

Code Kickstart: Python Programming

Panduan Belajar Lengkap untuk Pemula

Daftar Isi

1. [Getting Started with Python](#)
2. [Variables & Data Types](#)
3. [Control Flow: If-Else](#)
4. [Loops & Patterns](#)
5. [Functions & Reusability](#)
6. [Lists & Dictionaries](#)
7. [File Handling & CSV](#)
8. [Hands-on Session: Python Project](#)

Tujuan Pembelajaran

Setelah menyelesaikan panduan ini, Anda akan mampu:

1. **Memahami logika dasar pemrograman** menggunakan Python dan mampu menulis program sederhana untuk mendukung ide inovasi IoT secara logis dan terstruktur.
2. **Pembelajaran kelas tentang program Python**, kondisional, perulangan, dan fungsi.
3. **Pembelajaran kelas tentang konsep variabel**, tipe data, dan struktur kontrol.
4. **Pengerjaan mini tugas peserta tentang kasus penggunaan Python** (kalkulator dasar, sistem pemungutan suara).
5. **Peserta menjembatankan ide ke dalam alur logika pemrograman.**

Tools yang Dibutuhkan

- **Python 3.x** - Bahasa pemrograman utama
- **Text Editor/IDE** - VS Code, PyCharm, atau editor lainnya
- **Terminal/Command Prompt** - Untuk menjalankan program

1. Getting Started with Python

Selamat datang di dunia Python! Bagian ini akan memandu Anda melalui langkah-langkah awal untuk memulai pemrograman dengan Python. Python adalah bahasa pemrograman tingkat tinggi yang sangat populer, mudah dipelajari, dan memiliki banyak kegunaan, mulai dari pengembangan web, analisis data, kecerdasan buatan, hingga otomatisasi.

Apa itu Python?

Python diciptakan oleh Guido van Rossum dan pertama kali dirilis pada tahun 1991. Filosofi desain Python menekankan keterbacaan kode dengan penggunaan indentasi yang signifikan. Ini membuatnya menjadi pilihan yang sangat baik untuk pemula.

Mengapa Belajar Python?

1. **Mudah Dipelajari:** Sintaksis Python yang sederhana dan mirip bahasa Inggris membuatnya mudah dipahami dan ditulis.
2. **Serbaguna:** Dapat digunakan untuk berbagai aplikasi, termasuk pengembangan web (Django, Flask), analisis data (Pandas, NumPy), machine learning (TensorFlow, PyTorch), otomatisasi skrip, dan banyak lagi.
3. **Komunitas Besar:** Memiliki komunitas yang sangat aktif, yang berarti banyak sumber daya, tutorial, dan dukungan tersedia secara online.
4. **Ekosistem Pustaka yang Kaya:** Ribuan pustaka (libraries) dan kerangka kerja (frameworks) tersedia untuk mempercepat pengembangan.

Instalasi Python

Langkah pertama adalah menginstal Python di komputer Anda. Ikuti panduan di bawah ini sesuai dengan sistem operasi Anda.

Untuk Windows:

1. Kunjungi situs resmi Python: python.org/downloads/
2. Unduh installer versi terbaru (disarankan Python 3.x).
3. Jalankan installer. **Sangat penting** untuk mencentang opsi "Add Python X.X to PATH" selama instalasi. Ini akan memudahkan Anda menjalankan Python dari Command Prompt.
4. Ikuti langkah-langkah instalasi hingga selesai.

Untuk macOS:

Python 2.x mungkin sudah terinstal secara default di macOS, tetapi kita akan menggunakan Python 3.x. Ikuti langkah-langkah berikut:

1. Menggunakan Homebrew (Disarankan):

- Buka Terminal (Applications > Utilities > Terminal).
- Instal Homebrew jika belum ada:

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

- Setelah Homebrew terinstal, instal Python 3: `brew install python`

2. Menggunakan Installer Resmi: Sama seperti Windows, unduh installer dari python.org/downloads/ dan ikuti instruksinya.

Untuk Linux (Ubuntu/Debian):

Sebagian besar distribusi Linux sudah memiliki Python terinstal. Anda bisa menginstal Python 3.x dengan perintah berikut:

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

Memverifikasi Instalasi

Setelah instalasi selesai, buka Terminal (macOS/Linux) atau Command Prompt (Windows) dan ketik perintah berikut untuk memverifikasi bahwa Python telah terinstal dengan benar:

```
python3 --version
```

Anda seharusnya melihat versi Python yang terinstal, misalnya `Python 3.9.7`.

Menjalankan Program Python Pertama Anda

Mari kita tulis program Python pertama kita. Ini adalah tradisi di dunia pemrograman untuk memulai dengan program "Hello, World!".

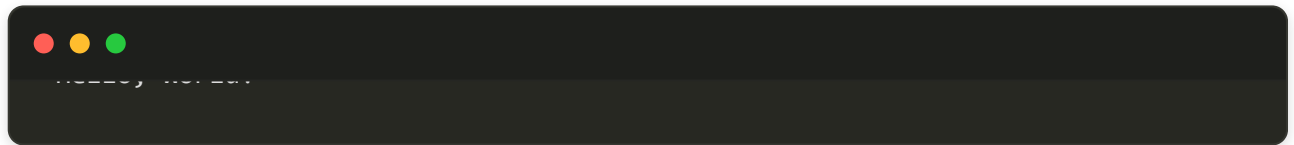
1. **Buka Text Editor:** Gunakan editor teks pilihan Anda (VS Code, Sublime Text, Notepad++, dll.).
2. **Tulis Kode:** Ketik baris kode berikut:

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

3. **Simpan File:** Simpan file dengan nama `hello.py` (ekstensi `.py` sangat penting) di lokasi yang mudah diakses, misalnya di folder `Documents` atau `Desktop`.
4. **Jalankan Program:** Buka Terminal atau Command Prompt, navigasikan ke direktori tempat Anda menyimpan file, lalu jalankan program dengan perintah:

```
python3 hello.py
```

Anda akan melihat output:



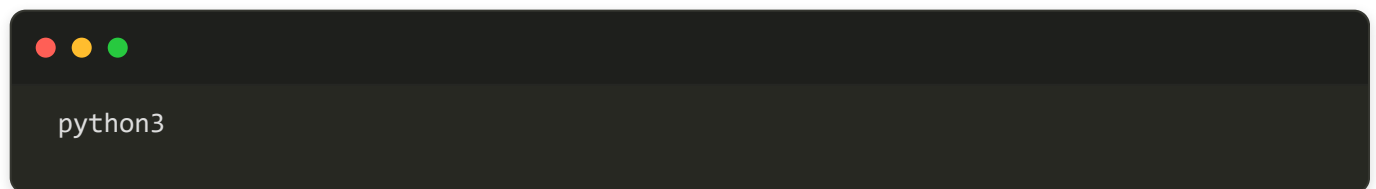
```
Halo, ini dari REPL!
```

Selamat! Anda telah berhasil menjalankan program Python pertama Anda.

Interaksi dengan Python Interpreter (REPL)

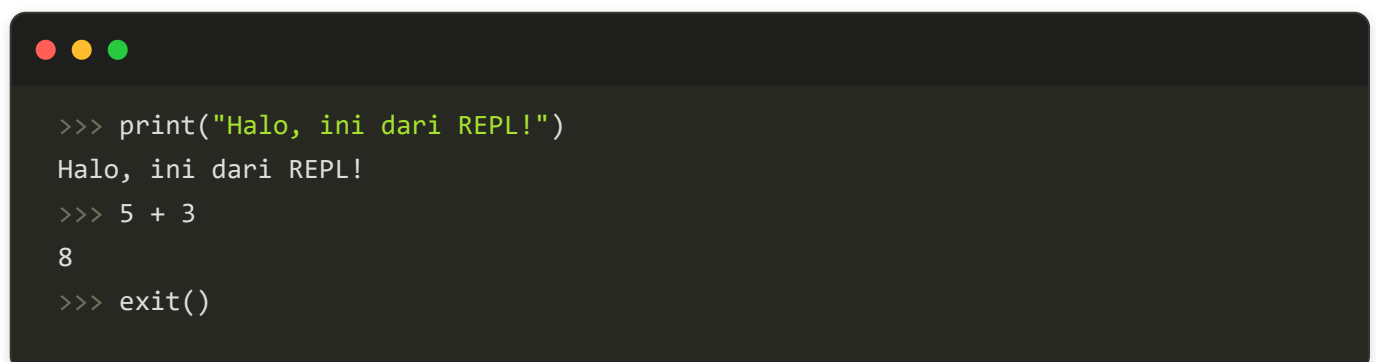
Python juga dilengkapi dengan interpreter interaktif yang disebut REPL (Read-Eval-Print Loop). Ini sangat berguna untuk menguji potongan kode kecil atau memahami bagaimana suatu fungsi bekerja.

