Batch: P5-3 Roll No.: 16010422185
Experiment / assignment / tutorial No. 6
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

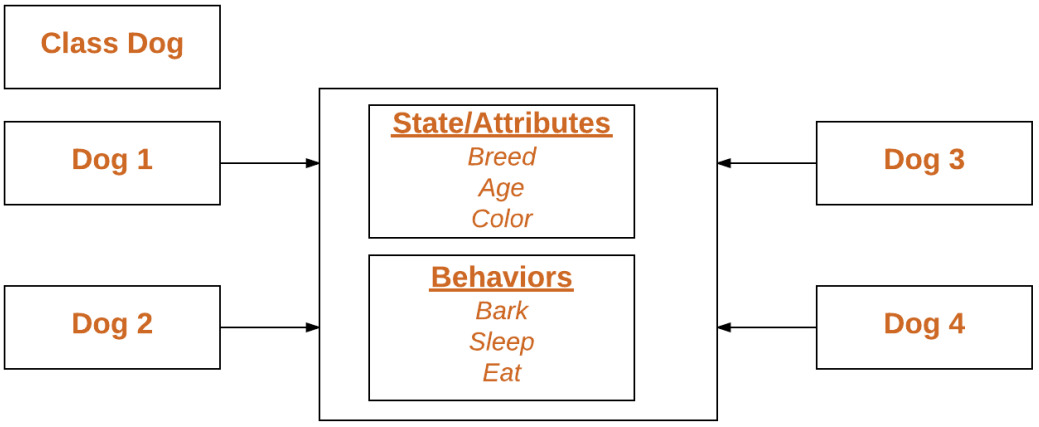
Theory:

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	5

	p1 = MyClass() print(p1.x)	

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 Joesph

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

---

Implementation details:

Code :

```
class student:
  total_Marks = 400
  def __init__(self,roll_no,m1,m2,m3,m4):
    self._roll_no=roll_no
    self._m1=m1
    self._m2=m2
    self._m3=m3
    self._m4=m4
  def get_m1(self):
    return self.m1
  def set_m1(self,num):
    return self.num
  def get_rn(self):
    return self._roll_no
  def get_tm(self):
    return self._m1+self._m2+self._m3+self._m4
  def get_p(self):
    return(self._m1+self._m2+self._m3+self._m4)*100/self.total_Marks

s1=student(25,80,56,78,98)
print(s1.get_rn())
print(s1.get_tm())
print(s1.get_p())
```

Output(s):

```
25
312
78.0
```

Conclusion:
In this experiment we learned the concepts of object oriented programming in python . We also learned how to define a constructor in python . We also learned how to access a class variable and the use of set and get methods in python .

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 - CODE :

```
class Store:
    def __init__(self):
        self.products = {
            '001': {'name': 'Product 1', 'price': 10},
            '002': {'name': 'Product 2', 'price': 20},
            '003': {'name': 'Product 3', 'price': 30},
            '004': {'name': 'Product 4', 'price': 40},
            '005': {'name': 'Product 5', 'price': 50},
        }

    def display_products(self):
        print("Product Code \t Product Name \t Price")
        for code, product in self.products.items():
            print(f"{code} \t {product['name']} \t {product['price']}")

    def generate_bill(self, products_quantity):
        total_amount = 0
        print("\n\nProduct Code \t Product Name \t Quantity \t Price \t Amount")
        for code, quantity in products_quantity.items():
            product = self.products[code]
            amount = product['price'] * quantity
            total_amount += amount
                print(f"{code} \t {product['name']} \t {quantity} \t\t {product['price']} \t
{amount}")
        print(f"\nTotal Amount: {total_amount}")


store = Store()
store.display_products()

products_quantity = {}
while True:
    code = input("\nEnter the product code (press 'q' to quit): ")
    if code == 'q':
        break
    if code not in store.products:
        print("Invalid product code!")
        continue
    quantity = int(input("Enter the quantity: "))
    products_quantity[code] = quantity

store.generate_bill(products_quantity)
```

OUTPUT :

```
Product Code     Product Name    Price
001      Product 1       10
002      Product 2       20
003      Product 3       30
004      Product 4       40
005      Product 5       50

Enter the product code (press 'q' to quit): 005
Enter the quantity: 5

Enter the product code (press 'q' to quit): 003
Enter the quantity: 9

Enter the product code (press 'q' to quit): 004
Enter the quantity: 4

Enter the product code (press 'q' to quit): q


Product Code     Product Name    Quantity        Price
005      Product 5       5               50      250
003      Product 3       9               30      270
004      Product 4       4               40      160

Total Amount: 680
```

2. Explain the concept of Method Resolution order MRO.

Ans -

Method Resolution Order (MRO) is the order in which a programming language resolves method and attribute lookups in a class hierarchy. In Python, MRO is determined by the C3 algorithm, which is a linearization algorithm that ensures that all base classes are searched in the order specified by depth-first search, from left to right.
The MRO is important when dealing with classes that inherit from multiple base classes. When a method is called on an object that has multiple base classes, Python will search for the method in each of the classes in the MRO order. This ensures that the correct implementation of the method is used based on the inheritance hierarchy.

MRO is an important concept in object-oriented programming that helps ensure that method and attribute lookups are resolved correctly in class hierarchies with multiple base classes.

Date: **Signature of faculty in-charge**