Q1. What is the meaning of multiple inheritance?

Ans. Multiple inheritance is a feature of object-oriented programming languages that allows a class to inherit from more than one parent class. This means that a subclass can inherit attributes and methods from multiple parent classes, giving it access to a wider range of functionality than would be possible with single inheritance.

In multiple inheritance, a subclass inherits all the attributes and methods of each of its parent classes. This can be useful in cases where a subclass needs to combine the functionality of two or more parent classes. For example, a class representing a hybrid car might inherit from both a gas-powered car class and an electric car class to combine their features.

However, multiple inheritance can also be complex and potentially lead to issues such as diamond inheritance, where a subclass indirectly inherits the same attribute or method from two different parent classes. To avoid these issues, some programming languages offer alternative solutions such as interfaces or mixins.

Q2. What is the concept of delegation?

Ans. Delegation is a programming concept that involves one object passing off a responsibility or task to another object. Rather than performing the task itself, the object delegates the task to another object that is better suited to handle it. In object-oriented programming, delegation is typically implemented by creating a reference to another object within a class and then delegating certain methods or responsibilities to that object. This is often done to separate concerns and create more modular and maintainable code.

Delegation can also be used to create more flexible and customizable classes. By delegating certain behaviors or methods to external objects, classes can be composed in different ways to create different behaviors or functionality. This can be useful in cases where different instances of a class need to behave in different ways, or when a class needs to be extended or customized in different ways for different use cases.

Q3. What is the concept of composition?

Ans. Composition is a programming concept in which a class is composed of one or more instances of other classes, rather than inheriting from those classes. Composition allows for more flexible and modular design than inheritance, as it allows classes to be combined and reconfigured in different ways to create new functionality. Composition allows for more modular and maintainable code, as changes to the composition of a class can be made independently of the classes being composed. It also allows for greater flexibility and customization, as different instances of a class can be composed in different ways to create different functionality.

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

Ans. In Python, a bound method is a method that is bound to an instance of a class. When a method is bound to an instance, it can be called on that instance, and the instance is automatically passed as the first argument to the method. This is similar to how instance variables are accessed in Python.

To create a bound method in Python, you need to define a method within a class, and then create an instance of that class. You can then call the method on the instance, and the instance will be automatically passed as the first argument.

Bound methods are a fundamental concept in Python object-oriented programming, as they allow classes to define behavior that is specific to instances of the class. By using bound methods, you can create more flexible and modular code that can be customized for different use cases.

Q5. What is the purpose of pseudoprivate attributes?

Ans. In Python, pseudoprivate attributes are attributes that are marked as private by using a double underscore prefix (‘\_\_’) in their names, but are not actually private in the sense that they can still be accessed and modified from outside the class.

The purpose of pseudoprivate attributes is to avoid naming conflicts with attributes of subclasses or other classes. By using a double underscore prefix, Python automatically mangles the attribute name by adding the name of the class to the front of the attribute name, making it more difficult to accidentally overwrite or modify the attribute from outside the class. While pseudoprivate attributes are not truly private, they can be useful in preventing naming conflicts and making it more difficult to accidentally modify or overwrite important attributes. However, it's important to note that pseudoprivate attributes should not be relied upon for security or data hiding, as they can still be accessed and modified with a bit of effort.