Q1. What is the meaning of multiple inheritance?

Multiple inheritance is a concept in object-oriented programming (OOP) where a class can inherit attributes and behaviors from more than one parent class. In other words, a class can inherit properties and methods from multiple base classes. This allows a derived class (also called a subclass) to have characteristics from multiple sources, incorporating the features of different parent classes.

Q2. What is the concept of delegation?

Delegation is a design pattern in which an object, called the delegate, is responsible for performing certain tasks on behalf of another object, called the delegator. This can be done by the delegator forwarding method calls and attribute access to the delegate. In its most basic form, delegation can be implemented using the following approach: the delegator passes requests for certain actions to the delegate, which then performs the actions on behalf of the delegator.

Q3. What is the concept of composition?

Composition is a fundamental concept in object-oriented programming (OOP) that involves building complex objects by combining simpler objects or components. It's a design principle that promotes creating classes by composing them from other classes, as opposed to using inheritance to extend or specialize existing classes. Composition allows Us to achieve code reuse, modularity, and flexibility while avoiding some of the issues associated with deep class hierarchies and multiple inheritance.

Key features and benefits of composition include:

1. **Modularity:** Composition allows us to build classes with well-defined and self-contained components. Each component can have its own behavior and encapsulate its own state.
2. **Code Reuse:** By composing classes from existing components, we can reuse well-tested and established functionality without having to recreate it.
3. **Flexibility:** We can easily modify or extend the behavior of a composite class by adding, removing, or replacing components. This leads to more adaptable and maintainable code.

4. **Avoiding Complex Hierarchies:** Unlike deep inheritance hierarchies, which can become difficult to manage, composition lets We avoid the issues of "class explosion" and the complexities that come with it.

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

Bound methods are a type of method that is associated with an instance of a class. When we define a method within a class and then create an object (instance) of that class, the methods of the class become bound methods when called on that specific instance. Bound methods have a reference to the instance they belong to, and they can access and manipulate the instance's attributes and properties.

Q5. What is the purpose of pseudoprivate attributes?

pseudoprivate attributes are attributes that have names with a double underscore prefix (__) but do not have a double underscore suffix. For example, an attribute named __variable is considered pseudoprivate. The purpose of pseudoprivate attributes is to introduce a form of name mangling to make attributes more unique, minimizing accidental name clashes in subclasses.

When we define an attribute with a double underscore prefix, Python performs name mangling on the attribute name by adding a prefix of _classname to it, where classname is the name of the class containing the attribute. This helps prevent subclasses from accidentally overriding or accessing attributes meant to be internal to the parent class.

```
class CustomClass:
    def __init__(self):
        self.__value = 42

obj = CustomClass()
print(obj._CustomClass__value)
```