MEMULAI PYTHON

Belajar Python dari Nol

IRFAN ARDIANSAH
RYAN HARA PERMANA

MEMULAI PYTHON

Belajar Python dari Nol

IRFAN ARDIANSAH
RYAN HARA PERMANA

Penerbit **Cendekia Press** - Bandung

Judul:	MEMULAI PYTHON Belajar Python dari Nol
Penulis:	Irfan Ardiansah Ryan Hara Permana
Editor:	Nabilla Septaryana
Desain Sampul:	Nabilla Septaryana
Desain Layout:	Irfan Ardiansah

ISBN: 978-623-5466-44-6
Cetakan Pertama, Juli 2023

Diterbitkan oleh
CV. CENDEKIA PRESS
NIB: 8120107982776
Komp. GBA Barat Blok C – 4 No. 7 Bandung
Email: penerbit@cendekiapress.com
Website: www.cendekiaoress.com
Anggota IKAPI No 328/JBA/2018

Sanksi Pelanggaran Pasal 72

1) Barang siapa dengan sengaja dan tanpa hak melakukan perbuatan sebagaimanadimaksud dalam Pasal 2 ayat (1) dan ayat (2) dipidana dengan Pidana penjaramasing-masing paling singkat 1 (satu) bulan dan atau paling sedikit Rp. 1.000.000,-(satu juta rupiah) atau pidana penjara paling lama 7 (tujuh) tahun atau dendapaling banyak Rp. 5.000.000.000,- (lima milyar rupiah).
2) Barang siapa dengan sengaja menyiarkan memamerkan, mengedarkan, ataumenjual kepada umum atau Ciptaan atau barang hasil pelanggaran Hak Cipta atauHak Terkait sebagaimana dimaksud pada ayat (1) dipidana dengan pidana penjarapaling lama 5 (lima) tahun dan/ atau denda paling banyak Rp. 500.000.000,- (limaratus juta rupiah)

PRAKATA

Python telah menjadi bahasa pemrograman yang populer dan banyak digunakan di berbagai bidang, mulai dari pengembangan aplikasi web, data analisis, kecerdasan buatan, hingga otomasi. Kemudahan dalam belajar dan penggunaan Python membuatnya menjadi pilihan utama bagi pemula yang ingin memulai perjalanan mereka di dunia pemrograman.

Buku "Memulai Python: Belajar Python dari Nol" ini hadir sebagai panduan bagi Anda yang ingin mempelajari dasar-dasar pemrograman Python. Buku ini mencakup berbagai topik penting yang perlu Anda ketahui saat memulai belajar Python, mulai dari pengenalan bahasa, tipe data, percabangan, perulangan, hingga struktur data dan fungsi.

Dalam setiap bab, kami berusaha memberikan penjelasan yang mudah dipahami dan menyertakan contoh kode yang relevan untuk membantu Anda memahami konsep yang dijelaskan dengan menyertakan latihan dan contoh aplikasi untuk membantu Anda mengasah keterampilan Anda dalam Python. Pembaca diharapkan untuk secara aktif mencoba dan

mempraktikkan kode yang diberikan, serta mengeksplorasi konsep-konsep yang dijelaskan lebih lanjut.

Kami mengajak Anda untuk terus belajar dan menggali lebih dalam tentang Python dan berbagai kemungkinan yang dapat dijangkau dengan bahasa pemrograman ini. Seperti pepatah "practice makes perfect", semakin sering Anda berlatih dan mengeksplorasi, semakin mahir Anda dalam menggunakan Python.

Kami berterima kasih kepada semua pihak yang telah mendukung pembuatan buku ini, dan kami sangat menghargai setiap umpan balik, saran, dan kritik yang konstruktif dari para pembaca. Semoga buku "Memulai Python: Belajar Python dari Nol" ini menjadi awal yang baik dalam perjalanan Anda untuk menjadi seorang programmer Python yang handal.

Selamat belajar, dan semoga sukses!

Bandung, Juli 2023

Penulis

DAFTAR ISI

BAB I
PENGANTAR PYTHON

1.1. Mengenal Python

Python adalah bahasa pemrograman yang populer dan banyak digunakan di berbagai bidang, mulai dari pengembangan web, analisis data, kecerdasan buatan, hingga pengembangan perangkat lunak umum. Python diciptakan oleh Guido van Rossum dan pertama kali dirilis pada tahun 1991. Python dikenal dengan sintaks yang mudah dibaca dan dipahami, serta dukungan komunitas yang luas.

1. Sejarah dan Evolusi Python

Python pertama kali dikembangkan oleh Guido van Rossum pada akhir tahun 1980-an di Centrum Wiskunde & Informatica (CWI) di Belanda. Nama Python sendiri diambil dari acara televisi favorit Van Rossum saat itu, yaitu "Monty Python's Flying Circus".

Berikut adalah beberapa momen penting dalam sejarah Python:

- Python 1.0: Dirilis pada Januari 1994, versi ini menampilkan fitur dasar Python yang masih

digunakan hingga saat ini, seperti penanganan pengecualian dan dukungan untuk fungsi dan modul.

- Python 2.0: Dirilis pada Oktober 2000, versi ini menghadirkan fitur-fitur penting seperti dukungan Unicode dan garbage collector yang lebih baik.
- Python 3.0: Dirilis pada Desember 2008, versi ini dikenal sebagai versi yang "memecah" kompatibilitas dengan versi sebelumnya, karena beberapa perubahan pada sintaks dan fitur. Namun, Python 3 dirancang untuk menjadi lebih bersih dan lebih konsisten daripada versi sebelumnya.

2. Mengapa Python Populer?

Python menjadi populer karena beberapa alasan:

- Mudah dipelajari: Python memiliki sintaks yang sederhana dan mudah dibaca, sehingga cocok bagi pemula yang baru memulai belajar pemrograman.
- Portabilitas: Python tersedia di berbagai platform, seperti Windows, macOS, dan Linux. Hal ini memudahkan pengguna untuk menjalankan kode Python di lingkungan yang berbeda.
- Komunitas yang besar: Python memiliki komunitas yang besar dan aktif, yang terus berkontribusi untuk

mengembangkan bahasa ini serta pustaka yang mendukungnya.

- Pustaka yang kaya: Python memiliki pustaka standar yang sangat lengkap, serta banyak pustaka eksternal yang dikembangkan oleh komunitas. Hal ini memudahkan pengguna Python untuk menemukan solusi bagi berbagai masalah yang mereka hadapi.
- Fleksibilitas: Python dapat digunakan untuk berbagai keperluan, seperti pengembangan web, analisis data, kecerdasan buatan, automasi, dan sebagainya.
- Python terus berkembang dan menjadi pilihan utama bagi banyak pengembang, serta perusahaan di seluruh dunia, karena kelebihan-kelebihan yang dimilikinya.

1.2. Keunggulan Python

Python menjadi salah satu bahasa pemrograman yang paling populer karena memiliki berbagai keunggulan dibandingkan dengan bahasa pemrograman lain. Berikut ini beberapa keunggulan Python yang menjadikannya pilihan utama bagi banyak pengembang dan perusahaan:

- Sintaks yang sederhana: Python memiliki sintaks yang sederhana dan mudah dibaca, yang memungkinkan pemula untuk lebih cepat memahami konsep pemrograman. Selain itu, sintaks yang sederhana ini juga memudahkan pengembang dalam menulis dan memelihara kode.
- Portabilitas dan interoperabilitas: Python tersedia di berbagai platform seperti Windows, macOS, dan Linux. Kode yang ditulis dalam Python bisa dengan mudah dijalankan di platform yang berbeda tanpa perlu modifikasi yang signifikan. Selain itu, Python juga bisa berinteraksi dengan kode yang ditulis dalam bahasa pemrograman lain, seperti C, C++, dan Java.
- Dukungan pustaka yang kaya: Python memiliki pustaka standar yang sangat lengkap, serta ribuan pustaka eksternal yang dikembangkan oleh komunitas. Hal ini memudahkan pengguna Python untuk menemukan solusi bagi berbagai masalah yang mereka hadapi, tanpa perlu menulis kode dari awal.
- Komunitas yang besar dan aktif: Python didukung oleh komunitas yang besar dan aktif, yang terus berkontribusi untuk mengembangkan bahasa ini serta pustaka yang mendukungnya. Dukungan

komunitas ini membantu pengguna Python dalam memecahkan masalah, serta memperoleh sumber daya dan tutorial yang berkualitas.

- Fleksibilitas: Python dapat digunakan untuk berbagai keperluan, seperti pengembangan web, analisis data, kecerdasan buatan, automasi, dan sebagainya. Hal ini membuat Python menjadi bahasa pemrograman yang fleksibel dan dapat disesuaikan dengan kebutuhan yang beragam.
- Skalabilitas: Python dapat digunakan untuk mengembangkan proyek kecil hingga proyek berskala besar, berkat dukungan modul dan paket yang memungkinkan pengembang untuk mengorganisir kode dengan lebih baik. Selain itu, Python juga mendukung pemrograman multiproses dan multithreading, yang memungkinkan pengguna untuk memanfaatkan sumber daya komputasi secara efisien.
- Pemrograman berorientasi objek: Python mendukung konsep pemrograman berorientasi objek (OOP), yang memudahkan pengembang dalam merancang dan mengelola kode yang kompleks. Dengan OOP, pengembang bisa membangun aplikasi yang modular dan mudah diperluas.

Keunggulan-keunggulan Python ini menjadikannya pilihan yang menarik bagi pengembang yang ingin belajar bahasa pemrograman baru, serta perusahaan yang mencari solusi pemrograman yang efisien dan mudah dikelola. Python terus tumbuh dan berkembang, sehingga memberikan manfaat lebih bagi pengguna yang memilih untuk menggunakan bahasa ini.

1.3. Instalasi dan Persiapan

Sebelum kita mulai mempelajari dasar-dasar Python dan menulis kode, penting bagi kita untuk mempersiapkan lingkungan pemrograman yang sesuai. Dalam bab ini, kita akan membahas cara menginstal Python di berbagai sistem operasi, serta mengatur lingkungan pemrograman yang optimal untuk memulai perjalanan kita dalam mempelajari Python.

Kami akan membahas langkah-langkah yang diperlukan untuk menginstal Python di Windows, macOS, dan Linux, serta memberikan rekomendasi mengenai alat pengembangan dan editor kode yang cocok untuk bekerja dengan Python. Selain itu, kita juga akan mempelajari cara menginstal pustaka eksternal yang mungkin

diperlukan seiring perkembangan kita dalam mempelajari Python.

Mari kita mulai dengan menginstal Python dan mengatur lingkungan pemrograman yang akan menjadi fondasi bagi perjalanan kita dalam mempelajari bahasa pemrograman yang menarik ini.

1. Instalasi Python di Windows

Dalam subbab ini, kita akan belajar cara mengunduh dan menginstal Python di sistem operasi Windows. Ikuti langkah-langkah berikut untuk menginstal Python dengan benar, termasuk konfigurasi PATH Environment dan pustaka TCL/TK.

Langkah 1: Mengunduh Python

1. Buka browser Anda dan kunjungi situs web resmi Python di https://www.python.org/.
2. Klik tombol "Downloads" di bagian atas halaman.
3. Di bawah judul "Python for Windows", klik tombol "Download Python x.x.x" (x.x.x adalah versi terbaru saat ini).
4. File instalasi Python akan diunduh ke komputer Anda.

Langkah 2: Menginstal Python

5. Buka folder tempat Anda menyimpan file instalasi Python yang baru saja diunduh.

6. Klik kanan pada file tersebut dan pilih "Run as administrator".
7. Jendela "Python Setup" akan muncul. Pada jendela ini, pastikan untuk mencentang opsi "Add Python x.x to PATH" di bagian bawah. Opsi ini akan menambahkan Python ke PATH Environment sehingga Anda dapat menjalankan Python dari Command Prompt atau terminal lainnya.
8. Klik tombol "Install Now" untuk memulai proses instalasi. Anda juga dapat memilih "Customize installation" jika ingin mengubah pengaturan default, seperti lokasi instalasi atau komponen yang diinstal.

2. **Mengenai PATH Environment**

Menambahkan Python ke PATH Environment sangat penting karena ini memungkinkan Anda untuk menjalankan Python dari mana saja di sistem Anda melalui Command Prompt atau terminal lainnya. Jika Python tidak ditambahkan ke PATH, Anda harus menavigasi ke direktori tempat Python diinstal setiap kali ingin menjalankannya.

3. **Mengenai Pustaka TCL/TK**

Pustaka TCL/TK digunakan oleh Python untuk membuat antarmuka grafis (GUI) menggunakan modul

tkinter. Ketika menginstal Python, pustaka TCL/TK akan diinstal secara default. Jika Anda berencana untuk membuat aplikasi dengan antarmuka grafis, pastikan untuk tidak melewatkan instalasi pustaka TCL/TK saat menginstal Python.

