# Python Tutorials and Notes

## Python OOPs: Class, Object, Inheritance and Constructor with Example


SETScholars & WACAMLDS

# OOPs in Python

**OOPs in Python** is a programming approach that focuses on using objects and classes as same as other general programming languages. The objects can be any real-world entities. Python allows developers to develop applications using the OOPs approach with the major focus on code reusability. It is very easy to create classes and objects in Python.

# What is a Class?

A Class in Python is a logical grouping of data and functions. It gives the freedom to create data structures that contains arbitrary content and hence easily accessible.

For example, for any bank employee who want to fetch the customer details online would go to **customer class**, where all its attributes like transaction details, withdrawal and deposit details, outstanding debt, etc. would be listed out.

In this tutorial, we will learn,

- How to define Python classes
- How Inheritance works
- Python Constructors

# How to define Python classes

To define class you need to consider following points

**Step 1)** In Python, classes are defined by the **"Class"** keyword

```
class myClass():
```

**Step 2)** Inside classes, you can define functions or methods that are part of this class

```
def method1 (self):
  print "Guru99"
def method2 (self,someString):
  print "Software Testing:" + someString
```

- Here we have defined method1 that prints "Guru99."
- Another method we have defined is method2 that prints "Software Testing"+ SomeString. SomeString is the variable supplied by the calling method

**Step 3)** Everything in a class is indented, just like the code in the function, loop, if statement, etc. Anything not indented is not in the class

```
2 class myClass():
3     def method1(self):
4         print "Guru99"
5
6     def method2(self,someString):
7         print "Software Testing:" + someString
8
```

**NOTE:** About using "self" in Python

- The self-argument refers to the object itself. Hence the use of the word self. So inside this method, self will refer to the specific instance of this object that's being operated on.
- Self is the name preferred by convention by Pythons to indicate the first parameter of instance methods in Python. It is part of the Python syntax to access members of objects

**Step 4)** To make an object of the class

```
c = myClass()
```

**Step 5)** To call a method in a class

```
c.method1()
c.method2(" Testing is fun")
```

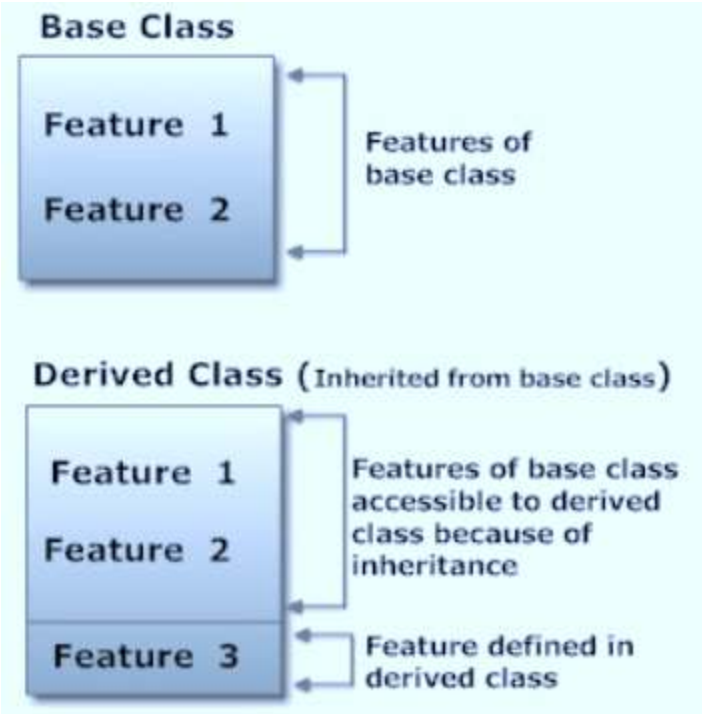
- Notice that when we call the method1 or method2, we don't have to supply the self-keyword. That's automatically handled for us by the Python runtime.
- Python runtime will pass "self" value when you call an instance method on in instance, whether you provide it deliberately or not
- You just have to care about the non-self arguments

