Q1. What is the meaning of multiple inheritance?

A) Multiple inheritance is a feature in object-oriented programming languages where a class can inherit attributes and methods from more than one parent class. In other words, a subclass can have more than one immediate superclass.

For example, consider a class hierarchy where you have a class Vehicle and another class ElectricDevice. Now, if you have a class ElectricCar, which can inherit properties from both Vehicle and ElectricDevice, it would be a case of multiple inheritance.

Q2. What is the concept of delegation?

A) Delegation is a design pattern in object-oriented programming where an object passes responsibility for a particular task or set of tasks to another object. Instead of performing the task itself, the delegating object (also known as the client object) forwards the request to the delegate object, which then handles the task.

Delegation is commonly used to achieve code reuse, modularize functionality, and promote loose coupling between objects. It allows objects to collaborate by distributing responsibilities among them, making the codebase more maintainable and flexible.

Q3. What is the concept of composition?

A) Composition is a design technique in object-oriented programming where objects are composed of other objects as parts, rather than inheriting functionality through inheritance. In composition, a class contains references to other classes (or objects) as member variables, and it delegates certain tasks or responsibilities to these contained objects.

The key idea behind composition is to create complex objects by combining simpler ones, promoting code reuse, maintainability, and flexibility. It allows for building modular and loosely coupled systems where each object focuses on a specific responsibility.

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

A) Bound methods are methods that are bound to a specific instance of a class. When a method is bound, it means that the method is associated with a particular object instance, and when it is called, the instance is automatically passed as the first argument to the method (traditionally named **self** in Python). This binding process allows the method to access and operate on the instance's attributes and properties.

Q5. What is the purpose of pseudoprivate attributes?

A) Pseudoprivate attributes in Python are a convention rather than a strict language feature. They are typically implemented by prefixing attribute names with double underscores (\_\_). The purpose of pseudoprivate attributes is to emulate private attributes in classes, providing a way to encapsulate data and implementation details within a class and prevent accidental modification or access from outside the class.

While Python does not have strict access control mechanisms like other languages (e.g., Java or C++), pseudoprivate attributes are a way to indicate to developers that certain attributes are intended for internal use within the class and should not be accessed or modified directly from outside the class.