang sangat populer untuk analisis data, statistik, dan visualisasi data.

- Web Development: Framework seperti Django, Flask, dan FastAPI memudahkan pengembang dalam membangun aplikasi web yang kuat dan dinamis.
- Kecerdasan Buatan (AI) dan Pembelajaran Mesin (Machine Learning): Python juga memiliki pustaka yang sangat kuat untuk AI, seperti TensorFlow, Keras, PyTorch, dan scikit-learn, yang memungkinkan pengembang untuk membuat model kecerdasan buatan dan pembelajaran mesin dengan mudah.

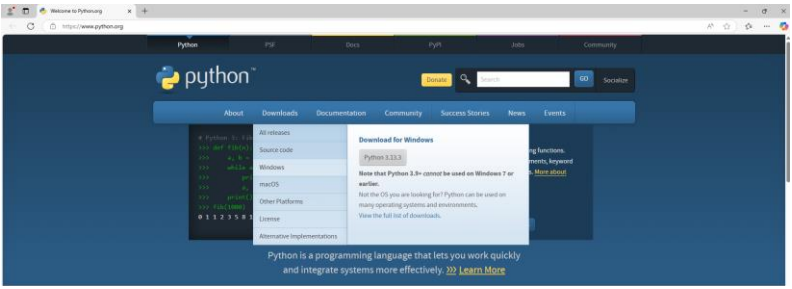
## 1.4 Instalasi Python

Untuk menggunakan Python, Anda perlu mengikuti beberapa langkah instalasi untuk memastikan bahwa Python terpasang dengan benar di komputer Anda. Berikut adalah langkah-langkah yang dapat dilakukan untuk menginstal Python:

### 1.4.1 Unduh Python dari Situs Resmi

Langkah pertama adalah mengunduh installer Python dari situs resmi Python, yang menjamin Anda mendapatkan versi terbaru dan aman dari bahasa pemrograman ini.

- Kunjungi situs resmi Python: Buka situs web resmi Python di https://www.python.org.
- Pilih versi Python yang sesuai: Pilih versi Python yang sesuai dengan sistem operasi yang Anda gunakan (Windows, macOS, atau Linux). Disarankan untuk memilih versi stabil terbaru.




*Gambar Tampilan Download Software Python*

1.4.2 Instal Python Sesuai dengan Sistem Operasi yang Digunakan

Setelah mengunduh file installer, langkah berikutnya adalah menginstal Python di komputer Anda. Proses ini akan sedikit berbeda tergantung pada sistem operasi yang digunakan.

- Windows: Jalankan file installer yang telah diunduh, kemudian pilih opsi untuk menambahkan Python ke PATH. Pastikan untuk memilih pilihan "Install Now" untuk menginstal Python secara otomatis dengan pengaturan default.
- macOS: Untuk pengguna macOS, Anda dapat mengunduh file .pkg dan mengikuti petunjuk untuk menginstal Python. Anda juga bisa menggunakan Homebrew untuk menginstal Python dengan perintah brew install python.
- Linux: Pada sebagian besar distribusi Linux, Python sudah terinstal secara default. Jika belum terinstal, Anda dapat menginstalnya melalui manajer paket, seperti sudo apt install python3 untuk distribusi berbasis Debian atau Ubuntu.

1.4.3 Pastikan Python Telah Terinstal dengan Benar

Setelah instalasi selesai, langkah terakhir adalah memastikan bahwa Python terinstal dengan benar di sistem Anda. Jalankan perintah python –version di CMD. Maka hasilnya akan muncul seperti ini:


*Gambar Hasil Instalasi Software Python*

## 1.5 Latihan Soal

1. Apa yang dimaksud dengan pemrograman?
2. Sebutkan tiga keunggulan utama dari Python.
3. Bagaimana langkah-langkah instalasi Python?
4. Tulis contoh program sederhana menggunakan Python.
5. Jelaskan perbedaan antara mode interaktif dan mode skrip dalam Python.

# Bab 2: Struktur Dasar Program Python

## 2.1 Mengenal Python sebagai Bahasa Pemrograman

**Python** adalah bahasa pemrograman yang sangat populer di kalangan pengembang perangkat lunak, ilmuwan data, dan penggemar pemrograman karena sintaksisnya yang sederhana, intuitif, dan mudah dipahami. Salah satu alasan Python menjadi pilihan utama adalah fleksibilitasnya, yang memungkinkan penggunaan untuk berbagai macam aplikasi, mulai dari pengembangan web, analisis data, pembelajaran mesin, hingga pengembangan perangkat keras. Python juga memiliki komunitas yang besar dan banyak dukungan pustaka (library) yang dapat membantu mempercepat proses pengembangan.

Pada bab ini, kita akan membahas **struktur dasar program Python**, yang mencakup berbagai elemen penting yang digunakan dalam penulisan kode Python. Struktur dasar ini sangat penting untuk dipahami oleh pemula, karena mempengaruhi bagaimana kita menulis dan mengorganisir kode dalam program Python.

## 2.2 Elemen Dasar dalam Program Python

Python adalah bahasa pemrograman yang banyak digunakan karena sintaksnya yang sederhana dan mudah dipahami. Dalam pengembangan program, terdapat beberapa elemen dasar yang menjadi fondasi utama dalam menulis kode secara efektif. Elemen-elemen ini memungkinkan pengembang menciptakan program yang lebih efisien, mudah dibaca, dan dapat dikelola dengan baik. Pemahaman terhadap elemen-elemen ini sangat penting, baik bagi pemula maupun bagi mereka yang ingin mengembangkan keterampilan pemrograman lebih lanjut.

### 2.2.1 Komentar

Komentar dalam Python digunakan untuk memberikan catatan atau penjelasan di dalam kode yang tidak akan dieksekusi oleh komputer. Tujuan utama dari komentar adalah untuk membantu pengembang memahami struktur dan fungsi dari suatu bagian kode, baik untuk keperluan pribadi maupun dalam kerja tim. Dengan adanya komentar, kode menjadi lebih mudah dipelajari, dimodifikasi, dan diperbaiki di kemudian hari. Python mendukung dua jenis komentar, yaitu komentar satu baris dan komentar banyak baris. Komentar satu baris digunakan untuk memberikan penjelasan singkat dan biasanya diletakkan di atas atau di samping kode yang ingin dijelaskan.

```
# ini adalah contoh komentar
# penulisan komentar untuk satu baris
# diawali dengan tanda pagar
# komentar ini tidak akan dieksekusi oleh mesin
# penulisan komentar bisa dilakukan sebelu kode program
# atau setelah kode program
# contoh

# mencetak hello
print("Hello")
print("World") # mencetak world
```

*Gambar Contoh Penulisan Komentar Satu Baris*

Sementara itu, komentar banyak baris digunakan untuk memberikan penjelasan yang lebih panjang, seperti dokumentasi untuk suatu fungsi atau program.

```
"""
Ini contoh penulisan komentar yang lebih dari satu baris
Penulisan Komentar lebih dari satu baris dilakukan
dengan menggunakan kutip dua 3 kali pada awal komentar dan
ditutup dengan kutip dua 3 kali
"""
```

*Gambar Contoh Penulisan Komentar Banyak Baris*

Penggunaan komentar yang baik akan meningkatkan keterbacaan kode serta membantu dalam proses debugging dan pengembangan lebih lanjut.

### 2.2.2 Variabel dan Tipe Data

Variabel adalah tempat penyimpanan nilai dalam program yang dapat digunakan dan dimanipulasi selama eksekusi program berlangsung. Dalam Python, variabel tidak memerlukan deklarasi tipe secara eksplisit karena Python menggunakan sistem tipe data yang dinamis, yang berarti tipe data suatu variabel akan dikenali secara otomatis berdasarkan nilai yang diberikan. Karakter pada penulisan nama variabel bersifat sensitif (case-sensitif). Dimana

huruf kecil dan huruf kapital dibedakan. Sebagai contoh, variabel iniVariabel dan inivariabel adalah dua nama variabel yang berbeda.

```
#contoh untuk memasukan data ke dalam variabel
nama = "Budi"

#nilai dan tipe data dalam variabel dapat diubah
nilai = 50 #nilai awal
print(nilai) #mencetak variabel nilai
type(nilai) #mengecek tipe data variabel nilai
nilai = "enam puluh" #nilai setelah diubah
print(nilai) #mencetak variabel nilai
type(nilai) #mengecek tipe data variabel nilai

namaDepan = "Budi"
namaBelakang = "Susanto"


#contoh variabel lainya
inivariabel = "Budi"
ini_juga_variabel = "Susanto"
inivariabel222 = "Budi Susanto"

panjang = 20
lebar = 15
luas = panjang \* lebar
print(luas)
```

*Gambar Contoh Penulisan Variabel dan Tipe Data*

Beberapa tipe data dasar yang sering digunakan dalam Python antara lain: integer (int), yang digunakan untuk menyimpan bilangan bulat; float, yang digunakan untuk menyimpan bilangan desimal; string (str), yang digunakan untuk menyimpan teks atau karakter; dan boolean (bool), yang digunakan untuk menyimpan nilai logika True atau False. Selain tipe data dasar tersebut, Python juga menyediakan tipe data yang lebih kompleks, seperti list, tuple, set, dan dictionary. Setiap tipe data memiliki kegunaan yang berbeda

dan dapat digunakan sesuai dengan kebutuhan program yang sedang dikembangkan.

2.2.3 Operasi Dasar

Python menyediakan berbagai operasi dasar yang memungkinkan pengembang melakukan perhitungan atau pemeriksaan kondisi dalam program. Operasi dasar ini berperan penting dalam memproses data serta membuat keputusan berdasarkan kondisi tertentu. Beberapa kategori operasi dasar yang sering digunakan dalam Python meliputi: operasi aritmetika, yang mencakup penjumlahan, pengurangan, perkalian, pembagian, dan operasi matematika lainnya;

*Tabel Operator Aritmatika*

*Sumber: https://belajarpython.com/tutorial/operator-python/*

| Operator | Contoh | Penjelasan |
|---|---|---|
| Penjumlahan `+` | `1 + 3 = 4` | Menjumlahkan nilai dari masing-masing operan atau bilangan |
| Pengurangan `-` | `4 - 1 = 3` | Mengurangi nilai operan di sebelah kiri menggunakan operan di sebelah kanan |
| Perkalian `*` | `2 * 4 = 8` | Mengalikan operan/bilangan |
| Pembagian `/` | `10 / 5 = 2` | Untuk membagi operan di sebelah kiri menggunakan operan di sebelah kanan |
| Sisa Bagi `%` | `11 % 2 = 1` | Mendapatkan sisa pembagian dari operan di sebelah kiri operator ketika dibagi oleh operan di sebelah kanan |
| Pangkat `**` | `8 ** 2 = 64` | Memangkatkan operan disebelah kiri operator dengan operan di sebelah kanan operator |
| Pembagian Bulat `//` | `10 // 3 = 3` | Sama seperti pembagian. Hanya saja angka dibelakang koma dihilangkan |

Operasi perbandingan, yang digunakan untuk membandingkan dua nilai dan menghasilkan output berupa nilai

boolean (True atau False), yang sangat berguna dalam pengambilan keputusan dan kontrol alur program;

*Tabel Operator Perbandingan*

*Sumber: https://belajarpython.com/tutorial/operator-python/*

| Operator | Contoh | Penjelasan |
|---|---|---|
| Sama dengan `==` | `1 == 1` | bernilai True Jika masing-masing operan memiliki nilai yang sama, maka kondisi bernilai benar atau True. |
| Tidak sama dengan `!=` | `2 != 2` | bernilai False Akan menghasilkan nilai kebalikan dari kondisi sebenarnya. |
| Tidak sama dengan `<>` | `2 <> 2` | bernilai False Akan menghasilkan nilai kebalikan dari kondisi sebenarnya. |
| Lebih besar dari `>` | `5 > 3` | bernilai True Jika nilai operan kiri lebih besar dari nilai operan kanan, maka kondisi menjadi benar. |
| Lebih kecil dari `<` | `5 < 3` | bernilai True Jika nilai operan kiri lebih kecil dari nilai operan kanan, maka kondisi menjadi benar. |
| Lebih besar atau sama dengan `>=` | `5 >= 3` | bernilai True Jika nilai operan kiri lebih besar dari nilai operan kanan, atau sama, maka kondisi menjadi benar. |
| Lebih kecil atau sama dengan `<=` | `5 <= 3` | bernilai True Jika nilai operan kiri lebih kecil dari nilai operan kanan, atau sama, maka kondisi menjadi benar. |

Serta operasi logika, seperti AND, OR, dan NOT, yang digunakan untuk menggabungkan beberapa kondisi logis dalam pengujian kondisi tertentu. Dengan memahami dan menggunakan operasi dasar ini, pengembang dapat memanipulasi data dengan lebih fleksibel serta mengontrol alur program sesuai dengan logika yang diinginkan.

### 2.2.4 Struktur Kontrol

Struktur kontrol dalam Python berfungsi untuk mengatur alur eksekusi program berdasarkan kondisi tertentu. Struktur ini sangat penting dalam pembuatan program yang kompleks karena memungkinkan pengembang untuk membuat keputusan dan mengulangi eksekusi kode berdasarkan situasi yang terjadi. Beberapa jenis struktur kontrol yang umum digunakan dalam Python antara lain: percabangan (conditional statements), yang digunakan untuk mengeksekusi blok kode tertentu berdasarkan kondisi yang diberikan, dan jika suatu kondisi terpenuhi, maka blok kode tersebut akan dijalankan, sedangkan jika tidak, program akan mengeksekusi alternatif lainnya. Dimana pada penulisan kode program percabangan meliputi kondisi sebagai berikut:

- if statement: Melakukan evaluasi kondisi dan menjalankan akan kode jika kondisi yang dibandingkan tersebut bernilai True.
- elif statement: Memberikan alternatif pilihan untuk hasil evaluasi kondisi yang sebelumnya salah.
- else statement: Menjalankan kode jika semua kondisi sebelumnya salah.

```
nilai = 85

if nilai >= 90:
    print("Grade A")
elif 80 <= nilai < 90:
    print("Grade B")
else:
    print("Grade C")
```

*Gambar Contoh Program Percabangan*

Perulangan (loops), yang digunakan untuk mengulangi eksekusi suatu blok kode selama kondisi tertentu masih terpenuhi. Perulangan memungkinkan otomatisasi proses yang berulang tanpa perlu menulis kode secara berulang-ulang. Pada struktur perulangan dapat dilakukan dengan langkah for loop maupun while loop. Dimana pada struktur for loop, kode akan dijalankan secara berulang untuk setiap item dalam sebuah urutan (misalnya, list, tuple, string). Sedangakan pada struktur while loop, kode akan dijalankan secara berulang selama kondisi yang dibandingkan bernilai true.

```
# For loop
angka = [1, 2, 3, 4, 5]
for i in angka:
    print(i)

# While loop
i = 0
while i < 5:
    print(i)
    i += 1
```

*Gambar Contoh Program Perulangan*

Dengan adanya struktur kontrol, program dapat berjalan lebih dinamis dan sesuai dengan kebutuhan pengguna atau data yang diproses.

## 2.3 Struktur Program Python

Struktur program Python umumnya sangat sederhana dan mudah diikuti, yang membuatnya menjadi pilihan yang populer di kalangan pengembang pemula maupun profesional. Program Python biasanya terdiri dari beberapa komponen utama, yang masing-masing memiliki peran penting dalam mengorganisir dan menjalankan kode dengan efisien. Berikut adalah komponen utama yang sering dijumpai dalam program Python:

### 2.3.1 Pernyataan Import

Pada awal program, biasanya terdapat pernyataan import untuk memanggil pustaka atau modul eksternal yang diperlukan. Python menyediakan pustaka standar yang sangat lengkap, dan juga memungkinkan penggunaan pustaka tambahan untuk memperluas fungsionalitas program. Dengan mengimpor modul, kita dapat memanfaatkan berbagai fitur yang sudah ada tanpa perlu menulis ulang kode dari awal.

```
# Mengimpor modul matematika
import math

# Menggunakan fungsi dari modul math
hasil = math.sqrt(20)  # Mengambil akar kuadrat dari 20
print(hasil)
```

*Gambar Contoh Program Import Pustaka*

### 2.3.2 Deklarasi Variabel

Deklarasi variabel merupakan langkah pertama dalam menyimpan data atau nilai yang akan digunakan sepanjang program. Di Python, Anda tidak perlu mendeklarasikan tipe data variabel

secara eksplisit karena Python bersifat dinamis, artinya variabel dapat menampung berbagai tipe data sesuai dengan nilai yang diberikan. Variabel digunakan untuk menyimpan angka, teks, dan bahkan objek lain, sehingga program dapat berfungsi secara dinamis dan fleksibel.

```
# Contoh deklarasi variabel
nama = "Budi"  # Variabel 'nama' dibuat dan diberi nilai "Budi"
umur =35       # Variabel 'umur' dibuat dan diberi nilai 35
is_active = False # Variabel 'is_active' dibuat dan diberi nilai False
```

*Gambar Contoh Program Deklarasi Variabel*

### 2.3.3 Fungsi

Fungsi adalah sekumpulan instruksi yang dikemas menjadi satu kesatuan dengan tujuan tertentu. Fungsi memungkinkan kita untuk mengorganisir kode sehingga lebih modular dan dapat digunakan berulang kali. Dengan mendefinisikan fungsi, kita dapat memecah program menjadi bagian-bagian yang lebih kecil dan lebih mudah dipahami. Fungsi juga memungkinkan kita untuk menerima input (parameter) dan memberikan output (nilai kembali), yang meningkatkan fleksibilitas dalam mengelola data dalam program.

```
def sapa(nama):
  """
  Fungsi ini menyapa pengguna dengan nama yang diberikan.
  """
  print("Halo, " + nama + "!")

# Pemanggilan fungsi
sapa("Budi")  # Output: Halo, Budi!
```

*Gambar Contoh Program Fungsi*

### 2.3.4 Kondisional (If-Else)

Bagian ini digunakan untuk mengontrol alur eksekusi program berdasarkan kondisi tertentu. Dengan menggunakan

pernyataan kondisional seperti if-else, program dapat memilih jalur eksekusi yang berbeda tergantung pada apakah kondisi yang diberikan bernilai benar atau salah. Ini memungkinkan pembuatan keputusan dalam kode, sehingga program bisa beradaptasi dengan berbagai situasi dan masukan yang berbeda.

```
nilai = 60

if nilai >= 70:
    print("Selamat, Anda lulus!")
else:
    print("Maaf, Anda belum lulus.")
```

*Gambar Contoh Program If-Else*

### 2.3.5 Perulangan (Looping)

Perulangan (looping) memungkinkan kita untuk menjalankan bagian dari program secara berulang-ulang berdasarkan kondisi tertentu. Python menyediakan dua jenis perulangan utama, yaitu for dan while. Perulangan for digunakan untuk mengulang sebuah blok kode untuk setiap item dalam suatu koleksi (seperti daftar atau rentang angka), sedangkan while digunakan untuk mengulang kode selama suatu kondisi tertentu terpenuhi. Perulangan sangat berguna untuk memproses data secara efisien tanpa menulis kode berulang.

```
# Mencetak setiap elemen dalam list
   nama_teman = ["Cia", "Budi", "Jono"]
   for nama in nama_teman:
       print(f"Halo, {nama}!")
```

*Gambar Contoh Program Perulangan*

2.3.6 Eksekusi Program

Bagian ini adalah inti dari program, di mana kode akan dieksekusi sesuai dengan urutan yang telah ditentukan. Eksekusi program dimulai dengan menjalankan pernyataan yang berada di luar fungsi atau kelas. Biasanya, pada bagian ini, kita akan melihat program berjalan berdasarkan input yang diberikan, pemrosesan data, atau interaksi dengan pengguna. Bagian ini mengendalikan bagaimana program bekerja pada saat dijalankan.

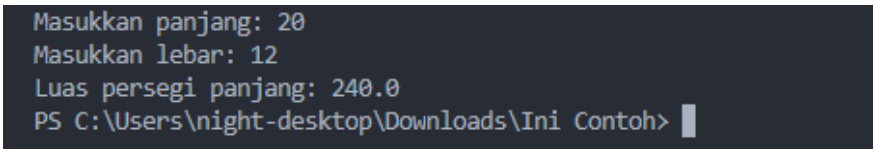
## 2.4 Contoh Program Sederhana

```python
def hitung_luas(panjang, lebar):
    return panjang * lebar

panjang = float(input("Masukkan panjang: "))
lebar = float(input("Masukkan lebar: "))

print(f"Luas persegi panjang: {hitung_luas(panjang, lebar)}")
```

*Gambar Contoh Program Sederhana*



```
Masukkan panjang: 20
Masukkan lebar: 12
Luas persegi panjang: 240.0
PS C:\Users\night-desktop\Downloads\Ini Contoh>
```

*Gambar Contoh Hasil Program Sederhana*

## 2.5 Latihan Soal

1. Apa saja elemen dasar dalam program Python?

2. Jelaskan fungsi dari pernyataan `if-else` dalam Python.
3. Bagaimana cara mendeklarasikan variabel dalam Python?
4. Tuliskan contoh program Python sederhana yang menggunakan perulangan.
5. Apa manfaat dari penggunaan fungsi dalam Python?

# Bab 3: Tipe Data dan Variabel

## 3.1 Pengertian Tipe Data dan Variabel

Dalam pemrograman, tipe data dan variabel adalah dua konsep fundamental yang digunakan untuk menyimpan, mengelola, dan memanipulasi data dalam suatu program. Tipe data menentukan jenis nilai yang dapat disimpan dalam variabel, sementara variabel berfungsi sebagai wadah atau tempat penyimpanan data yang dapat digunakan dan dimodifikasi selama program berjalan.

Tipe data sangat penting karena menentukan bagaimana komputer akan mengalokasikan memori, bagaimana data dapat diproses, serta operasi apa saja yang dapat dilakukan terhadap data tersebut. Secara umum, tipe data dapat dikategorikan menjadi tipe data primitif seperti **integer (bilangan bulat), float (bilangan desimal), char (karakter), dan boolean (true/false)**, serta tipe data kompleks seperti **array, string, dan struktur data lainnya**.

Sementara itu, variabel adalah nama yang diberikan untuk suatu lokasi di memori yang digunakan untuk menyimpan nilai tertentu. Setiap variabel memiliki nama unik yang digunakan untuk mengakses dan mengubah nilainya dalam program. Dalam banyak bahasa pemrograman, variabel harus dideklarasikan dengan tipe data tertentu agar dapat digunakan dengan benar. Sebagai contoh, dalam

bahasa pemrograman seperti Python, variabel dapat dibuat secara dinamis tanpa mendefinisikan tipe datanya secara eksplisit, sedangkan dalam bahasa seperti C dan Java, variabel harus dideklarasikan dengan tipe data tertentu sebelum digunakan.

Dengan memahami konsep tipe data dan variabel, seorang programmer dapat menulis kode yang lebih efisien, terstruktur, dan mudah dipahami. Penggunaan tipe data yang tepat juga membantu dalam optimasi penggunaan memori serta menghindari kesalahan dalam perhitungan atau manipulasi data dalam program

## 3.2 Jenis-Jenis Tipe Data

Dalam dunia pemrograman, tipe data adalah kategori yang digunakan untuk menentukan jenis nilai yang dapat disimpan dalam suatu variabel. Pemilihan tipe data yang tepat berpengaruh pada efisiensi penyimpanan, kecepatan eksekusi, dan keakuratan operasi dalam program.

Secara umum, tipe data dapat dikategorikan menjadi beberapa jenis utama, yaitu numerik, boolean, karakter dan string, serta koleksi.

### 3.2.1 Tipe Data Numerik

Tipe data numerik digunakan untuk menyimpan angka, baik dalam bentuk bilangan bulat maupun pecahan desimal. Jenis utama dalam kategori ini meliputi:

- Integer (bilangan bulat): Tipe data yang menyimpan angka tanpa desimal, baik positif maupun negatif. Contohnya adalah angka 10, -5, dan 1000.
- Floating Point (bilangan desimal): Digunakan untuk menyimpan angka dengan desimal, seperti 3.14, -2.5, dan 0.001.
- Bilangan Kompleks: Tipe data yang digunakan dalam operasi matematika khusus, seperti 2 + 3i atau -1.5 + 2.7i.

Tipe data numerik banyak digunakan dalam operasi matematika, perhitungan keuangan, dan analisis data.

### 3.2.2 Tipe Data Boolean

Tipe data boolean adalah tipe data yang hanya memiliki dua nilai, yaitu benar (True) atau salah (False).

Tipe data ini sering digunakan dalam percabangan logika, seperti dalam kondisi jika-maka, perulangan, atau perbandingan dua nilai. Misalnya, jika suatu kondisi terpenuhi, hasilnya adalah benar, jika tidak, hasilnya adalah salah.

Boolean juga sering muncul dalam operasi perbandingan, misalnya membandingkan apakah suatu nilai lebih besar atau lebih kecil dari nilai lainnya.

### 3.2.3 Tipe Data Karakter dan String

Tipe data ini digunakan untuk menyimpan teks atau kumpulan karakter.

- Karakter (char): Biasanya digunakan untuk menyimpan satu karakter tunggal, seperti ‘A’, ‘b’, atau ‘1’.

- String: Merupakan kumpulan karakter yang lebih panjang, seperti kata atau kalimat, misalnya "Hello", "Python", atau "2024".

String banyak digunakan dalam pemrograman, misalnya untuk menampilkan teks pada layar, menyimpan nama pengguna, atau mengolah data berbasis teks. String juga dapat diolah lebih lanjut, seperti dipecah, digabungkan, atau diubah formatnya sesuai kebutuhan.

### 3.2.4 Tipe Data Koleksi

Tipe data koleksi digunakan untuk menyimpan beberapa nilai dalam satu variabel. Tipe ini sangat berguna dalam pemrosesan data yang kompleks dan penyimpanan informasi dalam jumlah besar.

