Object-Oriented Programming

Penyusun Modul: Fitria Yunita Dewi

Editor: Rina Fitriyani

Object Oriented Programming (OOP)

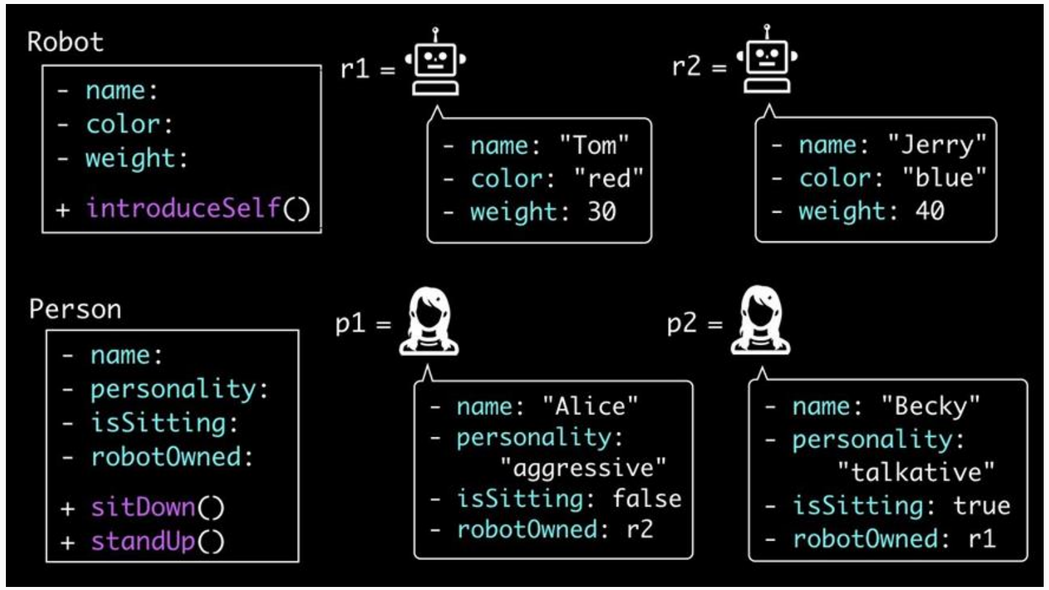
- Pemrograman yang didasarkan pada konsep “objek”, di mana setiap objek memiliki sifat (*attributes*) dan perilaku (*behavior*) seperti halnya semua objek di dunia nyata
- Hampir semua yang ada di Python bisa dilihat sebagai objek
- Pada python, objek memiliki:
  - *Properties* (sifat)
  - *Methods* (perilaku)

Contoh

Seekor pinguin (objek) memiliki

- ***Attributes***: nama, usia, tinggi, berat, warna
- ***Behaviors***: berenang, berjalan, menari, berburu makanan

Class and Object



Source: Youtube Channel CS Dojo

- **Class** adalah sebuah blueprint atau template dalam pembuatan objek
- **Object** adalah instansiasi/perwujudan dari class yang sudah didefinisikan
- **Instansiasi** merupakan instilah yang merujuk pada pembuatan objek dengan menanggil *class*-nya

Keterangan

- `Robot dan Person` adalah *Class*
- `r1, r2, p1,` dan `p2` adalah yang disebut sebagai *object/instance*
- Yang berwarna hijau adalah *attributes* dan ungu adalah *methods*

Pembuatan Class

Kita mendefinisikan sebuah kelas dengan menggunakan kata kunci **`class`** diikuti oleh nama kelas tersebut. Berikut adalah sintaks pembuatan kelas di Python.

```
class ClassName:
    '''class docstring'''
    class_body
```

Kelas memiliki docstring atau string dokumentasi yang bersifat opsional artinya bisa ada atau tidak. Docstring bisa diakses menggunakan format **`ClassName.__doc__`**

Contoh *Class*

```
class Karyawan:
    '''Dasar kelas untuk semua karyawan'''
    jumlah_karyawan = 0

    def __init__(self, nama, gaji):
        self.nama = nama
        self.gaji = gaji
        Karyawan.jumlah_karyawan += 1

    def tampilkan_jumlah(self):
        print("Total karyawan:", Karyawan.jumlah_karyawan)

    def tampilkan_profil(self):
        print("Nama :", self.nama)
        print("Gaji :", self.gaji)
        print()
```

Atribut kelas

Atribut instans/objek

Metode

- Variabel `jumlah_karyawan` adalah atribut kelas yang dibagi ke semua ***instance*** dari kelas ini. Variabel ini bisa diakses dari dalam atau luar kelas dengan menggunakan notasi titik, `Karyawan.jumlah_karyawan`

- **`__init__()`** adalah metode konstruktor, yaitu metode khusus yang digunakan Python untuk menginisialisasi pembuatan objek dari kelas tersebut.

