***Oops(Basic) Assignment***

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

In object-oriented programming, a class is a blueprint or template for creating objects, which are instances of the class. A class defines the properties (attributes) and behaviors (methods) that an object of that class will have.

An object, on the other hand, is an instance of a class that has its own set of values for the attributes defined by the class. In other words, an object is a specific instance of a class that has its own unique state and behavior.

For example, let's say we have a class called "Car" that defines the properties and methods that a car should have. The attributes of the class could be things like make, model, year, color, and so on. The methods of the class could be things like start\_engine(), stop\_engine(), accelerate(), and brake().

Once we have defined the Car class, we can create objects of that class. For example, we can create an object called "my\_car" that is an instance of the Car class. We can set the attributes of this object to specific values, such as make="Toyota", model="Camry", year=2021, color="red".

Now, my\_car is a unique object with its own state (the values of its attributes) and behavior (the ability to execute the methods defined by the Car class). We can call the methods of the object, such as my\_car.start\_engine() to start the engine of the car.

In summary, a class is a template for creating objects, and an object is an instance of a class with its own unique state and behavior.

Q2. Name the four pillars of OOPs.

1. Encapsulation: It is the process of enclosing the data and the methods that act upon the data within a single unit, called a class. Encapsulation helps to keep the data and methods safe from outside interference and misuse.
2. Abstraction: It is the process of hiding complex implementation details behind simpler and more manageable interfaces. Abstraction allows the user to interact with the system at a higher level of abstraction, without worrying about the internal details.
3. Inheritance: It is the process of creating new classes that are built upon existing classes. Inheritance allows the new class to inherit the properties and methods of the existing class, thus promoting code reuse and reducing redundancy.
4. Polymorphism: It is the process of allowing objects of different classes to be treated as if they were objects of the same class. Polymorphism allows us to write code that can work with objects of different types, without having to know the specific type of each object.

Q3. Explain why the \_\_init\_\_() function is used. Give a suitable example.

The \_\_init\_\_() function is a special method in Python that is used to initialize the attributes of an object when it is created. It is called a constructor because it is called automatically when a new object of a class is created. The purpose of \_\_init\_\_() is to ensure that an object is initialized with the correct values for its attributes, so that it is ready for use.

For example

class Person:

def \_\_init\_\_(self, name, age):

self.name = name

self.age = age

person1 = Person("Alice", 25)

person2 = Person("Bob", 30)

Q4. Why self is used in OOPs?

In Object-Oriented Programming (OOPs), self is a reference to the object itself. It is a special variable that is used within the methods of a class to refer to the object that the method is being called on.

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

Inheritance is a key feature of Object-Oriented Programming (OOPs) that allows a new class to be based on an existing class. Inheritance promotes code reuse and reduces redundancy by allowing the new class to inherit the attributes and methods of the existing class.

1. Single inheritance: In single inheritance, a new class is derived from a single base class. The new class inherits all the attributes and methods of the base class.

Example:

class Animal:

def speak(self):

print("Animal is speaking...")

class Dog(Animal):

pass

d = Dog()

d.speak() # Output: Animal is speaking...

1. Multiple inheritance: In multiple inheritance, a new class is derived from two or more base classes. The new class inherits all the attributes and methods of the base classes.

Example:

class A:

def method\_a(self):

print("method\_a")

class B:

def method\_b(self):

print("method\_b")

class C(A, B):

pass

c = C()

c.method\_a() # Output: method\_a

c.method\_b() # Output: method\_b

1. Hierarchical inheritance: In hierarchical inheritance, multiple sub-classes are derived from a single base class.

Example

class Vehicle:

def start(self):

print("Starting vehicle...")

class Car(Vehicle):

def drive(self):

print("Driving car...")

class Bike(Vehicle):

def ride(self):

print("Riding bike...")

c = Car()

c.start() # Output: Starting vehicle...

c.drive() # Output: Driving car...

b = Bike()

b.start() # Output: Starting vehicle...

b.ride() # Output: Riding bike...

1. Multilevel inheritance: In multilevel inheritance, a new class is derived from a base class, which is itself derived from another base class.

Example:

class A:

def method\_a(self):

print("method\_a")

class B(A):

def method\_b(self):

print("method\_b")

class C(B):

def method\_c(self):

print("method\_c")

c = C()

c.method\_a() # Output: method\_a

c.method\_b() # Output: method\_b

c.method\_c() # Output: method\_c