Untuk masuk ke REPL, cukup ketik `python3` di Terminal/Command Prompt Anda:



```
python3
```

Anda akan melihat prompt `>>>`. Sekarang Anda bisa mengetik kode Python langsung di sini:



```
>>> print("Halo, ini dari REPL!")
Halo, ini dari REPL!
>>> 5 + 3
8
>>> exit()
```

Untuk keluar dari REPL, ketik `exit()` atau tekan `Ctrl+D` (Linux/macOS) atau `Ctrl+Z` lalu `Enter` (Windows).

Komentar dalam Python

Komentar adalah baris dalam kode yang diabaikan oleh interpreter Python. Komentar digunakan untuk menjelaskan kode, membuatnya lebih mudah dibaca dan dipahami oleh Anda sendiri atau orang lain.

Ada dua jenis komentar di Python:

1. **Komentar Baris Tunggal:** Dimulai dengan tanda `#`.

```
# Ini adalah komentar satu baris
print("Ini adalah kode") # Komentar juga bisa diletakkan setelah kode
```

2. **Komentar Multi-Baris (Docstrings):** Menggunakan tiga tanda kutip (`'''` atau `"""`). Ini sering digunakan untuk dokumentasi fungsi atau modul.

```
'''
Ini adalah komentar multi-baris.
Bisa mencakup beberapa baris.
'''

print("Hello")

"""
Ini juga komentar multi-baris.
Sering disebut docstring.
"""
```

Latihan Praktik

1. Instal Python di komputer Anda jika belum.
2. Buat file Python baru bernama `nama_anda.py` .
3. Tulis program yang mencetak nama Anda ke konsol.
4. Tambahkan komentar di atas baris `print()` yang menjelaskan apa yang dilakukan program tersebut.
5. Jalankan program dari Terminal/Command Prompt.
6. Coba gunakan Python REPL untuk melakukan beberapa perhitungan sederhana (misalnya, perkalian, pembagian).

2. Variables & Data Types

Dalam pemrograman, variabel adalah wadah untuk menyimpan data. Setiap variabel memiliki nama dan dapat menyimpan nilai dari berbagai tipe data. Memahami variabel dan tipe data adalah fundamental dalam menulis kode yang efektif dan efisien.

Apa itu Variabel?

Variabel adalah lokasi memori bernama yang digunakan untuk menyimpan nilai. Dalam Python, Anda tidak perlu mendeklarasikan tipe data variabel secara eksplisit; Python akan secara otomatis menentukan tipe data

berdasarkan nilai yang Anda berikan.

Aturan Penamaan Variabel:

1. **Dimulai dengan huruf atau underscore (_)**: Tidak boleh dimulai dengan angka.
2. **Hanya berisi karakter alfanumerik dan underscore**: (A-z, 0-9, dan _).
3. **Case-sensitive**: `nama` dan `Nama` adalah dua variabel yang berbeda.
4. **Tidak boleh menggunakan kata kunci Python**: Seperti `if`, `else`, `for`, `while`, dll.

Contoh Penamaan Variabel:

```
# Contoh yang benar
my_variable = 10
_private_var = "Ini rahasia"
angka1 = 20

# Contoh yang salah (akan menyebabkan error)
# 1variable = 5
# my-variable = "hello"
# if = 10
```

Memberikan Nilai ke Variabel

Anda memberikan nilai ke variabel menggunakan operator penugasan (=):

```
pesan = "Halo Dunia!" # Variabel string
umur = 30             # Variabel integer
harga = 19.99         # Variabel float
is_aktif = True       # Variabel boolean
```

Tipe Data dalam Python

Python memiliki beberapa tipe data bawaan yang sering digunakan:

1. **Numeric Types**: Untuk menyimpan nilai numerik.

- `int` (**Integer**): Bilangan bulat positif atau negatif, tanpa desimal. Contoh: `10`, `-5`, `1000`

- **float (Floating Point Number)**: Bilangan desimal. Contoh: `3.14`, `-0.5`, `2.0`.
- **complex (Complex Number)**: Bilangan kompleks (jarang digunakan dalam pemrograman umum). Contoh: `1 + 2j`.

```
bilangan_bulat = 100
bilangan_desimal = 25.75
```

2. Text Type: Untuk menyimpan urutan karakter.

- **str (String)**: Urutan karakter yang diapit oleh tanda kutip tunggal (`'`) atau ganda (`"`).

```
nama = "Budi"
salam = 'Selamat Pagi'
```

3. Boolean Type: Untuk menyimpan nilai kebenaran.

- **bool (Boolean)**: Hanya memiliki dua nilai: `True` atau `False`. Digunakan dalam logika kondisional.

```
apakah_hujan = False
apakah_benar = True
```

4. Sequence Types: Untuk menyimpan koleksi item yang berurutan.

- **list**: Koleksi item yang terurut dan dapat diubah (mutable). Item dapat memiliki tipe data yang berbeda. Ditulis dengan kurung siku `[]`.
- **tuple**: Koleksi item yang terurut dan tidak dapat diubah (immutable). Ditulis dengan kurung biasa `()`.
- **range**: Urutan angka yang tidak dapat diubah (sering digunakan dalam loop).

```
daftar_angka = [1, 2, 3, 4, 5]
nama_buah = ("apel", "pisang", "ceri")
```

5. **Mapping Type**: Untuk menyimpan koleksi pasangan kunci-nilai.

- **dict (Dictionary)**: Koleksi item yang tidak terurut, dapat diubah, dan diindeks. Setiap item memiliki kunci dan nilai. Ditulis dengan kurung kurawal `{}`.

```
data_siswa = {"nama": "Andi", "umur": 20, "jurusan": "Informatika"}
```

6. **Set Types**: Untuk menyimpan koleksi item yang tidak terurut dan tidak terindeks, tanpa duplikasi.

- **set**: Koleksi item yang tidak terurut dan tidak memiliki anggota duplikat. Ditulis dengan kurung kurawal `{}`.
- **frozenset**: Mirip dengan set, tetapi tidak dapat diubah (immutable).

```
himpunan_warna = {"merah", "biru", "hijau"}
```

Mengecek Tipe Data

Anda dapat menggunakan fungsi `type()` untuk mengetahui tipe data dari suatu variabel:

```
angka = 10
print(type(angka)) # Output: <class 'int'>

kalimat = "Belajar Python"
print(type(kalimat)) # Output: <class 'str'>

pi = 3.14
print(type(pi)) # Output: <class 'float'>

benar = True
print(type(benar)) # Output: <class 'bool'>

daftar = [1, 2, 3]
print(type(daftar)) # Output: <class 'list'>
```



```
kamus = {"a": 1, "b": 2}
print(type(kamus)) # Output: <class 'dict'>
```

Konversi Tipe Data (Type Casting)

