Nama : Stanislaus Nicko Fasio Priyanjaga

NIM : 121140076

Kelas RB

Resume Materi PBO

Praktikum 5

Dasar Pemrograman Python

1. Sintax Dasar
   ~ Statement

   Semua perintah yang bisa dieksekusi Python disebut statement. Pada Python akhir dari sebuah statement adalah baris baru (newline) tapi dimungkinkan membuat statement yang terdiri dari beberapa baris menggunakan backslash (\).

   ~ Baris dan Identasi
2. Variable dan Tipe Data Primitif

   Variabel merupakan lokasi penyimpanan yang berguna untuk menyimpan suatu data atau suatu nilai.

```
var1 = True # Boolean
angka1 = 10 # Integer
desimal = 3.14 # Float
huruf1 = 'Praktikum Pemrograman Python' # String
```

3. Operator
   ~ Operator Aritmatika

   Operator aritmatika adalah operator yang digunakan untuk melakukan operasi matematika, seperti penjumlahan, pengurangan, perkalian, pembagian, dan sebagainya.

   ~ Operator Perbandingan

   Operator perbandingan adalah operator yang digunakan untuk membandingkan 2 buah nilai. Hasil perbandingannya adalah True atau False tergantung kondisi.

   ~ Operator Penugasan

   Operator penugasan adalah operator yang digunakan untuk memberi nilai ke variabel.

   ~ Operator Logika

   Operator logika adalah operator yang digunakan untuk melakukan operasi logika.

   ~ Operator Bitwise

   Operator bitwise adalah operator yang melakukan operasi bit terhadap operand. Operator ini beroperasi bit per bit sesuai dengan namanya.

4. Tipe Data Bentukan
    a. List
       Sebuah kumpulan data yang terurut, dapat diubah, dan memungkinkan ada anggota yang sama
    b. Tuple
       Sebuah kumpulan data yang terurut, tidak dapat diubah, dan memungkinkan ada anggota yang sama
    c. Set
       Sebuah kumpulan data yang tidak berurutan, tidak terindeks, dan tidak memungkinkan ada anggota yang sama
    d. Dictionary
       Sebuah kumpulan data yang tidak berurutan, dapat diubah, tidak memungkinkan ada anggota yang sama
5. Percabangan
    a. Percabangan IF

```
#menentukan bilangan positif dan negatif

print("Masukkan nilai N: ", end="")
N = int(input())

if(N<0):
    print(str(N) + " bilangan negatif")
```

    b. Percabangan IF-ELSE

```
#menentukan bilangan positif dan negatif

print("Masukkan nilai N: ", end="")
N = int(input())

if(N<0):
    print(str(N) + " bilangan negatif")
else:
    print(str(N) + " bilangan positif")
```

    c. Percabangan IF-ELSE-IF

```
#menentukan bilangan positif dan negatif

print("Masukkan nilai N: ", end="")
N = int(input())

if(N<0):
    print(str(N) + " bilangan negatif")
elif(N==0):
    print(str(N) + " bilangan nol")
else:
    print(str(N) + " bilangan positif")
```

6. Perulangan
    a. Perulangan For
       Pada perulangan for biasa digunakkan untuk iterasi pada urutan berupa list, tuple, atau string.

```python
#contoh perulangan for pada list
namaMahasiswa = ["Joko", "Budi", "Bambang", "Eka"]
for i in namaMahasiswa:
    print(i)
```

b. Perulangan While

Dengan menggunakkan while maka dapat dilakukan perulangan selama kondisi tertentu terpenuhi.

```python
#menghitung karakter c sebelum tanda !

count=int(0)

kalimat=(input())

while(kalimat!='!'):

    if kalimat=='c':
        count+=1
    kalimat=(input())

print("jumlah karakter c adalah ", count)
```

7. Fungsi

Dengan menggunakkan fungsi kita dapat mengeksekusi suatu blok kode tanpa harus menulisnya berulang-ulang.

