Pemrograman Berorientasi Obyek

Indah Agustien Siradjuddin

Tipe Data *Class*

Python merupakan bahasa pemrograman yang berbasis Object. Object adalah suatu variabel dengan tipe data *class* yang tidak hanya berisi **nilai atau *state* atau *property***, tetapi object juga memiliki **method atau fungsi-fungsi** yang melekat pada object tersebut dan dapat juga merubah nilai object tersebut.

Berikut beberapa hal yang harus diketahui untuk Pengenalan Pemrograman Berorientasi Obyek (PBO):

String dan List

Contoh tipe data *class* yang sudah dikenal, dan digunakan sebelumnya, adalah tipe data *lists* ataupun *string*.
Suatu variabel yang berbentuk *lists* ataupun *string*, memiliki dua buah elemen yang terkandung di dalam variabel tersebut, yaitu nilai atau yang disebut dengan *state*/*property*, serta *method* atau fungsi, yang dapat digunakan untuk mengolah nilai pada variabel tersebut.
Method atau fungsi yang melekat pada suatu variabel dapat diakses dengan menggunakan syntax

```
namaObyek.NamaMethod()
```

Code

Berikut adalah contoh tipe data string dan lists, yang memiliki nilai sekaligus method

In [0]:

```
# String

dataStr='Struktur Data'
print(dataStr)
dataStr=dataStr.upper()
print(dataStr)
```

Pada contoh code tersebut, terdapat variabel $dataStr$ yang berjenis *string*, dimana property dari $dataStr$ adalah 'Struktur Data'. Contoh method yang digunakan adalah method $upper()$, yang berfungsi untuk merubah semua karakter pada $dataStr$ menjadi huruf kapital. Beberapa contoh method lain yang dapat digunakan adalah $capitalize()$, $lower()$, $find()$, dll.

In [0]:
```
# Lists

dataLs=[4,10,21]
print(dataLs)
data.append(45)
print(dataLs)
```

Pada contoh code diatas, terdapat variabel $dataLs$ yang berbentuk *lists*. Variabel ini memiliki tiga buah nilai, yaitu 4, 10, dan 21. Contoh method yang digunakan adalah $append()$, yaitu menambah data pada variabel berbentuk *lists*. Contoh method method lain yang dimiliki tipe data lists yang bisa dimanfaatkan, misalkan $pop()$, $insert()$, $clear()$, $reverse()$, dsb.

Kembali ke Menu Awal

Istilah dalam Pemrograman Berorientasi Obyek

Kelebihan dari bahasa pemrogaraman berbasis object, adalah bahasa tersebut menyediakan beberapa fitur agar *programmer* dapat membuat *class* sendiri yang sesuai dengan kebutuhan. Beberapa istilah dasar yang terdapat pada pemrograman berbasis object ini antara lain :

- *class*, tipe data yang tidak hanya berisi data tetapi juga method
- *object*, suatu variabel dengan tipe data *class*
- *state, property*, bagian data dari suatu class yang berisi nilai, dapat berupa string, integer, float
- *method*, fungsi yang melekat pada class
- *constructor*, fungsi yang dijalankan oleh python ketika pertama kali suatu objek dibuat. Method constructor ini dibuat dengan cara mendefinisikan fungsi _ *init* _ (dua *underscore*)
- *self*, merupakan suatu argumen atau parameter spesial agar nilai balik dari method dikembalikan ke objek itu sendiri. Argumen ini tidak perlu dipanggil (walaupun sudah didefinisikan)pada saat pemanggilan method
- *instance*, suatu objek yang telah dibuat
- *override*, mendefinisikan kembali fungsi-fungsi umum yang sudah ada, agar sesuai dengan kebutuhan *programmer*

Kembali ke Menu Awal

Constructor

Penjelasan masing-masing fitur penting pada PBO, dilakukan langsung dengan pemberian contoh. Contoh class yang akan dibuat adalah bilangan kompleks.
Seperti yang telah diketahui, bilangan kompleks terdiri dari bilangan real dan imajiner sebagai berikut $a = x + yi$, dimana a merupakan bilangan kompleks yang tersusun atas bilangan real x,

dan bilangan imajiner y, dan $i = \sqrt{-1}$. Karena bilangan kompleks terdiri dari dua nilai yaitu real dan imajiner, maka bilangan kompleks ini memiliki operasi yang berbeda dibandingkan dengan bilangan lain, misalkan operasi penjumlahan, perkalian, pembagian, pengurangan, dll. Oleh karena itu, sangat tepat jika bilangan kompleks ini direpresentasikan dalam bentuk tipe data *class*.

