Object Orientation

Python is a multi-paradigm programming language.

Means python supports
- Procedural programming approach
- Object oriented programming approach
- Functional programming approach

One of the popular approach to solve a programming problem is by creating objects. This is known as Object-Oriented Programming (OOP).

In object oriented programming there are four concepts as

Encapsulation :
Encapsulation is considered as binding characteristics (Variables) and behaviours (Methods) together. To bind characteristics and behaviour we have to design class.

Abstraction :
Abstraction is considered as hiding something from outsider entity.

Polymorphism :
Polymorphism is defined as single name and multiple behaviour.

Inheritance :
Inheritance means reusability. By using inheritance one class can acquire all the characteristics and behaviour of another class.

To understand the all the above concepts first we have to understand the concept of Class.

Class :
Class is consider as blueprint of an object.
Class contains two things as

Characteristics :
Variables of class on which we can perform operations. When we create object of that class memory for the characteristics gets allocated.

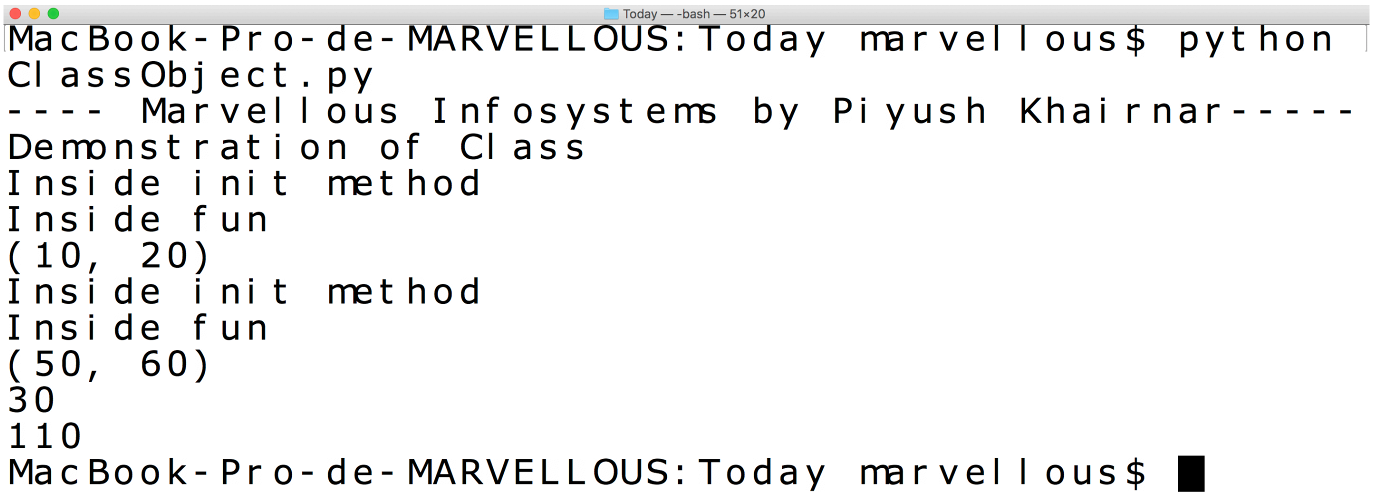
Behaviour :
Behaviours means the methods which performs the operations on characteristics. By using the object of class we can call that methods.

Object :
An object (instance) is an instantiation of a class. When class is defined, only the description for the object is defined. Therefore, no memory or storage is allocated.
But actual memory for the characteristics gets allocated when we create the object.
After creating the object we can access characteristics and behaviour of that class.

Consider below application which demonstrates concepts of Class, Object

```
print("---- Marvellous Infosystems by Piyush Khairnar-----")

print("Demonstration of Class")

class Demo:

    def __init__(self,value1,value2):
        print("Inside init method")
        self.i = value1
        self.j = value2

    def fun(self):
        print("Inside fun")
        print(self.i,self.j)

    def Add(self):
        print(self.i + self.j)

# Create object of Demo class
obj1 = Demo(10,20)

# Call the method fun
obj1.fun()

# Create object of Demo class
obj2 = Demo(50,60)

# Call the method fun
obj2.fun()

# Call method Add to perform addition of characteristics
obj1.Add()
obj2.Add()
```

Output of above application



```
MacBook-Pro-de-MARVELLOUS:Today marvellous$ python
ClassObject.py
---- Marvellous Infosystems by Piyush Khairnar-----
Demonstration of Class
Inside init method
Inside fun
(10, 20)
Inside init method
Inside fun
(50, 60)
30
110
MacBook-Pro-de-MARVELLOUS:Today marvellous$
```

In above application we have Demo class which contains
Two characteristics as i and j
Two behaviours as fun and Add.

We create two objects as Obj1 and Obj2.

In above class there is one method as init

__init__ :
"__init__" is a reserved method in python classes.
It is known as a constructor in object oriented concepts.
This method called when an object is created from the class and it allow the class to initialise the attributes of a class.

In python for every instance method there is one implicit parameter as self.

self :
self represents the instance of the class. By using the "self" keyword we can access the attributes and methods of the class in python.