

**RESUME PRAKTIKUM  
PEMROGRAMAN BERORIENTASI OBJEK  
RB**

**Oleh :**

**Ilham Yoga Pratama                      (121140081)**



**Program Studi Teknik Informatika**

**Institut Teknologi Sumatera**

**2023**

## Daftar Isi

<b>1. Pengenalan Bahasa Pemrograman Python .....</b>	<b>3</b>
<b>1.1 Pengenalan Bahasa Python .....</b>	<b>3</b>
<b>1.2 Dasar Bahasa Python.....</b>	<b>3</b>
<b>2. Konsep Object-Oriented Programming .....</b>	<b>10</b>
<b>2.1 Abstraksi.....</b>	<b>10</b>
<b>2.2 Enkapsulasi .....</b>	<b>11</b>
<b>2.3 Inheritance .....</b>	<b>12</b>
<b>2.4 Polymorphism .....</b>	<b>12</b>

# 1. Pengenalan Bahasa Pemrograman Python

## 1.1 Pengenalan Bahasa Python

Bahasa pemrograman python dibuat oleh Guido Van Rossum pada tahun 1980-an akhir di Centrum Wiskunde & Informatica Belanda. Python mendukung banyak paradigma pemrograman seperti object-oriented, functional dan structured. Bahasa Python menjadi populer sekarang ini dikarenakan sintaks nya yang mudah, didukung oleh library(modul) yang berlimpah dan Python bisa dipakai untuk pemrograman desktop maupun mobile, CLI, GUI, web, otomatisasi, hacking, IoT, robotika, dan lain sebagainya.

## 1.2 Dasar Bahasa Python

### a. Sintaks Dasar

- Statement

Semua perintah yang bisa dieksekusi Python disebut statement. Pada Python akhir dari sebuah statement adalah baris baru (newline) tapi dimungkinkan membuat statement yang terdiri dari beberapa baris menggunakan backslash (\).

- Baris dan Indentasi

Python tidak menggunakan kurung kurawal sebagai grouping blok kode melainkan menggunakan spasi ataupun tab (4 spasi). kode yang berada di blok yang sama harus memiliki jumlah spasi yang sama di awal.

### b. Variabel dan Tipe Data Primitif

Variabel merupakan lokasi penyimpanan yang berguna untuk menyimpan suatu data atau suatu nilai. Dalam mendeklarasikan suatu variabel dalam pemrograman, perlu diketahui tipe-tipe data yang berhubungan dengan variabel yang akan dideklarasikan.

### c. Operator

Python memiliki sejumlah operator, yaitu :

- Operator Aritmatika

Operator aritmatika adalah operator yang digunakan untuk melakukan operasi matematika, seperti penjumlahan, pengurangan, perkalian, pembagian, dan sebagainya. Tabel berikut menunjukkan jenis operator aritmatika:

Operator	Nama dan Fungsi	Contoh
+	Penjumlahan, menjumlahkan 2 buah operand	$x + y$
-	Pengurangan, mengurangi 2 buah operand	$x - y$
*	Perkalian, mengalikan 2 buah operand	$x * y$
/	Pembagian, membagi 2 buah operand	$x / y$
**	Pemangkatan, memangkatkan bilangan	$x ** y$
//	Pembagian bulat, menghasilkan bagi tanpa koma	$x // y$
%	Modulus, menghasilkan sisa pembagian 2 bilangan	$x \% y$

- Operator Perbandingan

Operator perbandingan adalah operator yang digunakan untuk membandingkan 2 buah nilai. Hasil perbandingannya adalah True atau False tergantung kondisi. Tabel berikut menunjukkan jenis operator perbandingan:

Operator	Nama dan Fungsi	Contoh
>	Lebih besar dari (hasilnya true jika nilai sebelah kiri lebih besar dari nilai sebelah kanan)	$x > y$
<	Lebih kecil dari (hasilnya true jika nilai sebelah kiri lebih kecil dari nilai sebelah kanan)	$x < y$
==	Sama dengan (hasilnya true jika nilai sebelah kiri sama dengan dari nilai sebelah kanan)	$x == y$
!=	Tidak sama dengan (hasilnya true jika nilai sebelah kiri tidak sama dengan dari nilai sebelah kanan)	$x != y$
>=	Lebih besar atau sama dengan (hasilnya true jika nilai sebelah kiri lebih besar atau sama dengan dari nilai sebelah kanan)	$x >= y$
<=	Lebih kecil atau sama dengan (hasilnya true jika nilai sebelah kiri lebih kecil atau sama dengan dari nilai sebelah kanan)	$x <= y$