- Kelas

Kelas atau class pada python bisa kita katakan sebagai sebuah blueprint (cetakan) dari objek (atau instance) yang ingin kita buat. Dengan menggunakan kelas kita dapat mendesain objek secara bebas. Kelas berisi dan mendefinisikan atribut/properti dan metode untuk objeknya nanti. Satu buah kelas dapat membuat banyak objek dan kelas sendiri tidak bisa langsung digunakan, harus diimplementasikan menjadi sebuah objek dulu, dapat disebut Instansiasi.

- Atribut/Property

Dalam suatu kelas, umumnya dideklarasikan sebuah variabel yang akan disebut sebagai sebuah atribut. Karena dideklarasikan di dalam sebuah kelas, maka setiap objek yang dibentuk dari kelas tersebut juga akan memiliki atribut yang dimiliki oleh kelas tersebut. 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.

- Method
  Selain atribut, elemen lain yang terdapat dalam suatu kelas adalah method Method adalah suatu fungsi yang terdapat di dalam kelas. Sama halnya seperti atribut, semua objek yang dibuat menggunakan kelas yang sama akan memiliki method yang sama pula. Method dapat diibaratkan sebagai sebuah aktivitas/proses yang dapat dilakukan oleh sebuah objek.

- Objek
  Objek adalah sesuatu yang "mewakili" kelas. Objek disini berfungsi sebagai pengganti pemanggilan sebuah kelas, maka sebuah objek hanya dapat mewakili sebuah kelas saja. Untuk membuat sebuah objek yang mewakili sebuah kelas, kita dapat membuatnya dengan cara memanggil nama dari kelas yang diinginkan, ditambah dengan tanda kurung ().

- Magic Method
  Magic method adalah metode yang diawali dan diakhiri dengan double underscore (dunder). Method ini tidak dipanggil secara langsung, tapi dipanggil sistem secara internal ketika melakukan sesuatu seperti menggunakan operator tambah (__add__), membuat objek (__init__), dan lain-lain.

- Konstruktor
  Konstruktor adalah method yang "pasti" dijalankan secara otomatis pada saat sebuah objek dibuat untuk mewakili kelas tersebut. Seperti dengan method lain, konstruktor dapat melakukan operasi seperti melakukan print. Selain operasi method dasar, konstruktor dapat menerima argumen yang diberikan ketika objek dibuat.

- Destruktor
  Destruktor adalah fungsi yang dipanggil ketika user menghapus objek. Fungsi ini bekerja secara otomatis, jadi tidak perlu dilakukan pemanggilan. Tujuan adanya destruktor adalah melakukan final cleaning up atau "bersih-bersih" terakhir sebelum sebuah objek benar-benar dihapus dari memori.

```
1  class Angka:
2      def __init__(self, angka):
3          self.angka = angka
4
5      def __del__(self):
6          print ("objek {self.angka} dihapus")
7
8  N=Angka(1)
9  P=Angka(2)
```

- 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 (akan diajarkan lebih jelas pada materi minggu selanjutnya), sedangkan getter digunakan untuk mengambil nilai.

```
class siswa:
    def __init__(self, umur = 0):
        self._umur = umur

    # getter method
    def get_umur(self):
        return self._umur

    # setter method
    def set_umur(self, x):
        self._umur=x

raj = siswa()

# setting the umur using setter
raj.set_umur(19)
# retrieving umur using getter
print(raj.get_umur())
print(raj._umur)
```

- Decorator

  Selain menggunakan fungsi setter dan getter tambahan seperti pada contoh sebelumnya, dalam Python kita juga dapat memanfaatkan property decorator untuk mendapatkan hasil serupa. Bedanya, melalui property decorator ini kita tidak perlu membuat fungsi lagi dengan nama yang berbeda-beda (cukup menggunakan 1 buah nama/variabel).

```
class Siswa:
    def __init__(self, nama, umur = 15):
        self.__nama = nama
        self.__umur = umur

    @property
    def umur(self):
        print("Fungsi getter umur dipanggil")
        return self.__umur

    @umur.setter
    def umur(self, x):
        print("Fungsi setter umur dipanggil")
        self.__umur = x

Bambang = Siswa("Bambang")

# Akan error karena bersifat private
#print(Bambang.__umur)

# Gunakan fungsi property decorator
print(Bambang.umur)
Bambang.umur += 5
print(Bambang.umur)
```