**Step 6)** Here is the complete code

```
# Example file for working with classes
class myClass():
  def method1(self):
     print("Guru99")

  def method2(self,someString):
     print("Software Testing:" + someString)


def main():
  # exercise the class methods
  c = myClass ()
  c.method1()
  c.method2(" Testing is fun")

if __name__== "__main__":
  main()
```

# How Inheritance works

Inheritance is a feature used in object-oriented programming; it refers to defining a new class with less or no modification to an existing class. The new class is called **derived class** and from one which it inherits is called the **base**. Python supports inheritance; it also supports **multiple inheritances**. A class can inherit attributes and behavior methods from another class called subclass or heir class.




## Python Inheritance Syntax

```
class DerivedClass(BaseClass):
    body_of_derived_class
```

**Step 1)** Run the following code

```
# Example file for working with classes
class myClass():
  def method1(self):
     print("Guru99")


class childClass(myClass):
  #def method1(self):
       #myClass.method1(self);
       #print ("childClass Method1")

  def method2(self):
       print("childClass method2")

def main():
  # exercise the class methods
  c2 = childClass()
  c2.method1()
  #c2.method2()

if __name__== "__main__":
  main()
```

Notice that the in childClass, method1 is not defined but it is derived from the parent myClass. The output is "Guru99."

**Step 2)** Uncomment Line # 8 & 10. Run the code

Now, the method 1 is defined in the childClass and output "childClass Method1" is correctly shown.

**Step 3)** Uncomment Line #9. Run the code

You can call a method of the parent class using the syntax

```
    ParentClassName.MethodName(self)
```

In our case, we call, myClass.method1(self) and Guru99 is printed as expected

**Step 4)** Uncomment Line #19. Run the code.

Method 2 of the child class is called and "childClass method2" is printed as expected.

# Python Constructors

A constructor is a class function that instantiates an object to predefined values.

It begins with a double underscore (_). It __init__() method

In below example we are taking name of the user using constructor.

```
class User:
    name = ""

    def __init__(self, name):
        self.name = name

    def sayHello(self):
        print("Welcome to Guru99, " + self.name)

User1 = User("Alex")
User1.sayHello()
```

Output will be:

Welcome to Guru99, Alex

### Python 2 Example

Above codes are Python 3 examples, If you want to run in Python 2 please consider following code.

```
# How to define Python classes
# Example file for working with classes
class myClass():
  def method1(self):
     print "Guru99"

  def method2(self,someString):
     print "Software Testing:" + someString



def main():
  # exercise the class methods
  c = myClass ()
  c.method1()
  c.method2(" Testing is fun")

if __name__== "__main__":
  main()


#How Inheritance works
# Example file for working with classes
class myClass():
  def method1(self):
     print "Guru99"


class childClass(myClass):
  #def method1(self):
       #myClass.method1(self);
       #print "childClass Method1"

  def method2(self):
       print "childClass method2"

def main():
  # exercise the class methods
  c2 = childClass()
  c2.method1()
  #c2.method2()

if __name__== "__main__":
  main()
```

## Summary:

"Class" is a logical grouping of functions and data. Python class provides all the standard features of Object Oriented Programming.

- Class inheritance mechanism
- A derived class that override any method of its base class
- A method can call the method of a base class with the same name
- Python Classes are defined by keyword **"class"** itself
- Inside classes, you can define functions or methods that are part of the class
- Everything in a class is indented, just like the code in the function, loop, if statement, etc.
- The self argument in Python refers to the object itself. Self is the name preferred by convention by Pythons to indicate the first parameter of instance methods in Python
- **Python runtime will pass "self" value automatically when you call an instance method on in instance, whether you provide it deliberately or not**

- In Python, a class can inherit attributes and behavior methods from another class called subclass or heir class.