- Operator Penugasan

Operator penugasan adalah operator yang digunakan untuk memberi nilai ke variabel.  $a = 7$  adalah contoh operator penugasan yang memberi nilai 7 di kanan ke variabel a yang ada di kiri. Tabel berikut menunjukkan jenis operator penugasan:

Operator	Penjelasan	Contoh
=	Menugaskan nilai yang ada di kanan ke operand di sebelah kiri	$c = a + b$ menugaskan $a + b$ ke c
+=	Menambahkan operand yang di kanan dengan operand yang ada di kiri dan hasilnya ditugaskan ke operand yang di kiri	$c += a$ sama dengan $c = c + a$
-=	Mengurangi operand yang di kanan dengan operand yang ada di kiri dan hasilnya ditugaskan ke operand yang di kiri	$c -= a$ sama dengan $c = c - a$
*=	Mengalikan operand yang di kanan dengan operand yang ada di kiri dan hasilnya ditugaskan ke operand yang di kiri	$c *= a$ sama dengan $c = c * a$
/=	Membagi operand yang di kanan dengan operand yang ada	$c /= a$ sama dengan $c = c / a$

	di kiri dan hasilnya ditugaskan ke operand yang di kiri	
<code>**=</code>	Memangkatkan operand yang di kanan dengan operand yang ada di kiri dan hasilnya ditugaskan ke operand yang di kiri	$c ** = a$ sama dengan $c = c ** a$
<code>//=</code>	Melakukan pembagian bulat operand di kanan terhadap operand di kiri dan hasilnya disimpan di operand yang di kiri	$c // = a$ sama dengan $c = c // a$
<code>%=</code>	Melakukan operasi sisa bagi operand di kanan dengan operand di kiri dan hasilnya disimpan di operand yang di kiri	$c \% = a$ sama dengan $c = c \% a$

- Operator Logika

Operator logika adalah operator yang digunakan untuk melakukan operasi logika. Tabel berikut menunjukkan jenis operator logika:

Operator	Penjelasan	Contoh
And	Hasilnya adalah true jika kedua operandnya bernilai benar	$x \text{ and } y$
Or	Hasilnya adalah true jika salah satu atau kedua operandnya bernilai benar	$x \text{ or } y$
Not	Hasilnya adalah true jika operandnya bernilai salah (kebalikan nilai)	$\text{not } x$

- Operator Bitwise

Operator bitwise adalah operator yang melakukan operasi bit terhadap operand. Operator ini beroperasi bit per bit sesuai dengan namanya. Tabel berikut menunjukkan jenis operator bitwise:

Operator	Nama	Contoh
<code>&amp;</code>	Bitwise AND	$x \& y = 0$ (0000 0000)
<code> </code>	Bitwise OR	$x   y = 14$ (0000 1110)
<code>~</code>	Bitwise NOT	$\sim x = -11$ (1111 0101)
<code>^</code>	Bitwise XOR	$x \wedge y = 14$ (0000 1110)
<code>&gt;&gt;</code>	Bitwise right shift	$x \gg 2 = 2$ (0 000 0010)
<code>&lt;&lt;</code>	Bitwise left shift	$x \ll 2 = 40$ (0010 1000)

- Operator Identitas

Operator identitas adalah operator yang memeriksa apakah dua buah nilai ( atau variabel ) berada pada lokasi memori yang sama. Tabel berikut menunjukkan jenis operator identitas:

Operator	Penjelasan	Contoh
is	True jika kedua operand identik (menunjuk ke objek yang sama)	x is true
is not	True jika kedua operand tidak identik (tidak merujuk ke objek yang sama)	x is <b>not</b> true

- Operator Keanggotaan

Operator keanggotaan adalah operator yang digunakan untuk memeriksa apakah suatu nilai atau variabel merupakan anggota atau ditemukan di dalam suatu data (string, list, tuple, set, dan dictionary). Tabel berikut menunjukkan jenis operator keanggotaan:

Operator	Penjelasan	Contoh
in	True jika nilai/variabel ditemukan di dalam data	5 in x
not in	True jika nilai/variabel tidak ada di dalam data	5 not in x

- Tipe Data Bentukan

Ada 4, dengan perbedaan penggunaan:

- List

Sebuah kumpulan data yang terurut, dapat diubah, dan memungkinkan ada anggota yang sama

