Python - Constructors

Python **constructor** is an instance method in a class, that is automatically called whenever a new object of the class is created. The constructor's role is to assign value to instance variables as soon as the object is declared.

Python uses a special method called **__init__()** to initialize the instance variables for the object, as soon as it is declared.

Creating a constructor in Python

The **`__init__()`** method acts as a constructor. It needs a mandatory argument named **`self`**, which is the reference to the object.

```
def __init__(self, parameters):
#initialize instance variables
```

The **`__init__()`** method as well as any instance method in a class has a mandatory parameter, **`self`**. However, you can give any name to the first parameter, not necessarily self.

Types of Constructor in Python

Python has two types of constructor −

- Default Constructor
- Parameterized Constructor

Learn **Python** in-depth with real-world projects through our **Python certification course**. Enroll and become a certified expert to boost your career.

Default Constructor in Python

The Python constructor which does not accept any parameter other than **`self`** is called as default constructor.

Example

Let us define the **`constructor`** in the Employee class to initialize name and age as instance variables. We can then access these attributes through its object.

</> Open Compiler

```
class Employee:
   'Common base class for all employees'
   def __init__(self):
      self.name = "Bhavana"
      self.age = 24

e1 = Employee()
print ("Name: {}".format(e1.name))
print ("age: {}".format(e1.age))
```

It will produce the following **output** −

```
Name: Bhavana
age: 24
```

For the above Employee class, each object we declare will have same value for its instance variables name and age. To declare objects with varying attributes instead of the default, define arguments for the __init__() method.

Parameterized Constructor

If a constructor is defined with multiple parameters along with **`self`** is called as parameterized constructor.

Example

In this example, the __init__() constructor has two formal arguments. We declare Employee objects with different values −

</> Open Compiler

```
class Employee:
   'Common base class for all employees'
   def __init__(self, name, age):
      self.name = name
      self.age = age

e1 = Employee("Bhavana", 24)
e2 = Employee("Bharat", 25)
```

```
print ("Name: {}".format(e1.name))
print ("age: {}".format(e1.age))
print ("Name: {}".format(e2.name))
print ("age: {}".format(e2.age))
```

It will produce the following **output** −

```
Name: Bhavana
age: 24
Name: Bharat
age: 25
```

You can also assign default values to the formal arguments in the constructor so that the object can be instantiated with or without passing parameters.

</> Open Compiler

```
class Employee:
   'Common base class for all employees'
   def __init__(self, name="Bhavana", age=24):
      self.name = name
      self.age = age

e1 = Employee()
e2 = Employee("Bharat", 25)

print ("Name: {}".format(e1.name))
print ("age: {}".format(e1.age))
print ("Name: {}".format(e2.name))
print ("age: {}".format(e2.age))
```

It will produce the following **output** −

```
Name: Bhavana
age: 24
Name: Bharat
age: 25
```

Python - Instance Methods

In addition to the __init__() constructor, there may be one or more instance methods defined in a class. A method with self as one of the formal arguments is called instance method, as it is called by a specific object.

Example

In the following example a displayEmployee() method has been defined as an instance method. It returns the name and age attributes of the Employee object that calls the method.

```
class Employee:
   def __init__(self, name="Bhavana", age=24):
      self.name = name
      self.age = age
   def displayEmployee(self):
      print ("Name : ", self.name, ", age: ", self.age)

e1 = Employee()
e2 = Employee("Bharat", 25)

e1.displayEmployee()
e2.displayEmployee()
```

It will produce the following **output** –

```
Name : Bhavana , age: 24
Name : Bharat , age: 25
```

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

```
# Add a 'salary' attribute
emp1.salary = 7000
# Modify 'name' attribute
emp1.name = 'xyz'
# Delete 'salary' attribute
del emp1.salary
```

Instead of using the normal statements to access attributes, you can use the following functions –

- The **getattr(obj, name[, default])** − to access the attribute of object.
- The **hasattr(obj,name)** − to check if an attribute exists or not.
- The **setattr(obj,name,value)** − to set an attribute. If attribute does not exist, then it would be created.
- The **delattr(obj, name)** − to delete an attribute.

```
# Returns true if 'salary' attribute exists
print (hasattr(e1, 'salary'))
# Returns value of 'name' attribute
print (getattr(e1, 'name'))
# Set attribute 'salary' at 8
setattr(e1, 'salary', 7000)
# Delete attribute 'age'
delattr(e1, 'age')
```

It will produce the following **output** −

```
False
Bhavana
```

Python Multiple Constructors

As mentioned earlier, we define the __init__() method to create a constructor. However, unlike other programming languages like C++ and Java, Python does not allow multiple constructors.

If you try to create multiple constructors, Python will not throw an error, but it will only consider the last __init__() method in your class. Its previous definition will be overridden by the last one.

But, there is a way to achieve similar functionality in Python. We can overload constructors based on the type or number of arguments passed to the __init__() method. This will allow a single constructor method to handle various initialization scenarios based on the arguments provided.

Example

The following example shows how to achieve functionality similar to multiple constructors.

</> Open Compiler

```
class Student:
   def __init__(self, *args):
      if len(args) == 1:
         self.name = args[0]

      elif len(args) == 2:
         self.name = args[0]
         self.age = args[1]

      elif len(args) == 3:
         self.name = args[0]
         self.age = args[1]
         self.gender = args[2]

st1 = Student("Shrey")
print("Name:", st1.name)
st2 = Student("Ram", 25)
print(f"Name: {st2.name} and Age: {st2.age}")
st3 = Student("Shyam", 26, "M")
print(f"Name: {st3.name}, Age: {st3.age} and Gender: {st3.gender}")
```

When we run the above code, it will produce the following **output** –

```
Name: Shrey
Name: Ram and Age: 25
Name: Shyam, Age: 26 and Gender: M
```