Python memungkinkan Anda untuk mengkonversi satu tipe data ke tipe data lain menggunakan fungsi bawaan seperti `int()`, `float()`, `str()`, dll.

```
# Konversi dari int ke float
angka_int = 10
angka_float = float(angka_int)
print(angka_float) # Output: 10.0
print(type(angka_float)) # Output: <class 'float'>

# Konversi dari float ke int (akan memotong bagian desimal)
angka_float_2 = 15.7
angka_int_2 = int(angka_float_2)
print(angka_int_2) # Output: 15
print(type(angka_int_2)) # Output: <class 'int'>

# Konversi dari int/float ke string
angka_str = str(123)
print(angka_str) # Output: "123"
print(type(angka_str)) # Output: <class 'str'>

# Konversi dari string ke int/float (hati-hati, string harus berupa angka)
str_angka = "200"
int_dari_str = int(str_angka)
print(int_dari_str) # Output: 200
print(type(int_dari_str)) # Output: <class 'int'>

str_float = "99.5"
float_dari_str = float(str_float)
print(float_dari_str) # Output: 99.5
print(type(float_dari_str)) # Output: <class 'float'>

# Contoh error jika string tidak bisa dikonversi
# int("hello") # Akan menghasilkan ValueError
```

Latihan Praktik

1. Buat tiga variabel dengan nama `nama_depan`, `nama_belakang`, dan `umur`. Berikan nilai yang sesuai dengan tipe data `str` dan `int`.
2. Cetak nilai dan tipe data dari masing-masing variabel tersebut menggunakan fungsi `print()` dan `type()`.
3. Buat variabel `tinggi_badan` dengan nilai float (misalnya, `1.75`). Konversikan nilai ini ke integer dan cetak hasilnya.
4. Buat variabel `angka_string` dengan nilai string yang berisi angka (misalnya, `"42"`). Konversikan nilai ini ke integer dan lakukan operasi penjumlahan dengan angka lain (misalnya, `int(angka_string) + 10`). Cetak hasilnya.
5. Jelaskan mengapa `"hello"` tidak bisa dikonversi menjadi `int`.

3. Control Flow: If-Else

Kontrol alur (control flow) adalah konsep fundamental dalam pemrograman yang memungkinkan Anda untuk membuat keputusan dan menjalankan blok kode tertentu berdasarkan kondisi yang terpenuhi. Ini adalah bagaimana program Anda dapat "berpikir" dan merespons berbagai situasi.

Pernyataan `if`

Pernyataan `if` digunakan untuk menjalankan blok kode hanya jika suatu kondisi bernilai `True`. Sintaks dasarnya adalah:

```
if kondisi:
    # Blok kode yang akan dieksekusi jika kondisi True
    # Perhatikan indentasi (spasi atau tab) di sini
```

Contoh:

```
umur = 18

if umur >= 17:
    print("Anda cukup umur untuk memiliki KTP.")
```

Dalam contoh di atas, `umur >= 17` adalah kondisi. Jika kondisi ini `True` (yaitu, `umur` adalah 17 atau lebih), maka baris `print()` akan dieksekusi. Jika `umur` kurang dari 17, baris `print()` akan dilewati.

Pernyataan `if-else`

Pernyataan `if-else` memungkinkan Anda untuk menjalankan satu blok kode jika kondisi `True`, dan blok kode lain jika kondisi `False`. Sintaksnya adalah:

```
if kondisi:
    # Blok kode jika kondisi True
else:
    # Blok kode jika kondisi False
```

Contoh:

```
nilai = 75

if nilai >= 60:
    print("Anda lulus ujian.")
else:
    print("Anda tidak lulus ujian.")
```

Di sini, jika `nilai` 60 atau lebih, pesan "Anda lulus ujian." akan dicetak. Jika tidak, pesan "Anda tidak lulus ujian." yang akan dicetak.

Pernyataan `if-elif-else`

Ketika Anda memiliki beberapa kondisi yang perlu diperiksa secara berurutan, Anda dapat menggunakan pernyataan `if-elif-else`. `elif` adalah singkatan dari "else if". Python akan memeriksa kondisi secara berurutan, dan akan mengeksekusi blok kode pertama yang kondisinya `True`.

```
if kondisi1:
    # Blok kode jika kondisi1 True
elif kondisi2:
    # Blok kode jika kondisi2 True
elif kondisi3:
    # Blok kode jika kondisi3 True
```

```
else:  
    # Blok kode jika tidak ada kondisi di atas yang True
```

Contoh:

```
skor = 85  
  
if skor >= 90:  
    print("Nilai Anda A")  
elif skor >= 80:  
    print("Nilai Anda B")  
elif skor >= 70:  
    print("Nilai Anda C")  
elif skor >= 60:  
    print("Nilai Anda D")  
else:  
    print("Nilai Anda E")
```

Dalam contoh ini, Python akan memeriksa `skor >= 90`. Jika `False`, ia akan melanjutkan ke `skor >= 80`, dan seterusnya. Hanya satu blok kode yang akan dieksekusi.

Operator Perbandingan

Kondisi dalam pernyataan `if` biasanya melibatkan operator perbandingan:

Operator	Deskripsi	Contoh	Hasil
<code>==</code>	Sama dengan	<code>5 == 5</code>	<code>True</code>
<code>!=</code>	Tidak sama dengan	<code>5 != 3</code>	<code>True</code>
<code><</code>	Kurang dari	<code>5 < 3</code>	<code>False</code>
<code>></code>	Lebih dari	<code>5 > 3</code>	<code>True</code>
<code><=</code>	Kurang dari atau sama dengan	<code>5 <= 5</code>	<code>True</code>
<code>>=</code>	Lebih dari atau sama dengan	<code>5 >= 3</code>	<code>True</code>

Operator Logika

Anda dapat menggabungkan beberapa kondisi menggunakan operator logika:

- `and` : `True` jika kedua kondisi `True` .
- `or` : `True` jika salah satu kondisi `True` .
- `not` : Membalikkan nilai kebenaran suatu kondisi (`True` menjadi `False` , `False` menjadi `True`).

Contoh `and` :

```
umur = 25
punya_sim = True

if umur >= 18 and punya_sim == True:
    print("Anda boleh mengemudi.")
else:
    print("Anda belum boleh mengemudi.")
```

Contoh `or` :

```
hari = "Minggu"
cuaca = "Cerah"

if hari == "Sabtu" or hari == "Minggu":
    print("Ini akhir pekan!")
else:
    print("Ini hari kerja.")
```

Contoh `not` :

```
is_libur = False

if not is_libur:
    print("Hari ini bukan hari libur.")
```

Indentasi Penting!

Python menggunakan indentasi (spasi atau tab) untuk menentukan blok kode. Ini sangat penting dan berbeda dengan bahasa pemrograman lain yang mungkin menggunakan kurung kurawal `{ }`. Indentasi yang salah akan menyebabkan `IndentationError`.

```
# Contoh indentasi yang benar
if True:
    print("Ini di dalam blok if")
    print("Baris ini juga di dalam blok if")

# Contoh indentasi yang salah (akan error)
# if True:
# print("Ini salah indentasi")
```

Latihan Praktik

1. Buat program yang meminta pengguna memasukkan sebuah angka. Periksa apakah angka tersebut positif, negatif, atau nol, lalu cetak hasilnya.
2. Buat program yang meminta pengguna memasukkan dua angka. Cetak angka yang lebih besar.
3. Buat program yang meminta pengguna memasukkan usia dan apakah mereka memiliki tiket (`True` atau `False`). Jika usia di atas 18 DAN memiliki tiket, cetak "Selamat datang di konser!". Jika tidak, cetak "Maaf, Anda tidak bisa masuk."
4. Buat program yang menentukan apakah sebuah tahun adalah tahun kabisat. Tahun kabisat adalah tahun yang habis dibagi 4, kecuali jika habis dibagi 100 tetapi tidak habis dibagi 400. (Contoh: 2000 adalah kabisat, 1900 bukan, 2004 adalah).

4. Loops & Patterns

Perulangan (loops) adalah salah satu konsep paling kuat dalam pemrograman. Mereka memungkinkan Anda untuk menjalankan blok kode berulang kali, yang sangat berguna untuk mengotomatisasi tugas, memproses daftar data, atau membuat pola. Python menyediakan dua jenis perulangan utama: `for` loop dan `while` loop.

