PRAKTIKUM
PEMROGRAMAN BERBASIS OBJEK

TUGAS 6

Nama : Bendry Lakburlawal

NIM : 121140111

Kelas : RB

INSTITUT TEKNOLOGI SUMATERA

TAHUN AJARAN 2022/2023

LAMPUNG SELATAN

2023

BAB I

1. ABSTRAKSI

A. Konsep Abstraksi

Abstraksi dalam python didefinisikan sebagai proses penanganan kompleksitas dengan menyembunyikan informasi yang tidak perlu dari pengguna. Ini adalah salah satu konsep inti dari bahasa pemrograman berorientasi objek (OOP). Ini memungkinkan pengguna untuk mengimplementasikan logika yang lebih kompleks di atas abstraksi yang disediakan tanpa memahami atau bahkan memikirkan semua kompleksitas latar belakang/back-end yang tersembunyi.

B. Kelas Abstrak dalam Python

Metode abstrak adalah metode yang dideklarasikan, tetapi tidak mengandung implementasi. Metode abstrak di kelas dasar mengidentifikasi fungsionalitas yang harus diimplementasikan oleh semua subkelasnya. Namun, karena penerapan metode abstrak akan berbeda dari satu subkelas ke subkelas lainnya, seringkali isi metode hanya terdiri dari pernyataan pass. Setiap subclass dari base class akan menggunakan metode ini dengan implementasinya. Kelas yang berisi metode abstrak disebut kelas abstrak.

C. Implementasi Kelas Abstrak dengan Modul ABC

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:

```
1  from abc import ABC, abstractmethod
2
3  class Mahasiswa(ABC): #--> mewarisi kelas ABC
4    @abstractmethod #decorator
5    def KTP(self):
6      pass
7
8    @abstractmethod #decorator
9    def Prodi(self):
10     pass
11
12 Bendry=Mahasiswa()
```

Pada kelas abstrak akan terjadi *Error* apabila tidak mengimplementasi kelas *child*. Untuk membuat kelas *child* perlu mengimplementasikan semua fungsi yang memiliki tanda *decorator @abstractmethod* yang ada pada *parent class*. Berikut contoh sintaknya:

main.py

```python
from abc import ABC, abstractmethod

class Mahasiswa(ABC): #--> mewarisi kelas ABC

  @abstractmethod #decorator
  def KTP(self):
    pass

  @abstractmethod #decorator
  def Prodi(self):
    pass

class Bendry(Mahasiswa):
  def __init__(self, nama, NIK):
    self.nama=nama
    self.NIK=NIK

  def KTP(self): #fungsi sama seperti pada parent
    print("Nama :",self.nama)
    print("KTP  :",self.NIK)

  def Prodi(self): #fungsi sama seperti pada parent
    print(f"{self.nama} adalah mahasiswa IF")

Mhs=Bendry("Bendry","8103010112010001")
Mhs.KTP()
Mhs.Prodi()
```

```
Nama : Bendry
KTP  : 8103010112010001
Bendry adalah mahasiswa IF
```

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:

main.py

```python
from abc import ABC, abstractmethod

class Mahasiswa(ABC): #--> mewarisi kelas ABC
  def __init__(self, nama, NIK): #konstruktor
    self.nama=nama
    self.NIK=NIK

  @abstractmethod #decorator
  def KTP(self):
    pass

  @abstractmethod #decorator
  def Prodi(self):
    pass

  def Kampus(self): #fungsi biasa
    print("ITERA")

class Bendry(Mahasiswa):
  def __init__(self, nama, NIK):
    super().__init__(nama, NIK)

  def KTP(self): #fungsi sama seperti pada parent
    print("Nama :",self.nama)
    print("KTP  :",self.NIK)

  def Prodi(self): #fungsi sama seperti pada parent
    print(f"{self.nama} adalah mahasiswa IF")

Mhs=Bendry("Bendry","8103010112010001")
Mhs.KTP()
Mhs.Prodi()
Mhs.Kampus()
```

```
Nama : Bendry
KTP  : 8103010112010001
Bendry adalah mahasiswa IF
ITERA
```

2. INTERFACE

A. Definisi Interface

Pada Python, konsep interface menggunakan sebuah konsep abstraksi (ABC) atau protokol. Kedua konsep tersebut mendefinisikan spesifikasi kepada pengguna dengan mengimplementasi fitur-fitur tertentu tanpa memperhatikan implementasi di dalamnya. Interface terbagi menjadi 2, yaitu:

a. Informal Interface

informal Interface di Python adalah kelas. Ini mendefinisikan metode yang dapat diganti tetapi tanpa penegakan paksa. Interface informal dalam python disebut sebagai protokol karena bersifat informal dan tidak dapat ditegakkan secara formal. Metode yang umum digunakan yang digunakan untuk melakukan beberapa operasi contoh:

main.py

```
class Manusia:
  def __init__(self, orang):
    self.__orang=orang

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

  def __contains__(self, orang):
    return orang in self.__orang

class Populasi(Manusia):
  pass

ITERA = Populasi(["Mahasiswa", "Dosen", "Satpam"])
print(len(ITERA))
print("Mahasiswa" in ITERA)
print("Buaya" in ITERA)
```

```
3
True
False
```

b. Formal Interface

Interface formal adalah Interface yang diberlakukan secara formal. Untuk membuat Interface formal perlu menggunakan ABC (Kelas Basis Abstrak). ABC dijelaskan saat mendefinisikan kelas yang bersifat abstrak, kita juga mendefinisikan metode pada kelas dasar sebagai metode abstrak.

