Q1. What is the meaning of multiple inheritance?

Multiple inheritance is a feature in object-oriented programming where a class can inherit attributes and methods from multiple parent classes. In other words, a subclass can inherit from more than one superclass. This allows the subclass to combine features from different parent classes. However, multiple inheritance can lead to complexities and conflicts, so it needs to be used carefully to avoid ambiguity.

Q2. What is the concept of delegation?

Delegation is a design pattern where a class delegates certain responsibilities or tasks to another class. Instead of inheriting the behavior directly, a class maintains an instance of another class and forwards specific method calls or operations to that instance. This promotes code reuse, flexibility, and separation of concerns. Delegation allows for more dynamic behavior composition compared to traditional inheritance.

Q3. What is the concept of composition?

Composition is a design principle where a class is built by combining smaller classes or components. It involves creating objects of other classes within the class and using them to achieve desired behavior. Composition allows for a more modular and flexible design, as changes to the smaller components do not affect the entire class. It is often used to achieve code reuse and to model "has-a" relationships.

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

Bound methods are methods of a class that are associated with instances of that class. When you call a bound method on an instance, the instance is automatically passed as the first argument (often named self) to the method. Bound methods are a way for instances to interact with their own data and behavior. You use them by calling the method on an instance, and the instance itself is automatically passed as the first argument.

Q5. What is the purpose of pseudoprivate attributes?

Pseudoprivate attributes are attributes in a class that are named with a double underscore prefix (\_\_attribute\_name). They are intended to provide a form of name mangling to make the attribute more difficult to accidentally override in subclasses. While not truly private, they discourage direct access and modification by other classes. They are useful for avoiding unintentional attribute conflicts when using inheritance and to define class-specific behavior without interfering with subclasses.