PRAKTIKUM
PEMROGRAMAN BERBASIS OBJEK

TUGAS 6

Nama : Umy Afifah

NIM : 121140087

Kelas : RB

INSTITUT TEKNOLOGI SUMATERA

TAHUN AJARAN 2022/2023

LAMPUNG SELATAN

2023

A. ABSTRAKSI

Abstraksi dalam Python merujuk pada proses mengatasi kompleksitas dengan cara menyembunyikan informasi yang tidak relevan dari pengguna. Dalam OOP, ini adalah konsep penting yang memungkinkan pengguna untuk mengimplementasikan logika yang lebih kompleks di atas abstraksi yang disediakan tanpa perlu memahami atau memikirkan semua kompleksitas latar belakang yang tersembunyi.

Metode abstrak adalah metode yang dideklarasikan tetapi tidak memiliki implementasi. Metode abstrak di kelas dasar memberikan identifikasi tentang fungsionalitas yang harus diimplementasikan oleh semua subkelasnya. Namun, implementasi metode abstrak dapat berbeda dari satu subkelas ke subkelas lainnya, sehingga isi dari metode seringkali hanya berisi pernyataan "pass". Setiap subkelas dari kelas dasar akan menggunakan metode ini dengan implementasinya sendiri. Kelas yang mengandung metode abstrak disebut kelas abstrak.

Python menyediakan modul ABC (*Abstract Base Classes*) untuk menggunakan abstraksi dalam program Python. Metode pada abstrak kelas biasanya ditandai dengan *decorator* @abstactmethod. Contoh sintaknya sebagai berikut:

```python
from abc import ABC, abstractmethod

# kelas Mobil merupakan kelas abstrak dan tidak bisa diinstansiasi
class Mobil(ABC):
    @abstractmethod
    def berjalan(self):
        # fungsi ini wajib di implementasikan pada child-class
        pass

    @abstractmethod
    def berhenti(self):
        # fungsi ini wajib di implementasikan pada child-class
        pass
```

Ketika sebuah kelas dijadikan kelas abstrak, maka error akan terjadi apabila kelas child tidak mengimplementasikan semua fungsi yang memiliki decorator @abstractmethod pada kelas induknya. Untuk membuat kelas child, kita perlu menuliskan implementasi dari semua fungsi yang diidentifikasi sebagai metode abstrak pada kelas induk dengan menggunakan decorator @abstractmethod. Dengan demikian, kelas child harus menyediakan implementasi dari semua metode abstrak yang diturunkan dari kelas induk

```python
class MobilSport(Mobil):
    def berjalan(self):
        print("Mobil Sport berjalan dengan cepat")

    def berhenti(self):
        print("Mobil Sport berhenti")
```

```
PS F:\Andhika P P\Code\Asprak-PBO\MG5>
/ASUS/AppData/Local/Programs/Python/Py
hon.exe "f:/Andhika P P/Code/Asprak-PB
O/MG5/ABC.py"
Mobil Sport berjalan dengan cepat
PS F:\Andhika P P\Code\Asprak-PBO\MG5>
```

Tidak memiliki perbedaan yang signifikan dengan class yang tidak menggunakan modul ABC, kelas abstrak juga dapat menggunakan konstruktor serta isi metodenya tidak selalu memiliki nilai *pass*, dapat menggunakan fungsi biasa. Contohnya:

```python
from abc import ABC, abstractmethod
# kelas Mobil merupakan kelas abstrak dan tidak bisa diinstansiasi
class Mobil(ABC):
    def __init__(self, warna):
        self.warna = warna

    @abstractmethod
    def berjalan(self):
        # fungsi ini wajib di implementasikan pada child-class
        pass

    @abstractmethod
    def berhenti(self):
        # fungsi ini wajib di implementasikan pada child-class
        pass

class MobilSport(Mobil):
    def __init__(self, warna):
        super().__init__(warna)

    def berjalan(self):
        print(f"Mobil Sport berjalan dengan warna {self.warna}")

    def berhenti(self):
        print("Mobil Sport berhenti")

car = MobilSport("Merah")
car.berjalan()
```

```
PS F:\Andhika P P\Code\Asprak-PBO\MG5> & C:/Users/AS
US/AppData/Local/Programs/Python/Python310/python.ex
e "f:/Andhika P P/Code/Asprak-PBO/MG5/ABC.py"
Mobil Sport berjalan dengan warna Merah
PS F:\Andhika P P\Code\Asprak-PBO\MG5>
```

B. INTERFACE

Di Python, interface diimplementasikan melalui konsep Abstract Base Class (ABC) atau protokol. Kedua konsep ini memungkinkan kita untuk mendefinisikan spesifikasi dengan cara mengimplementasikan fitur-fitur tertentu tanpa memperhatikan implementasi di dalamnya. Terdapat dua jenis interface di Python, yaitu informal dan formal.