`for` Loop

`for` loop digunakan untuk mengulang item dalam urutan (seperti list, tuple, string, atau range) atau objek lain yang dapat diulang (iterable). Ini sangat cocok ketika Anda tahu berapa kali Anda ingin mengulang, atau ketika Anda ingin mengulang setiap item dalam koleksi.

Sintaks Dasar `for` Loop:

```
for item in iterable:
    # Blok kode yang akan dieksekusi untuk setiap item
```

Contoh 1: Mengulang List

```
buah = ["apel", "pisang", "ceri"]
for x in buah:
    print(x)
```

Output:

```

--
pisang
ceri
```

Contoh 2: Mengulang String

```
for karakter in "Python":
    print(karakter)
```

Output:

```

y
t
h
o
n
```

Fungsi `range()`

Fungsi `range()` sering digunakan dengan `for` loop untuk menghasilkan urutan angka. Ini sangat berguna ketika Anda ingin mengulang sejumlah kali tertentu.

- `range(stop)` : Menghasilkan angka dari 0 hingga `stop-1` .
- `range(start, stop)` : Menghasilkan angka dari `start` hingga `stop-1` .
- `range(start, stop, step)` : Menghasilkan angka dari `start` hingga `stop-1` dengan peningkatan `step` .

Contoh Penggunaan `range()` :

```
# Mengulang 5 kali (dari 0 sampai 4)
for i in range(5):
    print(i)
# Output: 0, 1, 2, 3, 4

# Mengulang dari 2 sampai 5 (2, 3, 4, 5)
for i in range(2, 6):
    print(i)
# Output: 2, 3, 4, 5

# Mengulang dari 0 sampai 9 dengan langkah 2 (0, 2, 4, 6, 8)
for i in range(0, 10, 2):
    print(i)
# Output: 0, 2, 4, 6, 8
```

`while` Loop

`while` loop digunakan untuk menjalankan blok kode selama suatu kondisi bernilai `True` . Perulangan akan terus berlanjut sampai kondisi menjadi `False` . Ini cocok ketika Anda tidak tahu berapa kali perulangan akan terjadi, tetapi Anda tahu kapan perulangan harus berhenti.

Sintaks Dasar `while` Loop:

```
while kondisi:
    # Blok kode yang akan dieksekusi selama kondisi True
```

Contoh:


```
count = 1
while count <= 5:
    print(count)
    count += 1 # Penting: pastikan kondisi berubah agar loop tidak tak terbatas
```

Output:

```
-
1
2
3
4
5
```

Peringatan: Pastikan kondisi dalam `while` loop pada akhirnya akan menjadi `False`. Jika tidak, Anda akan membuat *infinite loop* (perulangan tak terbatas) yang akan membuat program Anda macet.

Pernyataan `break` dan `continue`

Anda dapat mengontrol alur perulangan lebih lanjut menggunakan pernyataan `break` dan `continue`.

- `break`: Menghentikan perulangan sepenuhnya. Eksekusi akan dilanjutkan di baris setelah perulangan.
- `continue`: Melewatkan sisa iterasi saat ini dan melanjutkan ke iterasi berikutnya dari perulangan.

Contoh `break`:

```
for i in range(10):
    if i == 5:
        break # Hentikan loop ketika i adalah 5
    print(i)
```

Output:

```
-
1
2
```

```
3
4
```

Contoh `continue` :

```
for i in range(10):
    if i % 2 == 0: # Jika i adalah angka genap
        continue # Lewati iterasi ini
    print(i)
```

Output:

```
-
3
5
7
9
```

Nested Loops (Perulangan Bersarang)

Anda dapat menempatkan satu perulangan di dalam perulangan lain. Ini disebut perulangan bersarang dan sering digunakan untuk bekerja dengan struktur data dua dimensi (seperti matriks) atau untuk membuat pola.

Contoh: Mencetak Pola Bintang

```
for i in range(3): # Loop luar untuk baris
    for j in range(4): # Loop dalam untuk kolom
        print("*", end="") # Cetak bintang tanpa baris baru
    print() # Cetak baris baru setelah setiap baris bintang selesai
```

Output:

```
****
```

```
****
```

Membuat Pola dengan Perulangan

Perulangan bersarang sangat berguna untuk membuat pola-pola geometris sederhana menggunakan karakter. Mari kita coba membuat segitiga bintang.

Contoh: Segitiga Bintang Kanan

```
n = 5
for i in range(1, n + 1):
    for j in range(i):
        print("*", end="")
    print()
```

Output:

```
**
***
****
*****
```

Contoh: Segitiga Bintang Terbalik

```
n = 5
for i in range(n, 0, -1):
    for j in range(i):
        print("*", end="")
    print()
```

Output:

```
****
***
**
*
```

Latihan Praktik

1. Gunakan `for` loop untuk mencetak semua angka genap dari 0 hingga 20.
2. Gunakan `while` loop untuk menghitung jumlah digit dalam sebuah angka (misalnya, jika angka adalah 12345, hasilnya adalah 5).
3. Buat program yang meminta pengguna memasukkan kata. Gunakan `for` loop untuk mencetak setiap huruf dari kata tersebut, tetapi lewati huruf vokal (a, i, u, e, o) menggunakan `continue`.
4. Buat pola segitiga bintang terbalik menggunakan perulangan bersarang, tetapi kali ini dengan spasi di depannya sehingga membentuk segitiga yang rata kanan.

```
**
 ***
****
*****
```

5. Buat program yang mencetak tabel perkalian dari 1 hingga 10 menggunakan perulangan bersarang.

5. Functions & Reusability

Fungsi adalah blok kode yang terorganisir dan dapat digunakan kembali untuk melakukan satu tindakan terkait. Fungsi membantu memecah program Anda menjadi bagian-bagian yang lebih kecil dan modular, membuat kode lebih mudah dibaca, dipelihara, dan digunakan kembali. Konsep reusability (penggunaan kembali) adalah inti dari pemrograman yang efisien.

Mengapa Menggunakan Fungsi?

1. **Modularitas:** Memecah masalah besar menjadi bagian-bagian yang lebih kecil dan mudah dikelola.
2. **Reusability (Penggunaan Kembali):** Kode yang ditulis dalam fungsi dapat dipanggil berkali-kali dari berbagai tempat dalam program tanpa perlu menulis ulang.
3. **Keterbacaan:** Kode menjadi lebih terstruktur dan mudah dipahami.
4. **Debugging Lebih Mudah:** Jika ada bug, Anda tahu di mana mencarinya karena kode dibagi menjadi unit-unit yang lebih kecil.

Mendefinisikan Fungsi

Dalam Python, Anda mendefinisikan fungsi menggunakan kata kunci `def`, diikuti dengan nama fungsi, tanda kurung `()`, dan titik dua `:`. Blok kode fungsi diindentasi.

Sintaks Dasar:

```
def nama_fungsi():  
    # Blok kode fungsi  
    print("Halo dari fungsi!")
```

Memanggil Fungsi:

Setelah fungsi didefinisikan, Anda dapat memanggilnya dengan menggunakan nama fungsi diikuti dengan tanda kurung:

```
nama_fungsi() # Memanggil fungsi yang telah didefinisikan
```

Contoh:

```
def sapa():  
    print("Selamat pagi!")  
  
sapa() # Output: Selamat pagi!  
sapa() # Output: Selamat pagi!
```

Fungsi dengan Parameter

Fungsi dapat menerima input yang disebut parameter (atau argumen). Parameter adalah variabel yang tercantum di dalam tanda kurung saat mendefinisikan fungsi. Mereka memungkinkan fungsi untuk bekerja dengan data yang berbeda setiap kali dipanggil.

