Batch: B2	**Roll No.: 16010121110**
Experiment / assignment / tutorial No.	
Grade: AA / AB / BB / BC / CC / CD /DD	
Signature of the Staff In-charge with date	

TITLE: Class, Object , Types of methods and Constructor

AIM: Write a program to create StudentInfo class .Calculate the percentage scored by the student

Expected OUTCOME of Experiment: Apply Object oriented programming concepts in Python

Resource Needed: Python IDE

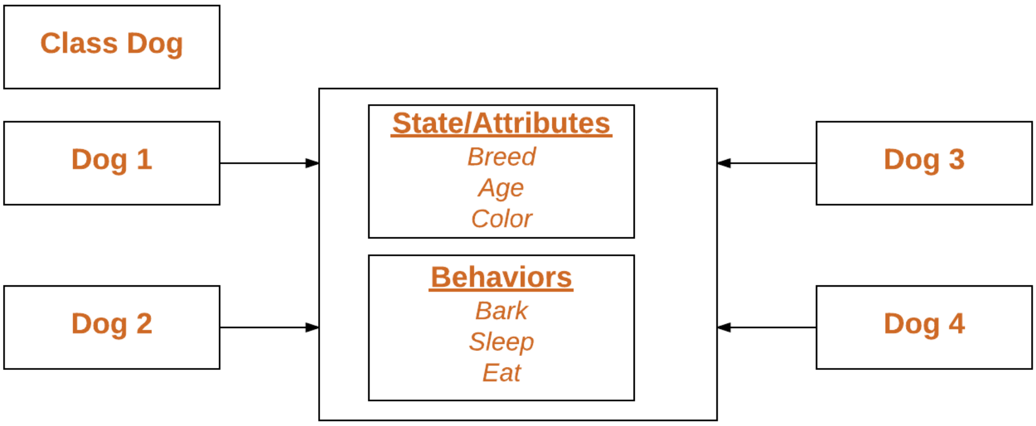
Theory:

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.



Public Members of a class (data and methods) are accessible from outside the class. Private members are inaccessible from outside the class. Private members by convention start with an underscore, as _name, _age, _salary.

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.

Python built-in function

The built-in functions defined in the class are described in the following table.

SN	Function	Description
1	getattr(obj,name,default)	It is used to access the attribute of the object.
2	setattr(obj, name,value)	It is used to set a particular value to the specific attribute of an object.
3	delattr(obj, name)	It is used to delete a specific attribute.
4	hasattr(obj, name)	It returns true if the object contains some specific attribute.

Problem Definition:

1. For given program find output

Sr.No	Program	Output

1	class MyClass:
	  x = 5
	
	p1 = MyClass()
	print(p1.x)
		5

2	class Person:
	  def __init__(self, name, age):
	    self.name = name
	    self.age = age
	
	p1 = Person("John", 36)
	
	print(p1.name)
	print(p1.age)
		John
		36

3	class Student:
	    # Constructor - non parameterized
	    def __init__(self):
	            print("This is non parametrized constructor")
	    def show(self,name):
	        print("Hello",name)
	student = Student()
	student.show("John")
		This is non parametrized constructor
		Hello John

4	class Student:
	    roll_num = 101
	    name = "Joseph"
	
	    def display(self):
	        print(self.roll_num,self.name)
	
	st = Student()
	st.display()
		101 Joseph

5	class Student:
	    # Constructor - parameterized
	    def __init__(self, name):
	        print("This is parametrized constructor")
	        self.name = name
	    def show(self):
	        print("Hello",self.name)
	student = Student("John")
	student.show()
		This is parametrized constructor
		Hello John

2. Write a program to accept Roll Number, Marks Obtained in four subjects, calculate total Marks and percentage scored by the student. Display the roll number, marks obtained, total marks and the percentage scored by the student. Use getter-setter methods.

Books/ Journals/ Websites referred:

1. Reema Thareja, *Python Programming: Using Problem Solving Approach*, Oxford University Press, First Edition 2017, India
2. Sheetal Taneja and Naveen Kumar, *Python Programming: A modular Approach*, Pearson India, Second Edition 2018,India
3. https://github.com/Aatmaj-Zephyr/Learning-Python/tree/main/Basic
4. https://github.com/Aatmaj-Zephyr/Learning-Python/tree/main/Intermediate

