Q1. What is the meaning of multiple inheritance?

A.1 Multiple inheritance is a feature in object-oriented programming that allows a class to inherit attributes and methods from more than one parent class. This means that a derived (child) class can have multiple base (parent) classes, and it can access the properties and behaviours of all its parent classes.

**How Multiple Inheritance Works**

In multiple inheritance, when you create a subclass, it inherits from all specified parent classes, and you can override or extend the functionality provided by those parent classes. Python supports multiple inheritance, allowing a class to inherit from multiple classes by listing them in the class definition.

Q2. What is the concept of delegation?

A2. Delegation is a design pattern in object-oriented programming where an object hands off or "delegates" a task or responsibility to another object. Instead of performing a task itself, the delegating object passes the task to a delegate (another object), which then carries out the task on behalf of the original object. This allows for greater modularity, code reuse, and flexibility in design.

**Key Concepts of Delegation**

1. **Delegating Object**: This is the object that wants to perform a certain task but instead passes it to another object.
2. **Delegate**: This is the object that actually performs the task on behalf of the delegating object.

**How Delegation Works**

In delegation, the delegating object typically has a reference to the delegate object and calls methods on the delegate object to perform specific tasks. The delegating object may intercept the call, modify arguments, or simply pass the call directly to the delegate.

Q3. What is the concept of composition?

A3. Composition is a design principle in object-oriented programming where a class is composed of one or more objects from other classes, enabling it to use their functionality. Instead of inheriting from a parent class, composition involves creating complex types by combining objects from other classes, which are typically referenced as attributes within the composed object.

**Key Concepts of Composition**

1. **Has-A Relationship**: Composition represents a "has-a" relationship between objects. For example, a Car class may have an Engine class as a component. This contrasts with inheritance, which represents an "is-a" relationship.
2. **Encapsulation**: Composition encourages encapsulation by allowing you to build complex behaviors by combining simpler, more focused objects without exposing their internal workings.
3. **Reusability**: Composition promotes code reuse because you can create new functionality by combining existing classes in different ways, without modifying or extending those classes.
4. **Flexibility**: With composition, the components (or member objects) can be swapped out or replaced at runtime, allowing for more flexible and maintainable code.

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

A4. A bound method in Python is a method that is associated with an instance of a class. When you access a method on an instance of a class, Python automatically binds the instance to the method, creating a bound method. This means that the method has access to the instance's attributes and other methods via the self parameter.

**Key Concepts of Bound Methods**

1. **Instance Association**: A bound method is tied to a specific instance of a class. When the method is called, Python implicitly passes the instance as the first argument (self), giving the method access to the instance's data and other methods.
2. **Difference from Unbound Methods**: Unbound methods are functions defined within a class that are not associated with an instance. When accessed via the class rather than an instance, they don't automatically receive an instance as the first argument.

Q5. What is the purpose of pseudoprivate attributes?

A5. Pseudoprivate attributes in Python are a mechanism used to prevent accidental access or modification of class attributes, particularly in cases of inheritance. They are not truly private, but Python provides a way to "mangle" the attribute names to make them less accessible from outside the class, thus avoiding name clashes in subclasses.

**How Pseudoprivate Attributes Work**

In Python, pseudoprivate attributes are created by prefixing an attribute name with two underscores (\_\_). This triggers name mangling, where Python internally changes the name of the attribute to include the class name, making it harder to accidentally access from outside the class.

For example:

* An attribute \_\_attribute in a class MyClass would be internally renamed to \_MyClass\_\_attribute.