Sintaks:

```
def nama_fungsi(parameter1, parameter2):  
    # Blok kode fungsi yang menggunakan parameter
```

Contoh:

```
def sapa_nama(nama):  
    print(f"Halo, {nama}!")  
  
sapa_nama("Budi")    # Output: Halo, Budi!  
sapa_nama("Andi")    # Output: Halo, Andi!
```

Anda juga bisa memiliki beberapa parameter:

```
def tambah(angka1, angka2):  
    hasil = angka1 + angka2  
    print(f"Hasil penjumlahan: {hasil}")  
  
tambah(5, 3)    # Output: Hasil penjumlahan: 8  
tambah(10, 20) # Output: Hasil penjumlahan: 30
```

Fungsi dengan Nilai Kembali (`return`)

Fungsi tidak hanya dapat melakukan tindakan, tetapi juga dapat mengembalikan nilai menggunakan kata kunci `return`. Nilai yang dikembalikan dapat disimpan dalam variabel atau digunakan dalam ekspresi lain.

Sintaks:

```
def nama_fungsi(parameter):  
    # Lakukan sesuatu  
    return nilai_yang_dikembalikan
```

Contoh:

```
def kali(a, b):  
    return a * b  
  
hasil_kali = kali(4, 6)  
print(f"Hasil perkalian: {hasil_kali}") # Output: Hasil perkalian: 24  
  
print(kali(7, 2)) # Output: 14
```

Fungsi dapat mengembalikan berbagai tipe data, termasuk angka, string, boolean, list, dictionary, dll.

Argumen Default

Anda dapat memberikan nilai default untuk parameter. Jika argumen tidak diberikan saat memanggil fungsi, nilai default akan digunakan.

```
def sapa_kota(nama, kota="Jakarta"):  
    print(f"Halo {nama}, selamat datang di {kota}!")  
  
sapa_kota("Dina") # Output: Halo Dina, selamat datang di Jakarta!  
sapa_kota("Rudi", "Bandung") # Output: Halo Rudi, selamat datang di Bandung!
```

Argumen Kata Kunci (Keyword Arguments)

Anda dapat memanggil fungsi dengan argumen kata kunci, di mana Anda secara eksplisit menyebutkan nama parameter saat memanggil fungsi. Ini membuat kode lebih mudah dibaca dan memungkinkan Anda untuk melewati urutan argumen.

```
def info_siswa(nama, usia, jurusan):  
    print(f>Nama: {nama}, Usia: {usia}, Jurusan: {jurusan}")  
  
info_siswa(nama="Lisa", usia=21, jurusan="Desain")  
info_siswa(jurusan="Teknik", nama="Bayu", usia=22) # Urutan tidak masalah
```

Argumen Arbitrary (`*args` dan `**kwargs`)

Kadang-kadang Anda mungkin tidak tahu berapa banyak argumen yang akan dilewatkan ke fungsi. Python menyediakan sintaks khusus untuk menangani ini:

- `*args` (**Arbitrary Positional Arguments**): Mengumpulkan argumen posisi yang tidak ditentukan ke dalam sebuah tuple.
- `**kwargs` (**Arbitrary Keyword Arguments**): Mengumpulkan argumen kata kunci yang tidak ditentukan ke dalam sebuah dictionary.

Contoh `*args` :

```
def jumlahkan_semua(*angka):
    total = 0
    for num in angka:
        total += num
    return total

print(jumlahkan_semua(1, 2, 3))          # Output: 6
print(jumlahkan_semua(10, 20, 30, 40)) # Output: 100
```

Contoh `**kwargs` :

```
def tampilkan_info(**data):
    for kunci, nilai in data.items():
        print(f"{kunci}: {nilai}")

tampilkan_info(nama="Citra", kota="Surabaya")
# Output:
# nama: Citra
# kota: Surabaya

tampilkan_info(produk="Laptop", harga=1200, stok=50)
# Output:
# produk: Laptop
# harga: 1200
# stok: 50
```


Docstrings (Dokumentasi Fungsi)

Sangat disarankan untuk mendokumentasikan fungsi Anda menggunakan docstrings. Docstring adalah string multi-baris yang ditempatkan tepat setelah definisi fungsi. Ini menjelaskan apa yang dilakukan fungsi, parameternya, dan apa yang dikembalikannya.

```
def hitung_luas_persegi(panjang, lebar):  
    """  
    Fungsi ini menghitung luas persegi panjang.  
  
    Args:  
        panjang (int/float): Panjang sisi persegi panjang.  
        lebar (int/float): Lebar sisi persegi panjang.  
  
    Returns:  
        int/float: Luas persegi panjang.  
    """  
    luas = panjang * lebar  
    return luas  
  
# Anda bisa mengakses docstring dengan help() atau __doc__  
help(hitung_luas_persegi)  
print(hitung_luas_persegi.__doc__)
```

Latihan Praktik

1. Buat fungsi bernama `salam_personal` yang menerima satu parameter `nama` dan mencetak pesan "Halo, [nama]! Selamat belajar Python!". Panggil fungsi ini dengan nama Anda.
2. Buat fungsi bernama `hitung_rata_rata` yang menerima tiga parameter angka dan mengembalikan nilai rata-ratanya. Panggil fungsi ini dan cetak hasilnya.
3. Buat fungsi bernama `cek_genap_ganjil` yang menerima satu angka sebagai parameter. Fungsi ini harus mencetak "Genap" jika angka genap, dan "Ganjil" jika angka ganjil. (Petunjuk: gunakan operator modulo `%`).
4. Buat fungsi `hitung_diskon` yang menerima `harga_total` dan `persen_diskon` (dengan nilai default 10%). Fungsi ini harus mengembalikan harga setelah diskon. Uji fungsi dengan dan tanpa memberikan `persen_diskon`.
5. Buat fungsi `cetak_profil` yang menerima `nama`, `usia`, dan `kota` sebagai argumen kata kunci. Panggil fungsi ini dengan urutan argumen yang berbeda untuk menunjukkan fleksibilitas argumen kata kunci.

6. Buat fungsi `maksimum_dari_list` yang menerima sejumlah angka arbitrer (`*args`) dan mengembalikan angka terbesar di antara mereka. (Petunjuk: Anda bisa menggunakan fungsi `max()` bawaan Python).

6. Lists & Dictionaries

Python menyediakan berbagai struktur data bawaan yang sangat kuat dan fleksibel untuk menyimpan dan mengelola koleksi data. Dua di antaranya yang paling sering digunakan adalah List dan Dictionary. Memahami cara kerja keduanya sangat penting untuk memanipulasi data secara efektif dalam program Python Anda.

Lists (Daftar)

List adalah koleksi item yang terurut dan dapat diubah (mutable). Artinya, Anda dapat mengubah, menambah, atau menghapus item setelah list dibuat. Item dalam list diindeks, dimulai dari 0 untuk item pertama. List didefinisikan dengan menempatkan item-item di dalam kurung siku `[]`, dipisahkan oleh koma.