Syntax untuk membuat suatu class, adalah sebagai berikut :

```
class namaClass:
    def __init__() : #constructor
    def namaMethod () : #Method
    ...
```

Sedangkan untuk membuat suatu obyek dengan tipe data class tertentu (proses pembuatan *instance*) adalah :

```
namaObyek=namaClass()
```

Constructor merupakan method yang otomatis dijalankan ketika suatu obyek dibuat. Syntax untuk pembuatan *constructor* ini adalah :

```
def __init__()
```

Code

Berikut adalah contoh pembuatan tipe data ini dimulai dari mendefinisikan *class* dan membuat constructor dari class tersebut. Class yang akan dibuat adalah class bilangan kompleks, dimana setiap bilangan kompleks memiliki dua buah nilai, yaitu bagian real, dan bagian imajiner.

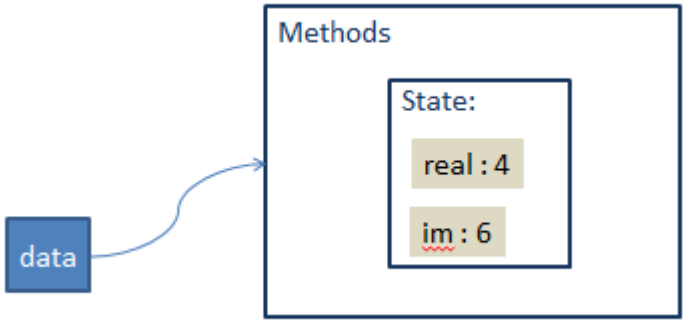
In [0]:
```
class BilanganKompleks:
    def __init__(self,a,b):
        self.real=a
        self.im=b
```

In [0]:
```
data=BilanganKompleks(4,6)
```

In [0]:
```
# untuk pengecekan tipe data dari variabel num
type(data)
```

Dari syntax-syntax tersebut, maka terdapat class dengan nama $BilanganKompleks$ sehingga terdapat tipe data baru yaitu *BilanganKompleks*.
Ketika suatu variabel dibuat, maka variabel tersebut akan memiliki dua buah nilai yang disimpan dalam variabel didalam class yaitu *real* dan *im*, yang merupakan representasi dari bagian real dan imajiner dari suatu bilangan kompleks.

Constructor ini tidak perlu dipanggil secara eksplisit, akan tetapi otomatis akan dijalankan ketika *instance* (atau sebuah variabel dengan tipe data class) dibuat. Pada code diatas, perintah `data=BilanganKompleks(4,6)` merupakan contoh instansiasi dari class bilangan kompleks, sehingga terbentuk obyek dengan nama $data$, yang memiliki nilai $real = 4$, dan nilai $im = 6$, yaitu $data = 4 + 6i$.
Argumen atau parameter *self* tidak perlu dipanggil pada saat pemanggilan method atau constructor. Ilustrasi obyek *data* dapat dilihat pada Gambar 1 berikut.



Gambar 1. Instansiasi obyek data

Pada contoh code sebelumnya, dilakukan syntax `type(data)` yaitu untuk mengetahui tipe data dari variabel data.
Misalkan isi dari data tersebut ingin ditampilkan, maka tidak bisa dilakukan dengan cara menulis syntax *print* seperti biasa, Jika tetap perintahkan print untuk menampilkan data tersebut, maka yang ditampilkan di layar hanyalah alamat tempat data tersebut disimpan, seperti contoh code berikut.

Code

In [0]:
```
class BilanganKompleks:
    def __init__(self,a,b):
        self.real=a
        self.im=b

data=BilanganKompleks(4,6)
print(data)
```

Oleh karena itu, untuk menampilkan property atau nilai yang dimiliki oleh suatu obyek, tidak dapat dilakukan dengan syntax `print()` seperti biasa.