Setelah proses instalasi selesai, Python siap digunakan di sistem Windows Anda. Sekarang Anda dapat mulai mempelajari dasar-dasar Python dan menulis kode menggunakan lingkungan pemrograman yang telah Anda siapkan.

4. Persiapan Lingkungan Pemrograman

Dalam subbab ini, kita akan membahas cara memilih IDE (Integrated Development Environment) terbaik untuk Python, di mana mengunduhnya, bagaimana menginstal dan menghubungkan IDE dengan Python, serta bagaimana menguji apakah IDE berfungsi dengan baik.

5. Memilih IDE Terbaik untuk Python

Beberapa IDE populer untuk Python adalah:

- Visual Studio Code (VSCode)
- PyCharm
- Jupyter Notebook
- Atom

- Sublime Text

Untuk pemula, kami merekomendasikan Visual Studio Code atau PyCharm Community Edition. Kedua IDE ini gratis, memiliki dukungan komunitas yang luas, dan menyediakan fitur-fitur yang membantu pengembangan Python, seperti linting, autocompletion, dan debugging.

6. Mengunduh dan Menginstal IDE

Dalam tutorial ini, kita akan menggunakan Visual Studio Code sebagai contoh:

- Buka browser Anda dan kunjungi situs web resmi Visual Studio Code di https://code.visualstudio.com/.
- Klik tombol "Download for Windows" (atau pilih sistem operasi yang sesuai) untuk mengunduh installer VSCode.
- Setelah selesai mengunduh, jalankan installer dan ikuti langkah-langkah yang disajikan untuk menginstal VSCode.
- Menghubungkan IDE dengan Python

Setelah menginstal IDE, kita perlu menghubungkannya dengan Python. Di Visual Studio Code, kita perlu menginstal ekstensi Python:

- Buka Visual Studio Code.
- Klik ikon "Extensions" di bilah samping sebelah kiri (atau tekan Ctrl + Shift + X).
- Di kolom pencarian, ketik "Python" dan tekan Enter.
- Cari ekstensi "Python" yang dikembangkan oleh Microsoft, lalu klik tombol "Install" di bawahnya.

7. **Menguji IDE**

Setelah menginstal ekstensi Python, kita perlu memastikan bahwa IDE berfungsi dengan baik dengan Python. Untuk mengujinya, kita akan membuat sebuah file Python sederhana dan menjalankannya:

- Buka Visual Studio Code.
- Klik "File" > "New File" untuk membuat file baru.
- Klik "File" > "Save As" untuk menyimpan file baru. Beri nama file "test.py" dan simpan di lokasi yang Anda inginkan.
- Ketik kode berikut di dalam file "test.py":

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

Klik kanan di area teks file, lalu pilih "Run Python File in Terminal". Anda juga dapat menggunakan pintasan

Ctrl + Alt + N jika sudah menginstal ekstensi "Code Runner".

Jika semuanya berjalan dengan baik, Anda akan melihat pesan "Hello, Python!" muncul di terminal. Ini berarti Anda telah berhasil mengatur lingkungan pemrograman Python dengan IDE yang Anda pilih. Sekarang Anda siap untuk mulai menulis kode Python menggunakan IDE Anda dan mempelajari lebih lanjut tentang bahasa pemrograman ini.

BAB II
DASAR-DASAR PYTHON

Setelah berhasil menginstal dan mengatur lingkungan pemrograman Python, saatnya untuk memulai perjalanan kita dalam mempelajari dasar-dasar bahasa pemrograman Python. Bab ini akan membahas konsep-konsep fundamental yang perlu Anda pahami sebelum melanjutkan ke topik yang lebih kompleks.

Kita akan memulai dengan mempelajari sintaks dasar Python, termasuk cara menulis dan menjalankan program sederhana. Selanjutnya, kita akan mengeksplorasi berbagai tipe data dan struktur data yang umum digunakan dalam Python, serta cara menggunakannya untuk memecahkan masalah sehari-hari.

Kemudian, kita akan membahas kontrol aliran dalam Python, seperti percabangan dan perulangan, yang akan membantu kita dalam menulis kode yang lebih fleksibel dan dinamis. Selain itu, kita juga akan mempelajari konsep fungsi, yang memungkinkan kita untuk mengorganisir kode dengan lebih efisien dan mengurangi redundansi.

Dengan memahami konsep-konsep dasar yang dibahas dalam bab ini, Anda akan memiliki fondasi yang kuat untuk mempelajari topik yang lebih maju dan mengembangkan aplikasi Python yang lebih kompleks. Mari kita mulai dengan mempelajari dasar-dasar Python dan membuka pintu menuju dunia pemrograman yang lebih luas!

2.1. Variabel dan Tipe Data

Variabel dan tipe data merupakan konsep fundamental dalam bahasa pemrograman, termasuk Python. Dalam subbab ini, kita akan mempelajari apa itu variabel, bagaimana cara mendeklarasikan dan menggunakannya dalam Python, serta berbagai tipe data yang umum digunakan.

Memahami variabel dan tipe data adalah langkah penting untuk mengorganisir dan menyimpan informasi dalam program Anda. Kita akan membahas berbagai tipe data dasar seperti bilangan bulat, bilangan pecahan, string, dan boolean, serta cara mengoperasikannya.

Dengan mengeksplorasi konsep ini, Anda akan memiliki pemahaman yang lebih baik tentang bagaimana Python menyimpan dan mengelola data, serta bagaimana

menggunakannya untuk memecahkan masalah dalam pengembangan aplikasi Anda. Mari kita mulai dengan mempelajari tentang variabel dan berbagai tipe data yang tersedia dalam Python.

1. Variabel

Variabel adalah tempat penyimpanan di dalam memori yang digunakan untuk menyimpan nilai. Dalam bahasa pemrograman, seperti Python, variabel digunakan untuk menyimpan data yang diperlukan oleh program. Setiap variabel memiliki nama yang unik yang mengidentifikasikannya, dan Anda dapat mengakses, memodifikasi, atau menghapus nilai yang disimpan dalam variabel dengan merujuk ke namanya.

2. Mendeklarasikan dan Menggunakan Variabel dalam Python

Dalam Python, Anda tidak perlu mendeklarasikan tipe data variabel secara eksplisit. Python akan secara otomatis menentukan tipe data berdasarkan nilai yang Anda berikan saat membuat variabel. Anda dapat mendeklarasikan variabel dengan menggunakan tanda sama dengan (=) untuk mengaitkan nama variabel dengan nilai yang ingin Anda simpan.

Berikut adalah cara membuat dan menggunakan variabel dalam Python:

```
# Mendeklarasikan variabel
nama = "John Doe"
umur = 25
tinggi = 175.5
apakah_pelajar = True

# Menggunakan variabel
print("Nama:", nama)
print("Umur:", umur)
print("Tinggi:", tinggi)
print("Apakah pelajar?", apakah_pelajar)
```

Dalam contoh di atas, kita mendeklarasikan empat variabel dengan nama nama, umur, tinggi, dan apakah_pelajar. Kemudian, kita menggunakan fungsi print() untuk menampilkan nilai dari masing-masing variabel.

3. Aturan Penamaan Variabel

Saat menamai variabel, ada beberapa aturan yang harus diikuti:

- Nama variabel harus dimulai dengan huruf atau karakter garis bawah (_). Angka tidak diperbolehkan sebagai karakter pertama.
- Nama variabel hanya boleh mengandung huruf, angka, dan karakter garis bawah (_).

- Nama variabel tidak boleh menggunakan kata kunci yang telah didefinisikan oleh Python, seperti if, else, while, dll.

Contoh nama variabel yang valid:

```
nama_lengkap = "John Doe"
_umur = 25
tinggi_badan = 175.5
```

Contoh nama variabel yang tidak valid:

```
1nama = "John Doe"  # Dimulai dengan angka
umur-pelajar = 25   # Menggunakan karakter
ilegal (-)
if = "condition"    # Menggunakan kata kunci
yang didefinisikan oleh Python
```

Sekarang Anda telah memahami apa itu variabel, bagaimana cara mendeklarasikan dan menggunakannya dalam Python, serta aturan penamaan variabel. Selanjutnya, kita akan membahas berbagai tipe data yang tersedia dalam Python dan bagaimana menggunakannya.

4. Tipe Data

Tipe data adalah klasifikasi data yang menentukan jenis nilai yang dapat disimpan dalam suatu variabel. Setiap bahasa pemrograman memiliki berbagai tipe data yang digunakan untuk menyimpan data dalam berbagai

bentuk, seperti angka, teks, atau nilai boolean. Dalam Python, beberapa tipe data dasar yang umum digunakan adalah:

- Integer (Bilangan Bulat)
- Float (Bilangan Pecahan)
- String (Teks)
- Boolean (Nilai Benar/Salah)

5. Menggunakan Tipe Data dalam Python

Berikut adalah cara menggunakan berbagai tipe data dalam Python:

a. Integer (Bilangan Bulat)

Integer adalah tipe data yang digunakan untuk menyimpan bilangan bulat. Anda dapat mendeklarasikan variabel integer dengan langsung memberikan nilai bilangan bulat ke variabel.

```
umur = 25
harga = 10000

print("Umur:", umur)
print("Harga:", harga)
```

b. Float (Bilangan Pecahan)

Float adalah tipe data yang digunakan untuk menyimpan bilangan pecahan atau desimal. Anda dapat

mendeklarasikan variabel float dengan memberikan nilai desimal ke variabel atau menggunakan tanda . setelah bilangan bulat.

```
tinggi = 175.5
pi = 3.14

print("Tinggi:", tinggi)
print("Nilai Pi:", pi)
```

c. String (Teks)

String adalah tipe data yang digunakan untuk menyimpan teks atau karakter. Anda dapat mendeklarasikan variabel string dengan menggunakan tanda kutip tunggal (') atau kutip ganda (").

```
nama = "John Doe"
alamat = 'Jl. Kebon Jeruk No. 10'

print("Nama:", nama)
print("Alamat:", alamat)
```

d. Boolean (Nilai Benar/Salah)

Boolean adalah tipe data yang digunakan untuk menyimpan nilai benar (True) atau salah (False). Tipe data ini sangat berguna saat menangani logika dan kondisi dalam program.

```
apakah_pelajar = True
```

```
apakah_karyawan = False

print("Apakah pelajar?", apakah_pelajar)
print("Apakah karyawan?", apakah_karyawan)
```

Dengan memahami berbagai tipe data dasar dalam Python, Anda dapat menyimpan dan mengelola data dalam berbagai bentuk, sesuai dengan kebutuhan aplikasi Anda. Selanjutnya, kita akan membahas tentang operasi yang dapat dilakukan pada tipe data ini dan bagaimana mengkonversi antara tipe data yang berbeda.

2.2. Operasi Aritmatika dan Operator

Dalam banyak program, Anda akan sering menemui kebutuhan untuk melakukan operasi aritmatika, seperti penjumlahan, pengurangan, perkalian, dan pembagian. Python menyediakan berbagai operator aritmatika yang memungkinkan Anda untuk menghitung nilai-nilai numerik dengan mudah dan efisien.

Subbab ini akan memperkenalkan operator aritmatika dasar yang tersedia dalam Python dan cara menggunakannya dalam program Anda. Selain itu, kita juga akan membahas beberapa operator lain yang dapat digunakan untuk memanipulasi tipe data dan

mengendalikan aliran program, seperti operator perbandingan dan logika.

Dengan memahami operasi aritmatika dan operator dalam Python, Anda akan memiliki alat yang diperlukan untuk menulis kode yang lebih kompleks dan menangani berbagai situasi yang mungkin muncul dalam pengembangan aplikasi Anda. Mari kita mulai dengan mempelajari operasi aritmatika dan operator yang tersedia dalam Python.

1. Operasi Aritmatika

Operasi aritmatika adalah operasi matematika dasar yang digunakan untuk menghitung nilai numerik. Dalam Python, Anda dapat melakukan operasi aritmatika seperti penjumlahan, pengurangan, perkalian, pembagian, dan lainnya dengan menggunakan operator aritmatika.