- List: Merupakan kumpulan data yang dapat diubah (mutable) dan bersifat terurut. List dapat berisi berbagai jenis data, seperti angka, teks, atau gabungan keduanya.
- Tuple: Mirip dengan list, tetapi bersifat tetap (immutable), sehingga tidak bisa diubah setelah dideklarasikan. Tujuan penggunaannya adalah untuk data yang tidak boleh dimodifikasi.
- Dictionary: Struktur data yang menyimpan pasangan kunci dan nilai. Setiap data disimpan dengan kunci unik, sehingga mudah diakses berdasarkan kuncinya, mirip dengan kamus yang memiliki kata dan definisinya.
- Set: Kumpulan elemen yang unik dan tidak berurutan. Set digunakan ketika hanya dibutuhkan kumpulan nilai tanpa

adanya duplikasi, misalnya untuk menyimpan daftar item yang tidak boleh memiliki elemen yang sama.

*Tabel Tipe Data pada Pemrograman Python*

*Sumber: https://belajarpython.com/tutorial/tipe-data-python/*

| Tipe Data | Contoh | Penjelasan |
|---|---|---|
| Boolean | `True` atau `False` | Menyatakan benar `True` yang bernilai `1`, atau salah `False` yang bernilai `0` |
| String | `"Ayo belajar Python"` | Menyatakan karakter/kalimat bisa berupa huruf angka, dll (diapit tanda `"` atau `'`) |
| Integer | `25` atau `1209` | Menyatakan bilangan bulat |
| Float | `3.14` atau `0.99` | Menyatakan bilangan yang mempunyai koma |
| Hexadecimal | `9a` atau `1d3` | Menyatakan bilangan dalam format heksa (bilangan berbasis 16) |
| Complex | `1 + 5j` | Menyatakan pasangan angka real dan imajiner |
| List | `['xyz', 786, 2.23]` | Data untaian yang menyimpan berbagai tipe data dan isinya bisa diubah-ubah |
| Tuple | `('xyz', 768, 2.23)` | Data untaian yang menyimpan berbagai tipe data tapi isinya tidak bisa diubah |
| Dictionary | `{'nama': 'adi','id':2}` | Data untaian yang menyimpan berbagai tipe data berupa pasangan penunjuk dan nilai |

```
#tipe data Boolean
print(True)

#tipe data String
print("Ayo belajar Python")
print('Belajar Python Sangat Mudah')

#tipe data Integer
print(20)

#tipe data Float
print(3.14)

#tipe data Hexadecimal
print(9a)

#tipe data Complex
print(5j)

#tipe data List
print([1,2,3,4,5])
print(["satu", "dua", "tiga"])

#tipe data Tuple
print((1,2,3,4,5))
print(("satu", "dua", "tiga"))

#tipe data Dictionary
print({"nama":"Budi", 'umur':20})
#tipe data Dictionary dimasukan ke dalam variabel biodata
biodata = {"nama":"Andi", 'umur':21} #proses inisialisasi variabel biodata
print(biodata) #proses pencetakan variabel biodata yang berisi tipe data Dictionary
print(type(biodata)) #fungsi untuk mengecek jenis tipe data. akan tampil <class 'dict'>
#yang berarti dict adalah tipe data dictionary
```

*Gambar Contoh Program Penggunaan Tipe Data*

## 3.3 Deklarasi dan Penggunaan Variabel

Variabel dalam pemrograman merupakan tempat penyimpanan sementara yang digunakan untuk menyimpan dan mengelola data. Setiap variabel memiliki nama yang digunakan untuk mengakses nilai yang tersimpan di dalamnya. Dalam berbagai bahasa pemrograman, variabel digunakan untuk menyimpan berbagai jenis data, seperti angka, teks, nilai logika, atau objek yang lebih kompleks.

Deklarasi dan penggunaan variabel bergantung pada aturan yang ditentukan oleh bahasa pemrograman yang digunakan. Beberapa bahasa memerlukan tipe data yang harus ditentukan secara eksplisit saat mendeklarasikan variabel, sementara yang lain memungkinkan penentuan tipe data secara otomatis berdasarkan nilai yang diberikan.

### 3.3.1 Deklarasi Variabel

Deklarasi variabel adalah proses mendefinisikan suatu variabel agar dapat digunakan dalam program. Dalam beberapa bahasa pemrograman, deklarasi variabel memerlukan penyebutan tipe data secara eksplisit untuk menentukan jenis nilai yang dapat disimpan dalam variabel tersebut. Tipe data ini dapat berupa bilangan bulat, bilangan desimal, teks, atau nilai logika.

Di sisi lain, ada juga bahasa pemrograman yang menggunakan pendekatan penentuan tipe data secara otomatis. Dalam hal ini, interpreter atau compiler akan menyesuaikan tipe variabel berdasarkan nilai yang diberikan tanpa memerlukan deklarasi eksplisit. Pendekatan ini sering digunakan dalam bahasa pemrograman modern untuk meningkatkan fleksibilitas dan kemudahan penggunaan.

### 3.3.2 Penggunaan Variabel

Setelah dideklarasikan, variabel dapat digunakan dalam berbagai operasi dan perhitungan dalam program. Variabel memungkinkan penyimpanan nilai yang dapat diperbarui atau dimanipulasi sesuai kebutuhan program. Nilai dalam variabel juga

dapat berubah selama eksekusi program, tergantung pada operasi yang dilakukan terhadapnya.

Variabel biasanya digunakan dalam berbagai operasi matematika, pengambilan keputusan, dan penyimpanan data untuk keperluan lebih lanjut. Dalam suatu program, variabel dapat digunakan kembali di berbagai bagian kode tanpa perlu mendeklarasikannya ulang, sehingga meningkatkan efisiensi dan keterbacaan kode.

Penggunaan variabel juga memungkinkan pengolahan data yang lebih dinamis, seperti menerima input dari pengguna, menyimpan hasil perhitungan, atau mengelola status dalam suatu sistem. Oleh karena itu, pemahaman mengenai deklarasi dan penggunaan variabel menjadi dasar yang penting dalam pemrograman.

```
#proses memasukan data ke dalam variabel
nama = "Budi"
#proses mencetak variabel
print(nama)

#nilai dan tipe data dalam variabel dapat diubah
umur = 30 #nilai awal
print(umur) #mencetak nilai umur
type(umur) #mengecek tipe data umur
umur = "dua puluh satu" #nilai setelah diubah
print(umur) #mencetak nilai umur
type(umur) #mengecek tipe data umur

namaDepan = "Budi"
namaBelakang = "Susanto"
nama = namaDepan + " " + namaBelakang
umur = 30
hobi = "Membaca"
print("Biodata\n", nama, "\n", umur, "\n", hobi)

#contoh variabel lainya
inivariabel = "Halo"
ini_juga_variabel = "Hai"
_inivariabeljuga = "Hi"
inivariabel222 = "Bye"

panjang = 10
lebar = 5
luas = panjang \* lebar
print(luas)
```

*Gambar Contoh Program Deklarasi dan Penggunaan Variabel*

## 3.4 Konversi Tipe Data

Dalam pemrograman, tipe data menentukan jenis nilai yang dapat disimpan dan diolah dalam suatu variabel. Terkadang, dalam proses pemrograman, kita perlu mengubah nilai dari satu tipe data ke tipe lainnya agar sesuai dengan kebutuhan pemrosesan. Proses ini dikenal sebagai **konversi tipe data** atau **type conversion**.

Konversi tipe data dalam Python terbagi menjadi dua jenis, yaitu **konversi eksplisit** dan **konversi implisit**.

**Konversi eksplisit** terjadi ketika programmer secara sadar dan sengaja mengubah tipe data menggunakan fungsi bawaan

Python. Misalnya, ketika ingin mengubah angka menjadi teks, kita bisa menggunakan fungsi yang sesuai. Beberapa fungsi yang sering digunakan untuk konversi eksplisit adalah:

- **int()** → Mengubah nilai menjadi bilangan bulat (integer). Biasanya digunakan untuk mengonversi angka desimal atau teks yang berisi angka ke dalam bentuk bilangan bulat.
- **float()** → Mengubah nilai menjadi bilangan desimal (floating point). Ini berguna ketika kita membutuhkan angka dengan presisi lebih tinggi.
- **str()** → Mengubah nilai menjadi string (teks). Digunakan ketika angka atau nilai lain ingin direpresentasikan dalam bentuk teks.
- **bool()** → Mengubah nilai menjadi boolean (True atau False). Nilai kosong atau nol akan dikonversi menjadi False, sedangkan nilai lainnya menjadi True.

Misalnya, dalam suatu program, kita mungkin perlu mengubah angka menjadi teks sebelum menampilkannya di layar, atau mengubah teks yang berisi angka menjadi bilangan untuk keperluan perhitungan. Dalam kasus ini, konversi eksplisit sangat membantu untuk memastikan bahwa tipe data yang digunakan sesuai dengan operasi yang dilakukan.

Selain konversi eksplisit, Python juga mendukung **konversi implisit**, yaitu konversi yang terjadi secara otomatis tanpa perlu instruksi langsung dari programmer. Python melakukan ini ketika dalam suatu operasi terdapat dua tipe data berbeda yang perlu disesuaikan agar hasilnya tetap akurat. Misalnya, ketika bilangan

bulat (integer) dikalikan dengan bilangan desimal (float), Python secara otomatis mengubah hasilnya menjadi bilangan desimal agar presisi tetap terjaga.

```python
# 3.4.1 Konversi Eksplisit (Manual)
print("=== Konversi Eksplisit ===")

# Konversi ke Integer
float_value = 10.75
string_number = "25"

int_from_float = int(float_value)  # Mengubah float ke integer (desimal dibulatkan ke bawah)
int_from_string = int(string_number)  # Mengubah string angka ke integer

print("Float ke Integer:", int_from_float)
print("String ke Integer:", int_from_string)

# Konversi ke Float
int_value = 7
string_decimal = "3.14"

float_from_int = float(int_value)  # Mengubah integer ke float
float_from_string = float(string_decimal)  # Mengubah string angka desimal ke float

print("Integer ke Float:", float_from_int)
print("String ke Float:", float_from_string)

# Konversi ke String
int_number = 100
float_number = 45.67

string_from_int = str(int_number)  # Mengubah integer ke string
string_from_float = str(float_number)  # Mengubah float ke string

print("Integer ke String:", string_from_int)
print("Float ke String:", string_from_float)

# Konversi ke Boolean
zero_value = 0
non_zero_value = 5
empty_string = ""
non_empty_string = "Python"

bool_from_zero = bool(zero_value)  # False
bool_from_non_zero = bool(non_zero_value)  # True
bool_from_empty_string = bool(empty_string)  # False
bool_from_non_empty_string = bool(non_empty_string)  # True

print("0 ke Boolean:", bool_from_zero)
print("5 ke Boolean:", bool_from_non_zero)
print('"" (string kosong) ke Boolean:', bool_from_empty_string)
print('"Python" (string isi) ke Boolean:', bool_from_non_empty_string)


# 3.4.2 Konversi Implisit (Otomatis)
print("\n=== Konversi Implisit ===")

# Python secara otomatis mengonversi integer ke float dalam operasi matematika
implicit_conversion = 5 + 3.2  # Integer + Float --> hasil otomatis Float

print("Hasil operasi 5 + 3.2:", implicit_conversion, "(Tipe:", type(implicit_conversion), ")")

# Konversi otomatis dalam ekspresi logika
boolean_result = 10 > 3  # Hasilnya otomatis Boolean (True)

print("Hasil dari 10 > 3:", boolean_result, "(Tipe:", type(boolean_result), ")")
```

*Gambar Contoh Program Konversi Tipe Data*

Namun, dalam beberapa kasus, konversi tipe data dapat menyebabkan error atau hasil yang tidak diharapkan jika tidak dilakukan dengan hati-hati. Sebagai contoh, jika kita mencoba mengubah teks yang bukan angka menjadi bilangan, program akan mengalami kesalahan. Oleh karena itu, sebelum melakukan konversi, penting untuk memastikan bahwa nilai yang akan dikonversi sesuai dengan tipe data tujuan.

Pemahaman tentang konversi tipe data sangat penting dalam pemrograman karena memungkinkan kita untuk menangani berbagai jenis input, menyimpan dan memanipulasi data dengan lebih fleksibel, serta menghindari kesalahan dalam perhitungan atau operasi lainnya.

## 3.5 Latihan Soal

1. Jelaskan perbedaan antara tipe data integer dan float.
2. Apa yang dimaksud dengan tipe data boolean?
3. Sebutkan perbedaan antara list dan tuple.
4. Bagaimana cara mengonversi string menjadi integer dalam Python?
5. Buat contoh deklarasi variabel dan penggunaannya dalam bahasa Python.

# Bab 4: Operator dan Ekspresi

## 4.1 Pengertian Operator dan Ekspresi

Dalam pemrograman, operator dan ekspresi merupakan elemen dasar yang digunakan untuk melakukan berbagai operasi pada data. **Operator** adalah simbol atau karakter khusus yang digunakan untuk menjalankan suatu operasi pada satu atau lebih nilai atau variabel. Operator memungkinkan komputer melakukan berbagai perhitungan dan manipulasi data, seperti penjumlahan, pengurangan, perbandingan, dan operasi logika.

Sementara itu, **ekspresi** adalah kombinasi dari variabel, konstanta, dan operator yang menghasilkan suatu nilai baru. Ekspresi dapat berupa operasi matematika sederhana seperti 5 + 3 yang menghasilkan nilai 8, atau ekspresi yang lebih kompleks seperti (a * b) + (c / d), di mana hasilnya bergantung pada nilai variabel yang digunakan.

Operator dalam pemrograman dikategorikan ke dalam beberapa jenis, antara lain:

- **Operator Aritmetika**: Digunakan untuk operasi matematika, seperti + (penjumlahan), - (pengurangan), * (perkalian), / (pembagian), dan % (modulus/sisa pembagian).

- **Operator Relasional (Perbandingan)**: Digunakan untuk membandingkan dua nilai, seperti == (sama dengan), != (tidak sama dengan), > (lebih besar), < (lebih kecil), >= (lebih besar atau sama dengan), dan <= (lebih kecil atau sama dengan).
- **Operator Logika**: Digunakan untuk operasi logika, seperti && (AND), || (OR), dan ! (NOT).
- **Operator Penugasan**: Digunakan untuk memberikan atau memperbarui nilai variabel, seperti = (penugasan langsung), += (penjumlahan dan penugasan), -= (pengurangan dan penugasan), *= (perkalian dan penugasan), dan /= (pembagian dan penugasan).
- **Operator Bitwise**: Digunakan untuk operasi pada level bit, seperti & (AND), | (OR), ^ (XOR), << (left shift), dan >> (right shift).
- **Operator Ternary**: Operator khusus yang memiliki tiga operand, biasanya digunakan sebagai alternatif dari pernyataan if-else, seperti kondisi ? nilai_jika_true : nilai_jika_false.

Dalam pemrograman, pemahaman yang baik tentang operator dan ekspresi sangat penting karena hampir semua kode yang ditulis melibatkan operasi terhadap data. Penggunaan operator yang tepat dapat meningkatkan efisiensi program dan membuat kode lebih mudah dipahami serta lebih optimal dalam pengolahan data.

## 4.2 Jenis-Jenis Operator

Operator dalam pemrograman adalah simbol atau kata kunci yang digunakan untuk melakukan operasi pada nilai atau variabel. Operator ini memungkinkan berbagai manipulasi data, seperti perhitungan matematika, perbandingan nilai, pengambilan keputusan logika, serta operasi pada bit dan struktur data.

Secara umum, operator dalam pemrograman dapat dikategorikan menjadi beberapa jenis utama, yaitu operator aritmatika, perbandingan, logika, penugasan, bitwise, dan keanggotaan.

4.2.1 Operator Aritmatika

Operator aritmatika digunakan untuk melakukan operasi matematika dasar. Operator ini sering digunakan dalam perhitungan angka dan ekspresi matematika dalam program.

Beberapa operator aritmatika yang umum digunakan adalah:

- Penjumlahan (+) → Menambahkan dua nilai.
- Pengurangan (-) → Mengurangi satu nilai dari nilai lainnya.
- Perkalian (*) → Mengalikan dua nilai.
- Pembagian (/) → Membagi satu nilai dengan nilai lainnya, menghasilkan hasil desimal jika perlu.
- Pembagian bulat (//) → Menghasilkan hasil pembagian dalam bentuk bilangan bulat.
- Modulus (%) → Menghasilkan sisa dari hasil pembagian dua angka.

- Pangkat ()** → Menghitung hasil pemangkatan dari suatu angka.

Operator ini sangat penting dalam pengolahan data numerik, baik dalam aplikasi matematika maupun keuangan.

4.2.2 Operator Perbandingan

Operator perbandingan digunakan untuk membandingkan dua nilai dan menghasilkan nilai benar (true) atau salah (false). Operator ini sering digunakan dalam kondisi logika, seperti dalam percabangan dan perulangan.

Berikut adalah beberapa operator perbandingan yang umum digunakan:

- Sama dengan (==) → Memeriksa apakah dua nilai sama.
- Tidak sama dengan (!=) → Memeriksa apakah dua nilai berbeda.
- Lebih besar (>) → Memeriksa apakah suatu nilai lebih besar dari nilai lainnya.
- Lebih kecil (<) → Memeriksa apakah suatu nilai lebih kecil dari nilai lainnya.
- Lebih besar atau sama dengan (>=) → Memeriksa apakah suatu nilai lebih besar atau sama dengan nilai lainnya.
- Lebih kecil atau sama dengan (<=) → Memeriksa apakah suatu nilai lebih kecil atau sama dengan nilai lainnya.

Operator perbandingan sangat penting dalam pengambilan keputusan di dalam program, misalnya dalam proses validasi data atau menentukan alur eksekusi berdasarkan suatu kondisi tertentu.

### 4.2.3 Operator Logika

Operator logika digunakan untuk melakukan operasi logis pada nilai boolean (true atau false). Operator ini sering digunakan dalam struktur kendali program, seperti percabangan if-else dan perulangan.

Beberapa operator logika yang umum digunakan:

- AND (and) → Menghasilkan true jika kedua kondisi bernilai true.
- OR (or) → Menghasilkan true jika salah satu kondisi bernilai true.
- NOT (not) → Membalikkan nilai boolean, true menjadi false dan false menjadi true.

Operator ini sangat berguna dalam menyusun ekspresi logika yang kompleks, misalnya saat memeriksa beberapa kondisi dalam satu waktu.

### 4.2.4 Operator Penugasan

Operator penugasan digunakan untuk memberikan atau memperbarui nilai variabel dalam program.

Beberapa operator penugasan yang umum digunakan:

- Assignment (=) → Menetapkan nilai ke suatu variabel.
- Penambahan dan assignment (+=) → Menambahkan nilai ke variabel yang sudah ada.
- Pengurangan dan assignment (-=) → Mengurangi nilai dari variabel yang sudah ada.
- Perkalian dan assignment (*=) → Mengalikan nilai variabel dengan suatu angka.

- Pembagian dan assignment (/=) → Membagi nilai variabel dengan suatu angka.

Operator ini sering digunakan untuk mengupdate nilai dalam perulangan atau untuk menghitung nilai yang diperbarui secara bertahap.

4.2.5 Operator Bitwise

Operator bitwise digunakan untuk melakukan operasi pada tingkat bit dalam bilangan biner. Operator ini sering digunakan dalam pengolahan data tingkat rendah, seperti dalam sistem operasi, enkripsi, dan pemrograman perangkat keras.

Beberapa operator bitwise yang umum digunakan:

- AND (&) → Melakukan operasi logika AND pada bit-bit dalam dua angka.
- OR (|) → Melakukan operasi logika OR pada bit-bit dalam dua angka.
- XOR (^) → Melakukan operasi exclusive OR, menghasilkan true jika satu bit bernilai true, tetapi bukan keduanya.
- Negasi (~) → Membalikkan semua bit dalam suatu angka.
- Left shift (<<) → Menggeser bit ke kiri, meningkatkan nilai angka.
- Right shift (>>) → Menggeser bit ke kanan, mengurangi nilai angka.

Operator ini sering digunakan dalam optimasi program, kompresi data, dan manipulasi warna dalam pemrograman grafis.

### 4.2.6 Operator Keanggotaan

Operator keanggotaan digunakan untuk memeriksa apakah suatu nilai ada atau tidak dalam suatu koleksi data. Operator ini sering digunakan dalam manipulasi struktur data seperti list, tuple, atau dictionary.

Beberapa operator keanggotaan yang umum digunakan:

- in → Memeriksa apakah suatu nilai ada dalam sebuah koleksi.
- not in → Memeriksa apakah suatu nilai tidak ada dalam sebuah koleksi.

Misalnya, operator ini berguna dalam mencari suatu elemen dalam daftar atau mengecek apakah suatu karakter ada dalam string tertentu.

```python
# 4.2.1 Operator Aritmatika
print("=== Operator Aritmatika ===")
a = 10
b = 3

print("Penjumlahan:", a + b)  # 10 + 3 = 13
print("Pengurangan:", a - b)  # 10 - 3 = 7
print("Perkalian:", a * b)  # 10 * 3 = 30
print("Pembagian:", a / b)  # 10 / 3 = 3.333...
print("Pembagian Bulat:", a // b)  # 10 // 3 = 3
print("Modulus:", a % b)  # 10 % 3 = 1 (sisa bagi)
print("Pangkat:", a ** b)  # 10^3 = 1000

# 4.2.2 Operator Perbandingan
print("\n=== Operator Perbandingan ===")
x = 5
y = 8

print("Sama dengan:", x == y)  # False
print("Tidak sama dengan:", x != y)  # True
print("Lebih besar:", x > y)  # False
print("Lebih kecil:", x < y)  # True
print("Lebih besar atau sama:", x >= y)  # False
print("Lebih kecil atau sama:", x <= y)  # True

# 4.2.3 Operator Logika
print("\n=== Operator Logika ===")
bool1 = True
bool2 = False

print("AND:", bool1 and bool2)  # False
print("OR:", bool1 or bool2)  # True
print("NOT:", not bool1)  # False

# 4.2.4 Operator Penugasan
print("\n=== Operator Penugasan ===")
num = 10
num += 5  # num = num + 5
print("Setelah += 5:", num)  # 15

num -= 3  # num = num - 3
print("Setelah -= 3:", num)  # 12

num *= 2  # num = num * 2
print("Setelah *= 2:", num)  # 24

num /= 4  # num = num / 4
print("Setelah /= 4:", num)  # 6.0

# 4.2.5 Operator Bitwise
print("\n=== Operator Bitwise ===")
bit1 = 5  # 5 dalam biner = 0101
bit2 = 3  # 3 dalam biner = 0011

print("AND:", bit1 & bit2)  # 0001 -> 1
print("OR:", bit1 | bit2)  # 0111 -> 7
print("XOR:", bit1 ^ bit2)  # 0110 -> 6
print("Negasi (~5):", ~bit1)  # -(5+1) -> -6
print("Left Shift (5 << 1):", bit1 << 1)  # 1010 -> 10
print("Right Shift (5 >> 1):", bit1 >> 1)  # 0010 -> 2

# 4.2.6 Operator Keanggotaan
print("\n=== Operator Keanggotaan ===")
my_list = [1, 2, 3, 4, 5]

print("Apakah 3 ada di list?", 3 in my_list)  # True
print("Apakah 6 tidak ada di list?", 6 not in my_list)  # True

my_string = "Hello Python"
print("Apakah 'Py' ada dalam string?", "Py" in my_string)  # True
print("Apakah 'Java' tidak ada dalam string?", "Java" not in my_string)  # True

```

## 4.3 Ekspresi dalam Pemrograman

Ekspresi dalam pemrograman adalah kombinasi dari operand dan operator yang digunakan untuk menghasilkan suatu nilai. Ekspresi dapat berupa perhitungan matematis, perbandingan nilai, atau evaluasi logika yang menentukan jalannya program. Dengan adanya ekspresi, program dapat melakukan berbagai operasi untuk memproses data dan menghasilkan output yang sesuai.

Ekspresi digunakan dalam berbagai konteks, seperti dalam perhitungan matematis, pengambilan keputusan, dan evaluasi kondisi dalam suatu program. Ekspresi juga dapat digunakan dalam perulangan dan fungsi untuk memanipulasi data sesuai kebutuhan.

### 4.3.1 Ekspresi Aritmatika

Ekspresi aritmatika digunakan untuk melakukan operasi matematika seperti penjumlahan, pengurangan, perkalian, dan pembagian. Ekspresi ini melibatkan angka sebagai operand dan simbol matematika sebagai operator. Hasil dari ekspresi aritmatika selalu berupa nilai numerik, baik dalam bentuk bilangan bulat maupun desimal.

Ekspresi ini sering digunakan dalam berbagai aplikasi, seperti perhitungan keuangan, pengolahan data statistik, dan perhitungan dalam ilmu sains serta teknik. Penggunaan tanda kurung dalam ekspresi aritmatika juga dapat mengubah prioritas operasi untuk mendapatkan hasil yang sesuai.

### 4.3.2 Ekspresi Perbandingan

Ekspresi perbandingan digunakan untuk membandingkan dua nilai dan menghasilkan hasil dalam bentuk nilai logika, yaitu benar atau salah. Operator perbandingan memungkinkan program untuk menentukan hubungan antara dua operand, seperti apakah suatu nilai lebih besar, lebih kecil, atau sama dengan nilai lainnya.

Ekspresi perbandingan sering digunakan dalam pengambilan keputusan di dalam program, seperti dalam pernyataan kondisional yang mengatur alur eksekusi berdasarkan hasil evaluasi ekspresi. Dengan adanya ekspresi perbandingan, program dapat menjalankan instruksi yang berbeda tergantung pada kondisi yang diberikan.

### 4.3.3 Ekspresi Logika

Ekspresi logika digunakan untuk menggabungkan atau membandingkan beberapa kondisi menggunakan operator logika. Operator ini memungkinkan program untuk mengevaluasi lebih dari satu kondisi dalam satu ekspresi dan menghasilkan nilai logika sebagai hasilnya.