- Tuple

Sebuah kumpulan data yang terurut, tidak dapat diubah, dan memungkinkan ada anggota yang sama

- Set

Sebuah kumpulan data yang tidak berurutan, tidak terindeks, dan tidak memungkinkan ada anggota yang sama

- Dictionary

Sebuah kumpulan data yang tidak berurutan, dapat diubah, tidak memungkinkan ada anggota yang sama

Percabangan  
Tabel berikut menunjukkan tipe data bentukan:

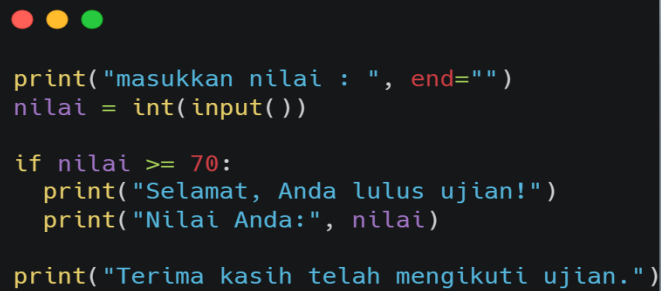
Tipe data dasar	Contoh nilai	Penjelasan
List	[1, 2, 3, 4, 5] atau ['apple', 'banana', 'cherry'] atau ['xyz', 768, 2.23]	Data untaian yang menyimpan berbagai tipe data dan isinya bisa diubah-ubah
Tuple	('xyz', 1, 3.14)	Data untaian yang menyimpan berbagai tipe data tapi isinya tidak bisa diubah
Dictionary	{ 'firstName': 'Joko', 'lastName': 'Widodo' }	Data untaian yang menyimpan berbagai tipe data berupa pasangan penunjuk dan nilai

Set	{'apple', 'banana', 'cherry'}	Data untaian yang menyimpan berbagai tipe data dan elemen datanya harus unik
-----	-------------------------------	--

- Percabangan

1. Percabangan IF

Sebagai contoh, akan dibuat suatu program dengan bahasa Python untuk menentukan apakah kita memiliki SIM.



```

print("masukkan nilai : ", end="")
nilai = int(input())

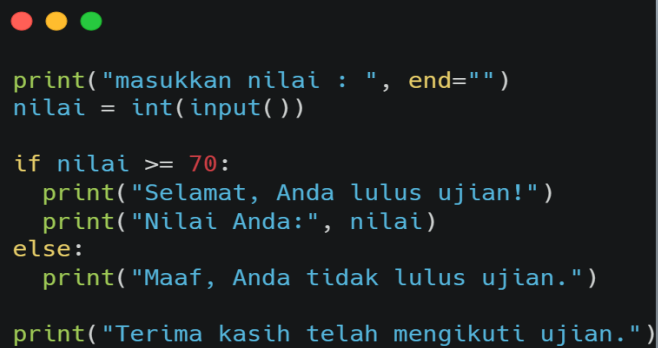
if nilai >= 70:
    print("Selamat, Anda lulus ujian!")
    print("Nilai Anda:", nilai)

print("Terima kasih telah mengikuti ujian.")

```

2. Percabangan IF-ELSE

Untuk mengeluarkan output ketika user memasukkan nilai kurang dari 70, perlu ditambahkan percabangan lagi. Kita dapat mengimplementasikan percabangan IF-ELSE.



```

print("masukkan nilai : ", end="")
nilai = int(input())

if nilai >= 70:
    print("Selamat, Anda lulus ujian!")
    print("Nilai Anda:", nilai)
else:
    print("Maaf, Anda tidak lulus ujian.")

print("Terima kasih telah mengikuti ujian.")

```

### 3. Percabangan IF-ELSE-IF

Kita dapat menggunakan percabangan IF-ELSE-IF untuk melengkapi program sebelumnya.

```
print("masukkan nilai : ", end="")
nilai = int(input())

if nilai >= 80:
    print("Selamat, Anda mendapatkan nilai A!")
elif nilai >= 75:
    print("Anda mendapatkan nilai B.")
elif nilai >= 70:
    print("Anda mendapatkan nilai C.")
else:
    print("Anda tidak lulus ujian.")

print("Terima kasih telah mengikuti ujian.")
```

