**Object-Oriented Programming**

**(Pemrograman berorientasi Obyek)**

OOP (Object Oriented Programming) adalah suatu metode pemrograman yang berorientasi kepada objek. Tujuan dari OOP diciptakan adalah untuk mempermudah pengembangan program dengan cara mengikuti model yang telah ada di kehidupan sehari-hari. Jadi setiap bagian dari suatu permasalahan adalah objek, nah objek itu sendiri merupakan gabungan dari beberapa objek yang lebih kecil lagi.

Istilah dalam pemrograman berorientasi objek

1. class : tipe data yang tidak hanya berisi data tetapi juga method.
2. object : suatu variabel dengan type data class.
3. state/property : bagian class yang berisi nilai.
4. method : fungsi yang melekat pada class.
5. construction : fungsi yang dijalankan oleh python ketika pertama kali suatu objek yang dibuat.
6. Self : 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.
7. Instance : objek yang telah dibuat.
8. Override : mendefinisikan kembali fungsi-fungsi umum yang sudah ada, agar sesuai dengan kebutuhan programmer.

Class merupakan sebuah objek yang di dalam nya biasanya terdapat beberapa metode yang memang merupakan isi dari sebuah class ini. Cara membuat class yaitu “class” yang kemudian di ikuti dengan “nama class nya”, kemudian diakhiri dengan tanda titik dua ‘:’. Statemen-statemen dalam tubuh kelas dapat berupa atribut kelas dan method. Class umumnya di definisikan pada level teratas dari sebuah modul, dengan begitu objek dari class dapat di dibuat dimanapun dalam source code dimana kelas tersebut didefinisikan.

Berikut adalah contoh penerapan class dalam python :

Ex : **class BilanganKompleks:**

**Atribut Kelas**

Atribut merupakan data atau bisa juga berupa fungsi-fungsi yang dimiliki oleh kelas

tersebut. Atribut diakses melalui notasi bertitik. Atribut-atribut kelas terikat hanya untuk

kelas-kelas dimana atribut tersebut didefinisikan. Atribut-atribut data merupakan veriabel-

variabel yang kita deklarasikan. Variabel-variabel tersebut dapat digunakan seperti variabel

lainnya dan dapat di ubah-ubah nilainya oleh method didalam kelas ataupun di dalam

program utama.

**Method**

Method merupakan fungsi yang melekat pada sebuah objek atau instan kelas. Contoh

berikut menunjukkan penggunaan method dalam kelas.

def search (self,item):

current = self.head

found = False

while current != None and not found:

if (current.getData() == item):

found = True

else :

current = current.getNext()

return found

## 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 dioverride 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.

**Linked List**

Linked List adalah suatu struktur data linier. Berbeda dengan array yang juga merupakan struktur data linier dan tipe data komposit, linked list dibentuk secara dinamik. Kekurangan dari linked list, jika ingin merubah suatu data

Pada saat awal program dijalankan elemen linked list belum data. Elemen linked list (disebut node) dibentuk sambil jalan sesuai instruksi. Apabila setiap elemen array dapat diakses secara langsung dengan menggunakan indeks, sebuah node linked list diakses dengan menggunakan pointer yang mengacu (menunjuk) ke node tersebut.

Lokasi data pertama dari **linked list** haruslah diketahui secara eksplisit, sehingga dari data pertama tersebut, data kedua data ditemukan, data ketiga, dan seterusnya. Informasi yang menunjuk pada lokasi data pertama atau **head of the list** tersebut disebut dengan **external reference**. Begitu juga dengan data terakhir, harus ada informasi yang menunjukkan bahwa data tersebut adalah data terakhir di dalam linked list, tidak ada lagi data berikutnya.

Tempat yang disediakan pada satu area memori tertentu untuk menyimpan data dikenal dengan sebutan node atau simpul. Setiap node memiliki pointer yang menunjuk ke simpul berikutnya sehingga terbentuk satu untaian, dengan demikian hanya diperlukan sebuah variabel pointer. Susunan berupa untaian semacam ini disebut Single Linked List (NULL memiliki nilai khusus yang artinya tidak menunjuk ke mana-mana. Biasanya Linked List pada titik akhirnya akan menunjuk ke NULL/NOL/*Ground*).