---

Implementation details:

```
class student:
    def __init__(self,name,Rollno):
        self.name=name
        self.Rollno=Rollno
    def setPhysicsMarks(self,marks):
        self.physicsmarks=marks

    def setMathsMarks(self,marks):
        self.Mathsmarks=marks

    def setEEMarks(self,marks):
        self.EEmarks = marks

    def setMechMarks(self,marks):
        self.MechMarks = marks

    def getPhysicsMarks(self):
        return self.physicsmarks

    def getMathsMarks(self):
        return self.Mathsmarks

    def getEEMarks(self):
        return self.EEmarks

    def getMechMarks(self):
        return self.MechMarks

    def total(self):
        return sum([self.physicsmarks,self.MechMarks,self.Mathsmarks,self.EEmarks])

    def percentge(self):
        return self.total()/4*100

Aditya=student("Aditya",121)
Aditya.setMechMarks(18)
Aditya.setEEMarks(10)
Aditya.setMathsMarks(25)
Aditya.setPhysicsMarks(27)
```

```
print(int(Aditya.getMechMarks()))
print(int(Aditya.getEEMarks()))
print(int(Aditya.getMathsMarks()))
print(int(Aditya.getPhysicsMarks()))
print(int(Aditya.total()))
print(int(Aditya.percentge()))
```

Output(s):

```
18
10
25
27
80
2000
```

Conclusion:

Thus we have successfully understood OOP in python. We understood how classes and methods work. Classes are collection of objects. We can write elegant code using OOP. This code is flexible and can be changed easily as per the changing requirements.

Post Lab Questions:

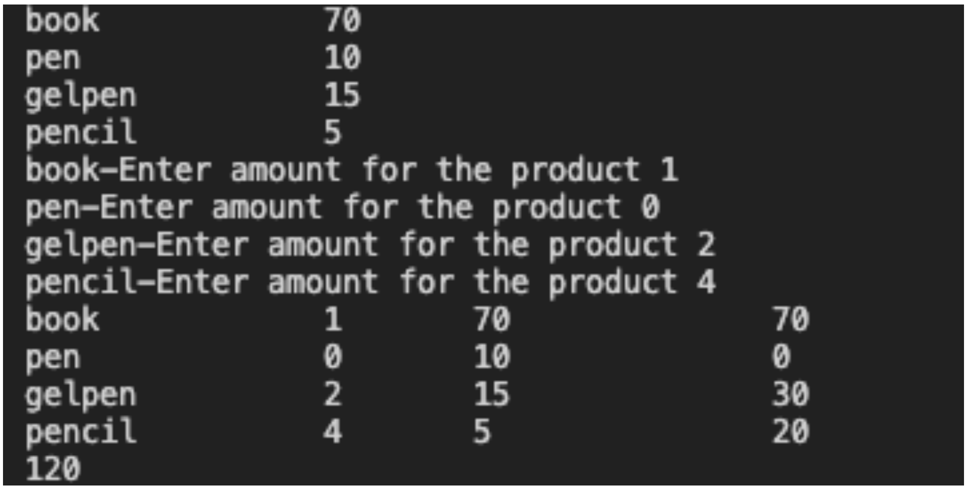
1. Write a program that has a class ‘store’ which keeps a record of code and price of each product. Display a menu of all products to the user and prompt them to enter the quantity of each item required. Generate a bill and display the total amount.

```
class store():
    products={'book':70,'pen':10,'gelpen':15,'pencil':5}
    #dictionary to store the product and prices
    cart= dict.fromkeys(products, 0) # make dictioanry with zero values

    def display(self):
        for key in self.products:
            print(key,end="\t\t")
            print(self.products[key])
    def prompt(self):
        for key in self.cart:
            print(key,end="-")
            self.cart[key]=input("Enter amount for the product ")

    def bill(self):
        net=0
        for key in self.products:
          print(key,end="\t\t")
          print(self.cart[key],end="\t")
          print(self.products[key],end="\t\t")
print(int(self.cart[key])*int(self.products[key])) #error 
handling not done for integer casting
          net+=int(self.cart[key])*int(self.products[key])
        print(net)
```

```
mystore=store()
mystore.display()
mystore.prompt()
mystore.bill()
```



```
book            70
pen             10
gelpen          15
pencil          5
book-Enter amount for the product 1
pen-Enter amount for the product 0
gelpen-Enter amount for the product 2
pencil-Enter amount for the product 4
book            1       70              70
pen             0       10              0
gelpen          2       15              30
pencil          4       5               20
120
```

2. What is the use of getter and setter methods?
   Getter and setter methods enable the access of class parameters from outside the class. Class methods must be made private and getter and setter methods must be used to access them. This improves code quality.

Date: 25 June 2022 **Signature of faculty in-charge**