Python Class

Python is a completely object-oriented language. This approach towards programming seeks to treat data and functions as part of a single unit called object. The class defines attributes and the behaviour of the object, while the object, on the other hand, represents the class.

We have been (unknowingly) working with classes and objects right from the beginning of these tutorials. Every element in a Python program is an object of a class. A number, string, list, dictionary, etc. used in a program is an object of a corresponding built-in class. Even the function defined using the `def` keyword belongs to a function class.

```
>>>num=20
>>>type(num)
<class 'int'>
>>> s="Python"
>>> type(s)
<class 'str'>
```

Defining a Class

A class in Python can be defined using the **class** keyword. A class typically includes the following members:

1. Constructor
2. Instance Attributes
3. Class Attributes
4. Methods

A class can also be defined without any members. The following is a `person` class with a doc string.

```
>>> class person:
...    '''doc string: This is empty person class'''
...    pass
...
```

We can declare an object of the above person class as we would for any built-in class.

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

The following class includes attributes, a constructor, instance attributes/variables and methods.

Example: Class

```
class person:
```

```
    count=0 #class attribute
    def __init__(self): #constructor
        self.name="unknown" #instance attribute
        self.age=0 #instance attribute
    def displayInfo(self): #method
        print(self.name, self.age)
```

In the above example of the `person` class, `count` is a class attribute, `__init__(self)` is a constructor, `name` and `age` are instance attributes/variables and `displayInfo` is a method of the `person` class.

Let's learn about each of these in detail.

Constructor

In Python, the constructor method is invoked automatically whenever a new object of a class is instantiated, same as constructors in C# or Java. The constructor must have a special name `__init__()` and a special parameter called `self`.

Note:
The first parameter of each method in a class must be the **self** which refers to the calling object. However, you can give any name to the first parameter, not necessary 'self'.

The following example defines a constructor.

Example: Constructor
```
class person:
    def __init__(self): # constructor method
        print('Constructor invoked')
```

Now, whenever you create an object of the person class, the `__init__()` constructor method will be called, as shown below.

```
>>>p1 = person()
Constructor invoked
>>>p2 = person()
Constructor invoked
```

The constructor in Python is used to define the attributes of an instance and assign values to them.

Instance Attributes

Instance attributes are attributes or properties attached to an instance of a class. Instance attributes are defined in the constructor.

The following example defines instance attributes `name` and `age` in the constructor.

Example: Instance Attributes

```
class person:
    def __init__(self): # constructor
        self.name="Unknown" # instance attribute
        self.age=0 # instance attribute
```

An instance attribute can be accessed using dot notation: `[instance name].[attribute name]`, as shown below.

```
>>> p1=person()
>>> p1.name
Unknown
>>> p1.age
0
```

You can set the value of attributes using the dot notation, as shown below.

```
>>> p1=person()
>>> p1.name="Bill"
>>> p1.age=25
>>> p1.name
Bill
>>> p1.age
25
```

You can specify the values of instance attributes through the constructor. The following constructor includes the name and age parameters, other than the `self` parameter.

Example: Setting Attribute Values

```
class person:
    def __init__(self, name, age):
        self.name=name
        self.age=age
```

Now, you can specify the values while creating an instance, as shown below.

```
>>> p1=person("Bill",25)
>>> p1.name
Bill
>>> p1.age
25
```

Note:
You don't have to specify the value of the **self** parameter. It will be assigned internally in Python.

You can also set default values for instance attributes. The following code sets the default values of the constructor parameters. So, if the values are not provided when creating an object, they values will be assigned latter.

Example: Setting Default Values of Attributes

```
class person:
    def __init__(self, name="Guest", age=25)
        self.name=name
        self.age=age
```

Now, you can create an object with default values as shown below.

```
>>> p1=person()
>>> p1.name
Guest
>>> p1.age
25
```

Class Attributes

Class attributes are different from instance attributes. An attribute whose value is the same for all instances of a class is called a class attribute. The value of class attribute is shared by all objects. Class attributes are defined at class level rather than inside the constructor method `__init__()`. Unlike instance attributes, class attributes are accessed using the name of the class.

Example: Class Attribute

```
class person:
    greet='Hello!'
```

The above person class includes a class attribute called `greet`. This attribute can be accessed using the class name, as shown below.

```
>>> person.greet
Hello!
```

Each object of the person class can have this class attribute accessed using object.[attribute name].

```
>>> p1=person()
>>> p1.greet
Hello!
>>> p2=person()
Hello!
```

Changing the class attribute using the class name will be reflected for all the instances of a class.

```
>>> person.greet
Hello!
>>> p1=person()
>>> p1.greet
Hello!
```

```
>>> person.greet='Hi!'
>>> p1.greet
Hi!
```

However, changing the class attribute using the instance will not reflect elsewhere. It will affect only that particular instance.

```
>>> person.greet
Hello!
>>> p1=person()
>>> p1.greet
Hello!
>>> p1.greet=â€How are you doing?â€
>>> p1.greet
How are you doing?
>>> person.greet
Hello!
>>> p2=person()
>>> p2.greet
Hello!
```

Consider the following example.

Example: person.py

```
class person:
    totalObjects=0
    def __init__(self):
        person.totalObjects=person.totalObjects+1
```

In the above example, `totalObject` is an attribute in the person class. Whenever a new object is created, the value of `totalObjects` is incremented by 1. You can now access the `totalObjects` attribute after creating the objects, as shown below.

```
>>> p1=person()
>>> p1.totalObjects
1
>>> p2=person()
>>> p2.totalObjects
2
```

Class Methods

You can define as many methods as you want in a class using the `def` keyword. Each method must have the first parameter, generally named as `self` which refers to the calling instance.

The following example includes the `displayInfo` method in a person class.

Example: Class Method

```
class person:
    def __init__(self):
        self.name="unknown"
        self.age=0
    def displayInfo(self): #method
        print(self.name, self.age)
```

As you can see, the instance attributes of an instance can be accessed using the `self` parameter.

The methods of a class can be called using an instance, as show below.

```
>>> p1=person()
>>> p1.displayInfo()
unknown 0
```