Nama : Tobyanto P. M. Manurung
NIM : 121140099
Kelas : RB

Resume Praktikum PBO Pertemuan 5 (Modul 1 - Modul 4)

Bahasa Pemrograman Python

Python merupakan salah satu bahasa pemrograman tingkat tinggi yang diciptakan oleh seseorang programmer asal Belanda yaitu Guido van Rossum pada akhir tahun 1980. Saat ini, python sering digunakan oleh berbagai macam kalangan karena sintaksnya yang mudah dibaca dan dipahami, bahkan python sering menjadi bahan pembelajaran bagi pemula yang ingin memasuki dunia pemrograman secara umum. Untuk bahasanya sendiri, python merupakan bahasa pemrograman berbasis objek dimana program yang dibuat oleh bahasa ini berpacu pada objek-objek atau data-data yang terdapat pada program tersebut. Belakangan ini, kita dapat melihat pengaplikasian python mulai dari pengembangan perangkat lunak desktop, perangkat lunak berbasis mobile, web-app, otomatisasi, *Internet of Things*, dan robotika.

Sintaks Dasar

Berikut merupakan sintaks dasar yang mesti dipelajari oleh setiap orang yang ingin memahami cara kerja python.

- Statement :
  Pada python, semua perintah yang dapat dieksekusi pada saat dirun merupakan statement. Selain itu, suatu statement baru itu dimulai pada saat programmer berpindah ke baris baru pada kode program yang ingin dibuat. Namun, backslash (\) memungkinkan programmer untuk membuat statement pada baris yang sama.
- Baris dan Indentasi :
  Beda dengan bahasa pemrograman C++ yang menggunakan kurung kurawal sebagai metode untuk melakukan *grouping* pada suatu blok kode, Python mempunyai metodenya sendiri yaitu dengan menggunakan 4 spasi atau yang sering disebut dengan indent.

Variabel dan Tipe Data Primitif

Variabel merupakan tempat penyimpanan yang dapat digunakan oleh pemrogram untuk menyimpan suatu data atau suatu nilai. Pada python sendiri, terdapat 4 tipe data yaitu :

- Bool / Boolean yang memiliki nilai True atau False.
- Int / Bilangan bulat yang memiliki nilai dari seluruh bilangan bulat

- Float / Bilangan real yang memiliki nilai dari seluruh bilangan real
- String / Teks yang memiliki nilai dari seluruh kumpulan karakter

Operator

Dalam python, ada 7 operator yang digunakan yaitu:

- Operator Aritmatika
  Operator yang digunakan untuk melakukan operasi matematika pada umumnya

Operator	Nama dan Fungsi	Contoh
+	Penjumlahan, menjumlahkan 2 buah operand	X + Y
-	Pengurangan, mengurangkan 2 buah operand	X - Y
*	Perkalian, mengalikan 2 buah operand	X * Y
/	Pembagian, membagi 2 buah operand	X / Y
**	Pemangkatan, memangkatkan bilangan	X ** Y
//	Pembagian bulat, menghasilkan hasil bagi tanpa koma	X // Y
%	Modulus, menghasilkan sisa pembagian 2 bilangan	X % Y

- Operator Perbandingan
  Operator yang digunakan untuk membandingkan dua buah nilai atau variabel.

Operator	Nama dan Fungsi	Contoh
>	Lebih besar dari - akan mengembalikan nilai True apabila nilai pada bagian kiri lebih besar dari pada nilai bagian kanan.	X > Y
<	Lebih kecil dari - akan mengembalikan nilai True apabila nilai pada bagian kiri lebih kecil dari pada nilai bagian kanan.	X < Y
==	Sama dengan - akan mengembalikan nilai True apabila kedua nilai memiliki nilai yang sama.	X == Y
!=	Tidak sama dengan - akan mengembalikan nilai True apabila kedua nilai tidak memiliki nilai yang sama.	X != Y

>=	Lebih besar atau sama dengan - akan mengembalikan nilai True apabila nilai pada bagian kiri lebih besar atau sama besar dengan nilai bagian kanan.	X >= Y
<=	Lebih kecil atau sama dengan - akan mengembalikan nilai True apabila nilai pada bagian kiri lebih kecil atau sama besar dengan nilai bagian kanan.	X <= Y

- Operator Penugasan
  Operator yang digunakan untuk memberi nilai ke suatu variabel.

