Materi Lab 9

Inheritance, Encapsulation, Operator Overloading, Default Parameter

Inheritance

Pada minggu lalu, kalian telah belajar suatu konsep baru dalam pemrograman yaitu ***Object Oriented Programming (OOP)*** menggunakan ***Class*** di python. Pada minggu ini, kalian akan belajar mengenai sebuah konsep lebih lanjut mengenai **OOP**, yaitu ***inheritance*** atau **pewarisan**.

Inheritance adalah sebuah konsep yang memungkinkan sebuah ***Class*** untuk mewarisi semua ***properties*** dan ***methods*** yang dimiliki oleh kelas lain. Dalam konsep inheritance terdapat dua class yang memiliki peran/role yang berbeda, yaitu:

- ***Parent class***

  Parent class adalah *class* yang *properties* dan *methods*-nya **diwariskan** kepada *class* lain. *Parent class* juga disebut sebagai ***base class***.
- ***Child class***

  Child class adalah class yang **mewarisi** *properties* dan *methods* dari kelas lainnya.

Supaya kalian mendapat gambaran yang lebih jelas, akan diberikan sebuah kasus sebagai berikut. Terdapat sebuah *class* bernama ***Person***, kemudian terdapat *class* lainnya yang mewarisi *class* ***Person,*** yaitu *class* ***Student***.

1. Pertama buat terlebih dahulu *class* ***Person*** yang memiliki atribut ***first_name*** dan ***last_name*** serta memiliki sebuah *method* ***printname()*** yang akan mencetak nama lengkap. Dalam kasus ini *class* ***Person*** memiliki peran sebagai ***parent class***. Dalam membuat ***parent class*** kalian dapat menggunakan *syntax* yang biasa kalian gunakan untuk membuat *class* pada umumnya.

```
class Person():
    def __init__(self, first_name, last_name):
        self.first_name = first_name
        self.last_name = last_name
    def printname(self):
        print(f"{self.first_name} {self.last_name}")
```

Ingatkah kalian mengenai *method* ***__init__(self, *args)*** ? Method tersebut berguna untuk meng-*assign* nilai dari atribut-atribut suatu *object* ketika *object* pertama kali diinisiasi.

2. Setelah ***parent class*** sudah dibuat, kalian bisa mulai membuat dua *class* lain yang mewarisi ***parent class*** tersebut, yaitu *class* ***Student***. *Class* ***Student*** ini mewarisi semua atribut dan *methods* yang dimiliki oleh *class* ***Person***.

```
class Student(Person):
    def __init__(self, first_name, last_name, faculty):
        super().__init__(first_name, last_name)
        self.faculty = faculty

    def printfaculty(self):
        print(self.faculty)
```

Jika kalian perhatikan terdapat beberapa perbedaan *syntax* dalam membuat sebuah *class* yang mewarisi *class* lainnya. Perbedaan tersebut dapat dirangkum sebagai berikut:

a. Memasukkan nama ***parent class*** sebagai parameter ketika membuat ***child class***. Secara umum, *syntax*-nya adalah sebagai berikut:

```
class ChildClass(ParentClass):
    pass
```

b. Memanggil *method* ***__init__()*** yang dimiliki oleh ***parent class***. Mengapa hal ini diperlukan? Ketika ***child class*** mendefinisikan *method* ***__init__()*** maka yang terjadi adalah *method* ***__init__()*** yang dimiliki oleh ***parent class*** tertimpa/ter-*override* oleh *method* ***__init__()*** yang dimiliki oleh ***child class.*** Supaya ***child class*** tetap mewarisi ***parent class*** maka perlu memanggil kembali *method* ***__init__() parent class.***

Terdapat dua cara untuk memanggil *method* ***__init__()*** yang dimiliki ***parent class,*** yaitu dengan menggunakan fungsi ***super()*** atau memanggil langsung nama dari ***parent class***.

```
class Student(Person):
    def __init__(self, first_name, last_name, faculty):
        Person.__init__(self, first_name, last_name)
        self.faculty = faculty
```

Apakah kalian tahu perbedaan *syntax* keduanya? (Perhatikan ada dan tidaknya parameter ***self***)

3. Saat membuat ***child class*** kalian juga dapat menambahkan atribut dan method baru (seperti pada nomor 2). Selain itu, kalian juga dapat melakukan ***method overriding***, yaitu menimpa implementasi *method* yang dimiliki oleh ***parent class*** dengan implementasi yang baru. Sebagai contoh kalian ingin membuat implementasi yang berbeda pada *method* ***printname()*** yang diwariskan.

```
class Student(Person):
    def __init__(self, first_name, last_name, faculty):
        Person.__init__(self, first_name, last_name)
        self.faculty = faculty

    def printfaculty(self):
        print(self.faculty)

    def printname(self):
        print(f"{self.first_name} from {self.faculty}")
```

