RESUME

PRAKTIKUM PBO

Disusun oleh:

Nama : Dimas Saputra

NIM : 121140059

Kelas : RB

PROGRAM STUDI TEKNIK INFORMATIKA

INSTITUT TEKNOLOGI SUMATERA

2023

Modul 1

Bahasa pemrograman python dibuat oleh Guido Van Rossum pada tahun 1980-an akhir di *Centrum Wiskunde & Informatica* Belanda. Python mendukung banyak paradigma pemrograman seperti object-oriented, functional, dan structured. Python sangat mementingkan *readability* pada kode, untuk mengimplementasikan filosofi itu python tidak menggunakan kurung kurawal ({}) atau *keyword* (ex. Start, begin, end) sebagai gantinya menggunakan spasi (*white space*) untuk memisahkan blok-blok kode.

a. Sintaks dasar
   Semua perintah yang bisa dieksekusi python disebut statement. Python tidak menggunakan kurung kurawal sebagai *grouping* blok kode melainkan menggunakan spasi ataupun tab (4 spasi) atau identasi. Kode yang berada di blok yang sama harus memiliki jumlah spasi yang sama di awal
b. Variabel dan tipe data primitive
   Variabel merupakan lokasi penyimpanan yang berguna untuk menyimpan suatu data atau suatu nilai. Dalam mendeklarasikan suatu variable dalam pemrograman diperlukan tipe-tipe data yaitu: bool, int, float, string.
c. Operator
   Python memiliki sejumlah operator yaitu: operator aritmatika, operator perbandingan, operator penugasan, operator logika, operator bitwise, operator identitas, dan operayor keanggotaan.
d. Tipe data bentukan
   Dalam python ada 4 tipe data bentukan dengan perbedaan penggunaan yaitu: list, tuple, set, dan dictionary.
e. Percabangan
   Dalam Bahasa pemrograman python, terdapat beberapa percabangan yaitu: IF, IF-ELSE, IF-ELIF, dan nested IF.
f. Perulangan
   Dalam Bahasa pemrograman python, terdapat dua jenis perulangan, yaitu perulangan for dan perulangan while.
g. Fungsi
   Dengan menggunakan fungsi, dapat mengeksekusi suatu blok kode tanpa harus menulisnya berulang-ulang.

Contoh kode variabel dan tipe data:

```
1   benar = True
2   angka = 12
3   pi = 3.14
4   nama = "Josua"
```

Contoh kode tipe data bentukan:

```
buah = ["apel", "jeruk", "semangka", "salak"] # list
angka = (1, 2, 3, 4) # tuple
data = {"satu", "dua", 3, 4} # set
alamat = {"kota":"Bandar Lampung","provinsi":"Lampung"} # dictionary
```

Contoh kode percabangan, operator, sintaks:

```
password = 12345

if password == 12345:
    print("password sesuai")
```

Contoh kode perulangan:

```
buah = ["apel", "jeruk", "semangka", "salak"]

for i in range(len(buah)):
    print(buah[i])
```

Contoh kode fungsi:

```
class Apel:
    def __init__(self):
        pass

apel1 = Apel()
```

Modul 2

1. Kelas
   Kelas atau class pada python bisa dikatakan sebagai sebuah blueprint dari objek yang ingin dibuat. Dengan menggunakan kelas dapat mendesain objek secara bebas. Kelas berisi dan mendefinisikan atribut/property dan metode untuk objeknya nanti. Satu buah kelas dapat membuat banyak objek dan kelas sendiri tidak bisa langsung digunakan, harus diimplementasikan sebuah objek dahulu, dapat disebut instansiasi.
   - Atribut/property
     Dalam suatu kelas, dideklarasikan sebuah variabel yang akan disebut sebagai sebuah atribut. Terdapat 2 jenis atribut, yaitu atribut kelas dan atribut objek. Atribut kelas merupakan sifat yang dimiliki oleh sebuah kelas dan juga akan dimiliki oleh setiap

objek. Sedangkan atribut objek adalah sebuah atribut dari masing-masing objek atau atribut yang ada pada method.