### 4. Nested IF

Selain menggunakan code seperti gambar diatas untuk menentukan mahasiswa lulus ujian atau tidak kita dapat mengubah format code menjadi percabangan bersarang (nested if) dengan menempatkan percabangan di dalam percabangan. Kode program dapat dilihat seperti gambar dibawah ini.

```
print("masukkan nilai : ", end="")
nilai = int(input())

if nilai >= 70:
    print("Selamat, Anda lulus ujian!")
    if nilai >= 80:
        print("Anda mendapatkan nilai A")
    elif nilai >= 75:
        print("Anda mendapatkan nilai B")
    else:
        print("Anda mendapatkan nilai C")
else:
    print("Maaf, Anda tidak lulus ujian.")

print("Terima kasih telah mengikuti ujian.")
```

- **Perulangan**

Dalam python terdapat dua jenis perulangan , yaitu perulangan for dan perulangan while. Dalam implementasinya perulangan digunakan jika ingin mengulang sesuatu sebanyak n kali.

#### 1. Perulangan For

Pada perulangan for biasa digunakan untuk iterasi pada urutan berupa list, tuple, atau string. Sintaks dasar pada perulangan for di python:

```
for i in (kondisi) :
    #perintah
```



## 2. Perulangan While

Dengan menggunakan while maka dapat dilakukan perulangan selama kondisi tertentu terpenuhi. sintaks umum pada perulangan while :



```
while (kondisi) :  
    #lakukan sesuatu
```

- Fungsi

Dengan menggunakan fungsi kita dapat mengeksekusi suatu blok kode tanpa harus menulisnya berulang-ulang. Contoh :



```
def sapa(nama):  
    print("Halo,", nama, "! Selamat datang.")  
  
sapa("Ahmad")  
sapa("Budi")  
sapa("Cici")
```

## 2. Konsep Object-Oriented Programming

### 2.1 Abstraksi

Abstraksi dalam pemrograman berorientasi objek (OOP) adalah konsep pemodelan sistem yang kompleks dengan memfokuskan pada fitur pentingnya dan mengabaikan yang tidak penting. Dalam konteks OOP, abstraksi adalah proses menyembunyikan detail implementasi suatu objek dan hanya mengekspos fitur pentingnya yang relevan dengan objek tersebut. Contohnya ketika mengendarai mobil, user mengetahui bagaimana menyalakan mobil, menjalankan, menghentikan, dll, tetapi tidak mengetahui mekanisme apa yang terjadi pada mobil ketika mendapat perintah di atas.

Contoh :

```
# Mendefinisikan kelas abstraksi bernama
Binatang
class Binatang:
    def __init__(self, nama):
        self.nama = nama    # Membuat atribut
        nama

    def bersuara(self):
        pass    # Membuat method kosong bernama
        bersuara

# Mendefinisikan kelas Kucing sebagai subkelas
dari Binatang
class Kucing(Binatang):
    def bersuara(self):
        print("Meow")    # Implementasi method
        bersuara untuk kucing

# Mendefinisikan kelas Anjing sebagai subkelas
dari Binatang
class Anjing(Binatang):
    def bersuara(self):
        print("Guk")    # Implementasi method
        bersuara untuk anjing

# Membuat dua objek dari kelas Kucing dan
Anjing
hewan1 = Kucing("Kitty")
hewan2 = Anjing("Doggy")

# Memanggil method bersuara dari objek hewan1
dan hewan2
hewan1.bersuara()    # Output: Meow
hewan2.bersuara()    # Output: Guk
```

## 2.2 Enkapsulasi

Enkapsulasi adalah sebuah metode yang digunakan untuk mengatur struktur kelas dengan cara menyembunyikan alur kerja dari kelas tersebut. Yang dimaksud dengan struktur kelas tersebut adalah property dan method. Dengan konsep ini, kita dapat “menyembunyikan” property dan method dari suatu kelas agar hanya property dan method tertentu saja yang dapat diakses dari luar kelas. Dengan menghalangi akses dari luar kelas tersebut, elemen penting yang terdapat dalam kelas dapat lebih terjaga, dan menghindari kesalahan jika elemen tersebut diubah secara tidak sengaja. Untuk membatasi hak akses terhadap property dan method dalam suatu kelas, terdapat 3 jenis access modifier yang terdapat dalam python, yaitu public access, protected access, dan private access.

- **Public Access Modifier**

Pada umumnya ketika kita mendeklarasikan suatu variabel atau method, maka itulah public access modifier. Setiap class, variable dan method yang dibuat secara default merupakan public.