- Fungsi – fungsi di dalam kelas (disebut ***metode***) pendefinisiannya sama dengan fungsi pada umumnya. Hanya saja, harus ada argumen pertama bernama `self`. Pada saat pemanggilan fungsi, argumen `self` ini otomatis ditambahkan oleh Python. Anda tidak perlu menambahkannya pada saat memanggil fungsi.

Instansiasi Objek

```
# Membuat objek pertama dari kelas Karyawan
karyawan1 = Karyawan("Sarah", 1000000)

# Membuat objek kedua dari kelas Karyawan
karyawan2 = Karyawan("Budi", 2000000)
```

Untuk membuat objek dari sebuah kelas, kita bisa memanggil nama kelas dengan isi argumen sesuai dengan argumen yang ada di dalam **`__init__()`** (tanpa argument `self`).

Mengakses **Atribut Kelas** dan **Atribut Instans/Objek**

Atribut kelas bisa diakses dengan menggunakan nama kelasnya (**`Karyawan`**).
Atribut instans bisa diakses dengan menggunakan nama instans tersebut (**`karyawan1`** atau **`karyawan2`**).

```
# atribut kelas
Karyawan.jumlah_karyawan
# atribut instans
karyawan1.nama
karyawan2.gaji
```

Mengakses **Metode**

Memanggil metode (sudah sering kita lakukan seperti pada saat memanggil metode list, string, tuple, dsb) sama seperti memanggil atribut namun menggunakan tanda kurung dengan isi argument disesuaikan dengan pendefinisian yang dilakukan di dalam kelas

```
karyawan1.tampilkan_profil()
karyawan2.tampilkan_profil2()
```

Encapsulation

Dalam OOP kita sering mendengar istilah *encapsulation* (pembungkusan) di mana data dibuat *private* agar tidak dapat diakses secara langsung dari luar kelas namun dapat diakses dari dalam kelas

Mengapa dibuat seperti itu?

- Untuk meningkatkan keamanan data
- Lebih mudah mengontrol atribut dan metode
- Class dapat diatur menjadi read-only maupun write-only
- Pembuat program dapat mengganti sebagian kode pada suatu instans tanpa mengubah instans lainnya dalam kelas yang sama

Atribut yang bersifat *private* ditandai dengan membubuhkan prefix *double underscores* (*dunder*)

Encapsulation: Getter and Setter

Suatu kelas dengan atribut *private* sebagai berikut:

```
class Telephone:
    def __init__(self):
        self.__price = 1000  #initialize standard prize

    def sell(self):
        print('Selling price is: $', self.__price)

    # setter
    def set_price(self, new_price):
        if new_price <= 0:
            print('Price must be POSITIVE.')
        else:
            self.__price = new_price

    # getter
    def get_price(self):
        return self.__price
```

Karena `self.__price` adalah atribut private, kita tidak bisa mengakses nilainya dari luar kelas

```
phone1 = Telephone()
phone1.__price
```

```
AttributeError: 'Telephone' object has no attribute '__price'
```

Serta tidak bisa mengubah nilainya

```
phone1.sell()
phone1.__price = 5000
phone1.sell()
```

```
Selling price is: $ 1000
Selling price is: $ 1000
```

Encapsulation: Getter and Setter

Untuk mengakses dan mengubah atribut *private* kita buat suatu metode *custom (getter* dan *setter)* di dalam kelas

```
class Telephone:
    def __init__(self):
        self.__price = 1000  #initialize standard prize

    def sell(self):
        print('Selling price is: $', self.__price)

    # setter
    def set_price(self, new_price):
        if new_price <= 0:
            print('Price must be POSITIVE.')
        else:
            self.__price = new_price

    # getter
    def get_price(self):
        return self.__price
```

Cara yang benar untuk mengakses nilai atribut *private*

```
phone1.get_price()
```

```
1000
```

Mengubah nilai atribut *private*

```
phone1.set_price(8000)
phone1.get_price()
```

```
8000
```

Inheritance

Kita bisa menurunkan karakteristik sebuah kelas ke kelas baru, dibandingkan dengan membuat kelas baru dari awal. Turunannya disebut kelas anak *(child class/sub class*) dan yang mewariskannya disebut kelas induk (*parent class/super class*).

