Module 4:

Object Oriented Programming

By Ninad Gaikwad

Reference - “Core Python Programming”
Dr. R. Nageshwara Rao
Dreamtech Press

Introduction to OOPS

- At a particular point, the programmers start losing the control of the code
- Programming in this approach is not developed from human being’s life
- Features of Object Oriented Programming System (OOPS)
    - Classes and Objects
    - Encapsulation
    - Abstraction
    - Inheritance
    - Polymorphism

13.1 Creating a class

- Class contains attributes and methods.
- __init__ is a special method used to initialize the attributes
- ‘self’ is a default variable that stores the memory location of the created instance
- Object represents the base class name from which all the classes in python are derived. Writing “object” in parentheses is optional.:
- Eg.

```
class Student:            # another way is: class Student(Object)
        #Below block defines attributes
        def __init__(self):
                self.name = “Vishnu”
                self.age = 20
                self.marks = 900
        # below block defines a method
        def talk(self):
                print(“Hi i am ”, self.name)
                print(“My age is ”, self.age)
                print(“My marks are ”, self.marks)
# Create an instance of student class
s1 = Student()
# Call the method using the instance
s1.talk()
```

13.2 Constructor

- Special method used to initialize the instance variables of a class
- First parameter of the constructor will be self variable
- To pass values to constructor we have to pass them after the constructor name
- Eg.

```
class Student:  # another way is: class Student(Object)
    # This is a constructor
    def __init__(self, n = “” , m = 0):
        self.name = n
        self.marks = m
    # This is an instance method
    def display(self):
        print(“Hi ”, self.name)
        print(“Your marks are ”, self.marks)
# Constructor is called without any arguments
s1 = Student()
s1.display()
print(“----------------------------”)
# Constructor is called with 2 arguments
s1 = Student(“Laxmi Roy”, 880)
s1.display()
print(“----------------------------”)
```

13.3 Types of Variables

- Instance Variables:
  - Variables whose separate copy is created in every instance
  - Eg.

    ```
    class Student:
              def __init__(self):
                        self.name = “Vishnu” # Instance variables
    ```
- Class Variable
  - Whose single copy is available for all the instances of the class
  - Eg.

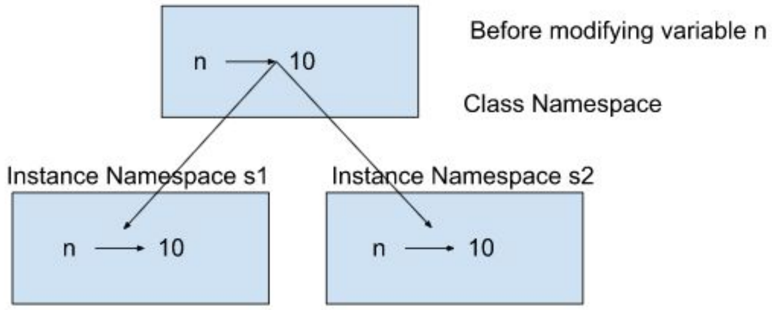
    ```
    class Student:
              x=10 # Class variable
              def __init__(self):
                        self.name = “Vishnu” # Instance variables
    ```

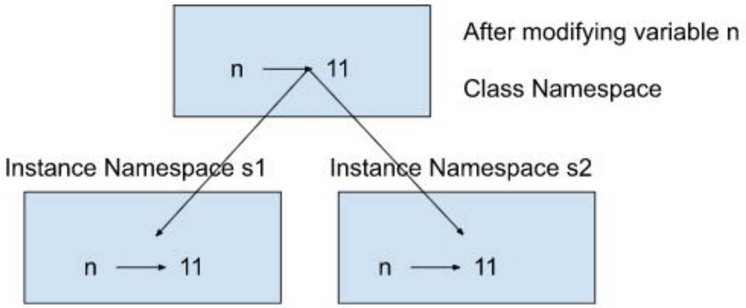
13.4 Namespaces

- Represents a memory block where names are mapped to objects
- Changes in class namespace does not affect the instance namespace
- Eg.

```
# Example of Class namespace
class Student:
        x=10 # Class variable
print(Student.n)  # displays 10
Student.n+=1  # modify it in class namespace
print(Student.n)  # displays 11

# modified class variable n is seen in all instances
s1=Student()  # Create s1 instance
print(s1.n) # Displays 11
s2=Student()  # Create s2 instance
print(s2.n) # Displays 11
```





13.5 Types of Methods

- Instance methods
    - Methods which act upon the instance variables of a class

      Eg.

      ```
      class Student:
          # this is a constructor
          def __init__(self, n=“”, m=0):
              self.name = n
              self.marks = m
          # this is an instance method
          def display(self):
              print(‘Hi ’, self.name)
              print(“Your marks ”, self.marks)
      ```

    - Accessor methods
        - Simply access or read the data of the variables. They do not modify the data in the variables.
        - Generally written as getXXX() hence also called as getter methods
    - Mutator methods
        - Methods which not only read but also modify the variables
        - They are written in the form setXXX() hence are also called as setter methods

Accessor and Mutator methods

Eg.

```
class Student:
    # Mutator method
    def setName(self, n):
        self.name = n
    # Mutator method
    def setMarks(self, m):
        self.marks = m
    # Accessor method
    def getName(self):
        return self.name
    # Accessor method
    def getMarks(self):
        return self.marks
# create student class instance
s=Student()
s.setName(input(“Enter your name”))
s.setMarks(int(input(“Enter your marks”)))

# Retrieve data of student class
print(“HI ”, s.getName())
print(“Your marks are ”, s.getMarks())
```

Class and Static Methods

- Class methods
    - Class methods act on class variables
    - They are written using @classmethod decorator
    - The first parameter of the class method is ‘cls’ which refers to the class itself
    - “cls.var” is the format to refer to class variables
    - Methods are generally called using “classname.method()”
    - Eg.

      ```
      class Bird:
                # This is a class var
                wings = 2

                # This is a class method
                @Classmethod
                def fly(cls, name):
                          print(‘{} flies with {} wings’ .format(name, cls.wings))

      # Display information for 2 birds
      Bird.fly(‘Sparrow’)
      Bird.fly(‘Pigeon’)
      ```
- Static methods
    - Static methods cannot modify class state
    - Static methods do not take any specific parameter
    - These methods are used to do some utility tasks by taking some parameters.

Experiment 7:

Problem Statement:

Design an Employee class using Python for reading, modifying and displaying the employee information.