Sifat-sifat PBO

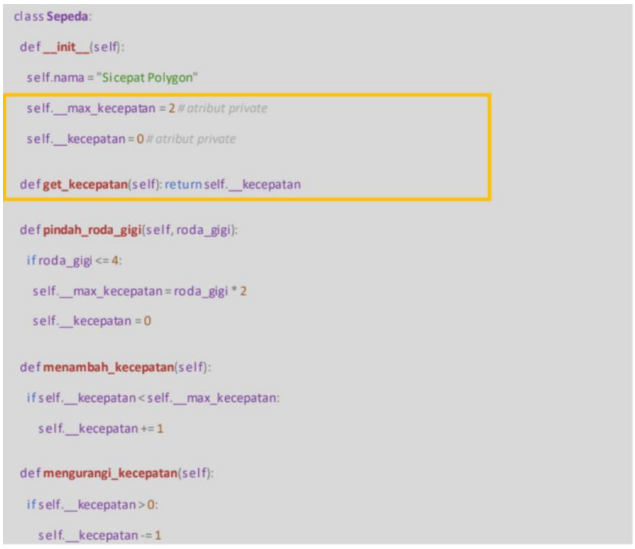
1. **Abstraksi**

   Abstraksi adalah konsep OOP dimana model yang dibuat hanya memperlihatkan atribut yang esensial dan menyembunyikan detail-detail yang tidak penting dari user. gunanya untuk mengurangi kompleksitas. User mengetahui apa yang objek lakukan, tapi tidak tau mekanisme yang terjadi di belakang layar bagaimana.

   Mengapa abstraksi itu penting?
   Saat mendesain sebuah kode program, kita dapat memisahkan interface dan implementasi sehingga menghasilkan beberapa manfaat, diantaranya:

   a. Menyembunyikan data yang tidak diperlukan untuk presentasi
   b. Menghindari terjadinya duplikasi kode
   c. Membantu untuk meningkatkan keamanan dari sebuah aplikasi atau program
   d. Memudahkan dalam pengembangan selanjutnya, karena kode di dalam dapat diubah secara individual tanpa menyebabkan error pada pengguna
   e. Umumnya abstraksi dapat diimplementasikan dengan menggunakan *interface, abstract class,* dan *acces modifier* pada atribut

   Contoh abstraksi yang menggunakan *access modifier*



```
class Sepeda:
 def __init__(self):
  self.nama = "Sicepat Polygon"
  self.__max_kecepatan = 2 # atribut private
  self.__kecepatan = 0 # atribut private

 def get_kecepatan(self): return self.__kecepatan

 def pindah_roda_gigi(self, roda_gigi):
  if roda_gigi <= 4:
   self.__max_kecepatan = roda_gigi * 2
   self.__kecepatan = 0

 def menambah_kecepatan(self):
  if self.__kecepatan < self.__max_kecepatan:
   self.__kecepatan += 1

 def mengurangi_kecepatan(self):
  if self.__kecepatan > 0:
   self.__kecepatan -= 1
```

2. **Enkapsulasi**

   Enkapsulasi dalam pemrograman Python adalah konsep pemrograman berorientasi objek yang mengizinkan Anda untuk membatasi akses ke data atau metode dalam kelas, sehingga hanya anggota tertentu yang dapat mengaksesnya. Di Python, enkapsulasi dapat

dilakukan dengan menggunakan konvensi penamaan atribut dan metode, yaitu dengan menambahkan underscore sebelum nama variabel atau metode yang ingin dibuat privat. Ini tidak secara teknis mencegah akses ke atribut atau metode, tetapi memberikan sinyal kepada pengguna kode bahwa variabel atau metode tersebut seharusnya tidak diakses secara langsung dari luar kelas.

