**CourseName**: Advance Python Programming **Course code:** 22CSH-623

# Experiment-2.5

**A) Aim:** Write a python program to demonstrate and define a class and object.

**Tools/Software Required:** Visual Studio Code

**Description:**

Class creates a user-defined data structure, which holds its own data members and member functions, which can be accessed and used by creating an instance of that class. A class is like a blueprint for an object. An Object is an instance of a Class. A class is like a blueprint while an instance is a copy of the class with actual values.

**Steps:**

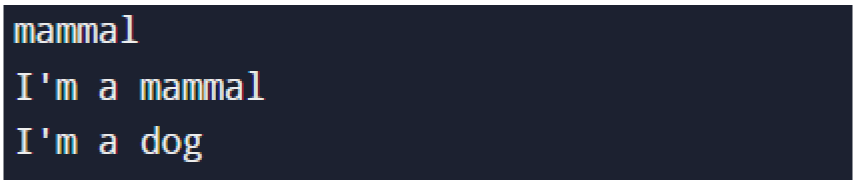
STEP1: Define a class using the class keyword and a class name.
STEP 2: Define the init method, which is called when the object is created and is used to initialize the object's attributes.
STEP 3: Define any additional methods that the class should have. These are functions that are defined within the class and can be called on the object.
STEP 4: Create an object of the class by calling the class name as if it were a function and passing in the required arguments.
STEP 5: Access the object's attributes using dot notation (e.g., object.attribute) and call the object's methods using the same dot notation (e.g., object.method()).

**Implementation:**

```
class Dog:
# A simple class attribute
attr1 = "mammal"
attr2 = "dog"
def fun(self):
print("I'm a", self.attr1)
print("I'm a", self.attr2)
```

```
# Object instantiation
Rodger = Dog()
print(Rodger.attr1)
Rodger.fun()
```

**Output:**



```
mammal
I'm a mammal
I'm a dog
```

**B) Aim:** Write a python program to create an employee class with some attributes and methods

**Tools/Software Required:** VS Code, Python

**Description:**
Class creates a user-defined data structure, which holds its own data members and member functions, which can be accessed and used by creating an instance of that class. A class is like a blueprint for an object. An Object is an instance of a Class. A class is like a blueprint while an instance is a copy of the class with actual values.

**Steps:**

STEP1: Define a class using the class keyword and a class name.
STEP 2: Define the init method, which is called when the object is created and is used to initialize the object's attributes.
STEP 3: Define any additional methods that the class should have. These are functions that are defined within the class and can be called on the object.

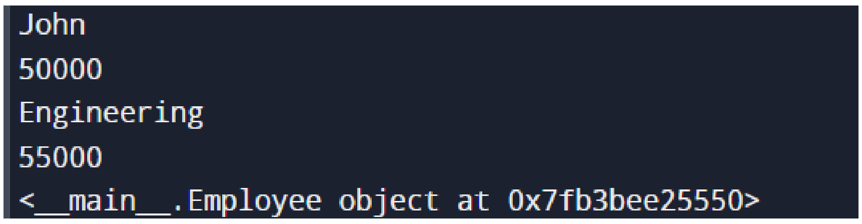
STEP 4: Create an object of the class by calling the class name as if it were a function and passing in the required arguments.
STEP 5: Access the object's attributes using dot notation (e.g., object.attribute) and call the object's methods using the same dot notation (e.g., object.method()).

## Implementation:

```
class Employee:
    def __init__ (self, name, salary, department):
        self.name = name
        self.salary = salary
        self.department = department
    def raise_salary(self, amount):
        self.salary += amount
    def str (self):
        return f"Employee: {self.name}, Salary: {self.salary},
Department:{self.department}"

employee = Employee("John", 50000, "Engineering")
print(employee.name)
print(employee.salary)
print(employee.department)
employee.raise_salary(5000)
print(employee.salary)
print(employee)
```

## Output:



```
John
50000
Engineering
55000
<__main__.Employee object at 0x7fb3bee25550>
```

**C) Aim:** Write a python program to implement operator overloading.
**Tools/Software Required:** VS Code, Python

**Description:**
You can overload operators by defining special methods in your class. These special methods have a double underscore prefix and suffix (e.g., add for the '+' operator) and are called when the corresponding operator is used on objects of the class.

**Steps:**
1. Define a custom class that you want to overload the operator for.
2. Inside the class definition, define a special method with a double underscore prefix and suffix that corresponds to the operator you want to overload. For example, to overload the + operator, you would define the add method.
3. Inside the special method, define the behavior of the operator when used with instances of your class. You can access the operands using the self and other arguments, which represent the left and right operands of the operator, respectively.
4. Use the operator with instances of your class to test the overloading.

**Implementation:**
```
class Vector:
    def __init__(self, x, y):
        self.x = x
        self.y = y
    def __add__(self, other):
        return Vector(self.x + other.x, self.y + other.y)
    def __str__(self):
        return "Vector ({}, {})".format(self.x, self.y)
v1 = Vector(2, 10)
v2 = Vector(5, -2)
print(v1 + v2)
```

**Output:** (4, 6)

**CourseName**: Advance Python Programming **Course code:** 22CSH-623

**D) Aim:** Write a python program to implement method overriding

**Tools/Software Required:** VS Code, Python

## Description:

Method overriding is a technique in object-oriented programming that allows a subclass to provide its own implementation for a method that is inherited from a superclass. This allows the subclass to modify or extend the behavior of the method in a way that is specific to that subclass.

## Steps:

1. Define a superclass with a method that you want to override.
2. Define a subclass that inherits from the superclass.
3. In the subclass, define a method with the same name as the one in the superclass. This will override the method in the superclass.
4. Define the behavior of the overridden method in the subclass. You can use the super() function to call the method from the superclass if you want to extend its behavior rather than replacing it completely.

## Implementation:

```
class Animal:
    def make_sound(self):
        print("Some generic animal sound")
class Dog(Animal):
    def make_sound(self):
        print("Bark")
class Cat(Animal):
    def make_sound(self):
        print("Meow")
dog = Dog()
dog.make_sound()
cat = Cat()
cat.make_sound()
```

**Output:**

```
Bark

Meow
```