# assignment=6

July 23, 2023

## 1 Q1. Explain Class and Object with respect to Object-Oriented Programming. Give a suitable example.

Ans: Class: In object-oriented programming, a class is a blueprint or a template for creating objects (an instance of the class). It defines a set of attributes and behaviors that an object of that class will have. A class can be considered as a blueprint for the object or objects that can be created using it.

Object: An object is an instance of a class. It is a real-world entity with attributes and behaviors that are defined by the class. An object has a state and a behavior, and it is created from a class. An object is also known as an instance of a class.

Example: Let's consider a class called "Car". A Car class might have attributes such as color, make, model, and year, and behaviors such as start, stop, and drive. An object of the Car class would be a specific car, such as a red 2022 Toyota Corolla. The red 2022 Toyota Corolla would have the attributes of color (red), make (Toyota), model (Corolla), and year (2022), and it would be able to perform the behaviors start, stop, and drive.

## 2 Q2. Name the four pillars of OOPs.

Ans: The four pillars of object-oriented programming (OOP) are:

Abstraction: Abstraction refers to the practice of exposing only the relevant information and hiding the complexity of the implementation. It allows the user to focus on what the object does rather than how it does it.

Encapsulation: Encapsulation is the mechanism of wrapping the data (variables) and functions that operate on that data within a single unit or object. This protects the data from accidental corruption and provides a secure way to access the data.

Inheritance: Inheritance is a mechanism where a new class can be derived from an existing class. The derived class inherits all the attributes and behaviors of the parent class and can also have its own attributes and behaviors.

Polymorphism: Polymorphism is the ability of an object to take on multiple forms. This is achieved in OOP by defining methods with the same name in different classes that are related by inheritance. When a method is called on an object, the correct implementation of that method is executed based on the object's actual class.

## 3 Q3. Explain why the init() function is used. Give a suitable example.

Ans: The init() function is used in object-oriented programming to initialize the attributes of an object at the time of its creation. It is a special method that is automatically called when an object is created from a class. The init() method is also known as a constructor.

The init() function allows you to set the default values for the object's attributes, perform any initializations, and allocate any required memory when the object is created. The method takes the newly created object as its first argument (usually referred to as self), and additional arguments can be passed to initialize the object's attributes.

Example: Consider a class Person that represents a person's name and age. The init() function can be used to initialize these attributes when a new object of the Person class is created.

[3]:
```python
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

person = Person("Jadavbhai", 30)
print(person.name) # outputs: "Jadavbhai"
print(person.age) # outputs: 30
```

```
Jadavbhai
30
```

## 4 Q4. Why self is used in OOPs?

Ans: In object-oriented programming (OOP), self is used to refer to the instance of the object being manipulated within a method. It is a convention in Python (and some other programming languages) to use self as the first argument in an instance method to refer to the instance of the object.

When a method is called on an object, the object is passed as the first argument to the method, and it can be accessed using the self reference. This allows the method to access and modify the object's attributes and perform other operations on the object.

For example:

[4]:
```python
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def set_name(self, name):
        self.name = name

    def set_age(self, age):
        self.age = age

person = Person("jadavbhai", 47)
print(person.name) # outputs: "Jadavbhai"
person.set_name("sonal")
print(person.name) # outputs: "sonal"
```

```
jadavbhai
sonal
```

## 5 Q5. What is inheritance? Give an example for each type of inheritance.

Ans: Inheritance is a mechanism in object-oriented programming (OOP) that allows a new class to be derived from an existing class. The derived class inherits all the attributes and behaviors of the parent class and can also have its own attributes and behaviors. Inheritance enables code reuse and makes it easier to create and maintain large, complex systems.

There are several types of inheritance, including:

Single inheritance: Single inheritance is a type of inheritance where a derived class inherits from a single base class. Example:

[10]:
```
class Animal:
    def __init__(self, species, legs):
        self.species = species
        self.legs = legs

    def make_sound(self):
        print("Some animal sound")

class Dog(Animal):
    def __init__(self, breed):
        Animal.__init__(self, "Dog", 4)
        self.breed = breed

    def make_sound(self):
        print("cute")

dog = Dog("bella")
print(dog.species) # outputs: "Dog"
print(dog.legs) # outputs: 4
dog.make_sound() # outputs: "cute"
```

```
Dog
4
cute
```

2. Multiple inheritance: Multiple inheritance is a type of inheritance where a derived class inherits from multiple base classes. Example:

[11]:
```python
class Animal:
    def __init__(self, species, legs):
        self.species = species
        self.legs = legs

    def make_sound(self):
        print("Some animal sound")

class DomesticAnimal:
    def __init__(self, domesticated):
        self.domesticated = domesticated

class Dog(Animal, DomesticAnimal):
    def __init__(self, breed):
        Animal.__init__(self, "Dog", 4)
        DomesticAnimal.__init__(self, True)
        self.breed = breed

    def make_sound(self):
        print("cute")

dog = Dog("bella")
print(dog.species) # outputs: "Dog"
print(dog.legs) # outputs: 4
print(dog.domesticated) # outputs: True
dog.make_sound() # outputs: "cute"
```

```
Dog
4
True
cute
```

3. Multi-level inheritance: Multi-level inheritance is a type of inheritance where a derived class inherits from a base class, which in turn inherits from another base class. Example:

[16]:
```python
class Animal:
    def __init__(self, species, legs):
        self.species = species
        self.legs = legs

    def make_sound(self):
        print("Some animal sound")

class Mor(Animal):
    def __init__(self, fur):
        Animal.__init__(self, "Mor", 2)
        self.fur = fur

class Dog(Mor):
    def __init__(self, breed):
        Mor.__init__(self, True)
        self.breed = breed

    def make_sound(self):
        print("cute")

dog = Dog("mor")
print(dog.species) # outputs: "Mor"
print(dog.legs) # outputs:
```

```
Mor
2
```

[ ]:

```
```