Membuat List:

```
# List kosong
list_kosong = []

# List dengan berbagai tipe data
angka = [1, 2, 3, 4, 5]
buah = ["apel", "pisang", "ceri"]
campur = [1, "hello", True, 3.14]
```

Mengakses Item List:

Anda dapat mengakses item list menggunakan indeksinya. Indeks dimulai dari 0.

```
my_list = ["a", "b", "c", "d", "e"]

print(my_list[0]) # Output: a (item pertama)
print(my_list[2]) # Output: c (item ketiga)

# Indeks negatif mengakses dari akhir list
print(my_list[-1]) # Output: e (item terakhir)
print(my_list[-2]) # Output: d (item kedua terakhir)
```

Slicing List:

Anda dapat mengambil sebagian dari list (sub-list) menggunakan slicing. Sintaksnya adalah

```
list[start:end:step] .
```

```
angka = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

print(angka[2:5])    # Output: [2, 3, 4] (dari indeks 2 hingga sebelum 5)
print(angka[:4])     # Output: [0, 1, 2, 3] (dari awal hingga sebelum 4)
print(angka[5:])     # Output: [5, 6, 7, 8, 9] (dari indeks 5 hingga akhir)
print(angka[:2])     # Output: [0, 2, 4, 6, 8] (setiap item kedua)
print(angka[::-1])   # Output: [9, 8, 7, 6, 5, 4, 3, 2, 1, 0] (membalik list)
```

Mengubah Item List:

Karena list bersifat mutable, Anda dapat mengubah nilai item tertentu:

```
buah = ["apel", "pisang", "ceri"]
buah[1] = "mangga" # Mengubah item di indeks 1
print(buah) # Output: ["apel", "mangga", "ceri"]
```

Menambah Item ke List:

- `append()` : Menambah item ke akhir list.
- `insert()` : Menambah item pada indeks tertentu.
- `extend()` : Menambah item dari iterable lain ke akhir list.

```
my_list = [1, 2, 3]
my_list.append(4)    # Menambah 4 ke akhir
print(my_list)       # Output: [1, 2, 3, 4]

my_list.insert(1, 99) # Menambah 99 di indeks 1
print(my_list)       # Output: [1, 99, 2, 3, 4]

list_lain = [5, 6]
```

```
my_list.extend(list_lain) # Menambah item dari list_lain
print(my_list)           # Output: [1, 99, 2, 3, 4, 5, 6]
```

Menghapus Item dari List:

- `remove()` : Menghapus item pertama yang cocok dengan nilai yang diberikan.
- `pop()` : Menghapus dan mengembalikan item pada indeks tertentu (atau item terakhir jika indeks tidak diberikan).
- `del` : Menghapus item pada indeks tertentu atau seluruh list.
- `clear()` : Menghapus semua item dari list.

```
my_list = [1, 2, 3, 2, 4]
my_list.remove(2) # Menghapus 2 yang pertama ditemukan
print(my_list)   # Output: [1, 3, 2, 4]

item_dihapus = my_list.pop(1) # Menghapus item di indeks 1 (nilai 3)
print(my_list)               # Output: [1, 2, 4]
print(item_dihapus)          # Output: 3

del my_list[0] # Menghapus item di indeks 0 (nilai 1)
print(my_list) # Output: [2, 4]

my_list.clear() # Mengosongkan list
print(my_list)  # Output: []
```

Metode List Lainnya:

- `len()` : Mengembalikan jumlah item dalam list.
- `count()` : Mengembalikan berapa kali suatu nilai muncul dalam list.
- `index()` : Mengembalikan indeks pertama dari nilai yang ditentukan.
- `sort()` : Mengurutkan list secara in-place.
- `sorted()` : Mengembalikan list baru yang diurutkan (tidak mengubah list asli).
- `reverse()` : Membalik urutan item dalam list secara in-place.

```
angka = [3, 1, 4, 1, 5, 9, 2]
print(len(angka)) # Output: 7
```

```
print(angka.count(1)) # Output: 2
print(angka.index(5)) # Output: 4

angka.sort()
print(angka)          # Output: [1, 1, 2, 3, 4, 5, 9]

angka.reverse()
print(angka)          # Output: [9, 5, 4, 3, 2, 1, 1]
```

Dictionaries (Kamus)

Dictionary adalah koleksi item yang tidak terurut, dapat diubah (mutable), dan diindeks oleh kunci (keys). Berbeda dengan list yang diindeks oleh angka, dictionary diindeks oleh kunci yang unik. Setiap item dalam dictionary adalah pasangan kunci-nilai (key-value pair). Dictionary didefinisikan dengan menempatkan item-item di dalam kurung kurawal {}, dengan pasangan kunci-nilai dipisahkan oleh koma, dan kunci serta nilai dipisahkan oleh titik dua :.

Membuat Dictionary:

```
# Dictionary kosong
dict_kosong = {}

# Dictionary dengan data
person = {
    "nama": "Alice",
    "umur": 30,
    "kota": "New York"
}

# Kunci harus unik, nilai bisa duplikat
barang = {"item": "Laptop", "harga": 1200, "stok": 50}
```

Mengakses Item Dictionary:

Anda dapat mengakses nilai menggunakan kuncinya:

```
person = {"nama": "Alice", "umur": 30, "kota": "New York"}
```

```
print(person["nama"]) # Output: Alice
print(person["umur"]) # Output: 30

# Menggunakan metode get() untuk menghindari KeyError jika kunci tidak ada
print(person.get("kota")) # Output: New York
print(person.get("pekerjaan")) # Output: None (tidak error)
print(person.get("pekerjaan", "Tidak diketahui")) # Output: Tidak diketahui
```

Mengubah dan Menambah Item Dictionary:

Anda dapat mengubah nilai item dengan mengakses kuncinya. Jika kunci tidak ada, item baru akan ditambahkan.

```
person = {"nama": "Alice", "umur": 30, "kota": "New York"}

person["umur"] = 31 # Mengubah nilai kunci "umur"
print(person) # Output: {"nama": "Alice", "umur": 31, "kota": "New York"}

person["pekerjaan"] = "Engineer" # Menambah item baru
print(person) # Output: {"nama": "Alice", "umur": 31, "kota": "New York", "pekerjaan": "Engineer"}
```

Menghapus Item dari Dictionary:

- `pop()` : Menghapus item dengan kunci yang ditentukan dan mengembalikan nilainya.
- `popitem()` : Menghapus dan mengembalikan pasangan kunci-nilai terakhir yang dimasukkan (sebelum Python 3.7, menghapus item acak).
- `del` : Menghapus item dengan kunci yang ditentukan atau seluruh dictionary.
- `clear()` : Menghapus semua item dari dictionary.

```
person = {"nama": "Alice", "umur": 31, "kota": "New York", "pekerjaan": "Engineer"}

pekerjaan_dihapus = person.pop("pekerjaan")
print(person) # Output: {"nama": "Alice", "umur": 31, "kota": "New York"}
print(pekerjaan_dihapus) # Output: Engineer

del person["kota"]
print(person) # Output: {"nama": "Alice", "umur": 31}
```

```
person.clear()
print(person) # Output: {}
```

Metode Dictionary Lainnya:

- `len()` : Mengembalikan jumlah pasangan kunci-nilai.
- `keys()` : Mengembalikan objek view dari semua kunci.
- `values()` : Mengembalikan objek view dari semua nilai.
- `items()` : Mengembalikan objek view dari semua pasangan kunci-nilai (sebagai tuple).

```
barang = {"item": "Laptop", "harga": 1200, "stok": 50}

print(len(barang))      # Output: 3
print(barang.keys())    # Output: dict_keys(["item", "harga", "stok"])
print(barang.values())  # Output: dict_values(["Laptop", 1200, 50])
print(barang.items())   # Output: dict_items([("item", "Laptop"), ("harga", 1200), ("stok", 50)])

# Iterasi melalui dictionary
for kunci, nilai in barang.items():
    print(f"{kunci}: {nilai}")
```

Latihan Praktik

1. Buat sebuah list bernama `daftar_belanja` yang berisi beberapa item belanja (misalnya, "telur", "susu", "roti").
 - Tambahkan "keju" ke akhir list.
 - Sisipkan "mentega" di indeks kedua.
 - Cetak item pertama dan item terakhir dari list.
 - Ubah "susu" menjadi "yogurt".
 - Hapus "roti" dari list.
 - Cetak seluruh `daftar_belanja` setelah semua operasi.
2. Buat sebuah dictionary bernama `data_buku` yang menyimpan informasi tentang sebuah buku dengan kunci `judul`, `penulis`, dan `tahun_terbit`.
 - Akses dan cetak nilai dari kunci `penulis`.
 - Tambahkan kunci baru `genre` dengan nilai "Fiksi Ilmiah".
 - Ubah nilai `tahun_terbit` menjadi tahun saat ini.
 - Hapus kunci `genre` dari dictionary.

