# Python Objects & Classes

Python adalah bahasa pemrograman berorientasi objek (OOP). Tidak seperti procedure-oriented programming, di mana penekanan utamanya adalah pada fungsi, OOP menekankan pada objek.

Objek adalah kumpulan data (variabel) dan method (fungsi) yang bekerja pada data tersebut. Demikian pula, class adalah blueprint dari objek itu.

Kita dapat menganggap class sebagai sketch/gambar atau prototipe sebuah rumah. Ini berisi semua detail tentang lantai, pintu, jendela, dll. Berdasarkan deskripsi ini kita dapat membangun rumah. Rumah adalah objeknya.

Karena banyak rumah dapat dibuat dari blueprint rumah, kita dapat membuat banyak objek dari sebuah class. Objek juga disebut instance dari kelas dan proses pembuatan objek ini disebut *instantiate*.

# OOP

Salah satu pendekatan populer untuk memecahkan masalah pemrograman adalah dengan membuat objek. Ini dikenal sebagai OOP.

Sebuah objek memiliki dua karakteristik:

* Attributes
* Behavior

Misalnya, seekor burung beo (parrot) adalah sebuah objek, karena memiliki sifat-sifat berikut:

* Nama, umur, warna adalah Attributes
* Bisa bernyanyi, menari adalah Behavior

Konsep OOP dalam Python berfokus pada pembuatan kode yang dapat digunakan kembali. Konsep ini juga dikenal sebagai DRY (Don't Repeat Yourself).

## Class

Class adalah blueprint untuk objek.

Kita dapat menganggap class sebagai sketsa dari parrot yang berisi semua detail tentang nama, warna, ukuran, dll. Berdasarkan uraian tersebut, kita dapat mempelajari tentang si parrot. Contoh class dengan nama Parrot

| class **Parrot**:  pass |
| --- |

## Object

Sebuah objek (instance) adalah sebuah instantiate (realisasi dari sebuah pre-defined object) dari sebuah class. Ketika class didefinisikan, hanya deskripsi untuk objek yang didefinisikan. Oleh karena itu, tidak ada memori atau penyimpanan yang dialokasikan. Di sini, obj adalah objek dari class Parrot.

Misalkan kita memiliki detail tentang si Parrot. Sekarang, kita akan menunjukkan bagaimana membangun class dan objek Parrot.

| class **Parrot**:   *# class attribute*  species = "bird"   *# instance attribute*  def **\_\_init\_\_**(self, name, age):  self.name = name  self.age = age  *# instantiate the Parrot class* blu = Parrot("Blu", 10) woo = Parrot("Woo", 15)  *# access the class attributes* print("Blu is a {}".format(blu.\_\_class\_\_.species)) print("Woo is also a {}".format(woo.\_\_class\_\_.species))  *# access the instance attributes* print("{} is {} years old".format( blu.name, blu.age)) print("{} is {} years old".format( woo.name, woo.age)) |
| --- |

| Blu **is** a bird Woo **is** also a bird Blu **is** 10 years old Woo **is** 15 years old |
| --- |

Pada program di atas, kita membuat sebuah class dengan nama Parrot. Kemudian, kita mendefinisikan attribute-nya. Attribute merupakan ciri khas dari sebuah objek.

Attribute ini didefinisikan di dalam metode \_\_init\_\_ class. Ini adalah metode penginisialisasi yang pertama kali dijalankan segera setelah objek dibuat.

Kemudian, kita membuat instance dari class Parrot. Di sini, blu dan woo adalah referensi (nilai) ke objek baru kita.

Kita dapat mengakses attribute class menggunakan \_\_class\_\_.species. Attribute class adalah sama untuk semua instance class. Demikian pula, kita dapat mengakses attribute instance menggunakan blu.name dan blu.age. Namun, attribute instance berbeda untuk setiap instance kelas.

## Methods

Method adalah fungsi yang didefinisikan di dalam body of class. Method digunakan untuk mendefinisikan perilaku suatu objek.

| class **Parrot**:    *# instance attributes*  def **\_\_init\_\_**(self, name, age):  self.name = name  self.age = age    *# instance method*  def **sing**(self, song):  return "{} sings {}".format(self.name, song)   def **dance**(self):  return "{} is now dancing".format(self.name)  *# instantiate the object* blu = Parrot("Blu", 10)  *# call our instance methods* print(blu.sing("'Happy'")) print(blu.dance()) |
| --- |

| Blu sings 'Happy' Blu **is** now dancing |
| --- |

Pada program di atas, kita mendefinisikan dua metode yaitu sing() dan dance(). Ini disebut instance method karena dipanggil pada objek instance yaitu blu.