Ekspresi logika sering digunakan dalam struktur kontrol, seperti pernyataan kondisional dan perulangan, untuk menentukan apakah suatu blok kode akan dieksekusi atau tidak. Dengan memanfaatkan ekspresi logika, program dapat membuat keputusan yang lebih kompleks berdasarkan beberapa kondisi yang harus dipenuhi secara bersamaan atau secara alternatif.

Penggunaan ekspresi dalam pemrograman sangat penting karena memungkinkan program untuk melakukan berbagai operasi, dari perhitungan dasar hingga pengambilan keputusan yang

kompleks. Dengan memahami berbagai jenis ekspresi, seorang programmer dapat menulis kode yang lebih efisien, fleksibel, dan mudah dipahami.

## 4.4 Prioritas Operator

Dalam pemrograman, ketika sebuah ekspresi mengandung lebih dari satu operator, urutan eksekusi ditentukan berdasarkan prioritas operator. Prioritas ini memastikan bahwa operasi dilakukan dalam urutan yang benar, menghindari kesalahan perhitungan, dan memastikan hasil yang sesuai dengan aturan matematika.

Operator dengan prioritas lebih tinggi akan dieksekusi lebih dahulu dibandingkan operator dengan prioritas lebih rendah. Jika terdapat beberapa operator dengan prioritas yang sama dalam satu ekspresi, evaluasi dilakukan dari kiri ke kanan, kecuali untuk operator pangkat yang dievaluasi dari kanan ke kiri.

Berikut adalah urutan prioritas operator dari yang tertinggi ke terendah:

1. Kurung () → Digunakan untuk menentukan bagian ekspresi yang harus dievaluasi terlebih dahulu. Operasi dalam tanda kurung selalu dieksekusi lebih dahulu dibandingkan operator lain.
2. Pangkat ** → Digunakan untuk operasi perpangkatan.
3. Perkalian, pembagian, pembagian bulat, dan modulus *, /, //, % → Operator ini memiliki prioritas yang lebih tinggi dibandingkan penjumlahan dan pengurangan.

4. Penjumlahan dan pengurangan +, - → Berada di bawah perkalian dan pembagian dalam hal prioritas.
5. Operator perbandingan >, <, >=, <=, ==, != → Digunakan untuk membandingkan dua nilai dan menentukan apakah suatu kondisi bernilai benar atau salah.
6. Operator logika not → Digunakan untuk melakukan negasi terhadap suatu kondisi.
7. Operator logika and → Memiliki prioritas lebih tinggi dibandingkan or karena harus mengevaluasi kedua operand sebelum memberikan hasil.
8. Operator logika or → Berada pada urutan prioritas terendah di antara operator logika.

Dengan memahami prioritas operator, kita dapat menulis ekspresi yang lebih jelas dan menghindari ambiguitas dalam perhitungan. Jika ada keraguan mengenai urutan eksekusi suatu ekspresi, penggunaan tanda kurung dapat membantu memastikan bahwa operasi dilakukan sesuai dengan yang diinginkan.

```python
# Prioritas operator dalam Python
print("=== Prioritas Operator dalam Eksekusi ===")

# 1. Kurung ()
hasil_1 = (2 + 3) * 4  # Operasi dalam kurung dilakukan lebih dulu
print("(2 + 3) * 4 =", hasil_1)  # 20

# 2. Pangkat **
hasil_2 = 2 ** 3 ** 2  # Evaluasi dari kanan ke kiri: 3**2 = 9 → 2**9 = 512
print("2 ** 3 ** 2 =", hasil_2)  # 512

# 3. Perkalian, Pembagian, Pembagian Bulat, Modulus *, /, //, %
hasil_3 = 10 + 3 * 2  # Perkalian dilakukan lebih dulu: 3*2 = 6 → 10+6 = 16
print("10 + 3 * 2 =", hasil_3)  # 16

hasil_4 = 20 / 4 + 5  # Pembagian lebih dulu: 20/4 = 5 → 5+5 = 10
print("20 / 4 + 5 =", hasil_4)  # 10.0

hasil_5 = 15 // 4 % 3  # Pembagian bulat lebih dulu: 15//4 = 3 → 3%3 = 0
print("15 // 4 % 3 =", hasil_5)  # 0

# 4. Penjumlahan dan Pengurangan +, -
hasil_6 = 8 - 3 + 2  # Evaluasi dari kiri ke kanan: (8-3) = 5 → 5+2 = 7
print("8 - 3 + 2 =", hasil_6)  # 7

# 5. Operator Perbandingan >, <, >=, <=, ==, !=
hasil_7 = 10 > 5 + 2  # Penjumlahan dulu: 5+2 = 7 → 10 > 7 = True
print("10 > 5 + 2 =", hasil_7)  # True

# 6. Operator Logika NOT
hasil_8 = not 10 > 5  # 10 > 5 = True → not True = False
print("not (10 > 5) =", hasil_8)  # False

# 7. Operator Logika AND
hasil_9 = True and False or True  # AND lebih dulu: False or True = True
print("True and False or True =", hasil_9)  # True

# 8. Operator Logika OR
hasil_10 = False or False or True  # OR dilakukan dari kiri ke kanan
print("False or False or True =", hasil_10)  # True

```

## 4.5 Latihan Soal

1. Apa yang dimaksud dengan operator dalam pemrograman?
2. Sebutkan dan jelaskan tiga jenis operator dalam pemrograman.

3. Bagaimana cara menggunakan operator perbandingan dalam ekspresi?
4. Tuliskan contoh penggunaan operator logika dalam Python.
5. Jelaskan konsep prioritas operator dalam evaluasi ekspresi.

# Bab 5: Struktur Kontrol Alur Program

## 5.1 Pengertian Struktur Kontrol Alur Program

Struktur kontrol alur program adalah mekanisme dalam pemrograman yang mengatur urutan eksekusi instruksi berdasarkan kondisi tertentu. Dengan adanya struktur kontrol ini, program dapat mengambil keputusan, mengeksekusi bagian kode tertentu, atau mengulang proses sesuai kebutuhan. Struktur kontrol sangat penting dalam pemrograman karena memungkinkan program berjalan secara dinamis dan responsif terhadap berbagai kondisi yang terjadi selama eksekusi.

Dalam pemrograman, struktur kontrol alur program terbagi menjadi tiga jenis utama, yaitu **struktur kontrol berurutan, percabangan (kondisional), dan perulangan (iterasi)**.

1. **Struktur Berurutan** adalah bentuk paling dasar dari eksekusi program, di mana instruksi dijalankan satu per satu dari atas ke bawah tanpa adanya percabangan atau pengulangan. Ini digunakan dalam kode sederhana yang tidak memerlukan pengambilan keputusan atau pengulangan.
2. **Struktur Percabangan (Kondisional)** memungkinkan program untuk mengambil keputusan berdasarkan suatu

kondisi. Contoh umum dari struktur ini adalah pernyataan **if, if-else, dan switch-case**. Dengan percabangan, program dapat mengeksekusi blok kode yang berbeda tergantung pada hasil evaluasi suatu kondisi, misalnya jika suhu lebih dari 30°C, program akan menampilkan pesan "Cuaca panas", sedangkan jika lebih rendah, akan menampilkan "Cuaca sejuk".

3. **Struktur Perulangan (Iterasi)** digunakan untuk mengeksekusi sekelompok instruksi secara berulang berdasarkan kondisi tertentu. Contoh perulangan yang umum digunakan adalah **for, while, dan do-while**. Dengan struktur ini, program dapat mengulangi eksekusi kode tanpa harus menulis ulang instruksi yang sama, misalnya dalam kasus pengolahan data dalam jumlah besar atau proses perhitungan yang berulang.

Pemahaman yang baik tentang struktur kontrol alur program sangat penting bagi seorang programmer karena membantu dalam membuat program yang lebih efisien, fleksibel, dan mudah dikelola. Dengan menggunakan struktur kontrol yang tepat, program dapat berjalan secara optimal sesuai dengan tujuan yang diinginkan

## 5.2 Jenis-Jenis Struktur Kontrol Alur Program

Struktur kontrol alur program adalah mekanisme dalam pemrograman yang menentukan bagaimana suatu program dieksekusi. Struktur ini memungkinkan program mengambil

keputusan, melakukan perulangan, dan mengontrol jalannya eksekusi berdasarkan kondisi tertentu.

Secara umum, struktur kontrol dalam pemrograman terbagi menjadi tiga jenis utama: Struktur Pemilihan (Selection), Struktur Perulangan (Looping), dan Struktur Percabangan (Branching).

### 5.2.1 Struktur Pemilihan (Selection)

Struktur pemilihan digunakan ketika program harus membuat keputusan berdasarkan suatu kondisi. Dengan kata lain, program akan memilih salah satu dari beberapa jalur eksekusi berdasarkan hasil evaluasi suatu ekspresi logika.

Beberapa jenis struktur pemilihan yang umum digunakan:

1. If Statement – Mengeksekusi blok kode jika suatu kondisi terpenuhi.
2. If-Else Statement – Menjalankan satu blok kode jika kondisi benar, dan blok kode lain jika kondisi salah.
3. If-Else If-Else Statement – Digunakan untuk menangani beberapa kondisi dengan urutan prioritas tertentu.
4. Switch (atau Case Statement, tergantung bahasa pemrograman) – Memilih satu dari banyak kemungkinan berdasarkan nilai ekspresi tertentu.

Struktur pemilihan ini sangat berguna dalam membuat program yang lebih fleksibel dan responsif terhadap berbagai input pengguna.

### 5.2.2 Struktur Perulangan (Looping)

Struktur perulangan digunakan untuk menjalankan suatu blok kode secara berulang sampai kondisi tertentu terpenuhi. Ini

berguna untuk menghindari pengulangan kode secara manual dan meningkatkan efisiensi program.

Jenis-jenis struktur perulangan yang umum digunakan:

1. For Loop – Digunakan ketika jumlah iterasi sudah diketahui sebelumnya.
2. While Loop – Digunakan ketika perulangan harus terus berjalan selama suatu kondisi masih terpenuhi.
3. Do-While Loop – Mirip dengan while loop, tetapi memastikan bahwa blok kode dijalankan setidaknya sekali sebelum kondisi diperiksa.

Struktur perulangan ini sering digunakan dalam proses seperti iterasi daftar, membaca file, atau menjalankan program berdasarkan input pengguna.

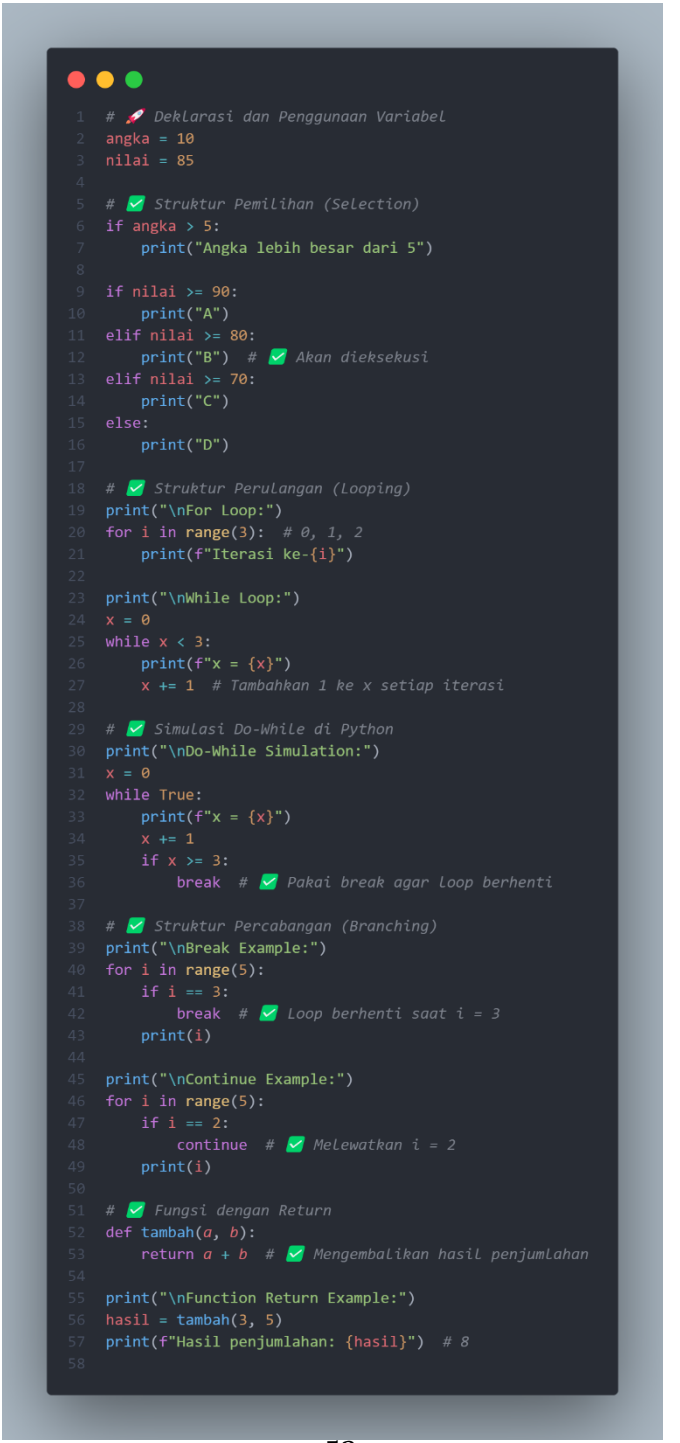
5.2.3 Struktur Percabangan (Branching)

Struktur percabangan digunakan untuk mengubah jalannya eksekusi program berdasarkan kondisi tertentu. Percabangan memungkinkan program untuk melompati atau menghentikan eksekusi kode berdasarkan suatu perintah.

Beberapa jenis struktur percabangan yang sering digunakan:

1. Break – Menghentikan eksekusi dari suatu perulangan atau switch-case lebih awal.
2. Continue – Melewatkan iterasi saat ini dalam perulangan dan langsung lanjut ke iterasi berikutnya.
3. Return – Menghentikan eksekusi fungsi dan mengembalikan nilai (jika ada).

Struktur percabangan ini penting untuk mengontrol bagaimana program dieksekusi dalam skenario yang lebih kompleks, seperti menangani kesalahan atau mengoptimalkan kinerja program.



```python
# 🚀 Deklarasi dan Penggunaan Variabel
angka = 10
nilai = 85

# ✅ Struktur Pemilihan (Selection)
if angka > 5:
    print("Angka lebih besar dari 5")

if nilai >= 90:
    print("A")
elif nilai >= 80:
    print("B")  # ✅ Akan dieksekusi
elif nilai >= 70:
    print("C")
else:
    print("D")

# ✅ Struktur Perulangan (Looping)
print("\nFor Loop:")
for i in range(3):  # 0, 1, 2
    print(f"Iterasi ke-{i}")

print("\nWhile Loop:")
x = 0
while x < 3:
    print(f"x = {x}")
    x += 1  # Tambahkan 1 ke x setiap iterasi

# ✅ Simulasi Do-While di Python
print("\nDo-While Simulation:")
x = 0
while True:
    print(f"x = {x}")
    x += 1
    if x >= 3:
        break  # ✅ Pakai break agar loop berhenti

# ✅ Struktur Percabangan (Branching)
print("\nBreak Example:")
for i in range(5):
    if i == 3:
        break  # ✅ Loop berhenti saat i = 3
    print(i)

print("\nContinue Example:")
for i in range(5):
    if i == 2:
        continue  # ✅ Melewatkan i = 2
    print(i)

# ✅ Fungsi dengan Return
def tambah(a, b):
    return a + b  # ✅ Mengembalikan hasil penjumlahan

print("\nFunction Return Example:")
hasil = tambah(3, 5)
print(f"Hasil penjumlahan: {hasil}")  # 8

```

## 5.3 Struktur Pemilihan (Selection)

Struktur pemilihan adalah konsep dalam pemrograman yang memungkinkan program untuk mengambil keputusan berdasarkan kondisi tertentu. Dengan menggunakan struktur ini, program dapat menentukan jalannya eksekusi berdasarkan nilai atau pernyataan logika yang dievaluasi. Struktur pemilihan sangat penting dalam pemrograman karena memungkinkan pengambilan keputusan yang fleksibel dan dinamis sesuai dengan input atau keadaan program.

Struktur pemilihan biasanya digunakan dalam berbagai situasi, seperti validasi input, pengambilan keputusan dalam permainan, perhitungan logika, serta kontrol alur eksekusi dalam aplikasi berbasis data.

5.3.1 Pernyataan If (If Statement)

Pernyataan if digunakan untuk mengeksekusi suatu blok kode hanya jika kondisi yang diberikan bernilai benar (true). Jika kondisi tidak terpenuhi (false), maka blok kode tidak akan dijalankan.

Struktur ini memungkinkan program untuk melakukan tindakan tertentu hanya ketika suatu kondisi spesifik terjadi. Sebagai contoh, dalam aplikasi perbankan, program dapat mengecek apakah saldo cukup sebelum mengizinkan transaksi. Jika saldo mencukupi, maka transaksi diproses; jika tidak, transaksi dibatalkan.

5.3.2 Pernyataan If-Else (If-Else Statement)

Pernyataan if-else digunakan ketika terdapat dua kemungkinan hasil dari suatu kondisi. Jika kondisi yang dievaluasi

bernilai benar, maka blok kode pertama akan dieksekusi. Sebaliknya, jika kondisi bernilai salah, maka blok kode alternatif akan dijalankan.

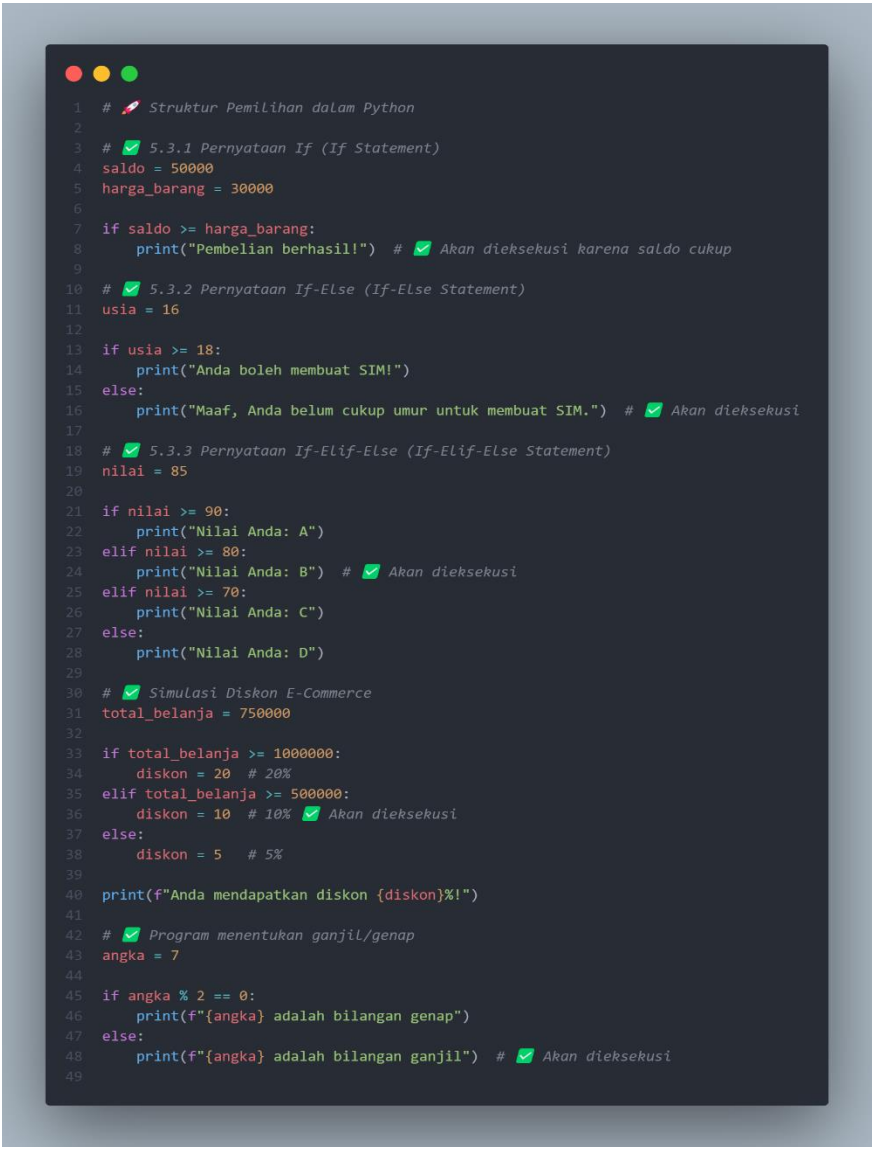
Struktur ini sangat berguna dalam situasi di mana program harus memberikan respons berbeda berdasarkan hasil evaluasi suatu kondisi. Misalnya, dalam sistem e-commerce, program dapat menentukan apakah pengguna mendapatkan diskon berdasarkan jumlah pembelian mereka. Jika jumlah pembelian memenuhi syarat, diskon diberikan; jika tidak, harga normal tetap berlaku.

5.3.3 Pernyataan If-Elif-Else (If-Elif-Else Statement)

Pernyataan if-elif-else digunakan ketika terdapat lebih dari dua kondisi yang perlu dievaluasi. Pernyataan elif (else if) memungkinkan program untuk memeriksa beberapa kondisi secara berurutan. Jika kondisi pertama tidak terpenuhi, maka program akan mengevaluasi kondisi berikutnya, dan seterusnya. Jika tidak ada kondisi yang terpenuhi, maka blok kode di dalam else akan dijalankan sebagai pilihan terakhir.

Struktur ini sangat berguna dalam situasi di mana terdapat banyak kemungkinan keputusan berdasarkan nilai tertentu. Misalnya, dalam sistem akademik, program dapat menentukan nilai huruf berdasarkan rentang skor ujian yang diperoleh siswa. Jika nilai berada dalam rentang tertentu, maka program akan menetapkan nilai huruf yang sesuai.

Struktur pemilihan dalam pemrograman sangat penting karena memungkinkan kode menjadi lebih dinamis dan responsif terhadap berbagai kondisi yang mungkin terjadi selama eksekusi. Dengan memahami dan menerapkan struktur pemilihan dengan baik,



```python
# 🚀 Struktur Pemilihan dalam Python

# ✅ 5.3.1 Pernyataan If (If Statement)
saldo = 50000
harga_barang = 30000

if saldo >= harga_barang:
    print("Pembelian berhasil!")  # ✅ Akan dieksekusi karena saldo cukup

# ✅ 5.3.2 Pernyataan If-Else (If-Else Statement)
usia = 16

if usia >= 18:
    print("Anda boleh membuat SIM!")
else:
    print("Maaf, Anda belum cukup umur untuk membuat SIM.")  # ✅ Akan dieksekusi

# ✅ 5.3.3 Pernyataan If-Elif-Else (If-Elif-Else Statement)
nilai = 85

if nilai >= 90:
    print("Nilai Anda: A")
elif nilai >= 80:
    print("Nilai Anda: B")  # ✅ Akan dieksekusi
elif nilai >= 70:
    print("Nilai Anda: C")
else:
    print("Nilai Anda: D")

# ✅ Simulasi Diskon E-Commerce
total_belanja = 750000

if total_belanja >= 1000000:
    diskon = 20  # 20%
elif total_belanja >= 500000:
    diskon = 10  # 10% ✅ Akan dieksekusi
else:
    diskon = 5   # 5%

print(f"Anda mendapatkan diskon {diskon}%!")

# ✅ Program menentukan ganjil/genap
angka = 7

if angka % 2 == 0:
    print(f"{angka} adalah bilangan genap")
else:
    print(f"{angka} adalah bilangan ganjil")  # ✅ Akan dieksekusi

```

program dapat menjadi lebih efisien, fleksibel, dan mudah diadaptasi sesuai kebutuhan.

## 5.4 Struktur Perulangan (Looping)

Dalam pemrograman, struktur perulangan atau looping digunakan untuk menjalankan suatu blok kode secara berulang selama kondisi tertentu masih terpenuhi. Perulangan memungkinkan eksekusi kode yang sama tanpa harus menuliskannya secara berulang-ulang, sehingga membuat program lebih efisien dan mudah dikelola.

Terdapat beberapa jenis perulangan yang umum digunakan:

1. Perulangan for

   Perulangan ini digunakan ketika jumlah iterasi sudah diketahui sebelumnya, seperti dalam iterasi pada rentang angka atau elemen dalam suatu koleksi data.

2. Perulangan while

   Perulangan ini dieksekusi selama kondisi tertentu masih bernilai benar (true). Biasanya digunakan ketika jumlah iterasi belum diketahui dengan pasti dan bergantung pada suatu kondisi yang dapat berubah selama eksekusi program.

3. Perulangan bersarang (Nested Loop)

   Perulangan ini terjadi ketika satu perulangan berada di dalam perulangan lainnya. Hal ini sering digunakan dalam pengolahan data bertingkat, seperti tabel atau array multidimensi.