Operator	Penjelasan	Contoh
=	Menugaskan nilai yang ada di kanan ke operand di sebelah kiri.	C = A + B
+=	Menambahkan operand yang di kanan dengan operand yang ada di kiri dan hasilnya ditugaskan ke operand yang di kiri.	C += A
-=	Mengurangi operand yang di kanan dengan operand yang ada di kiri dan hasilnya ditugaskan ke operand yang di kiri.	C -= A
*=	Mengalikan operand yang di kanan dengan operand yang ada di kiri dan hasilnya ditugaskan ke operand yang di kiri.	C *= B
/=	Membagi operand yang di kanan dengan operand yang ada di kiri dan hasilnya ditugaskan ke operand yang di kiri	C /= B
**=	Memangkatkan operand yang di kanan dengan operand yang ada di kiri dan hasilnya ditugaskan ke operand yang di kiri	C **= B
//=	Melakukan pembagian bulat operand di kanan terhadap operand di kiri dan hasilnya disimpan di operand yang di kiri.	C //= B
%=	Melakukan operasi sisa bagi operand kanan	C %= A

	dengan operand di kiri dan hasilnya disimpan di operand yang di kiri.	

- Operator Logika
  Operator yang digunakan untuk melakukan operasi logika. Operator pertama adalah **and** yang akan mengembalikan nilai True apabila kedua operandnya bernilai benar (X and Y). Operator kedua adalah **or** yang akan mengembalikan nilai True apabila satu atau kedua operandnya bernilai benar (x or y). Operator terakhir pada bagian ini adalah **not** yang akan mengembalikan nilai True apabila operandnya bernilai salah (kebalikan nilai).
- Operator Bitwise
  Operator yang digunakan untuk melakukan operasi bit terhadap suatu operand.

Operator	Nama	Contoh
&	Bitwise AND	X & Y
|	Bitwise OR	X | Y
~	Bitwise NOT	~X
^	Bitwise XOR	X ^ Y
>>	Bitwise right shift	X >> 2
<<	Bitwise left shift	X << 2

- Operator Identitas
  Operator yang digunakan untuk memeriksa apabila dua buah nilai (variabel) berada pada lokasi memori yang sama.

Operator	Penjelasan	Contoh
is	True apabila kedua operand identik.	X is True
is not	True jika kedua operand tidak identik.	X is not True

- Operator Keanggotaan
  Operator yang digunakan untuk memeriksa apabila suatu nilai atau variabel merupakan anggota atau ditemukan di dalam suatu data.

Operator	Penjelasan	Contoh
in	True apabila nilai/variabel tersebut dapat ditemukan	5 in X

	dalam suatu data.	
not in	Tue apabila nilai/variabel tidak ada di dalam suatu data.	5 not in X

Tipe Data Bentukan

- List
  Sebuah kumpulan data yang terurut, dapat diubah (*mutable*), dan memperbolehkan anggota yang sama didalamnya. (contoh : {a,b,c,d, 1, ‘yes’})
- Tuple
  Sebuah kumpulan data yang terurut, tidak dapat diubah (*immutable*), dan memperbolehkan anggota yang sama didalamnya. (contoh : (a,b,c,d, ‘yes’, 1))
- Set
  Sebuah kumpulan data yang tidak berurutan, tidak terindeks, dan tidak memperbolehkan anggota yang sama didalamnya. (contoh : {a, b, c, 1, ‘yes’})
- Dictionary
  Sebuah kumpulan data yang tidak berurutan, dapat diubah, tidak memperbolehkan anggota yang sama di dalamnya, dan disusun dengan struktur *key and value*. (contoh : { ‘fName’ : ‘Damian’, ‘lName’ : ‘Rapper’})

Percabangan

Sama seperti konsep-konsep percabangan yang kita temukan di bahasa pemrograman pada umumnya (c++), python juga memiliki percabagang sebagai berikut:

Tipe Percabangan	Penjelasan
Percabangan if	Percabangan yang akan memeriksa apabila kondisi dalam percabangan if tersebut benar atau salah. Apabila benar, maka sub-program yang terdapat didalam percabangan if tersebut akan dieksekusi.
Percabangan else	Percabangan yang akan diperiksa apabila terdapat percabangan if (yang ditemukan sebelum percabangan else) tidak memenuhi kondisi. Oleh karena itu, percabangan else akan dieksekusi.
Percabangan elif	Percabangan yang digunakan untuk memeriksa kondisi-kondisi baru apabila kondisi terhadap percabangan if utama tidak memenuhi kondisi.
Percabangan nested if	Percabangan if yang terdapat pada percabagan if lain.