Berikut adalah operator aritmatika dasar dalam Python:

- Penjumlahan (+)
- Pengurangan (-)
- Perkalian (*)
- Pembagian (/)
- Sisa bagi (modulo) (%)

- Eksponen (pangkat) (**)
- Pembagian bulat (//)

2. Menggunakan Operasi Aritmatika dalam Python

Berikut adalah cara menggunakan operator aritmatika dalam Python:

```
a = 10
b = 3

# Penjumlahan
jumlah = a + b
print("Penjumlahan:", jumlah)

# Pengurangan
kurang = a - b
print("Pengurangan:", kurang)

# Perkalian
kali = a * b
print("Perkalian:", kali)

# Pembagian
bagi = a / b
print("Pembagian:", bagi)

# Sisa bagi (modulo)
modulo = a % b
print("Modulo:", modulo)

# Eksponen (pangkat)
pangkat = a ** b
print("Pangkat:", pangkat)

# Pembagian bulat
pembagian_bulat = a // b
print("Pembagian Bulat:", pembagian_bulat)
```

Dalam contoh di atas, kita mendeklarasikan dua variabel a dan b dengan nilai 10 dan 3. Kemudian, kita menggunakan operator aritmatika untuk menghitung hasil berbagai operasi aritmatika dan menampilkannya menggunakan fungsi print().

Dengan memahami dan menguasai operasi aritmatika dalam Python, Anda akan dapat menulis kode yang lebih kompleks dan mengatasi berbagai masalah matematika yang mungkin muncul dalam pengembangan aplikasi Anda. Selanjutnya, kita akan membahas operator perbandingan dan logika yang digunakan untuk mengendalikan aliran program.

3. **Operator**

Selain operasi aritmatika, Python juga menyediakan berbagai operator lain yang dapat digunakan untuk memanipulasi data dan mengendalikan aliran program. Dalam subbab ini, kita akan membahas beberapa operator yang umum digunakan dalam Python, seperti operator perbandingan dan operator logika.

a. Operator Perbandingan

Operator perbandingan digunakan untuk membandingkan dua nilai dan mengembalikan nilai boolean (True atau False) berdasarkan hasil

perbandingan. Berikut adalah operator perbandingan yang tersedia dalam Python:

- Sama dengan (==)
- Tidak sama dengan (!=)
- Lebih besar dari (>)
- Lebih kecil dari (<)
- Lebih besar atau sama dengan (>=)
- Lebih kecil atau sama dengan (<=)

Berikut adalah cara menggunakan operator perbandingan dalam Python:

```
a = 10
b = 3

# Sama dengan
print("a sama dengan b?", a == b)

# Tidak sama dengan
print("a tidak sama dengan b?", a != b)

# Lebih besar dari
print("a lebih besar dari b?", a > b)

# Lebih kecil dari
print("a lebih kecil dari b?", a < b)

# Lebih besar atau sama dengan
print("a lebih besar atau sama dengan b?", a
>= b)

# Lebih kecil atau sama dengan
print("a lebih kecil atau sama dengan b?", a
<= b)
```

b. Operator Logika

Operator logika digunakan untuk menggabungkan atau memodifikasi ekspresi boolean. Ada tiga operator logika dasar dalam Python:

- and: Mengembalikan True jika kedua ekspresi benar
- or: Mengembalikan True jika salah satu ekspresi benar
- not: Mengembalikan kebalikan dari ekspresi boolean

Berikut adalah cara menggunakan operator logika dalam Python:

```
nilai = 85
hadir = 90

# Kondisi untuk lulus
lulus_nilai = nilai >= 75
lulus_hadir = hadir >= 80

# Menggunakan operator 'and'
lulus = lulus_nilai and lulus_hadir
print("Apakah siswa lulus?", lulus)

# Menggunakan operator 'or'
minimal_satu = lulus_nilai or lulus_hadir
print("Apakah siswa memenuhi minimal satu
kriteria?", minimal_satu)

# Menggunakan operator 'not'
tidak_lulus = not lulus
```

```
print("Apakah siswa tidak lulus?",
tidak_lulus)
```

Dengan memahami operator perbandingan dan operator logika, Anda dapat menulis kode yang lebih kompleks dan mengendalikan aliran program dengan lebih efisien. Ini akan membantu Anda dalam mengembangkan aplikasi yang lebih canggih dan memecahkan berbagai masalah yang mungkin dihadapi dalam pengembangan aplikasi Anda.

2.3. Percabangan (If, Else, Elif)

Percabangan adalah konsep fundamental dalam pemrograman yang memungkinkan Anda untuk mengendalikan aliran program berdasarkan kondisi tertentu. Dalam Python, Anda dapat menggunakan pernyataan if, else, dan elif untuk membuat percabangan dalam kode Anda. Dengan percabangan, Anda dapat membuat program yang lebih dinamis dan responsif terhadap berbagai situasi yang mungkin terjadi selama eksekusi program.

Subbab ini akan memperkenalkan cara menggunakan pernyataan if, else, dan elif dalam Python untuk mengendalikan aliran program berdasarkan

kondisi yang diberikan. Anda akan belajar bagaimana menulis kode yang bersyarat dan memahami bagaimana percabangan dapat membantu Anda dalam mengembangkan aplikasi yang lebih kompleks dan canggih. Mari kita mulai dengan mempelajari pernyataan if dalam Python.

1. Percabangan If

Percabangan if adalah struktur kontrol dasar yang digunakan untuk menjalankan blok kode berdasarkan kondisi tertentu. Jika kondisi dalam pernyataan if bernilai True, maka blok kode di dalam pernyataan tersebut akan dieksekusi. Sebaliknya, jika kondisinya bernilai False, blok kode tersebut akan diabaikan.

2. Menggunakan Percabangan If dalam Python

Berikut adalah cara menggunakan pernyataan if dalam Python:

```
umur = 20

if umur >= 17:
    print("Anda sudah cukup umur untuk
mengendarai mobil.")
```

Dalam contoh di atas, kita memiliki variabel umur dengan nilai 20. Kita menggunakan pernyataan if untuk

memeriksa apakah umur tersebut lebih besar atau sama dengan 17. Jika kondisi ini terpenuhi, maka blok kode di dalam pernyataan if akan dieksekusi, dan pesan "Anda sudah cukup umur untuk mengendarai mobil." akan ditampilkan.

Perhatikan bahwa blok kode di dalam pernyataan if diindentasi menggunakan spasi atau tab. Indentasi ini penting dalam Python, karena menentukan blok kode yang termasuk dalam pernyataan if.

a. Percabangan If Ganda

Percabangan if ganda digunakan ketika Anda ingin memeriksa beberapa kondisi secara terpisah. Dalam kasus ini, Anda dapat menggunakan beberapa pernyataan if yang berbeda untuk memeriksa setiap kondisi.

Berikut adalah contoh penggunaan percabangan if ganda dalam kode Python:

```
nilai = 75
hadir = 85

if nilai >= 60:
    print("Nilai Anda memenuhi syarat untuk
lulus.")

if hadir >= 80:
    print("Kehadiran Anda memenuhi syarat
untuk lulus.")
```

Dalam contoh di atas, kita memiliki variabel nilai dan hadir. Kita menggunakan dua pernyataan if yang berbeda untuk memeriksa apakah nilai dan kehadiran memenuhi syarat untuk lulus. Jika salah satu kondisi terpenuhi, blok kode di dalam pernyataan if yang bersangkutan akan dieksekusi.

b. Percabangan If Bertingkat

Percabangan if bertingkat digunakan ketika Anda ingin memeriksa beberapa kondisi dalam urutan tertentu. Dalam kasus ini, Anda dapat mengecek kondisi berikutnya menggunakan pernyataan if lain di dalam blok kode pernyataan if sebelumnya.

Berikut adalah contoh penggunaan percabangan if bertingkat dalam kode Python:

```
umur = 18

if umur >= 13:
    print("Anda sudah remaja.")

    if umur >= 17:
        print("Anda sudah cukup umur untuk
mengendarai mobil.")
```

Dalam contoh di atas, kita memiliki variabel umur dengan nilai 18. Kita menggunakan pernyataan if pertama untuk memeriksa apakah umur tersebut lebih besar atau sama dengan 13. Jika kondisi ini terpenuhi, kita menampilkan pesan "Anda sudah remaja." dan kemudian memeriksa kondisi berikutnya menggunakan pernyataan if kedua di dalam blok kode pernyataan if pertama. Jika umur juga lebih besar atau sama dengan 17, kita menampilkan pesan "Anda sudah cukup umur untuk mengendarai mobil."

Dengan memahami percabangan if ganda dan if bertingkat, Anda dapat menulis kode yang lebih kompleks dan mengendalikan aliran program dengan lebih efisien, tergantung pada kondisi yang diberikan.

Dengan memahami dan menguasai percabangan if, Anda dapat menulis kode yang lebih dinamis dan mengendalikan aliran program berdasarkan kondisi yang diberikan. Selanjutnya, kita akan membahas pernyataan else dan elif untuk membuat percabangan yang lebih kompleks.

3. Percabangan If - Else

Percabangan if - else adalah struktur kontrol yang digunakan untuk menjalankan satu blok kode jika kondisi

yang diberikan bernilai True, dan menjalankan blok kode lain jika kondisi tersebut bernilai False. Dengan percabangan if - else, Anda dapat membuat program yang lebih responsif dengan menentukan aksi yang berbeda tergantung pada hasil evaluasi kondisi.

4. Menggunakan Percabangan If - Else dalam Python

Berikut adalah cara menggunakan pernyataan if - else dalam Python:

```
umur = 15

if umur >= 17:
    print("Anda sudah cukup umur untuk
mengendarai mobil.")
else:
    print("Anda belum cukup umur untuk
mengendarai mobil.")
```

Dalam contoh di atas, kita memiliki variabel umur dengan nilai 15. Kita menggunakan pernyataan if untuk memeriksa apakah umur tersebut lebih besar atau sama dengan 17. Jika kondisi ini terpenuhi, maka blok kode di dalam pernyataan if akan dieksekusi, dan pesan "Anda sudah cukup umur untuk mengendarai mobil." akan ditampilkan. Jika kondisi ini tidak terpenuhi, maka blok kode di dalam pernyataan else akan dieksekusi, dan

pesan "Anda belum cukup umur untuk mengendarai mobil." akan ditampilkan.

Perhatikan bahwa blok kode di dalam pernyataan if dan else diindentasi menggunakan spasi atau tab. Indentasi ini penting dalam Python, karena menentukan blok kode yang termasuk dalam masing-masing pernyataan.

Berikut adalah contoh lain yang lebih menarik mengenai penggunaan percabangan if - else dalam situasi dunia nyata, dengan penjelasan yang komprehensif:

```
harga_baju = 150000
uang_dompet = 180000

if uang_dompet >= harga_baju:
    print("Anda bisa membeli baju ini.")
    sisa_uang = uang_dompet - harga_baju
    print("Sisa uang Anda adalah:", sisa_uang)
else:
    print("Maaf, uang Anda tidak cukup untuk
membeli baju ini.")
    kekurangan_uang = harga_baju - uang_dompet
    print("Anda kekurangan:", kekurangan_uang)
```

Penjelasan Baris per Baris:

- harga_baju = 150000: Menyimpan harga baju ke dalam variabel harga_baju.
- uang_dompet = 180000: Menyimpan jumlah uang dalam dompet ke dalam variabel uang_dompet.

- if uang_dompet >= harga_baju:: Memeriksa apakah jumlah uang dalam dompet lebih besar atau sama dengan harga baju.
- print("Anda bisa membeli baju ini."): Jika uang cukup, maka akan mencetak pesan bahwa Anda bisa membeli baju.
- sisa_uang = uang_dompet - harga_baju: Menghitung sisa uang setelah membeli baju.
- print("Sisa uang Anda adalah:", sisa_uang): Mencetak sisa uang yang Anda miliki setelah membeli baju.
- else:: Jika uang dalam dompet tidak cukup untuk membeli baju, maka blok kode berikutnya akan dieksekusi.
- print("Maaf, uang Anda tidak cukup untuk membeli baju ini."): Mencetak pesan bahwa uang Anda tidak cukup untuk membeli baju.
- kekurangan_uang = harga_baju - uang_dompet: Menghitung kekurangan uang yang Anda miliki untuk membeli baju.
- print("Anda kekurangan:", kekurangan_uang): Mencetak jumlah uang yang Anda kekurangan untuk membeli baju.

Dalam contoh ini, kita menggunakan percabangan if - else untuk menentukan apakah seseorang memiliki cukup uang untuk membeli baju. Jika uang cukup, program akan mencetak pesan bahwa baju dapat dibeli dan menampilkan sisa uang. Jika uang tidak cukup, program akan mencetak pesan bahwa uang tidak cukup dan menampilkan jumlah uang yang diperlukan untuk membeli baju.

Dengan memahami dan menguasai percabangan if - else, Anda dapat menulis kode yang lebih dinamis dan mengendalikan aliran program berdasarkan kondisi yang diberikan. Selanjutnya, kita akan membahas pernyataan elif untuk membuat percabangan yang lebih kompleks.