- Cetak semua kunci dan nilai dalam `data_buku` menggunakan loop.
3. Buat sebuah list berisi dictionary, di mana setiap dictionary merepresentasikan seorang siswa dengan kunci `nama` dan `nilai`. Hitung rata-rata nilai semua siswa.

```
siswa = [  
    {"nama": "Ani", "nilai": 85},  
    {"nama": "Budi", "nilai": 90},  
    {"nama": "Citra", "nilai": 78}  
]
```

7. File Handling & CSV

Kemampuan untuk membaca dan menulis file adalah aspek penting dalam banyak aplikasi Python. Ini memungkinkan program Anda untuk berinteraksi dengan data yang disimpan secara permanen di disk, bukan hanya data yang ada di memori saat program berjalan. Bagian ini akan membahas cara dasar menangani file teks dan file CSV (Comma Separated Values) di Python.

Membuka dan Menutup File

Untuk bekerja dengan file, langkah pertama adalah membukanya. Fungsi `open()` digunakan untuk membuka file, dan ia mengembalikan objek file. Penting untuk selalu menutup file setelah selesai menggunakannya untuk membebaskan sumber daya sistem dan memastikan semua perubahan disimpan.

Sintaks `open()` :

```
file_object = open("nama_file.txt", "mode")
```

Mode Pembukaan File:

- `"r"` (read): Membuka file untuk dibaca (default). File harus sudah ada.
- `"w"` (write): Membuka file untuk ditulis. Jika file sudah ada, isinya akan dihapus. Jika tidak ada, file baru akan dibuat.
- `"a"` (append): Membuka file untuk ditambahkan. Data baru akan ditulis di akhir file. Jika file tidak ada, file baru akan dibuat.
- `"x"` (create): Membuat file baru. Jika file sudah ada, operasi akan gagal.
- `"t"` (text): Mode teks (default). Digunakan untuk file teks.

- `"b"` (binary): Mode biner. Digunakan untuk file non-teks seperti gambar atau video.

Menutup File:

Setelah selesai, gunakan metode `close()` :

```
file = open("contoh.txt", "r")
# Lakukan operasi file
file.close()
```

Menggunakan `with` Statement (Disarankan)

Cara terbaik untuk menangani file di Python adalah dengan menggunakan `with` statement. Ini memastikan bahwa file akan secara otomatis ditutup, bahkan jika terjadi error selama operasi file. Ini adalah praktik terbaik karena Anda tidak perlu secara eksplisit memanggil `close()` .

```
with open("contoh.txt", "r") as file:
    # Lakukan operasi file di sini
    pass # Placeholder
# File secara otomatis ditutup setelah blok 'with' selesai
```

Membaca File Teks

Ada beberapa cara untuk membaca konten dari file teks:

1. `read()` : Membaca seluruh konten file sebagai satu string.

```
with open("data.txt", "r") as file:
    konten = file.read()
    print(konten)
```

2. `readline()` : Membaca satu baris dari file.

```
with open("data.txt", "r") as file:
    baris1 = file.readline()
    baris2 = file.readline()
    print(baris1)
    print(baris2)
```

3. `readlines()` : Membaca semua baris dari file dan mengembalikannya sebagai list string, di mana setiap string adalah satu baris.

```
with open("data.txt", "r") as file:
    semua_baris = file.readlines()
    for baris in semua_baris:
        print(baris.strip()) # strip() untuk menghapus karakter newline
```

4. **Iterasi Langsung pada Objek File (Paling Efisien)**: Mengulang baris demi baris.

```
with open("data.txt", "r") as file:
    for baris in file:
        print(baris.strip())
```

Menulis ke File Teks

1. `write()` : Menulis string ke file. Anda harus secara eksplisit menambahkan karakter newline (`\n`) jika Anda ingin setiap tulisan berada di baris baru.

```
with open("output.txt", "w") as file:
    file.write("Baris pertama.\n")
    file.write("Baris kedua.\n")
```

2. `writelines()` : Menulis list string ke file. Setiap string dalam list dianggap sebagai satu baris, tetapi Anda masih perlu menambahkan `\n` secara manual jika ingin baris baru.

```
daftar_baris = ["Ini baris A.\n", "Ini baris B.\n", "Ini baris C.\n"]
with open("output_list.txt", "w") as file:
    file.writelines(daftar_baris)
```

Menangani File CSV (Comma Separated Values)

File CSV adalah format file teks sederhana yang digunakan untuk menyimpan data tabular, di mana setiap baris adalah catatan data dan setiap bidang dipisahkan oleh koma (atau delimiter lain seperti titik koma atau tab).

Python memiliki modul bawaan bernama `csv` yang sangat memudahkan pekerjaan dengan file CSV.

Membaca File CSV:

```
import csv

with open("data.csv", "r") as file:
    reader = csv.reader(file) # Membuat objek reader
    header = next(reader)     # Membaca baris header
    print(f"Header: {header}")
    for row in reader:
        print(row)
```

Jika `data.csv` berisi:

```
nama,umur,kota
Andi,25,Jakarta
Budi,30,Bandung
Citra,22,Surabaya
```

Output akan terlihat seperti:

```
header = ['nama', 'umur', 'kota']
['Andi', '25', 'Jakarta']
```

```
['Budi', '30', 'Bandung']  
['Citra', '22', 'Surabaya']
```

Membaca CSV sebagai Dictionary (DictReader):

`csv.DictReader` sangat berguna karena membaca setiap baris sebagai dictionary, di mana kunci adalah nama kolom dari baris header.

```
import csv  
  
with open("data.csv", "r") as file:  
    reader = csv.DictReader(file)  
    for row in reader:  
        print(f>Nama: {row['Nama']], Usia: {row['Usia']], Kota: {row['Kota']})
```

Output:

```
Nama: Budi, Usia: 30, Kota: Bandung  
Nama: Citra, Usia: 22, Kota: Surabaya
```

Menulis ke File CSV:

```
import csv  
  
data = [  
    ["Nama", "Usia", "Kota"], # Header  
    ["Dina", 28, "Yogyakarta"],  
    ["Eko", 35, "Semarang"]  
]  
  
with open("output.csv", "w", newline="") as file:  
    writer = csv.writer(file) # Membuat objek writer  
    writer.writerows(data)    # Menulis semua baris
```

```
print("Data berhasil ditulis ke output.csv")
```

Penting: Gunakan `newline=""` saat membuka file CSV untuk menulis. Ini mencegah baris kosong tambahan muncul di file CSV Anda, yang bisa terjadi karena cara Python menangani newline secara default.

Menulis CSV dari Dictionary (DictWriter):

```
import csv

fieldnames = ["Nama", "Usia", "Kota"]
data_dict = [
    {"Nama": "Fajar", "Usia": 40, "Kota": "Medan"},
    {"Nama": "Gita", "Usia": 29, "Kota": "Makassar"}
]

with open("output_dict.csv", "w", newline="") as file:
    writer = csv.DictWriter(file, fieldnames=fieldnames)
    writer.writeheader() # Menulis baris header
    writer.writerows(data_dict)

print("Data dictionary berhasil ditulis ke output_dict.csv")
```

Latihan Praktik

1. Buat file teks baru bernama `catatan.txt` dan tulis beberapa baris teks ke dalamnya (misalnya, daftar tugas harian Anda).
2. Baca kembali `catatan.txt` dan cetak setiap baris ke konsol.
3. Buat file CSV baru bernama `produk.csv` dengan header `ID,Nama Produk,Harga,Stok` dan tambahkan setidaknya tiga baris data produk.
4. Baca `produk.csv` menggunakan `csv.DictReader` dan cetak nama produk dan harganya untuk setiap produk.
5. Buat program yang membaca file `data.csv` (yang Anda buat di contoh) dan menulis ulang hanya nama dan usia ke file CSV baru bernama `nama_usia.csv`.

8. Hands-on Session: Python Project

Selamat! Anda telah mempelajari dasar-dasar Python, termasuk variabel, tipe data, kontrol alur, perulangan, fungsi, list, dictionary, dan penanganan file. Sekarang saatnya untuk menerapkan pengetahuan ini dalam

proyek praktis. Sesi hands-on ini akan memandu Anda membangun dua proyek mini: Kalkulator Sederhana dan Sistem Pemungutan Suara.