Child class mewarisi atribut dari *parent class*, dan kita bisa menggunakan atribut tersebut seolah atribut itu didefinisikan juga di dalam kelas anak. Kelas anak juga bisa menimpa (*override*) data dan metode dari induknya dengan data dan metodenya sendiri.

Satu *child class* bisa mewarisi karakteristik dari satu atau beberapa *parent class (multiple inheritance)*

Inheritance Syntax

```
class SuperClass:
    # super class properties
    pass

class ChildClass1(SuperClass):
    # super class properties
    # child class 1 properties
    pass

class ChildClass2(SuperClass):
    # super class properties
    # child class 2 properties
    pass
```

Contoh:

```
# parent class
class Vehicle:
    def __init__(self,driver,wheels,seats):
        self.driver = driver
        self.noofwheels = wheels
        self.noofseats = seats

# child class
class Cab(Vehicle):
    pass

cab1 = Cab('Sandy', 4, 2)
print(cab1.driver)
print(cab1.noofseats)
```

```
Sandy
2
```

Inheritance

Jika terdapat beberapa kelas anak dengan sebagian atribut yang sama dan berbeda dengan kelas induk:

Cab		Bus
driver	↔	driver
wheels	↔	wheels
seats	↔	seats
kms	↔	kms
bill	↔	bill
cabtype		**color**

```
class Vehicle:
    minimumrate = 50
    def __init__(self,driver,wheels,seats,kms,bill):
        self.driver = driver
        self.noofwheels = wheels
        self.noofseats = seats
        self.running = kms
        self.bill = bill

    def rateperkm(self):
        return self.bill/self.running

class Cab(Vehicle):
    minimumrate = 75
    def __init__(self,driver,wheels,seats,kms,bill,cabtype):
        super().__init__(driver,wheels,seats,kms,bill)
        self.category = cabtype

class Bus(Vehicle):
    minimumrate = 25
    def __init__(self,driver,wheels,seats,kms,bill,color):
        super().__init__(driver,wheels,seats,kms,bill)
        self.color = color
```

Instansiasi objek `cab_1`

```
cab_1 = Cab('Anang', 4, 3, 50, 700, 'SUV')
print(cab_1.category)
print(cab_1.running)
print(cab_1.rateperkm())
```

```
SUV
50
14.0
```

Inheritance

Contoh 2:

Suatu kelas induk **`Shape`** dengan input parameter **`color`**

2 kelas anak:

- Kelas **`Circle`** dengan input **`radius`** dan **`color`** serta metode **`area()`**
- Kelas **`Rectangle`** dengan input **`width`**, **`length`** dan **`color`** serta metode **`area()`**

Kelas induk

```
import math

# super class -> Shape
class Shape(object):
    def __init__(self, color='red'):
        self.color = color
```

Kelas anak

```
class Circle(Shape):
    """Circle inherits from Shape."""
    def __init__(self, radius, color='blue'):
        super().__init__(color=color)
        self.radius = radius
    def area(self):
        return math.pi * self.radius**2
```

```
class Rectangle(Shape):
    """Rectangle inherits from Shape."""
    def __init__(self, width=1.0, length=1.0, color='orange'):
        super().__init__(color)
        self.width = width
        self.length = length
    def area(self):
        return self.width * self.length
```

Inheritance

```python
import math

# super class -> Shape
class Shape(object):
    def __init__(self, color='red'):
        self.color = color
```

Kelas induk

```python
# sub class -> Circle
class Circle(Shape):
    """Circle inherits from Shape."""
    def __init__(self, radius, color='blue'):
        Shape.__init__(self, color)
        self.radius = radius

    def area(self):
        return math.pi * self.radius**2

# sub class -> Rectangle
class Rectangle(Shape):
    """Rectangle inherits from Shape."""
    def __init__(self, width=1.0, length=1.0, color='orange'):
        Shape.__init__(self, color)
        self.width = width
        self.length = length

    def area(self):
        return self.width * self.length
```

Kelas anak

Catatan:

sintaks berikut `super().__init__(color)`

dapat juga diganti seperti berikut `Shape.__init__(self, color)`

```python
# Shape
shape1 = Shape('white')
print('Color of shape1:', shape1.color)
```

```
Color of shape1: white
```

```python
# Circle
circle1 = Circle(radius=5)
print('Radius of circle1:', circle1.radius)
print('Color of circle1:', circle1.color)
print('Area of circle1:', circle1.area())
```

```
Radius of circle1: 5
Color of circle1: blue
Area of circle1: 78.53981633974483
```

```python
# Rectangle
rect1 = Rectangle(2, 8, 'yellow')
print('Length of rect1:', rect1.length)
print('Color of rect1:', rect1.color)
print('Area of rect1:', rect1.area())
```

```
Length of rect1: 8
Color of rect1: yellow
Area of rect1: 16
```

