**Batch: B2 Roll No.: 121**

**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:

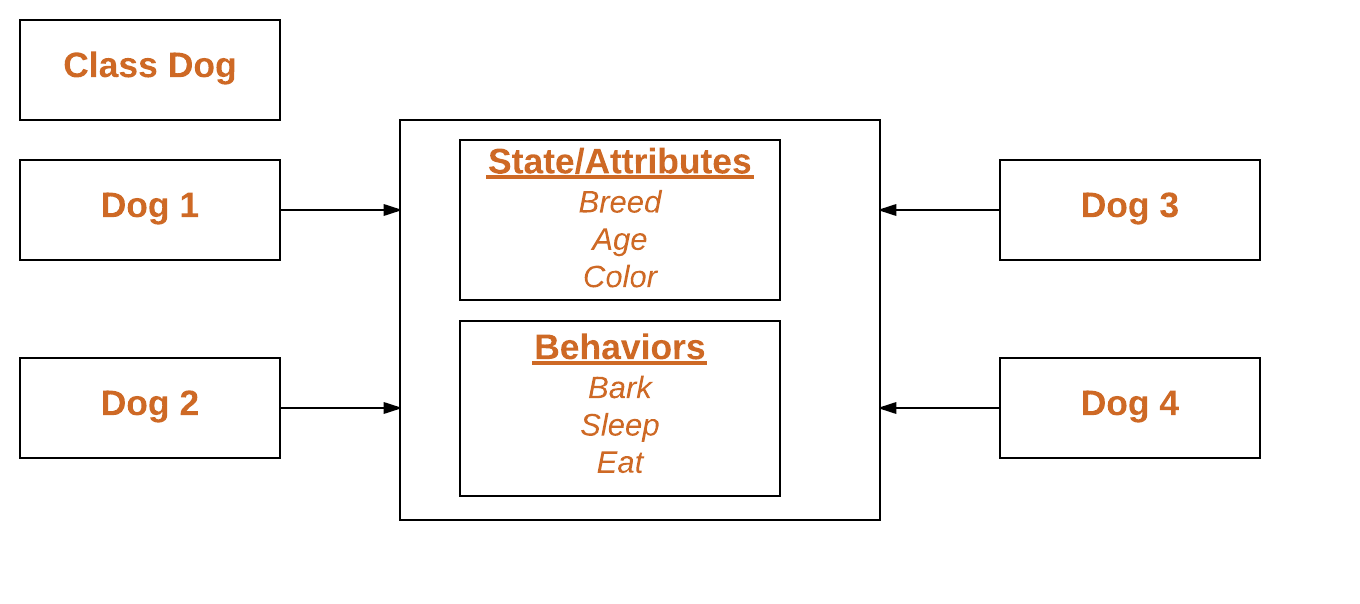
variable = "hello"

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-parameterized 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 parameterized 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

**Implementation details:**

class marks:

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

self.\_roll\_num = 0

self.\_marks1 = 0

self.\_marks2 = 0

self.\_marks3 = 0

self.\_marks4 = 0

self.\_totalmarks = 0

self.\_percentage = 0

#getter methods

def get\_roll\_num(self):

return self.\_roll\_num

def get\_marks1(self):

return self.\_marks1

def get\_marks2(self):

return self.\_marks2

def get\_marks3(self):

return self.\_marks3

def get\_marks4(self):

return self.\_marks4

def get\_totalmarks(self):

return self.\_totalmarks

def get\_percentage(self):

return self.\_percentage

#setter methods

def set\_roll\_num(self, a):

self.\_roll\_num = a

def set\_marks1(self, b):

self.\_marks1 = b

def set\_marks2(self, c):

self.\_marks2 = c

def set\_marks3(self, d):

self.\_marks3 = d

def set\_marks4(self, e):

self.\_marks4 = e

def set\_totalmarks(self):

self.\_totalmarks = int(self.\_marks1 )+int( self.\_marks2) + int(self.\_marks3 )+ int(self.\_marks4)

def set\_percentage(self):

self.\_percentage = int(self.get\_totalmarks())/4

#printing

student = marks()

student.set\_roll\_num((input("Enter your Roll No.: ")))

student.set\_marks1(int(input("Enter marks for subject 1: ")))

student.set\_marks2(int(input("Enter marks for subject 2: ")))

student.set\_marks3(int(input("Enter marks for subject 3: ")))

student.set\_marks4(int(input("Enter marks for subject 4: ")))

student.set\_totalmarks()

student.set\_percentage()

print("The roll no. is ", student.get\_roll\_num())

print("The marks in subject 1 is ", student.get\_marks1())

print("The marks in subject 2 is ", student.get\_marks2())

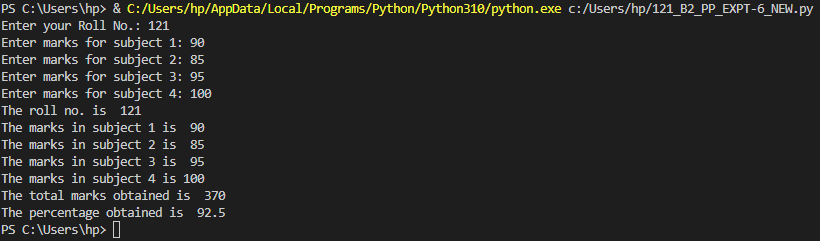
print("The marks in subject 3 is ", student.get\_marks3())

print("The marks in subject 4 is", student.get\_marks4())

print("The total marks obtained is ", student.get\_totalmarks())

print("The percentage obtained is ", student.get\_percentage())

**Output(s):**



**Conclusion:**

Thus, in this experiment, we have learnt about Class, Object, Types of methods and Constructors in Python Programming. The syntax of constructors in Python is different from other programming languages, but in a positive manner as it makes coding more efficient. Further, the concept of Object Orinted Programming in Python was applied. The class was understood to be an object factory, while the objects are groups of methods and functions. We saw the use of two methods – getter and setter. They are used to ensure data encapsulation. Again, the presence of these features in Python makes it a highly efficient high-level programming language. When expertly used, they can work wonders in coding.

**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.

**Ans.**

1. What is the use of getter and setter methods?

**Ans.** The primary use of Getters and Setters in Object Oriented Programs is to ensure data encapsulation. They are used to retrieve and update data. A getter retrieves an object’s current attribute value, whereas a setter changes an object’s current attribute value.

Getters and Setters in Python are often used:

1. to add validation logic around getting and setting a value.
2. To avoid direct access of a class field i.e., private variables cannot be accessed directly or modified by an external user.

**Date: \_\_26-06-2022\_\_ Signature of faculty in-charge**