5. Percabangan If - Elif

Percabangan if - elif adalah struktur kontrol yang digunakan untuk menjalankan blok kode berdasarkan kondisi tertentu, dan memeriksa kondisi lain jika kondisi sebelumnya bernilai False. Dengan percabangan if - elif, Anda dapat membuat program yang lebih responsif dengan menentukan aksi yang berbeda tergantung pada hasil evaluasi beberapa kondisi secara berurutan.

6. Menggunakan Percabangan If - Elif dalam Python

Berikut adalah cara menggunakan pernyataan if - elif dalam Python:

```
nilai = 80

if nilai >= 90:
    print("Nilai Anda: A")
elif nilai >= 80:
    print("Nilai Anda: B")
elif nilai >= 70:
    print("Nilai Anda: C")
elif nilai >= 60:
    print("Nilai Anda: D")
```

Dalam contoh di atas, kita memiliki variabel nilai dengan nilai 80. Kita menggunakan pernyataan if untuk memeriksa apakah nilai tersebut lebih besar atau sama dengan 90. Jika kondisi ini terpenuhi, maka blok kode di dalam pernyataan if akan dieksekusi, dan pesan "Nilai Anda: A" akan ditampilkan. Jika kondisi ini tidak terpenuhi, kita akan memeriksa kondisi berikutnya menggunakan pernyataan elif. Jika nilai lebih besar atau sama dengan 80, maka pesan "Nilai Anda: B" akan ditampilkan, dan seterusnya.

Perhatikan bahwa blok kode di dalam pernyataan if dan elif diindentasi menggunakan spasi atau tab. Indentasi ini penting dalam Python, karena menentukan

blok kode yang termasuk dalam masing-masing pernyataan.

Berikut adalah contoh lain yang lebih menarik mengenai penggunaan percabangan if - elif dalam situasi dunia nyata, dengan penjelasan yang komprehensif:

```
suhu = 25

if suhu < 0:
    print("Cuaca sangat dingin, kenakan
pakaian tebal.")
elif suhu < 10:
    print("Cuaca dingin, kenakan jaket.")
elif suhu < 20:
    print("Cuaca sejuk, kenakan sweater.")
elif suhu < 30:
    print("Cuaca hangat, kenakan pakaian
ringan.")
elif suhu < 40:
    print("Cuaca sangat hangat, pakailah
pakaian yang nyaman.")
else:
    print("Cuaca sangat panas, hati-hati
dengan kepanasan.")
```

Penjelasan Baris per Baris:

- suhu = 25: Menyimpan suhu ke dalam variabel suhu.
- if suhu < 0:: Memeriksa apakah suhu lebih rendah dari 0 derajat.
- print("Cuaca sangat dingin, kenakan pakaian tebal."): Jika suhu kurang dari 0 derajat, mencetak pesan

bahwa cuaca sangat dingin dan saran untuk mengenakan pakaian tebal.

- elif suhu < 10:: Jika suhu tidak kurang dari 0 derajat, memeriksa apakah suhu kurang dari 10 derajat.
- print("Cuaca dingin, kenakan jaket."): Jika suhu kurang dari 10 derajat, mencetak pesan bahwa cuaca dingin dan saran untuk mengenakan jaket.
- elif suhu < 20:: Jika suhu tidak kurang dari 10 derajat, memeriksa apakah suhu kurang dari 20 derajat.
- print("Cuaca sejuk, kenakan sweater."): Jika suhu kurang dari 20 derajat, mencetak pesan bahwa cuaca sejuk dan saran untuk mengenakan sweater.
- elif suhu < 30:: Jika suhu tidak kurang dari 20 derajat, memeriksa apakah suhu kurang dari 30 derajat.
- print("Cuaca hangat, kenakan pakaian ringan."): Jika suhu kurang dari 30 derajat, mencetak pesan bahwa cuaca hangat dan saran untuk mengenakan pakaian ringan.
- elif suhu < 40:: Jika suhu tidak kurang dari 30 derajat, memeriksa apakah suhu kurang dari 40 derajat.
- print("Cuaca sangat hangat, pakailah pakaian yang nyaman."): Jika suhu kurang dari 40 derajat, mencetak

pesan bahwa cuaca sangat hangat dan saran untuk mengenakan pakaian yang nyaman.

- else:: Jika suhu tidak kurang dari 40 derajat, menjalankan blok kode berikutnya.
- print("Cuaca sangat panas, hati-hati dengan kepanasan."): Mencetak pesan bahwa cuaca sangat panas dan saran untuk berhati-hati dengan kepanasan.

Dalam contoh ini, kita menggunakan percabangan if - elif untuk menentukan saran pakaian yang sesuai berdasarkan suhu. Jika suhu di bawah 0 derajat, program akan mencetak pesan bahwa cuaca sangat dingin dan menyarankan pengguna untuk mengenakan pakaian tebal. Jika suhu di antara 0 hingga 10 derajat, program akan mencetak pesan bahwa cuaca dingin dan menyarankan pengguna untuk mengenakan jaket. Kondisi ini berlanjut hingga mencapai kondisi else, yang akan menangani suhu di atas 40 derajat.

Dengan menggunakan percabangan if - elif, Anda dapat menulis kode yang lebih dinamis dan mengendalikan aliran program berdasarkan kondisi yang diberikan. Contoh di atas menunjukkan bagaimana Anda dapat menggunakan percabangan if - elif untuk membuat

program yang memberikan saran pakaian berdasarkan suhu yang ada.

Dengan memahami dan menguasai percabangan if - elif, Anda dapat menulis kode yang lebih dinamis dan mengendalikan aliran program berdasarkan kondisi yang diberikan. Anda juga bisa menambahkan pernyataan else pada akhir percabangan untuk menangani kasus ketika semua kondisi if dan elif sebelumnya bernilai False.

7. Percabangan If - Elif - Else

Percabangan if - elif - else adalah struktur kontrol yang digunakan untuk menjalankan blok kode berdasarkan kondisi tertentu, memeriksa kondisi lain jika kondisi sebelumnya bernilai False, dan menentukan blok kode terakhir yang akan dijalankan jika semua kondisi if dan elif sebelumnya bernilai False. Dengan percabangan if - elif - else, Anda dapat membuat program yang lebih responsif dengan menentukan aksi yang berbeda tergantung pada hasil evaluasi beberapa kondisi secara berurutan.

8. Menggunakan Percabangan If - Elif - Else dalam Python

Berikut adalah cara menggunakan pernyataan if - elif - else dalam Python:

```
nilai = 55

if nilai >= 90:
    print("Nilai Anda: A")
elif nilai >= 80:
    print("Nilai Anda: B")
elif nilai >= 70:
    print("Nilai Anda: C")
elif nilai >= 60:
    print("Nilai Anda: D")
else:
    print("Nilai Anda: E")
```

Dalam contoh di atas, kita memiliki variabel nilai dengan nilai 55. Kita menggunakan pernyataan if untuk memeriksa apakah nilai tersebut lebih besar atau sama dengan 90. Jika kondisi ini terpenuhi, maka blok kode di dalam pernyataan if akan dieksekusi, dan pesan "Nilai Anda: A" akan ditampilkan. Jika kondisi ini tidak terpenuhi, kita akan memeriksa kondisi berikutnya menggunakan pernyataan elif. Jika nilai lebih besar atau sama dengan 80, maka pesan "Nilai Anda: B" akan ditampilkan, dan seterusnya.

Jika semua kondisi if dan elif bernilai False, maka blok kode di dalam pernyataan else akan dijalankan. Dalam contoh ini, pesan "Nilai Anda: E" akan ditampilkan.

Perhatikan bahwa blok kode di dalam pernyataan if, elif, dan else diindentasi menggunakan spasi atau tab.

Indentasi ini penting dalam Python, karena menentukan blok kode yang termasuk dalam masing-masing pernyataan.

Berikut adalah contoh lain yang lebih menarik mengenai penggunaan percabangan if - elif - else dalam situasi dunia nyata, dengan penjelasan yang komprehensif:

```
umur = 35

if umur < 13:
    status = "anak-anak"
elif umur < 18:
    status = "remaja"
elif umur < 60:
    status = "dewasa"
else:
    status = "lansia"

print(f"Status usia Anda: {status}")
```

Penjelasan Baris per Baris:

- umur = 35: Menyimpan umur ke dalam variabel umur.
- if umur < 13:: Memeriksa apakah umur kurang dari 13 tahun.
- status = "anak-anak": Jika umur kurang dari 13 tahun, mengatur status usia sebagai "anak-anak".

- elif umur < 18:: Jika umur tidak kurang dari 13 tahun, memeriksa apakah umur kurang dari 18 tahun.
- status = "remaja": Jika umur kurang dari 18 tahun, mengatur status usia sebagai "remaja".
- elif umur < 60:: Jika umur tidak kurang dari 18 tahun, memeriksa apakah umur kurang dari 60 tahun.
- status = "dewasa": Jika umur kurang dari 60 tahun, mengatur status usia sebagai "dewasa".
- else:: Jika umur tidak kurang dari 60 tahun, menjalankan blok kode berikutnya.
- status = "lansia": Mengatur status usia sebagai "lansia" jika tidak ada kondisi sebelumnya yang terpenuhi.
- print(f"Status usia Anda: {status}"): Mencetak pesan yang menunjukkan status usia berdasarkan umur yang diberikan.

Dalam contoh ini, kita menggunakan percabangan if - elif - else untuk menentukan status usia seseorang berdasarkan umur mereka. Jika umur di bawah 13 tahun, program akan mencetak pesan bahwa status usia mereka adalah "anak-anak". Jika umur di antara 13 hingga 18

tahun, program akan mencetak pesan bahwa status usia mereka adalah "remaja", dan seterusnya.

Dengan menggunakan percabangan if - elif - else, Anda dapat menulis kode yang lebih dinamis dan mengendalikan aliran program berdasarkan kondisi yang diberikan. Contoh di atas menunjukkan bagaimana Anda dapat menggunakan percabangan if - elif - else untuk membuat program yang menentukan status usia seseorang berdasarkan umur yang ada.

Dengan memahami dan menguasai percabangan if - elif - else, Anda dapat menulis kode yang lebih dinamis dan mengendalikan aliran program berdasarkan kondisi yang diberikan.

2.4. Perulangan (For, While)

Dalam dunia pemrograman, kita sering kali perlu menjalankan blok kode berulang kali. Misalnya, kita ingin mencetak angka dari 1 hingga 10, atau kita ingin menghitung total jumlah elemen dalam daftar. Untuk melakukan tugas-tugas seperti ini, kita menggunakan konsep perulangan.

Perulangan adalah struktur kontrol yang memungkinkan kita untuk menjalankan blok kode

beberapa kali hingga kondisi tertentu terpenuhi. Python menyediakan dua jenis perulangan, yaitu perulangan for dan perulangan while. Keduanya memiliki kegunaan dan kelebihan tersendiri, tergantung pada situasi dan kebutuhan kita.

Dalam bab ini, kita akan membahas tentang perulangan for dan while. Anda akan mempelajari cara menggunakan perulangan for untuk mengulangi blok kode dengan jumlah iterasi yang telah ditentukan sebelumnya, dan perulangan while untuk mengulangi blok kode selama kondisi tertentu masih terpenuhi. Selain itu, kita akan membahas berbagai teknik dan contoh yang dapat membantu Anda memahami perulangan dan menggunakannya secara efisien dalam kode Python Anda.

1. Perulangan For

Perulangan for adalah struktur kontrol yang digunakan untuk mengulangi blok kode sejumlah kali yang telah ditentukan sebelumnya. Dalam Python, perulangan for umumnya digunakan bersama dengan fungsi range() atau objek iterable seperti daftar, tuple, dan set.

2. Menggunakan Perulangan For dalam Python

Berikut adalah cara menggunakan perulangan for dalam Python:

```
for i in range(5):
    print(i)
```

Dalam contoh di atas, kita menggunakan perulangan for untuk mengulangi blok kode sebanyak 5 kali. Fungsi range(5) menghasilkan rentang angka dari 0 hingga 4 (5 angka secara total). Pada setiap iterasi, variabel i akan diisi dengan angka berikutnya dalam rentang tersebut, dan blok kode di dalam perulangan for akan dieksekusi. Hasilnya, angka 0 hingga 4 akan dicetak satu per satu.

Perhatikan bahwa blok kode di dalam perulangan for diindentasi menggunakan spasi atau tab. Indentasi ini penting dalam Python, karena menentukan blok kode yang termasuk dalam perulangan.

Contoh Lain Menggunakan Perulangan For

Berikut adalah contoh lain yang menggunakan perulangan for dengan objek iterable (daftar):

```
buah = ['apel', 'pisang', 'jeruk', 'mangga']

for b in buah:
    print(b)
```

Dalam contoh ini, kita memiliki daftar buah yang berisi beberapa jenis buah. Kita menggunakan perulangan for untuk mengulangi setiap elemen dalam daftar tersebut. Pada setiap iterasi, variabel b akan diisi dengan elemen berikutnya dalam daftar buah, dan blok kode di dalam perulangan for akan dieksekusi. Hasilnya, nama-nama buah dalam daftar akan dicetak satu per satu.