Proyek-proyek ini dirancang untuk membantu Anda mengkonsolidasikan pemahaman Anda tentang konsep-konsep yang telah dipelajari dan melatih kemampuan Anda dalam menerjemahkan ide ke dalam alur logika pemrograman.

Proyek 1: Kalkulator Sederhana

Kita akan membangun kalkulator berbasis teks yang dapat melakukan operasi dasar: penjumlahan, pengurangan, perkalian, dan pembagian. Proyek ini akan melibatkan penggunaan fungsi, kontrol alur (`if-elif-else`), dan input/output pengguna.

Tujuan Proyek:

- Memahami bagaimana mengorganisir kode menggunakan fungsi.
- Menerapkan logika kondisional untuk memilih operasi.
- Menangani input pengguna dan konversi tipe data.
- Melakukan operasi matematika dasar.

Langkah-langkah Implementasi:

1. **Definisikan Fungsi Operasi:** Buat empat fungsi terpisah untuk penjumlahan, pengurangan, perkalian, dan pembagian.
2. **Tampilkan Menu Operasi:** Berikan pilihan operasi kepada pengguna.
3. **Ambil Input Pengguna:** Minta pengguna untuk memilih operasi dan memasukkan dua angka.
4. **Lakukan Perhitungan:** Panggil fungsi yang sesuai berdasarkan pilihan pengguna.
5. **Tampilkan Hasil:** Cetak hasil perhitungan.
6. **Penanganan Error (Opsional tapi Disarankan):** Tangani kasus pembagian dengan nol atau input yang tidak valid.

Kode Kalkulator Sederhana:

```
def tambah(x, y):  
    """Menjumlahkan dua angka."""  
    return x + y  
  
def kurang(x, y):  
    """Mengurangi dua angka."""  
    return x - y  
  
def kali(x, y):  
    """Mengalikan dua angka."""
```

```
    return x * y

def bagi(x, y):
    """Membagi dua angka. Menangani pembagian dengan nol."""
    if y == 0:
        return "Error: Tidak bisa dibagi dengan nol!"
    else:
        return x / y

def kalkulator():
    print("\n--- Kalkulator Sederhana ---")
    print("Pilih operasi:")
    print("1. Penjumlahan")
    print("2. Pengurangan")
    print("3. Perkalian")
    print("4. Pembagian")
    print("5. Keluar")

    while True:
        pilihan = input("Masukkan pilihan (1/2/3/4/5): ")

        if pilihan in ("1", "2", "3", "4"):
            try:
                num1 = float(input("Masukkan angka pertama: "))
                num2 = float(input("Masukkan angka kedua: "))
            except ValueError:
                print("Input tidak valid. Harap masukkan angka.")
                continue

            if pilihan == "1":
                print(f"{num1} + {num2} = {tambah(num1, num2)}")
            elif pilihan == "2":
                print(f"{num1} - {num2} = {kurang(num1, num2)}")
            elif pilihan == "3":
                print(f"{num1} * {num2} = {kali(num1, num2)}")
            elif pilihan == "4":
                hasil_bagi = bagi(num1, num2)
                print(f"{num1} / {num2} = {hasil_bagi}")
            elif pilihan == "5":
                print("Terima kasih telah menggunakan kalkulator.")
                break
        else:
            print("Pilihan tidak valid. Silakan coba lagi.")
```

```
# Panggil fungsi kalkulator untuk memulai
# kalkulator()
```

Cara Menjalankan:

1. Salin kode di atas ke file Python baru (misalnya, `kalkulator.py`).
2. Hapus tanda komentar pada baris `kalkulator()` di bagian paling bawah file.
3. Jalankan dari terminal: `python3 kalkulator.py`

Proyek 2: Sistem Pemungutan Suara Sederhana

Proyek ini akan mensimulasikan sistem pemungutan suara sederhana di mana pengguna dapat memilih dari beberapa kandidat. Kita akan menggunakan dictionary untuk menyimpan jumlah suara dan list untuk kandidat. Proyek ini akan melatih penggunaan loop, dictionary, dan penanganan input pengguna yang lebih kompleks.

Tujuan Proyek:

- Menggunakan dictionary untuk melacak data (jumlah suara).
- Menerapkan perulangan (`while` loop) untuk proses pemungutan suara.
- Menangani input pengguna dan validasi.
- Menampilkan hasil akhir pemungutan suara.

Langkah-langkah Implementasi:

1. **Definisikan Kandidat:** Buat list kandidat yang tersedia.
2. **Inisialisasi Suara:** Gunakan dictionary untuk menyimpan jumlah suara awal untuk setiap kandidat (mulai dari 0).
3. **Proses Pemungutan Suara:** Dalam loop, tampilkan kandidat, minta pengguna untuk memilih, dan perbarui jumlah suara.
4. **Validasi Input:** Pastikan pengguna memilih kandidat yang valid.
5. **Tampilkan Hasil:** Setelah pemungutan suara selesai (misalnya, pengguna memilih untuk berhenti), tampilkan hasil akhir.

Kode Sistem Pemungutan Suara:

```
def sistem_pemungutan_suara():
    print("\n--- Sistem Pemungutan Suara Sederhana ---")
    kandidat = ["Alice", "Bob", "Charlie"]
    suara = {kandidat_nama: 0 for kandidat_nama in kandidat} # Inisialisasi suara

    while True:
```



```
print("\nKandidat:")
for i, nama_kandidat in enumerate(kandidat):
    print(f"{i + 1}. {nama_kandidat}")
print("0. Selesai Pemungutan Suara")

pilihan = input("Masukkan nomor kandidat pilihan Anda (atau 0 untuk selesai)

if pilihan == "0":
    print("Pemungutan suara selesai.")
    break

try:
    pilihan_int = int(pilihan)
    if 1 <= pilihan_int <= len(kandidat):
        nama_kandidat_terpilih = kandidat[pilihan_int - 1]
        suara[nama_kandidat_terpilih] += 1
        print(f"Terima kasih! Anda memilih {nama_kandidat_terpilih}.")
    else:
        print("Nomor kandidat tidak valid. Silakan coba lagi.")
except ValueError:
    print("Input tidak valid. Harap masukkan nomor.")

print("\n--- Hasil Pemungutan Suara ---")
total_suara = sum(suara.values())
if total_suara == 0:
    print("Belum ada suara yang masuk.")
else:
    for nama_kandidat, jumlah_suara in suara.items():
        persentase = (jumlah_suara / total_suara) * 100
        print(f"{nama_kandidat}: {jumlah_suara} suara ({persentase:.2f}%)")

# Menentukan pemenang
if total_suara > 0:
    pemenang = max(suara, key=suara.get)
    print(f"\nPemenang adalah: {pemenang} dengan {suara[pemenang]} suara!")

# Panggil fungsi sistem_pemungutan_suara untuk memulai
# sistem_pemungutan_suara()
```

Cara Menjalankan:

1. Salin kode di atas ke file Python baru (misalnya, `voting.py`).
2. Hapus tanda komentar pada baris `sistem_pemungutan_suara()` di bagian paling bawah file.

3. Jalankan dari terminal: `python3 voting.py`

Latihan Tambahan (Pengembangan Proyek):

- **Kalkulator:** Tambahkan operasi modulus (`%`) atau pangkat (`**`). Implementasikan fitur untuk melanjutkan perhitungan dengan hasil sebelumnya.
- **Sistem Pemungutan Suara:** Tambahkan fitur untuk menyimpan hasil pemungutan suara ke file CSV. Implementasikan batas jumlah suara per pengguna (misalnya, setiap pengguna hanya bisa memilih sekali).

Selamat mencoba proyek-proyek ini! Ingat, praktik adalah kunci untuk menguasai pemrograman. Jangan ragu untuk memodifikasi dan bereksperimen dengan kode ini.