Perulangan sangat berguna dalam berbagai aspek pemrograman, seperti pengolahan data, pencarian, atau pengulangan tugas tertentu secara otomatis. Untuk memastikan perulangan tidak berjalan tanpa henti, diperlukan kondisi penghentian yang jelas agar

program tetap berjalan dengan efisien dan tidak mengalami infinite loop (perulangan tak terbatas).

```python
# Struktur Perulangan dalam Python

# 1. Perulangan For
print("Perulangan For: Menampilkan angka 1-5")
for i in range(1, 6):
    print(i)

# 2. Perulangan While
print("\nPerulangan While: Menghitung mundur dari 5")
countdown = 5
while countdown > 0:
    print(countdown)
    countdown -= 1  # Mengurangi nilai countdown agar tidak infinite loop

# 3. Perulangan Bersarang (Nested Loop)
print("\nPerulangan Bersarang: Pola Bintang")
baris = 5
for i in range(1, baris + 1):
    for j in range(1, i + 1):
        print("*", end=" ")
    print()  # Pindah ke baris berikutnya

# 4. Loop dengan Break (Menghentikan Perulangan)
print("\nLoop dengan Break: Berhenti di angka 3")
for i in range(1, 6):
    if i == 3:
        print("Berhenti di:", i)
        break
    print(i)

# 5. Loop dengan Continue (Melewati Iterasi Tertentu)
print("\nLoop dengan Continue: Lewati angka 3")
for i in range(1, 6):
    if i == 3:
        continue  # Lewati angka 3
    print(i)

# 6. Loop untuk Iterasi List
buah = ["Apel", "Jeruk", "Mangga"]
print("\nLoop pada List Buah:")
for item in buah:
    print(item)

```

## 5.5 Latihan Soal

1. Jelaskan perbedaan antara struktur pemilihan, perulangan, dan percabangan.
2. Buat contoh penggunaan if-elif-else dalam Python.
3. Apa perbedaan utama antara for loop dan while loop?
4. Jelaskan cara kerja break dan continue dalam perulangan.
5. Buat program sederhana yang menggunakan struktur kontrol untuk menentukan bilangan ganjil dan genap.

# Bab 6: Fungsi dan Modularisasi Kode

## 6.1 Pengertian Fungsi dan Modularisasi Kode

Dalam pemrograman, fungsi dan modularisasi kode merupakan konsep penting yang digunakan untuk meningkatkan efisiensi dan keterbacaan program. **Fungsi** adalah blok kode yang dirancang untuk menjalankan tugas tertentu dan dapat digunakan kembali di berbagai bagian program. Dengan menggunakan fungsi, seorang programmer dapat menghindari penulisan ulang kode yang sama, sehingga membuat program lebih ringkas, terstruktur, dan mudah dikelola.

Setiap fungsi biasanya memiliki nama, parameter (opsional), dan nilai yang dikembalikan (jika diperlukan). Fungsi dapat berupa **fungsi bawaan (built-in)** yang sudah tersedia dalam bahasa pemrograman tertentu, atau **fungsi yang didefinisikan sendiri (user-defined function)** sesuai dengan kebutuhan program. Misalnya, dalam bahasa Python, terdapat fungsi bawaan seperti print() untuk menampilkan output, sedangkan programmer juga dapat membuat fungsi sendiri untuk menghitung luas lingkaran atau memproses data tertentu.

Sementara itu, **modularisasi kode** adalah konsep dalam pemrograman yang membagi kode menjadi bagian-bagian kecil

yang lebih terorganisir. Dengan modularisasi, kode program tidak ditulis dalam satu blok besar, tetapi dipisahkan ke dalam beberapa fungsi atau modul yang masing-masing memiliki tugas spesifik. Hal ini membuat program lebih mudah dibaca, dipahami, dan dikelola, terutama ketika program menjadi lebih kompleks.

Keuntungan utama dari penggunaan fungsi dan modularisasi kode antara lain:

1. **Mempermudah Pemeliharaan** – Kode yang terorganisir dengan baik lebih mudah untuk diperbaiki, diperbarui, atau dikembangkan.
2. **Mengurangi Duplikasi Kode** – Dengan menggunakan fungsi, kode yang sama tidak perlu ditulis berulang kali, sehingga menghemat waktu dan mengurangi risiko kesalahan.
3. **Meningkatkan Keterbacaan** – Program yang terstruktur dengan baik lebih mudah dipahami oleh programmer lain maupun oleh diri sendiri di masa mendatang.
4. **Mempermudah Debugging** – Jika terjadi kesalahan, programmer hanya perlu memeriksa bagian kode tertentu tanpa harus mencari di seluruh program.
5. **Memungkinkan Penggunaan Kembali Kode** – Fungsi yang telah dibuat dapat digunakan kembali di berbagai bagian program atau bahkan dalam proyek yang berbeda.

Dengan menerapkan konsep fungsi dan modularisasi kode, programmer dapat membuat program yang lebih efisien, fleksibel, dan mudah dikembangkan dalam jangka panjang.

## 6.2 Kegunaan Fungsi sebagai Salah Satu Konsep Fundamental

Fungsi adalah salah satu konsep fundamental dalam pemrograman yang memungkinkan pengelompokan blok kode menjadi unit yang lebih kecil dan lebih modular. Dengan menggunakan fungsi, pengembangan perangkat lunak menjadi lebih terstruktur, efisien, dan mudah dikelola.

Berikut adalah beberapa manfaat utama penggunaan fungsi dalam pemrograman:

### 6.2.1 Mengurangi Redundansi Kode

Salah satu manfaat terbesar dari fungsi adalah menghindari duplikasi kode. Jika suatu bagian kode perlu digunakan berkali-kali, cukup dibuat dalam satu fungsi dan dipanggil kapan saja diperlukan. Ini menghemat waktu dalam penulisan kode serta mengurangi kemungkinan kesalahan akibat pengulangan kode yang tidak konsisten.

Contoh manfaat dalam skenario nyata:

- Dalam program perhitungan matematika, fungsi dapat digunakan untuk operasi seperti penjumlahan, pengurangan, atau faktorial.
- Dalam pengolahan data, fungsi dapat digunakan untuk membaca, menulis, atau memproses informasi tanpa menulis ulang kode yang sama.

### 6.2.2 Meningkatkan Keterbacaan Kode

Kode yang panjang dan kompleks sulit dibaca serta dipahami. Dengan membagi kode ke dalam fungsi-fungsi yang lebih kecil, program menjadi lebih modular dan mudah dipahami.

Keuntungan dalam keterbacaan ini sangat penting untuk:

- Kolaborasi dalam tim, karena anggota tim dapat memahami bagian-bagian kode dengan lebih cepat.
- Debugging yang lebih mudah, karena setiap bagian kode memiliki tanggung jawab yang jelas.
- Pengembangan jangka panjang, karena kode yang terstruktur baik lebih mudah dimodifikasi dan dikembangkan.

### 6.2.3 Memudahkan Pemeliharaan dan Debugging

Program yang menggunakan fungsi lebih mudah dikelola dibandingkan kode yang tidak terstruktur. Jika terjadi kesalahan atau ingin dilakukan perubahan, cukup perbaiki fungsi yang terkait tanpa harus menelusuri seluruh kode program.

Manfaat dalam pemeliharaan kode meliputi:

- Memperbaiki bug lebih cepat: Jika suatu fitur mengalami masalah, cukup fokus pada fungsi tertentu tanpa mengubah keseluruhan program.
- Meningkatkan fleksibilitas: Jika suatu algoritma perlu diubah atau ditingkatkan, cukup dengan memperbarui fungsi terkait tanpa mengganggu bagian lain dari program.

### 6.2.4 Meningkatkan Efisiensi dan Penggunaan Ulang Kode

Dengan adanya fungsi, kode dapat digunakan kembali tanpa perlu menulis ulang. Ini tidak hanya menghemat waktu, tetapi juga meningkatkan efisiensi penggunaan sumber daya.

Keuntungan utama dari penggunaan ulang kode adalah:

- Mengurangi waktu pengembangan, karena fitur yang sering digunakan bisa langsung dipanggil tanpa perlu menulis ulang.
- Meningkatkan konsistensi, karena fungsi yang sama memberikan hasil yang sama di berbagai bagian program.
- Membantu dalam pengembangan skala besar, karena fungsi yang dibuat dalam satu proyek bisa digunakan kembali dalam proyek lain.

### 6.2.5 Mempermudah Penggunaan Modularitas dan Skalabilitas

Dalam proyek besar, fungsi memungkinkan pengembangan yang lebih modular. Setiap fungsi dapat dikembangkan secara terpisah, memungkinkan tim untuk bekerja pada bagian yang berbeda secara bersamaan tanpa saling mengganggu.

Skalabilitas dalam pemrograman didukung oleh:

- Modularisasi kode, yang memungkinkan bagian program dikembangkan atau diperbaiki secara independen.
- Penggunaan library atau API, di mana fungsi yang sudah ada bisa langsung digunakan tanpa perlu mengembangkan dari nol.

## 6.3 Cara Membuat dan Menggunakan Fungsi

Fungsi adalah blok kode yang dirancang untuk menjalankan tugas tertentu dan dapat digunakan kembali di berbagai bagian program. Dengan menggunakan fungsi, kode menjadi lebih modular, mudah dibaca, dan lebih efisien karena menghindari duplikasi kode. Fungsi juga memudahkan proses debugging dan pemeliharaan program.

Dalam pemrograman, fungsi biasanya terdiri dari tiga bagian utama: pendeklarasian, parameter (opsional), dan pemanggilan. Fungsi dapat dibuat untuk melakukan berbagai operasi, mulai dari perhitungan sederhana hingga pemrosesan data yang kompleks.

6.3.1 Deklarasi dan Definisi Fungsi

Fungsi harus dideklarasikan terlebih dahulu sebelum dapat digunakan dalam program. Saat mendeklarasikan fungsi, diperlukan nama fungsi yang unik dan, jika perlu, parameter yang akan digunakan sebagai input. Struktur fungsi biasanya terdiri dari nama fungsi, parameter (jika ada), dan serangkaian perintah yang akan dijalankan ketika fungsi dipanggil.

Fungsi dapat dideklarasikan dengan atau tanpa parameter. Jika menggunakan parameter, nilai yang dikirim ke fungsi saat pemanggilan akan digunakan dalam proses eksekusi. Parameter memungkinkan fungsi menjadi lebih fleksibel dan dapat menangani berbagai jenis input.

### 6.3.2 Memanggil Fungsi

Setelah fungsi dideklarasikan, fungsi tersebut dapat dipanggil kapan saja dalam program. Pemanggilan fungsi dilakukan dengan menyebutkan nama fungsi diikuti oleh tanda kurung, dengan memasukkan nilai sebagai argumen jika fungsi menerima parameter.

Ketika fungsi dipanggil, program akan mengeksekusi perintah-perintah yang ada dalam fungsi tersebut dan kemudian kembali ke lokasi pemanggilan setelah selesai menjalankan tugasnya. Fungsi dapat dipanggil berkali-kali, memungkinkan kode menjadi lebih ringkas dan terorganisir.

### 6.3.3 Fungsi dengan Nilai Kembali (Return Value)

Fungsi dapat mengembalikan nilai menggunakan pernyataan return. Nilai yang dikembalikan oleh fungsi dapat digunakan di bagian lain dari program, misalnya dalam perhitungan atau dalam logika pengambilan keputusan.

Menggunakan return dalam fungsi sangat berguna ketika hasil dari fungsi perlu digunakan lebih lanjut dalam program. Jika fungsi tidak memiliki return, maka fungsi hanya akan menjalankan tugasnya tanpa memberikan nilai kembali.

### 6.3.4 Manfaat Penggunaan Fungsi dalam Pemrograman

Penggunaan fungsi dalam pemrograman memiliki banyak manfaat, di antaranya:

- Modularitas: Program dapat dibagi menjadi beberapa bagian kecil yang lebih mudah dikelola.
- Reusability: Fungsi dapat digunakan kembali di berbagai bagian program tanpa perlu menulis ulang kode.

- Kemudahan Pemeliharaan: Jika ada kesalahan atau perubahan yang perlu dilakukan, cukup memperbaiki satu fungsi tanpa harus mengedit banyak bagian kode.
- Meningkatkan Keterbacaan: Kode yang menggunakan fungsi lebih terstruktur dan mudah dipahami dibandingkan kode tanpa fungsi.

Dengan memahami konsep dan cara menggunakan fungsi, pengembang dapat menulis program yang lebih efisien, modular, dan mudah dikelola.

```python
# 6.3.1 Deklarasi dan Definisi Fungsi
# Fungsi tanpa parameter
def sapa():
    print("Halo, selamat datang di program ini!")

# Fungsi dengan parameter
def luas_persegi(sisi):
    return sisi * sisi

# Fungsi dengan lebih dari satu parameter
def hitung_luas_persegi_panjang(panjang, lebar):
    return panjang * lebar

# 6.3.2 Memanggil Fungsi
sapa()  # Memanggil fungsi tanpa parameter

hasil_luas_persegi = luas_persegi(5)  # Memanggil fungsi dengan parameter
print(f"Luas persegi dengan sisi 5: {hasil_luas_persegi}")

hasil_luas_persegi_panjang = hitung_luas_persegi_panjang(4, 6)
print(f"Luas persegi panjang dengan panjang 4 dan lebar 6: {hasil_luas_persegi_panjang}")

# 6.3.3 Fungsi dengan Nilai Kembali (Return Value)
def hitung_keliling_persegi(sisi):
    return 4 * sisi

keliling = hitung_keliling_persegi(7)
print(f"Keliling persegi dengan sisi 7: {keliling}")

# 6.3.4 Manfaat Penggunaan Fungsi dalam Pemrograman
# Menggunakan fungsi untuk menghitung luas beberapa persegi dengan ukuran berbeda
sisi_list = [2, 4, 6, 8]
for s in sisi_list:
    print(f"Luas persegi dengan sisi {s}: {luas_persegi(s)}")
```

## 6.4 Parameter dan Argumen dalam Fungsi

Dalam pemrograman, fungsi digunakan untuk mengorganisir kode agar lebih modular dan dapat digunakan kembali. Salah satu fitur penting dalam fungsi adalah parameter, yang memungkinkan pengguna mengirimkan data ke dalam fungsi agar dapat diproses. Data yang dikirim ke dalam fungsi disebut sebagai argumen.

Terdapat beberapa jenis parameter yang dapat digunakan dalam sebuah fungsi:

1. Parameter Posisional

   Parameter ini mengharuskan argumen dikirim berdasarkan urutan yang sesuai dengan deklarasi parameter dalam fungsi. Jika urutannya tidak sesuai, hasil yang diperoleh bisa berbeda atau menyebabkan error.

2. Parameter Default

   Parameter ini memiliki nilai awal atau default yang akan digunakan jika tidak ada argumen yang diberikan saat pemanggilan fungsi. Hal ini berguna untuk membuat fungsi lebih fleksibel dan mengurangi kebutuhan memasukkan semua argumen secara manual.

3. Parameter Keyword

   Parameter ini memungkinkan pengguna untuk mengirimkan nilai argumen dengan menyebutkan nama parameter secara eksplisit. Dengan cara ini, urutan argumen tidak lagi menjadi masalah, sehingga kode lebih mudah dibaca dan dipahami.

4. Parameter Variabel

   Parameter ini digunakan untuk menerima jumlah argumen yang tidak ditentukan sebelumnya. Ada dua cara utama menggunakan parameter variabel:

   - *args: Digunakan untuk menerima sejumlah argumen dalam bentuk tuple.
   - **kwargs: Digunakan untuk menerima argumen dalam bentuk pasangan key-value (dictionary).

Penggunaan parameter dan argumen dalam fungsi sangat membantu dalam membangun program yang fleksibel, efisien, dan mudah digunakan kembali. Dengan memahami jenis-jenis parameter

```python
# 1. Parameter Posisional
def info_mahasiswa(nama, jurusan):
    print(f"Nama: {nama}, Jurusan: {jurusan}")

info_mahasiswa("Rina", "Teknik Informatika")  # Urutan harus sesuai

# 2. Parameter Default
def sapa_pengguna(nama="User"):
    print(f"Halo, {nama}!")

sapa_pengguna()  # Menggunakan nilai default
sapa_pengguna("Andi")  # Menggunakan argumen yang diberikan

# 3. Parameter Keyword
def detail_produk(nama, harga, stok):
    print(f"Produk: {nama}, Harga: {harga}, Stok: {stok}")

detail_produk(nama="Laptop", harga=15000000, stok=5)  # Menyebutkan nama parameter
detail_produk(stok=10, harga=50000, nama="Mouse")  # Urutan bisa berbeda

# 4. Parameter Variabel (*args - Tuple)
def total_harga(*harga):
    return sum(harga)

print(f"Total harga: {total_harga(10000, 20000, 15000)}")

# 5. Parameter Variabel (**kwargs - Dictionary)
def info_pelanggan(**data):
    for key, value in data.items():
        print(f"{key}: {value}")

info_pelanggan(nama="Budi", umur=25, kota="Jakarta")

```

ini, kita dapat menulis fungsi yang lebih dinamis dan mampu menangani berbagai skenario penggunaan tanpa harus mendefinisikan fungsi baru setiap kali.

## 6.5 Latihan Soal

1. Jelaskan manfaat utama dari penggunaan fungsi dalam pemrograman.
2. Buat contoh fungsi yang menerima dua parameter dan mengembalikan hasil penjumlahan.
3. Apa perbedaan antara parameter posisional dan parameter default?
4. Bagaimana cara menggunakan fungsi dari file lain dalam Python?
5. Buat program sederhana yang menggunakan fungsi untuk menghitung luas persegi panjang.

# Bab 7: Struktur Data: List dan Tuple

## 7.1 Pengertian List dan Tuple

Dalam pemrograman, struktur data adalah cara penyimpanan dan pengorganisasian data agar dapat digunakan secara efisien. Dalam bahasa pemrograman Python, **List** dan **Tuple** adalah dua jenis struktur data yang sering digunakan untuk menyimpan sekumpulan nilai dalam satu variabel.

**List** adalah struktur data yang bersifat **mutable**, yang berarti elemen di dalamnya dapat diubah setelah list dibuat. Keunggulan list: fleksibilitas dalam menyimpan data dari berbagai tipe (misalnya, integer, string, objek lain). Elemen dalam list bisa ditambahkan, dihapus, atau diperbarui sesuai kebutuhan. Karena sifatnya yang fleksibel, list sering digunakan untuk menyimpan data yang dapat berubah selama program berjalan.

#Contoh sederhana pembuatan list pada bahasa pemrograman python

list1 = ['Fakultas Vokasi USD', 'Teknologi Rekayasa mekatronika', 2024, 2025]

list2 = [1, 2, 3, 4, 5 ]

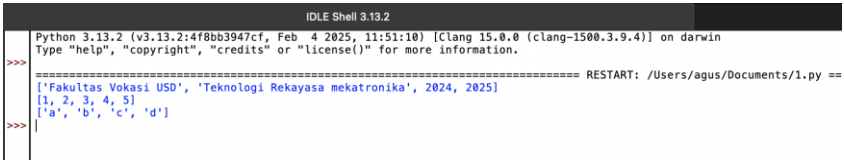
list3 = ["a", "b", "c", "d"]

print(list1)

print(list2)

print(list3)

Hasil Run :




Sebaliknya, **Tuple** adalah struktur data yang bersifat **immutable**, yang berarti elemen di dalamnya **tidak dapat** diubah setelah tuple dibuat dan dapat menyimpan berbagai tipe data. Sifat ini membuat tuple lebih aman digunakan untuk menyimpan data yang bersifat tetap dan tidak perlu dimodifikasi.

#Contoh sederhana pembuatan tuple pada bahasa pemrograman python

tup1 = ('Fakultas Vokasi USD', 'Teknologi Rekayasa mekatronika', 2024, 2025)

tup2 = (1, 2, 3, 4, 5 )

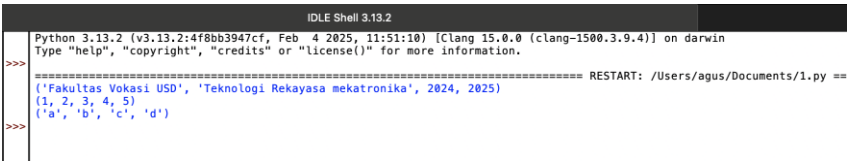
tup3 = "a", "b", "c", "d"

print(tup1)

print(tup2)

print(tup3)

Hasil Run :



Tupel kosong ditulis sebagai dua tanda kurung yang tidak berisi apa-apa, contohnya : tup1 = (); Untuk menulis tupel yang berisi satu nilai, Anda harus memasukkan koma, meskipun hanya ada satu nilai, contohnya : tup1 = (50,) Seperti indeks String, indeks tuple mulai dari 0, dan mereka dapat diiris, digabungkan, dan seterusnya.

Perbedaan utama antara list dan tuple terletak pada sifatnya. List lebih fleksibel karena dapat diubah, sedangkan tuple lebih cepat dan efisien karena bersifat tetap. Pemilihan antara list dan tuple bergantung pada kebutuhan program. Jika data perlu sering dimodifikasi, list lebih cocok digunakan. Namun, jika data harus tetap dan tidak boleh diubah, tuple menjadi pilihan yang lebih aman dan efisien.

Python menyediakan banyak fungsi built-in yang sangat berguna untuk bekerja dengan list. Fungsi-fungsi ini memungkinkan kita untuk melakukan berbagai operasi pada list secara efisien dan dengan sintaks yang sederhana. Berikut adalah penjelasan mendalam tentang beberapa fungsi built-in yang umum digunakan untuk list:

1. Fungsi len()

   Deskripsi: Fungsi len() digunakan untuk menghitung jumlah elemen dalam sebuah list. Contoh : Sintaksnya len(list).

   *numbers = [10, 20, 30, 40, 50]*

   *print(len(numbers))  # Output: 5*

   Fungsi ini akan mengembalikan jumlah elemen dalam list. Dalam contoh di atas, terdapat lima elemen dalam list, sehingga outputnya adalah 5.

2. Fungsi min() dan max().

   Fungsi min() digunakan untuk mencari elemen terkecil dalam list, sedangkan max() digunakan untuk mencari elemen terbesar.Sintaksnya min(list) dan max(list).

   Contoh :

   *numbers = [10, 20, 30, 40, 50]*

   *print(min(numbers))  # Output: 10*

   *print(max(numbers))  # Output: 50*

   Fungsi min() dan max() akan mengembalikan nilai terkecil dan terbesar dalam list, masing-masing. Dalam contoh di atas, 10 adalah nilai terkecil, dan 50 adalah nilai terbesar.

3. Fungsi sum().

   Fungsi sum() digunakan untuk menjumlahkan semua elemen dalam list yang berisi angka. Sintaksnya sum(list).

   Contoh :

   *numbers = [10, 20, 30, 40, 50]*

   *print(sum(numbers))  # Output: 150*

   Fungsi sum() menjumlahkan semua elemen dalam list. Dalam contoh di atas, hasil penjumlahannya adalah 150.

4. Fungsi sorted() dan Reverse()

   Fungsi sorted() digunakan untuk mengurutkan elemen dalam list. Secara default, fungsi ini mengurutkan list dalam urutan menaik (ascending). Fungsi reverse() digunakan untuk membalik urutan elemen dalam list. Sintaksnya sorted(list) dan list.reverse().

   *numbers = [50, 30, 40, 10, 20]*

   *print(sorted(numbers))  # Output: [10, 20, 30, 40, 50]*

   *numbers.reverse()  # Membalik urutan list*

*print(numbers)  # Output: [20, 10, 40, 30, 50]*

Fungsi sorted() mengembalikan list baru yang terurut, tanpa mengubah list asli. Sedangkan reverse() mengubah list secara langsung, membalik urutan elemen dalam list tersebut.

5. Fungsi count() dan index().

   Fungsi count() digunakan untuk menghitung jumlah kemunculan elemen tertentu dalam list. Fungsi index() digunakan untuk mencari indeks pertama dari elemen yang dicari dalam list. Sintaksnya list.count(element) dan list.index(element).

   *fruits = ['apple', 'banana', 'cherry', 'apple', 'apple']*

   *print(fruits.count('apple'))  # Output: 3*

   *print(fruits.index('banana'))  # Output: 1*

   Fungsi count() akan mengembalikan jumlah elemen 'apple' yang ada dalam list, yaitu 3. Sedangkan index() akan mengembalikan indeks pertama di mana 'banana' ditemukan, yaitu 1.

List comprehension adalah fitur Python yang memungkinkan kita membuat list baru dengan cara yang lebih ringkas dan efisien. Ini adalah salah satu fitur Python yang sangat populer karena dapat membuat kode lebih bersih dan mudah dibaca, serta lebih efisien dalam hal kinerja dibandingkan dengan cara tradisional dalam membuat list menggunakan loop. Sintaksnya new_list = [expression for item in iterable if condition]. Misalnya kita ingin membuat list yang berisi kuadrat dari angka-angka yang ada dalam rentang 0 sampai 9. Dengan list comprehension, kita dapat melakukannya dengan cara berikut:

*squares = [x**2 for x in range(10)]*

*print(squares)  # Output: [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]*

Pada contoh ini, x**2 adalah ekspresi yang menghasilkan kuadrat dari setiap angka x dalam rentang 0 hingga 9. Kita juga dapat menambahkan

kondisi untuk memilih elemen mana yang akan dimasukkan ke dalam list. Misalnya, kita ingin membuat list yang berisi kuadrat hanya dari angka genap dalam rentang 0 hingga 9:

```
even_squares = [x**2 for x in range(10) if x % 2 == 0]
print(even_squares)  # Output: [0, 4, 16, 36, 64]
```

Di sini, hanya angka yang genap (yaitu, yang memenuhi kondisi x % 2 == 0) yang diproses dan ditambahkan ke list even_squares. List comprehension tidak hanya terbatas pada operasi sederhana. Kita bisa melakukan operasi yang lebih kompleks, misalnya, memanipulasi string atau menggabungkan beberapa elemen menjadi satu:

```
words = ["hello", "world", "python", "is", "awesome"]
uppercase_words = [word.upper() for word in words if len(word) > 3]
print(uppercase_words)  # Output: ['HELLO', 'WORLD', 'PYTHON', 'AWESOME']
```

Dalam contoh ini, kita mengambil semua kata dengan panjang lebih dari 3 karakter dan mengubahnya menjadi huruf kapital menggunakan upper(). Beberapa manfaat list comprehension:

- Keterbacaan: List comprehension sering kali membuat kode lebih mudah dibaca dan lebih ringkas. Alih-alih menulis beberapa baris kode untuk loop dan kondisi, kita dapat melakukannya dalam satu baris.
- Efisiensi: List comprehension lebih efisien dalam hal kinerja dibandingkan dengan penggunaan loop tradisional. Python mengoptimalkan pembuatan list menggunakan fitur ini.
- Fleksibilitas: List comprehension dapat digunakan untuk berbagai operasi kompleks, termasuk menggabungkan elemen, memfilter data, dan memodifikasi data dalam satu langkah.

