



# Object Oriented Programming (Python)



# Table of Content

## What will We Learn Today?

1. OOP
2. Why OOP?
3. Class
4. Instance
5. Attributes & Methods
6. Inheritance, Override, Overloadng





OOP atau Object Oriented Programming merupakan sebuah paradigma dalam pemrograman yang didasarkan pada konsep objek, yang bisa berisi data, dalam bentuk field atau attributes, dan code, dalam bentuk **prosedur** atau **methods** .



# Why OOP?

OOP memungkinkan programmer untuk mengemas suatu state dari data dan fungsionalitas untuk memodifikasi state dari data tersebut, dengan tetap menjaga detailnya secara tersembunyi. Sehingga, kode dengan desain OOP bersifat modular & fleksibel.







# Contoh Tanpa OOP

Enkapsulasi data dalam bentuk “list” memiliki kelemahan seperti:

- Sulit dimaintain (cth: harus ingat index ke 0 itu merepresentasikan “nama”)
- Jika salah satu “field” tidak terisi (panjang list berbeda), maka susunan enkapsulasi data tidak konsisten
- Tidak bisa mengenkapsulasi behavior

```
# Tanpa OOP  
kucing = ["tom", 2]  
ikan = ["deni", 1]
```



# Contoh Dengan OOP

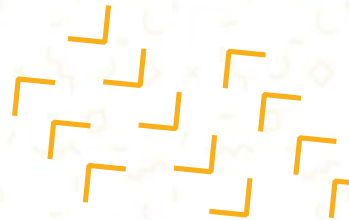
# Dengan OOP

```
class Binatang:
    def __init__(self, nama, umur):
        self.nama = nama
        self.umur = umur

    def bersuara(self, bunyi):
        print(self.nama, "bersuara", bunyi)

kucing = Binatang("tom", 2)
ikan = Binatang("deni", 1)
```

- “Class” merupakan reserved keyword untuk mendefinisikan suatu Class
- Method init () merupakan “constructor” atau method yang akan dipanggil ketika suatu object dibuat dari suatu class.
- init () memungkinkan kita untuk menginisiasi attributes yang ada pada suatu class





# Contoh Dengan OOP

# Dengan OOP

```
class Binatang:
    def __init__(self, nama, umur):
        self.nama = nama
        self.umur = umur

    def bersuara(self, bunyi):
        print(self.nama, "bersuara", bunyi)

kucing = Binatang("tom", 2)
ikan = Binatang("deni", 1)
```

- “Self” merepresentasikan instance dari sebuah class.
- “self.nama” & “self.umur” merupakan instance-attributes
- “Self” memungkinkan kita untuk mengakses attributes & methods dari suatu class.
- “bersuara()” merupakan methods dari class Binatang





# Class

Class merupakan sebuah blueprint dari suatu objects yang menyediakan nilai awal untuk

suatu state (attributes), dan implementasi dari suatu behaviour(methods).

Attributes merupakan data yang disimpan didalam class atau instance yang merepresentasikan state dari suatu class atau instance.

Methods merupakan behavior suatu class atau instance.





# Class vs Instance

```
# Dengan OOP

class Binatang:
    def __init__(self, nama, umur):
        self.nama = nama
        self.umur = umur

    def bersuara(self, bunyi):
        print(self.nama, "bersuara", bunyi)

kucing = Binatang("tom", 2)
ikan = Binatang("deni", 1)
```

- “class Binatang” merupakan blueprint
- “Binatang()” disebut dengan “instantiation”
- Variable “kucing” & “ikan” merupakan object dari class “Binatang”



# Exercise

Buatlah sebuah Class dengan nama “Kendaraan”, yang memiliki attributes: nama, warna & jumlah\_roda. Class “Kendaraan” juga memiliki behavior “maju” & “mundur”. “Maju” akan menampilkan bahwa kendaraan sedang maju, “Mundur” akan menampilkan bahwa kendaraan sedang mundur”. Dan buatlah 2 buah object, dengan nama “mobil” & “pesawat”

```
Mobil sedang maju  
Mobil sedang mundur
```

# Class Attributes

```
class Binatang:
    nama_latin = "Animalia"

    def __init__(self, nama, umur):
        self.nama = nama
        self.umur = umur

kucing = Binatang("tom", 2)
ikan = Binatang("deni", 1)

print(kucing.nama, kucing.nama_latin)
print(ikan.nama, ikan.nama_latin)
print(Binatang.nama_latin)
```