Terdapat dua cara agar objek yang sudah dibuat dapat ditampilkan di layar sesuai dengan yang diinginkan, yaitu:

- Membuat method baru untuk menampilkan data
- *override* method standar dari sebuah class

[Kembali ke Menu Awal]

Method

Method merupakan fungsi-fungsi yang terdapat di dalam suatu class. Contoh method yang akan dibuat adalah method yang berfungsi untuk menampilkan *property* atau *state* yang terdapat di dalam suatu obyek.
Syntax pembuatan method adalah :

```
def namaMethod():
```

Sedangkan untuk memanggil method suatu class adalah :

```
namaObyek.namaMethod()
```

Code

Berikut adalah contoh pembuatan method `display()` yang berfungsi untuk menampilkan *property* yang terdapat pada class Bilangan Kompleks.

In [0]:
```
class BilanganKompleks:
    def __init__(self,a,b): # constructor
        self.real=a
        self.im=b
    def display(self): # method untuk menampilkan state
        print (self.real,'+',self.im,'i')
```

In [0]:
```
data1=BilanganKompleks(4,6)
data1.display()
data2=BilanganKompleks(2,3)
data2.display()
```

Pada contoh code diatas, terdapat method tambahan pada class $BilanganKompleks$, yaitu method `display()` , yang berfungsi untuk menampilkan *state* atau *property* obyek dari class $BilanganKompleks$.
Di dalam method tersebut, digunakan syntax print yang berfungsi untuk menampilkan suatu data. Perintah `self.real` adalah perintah untuk mengakses *state* atau *property* $real$, yang dimiliki oleh class $BilanganKompleks$, begitu juga dengan perintah `self.im` , digunakan untuk mengakses *property* im. Karena property $real$ dan im bertipe data integer, maka syntax `print()` dapat dilakukan pada dua *property* tersebut

[Kembali ke Menu Awal]

Override Method

Override Method merupakan penambahan fungsi-fungsi pada syntax-syntax yang sudah ada. Seperti yang dijelaskan sebelumnya, untuk menampilkan *property* yang terdapat pada suatu obyek, tidak dapat dilakukan dengan menggunakan perintah `print()` .

Cara pertama yang sudah dijelaskan adalah membuat method baru di dalam class, yang berfungsi untuk menampilkan *property* suatu class.
Cara kedua adalah dengan cara override method fungsi yang sudah tersedia oleh bahasa pemrograman. Syntax yang akan di*override* adalah syntax `print()` .
Yang perlu dilakukan agar suatu object dapat dipanggil dengan perintah *print* adalah object tersebut harus dirubah menjadi string. Untuk merubah suatu variabel menjadi string, maka suatu class dilengkapi dengan method `__str__` . Hanya saja method ini tidak bisa berfungsi secara sempurna sesuai dengan apa yang diinginkan oleh programmer, yaitu agar syntax `print` dapat digunakan langsung untuk menampilkan *property* yang terdapat pada suatu class.

Code

Berikut contoh override method `__str__` pada class BilanganKompleks

In [0]:
```
class BilanganKompleks:
    def __init__(self,a,b):
        self.real=a
        self.im=b
    def display(self):
        print (self.real,"+",self.im,"i")
    def __str__(self):
        return str(self.real)+" + "+str(self.im)+" i "
```

In [0]:
```
data1=BilanganKompleks(4,6)
data2=BilanganKompleks(2,5)
```

In [18]:
```
print('Override Method')
print(data1)
print(data2)
print('Method dalam Class')
data1.display()
data2.display()
```

```
Override Method
4+6i
2+5i
Method dalam Class
4 + 6 i
2 + 5 i
```

Penjumlahan dua buah bilangan kompleks berbeda dengan penjumlahan bilangan yang lain, karena bilangan kompleks ini terdiri dari dua bagian yaitu real dan imajiner. Untuk menjumlahkan bilangan kompleks, maka bagian real harus dijumlahkan pada bagian real juga dari bilangan lain, dan bagian imajiner harus ditambahkan dengan bagian imajiner, seperti persamaan berikut:

$$
\begin{aligned}
x1 &= x_1 + y_1 i \\
x2 &= x_2 + y_2 i \\
x1 + x2 &= (x_1 + y_1 i) + (x_2 + y_2 i) \\
&= (x_1 + x_2) + ((y_1 + y_2) i)
\end{aligned}
$$

Seperti halnya fungsi untuk menampilkan *property* yang terdapat pada suatu class, maka untuk melakukan operasi penjumlahan ini juga dapat dilakukan dengan dua buah cara, yaitu :

- Membuat metode di dalam class untuk operasi penjumlahan
- Override method

Code

Contoh code berikut adalah pembuatan method baru di dalam class, yang berfungsi untuk menjumlahkan dua buah bilangan kompleks.