Contoh kode percabangan :

```python
x = 5

# percabangan if
if x > 0:
    print("x is positive")

# percabangan if else
if x % 2 == 0:
    print("x is even")
else:
    print("x is odd")

# percabangan yang menggunakan if, elif, dan else
if x > 10:
    print("x is greater than 10")
elif x > 5:
    print("x is between 6 and 10")
else:
    print("x is less than or equal to 5")

# percabangan nested if
if x > 0:
    if x < 10:
        print("x is a single digit number")
    else:
        print("x is a double digit number")
else:
    print("x is a negative number")
```

Perulangan

Perulangan pada python datang dalam dua bentuk yaitu perulangan for dan perulangan while.

Tipe perulangan	Penjelasan	Contoh

Perulangan for	Perulangan ini digunakan untuk melakukan iterasi pada urutan berupa list, tuple, ataupun string.	`x = 5`
		`for i in (len(x)):`
		`  #perintah`

Perulangan while	Perulangan ini digunakan untuk melakukan iterasi dengan ketentuan apabila kondisi dalam while tersebut terpenuhi. Perulangan akan berakhir apabila kondisi tersebut sudah tidak terpenuhi.	`x = 5`
		`while(x<100):`
		`  #perintah`
		`  x+=1`

Metode (Fungsi)

Pada python, programmer sering menggunakan metode untuk melakukan suatu aksi (sub-program) yang sering digunakan berkali-kali.

```
def sum_angka(x, y):
    return x + y
result = sum_angka(5, 10)
```

Enkapsulasi

Sama seperti abstraksi, enkapsulasi juga merupakan konsep yang penting dalam pemrograman berbasis objek (bahasa pemrograman python). Pada intinya, Enkapsulasi membantu kita untuk menyembunyikan sebagian kode/data yang tidak butuh untuk ditujukan ke pengguna (dengan alasan pengamanan dan penampilan program) dan juga menampilkan kode/data yang butuh untuk ditampilkan kepada pengguna. Untuk implementasinya sendiri, enkapsulasi dilakukan dengan suatu proses dimana kita menciptakan suatu kelas yang isinya merupakan atribut dan metode. Beda dengan abstraksi yang mengoperasikan masalah-masalah di tingkat desain, enkapsulasi membantu kita dalam mengoperasikan masalah-masalah di tingkat implementasi.

Untuk mengimplementasikan konsep enkapsulasi, kita cukup menempatkan variabel dan metode dalam suatu kelas.

Contoh kode enkapsulasi:

```
class Mobil:
      def __init__(self, gears_ratio, machine):
            self.nama = "Mitsubishi Evo Lancer 2020"
            self._gears_ratio = gears_ratio #protected variables (_<variableName>)
            self.__machine = machine #private variables (__<variableName>)

      def __str__(self):
       return "Mobil ini merupakan mobil pabrik, bukan mobil second"
```

Selanjutnya, terdapat Getters dan Setters yang membantu programmer dalam menentukan data-data yang terenkapsulasi. Getter merupakan metode dalam suatu kelas yang dapat menambahkan validasi suatu nilai, sedangkan setter dapat menetapkan suatu nilai.

Contoh kode dari Getters dan Setters:

```
def contoh_setter(self):
      return self.contoh_variabel = variabel

    def contoh_getter(self):
      return self.contoh_variabel
```

Abstraksi

Abstraksi merupakan konsep yang sering digunakan dalam pemrograman berbasis objek (terkhususnya untuk python). Pada dasarnya, abstraksi merupakan suatu teknik untuk mengekstrak atau mengambil data penting tentang suatu item/objek. Selain itu, abstraksi juga dapat membantu kita dalam menyembunyikan data yang tidak perlu ditunjukan kepada pengguna/pelanggan sewaktu mereka menggunakan perangkat lunak yang kita kembangkan. Kita juga dapat menampilkan data yang relevan dan juga mengoperasikan masalah-masalah di tingkat desain (tahap planning untuk suatu perangkat lunak).

Untuk membuat suatu atribut atau metode kelas bersifat private dengan menempatkan dua underscore sebelum atribut atau metode tersebut (contoh : __gears_ratio). Jika ingin membuat atribut atau metode dengan sifat protected, maka tempatkan satu underscore sebelum metode dan atribut (contoh : _gears_ratio). Apa perbedaan antara protected dan private? Objek-objek yang bersifat protected hanya bisa digunakan dalam kelas tersebut dan turunannya, sedangkan objek-objek yang bersifat private hanya bisa digunakan dalam sub-kelas atau kelas tersebut sendiri.

