LAB # 17

CLASSES AND OBJECTS

OBJECTIVE

Implement classes and objects in python programming.

THEORY

Object-oriented programming is one of the most effective approaches to writing software. In object-oriented programming you can organize code into classes and objects, supports encapsulation to group data and methods together, enables inheritance for reusability and hierarchy, allows polymorphism for flexible method implementation and improves modularity, scalability, and maintainability.

Creaing a Class

A class is a collection of objects. Classes are blueprints for creating objects. A class defines a set of attributes and methods that the created objects (instances) can have.

- Classes are created by keyword class.
- Attributes are the variables that belong to a class.
- Attributes are always public and can be accessed using the dot (.) operator. Example: Myclass.Myattribute

Syntax to define a class:

```
class ClassName:
    initializer
    methods
```

Example: Creating a Class

```
class Student:                              # Created class
    def __init__(self, name, roll_no):      # Constructor method
        self.name = name                    # instance variable
        self.roll_no = roll_no              # instance variable
```

Constructing Objects

An Object is **an instance** of a Class. It represents a specific implementation of the class and holds its own data.
An object consists of:

- **State:** It is represented by the **attributes** and reflects the **properties** of an object.
- **Behavior:** It is represented by the methods of an object and reflects the response of an object to other objects.
- **Identity:** It gives a unique name to an object and enables one object to interact with other objects.

LAB # 17

Syntax for a constructor object: ClassName(arguments)

Example: Creating a Class and Object

```
class Student:
    pass

# Creating objects
s1 = Student()
s2 = Student()

print(s1)
print(s2)
```

Output

```
>>> %Run Lab_17.py
<__main__.Student object at 0x0000020978DE3B50>
<__main__.Student object at 0x0000020978DE3B80>
>>>
```

__init__Method

It is the constructor in Python, automatically called when a new object is created. It initializes the attributes of the class.

Example: Creating a Student Class and use __init__ method to set name and marks

```
class Student:
    def __init__(self, name, marks):
        self.name = name
        self.marks = marks

    def display(self):
        print("Name:", self.name)
        print("Marks:", self.marks)

s1 = Student("Ahmed", 85)
s1.display()
```

Output

```
>>> %Run Lab_17.py
Name: Ahmed
Marks: 85
>>>
```

Class and Instance Variables

A **class variable** belongs to the class itself and is shared by all objects created from that class. This means that if you change the value of a class variable, the change is reflected across all objects. On the other hand, an **instance variable** belongs to a specific object and is unique for each object. Changing the value of an instance variable affects only that particular object and does not impact other objects of the class.

Example:

```
class Student:
    # Class variable
    school_name = "Falcon House"

    def __init__(self, name, grade):
        # Instance variables
        self.name = name
        self.grade = grade

# Create objects
student1 = Student("Kabeer", 12)
student2 = Student("Mohsin", 14)

# Access class and instance variables
print(student1.school_name)  # Class variable
print(student1.name)         # Instance variable
print(student2.name)         # Instance variable

# Modify instance variable
student1.name = "Zubair"
print(student1.name)         # Updated instance variable

# Modify class variable
Student.school_name = "Bela Academy"
print(student1.school_name)  # Updated class variable
print(student2.school_name)  # Updated class variable
```

Output

```
>>> %Run Lab_17.py
Falcon House
Kabeer
Mohsin
Zubair
Bela Academy
Bela Academy
>>>
```

LAB # 17

EXERCISE

A. Point out the errors, if any, in the following Python programs.

1. Code

```
class A():
    def __init__(self, i):
        self.i = i
def main():
    a = A()
    print(a.i)
main() # Call the main function
```

Output:

2. Code

```
class Dog:
    attr1 = "mamal"
    def fun(self):
        print("I'm a", self.attr1)
```

Output:

B. Execute it and show the output

1. Code

```
class Car():
    def __init__(self, make, model, year):
        self.make = make
        self.model = model
        self.year = year
    def get_descriptive_name(self):
        long_name = str(self.year) + ' ' + self.make + ' ' + self.model
        return long_name.title()
my_new_car = Car('audi', 'a4', 2016)
print(my_new_car.get_descriptive_name())
```

Output:

LAB # 17

2. Code

```
class Person:
    # init method or constructor
    def __init__(self, name):
        self.name = name

    def say_hi(self):
        print('Hello, my name is', self.name)

p = Person('XYZ')
p.say_hi()
```

Output:

C. Write Python programs for the following:

1. Write a program that create a class called Restaurant. The __init__() method for Restaurant should store two attributes: a restaurant_name and a cuisine_type. Make a method called infor_restaurant() that prints these two pieces of information, and a method called open_restaurant() that prints a message indicating that the “restaurant is open”.