**Experiment No. 6A**

**Title: Program on OOP in Python: Class, Self Variables, Types of methods, Constructor**

**Batch: B1 Roll No: 1914078 Experiment No.:6**

Aim: **Program on OOP in Python: Class, Self Variables, Types of methods, Constructor**

**Resources needed:** Python IDE

### Theory:

Python is an object oriented programming language. Almost everything in Python is an object, with its properties and methods .A Class is like an object constructor, or a "blueprint" for creating objects. Objects are an encapsulation of variables and functions into a single entity. Objects get their variables and functions from classes. Classes are essentially a template to create your objects.

**Example :**

class MyClass:

variable = "hello"

def function(self):

print("This is a message inside the class.")

myobjectx = MyClass()

### The self parameter is a reference to the current instance of the class, and is used to access variables that belong to the class. It does not have to be named self you can call it whatever you like, but it has to be the first parameter of any function in the class:

### There are three types of methods in Python: instance methods, static methods, and class methods.

### Instance methods:

### Instance methods are the most common type of methods in Python classes. These are so called because they can access unique data of their instance. Instance methods must have self as a parameter.Inside any instance method, you can use self to access any data or methods that may reside in your class. You won’t be able to access them without going through self.

### Static methods:

### Static methods are methods that are related to a class in some way, but don’t need to access any class-specific data. You don’t have to use self, and you don’t even need to instantiate an instance

**Class methods: They** can’t access specific instance data, but they can call other static methods. Class methods don’t need **self** as an argument, but they do need a parameter called **cls**. This stands for **class**, and like self, gets automatically passed in by Python. Class methods are created using the **@classmethod** decorator.

### Example:

class MyClass:

def method(self):

return 'instance method called', self

@classmethod

def classmethod(cls):

return 'class method called', cls

@staticmethod

def staticmethod():

return 'static method called

# Constructors in Python

Constructors are generally used for instantiating an object. The task of constructors is to initialize (assign values) to the data members of the class when an object of class is created. In Python the \_\_init\_\_() method is called the constructor and is always called when an object is created.

**Syntax of constructor declaration:**

def \_\_init\_\_(self):

# body of the constructor

**Types of constructors:**

* Default constructor: The default constructor is simple constructor which doesn’t accept any arguments. It’s definition has only one argument which is a reference to the instance being constructed.
* Parameterized constructor: constructor with parameters is known as parameterized constructor. The parameterized constructor take its first argument as a reference to the instance being constructed known as self and the rest of the arguments are provided by the programmer.

### Activities:

* Write a program to create a class Student with data members Roll Number, Marks obtained for 5 subjects and Total Marks and initialize it. Calculate the percentage scored by the student and display the roll number, marks obtained, total marks and the percentage scored by the student(create at least 3 instances of class).

### Result:

class Student:

    rollnum = 0

    marks = []

    tot = 0

    def \_\_init\_\_(self, rollnum, marks, tot):

        self.rollnum = rollnum

        self.marks = marks

        self.tot = tot

    def display(self):

        print("Roll number: "+str(self.rollnum))

        print("Marks obtained : ", end="")

        for i in self.marks:

            print(i, end=" ")

        print("\nTotal marks : "+str(self.tot))

        print("Percentage = "+str(sum(self.marks)\*100/self.tot)+"\n")

std1 = Student(1914078, [10, 20, 30, 40, 50], 250)

std2 = Student(1914092, [15, 25, 35, 45, 50], 250)

std3 = Student(1914094, [11, 21, 31, 41, 50], 250)

std1.display()

std2.display()

std3.display()

### Output:

### 

### Outcomes: Apply Object Oriented Programming Concepts in Python

### Questions:

### a)What is the use of getter and setter methods?

In Python, getters and setters are not the same as those in other object-oriented programming languages. Basically, the main purpose of using getters and setters in object-oriented programs is to ensure data encapsulation.

We use getters & setters to add validation logic around getting and setting a value and to avoid direct access of a class field i.e. private variables cannot be accessed directly or modified by external user.

**Conclusion:** We implemented program on OOP in Python: Class, Self Variables, Types of methods, Constructor

**References:**

* 1. **Reema Thareja , “Python Programming: Using Problem Solving Approach”, Oxford University Press, First Edition 2017, India**
  2. **Sheetal Taneja and Naveen Kumar,” Python Programing: A Modular Approach”, Pearson India, Second Edition 2018, India**