**Assignment - 5**

Q1. What is the meaning of multiple inheritance?

Ans: Multiple inheritance in object-oriented programming refers to a feature where a subclass can inherit attributes and methods from more than one parent class. This means that a class can inherit behavior from multiple classes, allowing for code reuse and the creation of complex class hierarchies. However, multiple inheritance can lead to issues such as the diamond problem, where the same method or attribute is inherited from multiple parent classes, causing conflicts.

Q2. What is the concept of delegation?

Ans: Delegation is a design pattern where an object forwards a method call or message to another object, known as the delegate, to perform a specific task on its behalf. Instead of implementing the behavior directly, the delegating object relies on the delegate to handle the task. Delegation promotes code reuse, modularity, and separation of concerns by allowing objects to collaborate and specialize in specific responsibilities.

Q3. What is the concept of composition?

Ans: Composition is a design principle in object-oriented programming where objects are composed of other objects as part of their internal structure. Unlike inheritance, where objects inherit behavior from parent classes, composition involves creating complex objects by combining simpler objects, known as components or parts. This promotes code reuse, encapsulation, and flexibility by allowing objects to be constructed from interchangeable components.

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

Ans: Bound methods are methods that are associated with a specific instance of a class. When a method is called on an instance, Python automatically passes the instance itself as the first argument to the method, commonly referred to as self. This binding process creates a bound method, where the method is bound to the instance and can access its attributes and methods. Bound methods are invoked using dot notation (instance.method()), and they maintain a reference to the instance, allowing them to operate on instance-specific data.

Q5. What is the purpose of pseudoprivate attributes?

Ans: Pseudoprivate attributes in Python are attributes that are prefixed with double underscores (\_\_) but do not undergo name mangling like true private attributes. Instead, they are subject to a name-mangling mechanism that prefixes the attribute name with \_classname to make it less likely to collide with attributes in subclasses. Pseudoprivate attributes are primarily used to emulate private access control and prevent accidental access or modification from outside the class. However, they are not truly private and can still be accessed or modified through name mangling. Their purpose is mainly to signal to other developers that the attribute is intended for internal use within the class.