Latihan List Comprehension contohnya Buatlah list comprehension untuk mencetak angka yang merupakan hasil perkalian dua angka berturut-turut dalam rentang 1 hingga 5.

*result = [x * (x + 1) for x in range(1, 6)]*

*print(result)  # Output: [2, 6, 12, 20, 30]*

Fungsi built-in untuk list di Python, seperti `len()`, `min()`, `max()`, `sum()`, dan `sorted()`, memberikan kemudahan dan efisiensi dalam memanipulasi dan mengelola data dalam list, memungkinkan kita untuk melakukan berbagai operasi dengan cepat. Sementara itu, list comprehension adalah teknik yang sangat berguna untuk membuat list baru dengan cara yang lebih ringkas dan deklaratif, membuat kode lebih bersih dan meningkatkan efisiensi program. Dengan kedua alat ini, pemrograman Python menjadi lebih efisien dan efektif, memungkinkan pengembang untuk menangani data dengan lebih mudah dan elegan.

## 7.2 Perbedaan List dan Tuple

Dalam pemrograman, terutama dalam bahasa seperti Python, List dan Tuple adalah dua jenis struktur data yang sering digunakan untuk menyimpan kumpulan elemen. Meskipun keduanya serupa dalam beberapa aspek, terdapat perbedaan mendasar yang membuat masing-masing lebih cocok untuk kasus penggunaan tertentu.

Berikut adalah perbedaan utama antara List dan Tuple:

### 7.2.1 Mutability (Dapat Diubah atau Tidak)

- List bersifat mutable, yang berarti elemen di dalamnya dapat diubah setelah deklarasi. Elemen dapat ditambah, dihapus, atau dimodifikasi.

- Tuple bersifat immutable, sehingga tidak bisa diubah setelah dideklarasikan. Semua elemen di dalam Tuple bersifat tetap.

Keuntungan dari sifat immutable pada Tuple adalah membuatnya lebih stabil dan aman untuk data yang tidak boleh berubah.

7.2.2 Kecepatan Akses dan Performa

- Tuple lebih cepat dibandingkan List, terutama dalam operasi pencarian dan iterasi.
- Karena bersifat immutable, Tuple lebih dioptimalkan dalam penggunaan memori dan dapat diakses lebih efisien oleh interpreter.
- List lebih lambat, karena adanya fleksibilitas dalam modifikasi elemen.

Dalam program yang memerlukan kinerja tinggi dan banyak pencarian data, Tuple lebih disarankan dibandingkan List.

7.2.3 Penggunaan Memori

- Tuple menggunakan lebih sedikit memori dibandingkan List, karena tidak perlu menyimpan informasi tambahan untuk perubahan elemen.
- List membutuhkan lebih banyak memori, terutama karena adanya fleksibilitas dalam perubahan ukuran dan elemen.

Hal ini menjadikan Tuple lebih hemat sumber daya, terutama dalam aplikasi dengan jumlah data besar.

7.2.4 Keamanan Data

- Tuple lebih aman karena bersifat immutable, sehingga data di dalamnya tidak bisa diubah secara tidak sengaja.

- List lebih rentan terhadap perubahan, terutama jika digunakan dalam kode yang kompleks dan melibatkan banyak fungsi.

Jika ada data yang tidak boleh berubah, seperti koordinat geografis atau konfigurasi sistem, Tuple lebih disarankan untuk menghindari modifikasi yang tidak disengaja.

7.2.5 Penggunaan dalam Konteks yang Berbeda

- List cocok digunakan jika data sering berubah, seperti daftar item dalam keranjang belanja atau daftar tugas yang bisa diperbarui.
- Tuple lebih cocok digunakan untuk data tetap, seperti hari dalam seminggu, konstanta matematika, atau koordinat lokasi.

## 7.3 Cara Membuat List dan Tuple

Dalam pemrograman, List dan Tuple adalah dua struktur data yang digunakan untuk menyimpan sekumpulan nilai dalam satu variabel. Meskipun memiliki fungsi yang serupa, terdapat perbedaan utama dalam fleksibilitas dan penggunaannya.

7.3.1 List: Struktur Data yang Dinamis

List adalah struktur data yang bersifat dinamis, yang berarti elemen-elemen di dalamnya dapat diubah setelah deklarasi. List memungkinkan penambahan, penghapusan, atau pengubahan elemen sesuai kebutuhan program.

List ditulis menggunakan tanda kurung siku [] dan dapat berisi berbagai jenis data, termasuk angka, string, atau bahkan list lainnya. Salah satu keunggulan utama list adalah kemampuannya untuk diubah, sehingga cocok untuk menyimpan kumpulan data yang bersifat variatif atau sering mengalami pembaruan.

Selain itu, list mendukung berbagai operasi seperti menambahkan elemen dengan fungsi append(), menghapus elemen dengan remove() atau pop(), serta mengurutkan elemen dengan sort(). Karena fleksibilitasnya, list sering digunakan dalam pengolahan data, pembuatan daftar tugas, atau sebagai struktur data untuk algoritma yang memerlukan perubahan nilai secara dinamis.

### 7.3.2 Tuple: Struktur Data yang Statis

Tuple adalah struktur data yang bersifat statis, yang berarti elemen-elemen di dalamnya tidak dapat diubah setelah deklarasi. Tuple digunakan ketika kita ingin menyimpan sekumpulan data yang tidak perlu mengalami perubahan, sehingga lebih aman dari modifikasi yang tidak disengaja.

Tuple ditulis menggunakan tanda kurung () dan memiliki sifat immutable, yang berarti tidak mendukung operasi seperti penambahan atau penghapusan elemen setelah deklarasi. Karena sifatnya yang tetap, tuple memiliki keunggulan dalam efisiensi memori dan kecepatan eksekusi dibandingkan list.

Tuple sering digunakan untuk menyimpan data yang bersifat konstan, seperti koordinat dalam pemrograman grafis, nilai konfigurasi yang tidak boleh berubah, atau kumpulan data yang tidak memerlukan manipulasi lebih lanjut.

### 7.3.3 Perbandingan Kinerja antara List dan Tuple

Selain perbedaan dalam mutabilitasnya, list dan tuple juga memiliki perbedaan dalam hal kinerja. Tuple lebih cepat dalam operasi baca dibandingkan list karena sifatnya yang tetap. Ini disebabkan oleh alokasi memori yang lebih efisien dan pengelolaan referensi data yang lebih sederhana.

Namun, list lebih fleksibel karena memungkinkan modifikasi data, menjadikannya pilihan yang lebih baik untuk struktur data yang dinamis. Oleh karena itu, pemilihan antara list dan tuple bergantung

pada kebutuhan program—apakah data perlu sering diperbarui atau tetap konstan sepanjang eksekusi program.

```python
import timeit

# === List: Struktur Data yang Dinamis ===
buah = ["Apel", "Mangga", "Pisang", "Jeruk"]

# Menambahkan elemen ke dalam list
buah.append("Semangka")
print("Setelah append:", buah)

# Menghapus elemen dari list
buah.remove("Pisang")
print("Setelah remove:", buah)

# Mengubah elemen dalam list
buah[1] = "Durian"
print("Setelah perubahan:", buah)

# Mengurutkan list
buah.sort()
print("Setelah sort:", buah)


# === Tuple: Struktur Data yang Statis ===
koordinat = (10, 20)
print("Koordinat X:", koordinat[0])
print("Koordinat Y:", koordinat[1])

data_mahasiswa = ("Budi", 21, "Teknik Informatika")
print("Nama:", data_mahasiswa[0])
print("Umur:", data_mahasiswa[1])
print("Jurusan:", data_mahasiswa[2])

# === Perbandingan Kinerja List vs Tuple ===
list_test = [i for i in range(1000000)]
tuple_test = tuple(list_test)

waktu_list = timeit.timeit(lambda: list_test[500000], number=100000)
waktu_tuple = timeit.timeit(lambda: tuple_test[500000], number=100000)

print(f"Akses elemen List: {waktu_list:.6f} detik")
print(f"Akses elemen Tuple: {waktu_tuple:.6f} detik")

```

## 7.4 Operasi pada List dan Tuple

List dan Tuple adalah dua tipe data yang sering digunakan untuk menyimpan kumpulan elemen dalam Python. Meskipun keduanya memiliki kesamaan dalam cara menyimpan data, List bersifat mutable (dapat diubah), sedangkan Tuple bersifat immutable (tidak dapat diubah setelah didefinisikan).

Berbagai operasi yang dapat dilakukan pada List dan Tuple meliputi:

1. Mengakses Elemen

   Elemen dalam List atau Tuple dapat diakses menggunakan indeks. Indeks dimulai dari 0 untuk elemen pertama, 1 untuk elemen kedua, dan seterusnya.

   #Cara mengakses nilai di dalam list Python

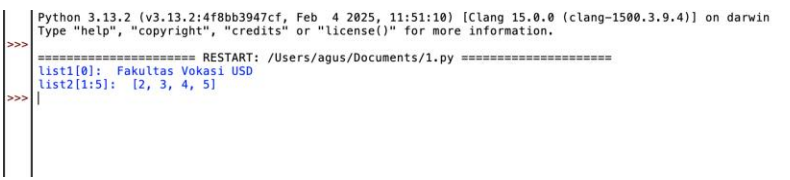
   list1 = ['Fakultas Vokasi USD', 'Teknologi Rekayasa Mekatronika', 2024, 2025]

   list2 = [1, 2, 3, 4, 5, 6, 7 ]

   print ("list1[0]: ", list1[0])

   print ("list2[1:5]: ", list2[1:5])

   Hasil Run :




   #Cara mengakses nilai tuple

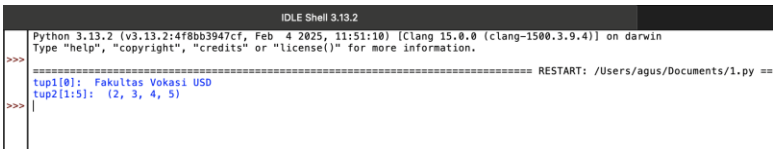
   tup1 = ('Fakultas Vokasi USD', 'Teknologi Rekayasa Mekatronika', 2024, 2025)

tup2 = (1, 2, 3, 4, 5, 6, 7 )

print ("tup1[0]: ", tup1[0])

print ("tup2[1:5]: ", tup2[1:5])

Hasil Run :




2. Menambahkan Elemen

Pada List, elemen baru dapat ditambahkan menggunakan metode append() untuk menambahkan di akhir, atau insert() untuk menambahkan pada posisi tertentu. Namun, pada Tuple, elemen tidak bisa ditambahkan karena sifatnya yang tetap. Contoh 1:

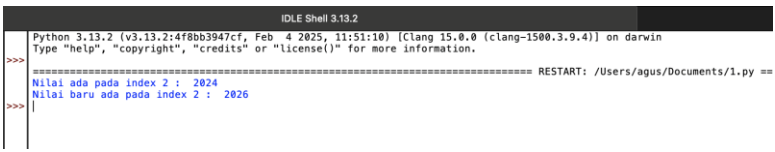
list = ['Fakultas Vokasi USD', 'Teknologi Rekayasa Mekatronika', 2024, 2025]

print ("Nilai ada pada index 2 : ", list[2])

list[2] = 2026

print ("Nilai baru ada pada index 2 : ", list[2])

Hasil run :




Contoh 2 :

Menggunakan append(), insert(), dan extend().

my_list = [1, 2, 3]

my_list.append(4) # Menambah elemen 4 di akhir list

```
my_list.insert(1, 5)  # Menyisipkan elemen 5 pada posisi index 1
my_list.extend([6, 7])  # Menambah beberapa elemen sekaligus
print(my_list)  # Output: [1, 5, 2, 3, 6, 7, 4]
```

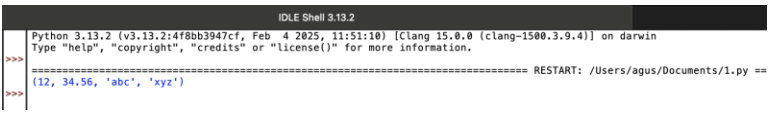
Tuple tidak berubah, yang berarti Anda tidak dapat memperbarui atau mengubah nilai

elemen tupel. Anda dapat mengambil bagian dari tupel yang ada untuk membuat

tupel baru seperti ditunjukkan oleh contoh berikut.

```
tup1 = (12, 34.56)
tup2 = ('abc', 'xyz')
# Aksi seperti dibawah ini tidak bisa dilakukan pada tuple python
# Karena memang nilai pada tuple python tidak bisa diubah
# tup1[0] = 100;
# Jadi, buatlah tuple baru sebagai berikut
tup3 = tup1 + tup2
print (tup3)
```

Hasil run :




3. Menghapus Elemen

Pada List, elemen dapat dihapus menggunakan metode remove() untuk menghapus elemen berdasarkan nilai, atau pop() untuk menghapus elemen berdasarkan indeks.

Sementara itu, Tuple tidak mendukung penghapusan elemen karena tidak bisa diubah setelah dibuat.

*#Contoh 1 cara menghapus nilai pada list python*

list = ['Fakultas Vokasi USD', 'Teknologi Rekayasa Mekatronika', 2024, 2025]

print (list)

**del** list[2]

print ("Setelah dihapus nilai pada index 2 : ", list)

Hasil run :


*#Contoh 2 cara menghapus nilai pada list python*

my_list = [1, 2, 3, 4]

my_list.remove(2) # Menghapus elemen pertama yang bernilai 2

my_list.pop(1) # Menghapus elemen di index 1

my_list.clear() # Menghapus semua elemen

print(my_list) # Output: []

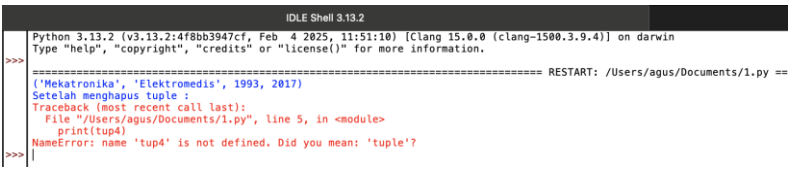
Menghapus elemen tuple individual tidak mungkin dilakukan. Tentu saja, tidak ada yang salah dengan menggabungkan tuple lain dengan unsur-unsur yang tidak diinginkan dibuang. Untuk secara eksplisit menghapus keseluruhan tuple, cukup gunakan del statement.

Sebagai contohContoh menghapus tuple,

tup4= ('Mekatronika', 'Elektromedis', 1993, 2017)

```
print(tup4)
del tup4
print("Setelah menghapus tuple : ")
print(tup4)
```

Hasil run:




4. Menggabungkan List dan Tuple

   Kedua struktur ini dapat digabungkan menggunakan operator +, yang akan menghasilkan List atau Tuple baru dengan elemen dari kedua struktur tersebut.

5. Mengetahui Panjang List atau Tuple

Fungsi len () dapat digunakan untuk menghitung jumlah elemen yang terdapat dalam List atau Tuple.

```python
# === 1. Mengakses Elemen dalam List dan Tuple ===
list_buah = ["Apel", "Mangga", "Pisang", "Jeruk"]
tuple_buah = ("Apel", "Mangga", "Pisang", "Jeruk")

print("Elemen pertama List:", list_buah[0])  # Apel
print("Elemen kedua Tuple:", tuple_buah[1])  # Mangga

# === 2. Menambahkan Elemen (Hanya untuk List) ===
list_buah.append("Semangka")  # Menambahkan di akhir
list_buah.insert(2, "Durian")  # Menambahkan di indeks tertentu
print("Setelah menambahkan elemen:", list_buah)

# === 3. Menghapus Elemen (Hanya untuk List) ===
list_buah.remove("Pisang")  # Menghapus berdasarkan nilai
list_buah.pop(1)  # Menghapus berdasarkan indeks
print("Setelah menghapus elemen:", list_buah)

# === 4. Menggabungkan List dan Tuple ===
list_sayur = ["Bayam", "Wortel"]
tuple_sayur = ("Tomat", "Kentang")

gabungan_list = list_buah + list_sayur
gabungan_tuple = tuple_buah + tuple_sayur

print("Gabungan List:", gabungan_list)
print("Gabungan Tuple:", gabungan_tuple)

# === 5. Mengetahui Panjang List atau Tuple ===
print("Panjang List Buah:", len(list_buah))
print("Panjang Tuple Buah:", len(tuple_buah))

```

Penggunaan List dan Tuple dalam Program Nyata :

1. Menyimpan Data Siswa

   Menggunakan tuple untuk menyimpan data yang tidak berubah (misalnya, nama, ID siswa) dan list untuk menyimpan nilai yang dapat berubah.

```
siswa1 = ("Ali", 101, [85, 90, 92])
siswa2 = ("Budi", 102, [78, 88, 91])
# Menambahkan nilai baru untuk siswa 1
siswa1[2].append(95)  # Menambah nilai 95 pada list nilai
```

2. Menyimpan Koordinat

   Menggunakan tuple untuk menyimpan koordinat lokasi yang tidak berubah.

```
koordinat_titik = (25.774, -80.190)
print("Koordinat titik:", koordinat_titik)
```

3. Menggabungkan List dan Tuple

   Menggunakan list untuk menyimpan beberapa tuple.

```
daftar_koordinat = [(25.774, -80.190), (30.267, -97.743), (34.052, -118.243)]
for koordinat in daftar_koordinat:
print(koordinat)
```

## 7.5 Latihan Soal

1. Jelaskan perbedaan utama antara List dan Tuple.
2. Bagaimana cara menambahkan elemen ke dalam List? Mengapa tidak bisa dilakukan pada Tuple?
3. Buat contoh List yang berisi nama-nama kota dan lakukan iterasi untuk mencetak setiap kota.
4. Tuliskan cara menghapus elemen dari List menggunakan fungsi `remove()` dan `pop()`.

5. Jelaskan keunggulan Tuple dibandingkan List dalam aspek kecepatan dan keamanan.
6. Buatlah tuple yang berisi nama, umur, dan alamat seorang individu, dan gunakan tuple tersebut untuk mengganti atau mengakses elemen tertentu.
7. Gunakan list comprehension untuk membuat list yang berisi kuadrat dari angka-angka genap antara 1 hingga 20.
8. Implementasikan sebuah aplikasi yang menerima masukan dari pengguna berupa list angka dan tuple nama serta umur. Kemudian, program ini harus mencetak daftar siswa dengan umur lebih dari 18 tahun.

# Bab 8: Dictionary dan Set

## 8.1 Pengertian Dictionary dan Set

Dalam pemrograman Python, **Dictionary** dan **Set** adalah dua struktur data yang digunakan untuk menyimpan kumpulan data dengan cara yang efisien dan fleksibel. Meskipun keduanya digunakan untuk menyimpan beberapa nilai, keduanya memiliki perbedaan dalam cara menyusun dan mengakses data.

**Dictionary** adalah struktur data yang menyimpan pasangan **kunci-nilai**, di mana setiap kunci harus unik dan digunakan untuk mengakses nilai yang terkait. Dengan sistem ini, pencarian data menjadi lebih cepat karena nilai dapat diakses langsung melalui kunci tanpa harus mencari satu per satu. Dictionary sering digunakan untuk menyimpan informasi yang berpasangan, seperti data pengguna dengan ID, daftar harga barang, atau konfigurasi sistem. Contoh sintaksnya my_dict = {'key1': 'value1', 'key2': 'value2', 'key3': 'value3'}. Contoh penerapannya:

```
student = {'name': 'Alice', 'age': 21, 'grade': 'A'}
print(student)
# Output: {'name': 'Alice', 'age': 21, 'grade': 'A'}
```

Sementara itu, **Set** adalah struktur data yang menyimpan kumpulan **elemen unik** yang tidak berurutan dan tidak memiliki indeks. Set berguna ketika hanya diperlukan sekumpulan elemen yang tidak boleh duplikat, seperti daftar anggota dalam sebuah grup

atau kumpulan kata unik dalam sebuah teks. Karena tidak memiliki indeks, set tidak mendukung akses langsung ke elemen seperti list atau dictionary, tetapi memiliki performa yang lebih cepat dalam operasi seperti pencarian atau pengecekan keberadaan elemen dalam koleksi. Contoh sintaksnya my_set = {1, 2, 3, 4, 5}.

```
my_set = {1, 2, 3, 4, 5}
print(my_set)
# Output: {1, 2, 3, 4, 5}
```

Perbedaan utama antara dictionary dan set terletak pada cara penyimpanan dan akses data. Dictionary menyimpan data dalam bentuk pasangan kunci-nilai yang memungkinkan pencarian cepat, sedangkan set hanya menyimpan elemen unik tanpa urutan tertentu. Pemilihan antara keduanya bergantung pada kebutuhan program. Jika diperlukan pencocokan data berdasarkan kunci tertentu, dictionary lebih cocok digunakan. Namun, jika hanya perlu memastikan kumpulan elemen unik tanpa memperhatikan urutannya, set menjadi pilihan yang lebih efisien.

## 8.2 Perbedaan Dictionary dan Set

Dalam pemrograman, terutama dalam bahasa seperti Python, Dictionary dan Set adalah dua jenis struktur data yang sering digunakan untuk menyimpan dan mengelola data. Meskipun keduanya memiliki kemiripan dalam penggunaan elemen unik, terdapat perbedaan mendasar yang membuat masing-masing lebih cocok untuk kasus tertentu.

Berikut adalah perbedaan utama antara Dictionary dan Set:

8.2.1 Struktur Data

- Dictionary menyimpan data dalam bentuk pasangan kunci-nilai, di mana setiap kunci unik memiliki nilai yang terkait.
- Set hanya menyimpan kumpulan elemen unik tanpa pasangan nilai.

Dictionary lebih cocok untuk menyimpan data yang memiliki hubungan antara kunci dan nilai, sedangkan Set lebih digunakan untuk mengelola data unik tanpa keterkaitan khusus.

8.2.2 Akses dan Penyimpanan Data

- Dictionary memungkinkan akses data dengan cepat melalui kunci, sehingga pencarian nilai menjadi lebih efisien.
- Set tidak memiliki indeks atau kunci, sehingga elemen hanya bisa diakses melalui iterasi.

Jika membutuhkan akses cepat ke data berdasarkan identifikasi tertentu, Dictionary lebih disarankan. Namun, jika hanya perlu menyimpan kumpulan elemen unik dan tidak memerlukan akses langsung, Set adalah pilihan yang lebih baik.

8.2.3 Unik atau Tidaknya Elemen

- Dictionary dapat memiliki nilai duplikat, tetapi kunci harus unik.
- Set tidak mengizinkan duplikasi elemen, setiap elemen dalam Set harus unik.

Jika ingin memastikan tidak ada elemen yang duplikat, Set lebih sesuai. Sementara itu, jika perlu menyimpan data yang

memiliki hubungan unik antara kunci dan nilai, Dictionary lebih cocok.

8.2.4 Penggunaan dalam Pemrograman

- Dictionary biasanya digunakan untuk menyimpan data dengan struktur yang memiliki keterkaitan, seperti informasi pengguna (nama, usia, alamat) atau pengaturan konfigurasi dalam program.
- Set lebih sering digunakan untuk mengelola data unik, seperti menghapus duplikasi dari daftar atau mengecek keberadaan suatu elemen dalam sekumpulan data.

8.2.5 Kecepatan dan Efisiensi

- Dictionary lebih efisien untuk pencarian data berdasarkan kunci.
- Set lebih efisien untuk operasi himpunan seperti union, intersection, dan difference.

Jika membutuhkan pencarian cepat berdasarkan kunci tertentu, Dictionary lebih optimal. Namun, jika sering menggunakan operasi matematika pada kumpulan data, Set lebih disarankan.

## 8.3 Cara Membuat Dictionary dan Set

```python
# === 1. Membuat Dictionary ===
data_pengguna = {
    "nama": "Kevin",
    "usia": 25,
    "pekerjaan": "Desainer",
    "hobi": ["Membaca", "Menggambar", "Olahraga"]
}

# Mengakses nilai dalam dictionary
print("Nama:", data_pengguna["nama"])  # Mengakses nilai berdasarkan kunci
print("Hobi:", data_pengguna["hobi"])

# Menambahkan dan mengubah elemen dalam dictionary
data_pengguna["kota"] = "Jakarta"  # Menambahkan kunci baru
data_pengguna["usia"] = 26  # Mengubah nilai pada kunci yang sudah ada
print("Data setelah diupdate:", data_pengguna)

# Menghapus elemen dalam dictionary
del data_pengguna["pekerjaan"]
print("Data setelah penghapusan:", data_pengguna)

# Mendapatkan daftar kunci dan nilai dalam dictionary
print("Kunci dalam dictionary:", list(data_pengguna.keys()))
print("Nilai dalam dictionary:", list(data_pengguna.values()))

# === 2. Membuat Set ===
set_angka = {1, 2, 3, 4, 5}
set_huruf = set(["a", "b", "c", "d", "e"])

# Menambahkan elemen ke dalam set
set_angka.add(6)
print("Set setelah menambahkan elemen:", set_angka)

# Menghapus elemen dalam set
set_angka.remove(3)
print("Set setelah menghapus elemen:", set_angka)

# Operasi matematika pada set
set_a = {1, 2, 3, 4}
set_b = {3, 4, 5, 6}

print("Union (gabungan):", set_a | set_b)  # Menggabungkan dua set
print("Intersection (irisan):", set_a & set_b)  # Mencari elemen yang sama
print("Difference (selisih):", set_a - set_b)  # Mencari elemen unik di set_a

```

Dalam pemrograman, Dictionary dan Set adalah dua struktur data yang memiliki karakteristik dan kegunaan yang berbeda. Dictionary digunakan untuk menyimpan data dalam bentuk pasangan kunci-nilai, sementara Set digunakan untuk menyimpan kumpulan elemen unik tanpa urutan tertentu.

