**Q1. What is the meaning of multiple inheritance?**

**Answer:**  
Multiple inheritance refers to a class inheriting from more than one parent class. This allows a child class to access the methods and attributes of multiple base classes. In Python, a class can inherit from multiple classes by specifying them in the class declaration.

**Example:**

class Animal:

def speak(self):

print("Animal speaks")

class Pet:

def play(self):

print("Pet plays")

class Dog(Animal, Pet): # Multiple inheritance

def bark(self):

print("Dog barks")

dog = Dog()

dog.speak() # From Animal class

dog.play() # From Pet class

dog.bark() # From Dog class

In this example, Dog inherits from both Animal and Pet classes.

**Q2. What is the concept of delegation?**

**Answer:**  
Delegation refers to a design pattern where an object relies on another object (delegate) to perform a specific task or handle a behavior. Instead of implementing functionality directly, a class delegates responsibility to another class, which provides that functionality.

**Example:**

class Engine:

def start(self):

print("Engine started")

class Car:

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

self.engine = Engine() # Delegation

def start(self):

self.engine.start() # Delegated to Engine class

car = Car()

car.start() # Output: Engine started

Here, the Car class delegates the responsibility of starting the engine to the Engine class.

**Q3. What is the concept of composition?**

**Answer:**  
Composition is a design principle where one class is composed of objects of other classes, rather than inheriting from them. This allows a class to use the functionality of another class by holding an instance of that class as an attribute. It helps with code reuse and maintaining flexibility.

**Example:**

class Engine:

def start(self):

print("Engine started")

class Car:

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

self.engine = Engine() # Composition

def start(self):

self.engine.start() # Using Engine's functionality

car = Car()

car.start() # Output: Engine started

Here, Car has a Engine as a part of it, demonstrating composition.

**Q4. What are bound methods and how do we use them?**

**Answer:**  
A bound method is a method that is associated with an instance of a class. It is "bound" to that instance, meaning it has access to the instance’s attributes and can operate on them. Bound methods are created when a method is called on an instance, as opposed to being called on the class itself.

**Example:**

class Dog:

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

self.name = name

def speak(self):

print(f"{self.name} barks")

dog = Dog("Buddy")

method = dog.speak # Bound method

method() # Output: Buddy barks

In this example, dog.speak is a bound method because it is tied to the specific dog instance.

**Q5. What is the purpose of pseudoprivate attributes?**

**Answer:**  
Pseudoprivate attributes are used to indicate that an attribute is intended for internal use within a class and should not be accessed directly from outside the class. In Python, this is achieved by prefixing the attribute name with a double underscore (\_\_). It causes name mangling, which changes the attribute name to make it harder (but not impossible) to access from outside the class.

**Example:**

class MyClass:

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

self.\_\_secret = 42 # Pseudoprivate attribute

def reveal\_secret(self):

return self.\_\_secret

obj = MyClass()

# print(obj.\_\_secret) # Raises AttributeError

print(obj.reveal\_secret()) # Output: 42

Here, \_\_secret is pseudoprivate, and you can't directly access it from outside the class (it would raise an error). However, the attribute is still accessible using name mangling (\_MyClass\_\_secret), but it’s discouraged as it’s considered internal.

**Summary of Concepts:**

| **Concept** | **Description** |
| --- | --- |
| **Multiple Inheritance** | A class inherits from more than one parent class. |
| **Delegation** | One object relies on another to handle certain behaviors. |
| **Composition** | A class is composed of instances of other classes to reuse functionality. |
| **Bound Methods** | Methods bound to instances, allowing access to instance attributes. |
| **Pseudoprivate Attributes** | Attributes prefixed with \_\_ to indicate they are for internal use, using name mangling to prevent access. |