



***LAB NOTES OF PYTHON
PROGRAMMING
(Classes / Objects)***

By

Dr. Samaher Hussein Ali

Bashar Hamid

Anaam Ghanem

Samaher@itnet.uobabylon.edu.iq

College of Information Technology, University of Babylon, Iraq

Notes of Lab 7

Python Object Oriented

- **OOP Overview:**
- **Object-oriented programming (OOP):** Python is an object oriented programming language. Unlike procedure oriented programming, in which the main emphasis is on functions, object oriented programming stress on objects. Object is simply a collection of data (variables) and methods (functions) that act on those data.
- The class is a fundamental building block in Python.

Python Object Oriented

- **Everything Is An Object :**
- What is the class keyword used for, exactly? Like its function-based def, it concerns the definition of things. While def is used to define a function, And what is a class? Simply a logical grouping of data and functions (the latter of which are frequently referred to as "**methods**" when defined within a class).
- What do we mean by "logical grouping"? Well, a class can contain any data we'd like it to, and can have any functions (methods) attached to it that we please. Rather than just throwing random things together under the name "class", we try to create classes where there is a logical connection between things.

Python Object Oriented

- **OOP Terminology :**
- **Class:** A user-defined prototype for an object that defines a set of attributes that characterize any object of the class. The attributes are data members (class variables and instance variables) and methods, accessed via dot notation.
- **Class variable:** A variable that is shared by all instances of a class. Class variables are defined within a class but outside any of the class's methods.
- **Method :** A special kind of function that is defined in a class definition.

Python Object Oriented

- **OOP Terminology :**
- **Instance:** An individual object of a certain class. An object **person1** that belongs to a class **person**, for example, is an instance of the class **person**.
- **Object :** A unique instance of a data structure that's defined by its class. An object comprises both data members (class variables and instance variables) and methods.
- **An object is also called an instance of a class and the process of creating this object is called instantiation.**

Creating Classes

- The ***class*** statement creates a new class definition. The name of the class immediately follows the keyword *class* followed by a colon as follows:

```
class ClassName  
    class_suite
```

- The ***class_suite*** consists of all the component statements defining class members, data attributes and functions.

Creating Instance Objects

- To create instances of a class, you call the class using **class name**, like :

```
Obj = MyClass()
```

- This will create a new instance object named *obj*. We can access attributes of objects using the object name prefix. Attributes may be data or method. Method of an object are corresponding functions of that class. Any function object that is a class attribute defines a method for objects of that class.

```
p1 = person()  
p2 = person()
```

Accessing Attributes

- You access the object's attributes using the dot operator with object. Class variable would be accessed using class name as follows :

```
Obj = MyClass()  
Obj.funName()  
Obj.attribute
```

```
p1=person()  
p1.info()  
p1.name  
p1.age
```


Creating Classes

- Example: Create a simple Class *person*.

```
class Person:
```

```
    name='noor'
```

← Attribute

```
    def sayHi(self):
```

← Method

```
        print 'Hello, how are you?'
```

```
p = Person()
```

← Instance of person

```
print p.name
```

```
p.sayHi()
```

```
>>>
```

```
noor
```

```
Hello, how are you?
```

Creating Classes

- The variable ***name*** is a **class variable** whose value is shared among all instances of a this class. This can be accessed as ***Person.name*** from inside the class or outside the class.
- You declare other class methods like normal functions with the exception that the first argument to each method is ***self***. Python adds the ***self*** argument to the list for you; you do not need to include it when you call the methods.

