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

Ans-Multiple inheritance is a feature in object-oriented programming languages that allows a class to inherit from more than one parent class. In other words, a class can inherit characteristics and behaviours from multiple classes.

For example, let's say we have two classes: "Vehicle" and "Animal". A "Car" class that inherits from both "Vehicle" and "Animal" classes using multiple inheritance would have access to the characteristics and behaviours of both parent classes, such as the ability to move and the ability to make sounds.

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

Ans-Delegation is a programming concept in which an object passes on a task or responsibility to another object to perform on its behalf. Instead of performing the task itself, the object delegates the task to another object, which is then responsible for carrying out the task. In delegation, there are two objects involved: the delegator (the object that delegates the task) and the delegate (the object that performs the task on behalf of the delegator). The delegator is responsible for deciding which tasks to delegate, while the delegate is responsible for carrying out those tasks.

Delegation is often used to separate concerns and promote modular design in object-oriented programming. By delegating tasks to specialized objects, an object can focus on its core responsibilities and rely on other objects to handle specific tasks. This can make code easier to understand, modify, and maintain.

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

Ans- Composition is a programming concept in which an object is made up of one or more other objects as its parts. Unlike inheritance, where a class inherits properties and behaviours from a superclass, composition involves creating an object that is composed of other objects that work together to provide the desired functionality.

In composition, each object that makes up the composite object is responsible for a specific part of the overall functionality. The composite object coordinates the behaviour of its parts to achieve the desired outcome. For example, imagine you have a class called Car that has a Engine object, a Transmission object, and a Chassis object as its parts. The Car object can coordinate the behaviour of its parts to start the engine, shift gears, and move the vehicle.

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

Ans- Bound methods are a concept in object-oriented programming that refers to methods that are associated with a specific instance of a class. When a method is called on an instance of a class, the method is bound to that instance, which means that it has access to the instance's data and can modify it if necessary.

To create a bound method in Python, you define a method within a class, and then create an instance of the class. When you call the method on the instance, it becomes bound to that instance.

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

Ans- Pseudoprivate attributes (sometimes called "name mangling") in object-oriented programming are used to provide a level of data hiding and encapsulation within a class.

The purpose of pseudoprivate attributes is to make an attribute inaccessible from outside the class, while still allowing the attribute to be accessed within the class and its subclasses. This helps prevent accidental modification or access to the attribute from outside the class, which can lead to unintended behaviour or bugs.

Pseudoprivate attributes are created by adding two underscores (‘\_\_’) as a prefix to the attribute name. This causes the attribute to be "mangled" by the Python interpreter, changing its name to ‘\_classname\_\_attribute’.

For example, if a class called ‘Person’ had a pseudoprivate attribute called ‘\_\_age’, it would be mangled to ‘\_Person\_\_age’. This means that the attribute can still be accessed and modified within the ‘Person’ class, but attempting to access it from outside the class using its original name (‘\_\_age’) would result in an ‘AttributeError’.