### 8.3.1 Dictionary: Struktur Data dengan Kunci dan Nilai

Dictionary adalah struktur data yang memungkinkan penyimpanan data dalam format key-value (kunci-nilai). Setiap nilai dalam dictionary dapat diakses menggunakan kuncinya, bukan berdasarkan indeks seperti pada list atau tuple.

Dictionary ditulis menggunakan tanda kurung kurawal {} dengan setiap elemen berupa pasangan kunci: nilai. Kunci dalam dictionary harus bersifat unik dan umumnya berupa string atau angka, sedangkan nilainya bisa berupa tipe data apa pun, termasuk list atau dictionary lain.

Salah satu keunggulan dictionary adalah kemampuannya untuk menyimpan data secara terstruktur dan memungkinkan akses cepat ke elemen tertentu. Dictionary sering digunakan untuk menyimpan informasi dalam bentuk data objek, seperti data pengguna, konfigurasi sistem, atau hasil pemrosesan data.

Selain itu, dictionary mendukung berbagai operasi seperti menambahkan elemen baru, menghapus elemen dengan del, atau mendapatkan daftar kunci dan nilai menggunakan metode keys() dan values(). Karena fleksibilitasnya, dictionary sangat berguna dalam pemrograman berbasis data, pemetaan nilai, atau pembuatan struktur data kompleks.

### 8.3.2 Set: Kumpulan Elemen Unik tanpa Urutan

Set adalah struktur data yang digunakan untuk menyimpan kumpulan elemen unik tanpa memperhatikan urutan. Tidak seperti list atau tuple, set tidak mengizinkan elemen duplikat, sehingga berguna dalam kasus di mana keunikan data harus dipertahankan.

Set dapat dibuat menggunakan tanda kurung kurawal {} atau fungsi set(). Karena tidak memiliki indeks, elemen dalam set tidak dapat diakses secara langsung seperti dalam list atau tuple. Namun, set mendukung operasi matematika seperti union, intersection, dan difference, yang membuatnya berguna dalam analisis data atau pemrosesan himpunan.

Salah satu keunggulan utama set adalah kinerjanya yang cepat dalam pencarian elemen, karena menggunakan struktur data berbasis hash. Set sering digunakan dalam pemrosesan data unik, seperti menghapus duplikasi dalam kumpulan data atau memeriksa keberadaan elemen tertentu dengan cepat.

8.3.3 Perbandingan antara Dictionary dan Set

Dictionary dan Set memiliki kesamaan dalam penggunaan tanda {} dan sifatnya yang berbasis hash, tetapi keduanya memiliki tujuan yang berbeda. Dictionary digunakan untuk menyimpan data yang memiliki hubungan kunci-nilai, sementara Set digunakan untuk menyimpan sekumpulan nilai unik tanpa urutan tertentu.

Dari segi kinerja, set lebih cepat dalam pencarian elemen dibandingkan list, karena tidak menggunakan indeks tetapi langsung melakukan hashing pada elemen. Namun, dictionary lebih fleksibel karena memungkinkan penyimpanan dan pengelolaan data yang lebih kompleks dengan pasangan kunci-nilai.

Pemilihan antara dictionary dan set bergantung pada kebutuhan program. Jika membutuhkan struktur data untuk memetakan informasi dengan jelas, dictionary adalah pilihan yang

tepat. Jika hanya memerlukan kumpulan elemen unik tanpa urutan tertentu, set lebih efisien dan sederhana untuk digunakan.

## 8.4 Operasi pada Dictionary dan Set

Dictionary adalah struktur data dalam Python yang menyimpan data dalam bentuk pasangan key-value (kunci-nilai). Setiap kunci dalam dictionary bersifat unik, dan nilainya dapat berupa berbagai tipe data, termasuk string, angka, atau bahkan list atau dictionary lain. Dictionary memungkinkan kita untuk mengakses nilai dengan sangat cepat menggunakan kunci yang terkait. Beberapa operasi umum pada dictionary antara lain menambah elemen baru, menghapus elemen berdasarkan kunci, serta mengakses nilai dengan kunci tertentu. Fungsi-fungsi built-in seperti `get()`, `pop()`, dan `update()` memungkinkan kita untuk memanipulasi data dalam dictionary dengan mudah, membuatnya menjadi struktur data yang sangat fleksibel dan efisien.

Sementara itu, Set adalah struktur data yang menyimpan elemen-elemen unik tanpa urutan tertentu dan tanpa menggunakan indeks. Set tidak mengizinkan adanya elemen duplikat, yang menjadikannya sangat berguna untuk menyaring data yang memiliki elemen-elemen yang sama. Operasi pada set meliputi penambahan elemen dengan metode `add()`, penghapusan elemen dengan `remove()` atau `discard()`, serta operasi himpunan seperti union, intersection, dan difference yang memungkinkan kita untuk bekerja dengan koleksi data secara matematis. Karena tidak memiliki urutan,

set lebih efisien dalam pencarian elemen, dan sangat berguna dalam kasus-kasus yang membutuhkan pengelolaan data unik dan operasi set teori.

```python
# === Operasi pada Dictionary ===
data_pengguna = {
    "nama": "Kevin",
    "usia": 25,
    "pekerjaan": "Desainer"
}

# 1. Mengakses Elemen Dictionary
print("Nama:", data_pengguna["nama"])  # Mengakses elemen berdasarkan kunci

# 2. Menambahkan atau Memperbarui Elemen
data_pengguna["kota"] = "Jakarta"  # Menambahkan kunci baru
data_pengguna["usia"] = 26  # Memperbarui nilai yang sudah ada
print("Data setelah update:", data_pengguna)

# 3. Menghapus Elemen
del data_pengguna["pekerjaan"]  # Menghapus menggunakan del
usia_dihapus = data_pengguna.pop("usia")  # Menghapus dengan pop() dan mengembalikan nilai
print("Data setelah penghapusan:", data_pengguna)
print("Usia yang dihapus:", usia_dihapus)

# 4. Pengecekan Keanggotaan
print("Apakah 'nama' ada dalam dictionary?", "nama" in data_pengguna)


# === Operasi pada Set ===
himpunan = {1, 2, 3, 4, 5}

# 1. Mengakses Elemen (Menggunakan perulangan)
print("Isi set:")
for elemen in himpunan:
    print(elemen, end=" ")
print()

# 2. Menambahkan Elemen
himpunan.add(6)  # Menambahkan satu elemen
himpunan.update([7, 8, 9])  # Menambahkan beberapa elemen sekaligus
print("Set setelah penambahan:", himpunan)

# 3. Menghapus Elemen
himpunan.remove(3)  # Menghapus elemen (akan error jika elemen tidak ada)
himpunan.discard(10)  # Menghapus elemen tanpa error jika elemen tidak ditemukan
elemen_dihapus = himpunan.pop()  # Menghapus elemen secara acak
print("Set setelah penghapusan:", himpunan)
print("Elemen yang dihapus:", elemen_dihapus)

# 4. Pengecekan Keanggotaan
print("Apakah 5 ada dalam set?", 5 in himpunan)

```

Operasi pada Dictionary

1. Mengakses Elemen

Elemen dalam Dictionary diakses menggunakan kunci (*key*), bukan indeks seperti pada List atau Tuple. Contohnya Mengakses Data: Data dalam dictionary diakses dengan menggunakan key.

*student = {'name': 'Alice', 'age': 21, 'grade': 'A'}*

*print(student['name'])  # Output: Alice*

2. Menambahkan Elemen

   Elemen baru dapat ditambahkan dengan menetapkan nilai pada kunci yang baru. Jika kunci sudah ada, nilainya akan diperbarui. Menambah atau Memodifikasi Data: Menambah pasangan key-value baru atau memodifikasi nilai dari key yang sudah ada.

   *student['age'] = 22  # Memodifikasi nilai*

   *student['city'] = 'New York'  # Menambahkan pasangan key-value*

   *print(student)*

   *# Output: {'name': 'Alice', 'age': 22, 'grade': 'A', 'city': 'New York'}*

3. Menghapus Elemen

   Penghapusan dapat dilakukan dengan del untuk menghapus berdasarkan kunci, atau dengan metode pop() yang juga mengembalikan nilai yang dihapus. Contoh

   Menghapus pasangan key-value dari dictionary menggunakan del.

   *del student['age']*

   *print(student) # Output: {'name': 'Alice'}*

Menghapus dan mengembalikan nilai dari key tertentu menggunakan pop.

*name = student.pop('name')*

*print(name) # Output: Alice*

*print(student) # Output: {}*

Menghapus semua pasangan key-value dalam dictionary menggunakan clear.

*student.clear()*

*print(student) # Output: {}*

4. Pengecekan Keanggotaan

   Operator in digunakan untuk mengecek apakah suatu kunci terdapat dalam Dictionary.

Dalam Python, dictionary menyediakan berbagai fungsi dan metode built-in yang sangat berguna untuk memanipulasi data. Fungsi dan metode ini memungkinkan kita untuk mengakses, memodifikasi, dan mengelola pasangan key-value dengan cara yang efisien. Berikut adalah penjelasan rinci tentang beberapa metode built-in yang umum digunakan dengan dictionary:

- Fungsi len() digunakan untuk menghitung jumlah pasangan key-value yang ada dalam dictionary. Contonya :

  *student = {'name': 'Alice', 'age': 21, 'grade': 'A'}*

  *print(len(student)) # Output: 3*

  Fungsi ini mengembalikan jumlah total pasangan kunci-nilai dalam dictionary. Dalam contoh ini, dictionary student memiliki tiga pasangan, sehingga hasilnya adalah 3.

- Metode keys () mengembalikan semua key dalam dictionary. Ini mengembalikan objek dict_keys yang bisa diiterasi. Contohnya:

  *student = {'name': 'Alice', 'age': 21, 'grade': 'A'}*

  *print(student.keys()) # Output: dict_keys(['name', 'age', 'grade'])*

  keys () digunakan untuk mendapatkan daftar semua kunci dalam dictionary. Kunci-kunci ini bisa digunakan untuk iterasi atau untuk mengambil nilai yang terkait.

- Metode values () mengembalikan semua value dalam dictionary. Ini juga mengembalikan objek dict_values yang bisa diiterasi.

  *student = {'name': 'Alice', 'age': 21, 'grade': 'A'}*

  *print(student.values()) # Output: dict_values(['Alice', 21, 'A'])*

  Values () memberikan kita akses langsung ke semua nilai dalam dictionary yang dapat digunakan dalam berbagai operasi.

- Metode items () mengembalikan daftar pasangan key-value dalam bentuk dict_items, yang dapat diiterasi. Ini sangat berguna ketika kita ingin mengakses baik kunci maupun nilai secara bersamaan.

  *student = {'name': 'Alice', 'age': 21, 'grade': 'A'}*

  *print(student.items()) # Output: dict_items([('name', 'Alice'), ('age', 21), ('grade', 'A')])*

Dengan menggunakan items (), kita dapat mengakses baik key maupun value dalam dictionary sekaligus, seperti yang terlihat pada contoh di atas.

- Metode get () digunakan untuk mengakses nilai yang terkait dengan key tertentu dalam dictionary. Jika key tidak ada, metode ini akan mengembalikan None (atau nilai default yang dapat diberikan).

  *student = {'name': 'Alice', 'age': 21, 'grade': 'A'}*

  *print(student.get('name')) # Output: Alice*

  *print(student.get('city', 'Unknown')) # Output: Unknown*

  Dengan get(), kita bisa menangani kasus di mana kunci yang dicari tidak ada dalam dictionary tanpa menyebabkan error, serta memberikan nilai default jika diperlukan.

- Metode pop() digunakan untuk menghapus dan mengembalikan nilai yang terkait dengan key tertentu. Jika key tidak ada, maka akan menghasilkan error KeyError, kecuali jika Anda memberikan nilai default.

  *student = {'name': 'Alice', 'age': 21, 'grade': 'A'}*

  *grade = student.pop('grade')*

  *print(grade) # Output: A*

  *print(student) # Output: {'name': 'Alice', 'age': 21}*

  Metode pop() memungkinkan kita untuk menghapus elemen tertentu berdasarkan kunci dan mendapatkan nilainya. Jika key tidak ditemukan, kita bisa memberikan nilai default agar tidak terjadi error.

- Metode popitem() digunakan untuk menghapus dan mengembalikan pasangan key-value terakhir yang ada dalam dictionary. Ini sangat berguna saat kita ingin menghapus elemen terakhir dari dictionary.

  *student = {'name': 'Alice', 'age': 21, 'grade': 'A'}*

  *item = student.popitem()*

  *print(item)  # Output: ('grade', 'A')*

  *print(student)  # Output: {'name': 'Alice', 'age': 21}*

  Dengan popitem(), kita dapat menghapus dan mendapatkan pasangan key-value terakhir dalam dictionary.
- Metode clear() digunakan untuk menghapus semua elemen dalam dictionary, menjadikannya kosong.

  *student = {'name': 'Alice', 'age': 21, 'grade': 'A'}*

  *student.clear()*

  *print(student)  # Output: {}*

  clear() menghapus seluruh isi dictionary. Ini berguna ketika Anda ingin mengosongkan dictionary namun tetap mempertahankan variabel yang sama.
- Metode update() digunakan untuk memperbarui dictionary dengan pasangan key-value baru dari dictionary lain, atau dari iterable yang berisi pasangan key-value.

  *student = {'name': 'Alice', 'age': 21}*

  *student.update({'grade': 'A', 'city': 'New York'})*

  *print(student)  # Output: {'name': 'Alice', 'age': 21, 'grade': 'A', 'city': 'New York'}*

update() memungkinkan Anda untuk menambahkan atau memperbarui beberapa pasangan key-value sekaligus dalam dictionary.

- Metode setdefault() mengembalikan nilai untuk key yang ada, dan jika key tidak ada, maka akan menambah pasangan key-value baru dengan nilai default yang diberikan.

  *student = {'name': 'Alice', 'age': 21}*

  *print(student.setdefault('age', 25)) # Output: 21 (karena 'age' sudah ada)*

  *print(student.setdefault('grade', 'A')) # Output: A (karena 'grade' belum ada, jadi ditambahkan)*

  *print(student) # Output: {'name': 'Alice', 'age': 21, 'grade': 'A'}*

  Metode setdefault() mengembalikan nilai untuk key yang ada, dan jika key tidak ada, maka akan menambah pasangan key-value baru dengan nilai default yang diberikan.

Operasi pada Set

1. Mengakses Elemen

   Karena Set tidak memiliki indeks atau kunci, elemen diakses melalui iterasi menggunakan perulangan.

2. Menambahkan Elemen

   Elemen baru dapat ditambahkan menggunakan metode add() untuk satu elemen atau update() untuk menambahkan beberapa elemen sekaligus.

   Contoh menggunakan add() untuk menambah satu elemen.

   *my_set = {1, 2, 3}*

*my_set.add(4)*

*print(my_set)  # Output: {1, 2, 3, 4}*

3. Menghapus Elemen

   Elemen dapat dihapus dengan remove() jika dipastikan ada dalam Set, atau discard() untuk menghapus tanpa menimbulkan error jika elemen tidak ditemukan. Metode pop() juga bisa digunakan untuk menghapus elemen secara acak. Menggunakan remove() atau discard() untuk menghapus elemen.

   *my_set.remove(3)*

   *print(my_set)  # Output: {1, 2, 4}*

   *my_set.discard(5)  # Tidak ada error meskipun 5 tidak ada*

   Atau contoh Menghapus semua elemen dalam set.

   *my_set.clear()*

   *print(my_set)  # Output: set()*

4. Pengecekan Keanggotaan

   Operator in digunakan untuk mengecek apakah suatu elemen terdapat dalam Set. Contohnya:

5. Operasi Matematika pada Set

- Union menggabungkan dua set. Contohnya:

  *set1 = {1, 2, 3}*

  *set2 = {3, 4, 5}*

  *print(set1.union(set2))  # Output: {1, 2, 3, 4, 5}*

- Intersection menemukan elemen yang ada pada kedua set. Contohnya:

  *print(set1.intersection(set2))  # Output: {3}*

- Difference menemukan elemen yang ada pada set pertama tetapi tidak pada set kedua. Contohnya:

  *print(set1.difference(set2))  # Output: {1, 2}*

Meskipun set tidak mendukung pengindeksan dan pengurutan elemen, ia menyediakan berbagai fungsi dan metode built-in yang sangat berguna untuk operasi himpunan, seperti penambahan, penghapusan, pengecekan keanggotaan, serta operasi matematika seperti union, intersection, dan difference. Berikut adalah penjelasan tentang fungsi dan metode built-in yang tersedia untuk set:

- Fungsi len() digunakan untuk menghitung jumlah elemen dalam set.

  *my_set = {1, 2, 3, 4, 5}*

  *print(len(my_set))  # Output: 5*

  Fungsi len() akan mengembalikan jumlah elemen dalam set. Dalam contoh di atas, set my_set memiliki lima elemen, sehingga hasilnya adalah 5.

- Fungsi in digunakan untuk memeriksa apakah suatu elemen ada dalam set. Ini mengembalikan nilai boolean (True atau False).

  *my_set = {1, 2, 3, 4, 5}*

  *print(3 in my_set)  # Output: True*

  *print(6 in my_set)  # Output: False*

  Dengan menggunakan operator in, kita dapat memeriksa apakah elemen tertentu ada di dalam set. Operator ini

berguna ketika kita perlu memeriksa keberadaan suatu elemen dalam set.

- Metode add() digunakan untuk menambahkan satu elemen ke dalam set. Jika elemen tersebut sudah ada dalam set, tidak ada perubahan yang terjadi karena set tidak *mengizinkan duplikasi.*

  *my_set = {1, 2, 3}*

  *my_set.add(4)*

  *print(my_set)  # Output: {1, 2, 3, 4}*

  Fungsi add() menambahkan elemen baru ke dalam set, jika elemen tersebut belum ada. Jika elemen sudah ada, tidak ada perubahan yang dilakukan.

- Metode update() digunakan untuk menambahkan beberapa elemen sekaligus ke dalam set. Kita dapat menambahkan elemen dari iterable lain (seperti list, tuple, atau set lainnya).

  *my_set = {1, 2, 3}*

  *my_set.update([4, 5, 6])*

  *print(my_set)  # Output: {1, 2, 3, 4, 5, 6}*

  Dengan menggunakan update(), kita dapat menambahkan banyak elemen ke dalam set dalam satu kali operasi.

- Metode remove() digunakan untuk menghapus elemen dari set. Jika elemen yang ingin dihapus tidak ada dalam set, metode ini akan menghasilkan KeyError.

  *my_set = {1, 2, 3, 4}*

  *my_set.remove(3)*

  *print(my_set)  # Output: {1, 2, 4}*

remove() menghapus elemen yang ada dalam set berdasarkan nilai yang diberikan. Namun, jika elemen tersebut tidak ada dalam set, maka akan muncul error.

- Metode discard() juga digunakan untuk menghapus elemen dari set. Namun, jika elemen yang ingin dihapus tidak ada, metode ini tidak akan menghasilkan error seperti remove().

  *my_set = {1, 2, 3, 4}*

  *my_set.discard(3)*

  *print(my_set)  # Output: {1, 2, 4}*

  *# Tidak ada error meskipun elemen tidak ada*

  *my_set.discard(5)  # Tidak mengubah set, tidak ada error*

  *print(my_set)  # Output: {1, 2, 4}*

  discard() lebih aman digunakan jika kita tidak yakin apakah elemen tersebut ada di dalam set.

- Metode pop() digunakan untuk menghapus dan mengembalikan elemen acak dari set. Karena set tidak terurut, kita tidak dapat memilih elemen tertentu untuk dihapus, sehingga metode ini menghapus elemen secara acak.

  *my_set = {1, 2, 3, 4}*

  *popped_element = my_set.pop()*

  *print(popped_element)  # Output: 1 (elemen yang terhapus bisa berbeda)*

  *print(my_set)  # Output: {2, 3, 4}*

  pop() menghapus dan mengembalikan elemen acak dari set. Jika set kosong, metode ini akan menghasilkan KeyError.

- Metode clear() digunakan untuk menghapus semua elemen dari set, menjadikannya kosong.

  *my_set = {1, 2, 3, 4}*

  *my_set.clear()*

  *print(my_set)  # Output: set()*

  Dengan clear(), kita bisa menghapus seluruh isi set, menjadikannya kosong.

Ringkasan tentang Dictionary dan Set menunjukkan bahwa keduanya adalah struktur data penting dalam Python yang memudahkan pengelolaan dan manipulasi data. Dictionary memungkinkan kita untuk menyimpan data dalam bentuk pasangan key-value, yang memudahkan akses dan modifikasi data berdasarkan kunci, sementara Set menawarkan cara efisien untuk menangani koleksi elemen unik tanpa urutan dan mendukung operasi matematika seperti union, intersection, dan difference. Memahami kedua struktur data ini adalah langkah penting dalam membangun aplikasi Python yang efisien dan efektif, karena keduanya sering digunakan dalam berbagai konteks pengolahan data.

## 8.5 Latihan Soal

1. Jelaskan perbedaan utama antara Dictionary dan Set.
2. Bagaimana cara menambahkan elemen baru ke dalam Dictionary dan Set?
3. Buat contoh Dictionary yang berisi informasi mahasiswa dan cetak semua elemennya.
4. Tuliskan cara menghapus elemen dari Dictionary menggunakan `del` dan `pop()`.

5. Jelaskan kapan lebih baik menggunakan Set dibandingkan List.

# Bab 9: Penanganan Kesalahan dan Debugging

## 9.1 Pengertian Penanganan Kesalahan dan Debugging

Dalam pengembangan perangkat lunak, **penanganan kesalahan (error handling)** dan **debugging** adalah dua aspek penting yang memastikan program berjalan dengan baik dan sesuai dengan harapan. Kesalahan dalam program bisa terjadi karena berbagai faktor, seperti kesalahan penulisan kode, kesalahan logika, atau kondisi yang tidak terduga saat program dijalankan.

**Penanganan kesalahan** adalah proses mengidentifikasi, menangani, dan mencegah kesalahan agar tidak menyebabkan program berhenti secara tiba-tiba. Dengan menerapkan strategi penanganan kesalahan, program dapat memberikan respons yang lebih baik saat terjadi masalah, misalnya dengan menampilkan pesan kesalahan yang informatif atau mengambil langkah alternatif untuk tetap melanjutkan eksekusi.

Sementara itu, **debugging** adalah proses mencari, menganalisis, dan memperbaiki kesalahan atau bug dalam kode program. Debugging dilakukan untuk memastikan bahwa program berjalan sesuai dengan yang diharapkan. Proses ini biasanya melibatkan pemeriksaan kode secara manual, penggunaan alat bantu

debugging, serta pengujian berbagai skenario untuk menemukan sumber kesalahan dan memperbaikinya.

Kedua proses ini sangat penting dalam pengembangan perangkat lunak, karena membantu meningkatkan kualitas dan stabilitas program. Dengan penanganan kesalahan yang baik, program dapat lebih tangguh menghadapi situasi tak terduga. Sementara itu, debugging yang efektif memastikan bahwa semua fungsi dalam program bekerja dengan benar sebelum digunakan oleh pengguna. Penanganan kesalahan (error handling) dan teknik debugging adalah dua bagian penting dalam pemrograman yang membantu mengidentifikasi dan memperbaiki masalah dalam kode. Penanganan kesalahan memungkinkan kita untuk mencegah program crash akibat kesalahan yang tidak terduga, sedangkan debugging memberikan alat untuk menemukan dan memperbaiki bug dalam kode.

Di bab ini, kita akan mempelajari berbagai jenis kesalahan yang mungkin terjadi selama program berjalan, serta bagaimana cara menangani kesalahan tersebut menggunakan struktur kontrol seperti try, except, else, dan finally. Selain itu, kita juga akan membahas teknik debugging yang membantu kita menemukan dan memperbaiki bug dalam kode Python dengan menggunakan alat yang tersedia.

## 9.2 Jenis-Jenis Kesalahan dalam Pemrograman

Dalam pemrograman, kesalahan atau bug adalah hal yang umum terjadi dan dapat menyebabkan program tidak berjalan dengan benar. Kesalahan ini dapat dikategorikan ke dalam tiga jenis utama, yaitu Syntax Error, Runtime Error, dan Logical Error.

9.2.1 Syntax Error

- Terjadi ketika kode yang ditulis tidak sesuai dengan aturan sintaksis dari bahasa pemrograman yang digunakan.
- Biasanya dideteksi oleh compiler atau interpreter sebelum program dijalankan.
- Contoh umum: kesalahan dalam penulisan perintah, lupa tanda kurung, atau kesalahan dalam deklarasi variabel.
- Dampak: Program tidak bisa dijalankan sebelum semua syntax error diperbaiki.

Kesalahan ini biasanya dapat diatasi dengan memeriksa kembali sintaks kode dan menggunakan debugging tools yang disediakan oleh IDE atau compiler.

Contoh :

*print("Hello World'*

Pada contoh di atas, ada kesalahan karena tanda kutip yang tidak sepasang. Program akan menghasilkan error berikut:

*SyntaxError: EOL while scanning string literal*

Cara Mengatasi periksa kembali tanda kutip dan pastikan setiap tanda buka memiliki pasangan penutup,

9.2.2 Runtime Error

- Kesalahan yang terjadi saat program sedang berjalan, meskipun kode secara sintaksis benar.
- Penyebab umum meliputi:
    a. Pembagian dengan nol
    b. Akses elemen dalam list yang berada di luar indeks
    c. Penggunaan variabel yang belum dideklarasikan
    d. Kesalahan dalam alokasi memori
- Dampak: Program tiba-tiba berhenti atau menampilkan pesan error saat dijalankan.

Contoh :

```
x = 10
print(x / 0)
```

Pada contoh di atas, program mencoba membagi angka dengan nol, yang menghasilkan error:

*ZeroDivisionError: division by zero*

Cara Mengatasi tambahkan penanganan kesalahan untuk menghindari pembagian dengan nol. Untuk menangani runtime error, dapat digunakan exception handling atau pengecekan kondisi sebelum menjalankan suatu operasi yang berisiko.