- Method
  Method adalah suatu fungsi yang terdapat di dalam kelas. Method dapat diibaratkan sebagai sebuah aktivitas/proses yang dapat dilakukan oleh sebuah objek.

2. Objek
   Objek adalah sesuatu yang mewakili kelas. Objek disini berfungsi sebagai pengganti pemanggilan sebuah kelas, maka sebuah objek hanya dapat mewakili sebuah kelas saja.
3. Magic method
   Magic method adalah metode yang diawali dan diakhiri dengan *double underscore*. Method ini tidak dipanggil secara langsung, tapi dipanggil sistem secara internal Ketika melakukan sesuatu seperti menggunakan operator, membuat objek.
4. Konstruktor
   Konstruktor adalah method yang dijalankan secara otomatis pada saat sebuah objek dibuat untuk mewakili kelas tersebut.
5. Destruktor
   Destruktor adalah fungsi yang dipanggil Ketika user menghapus objek. Fungsi ini bekerja secara otomatis, jadi tidak diperlukan pemanggilan. Tujuannya adalah melakukan final cleaning up atau bersih-bersih.
6. Setter dan Getter
   Setter dan getter digunakan untuk melakukan enkapsulasi agar tidak terjadi perubahan data secara tidak sengaja. Setter adalah method yang digunakan untuk menetapkan nilai suatu atribut khususnya atribut private dan protected. Sedangkan getter digunakan untuk mengambil nilai.
7. Decorator
   Sama halnya dengan setter dan getter untuk memodifikasi sebuah atribut bersifat private dan protected.

Contoh kode kelas, objek, dan magic method:

```
1  class Mahasiswa:
2      def __init__(self,nama,nim):
3          self.nama = nama
4          self.nim = nim
5
6  mahasiswa1 = Mahasiswa("Josua", 123456)
7  mahasiswa2 = Mahasiswa("David", 654321)
```

Contoh kode setter dan getter:

```python
class Mahasiswa:
    def __init__(self,nama,nim):
        self.nama = nama
        self.__nim = nim

    def get_nim(self): # getter
        return self.__nim

    def set_nim(self,nim_baru): # setter
        self.__nim = nim_baru

mahasiswa1 = Mahasiswa("Josua", 123456)
mahasiswa2 = Mahasiswa("David", 654321)
```

Contoh kode decorator:

```python
class Mahasiswa:
    def __init__(self,nama,nim):
        self.nama = nama
        self.__nim = nim

    @property
    def nim(self):
        return self.__nim

    @nim.setter
    def nim(self,other):
        self.__nim = other

mahasiswa1 = Mahasiswa("Josua", 123456)
mahasiswa2 = Mahasiswa("David", 654321)
```

Modul 3

1. Abstraksi

Abtraksi digunakan untuk menyembunyikan detail fungsionalitas sebuah fungsi dari pengguna. Pengguna hanya berinteraksi dengan implementasi dasar dari funggsi tersebut, tetapi detail kerja bagian dalam disembunyikan.

Manfaat penggunaan abstraksi:

- Menyembunyikan data yang tidak relevan untuk mengurangi kompleksitas
- Menghindarkan terjadinya duplikasi kode
- Membantu untuk meningkatkan keamanan dari sebuah aplikasi atau program

- Memudahkan dalam pengembangan selanjutnya, karena kode di dalam dapat di ubah secara individual tanpa menyebabkan error pada pengguna
- Umumnya abstraksi dapat di implementasikan menggunakan interface, abstract class dan acces modifier pada atribut

a. Interface
   - Semacam kontrak yang harus dipenuhi
   - Syarat-syaratnya harus di implementasikan

b. Abstract class
   - Sebuah konsep yang menerapkan sifat pewarisan dimana atribut dan fungsi pada induk bersifat abstrak (tidak dibuat objeknya) implementasi nyata dilakukan di anak kelas
   - Kelas abstrak ditandai dengan mewarisi kelas ABC (Abstract Base Class).
   - Fungsi abstrak pada kelas abstrak ditandai dengan memberikan decorator @abstractmethod pada fungsi

c. Access modifier
   - Menggunakan visibilitas fungsi untuk data yang bisa diakses di luar kelas atau yang ingin disembunyikan
   - Dapat diberikan pada atribut
   - Terdapat atribut dan metode yang bersifat private dengan menambahkan awalan garis bawah ganda (__).