- “nama\_latin” merupakan class attributes. Class Attributes merupakan suatu data yang memiliki value yang sama pada semua object yang dibuat menggunakan suatu class.
- Class Attributes, bisa dipanggil tanpa menginstansiate suatu class.
- kucing.nama\_latin akan sama dengan ikan.nama\_latin dan Binatang.nama\_latin





# Instance Attributes

```
class Binatang:
    nama_latin = "Animalia"

    def __init__(self, nama, umur):
        self.nama = nama
        self.umur = umur

kucing = Binatang("tom", 2)
ikan = Binatang("deni", 1)

print(kucing.nama, kucing.nama_latin)
print(ikan.nama, ikan.nama_latin)
print(Binatang.nama_latin)
```

- “self.nama”, “self.umur” merupakan instance attributes. Instance attributes adalah suatu data, yang dimiliki oleh spesifik instance / object dari suatu class.
- kucing.nama bisa berbeda dengan ikan.nama
- Instance attributes hanya bisa dipanggil setelah proses instantiate dilakukan





# Class & Instance Attributes

```
class Binatang:
    nama_latin = "Animalia"

    def __init__(self, nama, umur):
        self.nama = nama
        self.umur = umur

kucing = Binatang("tom", 2)
ikan = Binatang("deni", 1)


print(kucing.nama, kucing.nama_latin)
print(ikan.nama, ikan.nama_latin)
print(Binatang.nama_latin)
```

- “self.nama”, “self.umur” merupakan instance attributes. Instance attributes adalah suatu data, yang dimiliki oleh spesifik instance / object dari suatu class.
- kucing.nama bisa berbeda dengan ikan.nama





# Exercise



Pada class “Kendaraan” yang telah dibuat sebelumnya, tambahkan satu class attributes  
“bahan\_bakar” dengan value “bensin”



# Class Methods

```
class Binatang:
    nama_latin = "Animalia"

    def __init__(self, nama, umur):
        self.nama = nama
        self.umur = umur

    @classmethod
    def tidur(cls):
        print(f"{cls.nama_latin} sedang tidur")

    @staticmethod
    def bangun():
        print(f"{Binatang.nama_latin} sudah bangun")

kucing = Binatang("tom", 2)
ikan = Binatang("deni", 1)

kucing.bangun()
ikan.bangun()
Binatang.bangun()
```

- @staticmethod adalah method yang terikat dengan Class ditempat method tersebut di definisikan.
- Class method hanya bisa mengakses class variables dari class yang memanggilnya. & tidak bisa mengakses Instance variable.
- "cls" adalah parameter yang argumennya akan diisi ketika dipanggil. Isi dari "cls" merupakan class pemanggil method tersebut, bukan parent class.



# Class Methods

```
class Binatang:
    nama_latin = "Animalia"

    def __init__(self, nama, umur):
        self.nama = nama
        self.umur = umur

    @classmethod
    def tidur(cls):
        print(f"{cls.nama_latin} sedang tidur")

    @staticmethod
    def bangun():
        print(f"{Binatang.nama_latin} sudah bangun")

kucing = Binatang("tom", 2)
ikan = Binatang("deni", 1)

kucing.bangun()
ikan.bangun()
Binatang.bangun()
```

- @classmethod adalah method yang terikat dengan Class pemanggil methodnya.
- Class method hanya bisa mengakses class variables dari class yang memanggilnya. & tidak bisa mengakses Instance variable.
- "cls" adalah parameter yang argumennya akan diisi ketika dipanggil. Isi dari "cls" merupakan class pemanggil method tersebut, bukan parent class.





# Instance Method

```
class Binatang:
    nama_latin = "Animalia"

    def __init__(self, nama, umur):
        self.nama = nama
        self.umur = umur

    @classmethod
    def tidur(cls):
        print(f"{cls.nama_latin} sedang tidur")

    @staticmethod
    def bangun():
        print(f"{Binatang.nama_latin} sudah bangun")

kucing = Binatang("tom", 2)
ikan = Binatang("deni", 1)

kucing.bangun()
ikan.bangun()
Binatang.bangun()
```

Instance method merupakan behavior spesifik yang terikat pada

instance dari suatu class.

- Instance method, memiliki parameter wajib yaitu "self" yang akan dimasukkan secara otomatis ketika dipanggil.
- Instance method bisa mengakses instance variable & class variable dari class dimana method tersebut di definisikan.