Objek apapun yang kami peroleh dari kelas dasar dipaksa untuk mengimplementasikan metode tersebut. Contohnya seperti yang telah dilampirkan pada bagian Abstraksi, yaitu:

main.py

```
from abc import ABC, abstractmethod

class Mahasiswa(ABC): #--> mewarisi kelas ABC

  @abstractmethod #decorator
  def KTP(self):
    pass

  @abstractmethod #decorator
  def Prodi(self):
    pass

```

Gambar lanjutan

```python
class Bendry(Mahasiswa):
  def __init__(self, nama, NIK):
    self.nama=nama
    self.NIK=NIK

  def KTP(self): #fungsi sama seperti pada parent
    print("Nama :",self.nama)
    print("KTP  :",self.NIK)

  def Prodi(self): #fungsi sama seperti pada parent
    print(f"{self.nama} adalah mahasiswa IF")

Mhs=Bendry("Bendry","8103010112010001")
Mhs.KTP()
Mhs.Prodi()
```

```
Nama : Bendry
KTP  : 8103010112010001
Bendry adalah mahasiswa IF
```

3. METACLASS

Metaclass adalah konsep OOP yang ada dalam semua kode Python secara default. Python menyediakan fungsionalitas untuk membuat metaclass kustom dengan menggunakan kata kunci type. Type adalah sebuah metaclass yang instance-nya adalah class. Setiap class yang dibuat di Python adalah sebuah instance dari tipe metaclass. Fungsi type() dapat membuat class secara dinamis karena memanggil type() akan membuat instance baru dari metaclass type. Contoh implementasi menggunakan type:

main.py

```python
iniinteger=10
inifloat=0.1
inistring="Bendry"

print(type(iniinteger))
print(type(inifloat))
print(type(inistring))
```

```
<class 'int'>
<class 'float'>
<class 'str'>
```

Contoh implementasi parameter *metaclass*

main.py

```python
class Manusia(type):
  def __new__(cls, name, bases, class_dict):
    meta_=super().__new__(cls, name, bases, class_dict)
    meta_.test="Metakelassss"
    return meta_

class Bendry(object, metaclass=Manusia):
  pass

orang=Bendry()
print(orang.test)
```

```
Metakelassss
```

BAB II

KESIMPULAN

1. Interface merupakan spesifikasi yang menjelaskan setiap fungsi yang harus ada di dalam suatu kelas tanpa memberikan implementasi dari fungsi tersebut. Interface diperlukan saat memastikan *class-class* di dalamnya memiliki implementasi fungsi tertentu.
2. Kelas abstrak adalah kelas yang tidak dapat dibuat sendiri tetapi berfungsi sebagai basis atau tempat untuk mewarisi kelas lain. Ini biasanya berisi beberapa fungsi umum yang dapat digunakan kembali oleh kelas turunannya. Penggunaan kelas abstrak ketika ingin mendefinisikan kelas dasar yang menyediakan beberapa fungsi umum tetapi tidak dapat dibuat sendiri karena tidak lengkap atau tidak sepenuhnya diimplementasikan. Kelas abstrak sering digunakan terdapat beberapa fitur yang secara umum dimiliki oleh semua objek, berbeda dengan interface yang digunakan jika terdapat fitur yang ingin diimplementasi.
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 kelas yang mendefinisikan perilaku kelas lain. Ini dapat dianggap sebagai kelas dari kelas. Saat kita membuat kelas baru, kita bisa menentukan metaclass yang akan menentukan bagaimana kelas baru dibuat dan bagaimana perilakunya. Kita perlu menggunakan metaclass saat kita ingin menyesuaikan perilaku kelas di luar apa yang bisa dicapai melalui *inheritance* biasa. Metaclass berbeda dari pewarisan biasa di mana pewarisan menentukan hubungan antara kelas dan kelas induknya, sedangkan metaclass menentukan perilaku kelas dan bagaimana kelas-kelas itu dibuat.

DAFTAR PUSTAKA

Kumar, B. (2020, December 24). *Introduction To Python Interface*. Python Guides. Retrieved April 11, 2023, from https://pythonguides.com/python-interface/

Lutz, M. (2009). *Learning Python*. O'Reilly Media, Incorporated.

Murphy, W. (n.d.). *Implementing an Interface in Python – Real Python*. Real Python. Retrieved April 11, 2023, from https://realpython.com/python-interface/#python-interface-overview

Pratama, A. P., B, A. W., Larasati, N. A., Rachman, I. F., S, E. J., & Fadhila R, M. A. (2022). KELAS ABSTRAK DAN INTERFACE (KELAS ABC, METACLASS). *16*.

Singh, S. (2022, August 18). *What is a metaclass in Python*. Tutorialspoint. Retrieved April 11, 2023, from https://www.tutorialspoint.com/What-is-a-metaclass-in-Python

Tim Pengajar ITERA. (2022). Kelas Abstrak dan Interface. In (Vol. 31).