Berikut adalah contoh menarik yang melibatkan perulangan for dalam operasi matematika. Kita akan mencari jumlah deret aritmatika menggunakan perulangan for. Misalnya, kita ingin menghitung jumlah deret aritmatika dari 1 hingga 10.

```
n = 10
jumlah = 0

for i in range(1, n+1):
    jumlah += i

print(f"Jumlah deret aritmatika dari 1 hingga
{n} adalah {jumlah}")
```

Penjelasan Baris per Baris:

- n = 10: Menyimpan nilai n yang merupakan batas atas deret aritmatika yang ingin dijumlahkan.

- jumlah = 0: Inisialisasi variabel jumlah untuk menyimpan hasil penjumlahan deret.
- for i in range(1, n+1):: Membuat perulangan for dengan menggunakan fungsi range() yang mulai dari angka 1 hingga angka n (dalam hal ini 10). Perhatikan penggunaan n+1 karena fungsi range() tidak mencakup batas atas dalam rentang angkanya.
- jumlah += i: Menambahkan nilai i ke variabel jumlah pada setiap iterasi perulangan.
- print(f"Jumlah deret aritmatika dari 1 hingga {n} adalah {jumlah}"): Mencetak hasil penjumlahan deret aritmatika setelah selesai melakukan perulangan.

Dalam contoh ini, kita menggunakan perulangan for untuk menghitung jumlah deret aritmatika dari angka 1 hingga angka 10. Pada setiap iterasi perulangan, kita menambahkan nilai i ke variabel jumlah. Setelah perulangan selesai, kita mencetak hasil penjumlahan deret aritmatika.

Perulangan for sangat berguna dalam operasi matematika seperti ini, di mana kita perlu mengulangi blok kode dengan jumlah iterasi yang telah ditentukan sebelumnya. Contoh ini menunjukkan bagaimana Anda dapat menggunakan perulangan for untuk menyelesaikan

permasalahan matematika yang melibatkan deret aritmatika.

Berikut adalah contoh aplikasi dunia nyata yang menggunakan kombinasi perulangan for dan percabangan if-elif-else. Misalkan kita ingin mengelompokkan siswa berdasarkan nilai yang mereka peroleh dalam suatu ujian.

```
nilai_siswa = [75, 90, 65, 85, 55, 92, 78, 80,
48, 67]

for nilai in nilai_siswa:
    if nilai >= 90:
        kategori = 'A'
    elif nilai >= 80:
        kategori = 'B'
    elif nilai >= 70:
        kategori = 'C'
    elif nilai >= 60:
        kategori = 'D'
    else:
        kategori = 'E'

    print(f"Siswa dengan nilai {nilai} masuk
dalam kategori {kategori}.")
```

Penjelasan Baris per Baris:

- nilai_siswa = [75, 90, 65, 85, 55, 92, 78, 80, 48, 67]: Membuat daftar nilai_siswa yang berisi nilai siswa dalam ujian.

- for nilai in nilai_siswa:: Membuat perulangan for yang akan beriterasi melalui setiap nilai dalam daftar nilai_siswa.
- 3-11. Percabangan if-elif-else untuk mengelompokkan siswa berdasarkan nilai yang mereka peroleh:
- Jika nilai siswa lebih dari atau sama dengan 90, mereka akan masuk kategori 'A'.
- Jika nilai siswa lebih dari atau sama dengan 80, mereka akan masuk kategori 'B'.
- Jika nilai siswa lebih dari atau sama dengan 70, mereka akan masuk kategori 'C'.
- Jika nilai siswa lebih dari atau sama dengan 60, mereka akan masuk kategori 'D'.
- Jika nilai siswa kurang dari 60, mereka akan masuk kategori 'E'.
- print(f"Siswa dengan nilai {nilai} masuk dalam kategori {kategori}."): Mencetak hasil kategori siswa berdasarkan nilai yang mereka peroleh dalam ujian.

Dalam contoh ini, kita menggunakan perulangan for untuk mengulangi setiap elemen dalam daftar nilai_siswa. Pada setiap iterasi, kita menggunakan percabangan if-

elif-else untuk menentukan kategori siswa berdasarkan nilai yang mereka peroleh. Setelah itu, kita mencetak hasil kategori siswa.

Contoh ini menunjukkan bagaimana Anda dapat menggunakan kombinasi perulangan for dan percabangan if-elif-else untuk menyelesaikan permasalahan dunia nyata seperti mengelompokkan siswa berdasarkan nilai ujian.

Dengan memahami dan menguasai perulangan for, Anda dapat menulis kode yang lebih efisien dan ringkas, serta mengendalikan aliran program dengan lebih baik. Perulangan for sangat berguna dalam berbagai situasi, seperti mengulangi blok kode sejumlah kali yang telah ditentukan sebelumnya, mengakses elemen dalam objek iterable, dan banyak lagi.

3. Perulangan While

Perulangan while adalah struktur kontrol yang digunakan untuk mengulangi blok kode selama kondisi tertentu masih terpenuhi. Perulangan while terus berjalan hingga kondisi yang diberikan menjadi salah atau False. Perulangan ini sering digunakan ketika jumlah iterasi tidak diketahui sebelumnya.

Menggunakan Perulangan While dalam Python

Berikut adalah cara menggunakan perulangan while dalam Python:

```
angka = 1

while angka <= 5:
    print(angka)
    angka += 1
```

Dalam contoh di atas, kita menggunakan perulangan while untuk mengulangi blok kode selama nilai angka kurang dari atau sama dengan 5. Pada setiap iterasi, blok kode di dalam perulangan while akan dieksekusi, dan nilai angka akan dicetak. Setelah itu, kita menambahkan 1 ke variabel angka. Perulangan akan berakhir ketika angka lebih besar dari 5.

Contoh Lain Menggunakan Perulangan While

Berikut adalah contoh lain yang menggunakan perulangan while untuk menghitung jumlah deret bilangan ganjil:

```
n = 10
i = 1
jumlah_ganjil = 0

while i <= n:
    if i % 2 != 0:
        jumlah_ganjil += i
    i += 1
```

```
print(f"Jumlah deret bilangan ganjil dari 1
hingga {n} adalah {jumlah_ganjil}")
```

Dalam contoh ini, kita menggunakan perulangan while untuk menghitung jumlah deret bilangan ganjil dari 1 hingga n (dalam hal ini 10). Pada setiap iterasi, kita memeriksa apakah i merupakan bilangan ganjil atau tidak dengan menggunakan percabangan if. Jika i adalah bilangan ganjil, kita menambahkannya ke variabel jumlah_ganjil. Kemudian, kita menambahkan 1 ke variabel i. Perulangan akan berakhir ketika i lebih besar dari n.

Berikut adalah contoh menarik yang menggunakan perulangan while dalam dunia nyata. Kita akan membuat program yang menemukan bilangan Fibonacci pertama yang lebih besar dari angka yang diberikan.

```
angka = 1000
fib1, fib2 = 0, 1
fibonacci = 0

while fibonacci <= angka:
    fibonacci = fib1 + fib2
    fib1, fib2 = fib2, fibonacci

print(f"Bilangan Fibonacci pertama yang lebih
besar dari {angka} adalah {fibonacci}")
```

Penjelasan Baris per Baris:

- angka = 1000: Menentukan angka yang ingin kita cari bilangan Fibonacci pertama yang lebih besar darinya.
- fib1, fib2 = 0, 1: Inisialisasi dua bilangan Fibonacci pertama.
- fibonacci = 0: Inisialisasi variabel fibonacci yang akan menyimpan bilangan Fibonacci saat ini.
- while fibonacci <= angka:: Membuat perulangan while yang akan berjalan selama bilangan Fibonacci saat ini (fibonacci) lebih kecil dari atau sama dengan angka yang ditentukan (angka).
- fibonacci = fib1 + fib2: Menghitung bilangan Fibonacci berikutnya dengan menjumlahkan dua bilangan Fibonacci sebelumnya (fib1 dan fib2).
- fib1, fib2 = fib2, fibonacci: Memperbarui nilai fib1 dan fib2 untuk iterasi berikutnya.
- print(f"Bilangan Fibonacci pertama yang lebih besar dari {angka} adalah {fibonacci}"): Mencetak bilangan Fibonacci pertama yang lebih besar dari angka yang diberikan.

Dalam contoh ini, kita menggunakan perulangan while untuk mencari bilangan Fibonacci pertama yang

lebih besar dari angka yang diberikan. Pada setiap iterasi, kita menghitung bilangan Fibonacci berikutnya dan memeriksa apakah bilangan ini lebih besar dari angka yang diberikan. Jika kondisi ini terpenuhi, perulangan berakhir, dan kita mencetak bilangan Fibonacci yang ditemukan.

Contoh ini menunjukkan bagaimana Anda dapat menggunakan perulangan while untuk menyelesaikan permasalahan dunia nyata yang melibatkan deret angka, seperti deret Fibonacci. Dalam hal ini, perulangan while memberikan kontrol yang baik atas jumlah iterasi yang tidak diketahui sebelumnya.

Berikut adalah contoh aplikasi dunia nyata yang menggunakan kombinasi perulangan while dan percabangan if-elif-else. Dalam contoh ini, kita akan membuat program yang meminta pengguna untuk memasukkan angka dan mengelompokkan angka tersebut berdasarkan kategorinya (positif, negatif, atau nol). Program akan terus meminta masukan hingga pengguna memasukkan angka -999.

```
while True:
    angka = int(input("Masukkan angka (-999
untuk keluar): "))
```

```
    if angka == -999:
        print("Terima kasih telah menggunakan
program ini!")
        break
    elif angka > 0:
        kategori = "positif"
    elif angka < 0:
        kategori = "negatif"
    else:
        kategori = "nol"

    print(f"Angka {angka} merupakan angka
{kategori}.")
```

Penjelasan Baris per Baris:

- while True:: Membuat perulangan while yang akan berjalan selama kondisinya benar (True).
- angka = int(input("Masukkan angka (-999 untuk keluar): ")): Meminta pengguna memasukkan angka dan menyimpannya dalam variabel angka.
- 3-11. Percabangan if-elif-else untuk mengelompokkan angka berdasarkan kategori:
- Jika angka yang dimasukkan adalah -999, cetak pesan terima kasih dan keluar dari perulangan menggunakan perintah break.
- Jika angka lebih dari 0, angka tersebut dikategorikan sebagai "positif".

- Jika angka kurang dari 0, angka tersebut dikategorikan sebagai "negatif".
- Jika angka sama dengan 0, angka tersebut dikategorikan sebagai "nol".
- print(f"Angka {angka} merupakan angka {kategori}."): Mencetak hasil kategori angka yang dimasukkan pengguna.

Dalam contoh ini, kita menggunakan perulangan while untuk meminta masukan angka dari pengguna hingga pengguna memutuskan untuk keluar dengan memasukkan angka -999. Pada setiap iterasi, kita menggunakan percabangan if-elif-else untuk menentukan kategori angka yang dimasukkan pengguna. Setelah itu, kita mencetak hasil kategori angka tersebut.

Contoh ini menunjukkan bagaimana Anda dapat menggunakan kombinasi perulangan while dan percabangan if-elif-else untuk menyelesaikan permasalahan dunia nyata seperti mengelompokkan angka berdasarkan kategorinya.

Dengan memahami dan menguasai perulangan while, Anda dapat menulis kode yang lebih efisien dan fleksibel, serta mengendalikan aliran program dengan lebih baik. Perulangan while sangat berguna dalam

berbagai situasi, terutama ketika jumlah iterasi tidak diketahui sebelumnya atau perlu berubah selama eksekusi program.

BAB III
STRUKTUR DATA

Dalam bab ini, kita akan membahas struktur data yang ada dalam bahasa pemrograman Python. Struktur data adalah cara yang digunakan untuk menyimpan dan mengatur data dalam komputer, sehingga dapat diakses dan dimanipulasi secara efisien dan efektif. Struktur data yang tepat akan membantu Anda menangani data dengan lebih mudah dan membuat kode Anda lebih mudah untuk dibaca dan dikelola.

Python menyediakan beberapa struktur data bawaan, seperti list, tuple, set, dan dictionary. Setiap struktur data ini memiliki karakteristik dan kegunaan yang berbeda, serta metode dan operasi yang berhubungan dengannya.