Sebagai contoh, Anda dapat membuat kelas berikut dengan dua atribut dan satu metode yang dienkapsulasi:

```
class Mobil:
    def __init__(self, merk, warna, harga):
        self._merk = merk
        self._warna = warna
        self._harga = harga

    def _harga_dalam_rupiah(self):
        return "Rp. " + str(self._harga)

mobil1 = Mobil("Toyota", "Merah", 300000000)
print(mobil1._merk) # akan menghasilkan "Toyota"
print(mobil1._harga_dalam_rupiah()) # akan menghasilkan "Rp. 300000000"
```

Perhatikan bahwa atribut dan metode yang dienkapsulasi diberi underscore di depan namanya, sehingga memberi sinyal kepada pengguna kode bahwa mereka seharusnya tidak diakses secara langsung dari luar kelas. Namun, pada kenyataannya, mereka masih dapat diakses seperti contoh di atas.

3. Inheritance

Inheritance atau pewarisan pada program Python adalah konsep pemrograman berorientasi objek yang memungkinkan Anda untuk membuat kelas baru yang mendapatkan atribut dan metode dari kelas yang sudah ada. Dengan inheritance, Anda dapat menulis kode yang lebih efisien dan mudah dipelihara.

Untuk membuat kelas yang mewarisi atribut dan metode dari kelas lain, Anda dapat menggunakan sintaksis berikut:

```
class KelasInduk:
    def metode_induk(self):
        print("Ini adalah metode dari kelas induk")

class KelasAnak(KelasInduk):
    def metode_anak(self):
        print("Ini adalah metode dari kelas anak")

    ukuran = "besar"

objek_anak = KelasAnak()
objek_anak.metode_induk() # akan menghasilkan "Ini adalah metode dari kelas induk"
objek_anak.metode_anak() # akan menghasilkan "Ini adalah metode dari kelas anak"
print(objek_anak.ukuran) # akan menghasilkan "besar"
```

Pada contoh diatas**,** ***KelasAnak*** mewarisi atribut dan metode dari ***KelasInduk***. Dalam contoh diatas juga ditambahkan ***metode_anak*** dan atribut ***ukuran*** ke ***KelasAnak***. Metode dan atribut ini sekarang tersedia dikelas anak, namun metode dan atribut dari induk tetap dapat digunakan.

4. **Polymorphism**

Polymorphism dalam pemrograman python adalah konsep pemrograman berorientasi objek yang memungkinkan suatu objek untuk memiliki banyak bentuk/tipe, tergantung pada konteks penggunaannya. Dalam hal ini, metode yang sama dapat digunakan untuk objek yang berbeda-beda dengan cara yang berbeda-beda. Polymorphism dapat diimplementasikan dengan metode yang sama pada beberapa kelas yang berbeda. Setiap kelas akan memiliki implementasi sendiri dari metode tersebut.

Contoh :

```
class Kucing:
    def bersuara(self):
        print("Meow")

class Anjing:
    def bersuara(self):
        print("Guk")

class Sapi:
    def bersuara(self):
        print("Moo")

hewan = [Kucing(), Anjing(), Sapi()]

for h in hewan:
    h.bersuara()
```

Dalam contoh diatas, kita memiliki tiga kelas: ***Kucing, Anjing, Sapi***. Ketiga kelas ini memiliki metode ***bersuara***. Buat sebuah list ***hewan*** yang berisi satu objek dari setiap kelas. Kemudian, kita melakukan iterasi pada list ***hewan*** dan memanggil metode ***bersuara*** untuk setiap objek. Karena setiap objek memanggil metode yang sama, kita menggunakan polymorphism untuk memanggil metode ***bersuara*** pada setiap objek.

Contoh implementasi program pada abstraksi

```python
from abc import ABC, abstractmethod

class Bentuk(ABC):
    @abstractmethod
    def hitung_luas(self):
        pass

class PersegiPanjang(Bentuk):
    def __init__(self, panjang, lebar):
        self.panjang = panjang
        self.lebar = lebar

    def hitung_luas(self):
        return self.panjang * self.lebar

class Lingkaran(Bentuk):
    def __init__(self, radius):
        self.radius = radius

    def hitung_luas(self):
        return 3.14 * (self.radius ** 2)

persegi_panjang = PersegiPanjang(4, 5)
lingkaran = Lingkaran(3)

print("Luas Persegi Panjang: ", persegi_panjang.hitung_luas())
print("Luas Lingkaran: ", lingkaran.hitung_luas())
```