Contoh kode enkapsulasi :

```
class Mobil:
      def __init__(self, gears_ratio, machine):
            self.nama = "Mitsubishi Evo Lancer 2020"
            self._gears_ratio = gears_ratio #protected variables (_<variableName>)
            self.__machine = machine #private variables (__<variableName>)
```

Inheritance

Secara garis besar, inheritance merupakan teknik yang digunakan untuk menurunkan ciri-ciri dari suatu kelas kepada kelas lain (dimana relasinya berupa kelas *parent* dan kelas *child*). Dengan ini kita dapat menurunkan metode-metode (fungsi) dan atribut suatu kelas

kepada kelas-kelas lain yang memiliki ciri khas sama. Di sisi lain, kelas anak (*child*) juga dapat memiliki metode ataupun atributnya sendiri, berbeda dengan kelas *parent*nya.

Contoh kode yang menggunakan konsep inheritance :

```
class Komputer:
    def __init__(self,nama,jenis,harga,merk) -> None:
        self.nama = nama
        self.jenis = jenis
        self.harga = harga
        self.merk = merk

class Processor(Komputer):
    def __init__(self, nama, jenis, harga, merk,jumlah_core,kecepatan_processor)
        super().__init__(nama, jenis, harga, merk)
        self.nama = "Processor"
        self.jumlah_core = jumlah_core
        self.kecepatan_processor = kecepatan_processor
    def __str__(self):
        return f"{self.nama} ini merupakan tipe {self.jenis} yang dibuat oleh
{self.merk}"

class RAM(Komputer):
    def __init__(self, nama, jenis, harga, merk,ram_capacity) -> None:
        super().__init__(nama, jenis, harga, merk)
        self.nama = "RAM"
        self.ram_capacity = ram_capacity
    def __str__(self):
        return f"{self.nama} ini memiliki kapasitas {self.jenis} yang dibuat oleh
{self.merk}"
```

Pada contoh program yang dapat dilihat di atas, kita dapat melihat bahwa kelas Processor dan Kelas RAM merupakan kelas turunan dari kelas Komputer. Melalui inheritance, metode dan atribut yang terdapat pada kelas Komputer dapat diturunkan ke kelas-kelas lain (yang kita akan sebut sebagai kelas child/ kelas turunan). Selain itu, kita juga dapat mengoptimasi kode dari program kita dengan menghilangkan bagian-bagian yang *redundant* layaknya seperti membuat fungsi untuk melakukan suatu aksi namun untuk objek.

Polymorphism

Polymorphism merupakan kemampuan dimana sekumpulan objek dalam suatu program dapat melakukan hal yang berbeda walaupun memiliki nama yang sama. Layaknya suatu pensil yang dapat digunakan untuk menulis kabar baik, namun pada situasi yang berbeda pensil tersebut juga dapat digunakan untuk menyebarkan berita palsu atau negatif. Dalam kata lain, dengan *polymorphism,* kita dapat memodifikasi cara perilaku dari suatu objek dalam program kita.

Contoh kode *polymorphism* :

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

    def speak(self):
        pass

class Dog(Animal):
    def speak(self):
        return "Woof"

class Cat(Animal):
    def speak(self):
        return "Meow"

class Cow(Animal):
    def speak(self):
        return "Moo"

def animal_speak(animal):
    print(animal.speak())

dog = Dog("Buddy")
cat = Cat("Whiskers")
cow = Cow("Betsy")

animal_speak(dog)
animal_speak(cat)
animal_speak(cow)
```

Pada contoh program di atas, kita dapat melihat bahwa kelas Dog, Cat, dan Cow merupakan kelas turunan dari kelas Animal yang memiliki metode “speak”. Perhatikan bahwa tiap kelas turunan memiliki output yang berbeda untuk metode “speak” masing-masing. Cat sewaktu memanggil metode “speak’ akan memberi keluaran “Meow”, Cow akan memberikan keluaran “Moo”, dan Dog akan memberi keluaran “Woof”. Pada kasus ini, kita dapat melihat bahwa metode “speak” merupakan contoh dari konsep *polymorphism* dalam pemrograman berbasis objek.