Dalam bab ini, Anda akan mempelajari tentang:

- List: koleksi data yang berurutan, dapat diubah, dan dapat berisi elemen yang berbeda.
- Tuple: koleksi data yang berurutan, tidak dapat diubah, dan dapat berisi elemen yang berbeda.
- Set: koleksi data yang tidak berurutan dan tidak berindeks, serta tidak memiliki elemen duplikat.

- Dictionary: koleksi data yang tidak berurutan dan berindeks, dengan pasangan kunci-nilai yang unik.

Dengan memahami dan menguasai struktur data ini, Anda akan dapat menyimpan dan mengelola data dengan lebih efisien, serta menemukan solusi yang lebih efektif untuk berbagai permasalahan pemrograman.

3.1. List

List adalah struktur data di Python yang digunakan untuk menyimpan kumpulan data yang berurutan. Elemen dalam list dapat diakses melalui indeks, yang dimulai dari angka 0 untuk elemen pertama. List bersifat dinamis, sehingga Anda dapat menambahkan, menghapus, atau memodifikasi elemen dengan mudah.

Berikut adalah beberapa contoh dasar tentang cara menggunakan list dalam Python:

1. Membuat List

Untuk membuat list, Anda perlu menggunakan tanda kurung siku [] dan memisahkan setiap elemen dengan koma. Elemen dalam list dapat berupa tipe data apa pun, termasuk angka, string, dan objek lainnya.

```
angka = [1, 2, 3, 4, 5]
nama = ["Budi", "Susi", "Rina"]
```

```
campuran = [1, "dua", 3.0, True]
```

2. Mengakses Elemen dalam List

Anda dapat mengakses elemen dalam list menggunakan indeks yang dimulai dari 0.

```
angka = [1, 2, 3, 4, 5]
print(angka[0])  # Output: 1
print(angka[2])  # Output: 3
```

Anda juga dapat menggunakan indeks negatif untuk mengakses elemen dari belakang list.

```
print(angka[-1])  # Output: 5
print(angka[-3])  # Output: 3
```

3. Menambahkan Elemen ke List

Untuk menambahkan elemen ke list, Anda dapat menggunakan metode append().

```
angka = [1, 2, 3, 4, 5]
angka.append(6)
print(angka)  # Output: [1, 2, 3, 4, 5, 6]
```

4. Menghapus Elemen dari List

Anda dapat menghapus elemen dari list menggunakan perintah del, metode remove(), atau metode pop().

```
angka = [1, 2, 3, 4, 5]

del angka[1]  # Menghapus elemen dengan indeks
1 (angka 2)
print(angka)  # Output: [1, 3, 4, 5]

angka.remove(4)  # Menghapus elemen 4
print(angka)  # Output: [1, 3, 5]

angka.pop()  # Menghapus elemen terakhir
print(angka)  # Output: [1, 3]
```

5. Slicing List

Slicing adalah teknik untuk memotong bagian dari list. Anda dapat menggunakan indeks dan titik dua : untuk menentukan awal dan akhir potongan.

```
angka = [1, 2, 3, 4, 5]
potongan = angka[1:4]
print(potongan)  # Output: [2, 3, 4]
```

Dalam contoh ini, kita menggunakan list untuk menyimpan dan mengelola kumpulan data yang berurutan. Kita mempelajari cara membuat list, mengakses elemen, menambahkan dan menghapus elemen, serta cara slicing list. List merupakan struktur data yang sangat fleksibel dan umum digunakan dalam berbagai situasi pemrograman.

Berikut adalah contoh penggunaan list dalam dunia nyata untuk mengelola data pelanggan di sebuah toko:

```
pelanggan = ["Budi", "Susi", "Rina", "Agus",
"Mira"]

# Menambahkan pelanggan baru
pelanggan.append("Yuni")
print(pelanggan)  # Output: ['Budi', 'Susi',
'Rina', 'Agus', 'Mira', 'Yuni']

# Menghapus pelanggan
pelanggan.remove("Agus")
print(pelanggan)  # Output: ['Budi', 'Susi',
'Rina', 'Mira', 'Yuni']

# Mengurutkan daftar pelanggan
pelanggan.sort()
print(pelanggan)  # Output: ['Budi', 'Mira',
'Rina', 'Susi', 'Yuni']

# Menampilkan jumlah pelanggan
print("Jumlah pelanggan:", len(pelanggan))  #
Output: Jumlah pelanggan: 5

# Menampilkan 3 pelanggan pertama
print("Tiga pelanggan pertama:",
pelanggan[:3])  # Output: Tiga pelanggan
pertama: ['Budi', 'Mira', 'Rina']
```

Penjelasan Baris per Baris:

- pelanggan = ["Budi", "Susi", "Rina", "Agus", "Mira"]: Membuat list dengan nama pelanggan.
- 2-3. Menambahkan pelanggan baru "Yuni" ke list.
- 4-5. Menghapus pelanggan "Agus" dari list.

- 6-7. Mengurutkan daftar pelanggan menggunakan metode sort().
- Menampilkan jumlah pelanggan dengan fungsi len().
- Menampilkan tiga pelanggan pertama dengan slicing list.

Dalam contoh ini, kita menggunakan list untuk menyimpan dan mengelola data pelanggan di sebuah toko. Kita menambahkan dan menghapus pelanggan, mengurutkan daftar pelanggan, menampilkan jumlah pelanggan, dan menampilkan beberapa pelanggan pertama dari list. Contoh ini menunjukkan bagaimana list dapat digunakan untuk mengelola data dalam aplikasi dunia nyata.

3.2. Tuple

Tuple adalah struktur data di Python yang mirip dengan list, tetapi bersifat tidak dapat diubah (immutable). Artinya, setelah tuple dibuat, Anda tidak dapat menambahkan, menghapus, atau mengubah elemennya. Karena sifatnya yang tidak dapat diubah, tuple biasanya digunakan untuk menyimpan koleksi data yang tidak akan berubah selama eksekusi program, seperti konstanta atau konfigurasi.

Berikut adalah beberapa contoh dasar tentang cara menggunakan tuple dalam Python:

1. Membuat Tuple

Untuk membuat tuple, Anda perlu menggunakan tanda kurung () dan memisahkan setiap elemen dengan koma. Seperti list, elemen dalam tuple dapat berupa tipe data apa pun, termasuk angka, string, dan objek lainnya.

```
angka = (1, 2, 3, 4, 5)
nama = ("Budi", "Susi", "Rina")
campuran = (1, "dua", 3.0, True)
```

2. Mengakses Elemen dalam Tuple

Anda dapat mengakses elemen dalam tuple menggunakan indeks yang dimulai dari 0, sama seperti list.

```
angka = (1, 2, 3, 4, 5)
print(angka[0])  # Output: 1
print(angka[2])  # Output: 3
```

3. Penggabungan Tuple

Anda dapat menggabungkan dua atau lebih tuple dengan menggunakan operator +.

```
tuple1 = (1, 2, 3)
tuple2 = (4, 5, 6)
hasil = tuple1 + tuple2
```

```
print(hasil)  # Output: (1, 2, 3, 4, 5, 6)
```

4. Menghitung Jumlah Elemen

Anda dapat menghitung jumlah elemen dalam tuple dengan menggunakan fungsi len().

```
angka = (1, 2, 3, 4, 5)
print(len(angka))  # Output: 5
```

Dalam contoh ini, kita menggunakan tuple untuk menyimpan dan mengelola kumpulan data yang tidak akan berubah selama eksekusi program. Kita mempelajari cara membuat tuple, mengakses elemen, menggabungkan tuple, dan menghitung jumlah elemennya. Meskipun tuple memiliki keterbatasan dibandingkan list, mereka berguna dalam situasi di mana kekebalan data dan integritas adalah prioritas utama.

Berikut adalah contoh penggunaan tuple dalam dunia nyata untuk menyimpan informasi mengenai koordinat geografis dari beberapa kota:

```
jakarta = (-6.200000, 106.816666)
bandung = (-6.917464, 107.619123)
surabaya = (-7.257472, 112.752088)
medan = (3.585242, 98.675598)

# Menampilkan koordinat Jakarta
```

```
print("Koordinat Jakarta:", jakarta)  #
Output: Koordinat Jakarta: (-6.200000,
106.816666)

# Menghitung jarak antara kota dengan rumus
haversine (contoh sederhana)
import math

def haversine(lat1, lon1, lat2, lon2):
    r = 6371  # radius Bumi dalam km
    dlat = math.radians(lat2 - lat1)
    dlon = math.radians(lon2 - lon1)
    a = (math.sin(dlat / 2) ** 2 +
         math.cos(math.radians(lat1)) *
math.cos(math.radians(lat2)) *
         math.sin(dlon / 2) ** 2)
    c = 2 * math.atan2(math.sqrt(a),
math.sqrt(1 - a))
    return r * c

jarak = haversine(jakarta[0], jakarta[1],
bandung[0], bandung[1])
print("Jarak Jakarta-Bandung:", jarak, "km")
# Output: Jarak Jakarta-Bandung:
116.28327294319566 km
```

Penjelasan Baris per Baris:

- 1-4. Membuat tuple yang berisi koordinat geografis (latitude, longitude) dari beberapa kota di Indonesia.
- 5. Menampilkan koordinat Jakarta.
- 7-17. Mendefinisikan fungsi haversine untuk menghitung jarak antara dua titik koordinat dengan rumus haversine. Fungsi ini merupakan contoh

sederhana dan mungkin tidak akurat untuk penggunaan sebenarnya.

Dalam contoh ini, kita menggunakan tuple untuk menyimpan dan mengelola data koordinat geografis dari beberapa kota. Kita menampilkan koordinat salah satu kota dan menghitung jarak antara dua kota menggunakan rumus haversine. Contoh ini menunjukkan bagaimana tuple dapat digunakan untuk mengelola data dalam aplikasi dunia nyata yang memerlukan kekebalan data dan integritas.

3.3. Set

Set adalah struktur data di Python yang digunakan untuk menyimpan kumpulan data yang tidak memiliki urutan dan elemen yang unik. Set tidak mengizinkan adanya elemen duplikat dan tidak mengindikasikan posisi elemen. Set sangat berguna dalam operasi matematika seperti penggabungan, perpotongan, dan selisih.

Berikut adalah beberapa contoh dasar tentang cara menggunakan set dalam Python:

1. Membuat Set

Untuk membuat set, Anda perlu menggunakan tanda kurung kurawal {} dan memisahkan setiap elemen dengan koma. Seperti list dan tuple, elemen dalam set dapat berupa tipe data apa pun, termasuk angka, string, dan objek lainnya. Anda juga bisa membuat set menggunakan fungsi set().

```
angka = {1, 2, 3, 4, 5}
nama = {"Budi", "Susi", "Rina"}
campuran = {1, "dua", 3.0, True}
kosong = set()
```

2. Menambahkan Elemen ke Set

Anda dapat menambahkan elemen ke set dengan menggunakan metode add().

```
buah = {"apel", "pisang", "jeruk"}
buah.add("mangga")
print(buah)  # Output: {'apel', 'pisang',
'jeruk', 'mangga'}
```

3. Menghapus Elemen dari Set

Anda dapat menghapus elemen dari set dengan menggunakan metode remove().

```
buah = {"apel", "pisang", "jeruk"}
buah.remove("pisang")
print(buah)  # Output: {'apel', 'jeruk'}
```

4. Operasi Set

Set mendukung operasi matematika seperti penggabungan, perpotongan, dan selisih.

```
A = {1, 2, 3, 4, 5}
B = {4, 5, 6, 7, 8}

# Penggabungan (union)
print(A | B)  # Output: {1, 2, 3, 4, 5, 6, 7,
8}

# Perpotongan (intersection)
print(A & B)  # Output: {4, 5}

# Selisih (difference)
print(A - B)  # Output: {1, 2, 3}
```

Dalam contoh ini, kita menggunakan set untuk menyimpan dan mengelola kumpulan data yang tidak memiliki urutan dan elemen yang unik. Kita mempelajari cara membuat set, menambahkan dan menghapus elemen, serta melakukan operasi matematika. Set sangat berguna dalam situasi di mana Anda perlu mengelola data yang tidak memiliki urutan dan elemen yang unik serta melakukan operasi matematika pada kumpulan data tersebut.

Berikut adalah contoh penggunaan set dalam dunia nyata untuk mengidentifikasi elemen yang unik dari

beberapa kumpulan data dan menemukan elemen yang sama di antara mereka:

```
pengguna_A = {"Budi", "Susi", "Rina", "Roni",
"Wati"}
pengguna_B = {"Andi", "Rina", "Wati", "Susi",
"Nita"}

# Menemukan pengguna unik yang mengikuti kedua
akun
unik = pengguna_A ^ pengguna_B
print("Pengguna unik:", unik)
# Output: Pengguna unik: {'Budi', 'Roni',
'Nita', 'Andi'}

# Menemukan pengguna yang mengikuti baik akun
A maupun akun B
sama = pengguna_A & pengguna_B
print("Pengguna yang sama:", sama)
# Output: Pengguna yang sama: {'Susi', 'Wati',
'Rina'}
```

Penjelasan Baris per Baris:

- 1-2. Membuat dua set pengguna_A dan pengguna_B yang berisi nama-nama pengguna yang mengikuti dua akun yang berbeda.
- 4. Menemukan pengguna unik yang mengikuti kedua akun menggunakan operator XOR (^). Operator ini mengembalikan elemen yang hanya ada di satu set, bukan di kedua set.