- **Protected Access Modifier**

Jika suatu variabel dan method dideklarasikan secara protected, maka variable dan method tersebut hanya dapat diakses oleh kelas turunan darinya. Cara mendeklarasikannya dengan menambahkan satu underscore (\_) sebelum variable atau method.

- **Private Access Modifier**

Jika suatu variabel dan method dideklarasikan secara private, maka variable dan method tersebut hanya dapat diakses di dalam kelas itu sendiri, private access merupakan yang paling aman. Dalam mendeklarasikannya, hanya perlu menambahkan double underscore (\_\_) sebelum nama variable dan methodnya.

- **Setter dan Getter**

Seperti yang sudah dijelaskan pada minggu yang lalu, setter adalah sebuah method yang digunakan untuk mengatur sebuah property yang ada di dalam suatu kelas/objek. Sedangkan getter adalah sebuah method yang digunakan untuk mengambil nilai dari suatu property. Dikarenakan property dengan access modifier private hanya dapat diakses dari dalam kelas tersebut, dengan metode inilah kita dapat mengaksesnya dari luar kelas. Terdapat 2 cara untuk membuat setter dan getter, yaitu dengan tanpa decorator dan dengan decorator.

Contoh :

```
# Mendefinisikan kelas bernama Mahasiswa
class Mahasiswa:
    def __init__(self, nama, nim, jurusan):
        self.__nama = nama # Atribut private
        self._nim = nim # Atribut protected
        self.jurusan = jurusan # Atribut public

    def tampilkan_data(self):
        print("Nama:", self.__nama) # Menampilkan nilai dari atribut private __nama menggunakan method tampilkan_data
        print("NIM:", self._nim) # Menampilkan nilai dari atribut protected _nim menggunakan method tampilkan_data
        print("Jurusan:", self.jurusan) # Menampilkan nilai dari atribut public jurusan menggunakan method tampilkan_data

    def get_nama(self):
        return self.__nama # Mengembalikan nilai atribut private __nama menggunakan method get_nama

    def set_nama(self, nama_baru):
        self.__nama = nama_baru # Mengubah nilai atribut private __nama dengan nilai yang baru menggunakan method set_nama

    def get_nim(self):
        return self._nim # Mengembalikan nilai atribut protected _nim menggunakan method get_nim

    def set_nim(self, nim_baru):
        self._nim = nim_baru # Mengubah nilai atribut protected _nim dengan nilai yang baru menggunakan method set_nim

mahasiswa1 = Mahasiswa("ILHAM", "121140001", "Teknik Informatika") # Membuat objek mahasiswa1 dengan nilai nama "ILHAM",
"121140001", "Teknik Informatika"

# Memanggil method tampilkan_data
mahasiswa1.tampilkan_data() # Output: Nama: ILHAM, NIM: 121140001, Jurusan: Teknik Informatika

# Mengakses dan mengubah nilai atribut nama dan nim
#mahasiswa1.__nama = "Yoga" # Akan menghasilkan error karena atribut __nama bersifat private
mahasiswa1.set_nama("Yoga") # Mengubah nilai atribut private __nama dengan method set_nama
mahasiswa1._nim = "00001" # Mengubah nilai atribut protected _nim secara langsung (dapat dilakukan karena atribut _nim bersifat
protected)
mahasiswa1.tampilkan_data() # Output: Nama: Yoga, NIM: 00001, Jurusan: Teknik Informatika
```

## 2.3 Inheritance

Inheritance atau pewarisan dalam pemrograman berorientasi objek (OOP) adalah konsep dimana suatu class dapat mewarisi properti dan method dari class lain. Class yang mewarisi disebut sebagai subclass atau child class, sedangkan class yang diwarisi disebut superclass atau parent class.

