

Python Classes

A class is considered as a blueprint of objects. We can think of the class as a sketch (prototype) of a house. It contains all the details about the floors, doors, windows, etc. Based on these descriptions we build the house. House is the object.

Since many houses can be made from the same description, we can create many objects from a class.

Define Python Class

We use the `class` keyword to create a class in Python. For example,

```
class ClassName:  
    # class definition
```

Here, we have created a class named `ClassName`.

Let's see an example,

```
class Bike:  
    name = ""  
    gear = 0
```

Here,

- `Bike` - the name of the class
- `name/gear` - variables inside the class with default values `""` and `0` respectively.

Note: The variables inside a class are called attributes.

Python Objects

An object is called an instance of a class. For example, suppose `Bike` is a class then we can create objects like `bike1`, `bike2`, etc from the class.

Here's the syntax to create an object.

```
objectName = ClassName()
```

Let's see an example,

```
# create class
class Bike:
    name = ""
    gear = 0

# create objects of class
bike1 = Bike()
```

Here, `bike1` is the object of the class. Now, we can use this object to access the class attributes.

Access Class Attributes Using Objects

We use the `.` notation to access the attributes of a class. For example,

```
# modify the name attribute
bike1.name = "Mountain Bike"

# access the gear attribute
bike1.gear
```

Here, we have used `bike1.name` and `bike1.gear` to change and access the value of `name` and `gear` attribute respectively.

Example 1: Python Class and Objects

```
# define a class
class Bike:
    name = ""
    gear = 0

# create object of class
bike1 = Bike()

# access attributes and assign new values
bike1.gear = 11
bike1.name = "Mountain Bike"

print(f"Name: {bike1.name}, Gears: {bike1.gear} ")
Run Code
```

Output

```
Name: Mountain Bike, Gears: 11
```

In the above example, we have defined the class named `Bike` with two attributes: `name` and `gear`.

We have also created an object `bike1` of the class `Bike`.

Finally, we have accessed and modified the attributes of an object using the `.` notation.

Create Multiple Objects of Python Class

We can also create multiple objects from a single class. For example,

```
# define a class
class Employee:
    # define an attribute
    employee_id = 0

# create two objects of the Employee class
employee1 = Employee()
employee2 = Employee()

# access attributes using employee1
employee1.employeeID = 1001
print(f"Employee ID: {employee1.employeeID}")

# access attributes using employee2
employee2.employeeID = 1002
print(f"Employee ID: {employee2.employeeID}")
Run Code
```

Output

```
Employee ID: 1001
Employee ID: 1002
```

In the above example, we have created two objects `employee1` and `employee2` of the `Employee` class.

Python Methods

We can also define a function inside a Python class. A [Python Function](#) defined inside a class is called a method.

Let's see an example,

```
# create a class
class Room:
    length = 0.0
    breadth = 0.0

    # method to calculate area
    def calculate_area(self):
        print("Area of Room =", self.length * self.breadth)

# create object of Room class
study_room = Room()

# assign values to all the attributes
study_room.length = 42.5
study_room.breadth = 30.8

# access method inside class
study_room.calculate_area()
Run Code
```

Output

```
Area of Room = 1309.0
```

In the above example, we have created a class named `Room` with:

- **Attributes:** `length` and `breadth`
- **Method:** `calculate_area()`

Here, we have created an object named `study_room` from the `Room` class. We then used the object to assign values to attributes: `length` and `breadth`.

Notice that we have also used the object to call the method inside the class,

```
study_room.calculate_area()
```

Here, we have used the `.` notation to call the method. Finally, the statement inside the method is executed.

Python Constructors

Earlier we assigned a default value to a class attribute,

```
class Bike:
    name = ""
    ...
# create object
bike1 = Bike()
```

However, we can also initialize values using the constructors. For example,

```
class Bike:

    # constructor function
    def __init__(self, name = ""):
        self.name = name

bike1 = Bike()
```

Here, `__init__()` is the constructor function that is called whenever a new object of that class is instantiated.

The constructor above initializes the value of the `name` attribute. We have used the `self.name` to refer to the `name` attribute of the `bike1` object.

If we use a constructor to initialize values inside a class, we need to pass the corresponding value during the object creation of the class.

```
bike1 = Bike("Mountain Bike")
```

Here, "Mountain Bike" is passed to the `name` parameter of `__init__()`.