9.2.3 Logical Error

- Terjadi ketika program berjalan tanpa menampilkan pesan error, tetapi menghasilkan output yang salah karena kesalahan dalam logika program.

- Contoh:
    a. Kesalahan dalam penggunaan operator dalam perhitungan
    b. Algoritma yang tidak sesuai dengan masalah yang ingin diselesaikan
    c. Kondisi dalam pernyataan if yang tidak tepat
- Dampak: Program tetap berjalan, tetapi memberikan hasil yang tidak sesuai dengan harapan.

Contoh kasus :

*x = 10*

*y = 5*

*z = x + y*

*print(z * 2)  # Seharusnya z seharusnya dikalikan 3, bukan 2*

Cara Mengatasi gunakan teknik debugging untuk memastikan bahwa setiap bagian program menghasilkan hasil yang diinginkan. Logical error adalah yang paling sulit ditemukan karena tidak ada pesan error yang muncul. Untuk mengatasinya, diperlukan pengujian menyeluruh, debugging, dan analisis alur program.

## 9.3 Penanganan Kesalahan dengan Try, Except, else dan finally

Dalam pemrograman, kesalahan atau exception dapat terjadi saat program dijalankan, terutama jika ada input yang tidak valid atau operasi yang tidak dapat dilakukan. Jika kesalahan ini tidak ditangani, program akan berhenti secara tiba-tiba dan menampilkan

pesan error. Oleh karena itu, Python menyediakan mekanisme try, except,else, dan finally untuk menangani kesalahan agar program tetap berjalan dengan baik.

### 9.3.1 Konsep Dasar Try-Except dalam Python

Blok try-except digunakan untuk menangkap dan menangani error yang terjadi dalam program. Try digunakan untuk menuliskan kode yang berpotensi menyebabkan kesalahan, sedangkan except digunakan untuk menangani kesalahan tersebut. Jika terjadi kesalahan dalam blok try, eksekusi akan berpindah ke blok except tanpa menghentikan program secara keseluruhan.

Mekanisme ini sangat berguna untuk mencegah crash pada aplikasi dan memberikan umpan balik yang lebih informatif kepada pengguna. Selain itu, Python memungkinkan penggunaan beberapa except untuk menangani berbagai jenis error yang berbeda, seperti ZeroDivisionError, ValueError, atau FileNotFoundError.

### 9.3.2 Jenis-Jenis Kesalahan yang Dapat Ditangani

Beberapa jenis error yang sering ditangani menggunakan try-except antara lain:

- ZeroDivisionError: Terjadi jika ada operasi pembagian dengan nol.
- ValueError: Terjadi saat tipe data yang dimasukkan tidak sesuai, misalnya pengguna memasukkan huruf padahal yang diharapkan adalah angka.
- IndexError: Terjadi jika mencoba mengakses indeks list atau tuple yang tidak ada.

- KeyError: Terjadi saat mencoba mengakses kunci yang tidak tersedia dalam dictionary.
- FileNotFoundError: Terjadi ketika mencoba membuka file yang tidak ditemukan.

Dengan memahami berbagai jenis error ini, programmer dapat menuliskan blok except yang sesuai untuk mengatasi kemungkinan kesalahan yang muncul.

9.3.3 Blok Finally untuk Eksekusi Akhir

Selain try dan except, Python juga menyediakan blok finally yang akan dieksekusi selalu, baik program mengalami error maupun tidak. Blok ini sering digunakan untuk membersihkan sumber daya, seperti menutup file atau koneksi database.

Blok finally sangat berguna dalam pengelolaan sumber daya agar tidak terjadi kebocoran memori atau kesalahan lain akibat tidak menutup suatu proses dengan benar.

9.3.4 Manfaat dan Penerapan Try-Except dalam Program

Penggunaan try-except memiliki beberapa manfaat utama dalam pengembangan perangkat lunak:

1. Mencegah Crash Program: Dengan menangani error, program dapat tetap berjalan meskipun terjadi kesalahan.
2. Meningkatkan Pengalaman Pengguna: Memberikan pesan error yang lebih informatif dibandingkan hanya menampilkan traceback error yang sulit dimengerti.
3. Mempermudah Debugging: Programmer dapat menangkap jenis error tertentu dan memberikan solusi atau peringatan yang sesuai.

4. Memastikan Pembersihan Sumber Daya: Dengan menggunakan blok finally, program dapat memastikan bahwa file, koneksi, atau sumber daya lainnya selalu ditutup dengan baik.

Sintaksnya seperti dibawah ini:

```
try:
    # Kode yang berpotensi menimbulkan kesalahan
except ErrorType:
    # Penanganan kesalahan
```

Contoh penerapannya:

```
try:
    x = 10 / 0
except ZeroDivisionError:
    print("Tidak bisa membagi dengan nol!")
```

Outputnya :

*Tidak bisa membagi dengan nol!*

Dalam praktiknya, mekanisme ini sering digunakan dalam aplikasi berbasis pengguna, seperti validasi input di aplikasi berbasis GUI atau penanganan error dalam komunikasi jaringan. Dengan memahami dan menerapkan try-except dengan baik, program akan menjadi lebih tangguh, fleksibel, dan user-friendly.

```python
# === 1. Contoh Dasar Try-Except ===
try:
    angka = int(input("Masukkan angka: "))  # Bisa menyebabkan ValueError jika input bukan angka
    hasil = 10 / angka  # Bisa menyebabkan ZeroDivisionError jika angka = 0
    print("Hasil:", hasil)
except ZeroDivisionError:
    print("Error: Pembagian dengan nol tidak diperbolehkan.")
except ValueError:
    print("Error: Input harus berupa angka.")
except Exception as e:  # Menangkap error lainnya
    print(f"Terjadi error: {e}")
else:
    print("Operasi berhasil tanpa error.")
finally:
    print("Blok finally: Eksekusi selesai.")

# === 2. Menangani IndexError ===
try:
    daftar = [1, 2, 3]
    print("Elemen ke-5:", daftar[4])  # IndexError karena hanya ada 3 elemen
except IndexError:
    print("Error: Indeks tidak tersedia dalam list.")

# === 3. Menangani KeyError dalam Dictionary ===
try:
    data = {"nama": "Kevin", "usia": 25}
    print("Pekerjaan:", data["pekerjaan"])  # KeyError karena tidak ada kunci 'pekerjaan'
except KeyError:
    print("Error: Kunci tidak ditemukan dalam dictionary.")

# === 4. Menangani FileNotFoundError ===
try:
    with open("data_tidak_ada.txt", "r") as file:  # File tidak ditemukan
        isi = file.read()
        print(isi)
except FileNotFoundError:
    print("Error: File tidak ditemukan.")

# === 5. Contoh Penggunaan Finally ===
try:
    file = open("contoh.txt", "w")
    file.write("Hello, world!")
except Exception as e:
    print("Terjadi error saat menulis file:", e)
finally:
    file.close()  # Pastikan file tetap ditutup
    print("File berhasil ditutup.")

```

Beberapa jenis kesalahan dapat ditangani dalam satu blok except dengan menggunakan tuple, atau menulis beberapa blok except untuk menangani jenis kesalahan yang berbeda.

*try:*

*x = int(input("Masukkan angka: "))*

*y = 10 / x*

*except ValueError:*

```
    print("Input tidak valid, harap masukkan angka.")
except ZeroDivisionError:
    print("Tidak bisa membagi dengan nol!")
```

Output:

- Jika input adalah angka non-angka: "Input tidak valid, harap masukkan angka."
- Jika input adalah nol: "Tidak bisa membagi dengan nol!"

Blok else digunakan setelah blok try dan hanya dijalankan jika tidak ada kesalahan yang terjadi dalam blok try. Contonnya:

```
try:
    x = 10 / 2
except ZeroDivisionError:
    print("Tidak bisa membagi dengan nol!")
else:
    print("Pembagian berhasil!")
```

Outputnya:

*Pembagian berhasil!*

Kemudian blok finally dijalankan selalu, baik ada kesalahan atau tidak. Ini digunakan untuk kode yang harus dieksekusi tidak peduli apakah terjadi kesalahan atau tidak, seperti menutup file atau melepaskan sumber daya lainnya. Contohnya :

```
try:
    file = open('data.txt', 'r')
except FileNotFoundError:
    print("File tidak ditemukan.")
finally:
```

```
    print("Menutup file...")
    file.close()
```

Ouputnya :

*Menutup file...*

## 9.4 Teknik Debugging

Debugging adalah proses mendeteksi, menganalisis, dan memperbaiki kesalahan dalam kode program. Kesalahan ini bisa berupa bug sintaksis, logika, atau runtime error yang menyebabkan program tidak berjalan sebagaimana mestinya. Salah satu teknik debugging yang paling sederhana adalah dengan menyisipkan pernyataan print() untuk memeriksa nilai variabel atau output di berbagai titik dalam program. Contohnya :

```
def calculate_area(radius):
    print(f"radius: {radius}")  # Debugging
    area = 3.14 * radius ** 2
    print(f"area: {area}")  # Debugging
    return area
calculate_area(5)
```

Outputnya :

*radius: 5*

*area: 78.5*

Berikut beberapa teknik debugging yang sering digunakan:

1. Print Statement

a. Menambahkan print() untuk menampilkan nilai variabel atau status program pada titik tertentu.
b. Digunakan untuk memeriksa apakah suatu bagian kode telah dieksekusi atau tidak.
c. Cocok untuk debugging sederhana dan cepat.

Contoh codingnya :

```
def hitung_total(a, b):
    total = a + b
    print(f"Nilai a: {a}, Nilai b: {b}")  # Debugging dengan print statement
    print(f"Total: {total}")  # Debugging dengan print statement
    return total
# Panggil fungsi
hitung_total(5, 10)
```

2. Logging
   a. Menggunakan modul logging untuk mencatat informasi selama program berjalan.
   b. Memiliki berbagai tingkat keparahan, seperti DEBUG, INFO, WARNING, ERROR, dan CRITICAL.
   c. Memungkinkan pencatatan informasi tanpa perlu menghapus kode setelah debugging selesai.

Contoh codingnya:

```
import logging
# Mengatur konfigurasi logging
logging.basicConfig(level=logging.DEBUG)
```

```
def hitung_total(a, b):
    logging.debug(f"Nilai a: {a}, Nilai b: {b}")  # Debug level
    total = a + b
    logging.info(f"Total: {total}")  # Info level
    return total
# Panggil fungsi
hitung_total(5, 10)
```

3. Debugger
    a. Menggunakan alat seperti pdb (*Python Debugger*) atau debugger bawaan IDE (PyCharm, VS Code).
    b. Memungkinkan eksekusi program dihentikan sementara (breakpoint) untuk memeriksa nilai variabel secara interaktif.
    c. Memudahkan analisis alur eksekusi program dan menemukan sumber kesalahan lebih cepat.

Contoh codingnya :

```
import pdb
def hitung_total(a, b):
    pdb.set_trace()  # Menetapkan breakpoint di sini
    total = a + b
    return total
# Panggil fungsi
hitung_total(5, 10)
```

4. Code Review
    a. Memeriksa kode secara manual atau meminta rekan kerja untuk meninjau kode (*peer review*).

b. Membantu menemukan kesalahan yang mungkin terlewat serta meningkatkan kualitas kode.
c. Dapat dilakukan dengan teknik *pair programming* atau menggunakan alat seperti GitHub Pull Request.

Contoh codingnya:

```
# Perintah Git untuk membuat pull request di GitHub
git checkout -b feature/penambahan-fitur
git add .
git commit -m "Menambahkan fitur baru"
git push origin feature/penambahan-fitur
# Setelah itu, buka GitHub dan buat Pull Request ke branch utama.
```

5. Unit Testing
   a. Menggunakan tes otomatis untuk memastikan bagian tertentu dari kode bekerja sesuai yang diharapkan.
   b. Menggunakan framework seperti unittest atau pytest.
   c. Membantu mendeteksi bug lebih awal dan memastikan perubahan kode tidak merusak fungsionalitas lain.

   Contoh codingnya:

```
import unittest
def hitung_total(a, b):
return a + b
class TestHitungTotal(unittest.TestCase):
 def test_hitung_total(self):
        self.assertEqual(hitung_total(5, 10), 15) # Memastikan 5 + 10 = 15
```

```
self.assertEqual(hitung_total(-5, 5), 0)   # Memastikan -5 +
5 = 0
# Jalankan unit test
if __name__ == "__main__":
 unittest.main()
```

Python memiliki pustaka built-in bernama pdb (Python Debugger) yang memungkinkan kita untuk menghentikan eksekusi program dan memeriksa keadaan program secara interaktif. Contoh dalam penerapannya:

```
import pdb
def calculate_area(radius):
   pdb.set_trace()  # Menyisipkan breakpoint
   area = 3.14 * radius ** 2
   return area
calculate_area(5)
```

Program akan berhenti di pdb.set_trace() dan memberi kita akses interaktif untuk mengevaluasi variabel atau melangkah melalui kode langkah demi langkah. Debugging Menggunakan IDE, Sebagian besar IDE (Integrated Development Environment) seperti PyCharm, VSCode, atau Eclipse menyediakan fitur debugging yang lebih canggih, seperti breakpoint, pelacakan variabel, dan visualisasi eksekusi kode secara real-time.ini bisa diatur dengan breakpoint di baris kode tertentu dan menjalankan kode dengan "debug mode". IDE memungkinkan pelacakan nilai variabel secara otomatis saat eksekusi.

Dalam pengembangan perangkat lunak, penanganan kesalahan dan debugging adalah keterampilan penting yang perlu dikuasai. Dengan memahami cara menangani kesalahan secara efektif menggunakan try, except, else, dan finally, serta menggunakan teknik debugging seperti print(), pdb, dan fitur debugging IDE, kita dapat menulis kode yang lebih aman, andal, dan mudah dipelihara. Menguasai teknik-teknik ini akan sangat membantu dalam meningkatkan produktivitas dan kualitas kode yang dibuat.

## 9.5 Latihan Soal

1. Jelaskan perbedaan antara Syntax Error, Runtime Error, dan Logical Error.
2. Buat contoh program sederhana yang menggunakan `try-except` untuk menangani kesalahan input pengguna.
3. Bagaimana cara menggunakan `print()` sebagai teknik debugging?
4. Sebutkan tiga teknik debugging selain `print()` dan berikan penjelasannya.
5. Apa fungsi dari blok `finally` dalam `try-except`?

# Bab 10: Latihan dan Proyek Mini

## 10.1 Pengertian Latihan dan Proyek Mini

Dalam pembelajaran pemrograman, **latihan** dan **proyek mini** merupakan langkah penting untuk memahami konsep secara lebih mendalam dan menerapkannya dalam situasi nyata. Belajar teori saja tidak cukup, karena pemrograman adalah keterampilan yang berkembang dengan praktik terus-menerus. Pemrograman adalah keterampilan yang diperoleh dengan terus berlatih. Latihan dan proyek mini memainkan peran penting dalam memahami dan menguasai konsep-konsep pemrograman. Latihan membantu memperkuat pemahaman dasar seperti sintaks dan struktur data, sementara proyek mini memungkinkan aplikasi dari konsep-konsep tersebut dalam skenario nyata. Dengan melibatkan latihan terarah dan proyek mini, seseorang dapat memperdalam pemahaman tentang cara mengembangkan solusi perangkat lunak sederhana yang fungsional.

**Latihan** membantu dalam memperkuat pemahaman tentang konsep dasar pemrograman, seperti sintaks, logika, dan struktur data. Dengan mengerjakan berbagai latihan, pengguna dapat mengasah keterampilan dalam menulis kode, mengenali pola pemrograman, dan menyelesaikan masalah secara sistematis. Latihan yang terarah

juga memungkinkan seseorang untuk menguasai teknik pemrograman secara bertahap.

Sementara itu, **proyek mini** adalah bentuk penerapan dari konsep-konsep yang telah dipelajari. Melalui proyek mini, pengguna dapat mengembangkan solusi nyata dengan menggabungkan berbagai aspek pemrograman, seperti penggunaan fungsi, struktur data, dan algoritma. Proyek mini bisa berupa pembuatan kalkulator sederhana, program manajemen data, atau aplikasi kecil lainnya yang memiliki tujuan fungsional tertentu.

Latihan dan proyek mini memiliki peran penting dalam meningkatkan keterampilan pemrograman. Latihan membantu memahami konsep, sementara proyek mini memungkinkan eksplorasi lebih lanjut dalam pengembangan perangkat lunak. Dengan kombinasi keduanya, seseorang dapat lebih siap untuk menghadapi tantangan dalam dunia pemrograman yang lebih kompleks.

## 10.2 Latihan Soal

Latihan ini bertujuan untuk mengasah pemahaman terkait konsep-konsep pemrograman yang telah dipelajari pada bab sebelumnya. Soal-soal berikut mencakup berbagai aspek pemrograman, mulai dari struktur data, fungsi, kontrol alur, hingga penanganan kesalahan.

10.2.1 Struktur Data – List dan Perulangan

Buatlah sebuah List yang berisi nama-nama kota, lalu tampilkan setiap elemennya menggunakan perulangan.

Pertanyaan:

- Buat list berisi minimal lima nama kota.
- Gunakan perulangan untuk mencetak setiap nama kota dalam list tersebut.

10.2.2 Fungsi – Menghitung Luas Persegi Panjang

Buat fungsi yang menghitung luas persegi panjang dengan rumus:

luas=panjang×lebar\text{luas} = \text{panjang} \times \text{lebar}luas=panjang×lebar

Pertanyaan:

- Buat fungsi bernama hitung_luas_persegi_panjang(panjang, lebar).
- Fungsi harus menerima dua parameter: panjang dan lebar.
- Fungsi harus mengembalikan hasil perhitungan luas.
- Panggil fungsi tersebut dengan beberapa nilai berbeda untuk menguji hasilnya.

10.2.3 Struktur Kontrol – Menentukan Bilangan Ganjil atau Genap

Buat program sederhana yang meminta pengguna memasukkan sebuah angka, lalu menentukan apakah angka tersebut ganjil atau genap.

Pertanyaan:

- Program harus menerima input angka dari pengguna.

- Gunakan struktur kontrol if-else untuk menentukan apakah angka tersebut ganjil atau genap.
- Cetak hasilnya ke layar.

10.2.4 Dictionary – Menyimpan Data Mahasiswa

Buat sebuah dictionary yang menyimpan informasi mahasiswa dengan atribut berikut:

- nama: Nama mahasiswa
- nim: Nomor Induk Mahasiswa
- jurusan: Program studi mahasiswa

Pertanyaan:

- Buat dictionary yang berisi data minimal tiga mahasiswa.
- Cetak seluruh elemen dalam dictionary dengan format yang rapi.

10.2.5 Try-Except – Menangani Kesalahan Input

Buat program yang menerima dua angka dari pengguna, lalu membagi angka pertama dengan angka kedua. Program harus menangani kesalahan jika pengguna memasukkan nol sebagai pembagi atau memasukkan input yang bukan angka.

Pertanyaan:

- Program harus meminta pengguna memasukkan dua angka.
- Gunakan try-except untuk menangani kesalahan berikut:
  a. Pembagian dengan nol
  b. Input yang bukan angka
- Jika terjadi kesalahan, tampilkan pesan yang sesuai.

## 10.3 Proyek Mini

Proyek mini bertujuan untuk mengaplikasikan konsep pemrograman dalam skenario yang lebih praktis dan kompleks. Dengan mengerjakan proyek-proyek ini, pemahaman terhadap struktur data, fungsi, serta kontrol alur dalam pemrograman dapat semakin diperkuat.

10.3.1 Kalkulator Sederhana

Kalkulator sederhana adalah proyek dasar yang mengimplementasikan operasi aritmatika seperti penjumlahan, pengurangan, perkalian, dan pembagian. Pengguna dapat memasukkan dua angka serta memilih operasi yang diinginkan. Proyek ini mengasah pemahaman tentang input-output serta penggunaan struktur kontrol seperti kondisi dan perulangan. Contoh codingnya :

```
# Fungsi untuk penjumlahan
def tambah(x, y):
    return x + y
# Fungsi untuk pengurangan
def kurang(x, y):
    return x - y
# Fungsi untuk perkalian
def kali(x, y):
    return x * y
# Fungsi untuk pembagian
def bagi(x, y):
```

```
    if y == 0:
        return "Error! Pembagian dengan nol."
    else:
        return x / y
# Menampilkan menu
print("Pilih operasi:")
print("1. Penjumlahan")
print("2. Pengurangan")
print("3. Perkalian")
print("4. Pembagian")
# Meminta input dari pengguna
pilih = input("Masukkan pilihan (1/2/3/4): ")
# Meminta input angka
num1 = float(input("Masukkan angka pertama: "))
num2 = float(input("Masukkan angka kedua: "))
# Menjalankan operasi sesuai pilihan
if pilih == '1':
    print(f"Hasil: {num1} + {num2} = {tambah(num1, num2)}")
elif pilih == '2':
    print(f"Hasil: {num1} - {num2} = {kurang(num1, num2)}")
elif pilih == '3':
    print(f"Hasil: {num1} * {num2} = {kali(num1, num2)}")
elif pilih == '4':
    print(f"Hasil: {num1} / {num2} = {bagi(num1, num2)}")
```

*else:*

*print("Input tidak valid")*

Contoh Output:

*Pilih operasi:*

*1. Penjumlahan*

*2. Pengurangan*

*3. Perkalian*

*4. Pembagian*

*Masukkan pilihan (1/2/3/4): 4*

*Masukkan angka pertama: 10*

*Masukkan angka kedua: 0*

*Hasil: 10.0 / 0 = Error! Pembagian dengan nol.*

Program ini sudah menangani beberapa kemungkinan kesalahan, seperti pembagian dengan nol. Anda dapat mengembangkan lebih lanjut dengan menambahkan antarmuka grafis menggunakan pustaka seperti Tkinter jika diinginkan.

10.3.2 Aplikasi Daftar Kontak

Aplikasi ini berfungsi sebagai buku telepon sederhana yang memungkinkan pengguna menambahkan, menghapus, dan mencari kontak berdasarkan nama. Dengan proyek ini, konsep penggunaan dictionary dalam menyimpan data berbasis kunci (nama) dan nilai (nomor telepon) dapat dipraktikkan dengan lebih mendalam.

Contoh codingnya :

*# Fungsi untuk menambahkan kontak*

*def tambah_kontak(buku_telepon, nama, nomor):*

*buku_telepon[nama] = nomor*

```
    print(f"Kontak {nama} berhasil ditambahkan!")
# Fungsi untuk menghapus kontak
def hapus_kontak(buku_telepon, nama):
    if nama in buku_telepon:
        del buku_telepon[nama]
        print(f"Kontak {nama} berhasil dihapus!")
    else:
        print(f"Kontak {nama} tidak ditemukan.")
# Fungsi untuk mencari kontak
def cari_kontak(buku_telepon, nama):
    if nama in buku_telepon:
        print(f"Nomor {nama}: {buku_telepon[nama]}")
    else:
        print(f"Kontak {nama} tidak ditemukan.")
# Fungsi untuk menampilkan semua kontak
def tampilkan_kontak(buku_telepon):
    if buku_telepon:
        print("Daftar Kontak:")
        for nama, nomor in buku_telepon.items():
            print(f"{nama}: {nomor}")
    else:
        print("Buku telepon kosong.")
# Program utama
buku_telepon = {}
while True:
    print("\nPilih opsi:")
```

```
        print("1. Tambah Kontak")
        print("2. Hapus Kontak")
        print("3. Cari Kontak")
        print("4. Tampilkan Semua Kontak")
        print("5. Keluar")
            pilihan = input("Masukkan pilihan (1/2/3/4/5): ")
        if pilihan == '1':
            nama = input("Masukkan nama: ")
            nomor = input("Masukkan nomor telepon: ")
            tambah_kontak(buku_telepon, nama, nomor)
            elif pilihan == '2':
            nama = input("Masukkan nama kontak yang ingin dihapus: ")
            hapus_kontak(buku_telepon, nama)
            elif pilihan == '3':
            nama = input("Masukkan nama kontak yang ingin dicari: ")
            cari_kontak(buku_telepon, nama)
            elif pilihan == '4':
            tampilkan_kontak(buku_telepon)
            elif pilihan == '5':
            print("Terima kasih!")
            break
            else:
            print("Pilihan tidak valid. Silakan coba lagi.")
```

Dictionary digunakan dalam program ini untuk menyimpan data kontak, dengan nama sebagai kunci dan nomor telepon sebagai nilai. Program ini menyediakan berbagai pilihan bagi pengguna untuk mengelola data kontak, seperti menambah, menghapus, mencari, dan menampilkan kontak yang ada. Fungsi-fungsi yang ada, seperti tambah_kontak(), hapus_kontak(), cari_kontak(), dan tampilkan_kontak(), masing-masing berfungsi untuk menangani operasi yang relevan, seperti menambahkan kontak baru, menghapus kontak yang sudah ada, mencari kontak berdasarkan nama, dan menampilkan seluruh daftar kontak yang telah disimpan. Dengan menggunakan struktur data dictionary, pengelolaan data kontak menjadi lebih efisien dan mudah diakses berdasarkan nama sebagai kunci.