Contoh kode abstraksi dengan abstract class:

```python
from abc import ABC, abstractmethod

class Browser(ABC):
    @abstractmethod
    def google(self):
        pass

class Laptop(Browser):
    def google(self):
        print("mengakses www.google.com")

laptop1 = Laptop()

laptop1.google() # output = "mengakses www.google.com"
```

Contoh kode abstraksi dengan Access modifier:

```python
class Mahasiswa:
    def __init__(self,nama,nim,alamat):
        self.__nama = nama
        self.__nim = nim
        self.__alamat = alamat

mahasiswa1 = Mahasiswa("josua", 12345, "Lampung Selatan")
```

2. **Enkapsulasi**

   Proses membungkus atribut dan metode dalam satu unit berupa kelas. Pengelompokan Bersama antara data dan fungsionalitas. Menyembunyikan kode dan data kedalam satu unit untuk melindungi data dari dunia luar. Berkomunikasi hanya lewat atribut dan metode yang disediakan oleh objek tersebut. Implementasi dapat menggunakan access modifiers:

   - Public
     Objek berjenis public baik atribut maupun metode dapat diakses dari dalam atau luar. Secara default, semua objek akan berjenis public. Bentuknya sama dengan membuat objek biasa.
   - Protected
     Objek berjenis protected baik atribut maupun metode hanya dapat diakses dari dalam kelas dan turunan kelasnya. Untuk membuat atribut atau metode objek berjenis protected dapat dilakukan dengan cara menambahkan awalan garis bawah tunggal (_) kedalam nama tersebut.
   - Private
     Objek berjenis private baik atribut maupun metode hanya dapat diakses dari dalam kelasnya saja. Setiap upaya untuk melakukannya akan menghasilkan AtributeError. Untuk membuat atribut atau metode objek berjenis private dapat dilakukan dengan cara menambahkan awalan garis bawah ganda (__) kedalam nama. Pada python atribut bersifat private masih bisa diakses dari luar kelas dengan cara: nama_object_instance._NamaKelas__nama_atribut

   Getter dan setter digunakan bertujuan untuk memastikan enkapsulasi data. Getter dan setter sering digunakan Ketika menambahkan logika validasi dari mendapatkan (get) dan menetapkan (set) suatu nilai. Untuk menghindari akses langsung yaitu variabel bersifat private, variabel tidak dapat diakses secara langsung atau dimodifikasi oleh pengguna eksternal maka digunakannya getter dan setter.

   Decorator adalah alat untuk memungkinkan pemrogram untuk mengubah perilaku fungsi atau kelas dengan cara membungkus fungsi lain untuk memperluas perilaku dari fungsi yang dibungkus. Decorator yang ada di python yaitu, @property, @classmethod, dan @staticmethod.

   Contoh kode enkapsulasi dengan Access modifier:

```python
class Mahasiswa:
    def __init__(self,nama,nim,alamat):
        self.nama = nama # public
        self._nim = nim # protected
        self.__alamat = alamat # private

mahasiswa1 = Mahasiswa("josua", 12345, "Lampung Selatan")
```

Contoh kode enkapsulasi dengan getter dan setter:

```python
class Mahasiswa:
    def __init__(self,nama,nim,alamat):
        self.nama = nama # public
        self._nim = nim # protected
        self.__alamat = alamat # private

    def get_alamat(self):
        return self.__alamat

    def set_alamat(self,alamat_baru):
        self.__alamat = alamat_baru

mahasiswa1 = Mahasiswa("josua", 12345, "Lampung Selatan")

print(mahasiswa1.get_alamat()) # Lampung Selatan
mahasiswa1.set_alamat("Kalianda")
print(mahasiswa1.get_alamat()) # Kalianda
```

Modul 4

1. Inheritance

Inheritance adalah salah satu konsep dasar dari OOP yaitu menurunkan kelas dari kelas lain untuk hirarki kelas yang saling berbagi atribut dan metode.