# Exercise

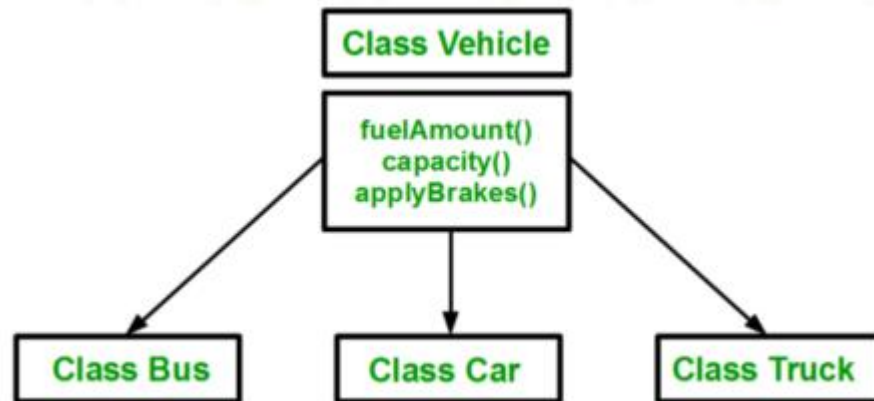
Pada class “Kendaraan” yang telah dibuat sebelumnya, tambahkan method “isi\_bahan\_bakar” yang akan menampilkan value dari “bahan\_bakar” sedang diisi. Dalam hal ini, output yang diinginkan adalah “Bensin sedang diisi”.



# Inheritance

Inheritance merupakan sebuah kapabilitas dari konsep OOP untuk mendapatkan attributes dari class lain

- Class Vehicle, merupakan parent class atau super class dari class bus, car, & truck.
- Class bus, car, & truck, disebut sebagai child class atau sub class.
- Class bus, car, & truck





# Inheritance

```
class Binatang:
    nama_latin = "Animalia"

    def __init__(self, nama, umur):
        self.nama = nama
        self.umur = umur

    @classmethod
    def tidur(cls):
        print(f"{cls.nama_latin} sedang tidur")

    @staticmethod
    def bangun():
        print(f"{Binatang.nama_latin} sudah bangun")

    def makan(self):
        print(f"{self.nama} sedang makan")

class Kucing(Binatang):
    pass

class Ikan(Binatang):
    pass

kucing = Kucing("toa", 2)
ikan = Ikan("deni", 1)

kucing.makan()
ikan.makan()
```

- Class “Kucing” dan “Ikan” merupakan subclass dari class “Binatang”
- Class “Kucing” dan “Ikan” dapat menggunakan behavior & attributes dari class “Binatang”.





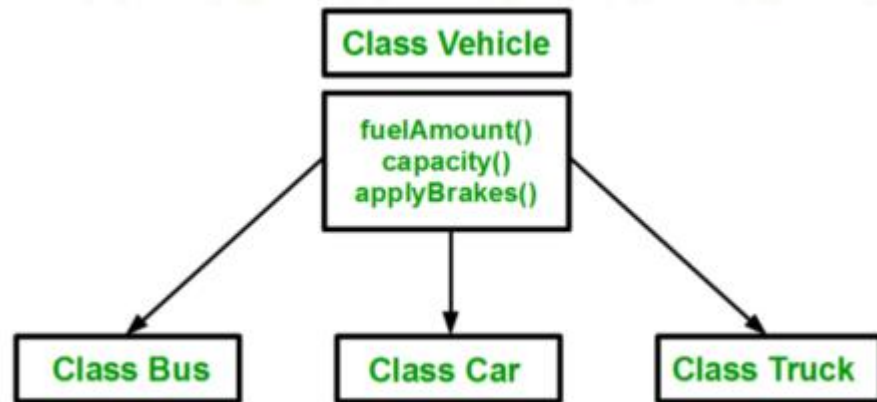
# Exercise

Buatlah sub class dari class “Kendaraan” dengan nama “Mobil” & “Pesawat”.  
Kemudian coba ganti object mobil menggunakan class “Mobil” & ganti object pesawat menggunakan class “Pesawat”.



