Python Objects and Class

What are classes and objects in Python?

Python is an object oriented programming language. Unlike procedure oriented programming, where 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. And, class is a blueprint for the object.

An object is also called an instance of a class and the process of creating this object is called **instantiation**.

Like function definitions begin with the keyword def, in Python, we define a class using the keyword class.

Create a Class

*The class has a documentation string, which can be accessed via *ClassName.__doc__*.

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

To create a class, use the keyword class:

Example:

Create a class named MyClass, with a property named x:

```
class MyClass:
  x = 5
print(MyClass)
```

Output: <class '__main__.MyClass'>

Class Objects

Class objects support two kinds of operations: attribute references and instantiation.

Attribute references use the standard syntax used for all attribute references in Python: obj.name. Valid attribute names are all the names that were in the class's namespace when the class object was created. So, if the class definition looked like this:

```
class MyClass:
    """A simple example class"""
    i = 12345

    def f(self):
        return 'hello world'
```

then MyClass.i and MyClass.f are valid attribute references, returning an integer and a function object, respectively. Class attributes can also be assigned to, so you can change the value of MyClass.i by assignment. __doc__ is also a valid attribute, returning the docstring belonging to the class: "A simple example class".

Class *instantiation* uses function notation. Just pretend that the class object is a parameterless function that returns a new instance of the class. For example (assuming the above class):

```
x = MyClass()
```

creates a new *instance* of the class and assigns this object to the local variable x.

Create Object

Now we can use the class named MyClass to create objects:

Example

Create an object named p1, and print the value of x:

```
class MyClass:
  x = 5
p1 = MyClass()
print(p1.x)
```

Output- 5

The __init__() Function

All classes have a function called __init__(), which is always executed when the class is being initiated.which is also called class constructor or initialization method that Python calls when you create a new instance of this class.

Use the __init__() function to assign values to object properties, or other operations that are necessary to do when the object is being created:

Example

Create a class named Person, use the __init__() function to assign values for name and age:

```
class Person:

  def __init__(self, name, age):

    self.name = name

    self.age = age

p1 = Person("John", 36)

print(p1.name)

print(p1.age)
```

Output: John
36

Note: The __init__() function is called automatically every time the class is being used to create a new object.

Object Methods

Objects can also contain methods. Methods in objects are functions that belong to the object.

Let us create a method in the Person class:

Example

Insert a function that prints a greeting, and execute it on the p1 object:

```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age
  def myfunc(self):
    print("Hello my name is " + self.name)


p1 = Person("John", 36)
p1.myfunc()
```

Output: Hello my name is John

The self Parameter

The self parameter is a reference to the current instance of the class, and is used to access variables that belongs to the class.

*It does not have to be named self , you can call it whatever you like, but it has to be the first parameter of any function in the class:

Accessing Attributes

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

emp1.displayEmployee()
emp2.displayEmployee()
print "Total Employee %d" % Employee.empCount

**You can add, remove, or modify attributes of classes and objects at any time

*Try below program considering all concepts together:

```
class Employee:
   'Common base class for all employees'
   empCount = 0

   def __init__(self, name, salary):
      self.name = name
      self.salary = salary
      Employee.empCount += 1

   def displayCount(self):
     print "Total Employee %d" % Employee.empCount

   def displayEmployee(self):
      print "Name : ", self.name,  ", Salary: ", self.salary

"This would create first object of Employee class"
emp1 = Employee("Zara", 2000)
"This would create second object of Employee class"
emp2 = Employee("Manni", 5000)
emp1.displayEmployee()
emp2.displayEmployee()
print "Total Employee %d" % Employee.empCount
```

Python Class: Exercises

1.Write a Python program to find validity of a string of parentheses, '(', ')', '{', '}', '[' and ']. These brackets must be close in the correct order,
for example "()" and "()[]{}" are valid but "[)", "({[)]" and "{{{" are invalid.

2. Write a Python class to implement pow(x, n).

3. Write a Python class which has two methods get_String and print_String. get_String accept a string from the user and print_String print the string in upper case.