Batch: P1 -2	**Roll No.: 16014022050**
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

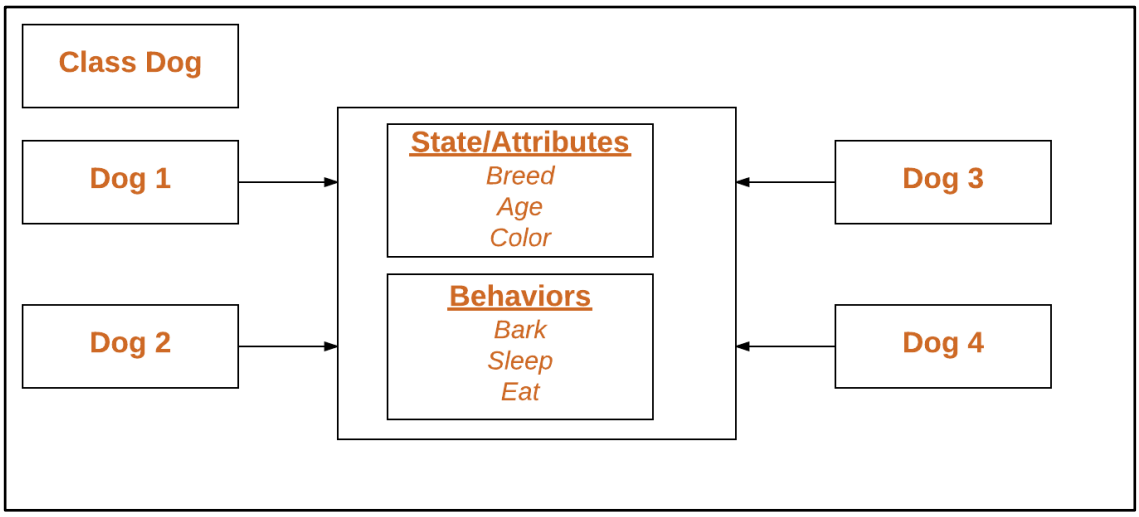
Theory:

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.

1. **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.

2. **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

3. **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:

1. **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.
2. **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 functions:

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

---

Implementation details:

2.

```
class student: # defines class student
    def __init__(name, self, roll, mark1, mark2, mark3, mark4): #
constructor method takes 7 arguments
        self.roll = roll
        self.name = name
        self.m2 = mark1
        self.m1 = mark2
        self.m3 = mark3
        self.m4 = mark4

name = input("enter your full name: ") # accept input from user
roll = int(input("enter the roll number: "))
mark1 = int(input("enter marks in subject 1: "))
mark2 = int(input("enter marks in subject 2: "))
mark3 = int(input("enter marks in subject 3: "))
mark4 = int(input("enter marks in subject 4: "))

stu1 = student # creating object of class "student"

setattr(stu1, 'name', name) # setting values of instance variables using
setattr() method
setattr(stu1, 'roll', roll)
setattr(stu1, 'm1', mark1)
setattr(stu1, 'm2', mark2)
setattr(stu1, 'm3', mark3)
setattr(stu1, 'm4', mark4)

print("\nMARK LIST") # displaying marks list
print("name: ", getattr(stu1, 'name'))
```

```
print("roll Number: ", getattr(stu1, 'roll'))
print("marks in subject 1: ", getattr(stu1, 'm1'))
print("marks in subject 2: ", getattr(stu1, 'm2'))
print("marks in subject 3: ", getattr(stu1, 'm3'))
print("marks in subject 4: ", getattr(stu1, 'm4'))

totalmarks = 0

for i in range(1,5):
    totalmarks = totalmarks + getattr(stu1, 'm' + str(i)) # calculates
total marks obtained by student
    i = i + 1

print("total marks scored by the student is: ", totalmarks)

percentage = (totalmarks / 400) * 100

print("Percentage scored by the student: ", percentage ,"%")
```