- Inheritance Identik
  Inheritance identic merupakan pewarisan yang menambahkan constructor pada class child sehingga class child memiliki constructornya sendiri tanpa mengholangkan constructor pada class parentnya. Metode ini ditandai dengan class child yang menggunakan kata kunci super().
- Subclass
  Subclass sama saja dengan class child yaitu kelas baru yang memperluas kelas yang sudah ada dengan memiliki atribut dan fungsi kelas yang ada dan dapat lebih banyak.
- Multiple inheritance
  Kelas dapat mewarisi dari banyak orang tua sehingga pada class child bisa ada 2 class parent. Jika terdapat banyak superclass maka pencarian dilakukan di kelas yang paling kiri dan dilanjutkan ke kelas sebelah kanan.

Contoh kode inheritance:

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

class Biodata(Mahasiswa):
    def biodata(self):
        print(f"Nama mahasiswa : {self.nama}")
        print(f"Nim mahasiswa : {self.nim}")

mahasiswa1 = Biodata("Josua", 123456)

mahasiswa1.biodata()
```

Contoh kode multiple inheritance:

```python
class Ram:
    def __init__(self,ram):
        self.ram = ram

class Procecor:
    def __init__(self,procecor):
        self.procecor = procecor

class Storage:
    def __init__(self,storage):
        self.storage = storage

class Komputer(Ram,Procecor,Storage):
    def __init__(self,ram,procecor,storage):
        Ram.__init__(self, ram)
        Procecor.__init__(self, procecor)
        Storage.__init__(self, storage)

    def informasi_komputer(self):
        print(f"Ram : {self.ram} GB")
        print(f"Procecor : {self.procecor}")
        print(f"Storage : {self.storage} GB")
```

2. Polymorphism

Polymorphism berarti poly = banyak dan morphism = bentuk. Jadi polymorphism memungkinkan digunakannya suatu interface yang sama untuk memerintah objek agar melakukanb aksi atau Tindakan yang mungkin secara prinsip sama namun secara prosesnya berbeda. Polymorphism merupakan kemampuan suatu method untuk bekerja dengan lebih dari satu tipe argument.

Polymorphism memiliki 4 cara berbeda untuk mendefinisikan polymorphism pada OOP python:

- Polymorphism dengan operator

- Polymorphism dengan fungsi
- Polymorphism dengan kelas
- Polymorphism dengan inheritance

- Override/ Overriding
  Pada konsep OOP python dapat menimpa suatu metode yang ada pada class parent dengan mendefinisikan Kembali method dengan nama yang sama pada child class. Dengan begitu maka method yang ada di class parent tidak berlaku dan akan dijalankan adalah method yang ada di class child.
- Overloading
  Metode overloading mengizinkan sebuah class untuk memiliki sekumpulan fungsi dengan nama yang sama dan argument yang berbeda. Secara umum overloading memiliki beberapa signature, yaitu jumlah argument, tipe argument, tipe keluaran dan urutan argument.

Contoh kode polymorphism:

```
class AlatMusik:
    def cara_memainkan(self):
        print("dimainkan dengan ditiup, dan dipetik")

class Gitar(AlatMusik):
    def cara_memainkan(self):
        print("dimainkan dengan dipetik")

class Saxophone(AlatMusik):
    def cara_memainkan(self):
        print("dimainkan dengan ditiup")

pemain1 = Gitar()
pemain2 = Saxophone()

pemain1.cara_memainkan()
pemain2.cara_memainkan()
```

Output:

```
dimainkan dengan dipetik
dimainkan dengan ditiup
```