Untuk membuat struktur data linked list, terlebih dahulu dibuat **node-node** penyusun linked list tersebut. **Node** ini harus memiliki setidaknya dua informasi, yaitu informasi mengenai data atau nilai, dan informasi mengenai node berikutnya. Oleh karena itu **node** dibuat menjadi sebuah tipe data baru yang bertipe **class**, dengan dua informasi yaitu *data* dan *next*.

Terdapat beberapa method penting pada class **node** ini, antara lain:

* *constructor*, yang akan dijalankan setiap instansiasi class
* *getData*, untuk mengetahui informasi data yang terdapat pada node tersebut
* *getNext*, untuk mengetahui informasi node berikutnya, jika tidak ada node berikutnya maka nilai balik berupa *None*
* *setData*, untuk merubah informasi data yang terdapat pada node tersebut
* *setNext*, untuk menentukan node berikutnya yang ditunjukan oleh informasi *next* dari node tersebut

Pointer

Pointer adalah suatu variabel penunjuk, berisi nilai yang menunjuk alamat suatu lokasi memori tertentu. Jadi pointer tidak berisi nilai data, melainkan berisi suatu alamat memori atau null jika tidak berisi data. Pointer yang tidak diinisialisasi disebut dangling   pointer. Lokasi memorit ersebut bisa diwakili sebuah variabel atau dapat juga berupa nilai alamat memori secara langsung.

**Contoh penerapan *class***

**1**

class Node:

def \_\_init\_\_(self,init\_data):

self.data = init\_data

self.next = None

def getData(self):

return self.data

def getNext(self):

return self.next

def setData(self,newdata):

self.data = newdata

def setNext(self,new\_next):

self.next = new\_next

class LinkedList:

def \_\_init\_\_(self):

self.head = None

def isEmpty(self):

return self.head==None

def add(self,item):

temp = Node(item)

temp.setNext(self.head)

self.head = temp

def size(self):

current = self.head

count = 0

while current != None:

count = count + 1

current = current.getNext()

return count

def display(self):

current = self.head

while current != None:

print(current.getData())

current = current.getNext()

def search (self,item):

return found

def remove (self,item):

current = self.head

previous = None

found = False

while not found:

if current.getData() == item:

found = True

else:

previous = current

current = current.getNext()

if previous == None:

self.head = current.getNext()

else :

previous.setNext(current.getNext())

a = Node(93)

b = Node(20)

c = Node(45)

myList = LinkedList()

print(myList.head)

myList.add(34)

print(myList.head.data)

myList.add(34)

myList.add(45)

myList.add(21)

print(myList.head.data)

print(myList.display())

print (myList.search(21))

print(myList.remove(45))

**2**

class BilanganKompleks:

def \_\_init\_\_(self,a,b):

self.re = a

self.im = b

def display(self):

print(self.re,'+',self.im,'i')

def \_\_str\_\_(self):

return str(self.re)+"+"+str(self.im)+"i"

#def addKompleks(self,obj):

#a=self.re+obj.re

#b=self.im+obj.im

#return BilanganKompleks(a,b)

def \_\_add\_\_(self,obj):

a=self.re+obj.re

b=self.im+obj.im

return BilanganKompleks(a,b)

data1 = BilanganKompleks(2,3)

data2 = BilanganKompleks(5,4)

print(data1)

print(data2)

**3**

class BilanganKompleks:

def \_\_init\_\_(self,a,b):

self.re = a

self.im = b

def display2(self):

print(self.re,'+',self.im,'i')

def display(self):

# a = self.re

# b = self.im

print (self.re,"+",self.im,"i")

#temp=str(self.re)+'+'+str(self.im)+'i'

#return (temp)

def \_\_str\_\_(self):

return str(self.re)+"+"+str(self.im)+"i"

def addNum(self,num1,num2):

self.re=num1.re+num2.re

self.im=num1.im+num2.im

def addNum2(self,num):

self.re=self.re+num.re

self.im=self.im+num.im

def addKompleks(self,obj):

a=self.re+obj.re

b=self.im+obj.im

return BilanganKompleks(a,b)

def \_\_add\_\_(self,obj):

a=self.re+obj.re

b=self.im+obj.im

return BilanganKompleks(a,b)

data1 = BilanganKompleks(5,6)

data2= BilanganKompleks(3,4)

print(data1)

print(data2)

data3=BilanganKompleks(0,0)

data3.addNum(data1,data2)

print(data3)

data1.addNum(data1,data2)

print(data1)

data1.addNum2(data2)

print(data1)

jumlah=data1.addKompleks(data2)

print(jumlah)

print (data1+data2)