Contoh :

```
# Mendefinisikan kelas bernama Kendaraan
class Kendaraan:
    def __init__(self, merk, tahun):
        self.merk = merk # Inisialisasi atribut merk dari parameter merk
        self.tahun = tahun # Inisialisasi atribut tahun dari parameter tahun

    def info_kendaraan(self):
        print(f"Merk: {self.merk}") # Menampilkan nilai atribut merk
        print(f"Tahun: {self.tahun}") # Menampilkan nilai atribut tahun

# Mendefinisikan kelas turunan bernama Mobil
class Mobil(Kendaraan):
    def __init__(self, merk, tahun, warna):
        super().__init__(merk, tahun) # Memanggil method __init__() dari superclass dan mengirimkan parameter merk dan tahun
        self.warna = warna # Inisialisasi atribut warna dari parameter warna

    def info_kendaraan(self):
        super().info_kendaraan() # Memanggil method info_kendaraan() dari superclass
        print(f"Warna: {self.warna}") # Menampilkan nilai atribut warna

# Mendefinisikan kelas turunan bernama Motor
class Motor(Kendaraan):
    def __init__(self, merk, tahun, jenis):
        super().__init__(merk, tahun) # Memanggil method __init__() dari superclass dan mengirimkan parameter merk dan tahun
        self.jenis = jenis # Inisialisasi atribut jenis dari parameter jenis

    def info_kendaraan(self):
        super().info_kendaraan() # Memanggil method info_kendaraan() dari superclass
        print(f"Jenis: {self.jenis}") # Menampilkan nilai atribut jenis

mobil1 = Mobil("Toyota", 2018, "Merah") # Membuat objek mobil1 dari class Mobil
motor1 = Motor("Honda", 2020, "Bebek") # Membuat objek motor1 dari class Motor

mobil1.info_kendaraan() # Memanggil method info_kendaraan() dari objek mobil1
# Output: Merk: Toyota, Tahun: 2018, Warna: Merah

motor1.info_kendaraan() # Memanggil method info_kendaraan() dari objek motor1
# Output: Merk: Honda, Tahun: 2020, Jenis: bebek
```

## 2.4 Polymorphism

Polymorphism berarti banyak (poly) dan bentuk (morphism), dalam Pemrograman Berbasis Objek konsep ini memungkinkan digunakannya suatu interface yang sama untuk memerintah objek agar melakukan aksi atau tindakan yang mungkin secara prinsip sama namun secara proses berbeda. Polymorphism merupakan kemampuan suatu method untuk bekerja dengan lebih dari satu tipe argumen. Pada bahasa lain (khususnya C++), konsep ini sering disebut dengan method overloading. Pada dasarnya, Python tidak menangani hal ini secara khusus. Hal ini disebabkan karena Python merupakan suatu bahasa pemrograman yang bersifat duck typing(dynamic typing).

Contoh :

```
# membuat class BangunRuang sebagai parent class
class BangunRuang:
    # membuat method hitung_luas dengan nilai default yang sama dengan pass
    def hitung_luas(self):
        pass

    # membuat method hitung_volume dengan nilai default yang sama dengan pass
    def hitung_volume(self):
        pass

# membuat subclass Kubus yang meng-inherit dari class BangunRuang
class Kubus(BangunRuang):
    # membuat method __init__ yang akan dijalankan ketika objek Kubus dibuat
    def __init__(self, sisi):
        # menginisialisasi nilai sisi
        self.sisi = sisi

    # mengimplementasikan method hitung_luas dengan rumus luas kubus
    def hitung_luas(self):
        return 6 * self.sisi ** 2

    # mengimplementasikan method hitung_volume dengan rumus volume kubus
    def hitung_volume(self):
        return self.sisi ** 3

# membuat subclass Balok yang meng-inherit dari class BangunRuang
class Balok(BangunRuang):
    # membuat method __init__ yang akan dijalankan ketika objek Balok dibuat
    def __init__(self, panjang, lebar, tinggi):
        # menginisialisasi nilai panjang, lebar, dan tinggi
        self.panjang = panjang
        self.lebar = lebar
        self.tinggi = tinggi

    # mengimplementasikan method hitung_luas dengan rumus luas balok
    def hitung_luas(self):
        return 2 * (self.panjang * self.lebar + self.panjang * self.tinggi + self.lebar * self.tinggi)

    # mengimplementasikan method hitung_volume dengan rumus volume balok
    def hitung_volume(self):
        return self.panjang * self.lebar * self.tinggi

# membuat objek Kubus dengan sisi 5
kubus = Kubus(5)
# membuat objek Balok dengan panjang 3, lebar 4, dan tinggi 5
balok = Balok(3, 4, 5)

# mencetak luas dan volume Kubus dengan memanggil method yang sesuai
print("Luas kubus:", kubus.hitung_luas())      # Output: Luas kubus: 150
print("Volume kubus:", kubus.hitung_volume())  # Output: Volume kubus: 125

# mencetak luas dan volume Balok dengan memanggil method yang sesuai
print("Luas balok:", balok.hitung_luas())      # Output: Luas balok: 94
print("Volume balok:", balok.hitung_volume())  # Output: Volume balok: 60
```