Output(s):

```
PS C:\Users\Ketaki Mahajan\OneDrive\Desktop\python\pp> & "C:/Users/Ketaki Mahajan
/AppData/Local/Microsoft/WindowsApps/python3.11.exe" "c:/Users/Ketaki Mahajan/One
Drive/Desktop/python/pp/exp6_q2.py"
enter your full name: ketaki mahajan
enter the roll number: 50
enter marks in subject 1: 75
enter marks in subject 2: 78
enter marks in subject 3: 95
enter marks in subject 4: 86

MARK LIST
name:  ketaki mahajan
roll Number:  50
marks in subject 1:  75
marks in subject 2:  78
marks in subject 3:  95
marks in subject 4:  86
total marks scored by the student is:  334
Percentage scored by the student:  83.5 %
PS C:\Users\Ketaki Mahajan\OneDrive\Desktop\python\pp>
```

Conclusion:

Through this experiment, we have understand the use of classes, objects and methods to make user defined datatypes to create a blueprint for the objects.

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:
    def __init__(self,code,price):
        self.code = code
        self.price = price

st1 = []

st1.append(store("product1", 100))
st1.append(store("product2", 121))
st1.append(store("product3", 195))
st1.append(store("product4", 250))
st1.append(store("product5", 299))
st1.append(store("product6", 131))

print("product and price list: ")

for i in range(len(st1)):
    print(st1[i].code,"  ", st1[i].price)

dict1 = {}

while True:
    pq = input("\nenter the product code: ")
    qty = int(input("enter the quantity: "))
    dict1[pq] = qty

    x = input("do you want to continue [yes/no]: ")
    if x == "no" or x == "No":
        break
    else:
        continue

print("\nproduct and quantity purchased: ")
print(dict1)

totalPrice = 0

for x in dict1.keys():
    for i in range(len(st1)):
        if x == st1[i].code:
```

```
            totalPrice = totalPrice + (dict1[x]*st1[i].price)

print("Total Cost = ",totalPrice)
```

```
PS C:\Users\Ketaki Mahajan\OneDrive\Desktop\python\pp> & "C:/Users/Ketaki Mahajan
/AppData/Local/Microsoft/WindowsApps/python3.11.exe" "c:/Users/Ketaki Mahajan/One
Drive/Desktop/python/pp/exp6_plq1.py"
product and price list:
product1    100
product2    121
product3    195
product4    250
product5    299
product6    131

enter the product code: product2
enter the quantity: 4
do you want to continue [yes/no]: yes

enter the product code: product5
enter the quantity: 2
do you want to continue [yes/no]: yes

enter the product code: product1
enter the quantity: 4
do you want to continue [yes/no]: yes

enter the product code: product6
enter the quantity: 1
do you want to continue [yes/no]: no

product and quantity purchased:
{'product2': 4, 'product5': 2, 'product1': 4, 'product6': 1}
Total Cost =  1613
PS C:\Users\Ketaki Mahajan\OneDrive\Desktop\python\pp>
```

2. Explain the concept of Method Resolution order MRO.

Method Resolution Order (MRO) is the order in which Python searches for methods and attributes in a class hierarchy. MRO is used to determine which method or attribute will be used if there are multiple methods or attributes with the same name in a class hierarchy.

In Python, the MRO is determined using the C3 linearization algorithm, which is a variant of the topological sorting algorithm. The C3 algorithm is used to construct a linearization of the class hierarchy that satisfies three basic rules:

a. Child classes are listed before their parents.
b. If a class inherits from multiple classes, they are listed in the order specified in the inheritance declaration.
c. If there are conflicts, the first class listed in the inheritance declaration has priority.

The MRO is important because it determines the order in which methods and attributes are searched for in a class hierarchy. This means that the MRO can affect the behavior of code that relies on method or attribute resolution.

In Python, you can view the MRO for a class by calling the __mro__ attribute on the class or by using the built-in mro() method. This will return a tuple containing the classes in the MRO, in the order in which they will be searched for methods and attributes.

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