Dalam contoh ini, kita menggunakan set untuk menyimpan dan mengelola kumpulan pengguna yang mengikuti dua akun yang berbeda. Kita menemukan pengguna unik yang mengikuti kedua akun dan pengguna yang mengikuti baik akun A maupun akun B. Contoh ini menunjukkan bagaimana set dapat digunakan untuk mengelola data dalam aplikasi dunia nyata yang memerlukan pengidentifikasian elemen yang unik dan elemen yang sama di antara beberapa kumpulan data.

3.4. Dictionary

Dictionary adalah struktur data di Python yang digunakan untuk menyimpan pasangan kunci-nilai yang tidak memiliki urutan. Dalam dictionary, setiap kunci bersifat unik dan terkait dengan nilai tertentu. Dictionary sering digunakan untuk menyimpan data dalam format yang mudah dimengerti, seperti JSON.

Berikut adalah beberapa contoh dasar tentang cara menggunakan dictionary dalam Python:

1. Membuat Dictionary

Untuk membuat dictionary, Anda perlu menggunakan tanda kurung kurawal {} dan memisahkan setiap pasangan kunci-nilai dengan koma. Pasangan

kunci-nilai dalam dictionary ditulis dengan menggunakan format kunci: nilai.

```
siswa = {"nama": "Budi", "umur": 17, "kelas":
"12A"}
kota = {1: "Jakarta", 2: "Bandung", 3:
"Surabaya"}
```

2. Mengakses Elemen Dictionary

Anda dapat mengakses elemen dictionary dengan menggunakan kunci yang terkait dengan elemen tersebut.

```
siswa = {"nama": "Budi", "umur": 17, "kelas":
"12A"}
print(siswa["nama"])  # Output: Budi
```

3. Menambahkan atau Mengganti Elemen Dictionary

Anda dapat menambahkan elemen baru atau mengganti elemen yang ada dalam dictionary dengan menggunakan tanda =.

```
siswa = {"nama": "Budi", "umur": 17, "kelas":
"12A"}
siswa["umur"] = 18
siswa["alamat"] = "Jakarta"
print(siswa)  # Output: {'nama': 'Budi',
'umur': 18, 'kelas': '12A', 'alamat':
'Jakarta'}
```

4. Menghapus Elemen Dictionary

Anda dapat menghapus elemen dari dictionary dengan menggunakan perintah del.

```
siswa = {"nama": "Budi", "umur": 17, "kelas":
"12A"}
del siswa["umur"]
print(siswa)  # Output: {'nama': 'Budi',
'kelas': '12A'}
```

5. Iterasi Melalui Dictionary

Anda dapat mengiterasi melalui dictionary dengan menggunakan perulangan for.

```
siswa = {"nama": "Budi", "umur": 17, "kelas":
"12A"}

for kunci, nilai in siswa.items():
    print(kunci, ":", nilai)

# Output:
# nama : Budi
# umur : 17
# kelas : 12A
```

Dalam contoh ini, kita menggunakan dictionary untuk menyimpan dan mengelola pasangan kunci-nilai yang tidak memiliki urutan. Kita mempelajari cara membuat dictionary, mengakses elemen, menambahkan

atau mengganti elemen, menghapus elemen, dan mengiterasi melalui dictionary. Dictionary sangat berguna dalam situasi di mana Anda perlu mengelola data dalam format pasangan kunci-nilai yang mudah dimengerti.

Berikut adalah contoh penggunaan dictionary dalam dunia nyata untuk mengelola data tentang kurs mata uang. Kita akan mengkonversi mata uang antara USD, IDR, dan EUR.

```
# Kurs mata uang
kurs = {
    "USD": {
        "IDR": 14000,
        "EUR": 0.85
    },
    "IDR": {
        "USD": 0.000071,
        "EUR": 0.000061
    },
    "EUR": {
        "USD": 1.18,
        "IDR": 16500
    }
}

# Fungsi untuk mengkonversi mata uang
def konversi_mata_uang(jumlah, mata_uang_asal,
mata_uang_tujuan):
    return jumlah *
kurs[mata_uang_asal][mata_uang_tujuan]

# Contoh penggunaan
uang_asal = 1000
```

```
print("IDR 1000 ke USD:",
konversi_mata_uang(uang_asal, "IDR", "USD"))
# Output: IDR 1000 ke USD: 0.071
print("USD 100 ke EUR:",
konversi_mata_uang(100, "USD", "EUR"))
# Output: USD 100 ke EUR: 85.0
print("EUR 50 ke IDR:", konversi_mata_uang(50,
"EUR", "IDR"))           # Output: EUR 50 ke
IDR: 825000
```

Penjelasan Baris per Baris:

- 1-12. Membuat dictionary kurs yang berisi data tentang kurs mata uang antara USD, IDR, dan EUR. Setiap mata uang memiliki dictionary sendiri yang berisi informasi kurs terhadap mata uang lainnya.
- 14-16. Mendefinisikan fungsi konversi_mata_uang() yang menerima jumlah uang, mata uang asal, dan mata uang tujuan sebagai argumen, lalu mengembalikan jumlah uang yang dikonversi.
- Mendefinisikan uang_asal sebagai contoh uang yang akan dikonversi.
- 20-22. Contoh penggunaan fungsi konversi_mata_uang() untuk mengkonversi mata uang antara IDR, USD, dan EUR.

Dalam contoh ini, kita menggunakan dictionary untuk menyimpan dan mengelola informasi tentang kurs mata uang. Kita membuat fungsi konversi_mata_uang()

yang menggunakan dictionary ini untuk mengkonversi mata uang antara IDR, USD, dan EUR. Contoh ini menunjukkan bagaimana dictionary dapat digunakan untuk mengelola data yang lebih kompleks dalam aplikasi dunia nyata, seperti mengelola informasi tentang kurs mata uang.

3.5. Perbedaan List, Tuple, Set dan Dictionary

Berikut adalah perbedaan, kelebihan, dan kekurangan dari setiap struktur data (List, Tuple, Set, dan Dictionary):

List:

List adalah struktur data yang dapat menyimpan sekumpulan elemen yang dapat berubah dan memiliki urutan. Anda dapat menambahkan, menghapus, atau mengganti elemen di dalam list.

Kelebihan:

- Mudah digunakan dan fleksibel.
- Dapat menyimpan elemen dengan tipe data yang berbeda.
- Dukungan metode yang baik untuk manipulasi elemen (seperti .append(), .remove(), dll.).

Kekurangan:

- Tidak cocok untuk operasi yang memerlukan kunci (seperti pencarian elemen berdasarkan kunci atau pengindeksan yang lebih kompleks).
- Tidak efisien untuk operasi pencarian elemen yang besar.

Tuple:

Tuple mirip dengan list, tetapi tidak dapat diubah. Setelah tuple dibuat, Anda tidak dapat menambahkan, menghapus, atau mengganti elemen di dalamnya.

Kelebihan:

- Lebih cepat dan mengkonsumsi lebih sedikit memori daripada list.
- Cocok untuk data yang tidak berubah dan memiliki urutan.

Kekurangan:

- Tidak fleksibel seperti list (tidak dapat diubah).
- Tidak cocok untuk operasi yang memerlukan kunci.

Set:

Set adalah struktur data yang tidak memiliki urutan dan tidak dapat memiliki elemen duplikat. Set cocok untuk operasi matematika himpunan seperti gabungan, irisan, dan selisih.

Kelebihan:

- Operasi pencarian elemen lebih cepat daripada list dan tuple.
- Dapat digunakan untuk menghapus elemen duplikat dari kumpulan data.
- Mendukung operasi matematika himpunan.

Kekurangan:

- Tidak memiliki urutan.
- Tidak dapat menyimpan elemen duplikat.
- Tidak cocok untuk operasi yang memerlukan kunci.

Dictionary:

Dictionary adalah struktur data yang menyimpan pasangan kunci-nilai yang tidak memiliki urutan. Dictionary sangat berguna dalam situasi di mana Anda perlu mengelola data dalam format pasangan kunci-nilai yang mudah dimengerti.

Kelebihan:

- Pencarian elemen berdasarkan kunci sangat cepat.
- Fleksibel dan mudah digunakan untuk data yang berpasangan (kunci-nilai).

Kekurangan:

- Lebih memakan memori daripada list dan tuple.

- Tidak memiliki urutan.

Berikut adalah tabel yang menunjukkan perbedaan antara struktur data tersebut:

Struktur Data	Berurutan	Bisa Diubah	Elemen Unik	Berpasangan (Kunci-Nilai)	Kecepatan Pencarian
List	Ya	Ya	Tidak	Tidak	Lambat
Tuple	Ya	Tidak	Tidak	Tidak	Lambat
Set	Tidak	Ya	Ya	Tidak	Cepat
Dictionary	Tidak	Ya	Ya	Ya	Cepat

Dalam tabel di atas, Anda dapat melihat perbedaan, kelebihan, dan kekurangan dari setiap struktur data (List, Tuple, Set, dan Dictionary). Anda dapat menggunakan tabel ini sebagai referensi saat menentukan struktur data yang paling sesuai untuk digunakan dalam berbagai situasi pemrograman.

Setiap struktur data memiliki kegunaannya masing-masing, dan pemilihan yang tepat tergantung pada kebutuhan dan situasi dalam kode Anda. Misalnya, gunakan List atau Tuple saat Anda memerlukan urutan yang tetap dan tidak memerlukan operasi pencarian yang cepat. Gunakan Set untuk situasi di mana Anda memerlukan operasi himpunan atau menghapus duplikat. Terakhir, gunakan Dictionary untuk mengelola

data yang berpasangan (kunci-nilai) dan memerlukan akses cepat berdasarkan kunci.

BAB IV
FUNGSI DAN PROSEDUR DALAM PYTHON

Dalam bab ini, kita akan mempelajari tentang fungsi dan prosedur dalam Python. Fungsi dan prosedur adalah komponen penting dalam pemrograman, karena mereka membantu kita mengorganisasi dan menyederhanakan kode. Dengan menggunakan fungsi dan prosedur, kita dapat memecah kode yang kompleks menjadi bagian-bagian yang lebih kecil dan mudah dikelola, yang akan membuat kode lebih mudah untuk dibaca, dipahami, dan diperbaiki.

Kita akan memulai dengan memahami apa itu fungsi dan prosedur, bagaimana cara membuat dan menggunakannya, serta menggali lebih dalam ke beberapa konsep terkait, seperti parameter, argumen, dan variabel lokal dan global. Selanjutnya, kita akan melihat beberapa contoh kode untuk menggambarkan bagaimana fungsi dan prosedur dapat digunakan dalam berbagai situasi pemrograman.

Setelah menyelesaikan bab ini, Anda akan memiliki pemahaman yang baik tentang bagaimana menggunakan

fungsi dan prosedur untuk membuat kode yang lebih efisien dan mudah dikelola. Selamat belajar!

4.1. Bagaimana Fungsi Bekerja?

Fungsi adalah blok kode yang dirancang untuk melakukan tugas tertentu dan dapat dipanggil berkali-kali dalam program. Fungsi membuat kode lebih modular dan memungkinkan kita untuk menghindari pengulangan kode. Di Python, kita menggunakan kata kunci def untuk mendefinisikan fungsi.

Berikut adalah cara umum untuk mendefinisikan dan menggunakan fungsi di Python:
Mendefinisikan fungsi: Gunakan kata kunci def diikuti oleh nama fungsi dan tanda kurung yang berisi parameter (jika ada). Kemudian, tambahkan titik dua (:) dan mulailah menulis blok kode yang akan dijalankan ketika fungsi dipanggil.

```
def nama_fungsi(parameter):
    # Blok kode yang akan dijalankan
    ...
```

Memanggil fungsi: Untuk memanggil fungsi, cukup tulis nama fungsi diikuti oleh tanda kurung yang berisi argumen (nilai yang dilewatkan ke parameter).

```
nama_fungsi(argumen)
```

Contoh:

```
# Mendefinisikan fungsi yang mencetak "Halo,
dunia!"
def sapa():
    print("Halo, dunia!")

# Memanggil fungsi sapa()
sapa()
```

Output:

```
Halo, dunia!
```

Fungsi dapat menerima parameter, yang merupakan variabel yang didefinisikan dalam tanda kurung fungsi. Parameter memungkinkan kita untuk mempassing nilai ke fungsi saat dipanggil. Berikut adalah contoh fungsi dengan parameter:

```
# Mendefinisikan fungsi yang mencetak pesan
sapaan dengan nama yang diberikan
def sapa(nama):
    print(f"Halo, {nama}!")

# Memanggil fungsi sapa() dengan argumen
"Budi"
sapa("Budi")
```

Output:

```
Halo, Budi!
```

Fungsi juga dapat mengembalikan nilai dengan menggunakan kata kunci return. Nilai yang dikembalikan dapat disimpan dalam variabel atau digunakan dalam ekspresi lain.