Untuk lebih memahami perbedaannya, cobalah kode di bawah ini.

```
person1 = Person("Dekdepe", "Keren")
person1.printname()

student1 = Student("Sung", "Jin-Woo", "Computer Science")
student1.printname()
```

Encapsulation

Dalam konsep OOP, pembatasan akses terhadap method dan variabel dapat dilakukan. Ini untuk mencegah pengubahan data langsung. Konsep ini dinamakan *encapsulation*.

```
class Angkatan2020:

    def __init__(self):
        self.__nama_angkatan = "CSUI2020"

    def __str__(self):
        return "Nama Angkatan: {}".format(self.__nama_angkatan)

    def set_nama_angkatan(self, nama):
        self.__nama_angkatan= nama

angkatan_2020 = Angkatan2020()
# Angkatan2020 diterima di Fasilkom
print(angkatan_2020) #CSUI2020

# GAGAL mengubah nama angkatan TANPA menggunakan method setter
angkatan_2020.__nama_angkatan="Chronos"
print(angkatan_2020) #CSUI2020

# BERHASIL mengubah nama angkatan DENGAN menggunakn method setter
angkatan_2020.set_nama_angkatan("Chronos")
print(angkatan_2020) #Chronos
```

Penggunaan double underscore di depan variable dalam variabel *__nama_angkatan* menyebabkan interpreter mengategorikannya sebagai private attribute.

Pengubahan nilai dari private attribute tidak dapat dilakukan langsung sehingga terbatas pengubahannya didalam fungsionalitas class. Untuk mengubahnya diperlukan method setter yang merupakan bagian dari fungsionalitas class. Dalam program di atas adalah method *set_nama_angkatan*.

Operator Overloading

Penggunaan method special dapat meng-*overload behavior* dari operasi-operasi standar suatu instance dari class.

```
class BilanganBulat:
    def __init__(self, x=0):
        self.x = x

    def __str__(self):
        return "Bilangan bulat bernilai {}".format(self.x)

    def __add__(self, other):
        x = self.x - other.x
        return x

a = BilanganBulat()
b = BilanganBulat(9)

print(a+b) # -9
```

Contoh program di atas akan mengeluarkan output -9 karena penjumlahan di-*overload* sehingga menjadi pengurangan. Lihat METHOD SPECIAL *__add__* di dalam program. Adapun daftar beberapa method special untuk overloading adalah sebagai berikut.

Math-like Operators		
Expression	**Method name**	**Description**
x + y	`__add__()`	Addition
x − y	`__sub__()`	Subtraction
x * y	`__mul__()`	Multiplication
x / y	`__div__()`	Division
x == y	`__eq__()`	Equality
x > y	`__gt__()`	Greater Than
x >= y	`__ge__()`	Greater Than or Equal
x < y	`__lt__()`	Less Than
x <= y	`__le__()`	Less Than or Equal
x != y	`__ne__()`	Not Equal
Sequence Operators		
len(x)	`__len__()`	length of the sequence
x in y	`__contains__()`	does the sequence y contain x
x[key]	`__getitem__()`	access element *key* of sequence x
x[key]=y	`__setitem__()`	set element *key* of sequence x to value y
General Class Operations		
x=myClass()	`__init__()`	constructor
print (x), str(x)	`__str__()`	convert to a readable string
	`__repr__()`	print a representation of x
	`__del__()`	finalizer, called when x is garbage collected

Tabel method special

Default Parameter

Penggunaan default parameter dalam pemanggilan function atau method mudah untuk dilakukan dalam Python. Perhatikan alur kode berikut.

```
class Angkatan2020:

    def __init__(self, universitas="Universitas Indonesia"):
        self.universitas=universitas

    def __str__(self):
        return "Angkatan 2020 dari institusi: {}".format(self.universitas)



# TIDAK MENAMBAHKAN argumen Universitas
angkatan_A = Angkatan2020()
print(angkatan_A) # Angkatan 2020 dari institusi: Universitas Indonesia

# MENAMBAHKAN argumen Universitas
angkatan_B = Angkatan2020("Institut Teknologi Bandung")
print(angkatan_B) # Angkatan 2020 dari institusi: Institut Teknologi Bandung
```

Dalam method *__init__* penggunaan parameter universitas digunakan untuk mengetahui asal universitas suatu angkatan. Apabila nama universitas tidak dimasukkan sebagai argumen, maka suatu angkatan secara *default* adalah angkatan dari Universitas Indonesia.