Multiple Inheritance

Kelas anak juga dapat mewarisi lebih dari 1 kelas induk

Contoh:

```python
# Clock Class
class Clock:
    """Simulates the clock."""

    def __init__(self, hours, minutes, seconds):
        self.__hours = hours
        self.__minutes = minutes
        self.__seconds = seconds

    def set_clock(self, hours, minutes, seconds):
        self.__hours = hours
        self.__minutes = minutes
        self.__seconds = seconds

    def time(self):
        return '{0}:{1}:{2}'.format(self.__hours, self.__minutes, self.__seconds)
```

```python
# Calendar Class
class Calendar(object):
    """Simulates the calendar"""

    def __init__(self, d, m, y):
        self.set_calendar(d, m, y)

    def set_calendar(self, d, m, y):
        self.__d = d
        self.__m = m
        self.__y = y

    def date(self):
        return '{0}:{1}:{2}'.format(self.__d, self.__m, self.__y)
```

Kelas **`CalendarClock`** berikut mewarisi kelas **`Clock`** dan **`Calendar`**

```python
# CalendarClock Class
class CalendarClock(Clock, Calendar):
    """Keeps calendar and clock together."""

    def __init__(self, day, month, year, hours, minutes, seconds):
        # call the super classes init methods
        Clock.__init__(self, hours, minutes, seconds)
        Calendar.__init__(self, day, month, year)
```

```python
# create a CalendarClock object
calendar_clock = CalendarClock(25, 2, 2022, 10, 30, 5)
print(calendar_clock.time())
print(calendar_clock.date())
```

```
10:30:5
25:2:2022
```

Atribut Built-in

Setiap kelas di Python memiliki atribut *built-in* (bawaan) yang biasanya diapit oleh *double underscores* (*dunder*). Beberapa *attribute* tersebut adalah sebagai berikut:

__dict__

Dictionary yang berisi namespace dari kelas

__doc__

Mengakses string dokumentasi (*docstring*) dari kelas

__name__

Nama kelas

__module__

nama modul tempat kelas didefinisikan. Nilai attribut ini di mode interaktif adalah "***__main__***"

Atribut Built-in

Pada kelas **`Karyawan`** yang sebelumnya sudah dibuat, berikut adalah atribut *built-in* nya

```
Karyawan.__doc__: Dasar kelas untuk semua karyawan
Karyawan.__name__: Karyawan
Karyawan.__module__: __main__
Karyawan.__dict__: {'tampilkan_jumlah': , '__module__':
'__main__', '__doc__': 'Dasar kelas untuk semua
karyawan', 'jumlah_karyawan': 2, '__weakref__':
<attribute '__weakref__' of 'Karyawan' objects>,
'tampilkan_profil': , '__dict__': <attribute '__dict__'
of 'Karyawan' objects>, '__init__': }
Karyawan.__bases__: (<class 'object'>,)
```

Latihan

Buat suatu kelas bernama **`Sepeda`** dengan parameter input dari atribut instansnya yaitu **`gigi`** dan **`kecepatan`**

Kelas ini memiliki metode

- `kayuh(x)` yang menambah atribut `kecepatan` sebesar `x`
- `pengereman()` yang akan mengubah `kecepatan` menjadi bernilai 0
- `gantiGigi(a)` yang akan mengganti atribut `gigi` menjadi bernilai `a`

Latihan

Buat suatu kelas bernama **`Sepeda`** dengan parameter input dari atribut instansnya yaitu **`gigi`** dan **`kecepatan`**

Kelas ini memiliki metode

- `kayuh(x)` yang menambah atribut `kecepatan` sebesar `x`
- `pengereman()` yang akan mengubah `kecepatan` menjadi bernilai 0
- `gantiGigi(a)` yang akan mengganti atribut `gigi` menjadi bernilai `a`

Solusi

```
class Sepeda:
    def __init__(self, gigi, kecepatan):
        self.gigi = gigi
        self.kecepatan = kecepatan
    def kayuh(self, x):
        self.kecepatan += x
    def pengereman(self):
        self.kecepatan = 0
    def gantiGigi(self, a):
        self.gigi = a
```

```
sepeda1 = Sepeda(2,30)
sepeda1.kayuh(40)
sepeda1.gantiGigi(1)
print("gigi:", sepeda1.gigi)
print("kecepatan:",sepeda1.kecepatan)
sepeda1.pengereman()
print("kecepatan:",sepeda1.kecepatan)
```

```
gigi: 1
kecepatan: 70
kecepatan: 0
```