## Inheritance

Inheritance adalah cara membuat class baru untuk menggunakan detail class yang sudah ada tanpa memodifikasinya. Class yang baru terbentuk adalah class turunan (atau child class). Demikian pula class yang sudah ada adalah base class (atau parent class).

| *# parent class* class **Bird**:    def **\_\_init\_\_**(self):  print("Bird is ready")   def **whoisThis**(self):  print("Bird")   def **swim**(self):  print("Swim faster")  *# child class* class **Penguin**(Bird):   def **\_\_init\_\_**(self):  *# call super() function*  super().\_\_init\_\_()  print("Penguin is ready")   def **whoisThis**(self):  print("Penguin")   def **run**(self):  print("Run faster")  peggy = Penguin() peggy.whoisThis() peggy.swim() peggy.run() |
| --- |

| Bird **is** ready Penguin **is** ready Penguin Swim faster Run faster |
| --- |

Pada program di atas, kita membuat dua class yaitu Bird (parent class) dan Penguin (child class). Child class mewarisi fungsi dari Parent class. Kita bisa melihat ini dari method swim().

Child class memodifikasi perilaku dari parent class. Kita bisa melihat ini dari metode whoisThis(). Selanjutnya, kita dapat memperluas fungsi parent class, dengan membuat method run() baru.

Selain itu, kami menggunakan fungsi super() di dalam metode \_\_init\_\_() . Ini memungkinkan kita untuk menjalankan metode \_\_init\_\_() dari kelas parent di dalam kelas child.

## 

## Encapsulation

Menggunakan OOP di Python, kita dapat membatasi akses ke method dan variable. Ini mencegah data kita dimodifikasi secara langsung, dapat disebut juga enkapsulasi. Dalam Python, kita dapat menunjukkan private attribute menggunakan garis bawah sebagai awalan yaitu \_ atau \_\_

| class **Computer**:   def **\_\_init\_\_**(self):  self.\_\_maxprice = 900   def **sell**(self):  print("Selling Price: {}".format(self.\_\_maxprice))   def **setMaxPrice**(self, price):  self.\_\_maxprice = price  c = Computer() c.sell()  *# change the price* c.\_\_maxprice = 1000 c.sell()  *# using setter function* c.setMaxPrice(1000) c.sell() |
| --- |

| Selling Price: 900 Selling Price: 900 Selling Price: 1000 |
| --- |

Pada program di atas, kita menggunakan metode \_\_init\_\_() untuk menyimpan harga jual maksimal dari Komputer.

| c.\_\_maxprice = 1000 |
| --- |

Di sini, kita mencoba mengubah nilai \_\_maxprice di luar class. Namun, karena \_\_maxprice adalah variabel private, modifikasi ini tidak terlihat pada output.

Seperti yang ditunjukkan, untuk mengubah nilainya, kita harus menggunakan fungsi setter yaitu setMaxPrice() yang mengambil harga sebagai parameter.

## Polymorphism

Polymorphism adalah kemampuan OOP untuk menggunakan common interface untuk berbagai tipe data.

Misalkan, kita perlu mewarnai sebuah bentuk, ada beberapa pilihan bentuk (persegi panjang, persegi, dan lingkaran). Kita bisa menggunakan method yang sama untuk mewarnai bentuk apapun. Konsep ini disebut Polymorphism.

| class **Parrot**:  def **fly**(self):  print("Parrot can fly")    def **swim**(self):  print("Parrot can't swim")  class **Penguin**:  def **fly**(self):  print("Penguin can't fly")    def **swim**(self):  print("Penguin can swim")  *# common interface* def **flying\_test**(bird):  bird.fly()  *#instantiate objects* blu = Parrot() peggy = Penguin()  *# passing the object* flying\_test(blu) flying\_test(peggy) |
| --- |

| Parrot can fly Penguin can't fly |
| --- |

Kita mendefinisikan dua classes Parrot dan Penguin. Masing-masing memiliki metode fly() yang sama. Namun, fungsinya berbeda.

Untuk menggunakan Polymorphism, kita dapat membuat sebuah common interface yaitu fungsi flying\_test() dan memanggil method fly(). ketika kita memberikan objek blu dan peggy dalam fungsi flying\_test(), kode akan berjalan menampilkan output dari method fly().