Python Constructor

A constructor is a special type of method (function) which is used to initialize the instance members of the class.

Constructors can be of two types.

1. Parameterized Constructor
2. Non-parameterized Constructor

Constructor definition is executed when we create the object of this class. Constructors also verify that there are enough resources for the object to perform any start-up task.

Creating the constructor in python

In python, the method **__init__** simulates the constructor of the class. This method is called when the class is instantiated. We can pass any number of arguments at the time of creating the class object, depending upon **__init__** definition. It is mostly used to initialize the class attributes. Every class must have a constructor, even if it simply relies on the default constructor.

Consider the following example to initialize the Employee class attributes.

Example

```
class Employee:
    def __init__(self,name,id):
        self.id = id;
        self.name = name;
    def display (self):
        print("ID: %d \nName: %s"%(self.id,self.name))
emp1 = Employee("John",101)
emp2 = Employee("David",102)

#accessing display() method to print employee 1 information

emp1.display();

#accessing display() method to print employee 2 information
emp2.display();
```

Output:

```
ID: 101
Name: John
ID: 102
Name: David
```

Example: Counting the number of objects of a class

```
class Student:
    count = 0
    def __init__(self):
        Student.count = Student.count + 1
s1=Student()
s2=Student()
s3=Student()
print("The number of students:",Student.count)
```

Output:

```
The number of students: 3
```

Python Non-Parameterized Constructor Example

```
class Student:
    # Constructor - non parameterized
    def __init__(self):
        print("This is non parametrized constructor")
    def show(self,name):
        print("Hello",name)
student = Student()
student.show("John")
```

Output:

```
This is non parametrized constructor
Hello John
```

Python Parameterized Constructor Example

```
class Student:
    # Constructor - parameterized
    def __init__(self, name):
        print("This is parametrized constructor")
        self.name = name
    def show(self):
        print("Hello",self.name)
student = Student("John")
student.show()
```

Output:

```
This is parametrized constructor
Hello John
```

Python In-built class functions

The in-built functions defined in the class are described in the following table.

SN	Function	Description
1	getattr(obj,name,default)	It is used to access the attribute of the object.
2	setattr(obj, name,value)	It is used to set a particular value to the specific attribute of an object.
3	delattr(obj, name)	It is used to delete a specific attribute.
4	hasattr(obj, name)	It returns true if the object contains some specific attribute.

Example

```
class Student:
    def __init__(self,name,id,age):
        self.name = name;
        self.id = id;
        self.age = age

#creates the object of the class Student
s = Student("John",101,22)

#prints the attribute name of the object s
print(getattr(s,'name'))

# reset the value of attribute age to 23
setattr(s,"age",23)

# prints the modified value of age
print(getattr(s,'age'))

# prints true if the student contains the attribute with name id

print(hasattr(s,'id'))
# deletes the attribute age
delattr(s,'age')

# this will give an error since the attribute age has been deleted
print(s.age)
```

Output:

```
John
23
True
AttributeError: 'Student' object has no attribute 'age'
```

Built-in class attributes

Along with the other attributes, a python class also contains some built-in class attributes which provide information about the class.

The built-in class attributes are given in the below table.

SN	Attribute	Description
1	__dict__	It provides the dictionary containing the information about the class namespace.
2	__doc__	It contains a string which has the class documentation
3	__name__	It is used to access the class name.
4	__module__	It is used to access the module in which, this class is defined.
5	__bases__	It contains a tuple including all base classes.

Example

```
1.  class Student:
2.      def __init__(self,name,id,age):
3.          self.name = name;
4.          self.id = id;
5.          self.age = age
6.      def display_details(self):
7.          print("Name:%s, ID:%d, age:%d"%(self.name,self.id))
8.  s = Student("John",101,22)
9.  print(s.__doc__)
10. print(s.__dict__)
11. print(s.__module__)
```

Output:

```
None
{'name': 'John', 'id': 101, 'age': 22}
__main__
```