Nama : Stefen Tjung

NIM : 121140057

Kelas : RB

1. Kelas, Objek, Konstruktor dan Decorator

   Kelas atau class pada python bisa kita katakan sebagai sebuah blueprint (cetakan) dari objek (atau instance) yang ingin kita buat. Kelas adalah cetakannya atau definisinya, sedangkan objek (atau instance) adalah objek nyatanya.

   Konstruktor adalah sebuah fungsi yang akan dipanggil pertama kali saat sebuah objek di-instantiasi-kan. Fungsi tersebut harus selalu bernama __init__() .

   Decorator adalah higher-order function yang bisa menerima fungsi sebagai argumen dan mengeluarkan fungsi. Jadi jika seseorang bertanya kepada anda apa itu decorator, anda bisa menjawabnya dengan jawaban yang sederhana.

   Berikut adalah contoh class nama dengan konstruktor dan method beserta decorator yang berfungsi untuk menambahkan prefix/awalan :

```python
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def greet(self):
        print(f"Hello, my name is {self.name} and I am {self.age} years old.")

def add_prefix(prefix):
    def decorator(func):
        def wrapper(*args, **kwargs):
            result = func(*args, **kwargs)
            return f"{prefix} {result}"
        return wrapper
    return decorator


@add_prefix("Greeting:")
def greet_with_prefix(self):
    self.greet()

# Create a Person object and call the decorated greet method
person = Person("Alice", 25)
greet_with_prefix(person)
```

2. Enkapsulasi dan Abstraksi

Abstraksi dan enkapsulasi data keduanya sering digunakan sebagai sinonim. Keduanya hampir sinonim karena abstraksi data dicapai melalui proses enkapsulasi.

Abstraksi digunakan untuk menyembunyikan detail internal dan hanya menampilkan fungsi. Mengabstraksi sesuatu berarti memberi nama pada sesuatu sehingga nama tersebut menangkap inti dari apa yang dilakukan suatu fungsi atau keseluruhan program.

Access Modifiers adalah sebuah konsep di dalam pemrograman berorientasi objek di mana kita bisa mengatur hak akses suatu atribut dan fungsi pada sebuah class. Konsep ini juga biasa disebut sebagai enkapsulasi, di mana kita akan mendefinisikan mana atribut atau fungsi yang boleh diakses secara terbuka, mana yang bisa diakses secara terbatas, atau mana yang hanya bisa diakses oleh internal kelas alias privat.

Access Modifiers dibagi menjadi 3 yakni :

1. Public
   Variabel atau atribut yang memiliki hak akses publik bisa diakses dari mana saja baik dari luar kelas mau pun dari dalam kelas.
2. Protected
   Variabel atau atribut yang memiliki hak akses protected hanya bisa diakses secara terbatas oleh dirinya sendiri (yaitu di dalam internal kelas), dan juga dari kelas turunannya.
3. Private
   Modifier selanjutnya adalah private. Setiap variabel di dalam suatu kelas yang memiliki hak akses private maka ia hanya bisa diakses di dalam kelas tersebut. Tidak bisa diakses dari luar bahkan dari kelas yang mewarisinya.

```python
class MyClass:
    def __init__(self, public_var, protected_var, private_var):
        self.public_var = public_var
        self._protected_var = protected_var
        self.__private_var = private_var

    def public_method(self):
        print("This is a public method.")

    def _protected_method(self):
        print("This is a protected method.")

    def __private_method(self):
        print("This is a private method.")


my_object = MyClass("public", "protected", "private")
print(my_object.public_var)          # prints "public"
print(my_object._protected_var)      # prints "protected"
print(my_object._MyClass__private_var)  # prints "private"
my_object.public_method()           # prints "This is a public method."
my_object._protected_method()       # prints "This is a protected method."
my_object._MyClass__private_method()   # prints "This is a private method."
```

3. Inheritance and Polymorphism

Inheritance adalah salah satu mekanisme di konsep OO yang mengizinkan kita untuk mendefinisikan sebuah class baru berdasarkan class yang sebelumnya telah dideklarasikan. Melalui konsep inheritance, sebuah class baru dapat memiliki atribut dan fungsi pada class yang sebelumnya telah didefinisikan. Pada konsep inheritance, atribut/fungsi yang akan diwariskan hanyalah atribut/fungsi dengan access modifier public, atribut/fungsi dengan access modifier private tidak akan diturunkan.

Selain dapat mendefinisikan ulang nilai dari atribut yang diwarisi oleh parent class seperti pada contoh di atas, kita juga dapat mendefinisikan ulang fungsi yang telah diwarisi oleh parent class. Saat kita mendefinisikan kembali fungsi yang telah diwarisi oleh parent class, secara tidak langsung kita telah menerapkan salah satu mekanisme yang secara khusus pada paradigma OO disebut dengan istilah polymorphism.

```python
class Animal:
    def __init__(self, name):
        self.name = name

    def speak(self):
        print("")


class Dog(Animal):
    def __init__(self, name):
        super().__init__(name)

    def speak(self):
        print(f"{self.name} barks.")


class Cat(Animal):
    def __init__(self, name):
        super().__init__(name)

    def speak(self):
        print(f"{self.name} meows.")


def make_animal_speak(animal):
    animal.speak()

animal = Animal("Generic Animal")
dog = Dog("Fido")
cat = Cat("Whiskers")

make_animal_speak(animal)   # prints ""
make_animal_speak(dog)      # prints "Fido barks."
make_animal_speak(cat)      # prints "Whiskers meows."
```

4. Setter and Getter

Untuk tujuan enkapsulasi data, sebagian besar bahasa berorientasi objek menggunakan metode getter dan setter. Ini karena kita ingin menyembunyikan atribut kelas objek dari kelas lain sehingga tidak ada modifikasi data yang tidak disengaja yang terjadi oleh metode di kelas lain.

Seperti namanya, getter adalah metode yang membantu mengakses atribut pribadi atau mendapatkan nilai atribut pribadi dan setter adalah metode yang membantu mengubah atau menetapkan nilai atribut pribadi.

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def get_name(self):
        return self.name

    def set_name(self, name):
        self.name = name

    def get_age(self):
        return self.age

    def set_age(self, age):
        if age < 0:
            raise ValueError("Age cannot be negative.")
        else:
            self.age = age

person = Person("Alice", 25)
person.set_name("Bob")
person.set_age(30)

print(person.get_name())   # prints "Bob"
print(person.get_age())    # prints 30

try:
    person.set_age(-5)
except ValueError as e:
    print(e)   # prints "Age cannot be negative."
```