Contoh:

```
# Mendefinisikan fungsi yang menghitung luas
persegi panjang
def luas_persegi_panjang(panjang, lebar):
    luas = panjang * lebar
    return luas

# Memanggil fungsi luas_persegi_panjang() dan
menyimpan hasilnya dalam variabel
hasil = luas_persegi_panjang(5, 10)
print(f"Luas persegi panjang: {hasil}")
```

Output:

```
Luas persegi panjang: 50
```

Itulah dasar-dasar tentang bagaimana fungsi bekerja dan cara menggunakannya dalam Python. Dalam sub-bab berikutnya, kita akan melihat lebih lanjut tentang

parameter, argumen, dan variabel lokal dan global dalam konteks fungsi.

4.2. Kapan return Digunakan?

Kata kunci return digunakan dalam fungsi untuk mengembalikan nilai dari fungsi tersebut. return memungkinkan kita untuk mengakhiri eksekusi fungsi dan mengembalikan hasilnya ke pemanggil. Dalam beberapa kasus, fungsi mungkin tidak mengembalikan nilai, dan hanya menjalankan beberapa operasi atau mencetak hasil. Namun, menggunakan return memungkinkan kita untuk membuat fungsi yang lebih fleksibel dan dapat digunakan dalam berbagai situasi.

Berikut ini beberapa situasi di mana return sangat berguna:

Menghitung hasil: Fungsi sering digunakan untuk menghitung hasil berdasarkan input yang diberikan. Dalam kasus ini, return digunakan untuk mengembalikan hasil perhitungan.

```
# Fungsi untuk menghitung kuadrat dari angka
def kuadrat(angka):
    return angka ** 2

hasil = kuadrat(5)
print(hasil)  # Output: 25
```

Mengembalikan nilai boolean: Fungsi bisa digunakan untuk menguji kondisi tertentu dan mengembalikan True atau False. Dalam hal ini, return digunakan untuk mengembalikan hasil evaluasi kondisi.

```
# Fungsi untuk memeriksa apakah suatu angka
ganjil
def apakah_ganjil(angka):
    return angka % 2 == 1

hasil = apakah_ganjil(7)
print(hasil)  # Output: True
```

Menghentikan eksekusi fungsi lebih awal: Terkadang, kita mungkin ingin menghentikan eksekusi fungsi sebelum semua kode di dalamnya dijalankan. Dalam hal ini, return digunakan untuk keluar dari fungsi pada titik yang diinginkan.

```
# Fungsi untuk mencetak angka positif dari
suatu list
def cetak_positif(angka_list):
    for angka in angka_list:
        if angka < 0:
            print("Angka negatif ditemukan!")
            return
        print(angka)

cetak_positif([1, 2, 3, -1, 5])
# Output:
# 1
```

```
# 2
# 3
# Angka negatif ditemukan!
```

Ingatlah bahwa jika tidak ada kata kunci return dalam fungsi atau jika tidak ada nilai yang dikembalikan, fungsi akan secara otomatis mengembalikan None.

```
def contoh_fungsi():
    print("Fungsi ini tidak mengembalikan
nilai")

hasil = contoh_fungsi()
print(hasil)  # Output: None
```

Sekarang Anda telah memahami bagaimana return bekerja dan kapan digunakan dalam Python. Dalam sub-bab berikutnya, kita akan membahas lebih lanjut tentang parameter, argumen, dan variabel lokal dan global dalam konteks fungsi.

4.3. Contoh Penggunaan Fungsi

Contoh 1: Menghitung Luas dan Keliling Persegi Panjang

```
# Fungsi untuk menghitung luas persegi panjang
def luas_persegi_panjang(panjang, lebar):
    luas = panjang * lebar
    return luas
```

```
# Fungsi untuk menghitung keliling persegi
panjang
def keliling_persegi_panjang(panjang, lebar):
    keliling = 2 * (panjang + lebar)
    return keliling

# Contoh penggunaan fungsi
panjang = 10
lebar = 5
luas = luas_persegi_panjang(panjang, lebar)
keliling = keliling_persegi_panjang(panjang,
lebar)

print(f"Luas: {luas}")
print(f"Keliling: {keliling}")
```

Penjelasan:

- Fungsi luas_persegi_panjang didefinisikan untuk menghitung luas persegi panjang dengan parameter panjang dan lebar. Fungsi ini mengembalikan nilai luas.
- Fungsi keliling_persegi_panjang didefinisikan untuk menghitung keliling persegi panjang dengan parameter panjang dan lebar. Fungsi ini mengembalikan nilai keliling.
- Menginisialisasi variabel panjang dan lebar dengan nilai yang diinginkan.

- Menggunakan fungsi luas_persegi_panjang dan keliling_persegi_panjang untuk menghitung luas dan keliling persegi panjang.

Contoh 2: Fungsi untuk Menentukan Bilangan Prima

```
# Fungsi untuk menentukan apakah suatu angka
merupakan bilangan prima
def apakah_prima(angka):
    if angka <= 1:
        return False
    for i in range(2, angka):
        if angka % i == 0:
            return False
    return True

# Contoh penggunaan fungsi
angka = 11
hasil = apakah_prima(angka)
if hasil:
    print(f"{angka} adalah bilangan prima")
else:
    print(f"{angka} bukan bilangan prima")
```

Penjelasan:

- Fungsi apakah_prima didefinisikan untuk mengecek apakah suatu angka merupakan bilangan prima dengan parameter angka. Fungsi ini mengembalikan True jika angka merupakan bilangan prima, dan False jika bukan.

- Jika angka lebih kecil atau sama dengan 1, fungsi mengembalikan False, karena bilangan prima harus lebih besar dari 1.
- Menggunakan for loop untuk mengiterasi dari 2 hingga angka - 1. Jika angka habis dibagi oleh salah satu angka dalam rentang, maka angka bukan bilangan prima, sehingga mengembalikan False.
- Jika loop selesai dan tidak ada angka yang habis membagi angka, maka angka adalah bilangan prima, dan mengembalikan True.
- Menggunakan fungsi apakah_prima untuk mengecek apakah angka yang diberikan merupakan bilangan prima atau bukan, lalu mencetak hasilnya.

Contoh 3: Menghitung Nilai Rata-rata dari Sebuah List

```
# Fungsi untuk menghitung rata-rata dari
sebuah list
def hitung_rata_rata(nilai_list):
    jumlah = sum(nilai_list)
    rata_rata = jumlah / len(nilai_list)
    return rata_rata

# Contoh penggunaan fungsi
nilai_siswa = [80, 85, 90, 78, 88]
rata_rata_nilai =
hitung_rata_rata(nilai_siswa)
print(f"Rata-rata nilai siswa:
{rata_rata_nilai:.2f}")
```

Penjelasan:

- Fungsi hitung_rata_rata didefinisikan untuk menghitung nilai rata-rata dari sebuah list dengan parameter nilai_list. Fungsi ini mengembalikan nilai rata-rata.
- Menggunakan fungsi sum() untuk menghitung jumlah dari elemen-elemen dalam list nilai_list.
- Menghitung rata-rata dengan cara membagi jumlah elemen dengan jumlah elemen dalam list menggunakan fungsi len().
- Menggunakan fungsi hitung_rata_rata untuk menghitung rata-rata nilai siswa.
- Mencetak hasil rata-rata nilai siswa dengan dua angka desimal.

Contoh 4: Menghitung Faktorial dari Suatu Bilangan

```
# Fungsi untuk menghitung faktorial dari suatu
bilangan
def faktorial(n):
    if n == 0:
        return 1
    else:
        return n * faktorial(n - 1)

# Contoh penggunaan fungsi
bilangan = 5
hasil_faktorial = faktorial(bilangan)
print(f"Faktorial dari {bilangan} adalah
{hasil_faktorial}")
```

Penjelasan:

- Fungsi faktorial didefinisikan untuk menghitung faktorial dari suatu bilangan dengan parameter n. Fungsi ini menggunakan rekursi untuk menghitung faktorial dan mengembalikan hasilnya.
- Jika n adalah 0, faktorial dari 0 adalah 1, sehingga mengembalikan 1.
- Jika n bukan 0, menghitung faktorial dengan cara mengalikan n dengan hasil faktorial dari n - 1.
- Menggunakan fungsi faktorial untuk menghitung faktorial dari suatu bilangan.
- Mencetak hasil faktorial dari bilangan tersebut.

4.4. Pengembangan Aplikasi Sederhana

Berikut adalah contoh aplikasi sederhana untuk menghitung gaji bersih karyawan berdasarkan jumlah jam kerja dan tarif per jam:

```
def input_data():
    nama = input("Masukkan nama karyawan: ")
    jam_kerja = float(input("Masukkan jumlah
jam kerja: "))
    tarif_per_jam = float(input("Masukkan
tarif per jam: "))
    return nama, jam_kerja, tarif_per_jam

def hitung_gaji(jam_kerja, tarif_per_jam):
```

```
    if jam_kerja <= 40:
        gaji = jam_kerja * tarif_per_jam
    else:
        gaji_normal = 40 * tarif_per_jam
        gaji_lembur = (jam_kerja - 40) *
(tarif_per_jam * 1.5)
        gaji = gaji_normal + gaji_lembur
    return gaji

def tampilkan_hasil(nama, gaji_bersih):
    print(f"Gaji bersih karyawan {nama}
adalah: Rp {gaji_bersih:.2f}")

def main():
    nama, jam_kerja, tarif_per_jam =
input_data()
    gaji_bersih = hitung_gaji(jam_kerja,
tarif_per_jam)
    tampilkan_hasil(nama, gaji_bersih)

main()
```

Penjelasan:

- Fungsi input_data digunakan untuk meminta input nama karyawan, jumlah jam kerja, dan tarif per jam, kemudian mengembalikan ketiga nilai tersebut.
- Fungsi hitung_gaji menerima parameter jam_kerja dan tarif_per_jam, kemudian menghitung gaji bersih berdasarkan jumlah jam kerja dan tarif per jam dengan mengaplikasikan tarif lembur 1.5 kali tarif normal jika jam kerja lebih dari 40 jam.

- Fungsi tampilkan_hasil menerima parameter nama dan gaji_bersih, kemudian mencetak hasil gaji bersih karyawan.
- Fungsi main digunakan untuk menggabungkan semua fungsi yang telah didefinisikan sebelumnya.
- Memanggil fungsi main untuk menjalankan aplikasi.

DAFTAR PUSTAKA

Lutz, M., 2013. Learning Python, 5th ed. O'Reilly Media, Inc.

Matthes, E., 2019. Python Crash Course, 2nd ed. No Starch Press.

Beazley, D., 2009. Python Essential Reference, 4th ed. Addison-Wesley.

Sweigart, A., 2019. Automate the Boring Stuff with Python, 2nd ed. No Starch Press.

Summerfield, M., 2009. Programming in Python 3, 2nd ed. Addison-Wesley.

Pilgrim, M., 2004. Dive Into Python. Apress.

Rossum, G. van, 1995. Python tutorial. Technical Report CS-R9526, Centrum voor Wiskunde en Informatica (CWI), Amsterdam.

Python Software Foundation, 2021. Python Documentation [online]. Available at: https://docs.python.org/3/ [Accessed 14 March 2023].

W3Schools, 2021. Python Tutorial [online]. Available at: https://www.w3schools.com/python/ [Accessed 03 March 2023]

Real Python, 2021. Python Tutorials [online]. Available at: https://realpython.com/ [Accessed 23 February 2023].

GeeksforGeeks, 2021. Python Programming Language [online]. Available at: https://www.geeksforgeeks.org/python-programming-language/ [Accessed 01 March 2023].

PEP 8, 2001. Style Guide for Python Code [online]. Available at:

https://www.python.org/dev/peps/pep-0008/ [Accessed 27 February 2023].
McKinney, W., 2017. Python for Data Analysis, 2nd ed. O'Reilly Media, Inc.
VanderPlas, J., 2016. Python Data Science Handbook. O'Reilly Media, Inc.
Saha, B.N., 2018. Python Algorithms, 2nd ed. Apress.