In [25]:
```
class BilanganKompleks:
    def __init__(self,a,b):
        self.real=a
        self.im=b
    def display(self):
        print (self.real,'+',self.im,'i')
    def __str__(self):
        return str(self.real) + " + " + str(self.im) + " i "
    def addKompleks(self,obj):
        a=self.real+obj.real
        b=self.im+obj.im
        return BilanganKompleks(a,b)
```

In [26]:
```
data1=BilanganKompleks(4,6)
data2=BilanganKompleks(2,5)
```

In [27]:
```
jumlah=data1.addKompleks(data2)
data1.display()
data2.display()
print(jumlah)
```

```
4 + 6 i
2 + 5 i
6 + 11 i
```

Jika ingin melakukan operasi penjumlahan dengan menggunakan operator '+', maka tidak dapat langsung dilakukan perintah sebagai berikut :

```
data1=BilanganKompleks(4,6)
data2=BilanganKompleks(2,5)
jumlah=data1+data2
```

Operasi penjumlahan dengan operator '+' hanya dapat dilakukan dengan *override* fungsi `__add__` yang sudah tersedia.

Code

Berikut adalah contoh override method `__add__` , agar operasi penjumlahan dua buah bilangan kompleks dapat dilakukan dengan menggunakan operator '+'.

In [29]:
```
class BilanganKompleks:
    def __init__(self,a,b):
        self.real=a
        self.im=b
    def display(self):
        print (self.real,'+',self.im,'i')
    def __str__(self):
        return str(self.real) + " + " + str(self.im) + " i "
    def addKompleks(self,obj):
        a=self.real+obj.real
        b=self.im+obj.im
        return BilanganKompleks(a,b)
    def __add__(self,obj):
        a=self.real+obj.real
        b=self.im+obj.im
        return BilanganKompleks(a,b)
```

In [32]:
```
data1=BilanganKompleks(4,6)
data1=BilanganKompleks(5,10)
jumlah=data1+data2
print(data1)
print(data2)
print(jumlah)
```

```
5 + 10 i
2 + 5 i
7 + 15 i
```

Perkalian dua bilangan kompleks juga berbeda dengan perkalian bilangan biasa, karena keistimewaan bilangan kompleks ini. Perkalian dua buah bilangan kompleks dapat dilakukan sebagai berikut :

$$
\begin{aligned}
x1 &= x_1 + y_1 i \\
x2 &= x_2 + y_2 i \\
x1 \times x2 &= (x_1 + y_1 i) \times (x_2 + y_2 i) \\
&= (x_1 \times x_2 - y_1 \times y_2) + ((x_1 \times y_2) + (y_1 \times x_2) i)
\end{aligned}
$$

Code

Berikut overriding method untuk perkalian dua buah bilangan kompleks:

In [33]:
```
class BilanganKompleks:
    def __init__(self,a,b):
        self.real=a
        self.im=b
    def display(self):
        print (self.real,'+',self.im,'i')
    def __str__(self):
        return str(self.real) + " + " + str(self.im) + " i "
    def addKompleks(self,obj):
        a=self.real+obj.real
        b=self.im+obj.im
        return BilanganKompleks(a,b)
    def __add__(self,obj):
        a=self.real+obj.real
        b=self.im+obj.im
        return BilanganKompleks(a,b)
    def __mul__(self,data):
        temp1=(self.real*data.real)-(self.im*data.im)
        temp2=(self.real*data.im)+(data.real*self.im)
        return BilanganKompleks(temp1,temp2)
```

In [34]:
```
a=BilanganKompleks(4,6)
b=BilanganKompleks(5,10)
c=a * b
print(a)
print(b)
print(c)
```

```
4 + 6 i
5 + 10 i
-40 + 70 i
```

Latihan 4

Override method `__sub__` agar dapat dilakukan operasi pengurangan dua buah bilangan kompleks.

[Kembali ke Menu Awal]