**1. What is a class?**

A **class** is a blueprint or template for creating objects. It defines attributes (variables) and methods (functions) that describe what the objects created from it will have and can do.

👉 Example:

class Dog:

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

self.name = name

def bark(self):

return f"{self.name} is barking!"

**2. What is an instance?**

An **instance** is a specific object created from a class. Each instance has its own data (attributes) but shares the class’s structure.

👉 Example:

dog1 = Dog("Buddy") # instance of Dog class

dog2 = Dog("Charlie") # another instance

**3. What is encapsulation?**

**Encapsulation** means bundling the data (attributes) and methods (functions) that work on that data into one unit (the class). It also allows restricting access to certain details using private/protected members.

👉 Example:

class BankAccount:

def \_\_init\_\_(self, balance):

self.\_\_balance = balance # private attribute

def deposit(self, amount):

self.\_\_balance += amount

def get\_balance(self):

return self.\_\_balance

**4. What is abstraction?**

**Abstraction** means hiding the complex implementation details and showing only the essential features to the user. It’s about focusing on **what an object does**, not **how it does it**.

👉 Example:

from abc import ABC, abstractmethod

class Shape(ABC):

@abstractmethod

def area(self):

pass

The user only needs to know area() exists, not how it is implemented.

**5. What is inheritance?**

**Inheritance** allows one class (child/subclass) to use properties and methods from another class (parent/superclass). It promotes code reuse.

👉 Example:

class Animal:

def speak(self):

return "Some sound"

class Dog(Animal):

def speak(self):

return "Woof!"

**6. What is multiple inheritance?**

**Multiple inheritance** means a class can inherit from more than one parent class.

👉 Example:

class A:

def method\_a(self):

return "A"

class B:

def method\_b(self):

return "B"

class C(A, B):

pass

c = C()

print(c.method\_a()) # from A

print(c.method\_b()) # from B

**7. What is polymorphism?**

**Polymorphism** means "many forms." It allows the same method name to behave differently depending on the object.

👉 Example:

class Cat:

def speak(self):

return "Meow"

class Dog:

def speak(self):

return "Woof"

def animal\_sound(animal):

print(animal.speak())

animal\_sound(Dog()) # Woof

animal\_sound(Cat()) # Meow

**8. What is method resolution order (MRO)?**

**MRO** is the order in which Python looks for a method or attribute in a hierarchy of classes during inheritance.

* In single inheritance → child → parent.
* In multiple inheritance → Python uses **C3 linearization** to decide the order.

👉 Example:

class A: pass

class B(A): pass

class C(B): pass

print(C.mro())

# [<class '\_\_main\_\_.C'>, <class '\_\_main\_\_.B'>, <class '\_\_main\_\_.A'>, <class 'object'>]