Pada contoh ini, buat kelas abstrak ***Bentuk*** dengan menggunakan modul ***abc***. Mendefinisikan metode ***hitung_luas*** sebagai metode abstrak dengan menggunakan decorator ***@abstractmethod***. Definisikan dua kelas mewarisi dari ***Bentuk*** : ***PersegiPanjang*** dan ***Lingkaran***. Setiap kelas mengimplementasikan metode ***hitung_luas*** sesuai dengan ciri khasnya. Buat objek ***persegi_panjang*** dari kelas ***PersegiPanjang*** dan objek ***lingkaran*** dari kelas ***Lingkaran***. Kemudian panggil metode ***hitung_luas*** pada masing_masing objek dan cetak hasilnya.

Contoh implementasi program pada enkapsulasi

```python
class Car:
    def __init__(self, make, model, year):
        self.__make = make
        self.__model = model
        self.__year = year

    # Menggunakan property untuk membaca nilai atribut
    @property
    def make(self):
        return self.__make

    @property
    def model(self):
        return self.__model

    @property
    def year(self):
        return self.__year

    # Menggunakan setter untuk mengubah nilai atribut
    @make.setter
    def make(self, make):
        self.__make = make

    @model.setter
    def model(self, model):
        self.__model = model

    @year.setter
    def year(self, year):
        self.__year = year

car = Car('Toyota', 'Corolla', 2022)

# Mengakses nilai atribut menggunakan property
print(car.make)
print(car.model)
print(car.year)

# Mengubah nilai atribut menggunakan setter
car.make = 'Honda'
car.model = 'Civic'
car.year = 2023

# Mengakses nilai atribut yang sudah diubah
print(car.make)
print(car.model)
print(car.year)
```

Contoh implementasi program pada inheritance

```python
# Kelas induk/parent
class Hewan:
    def __init__(self, nama, jenis):
        self.nama = nama
        self.jenis = jenis

    def bersuara(self):
        print("")

# Kelas turunan/child dari Hewan
class Kucing(Hewan):
    def __init__(self, nama, jenis, warna):
        super().__init__(nama, jenis)
        self.warna = warna

    def bersuara(self):
        print("Meow")

# Kelas turunan/child dari Hewan
class Anjing(Hewan):
    def __init__(self, nama, jenis, ras):
        super().__init__(nama, jenis)
        self.ras = ras
    def bersuara(self):
        print("Guk Guk")

# Membuat objek dari kelas Kucing
kucing1 = Kucing("Tom", "Kucing", "Oren")

# Membuat objek dari kelas Anjing
anjing1 = Anjing("Spike", "Anjing", "Bulldog")

# Memanggil method pada objek
kucing1.bersuara()
anjing1.bersuara()
```

Contoh implementasi program pada polymorphism

```python
class Mobil:
    def __init__(self, merek, model):
        self.merek = merek
        self.model = model

    def info(self):
        print("Merek:", self.merek)
        print("Model:", self.model)

    def kendarai(self):
        print("Mobil dikendarai.")

class Sedan(Mobil):
    def kendarai(self):
        print("Sedan dikendarai dengan nyaman.")

class SUV(Mobil):
    def kendarai(self):
        print("SUV dikendarai di medan yang berat.")

class Hatchback(Mobil):
    def kendarai(self):
        print("Hatchback dikendarai dengan lincah.")
```

```python
# Objek dari masing-masing kelas
mobil1 = Mobil("Toyota", "Camry")
sedan1 = Sedan("Honda", "City")
suv1 = SUV("Ford", "Explorer")
hatchback1 = Hatchback("Volkswagen", "Golf")

# Memanggil method info() dari objek-objek mobil
mobil1.info()
sedan1.info()
suv1.info()
hatchback1.info()

# Memanggil method kendarai() dari objek-objek mobil
mobil1.kendarai()
sedan1.kendarai()
suv1.kendarai()
hatchback1.kendarai()
```