10.3.3 Pengelolaan Data Mahasiswa

Dalam proyek ini, pengguna dapat memasukkan data mahasiswa yang mencakup nama, NIM, dan jurusan. Data tersebut dapat disimpan dalam struktur list atau dictionary, kemudian ditampilkan kembali sesuai kebutuhan. Proyek ini melatih keterampilan dalam manipulasi struktur data dan pengolahan input dari pengguna. Contoh codingnya menggunakan list:

```
# Fungsi untuk menambahkan data mahasiswa
def tambah_mahasiswa(data_mahasiswa, nama, nim, jurusan):
    data_mahasiswa.append({'Nama': nama, 'NIM': nim, 'Jurusan': jurusan})
    print(f"Data mahasiswa {nama} berhasil ditambahkan!")
```

```
    # Fungsi untuk menampilkan semua data mahasiswa
    def tampilkan_mahasiswa(data_mahasiswa):
        if data_mahasiswa:
            print("Daftar Mahasiswa:")
            for mahasiswa in data_mahasiswa:
                print(f"Nama: {mahasiswa['Nama']}, NIM: {mahasiswa['NIM']}, Jurusan: {mahasiswa['Jurusan']}")
        else:
            print("Data mahasiswa kosong.")
    # Program utama
    data_mahasiswa = []
    while True:
        print("\nPilih opsi:")
        print("1. Tambah Data Mahasiswa")
        print("2. Tampilkan Semua Data Mahasiswa")
        print("3. Keluar")

        pilihan = input("Masukkan pilihan (1/2/3): ")
        if pilihan == '1':
            nama = input("Masukkan nama mahasiswa: ")
            nim = input("Masukkan NIM mahasiswa: ")
            jurusan = input("Masukkan jurusan mahasiswa: ")
            tambah_mahasiswa(data_mahasiswa, nama, nim, jurusan)
```

```
    elif pilihan == '2':
        tampilkan_mahasiswa(data_mahasiswa)
        elif pilihan == '3':
        print("Terima kasih!")
        break
        else:
        print("Pilihan tidak valid. Silakan coba lagi.")
```

Outputnya:

*Pilih opsi:*

*1. Tambah Data Mahasiswa*

*2. Tampilkan Semua Data Mahasiswa*

*3. Keluar*

*Masukkan pilihan (1/2/3): 1*

*Masukkan nama mahasiswa: Budi*

*Masukkan NIM mahasiswa: 123456*

*Masukkan jurusan mahasiswa: Teknik Informatika*

*Data mahasiswa Budi berhasil ditambahkan!*

List digunakan untuk menyimpan data mahasiswa, di mana setiap mahasiswa disimpan dalam bentuk dictionary dengan kunci 'Nama', 'NIM', dan 'Jurusan'. Setiap data mahasiswa yang dimasukkan akan disimpan sebagai elemen dalam list, sehingga memungkinkan pengguna untuk mengelola dan mengakses data secara sistematis. Program ini memberikan pilihan kepada pengguna untuk menambah data mahasiswa baru ke dalam list, serta menampilkan seluruh data mahasiswa yang ada dalam format yang terstruktur. Dengan cara ini, pengguna dapat dengan mudah melihat

dan mengelola data mahasiswa yang telah dimasukkan. Contoh coding menggunakan dictionary, pada contoh ini, setiap data mahasiswa disimpan dalam dictionary dengan kunci NIM dan nilai berisi informasi lainnya seperti nama dan jurusan.

```
# Fungsi untuk menambahkan data mahasiswa
def tambah_mahasiswa(data_mahasiswa, nim, nama, jurusan):
    data_mahasiswa[nim] = {'Nama': nama, 'Jurusan': jurusan}
    print(f"Data mahasiswa {nama} berhasil ditambahkan!")
# Fungsi untuk menampilkan semua data mahasiswa
def tampilkan_mahasiswa(data_mahasiswa):
    if data_mahasiswa:
        print("Daftar Mahasiswa:")
        for nim, info in data_mahasiswa.items():
            print(f"NIM: {nim}, Nama: {info['Nama']}, Jurusan: {info['Jurusan']}")
    else:
        print("Data mahasiswa kosong.")
# Program utama
data_mahasiswa = {}
while True:
    print("\nPilih opsi:")
    print("1. Tambah Data Mahasiswa")
    print("2. Tampilkan Semua Data Mahasiswa")
    print("3. Keluar")
```

```
        pilihan = input("Masukkan pilihan (1/2/3): ")
    if pilihan == '1':
        nim = input("Masukkan NIM mahasiswa: ")
        nama = input("Masukkan nama mahasiswa: ")
        jurusan = input("Masukkan jurusan mahasiswa: ")
        tambah_mahasiswa(data_mahasiswa, nim, nama, jurusan)
        elif pilihan == '2':
        tampilkan_mahasiswa(data_mahasiswa)

    elif pilihan == '3':
        print("Terima kasih!")
        break
        else:
        print("Pilihan tidak valid. Silakan coba lagi.")
```

Outputnya:

*Pilih opsi:*

*1. Tambah Data Mahasiswa*

*2. Tampilkan Semua Data Mahasiswa*

*3. Keluar*

*Masukkan pilihan (1/2/3): 2*

*Daftar Mahasiswa:*

*NIM: 123456, Nama: Budi, Jurusan: Teknik Informatika*

Dictionary digunakan untuk menyimpan data mahasiswa, di mana NIM mahasiswa menjadi kunci dan informasi lain seperti Nama dan Jurusan disimpan sebagai nilai dalam dictionary tersebut.

Program ini memungkinkan pengguna untuk memasukkan data mahasiswa dengan NIM sebagai kunci, sehingga memudahkan akses dan pencarian data berdasarkan NIM yang bersangkutan. Dengan menggunakan dictionary, proses pengelolaan data mahasiswa menjadi lebih efisien, karena pencarian data dapat dilakukan dengan cepat menggunakan NIM sebagai identifikasi unik untuk setiap mahasiswa.

10.3.4 To-Do List

Proyek to-do list berfungsi sebagai pencatat daftar tugas harian. Pengguna dapat menambahkan tugas baru, melihat daftar tugas yang harus diselesaikan, serta menghapus tugas yang sudah selesai. Dalam proyek ini, konsep list dan perulangan menjadi aspek penting untuk dikuasai. Contoh codingnya :

```
# Fungsi untuk menambah tugas
def tambah_tugas(to_do_list, tugas):
    to_do_list.append(tugas)
    print(f"Tugas '{tugas}' berhasil ditambahkan!")
# Fungsi untuk menampilkan semua tugas
def tampilkan_tugas(to_do_list):
    if to_do_list:
        print("\nDaftar Tugas yang Harus Diselesaikan:")
        for index, tugas in enumerate(to_do_list, start=1):
            print(f"{index}. {tugas}")
    else:
        print("Tidak ada tugas yang harus diselesaikan.")
```

```
    # Fungsi untuk menghapus tugas yang sudah selesai
    def hapus_tugas(to_do_list, nomor_tugas):
        if nomor_tugas <= len(to_do_list):
            tugas_hapus = to_do_list.pop(nomor_tugas - 1)
            print(f"Tugas '{tugas_hapus}' telah dihapus dari daftar.")
        else:
            print("Nomor tugas tidak valid!")
    # Program utama
    to_do_list = []
    while True:
        print("\nPilih opsi:")
        print("1. Tambah Tugas")
        print("2. Tampilkan Daftar Tugas")
        print("3. Hapus Tugas")
        print("4. Keluar")

        pilihan = input("Masukkan pilihan (1/2/3/4): ")
        if pilihan == '1':
            tugas = input("Masukkan tugas yang ingin ditambahkan: ")
            tambah_tugas(to_do_list, tugas)
            elif pilihan == '2':
            tampilkan_tugas(to_do_list)

        elif pilihan == '3':
```

```
        tampilkan_tugas(to_do_list)
        if to_do_list:
            nomor_tugas = int(input("Masukkan nomor tugas yang sudah selesai untuk dihapus: "))
            hapus_tugas(to_do_list, nomor_tugas)
        elif pilihan == '4':
        print("Terima kasih! Program selesai.")
        break
        else:
        print("Pilihan tidak valid. Silakan coba lagi.")
```

Outputnya:

*Pilih opsi:*

*1. Tambah Tugas*

*2. Tampilkan Daftar Tugas*

*3. Hapus Tugas*

*4. Keluar*

*Masukkan pilihan (1/2/3/4): 1*

*Masukkan tugas yang ingin ditambahkan: Belajar Python*

*Tugas 'Belajar Python' berhasil ditambahkan!*

*Pilih opsi:*

*1. Tambah Tugas*

*2. Tampilkan Daftar Tugas*

*3. Hapus Tugas*

*4. Keluar*

*Masukkan pilihan (2):*

*Daftar Tugas yang Harus Diselesaikan:*

*1. Belajar Python*

*Pilih opsi:*

*1. Tambah Tugas*

*2. Tampilkan Daftar Tugas*

*3. Hapus Tugas*

*4. Keluar*

*Masukkan pilihan (3):*

*Daftar Tugas yang Harus Diselesaikan:*

*1. Belajar Python*

*Masukkan nomor tugas yang sudah selesai untuk dihapus: 1*

*Tugas 'Belajar Python' telah dihapus dari daftar.*

*Pilih opsi:*

*1. Tambah Tugas*

*2. Tampilkan Daftar Tugas*

*3. Hapus Tugas*

*4. Keluar*

*Masukkan pilihan (4): Terima kasih! Program selesai.*

List digunakan untuk menyimpan tugas-tugas yang harus diselesaikan, di mana setiap tugas dimasukkan sebagai elemen dalam list `to_do_list`. Fungsi `tambah_tugas()` digunakan untuk menambahkan tugas baru ke dalam daftar, sementara fungsi `tampilkan_tugas()` akan menampilkan seluruh tugas yang ada dalam daftar. Fungsi ini juga menggunakan perulangan (`for loop`) untuk menampilkan tugas satu per satu beserta nomor urutannya. Untuk menghapus tugas yang telah selesai, digunakan fungsi

`hapus_tugas()`, yang menghapus tugas berdasarkan nomor yang dipilih oleh pengguna. Perulangan `while` digunakan untuk memberikan menu pilihan kepada pengguna dan menjalankan opsi sesuai dengan input yang diberikan, memungkinkan program untuk terus berfungsi hingga pengguna memilih untuk keluar.

10.3.5 Konversi Suhu

Proyek ini berfokus pada pembuatan program yang dapat mengonversi suhu dari Celsius ke Fahrenheit dan sebaliknya. Pengguna akan diminta memasukkan nilai suhu dan memilih jenis konversi yang diinginkan. Proyek ini mengasah keterampilan dalam penggunaan fungsi serta perhitungan numerik dalam pemrograman. Contoh codingnya :

```
# Fungsi untuk konversi Celsius ke Fahrenheit
def celsius_ke_fahrenheit(celsius):
    return (celsius * 9/5) + 32
# Fungsi untuk konversi Fahrenheit ke Celsius
def fahrenheit_ke_celsius(fahrenheit):
    return (fahrenheit - 32) * 5/9
# Program utama
while True:
    print("\nPilih opsi konversi suhu:")
    print("1. Celsius ke Fahrenheit")
    print("2. Fahrenheit ke Celsius")
    print("3. Keluar")
        pilihan = input("Masukkan pilihan (1/2/3): ")
```

```
    if pilihan == '1':
        celsius = float(input("Masukkan suhu dalam Celsius: "))
        fahrenheit = celsius_ke_fahrenheit(celsius)
        print(f"{celsius}°C = {fahrenheit}°F")
        elif pilihan == '2':
        fahrenheit = float(input("Masukkan suhu dalam Fahrenheit: "))
        celsius = fahrenheit_ke_celsius(fahrenheit)
        print(f"{fahrenheit}°F = {celsius}°C")
        elif pilihan == '3':
        print("Terima kasih! Program selesai.")
        break
        else:
        print("Pilihan tidak valid. Silakan coba lagi.")
```

Ouput:

*Pilih opsi konversi suhu:*

*1. Celsius ke Fahrenheit*

*2. Fahrenheit ke Celsius*

*3. Keluar*

*Masukkan pilihan (1/2/3): 1*

*Masukkan suhu dalam Celsius: 25*

*25.0°C = 77.0°F*

*Pilih opsi konversi suhu:*

*1. Celsius ke Fahrenheit*

*2. Fahrenheit ke Celsius*

*3. Keluar*

*Masukkan pilihan (2): 77*

*77.0°F = 25.0°C*

*Pilih opsi konversi suhu:*

*1. Celsius ke Fahrenheit*

*2. Fahrenheit ke Celsius*

*3. Keluar*

*Masukkan pilihan (3): Terima kasih! Program selesai.*

Fungsi `celsius_ke_fahrenheit()` digunakan untuk mengonversi suhu dari Celsius ke Fahrenheit dengan rumus `(Celsius * 9/5) + 32`, sedangkan fungsi `fahrenheit_ke_celsius()` digunakan untuk mengonversi suhu dari Fahrenheit ke Celsius dengan rumus `(Fahrenheit - 32) * 5/9`. Program utama menyediakan menu kepada pengguna untuk memilih jenis konversi yang diinginkan, kemudian meminta input suhu dan menampilkan hasil konversi. Program ini terus berjalan hingga pengguna memilih untuk keluar dengan memilih opsi 3.

Dengan mengerjakan proyek-proyek mini ini, pemahaman terhadap berbagai aspek pemrograman dapat lebih meningkat, serta memberikan pengalaman praktis dalam menyusun program yang lebih terstruktur dan fungsional.

## 10.4 Implementasi Proyek Mini

10.4.1 Cara Implementasi

Dalam dunia pemrograman, proyek mini sering digunakan untuk memperkuat pemahaman konsep dasar seperti input-output, percabangan, dan penanganan kesalahan. Salah satu proyek mini yang mudah dibuat dan sangat bermanfaat adalah kalkulator sederhana. Kalkulator ini memungkinkan pengguna untuk melakukan operasi matematika dasar seperti penjumlahan, pengurangan, perkalian, dan pembagian.

Dengan mengembangkan proyek ini, pengguna dapat memahami cara menerima input dari pengguna, melakukan operasi berdasarkan pilihan tertentu, serta menangani berbagai skenario error yang mungkin terjadi selama proses eksekusi program.

10.4.2 Tujuan Proyek

Proyek kalkulator sederhana ini memiliki beberapa tujuan utama, di antaranya:

1. Mempraktikkan dasar-dasar pemrograman, seperti menangani input dan output serta menerapkan percabangan dalam program.
2. Meningkatkan pemahaman tentang operator aritmetika, termasuk penjumlahan, pengurangan, perkalian, dan pembagian.
3. Melatih kemampuan dalam menangani kesalahan yang mungkin muncul selama eksekusi program, seperti kesalahan input atau pembagian dengan nol.
4. Membantu pengguna dalam melakukan perhitungan matematika dasar dengan cepat dan akurat melalui program yang mudah digunakan.

### 10.4.3 Konsep Dasar

Kalkulator sederhana ini bekerja dengan menerima dua angka dari pengguna dan satu operator matematika yang menentukan jenis operasi yang akan dilakukan. Program akan membaca input tersebut, memproses operasi berdasarkan operator yang diberikan, lalu menampilkan hasilnya.

Beberapa konsep dasar yang diterapkan dalam kalkulator ini meliputi:

1. Input dan Output – Program akan meminta pengguna untuk memasukkan angka pertama, operator, dan angka kedua, lalu menampilkan hasil perhitungannya.
2. Percabangan (Conditional Statements) – Digunakan untuk menentukan operasi yang akan dijalankan berdasarkan input operator yang diberikan oleh pengguna.
3. Penanganan Kesalahan (Error Handling) – Program harus menangani kemungkinan error, seperti input yang bukan angka atau pembagian dengan nol, agar tidak terjadi crash.

### 10.4.4 Implementasi Kalkulator

Implementasi kalkulator sederhana mencakup beberapa langkah utama:

1. Meminta pengguna untuk memasukkan angka pertama.
2. Meminta pengguna untuk memasukkan operator matematika (+, -, *, /).
3. Meminta pengguna untuk memasukkan angka kedua.
4. Memproses operasi matematika berdasarkan input yang diberikan.

5. Menampilkan hasil perhitungan kepada pengguna.
6. Menangani kemungkinan kesalahan, seperti salah memasukkan input atau mencoba membagi dengan nol.

10.4.5 Kesimpulan

Proyek kalkulator sederhana ini merupakan contoh aplikasi pemrograman yang efektif untuk memahami dasar-dasar pengolahan input, percabangan, serta penanganan kesalahan. Kalkulator adalah salah satu alat yang sering digunakan dalam kehidupan sehari-hari untuk melakukan operasi matematika dasar seperti penjumlahan, pengurangan, perkalian, dan pembagian. Dengan mengembangkan proyek ini, pengguna tidak hanya belajar bagaimana cara menerima input dari pengguna dan mengolahnya, tetapi juga memahami bagaimana logika percabangan bekerja untuk menentukan operasi yang sesuai dengan perintah pengguna.

Selain itu, proyek ini memberikan kesempatan bagi pengguna untuk mengenal konsep penanganan kesalahan dalam pemrograman. Misalnya, ketika pengguna memasukkan nilai yang bukan angka atau mencoba melakukan pembagian dengan nol, program harus mampu menangani situasi ini dengan baik agar tidak menyebabkan kesalahan fatal yang dapat menghentikan jalannya program. Dengan menerapkan teknik seperti pengecekan tipe data dan penanganan exception, pengguna dapat membangun aplikasi yang lebih andal dan ramah pengguna.

Selain itu, pengembangan antarmuka grafis dapat meningkatkan pengalaman pengguna dengan membuat kalkulator lebih mudah digunakan dibandingkan versi berbasis teks. Dengan

menggunakan pustaka pemrograman seperti Tkinter, PyQt, atau Kivy, kalkulator dapat memiliki tampilan tombol interaktif, layar hasil yang lebih jelas, dan tata letak yang lebih menarik. Ini tidak hanya meningkatkan keterbacaan hasil perhitungan, tetapi juga membuat aplikasi lebih intuitif bagi pengguna yang kurang familiar dengan program berbasis terminal.

Di sisi lain, kalkulator juga dapat dikembangkan agar dapat berintegrasi dengan sistem lain, seperti kalkulator keuangan atau kalkulator konversi satuan. Misalnya, kalkulator dapat digunakan untuk menghitung bunga pinjaman, mengkonversi mata uang secara real-time dengan mengambil data dari API keuangan, atau membantu dalam perhitungan statistik. Dengan menambahkan konektivitas ke database atau layanan cloud, hasil perhitungan dapat disimpan dan diakses kembali, sehingga meningkatkan fungsionalitas aplikasi secara signifikan.

Dengan memahami konsep-konsep dasar dalam proyek kalkulator sederhana ini, pengguna memiliki dasar yang kuat untuk mengembangkan aplikasi yang lebih besar dan lebih canggih di masa mendatang. Kemampuan untuk mengolah input, menerapkan logika percabangan, menangani kesalahan, serta mengembangkan fitur tambahan adalah keterampilan penting dalam dunia pemrograman. Oleh karena itu, proyek ini tidak hanya memberikan wawasan tentang cara kerja sebuah kalkulator, tetapi juga membuka peluang untuk eksplorasi lebih lanjut dalam pengembangan perangkat lunak.

## 10.5 Latihan Soal

1. Apa perbedaan peran antara "latihan" dan "proyek mini" dalam pembelajaran pemrograman?
2. Sebutkan manfaat utama dari "latihan" dalam pembelajaran pemrograman!
3. Apa contoh proyek mini yang disebutkan dalam pembahasan di atas?

# Profile Penulis

Lahir di Gunung Kidul pada 9 Maret 1985, Ir. Agus Siswoyo S.T., M.T., saat ini berdomisili di Sosrodipuran GT.I/368, RT 19 RW 04, Sosromenduran, Kecamatan Gedongtengen, Kota Yogyakarta, Yogyakarta. Di tengah kesibukannya, ia gemar *traveling*, sebuah hobi yang mungkin memberinya perspektif baru dan inspirasi yang bisa diterapkan dalam karyanya.

Melalui buku ini, Agus Siswoyo ingin membawa pembaca memahami betapa mekatronika telah menjadi pilar utama dalam revolusi industri modern, khususnya di era Industri 4.0. Ia menyoroti dominasi otomasi, robotika, dan sistem cerdas di berbagai sektor industri. Ia berharap buku ini tidak hanya membantu pembaca memahami konsep dasar mekatronika, tetapi juga membimbing mereka dalam merancang dan menerapkan sistem mekatronika untuk solusi praktis permasalahan nyata.

Buku ini dirancang untuk berbagai kalangan, mulai dari mahasiswa, dosen, praktisi industri, hingga siapa saja yang tertarik belajar mekatronika. Agus Siswoyo menyadari bahwa mekatronika adalah bidang yang luas dan terus berkembang. Oleh karena itu, buku ini tidak hanya menyajikan teori dasar, tetapi juga aplikasi nyata yang relevan dengan perkembangan teknologi saat ini. Dengan rendah hati, ia juga sangat terbuka terhadap masukan, saran, atau kritik yang membangun dari para pembaca demi pengembangan lebih lanjut. Ia berharap buku ini dapat menjadi referensi yang bermanfaat dan menginspirasi Anda untuk terus belajar, berinovasi, dan berkontribusi dalam dunia teknologi mekatronika.

Lahir di Klaten pada 1 November 1986, Ir. Antonius Hendro Noviyanto, S.T., M.T., saat ini berdomisili di Gadingan RT 1 RW 4, Trunuh, Klaten Selatan. Di sela-sela kesibukannya, ia memiliki hobi *traveling*, yang mungkin memberinya perspektif segar dan ide-ide baru yang dituangkan dalam karyanya.

Melalui buku ini, Antonius Hendro Noviyanto mengajak pembaca dalam perjalanan mengeksplorasi dunia mekatronika. Ia menjelaskan bahwa bidang ini menggabungkan keahlian teknik mesin, elektronika, informatika, dan kontrol untuk menciptakan solusi inovatif di era modern. Baginya, mekatronika bukan hanya tentang teori, tetapi juga tentang penerapan nyata. Ia berharap pembaca akan diajak untuk berpikir kreatif, memecahkan masalah, dan mengembangkan sistem yang dapat mengubah dunia.

Ia berpesan agar pembaca tidak ragu mengeksplorasi setiap bab dengan penuh rasa ingin tahu, karena setiap halaman menyimpan pengetahuan yang dapat menginspirasi ide-ide brilian. Selamat membaca, semoga buku ini menjadi teman setia dalam perjalanan Anda memahami dan menguasai mekatronika. Mari

bersama-sama membangun masa depan yang lebih cerdas dan efisien melalui teknologi!

Lahir di Wonogiri pada 2 Februari 1987, Ir. Eko Arianto, S.T., M.T., saat ini berdomisili di Jongke Tengah, RT 04 RW 23, Sendangadi, Mlati, Sleman, Yogyakarta.

Melalui buku ini, Eko Arianto ingin menegaskan bahwa teknologi mekatronika adalah bidang yang terus berkembang dan memiliki potensi besar untuk mengubah dunia. Ia berharap buku ini dapat menjadi panduan yang bermanfaat bagi siapa pun, baik sebagai mahasiswa, profesional, atau individu yang tertarik untuk memahami dasar-dasar mekatronika. Pesan utamanya adalah untuk tidak pernah berhenti belajar dan mengeksplorasi, karena setiap langkah kecil hari ini bisa menjadi terobosan besar di masa depan. Eko Arianto berharap karyanya ini dapat menginspirasi pembaca untuk terus berinovasi dan berkontribusi dalam kemajuan teknologi.

# Daftar Pustaka

C++ Reference. (2023). Control Flow and Branching.

C++ Reference. (2023). Data Types and Variable Declaration.

C++ Reference. (2023). Error Handling and Debugging Tools.

C++ Reference. (2023). Flow Control and Branching.

C++ Reference. (2023). Functions and Code Reusability.

C++ Reference. (2023). Lists and Arrays.

C++ Reference. (2023). Maps and Unordered Sets.

C++ Reference. (2023). Operator Precedence and Associativity.

C++ Reference. (2023). Small Programming Projects.

GeeksforGeeks. (2023). Exception Handling in Python. Retrieved from https://www.geeksforgeeks.org/exception-handling-python/

GeeksforGeeks. (2023). Python Programming Language. Retrieved from https://www.geeksforgeeks.org/python-programming-language/

Java Documentation. (2023). Collections and Data Structures.

Java Documentation. (2023). Debugging and Error Handling.

Java Documentation. (2023). Loops and Decision Making.

Java Documentation. (2023). Methods and Modular Programming.

Java Documentation. (2023). Operators and Expressions.

Java Documentation. (2023). Project-Based Learning.

Java Documentation. (2023). Variable and Data Types.

Mulai Belajar Python. (n.d.). Retrieved from https://belajarpython.com/

Python Software Foundation. (2023). Control Flow Statements.

Python Software Foundation. (2023). Dictionaries and Sets Documentation.

Python Software Foundation. (2023). Exception Handling in Python.

Python Software Foundation. (2023). Functions and Modules.

Python Software Foundation. (2023). Lists and Tuples Documentation.

Python Software Foundation. (2023). Python Data Types Documentation.

Python Software Foundation. (2023). Python Official Documentation. https://docs.python.org

Python Software Foundation. (2023). Python Operators Documentation.

Python Software Foundation. (2023). Python Programming Guide.

Real Python. (2023). Python Debugging with pdb. Retrieved from https://realpython.com/python-debugging-pdb/

Stack Overflow. (2023). Common Python Errors and Solutions. Retrieved from https://stackoverflow.com/questions/tagged/python

Stack Overflow. (2023). Common Python Exceptions and How to Handle Them. Retrieved from https://stackoverflow.com/questions/tagged/exception-handling

Sweigart, A. (2019). *Automate the Boring Stuff with Python*. No Starch Press.

Van Rossum, G. (1991). Python Programming Language.

W3Schools. (2023). Python Try/Except. Retrieved from https://www.w3schools.com/python/python_try_except.asp

W3Schools. (2023). Python Tutorial. Retrieved from https://www.w3schools.com/python/

PENERBIT BUKULOKA

Buku ajar berjudul Dasar-Dasar Pemrograman Python merupakan panduan pemula yang menyajikan konsep-konsep inti pemrograman dengan pendekatan yang ringan, sistematis, dan mudah dipahami. Pembaca akan dikenalkan pada sintaks dasar Python, variabel, operator, struktur kontrol, fungsi, hingga pengantar pemrograman berorientasi objek.

Cocok bagi pembaca umum yang ingin memahami dasar teknis untuk mulai membangun program sederhana serta memahami logika di balik pemrograman. Dasar-Dasar Pemrograman Python adalah langkah awal yang tepat untuk terjun ke dunia teknologi dan pengembangan perangkat lunak di era digital.