# Override

Overriding merupakan kapabilitas dari konsep OOP yang memungkinkan subclass atau childclass memiliki behavior yang berbeda dengan parentclass namun dengan method yang sama



- Implementasi dari `fuelAmount()` pada Class Bus, Car dan Truck bisa berbeda dengan implementasi pada Class Vehicle



# Override

```
class Binatang:
    nama_latin = "Animalia"

    def __init__(self, nama, umur):
        self.nama = nama
        self.umur = umur

    @classmethod
    def tidur(cls):
        print(f'{cls.nama_latin} sedang tidur')

    @staticmethod
    def bangun():
        print(f'{Binatang.nama_latin} sudah bangun')

    def makan(self):
        print(f'{self.nama} sedang makan')

class Kucing(Binatang):
    nama_latin = "Felis Catus"

    def makan(self):
        print(f'{self.nama} sedang makan Ikan')

class Ikan(Binatang):
    nama_latin = "Chondrichthyes"

    def makan(self):
        print(f'{self.nama} sedang makan plankton')

kucing = Kucing("tom", 2)
ikan = Ikan("deni", 1)

kucing.makan()
ikan.makan()

kucing.tidur()
ikan.tidur()
```

- Class “Kucing” dan “Ikan” memiliki implementasi dari method “makan” yang berbeda dengan class “Binatang”
- Class “Kucing” dan “Ikan” juga memiliki attributes “nama\_latin” yang valuenya berbeda dengan class “Binatang”. “nama\_latin” adalah variable yang dimutate pada class “Kucing” dan “Ikan”
- @classmethod yang dipanggil pada class “Ikan” & “Kucing” akan menggunakan nama\_latin dari class “Ikan” & “Kucing”





## Exercise

Buatlah variable “bahan\_bakar” pada class “Mobil” dengan value “Solar” dan variable

“bahan\_bakar” pada class “Pesawat” dengan value “Avtur”. Kemudian coba panggil method

“isi\_bahan\_bakar” dari object mobil & pesawat.

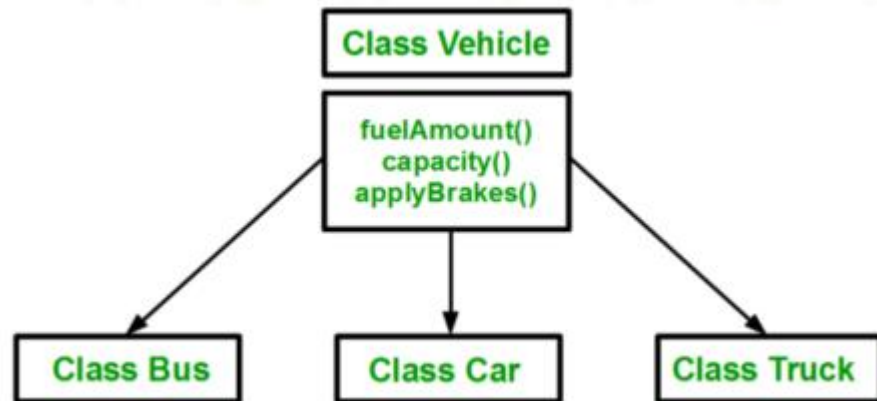




# Overload

Overloading merupakan kapabilitas dari konsep OOP yang memungkinkan sebuah class memiliki nama method yang sama dengan parameter yang berbeda.

- “fuelAmount” bisa memiliki 2 signature, fuelAmount() & fuelAmount(fuelType).





# Override

```
class Kucing(Binatang):
    nama_latin = "Felis Catus"

    def makan(self):
        print(f"{self.nama} sedang makan Ikan")

    def minum(self, minuman=None):
        if minuman == None:
            print("Tidak ada minuman")
        else:
            print(f"{self.nama} sedang minum {minuman}")

class Ikan(Binatang):
    nama_latin = "Chondrichthyes"

    def makan(self):
        print(f"{self.nama} sedang makan plankton")

kucing = Kucing("tom", 2)
ikan = Ikan("deni", 1)

kucing.minum()
kucing.minum("air putih")
```

Method “minum” memiliki 2 cara pemanggilan, bisa dengan argumen, bisa tidak. Sehingga method “minum” bisa memiliki beberapa implementasi.

- Overloading pada python diimplementasikan dengan memanfaatkan kapabilitas “default value” pada param



# Exercise

Buatlah sebuah method pada class “Mobil” dengan nama “belok”. Method “belok” bisa dipanggil dengan 2 cara “belok()” atau “belok(“kanan”)”. Jika methodelok dipanggil tanpa argumen, maka tampilkan “Tidak bisa belok karena tidak ada arah”, namun jika methodelok dipanggil menggunakan argumen dalam hal ini adalah “kanan”, maka tampilkan “Mobil sedang belok kanan”.





**Thank  
YOU**

