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 for the combination of behaviors and characteristics from different classes, providing flexibility and code reuse.

Q2. What is the concept of delegation?
Delegation is a design pattern in which an object passes on a specific behavior or task to another object to perform. Rather than implementing the behavior itself, the object delegates the responsibility to another object, which is specialized in that behavior. This allows for code reuse, modularity, and separation of concerns.

Q3. What is the concept of composition?
Composition is a design principle in object-oriented programming where objects are combined together to form more complex objects. It involves creating complex objects by composing simpler objects as their parts. In composition, the composed objects are typically encapsulated within the containing object, and their behavior is utilized to achieve the desired functionality.

Q4. What are bound methods and how do we use them?
Bound methods are methods that are associated with an instance of a class. When a method is accessed through an instance, it is automatically bound to that instance, allowing it to access the instance's attributes and perform operations specific to that instance. Bound methods are commonly used to manipulate and interact with object state, encapsulating behavior within the context of the object.

Q5. What is the purpose of pseudoprivate attributes?
Pseudoprivate attributes, denoted by a double underscore prefix (e.g., `__attribute`), are a convention in Python to indicate that an attribute should be treated as private, even though Python does not enforce true private access. The purpose of pseudoprivate attributes is to signal that an attribute is intended for internal use within the class and should not be accessed or modified directly from outside the class. It helps to maintain encapsulation, preventing external code from unintentionally modifying the internal state of an object and promoting better code organization and maintenance.