Interface informal diimplementasikan melalui kelas dan mendefinisikan metode yang dapat diganti tanpa ditegakkan secara formal. Dalam Python, informal interface disebut protokol karena sifatnya yang tidak formal dan tidak memaksa penggunaannya. Beberapa metode umum yang digunakan dalam informal interface adalah seperti yang telah dijelaskan sebelumnya.

```python
class Hewan:
    def __init__(self, hewan):
        self.__list_hewan = hewan

    def __len__(self):
        return len(self.__list_hewan)

    def __contains__(self, hewan):
        return hewan in self.__list_hewan

class KebunBinatang(Hewan):
  pass


ragunan = KebunBinatang(["Rusa", "Harimau", "Unta"])
# Cek banyaknya hewan
print(len(ragunan))
# Cek apakah terdapat hewan tertentu
print("Rusa" in ragunan)
print("Gajah" in ragunan)
print("Unta" not in ragunan)
```

```
/run/media/d
3
True
False
False
```

Sementara itu, interface formal diberlakukan secara formal dan diimplementasikan melalui penggunaan ABC (Abstract Base Class). Ketika kita mendefinisikan kelas abstrak, kita juga mendefinisikan metode pada kelas dasar sebagai metode abstrak. Semua objek yang berasal dari kelas dasar harus mengimplementasikan metode tersebut. Sebagai contoh, hal ini telah dijelaskan pada bagian tentang Abstraksi

```python
from abc import ABC
from abc import abstractmethod

class GPS(ABC):

    @abstractmethod
    def aktifkan_gps(self):
        pass

    @abstractmethod
    def matikan_gps(self):
        pass

    @abstractmethod
    def get_location(self):
        pass
```

```python
class Smartphone(GPS):
    def aktifkan_gps(self):
        print("gps aktif")

    def matikan_gps(self):
        print("gps mati")

    def get_location(self):
        print("lokasi anda di...")

samsung=Smartphone()

# samsung.aktifkan_asisten()
samsung.get_location()
```

C. METACLASS

Di Python, konsep metaclass tersedia secara default dan digunakan dalam OOP. Metaclass memungkinkan kita untuk membuat class secara kustom dengan menggunakan kata kunci "type". Secara default, setiap class yang dibuat di Python adalah sebuah instance dari tipe metaclass. Fungsi type() dapat digunakan untuk membuat class secara dinamis karena dapat membuat instance baru dari metaclass type.

Dengan menggunakan fungsi type(), kita dapat membuat class kustom dengan mendefinisikan beberapa argumen seperti nama kelas, tupel dari class parent, dan beberapa atribut lainnya. Hal ini memungkinkan pengguna untuk membuat class secara dinamis dan membuat struktur program menjadi lebih fleksibel.

Contoh implementasi menggunakan type():

```
angka = 420
pi = 3.14
nama = "Rangga"

print("Tipe dari angka adalah", type(angka))
print("Tipe dari pi adalah", type(pi))
print("Tipe dari nama adalah", type(nama))
```

```
/run/me/dhika/Multi/Project/P
Tipe dari angka adalah <class 'int'>
Tipe dari pi adalah <class 'float'>
Tipe dari nama adalah <class 'str'>
```

D. KESIMPULAN

1. Interface adalah sebuah blueprint yang mendefinisikan perilaku yang harus dimiliki oleh kelas-kelas yang mengimplementasikannya. Kita perlu menggunakan interface ketika kita ingin memastikan bahwa class-class yang kita buat memiliki perilaku yang konsisten.

2. Kelas abstrak adalah sebuah kelas yang tidak bisa diinstansiasi secara langsung, melainkan digunakan sebagai kerangka dasar atau blueprint untuk kelas turunannya. Kelas abstrak ini memiliki minimal satu atau lebih metode abstrak yang harus diimplementasi pada kelas turunannya. Kita perlu menggunakan kelas abstrak ketika kita ingin membuat kerangka dasar untuk kelas-kelas turunannya, dan ingin memastikan bahwa setiap kelas turunan yang dihasilkan memiliki implementasi metode yang sama. Dengan menggunakan kelas abstrak, kita dapat memastikan bahwa setiap kelas turunan akan memiliki perilaku yang konsisten, karena mereka harus mengimplementasikan metode yang sama.

3. Kelas konkret adalah kelas yang dapat dibuat sendiri, dan menyediakan implementasi lengkap untuk semua metodenya. Itu tidak mengandung metode abstrak, tidak seperti kelas abstrak. Kita perlu menggunakan kelas konkret ketika kita ingin membuat objek yang semua metodenya telah diimplementasikan sepenuhnya dan siap digunakan.

4. Metaclass adalah sebuah tipe kelas yang digunakan untuk membuat kelas. Kita perlu menggunakan metaclass ketika kita ingin membuat kelas dengan perilaku yang spesifik dan di luar batasan class-class biasa. Perbedaannya dengan inheritance biasa adalah metaclass memungkinkan kita untuk memodifikasi pembuatan class secara lebih spesifik, sementara inheritance hanya memungkinkan kita untuk mewarisi perilaku class dasar

DAFTAR PUSTAKA

https://realpython.com/python-interface/#python-interface-overview
https://pythonguides.com/python-interface/
https://www.tutorialspoint.com/What-is-a-metaclass-in-Python
https://www.mygreatlearning.com/blog/abstraction-in-python/#:~:text=Abstraction%20in%20python%20is%20defined,oriented%20programming%20(OOP)%20languages.
Modul Praktikum PBO 6 - ITERA
Modul Kelas Abstrak dan Interface Pertemuan 9 - ITERA