Name: Chandana Ramesh Galgali	
Batch: P6-1	**Roll No.: 16010422234**
Experiment / ~~assignment~~ / ~~tutorial~~ No. 06	
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

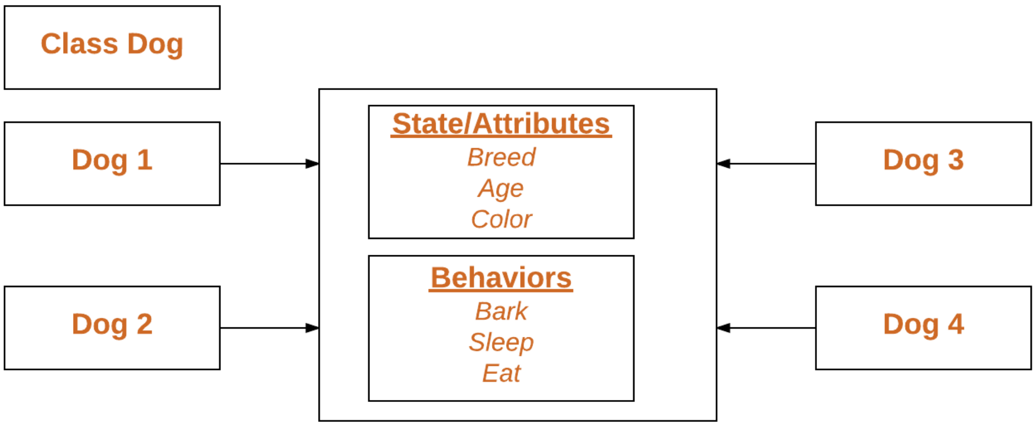
Theory:

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 a simple constructor which doesn't accept any arguments. Its 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 takes 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 a non-parameterized constructor.")
	    def show(self,name):
	        print("Hello",name)
	student = Student()
	student.show("John")
		This is a 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 a parameterized constructor.")
	        self.name = name
	    def show(self):
	        print("Hello",self.name)
	student = Student("John")
	student.show()
		This is a 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:

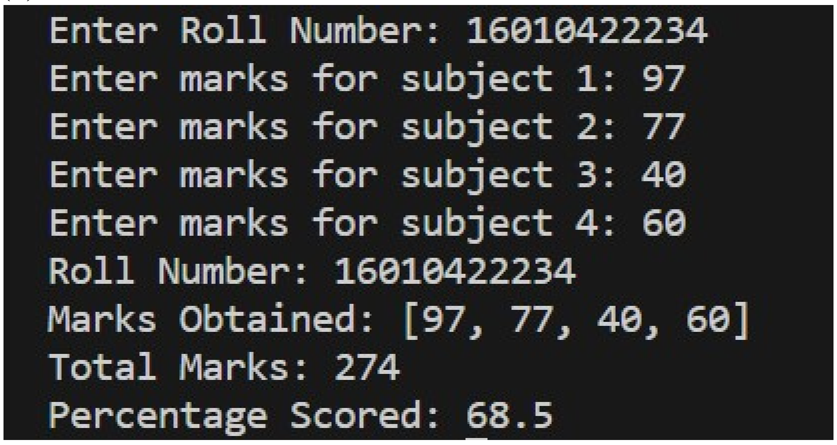
1. Reema Thareja, *Python Programming: Using Problem Solving Approach*, Oxford University Press, First Edition 2017, India
2. Sheetal Taneja and Naveen Kumar, *Python Programming: A modular Approach*, Pearson India, Second Edition 2018,India

Implementation details:

(2):
```
class Student:

    def __init__(self):
        self.__roll_number = 0
        self.__marks = [0, 0, 0, 0]
        self.__total_marks = 0
        self.__percentage = 0.0

    def set_roll_number(self, roll_number):
        self.__roll_number = roll_number
    def set_marks(self, marks):
        self.__marks = marks

    def calculate_total_marks(self):
        self.__total_marks = sum(self.__marks)
    def calculate_percentage(self):
        self.__percentage = (self.__total_marks / 400) * 100

    def get_roll_number(self):
        return self.__roll_number
    def get_marks(self):
        return self.__marks
    def get_total_marks(self):
        return self.__total_marks
    def get_percentage(self):
        return self.__percentage
    def display_student_details(self):
        print("Roll Number:", self.get_roll_number())
        print("Marks Obtained:", self.get_marks())
        print("Total Marks:", self.get_total_marks())
        print("Percentage Scored:", self.get_percentage())

# Create a Student object
student = Student()

# Accept Roll Number and Marks from the user
roll_number = int(input("Enter Roll Number: "))
marks = [int(input(f"Enter marks for subject {i + 1}: ")) for i in
range(4)]

# Set the Roll Number and Marks
student.set_roll_number(roll_number)
student.set_marks(marks)

# Calculate the Total Marks and Percentage
student.calculate_total_marks()
student.calculate_percentage()

# Display the Student Details
student.display_student_details()
```

Output(s):

(2):



```
Enter Roll Number: 16010422234
Enter marks for subject 1: 97
Enter marks for subject 2: 77
Enter marks for subject 3: 40
Enter marks for subject 4: 60
Roll Number: 16010422234
Marks Obtained: [97, 77, 40, 60]
Total Marks: 274
Percentage Scored: 68.5
```

Conclusion:

We learned how to apply Object oriented programming concepts in Python. We learned how to apply the getter and setter method.

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:

```python
class Store:
    def __init__(self):
        self.products = {}

    def add_product(self, code, price):
        self.products[code] = price

    def display_products(self):
        print("Available products:")
        for code, price in self.products.items():
            print(f"Code: {code}, Price: ${price}")

    def generate_bill(self, user_products):
        total_amount = 0
        print("\nBill:")
        for code, quantity in user_products.items():
            cost = self.products[code] * quantity
            total_amount += cost
                    print(f"Code: {code}, Quantity: {quantity}, Cost: ${cost:.2f}")

        print(f"\nTotal Amount: ${total_amount:.2f}")


def main():
    store = Store()
    store.add_product("P001", 10.0)
    store.add_product("P002", 15.0)
    store.add_product("P003", 20.0)
    store.display_products()

    user_products = {}
    while True:
        code = input("\nEnter product code (or 'done' to finish): ")
        if code.lower() == 'done':
            break

        if code not in store.products:
            print("Invalid product code. Please try again.")
            continue

        quantity = int(input("Enter quantity: "))
        user_products[code] = user_products.get(code, 0) + quantity

    store.generate_bill(user_products)


if __name__ == "__main__":
    main()
```

2. Explain the concept of Method Resolution order MRO.

Ans: Method Resolution Order (MRO) in Python is the order in which the base classes are searched when looking for a method in a class hierarchy. It is an essential aspect of Python's multiple inheritance system. MRO determines the order in which the base classes are traversed to find the appropriate method or attribute.
Python uses a linearization algorithm called C3 Linearization or C3 MRO to determine the MRO. This algorithm ensures that the MRO follows these rules:

1) A class always precedes its parents.
2) If a class inherits from multiple classes, the order of the base classes in the class definition is preserved.
3) The first two rules are applied recursively.
4) You can access the MRO of a class using the mro() method or the __mro__ attribute.

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