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

Creating the constructor in python

In Python, the method the **__init__()** simulates the constructor of the class. This method is called when the class is instantiated. It accepts the **self**-keyword as a first argument which allows accessing the attributes or method of the class.

We can pass any number of arguments at the time of creating the class object, depending upon the **__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("DISHA", 101)
emp2 = Employee("RAM
```

Counting the number of objects of a class

The constructor is called automatically when we create the object of the class. Consider the following example.

Example

```
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)
```

Python Non-Parameterized Constructor

The non-parameterized constructor uses when we do not want to manipulate the value or the constructor that has only self as an argument. Consider the following example.

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("DISHA")
```

Python Parameterized Constructor

The parameterized constructor has multiple parameters along with the **self**. Consider the following example.

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("DISHA")
student.show()
```

Python Default Constructor

When we do not include the constructor in the class or forget to declare it, then that becomes the default constructor. It does not perform any task but initializes the objects. Consider the following example.

Example

```
class Student:
    roll_num = 101
    name = "Jay"

    def display(self):
        print(self.roll_num,self.name)

st = Student()
st.display()
```

More than One Constructor in Single class

Let's have a look at another scenario, what happen if we declare the two same constructors in the class.

Example

```
class Student:
    def __init__(self):
        print("The First Constructor")
    def __init__(self):
        print("The second contructor")

st = Student()
```

Python built-in class functions

The built-in 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

s = Student("aman", 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)
```

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

```
class Student:
    def __init__(self,name,id,age):
        self.name = name;
        self.id = id;
        self.age = age
    def display_details(self):
        print("Name:%s, ID:%d, age:%d"%(self.name,self.id))
s = Student("disha",101,22)
print(s.__doc__)
print(s.__dict__)
print(s.__module__)
```

```
", 102)



emp1.display()


emp2.display()
```