Classes

- Example :Create class person with its attributes(name,age,eyecolor)

```
class Person:
    name='noor'
    age=22
    eyecolor='blue'
    def info(self):
        print 'Hello, how are you?'
        print 'Name is: ',self.name
        print "Age is: ",self.age
        print "Eyes color is: ",self.eyecolor

p = Person()
p.info()
```

Classes

- Example :Also you can add an arguments:

```
class Person:
    name=input("Enter Your Name:")
    age=input("Enter Age:")
    eyecolor=input("Enter Eyes Color:")
    def info(self , food):
        food=input("Enter Food You Like:")
        print 'Name is: ',self.name
        print "Age is: ",self.age
        print "Eyes color is: ",self.eyecolor
        print "I will eat: ",food
p = Person()
p.info("Apple")
```

Classes

- Result like this,

```
>>>  
Enter Your Name:'Ali'  
Enter Age:22  
Enter Eyes Color:'blue'  
Enter Food You Like:'Apple'  
Name is:  Ali  
Age is:  22  
Eyes color is:  blue  
I will eat:  Apple  
>>>
```

Classes

- Example :Create class Car, with attributes and 2 methods, and then change value of attributes when create instance of the class.

```
class car:
    name=''
    speed=0
    color=''
    model=''
    def move(self):
        print "OK, I will MOVE..."
    def printCar(self):
        print "Car Name: ",self.name
        print "Car Speed: ",self.speed
        print "Car Color: ",self.color
        print "Car Model: ",self.model

c = car()
c.name='BMW'
c.speed=180
c.model=1998
c.color='black'
c.printCar()
c.move()
|
```

```
>>>
Car Name:   BMW
Car Speed:  180
Car Color:  black
Car Model:  1998
OK, I will MOVE...
>>> |
```

Constructors in Python

- Class functions that begins with double underscore (__) are called special functions as they have special meaning. Of one particular interest is the **`__init__()`** function. This special function gets called whenever a new object of that class is instantiated. This type of function is also called constructors in Object Oriented Programming (OOP).
- It is called class **constructor** or **initialization method** that Python calls when you create a new instance of this class.
- To create instances of a class, you call the class using class name and pass in whatever arguments its `__init__` method accepts.

```
p1 = person('Ali',24)
p2 = person('noor',20)
```

Classes

- Example: Create Class with Constructor using `__init__()` method.

```
class person:
    def __init__(self, myname, myage):
        print "I Make a New Person !"
        self.name = myname
        self.age = myage
    def printInfo(self):
        print "Name is: ", self.name
        print "Age is: ", self.age

p = person('Bashar', 23)
p.printInfo()
```


Classes

- Example: Create Class customer with Constructor.

```
class Customer:
    def __init__(self,name,phone,email):
        print "I Make a New Customer !"
        self.name = name
        self.phone = phone
        self.email=email
    def printInfo(self):
        print "Name is: ",self.name
        print "phone is: ",self.phone
        print "email is: ",self.email

name=input("enter name:")
phone=input("enter phone:")
email=input("enter email:")
cust=Customer(name,phone,email)
cust.printInfo()
```

```
>>>
enter name:'Nada'
enter phone:'808080'
enter email:'nada@gmail.com'
I Make a New Customer !
Name is:  Nada
phone is:  808080
email is:  nada@gmail.com
>>> |
```

Classes

- Example: set attributes with default values in initialization method.

```
class car:
    def __init__(self,name='blank',speed=0):
        self.name = name
        self.speed = speed
    def printCar(self):
        print "Car Name: ",self.name
        print "Car Speed: ",self.speed
```

```
>>> mycar=car()
>>> mycar.printCar()
Car Name:  blank
Car Speed:  0
>>>
>>> ford=car("Ford",200)
>>> ford.printCar()
Car Name:  Ford
Car Speed:  200
>>> |
```

Class Inheritance

- Instead of starting from scratch, you can create a class by deriving it from a preexisting class by listing the parent class in parentheses after the new class name.
- The child class inherits the attributes of its parent class, and you can use those attributes as if they were defined in the child class. A child class can also override data members and methods from the parent.

```
class SubClassName (ParentClass1[, ParentClass2, ...]):  
    class_suite
```

Class Inheritance

- Example: set attributes with default values in initialization method.

```
class man:
    name=''
    def __init__(self,name):
        self.name=name
        print "Name : ",self.name

class superman(man):
    secret_ID=''
    def __init__(self,name,secret_ID):
        man.__init__(self,name)
        self.secret_ID=secret_ID
        print self.secret_ID

m=man("John")
s=superman("Ali","SUPER")
|
```