**Lesson 13:OOP Part 2**

**Research Work – Advanced OOP Concepts**

1. **What is encapsulation, and how does using private attributes prevent misuse?**

Encapsulation is a key principle of object-oriented programming that bundles data and the methods that operate on it into a single unit while restricting direct access to certain details. By making attributes private, a class hides its internal state and only allows controlled access through methods such as getters and setters. This prevents accidental misuse, such as setting invalid values or breaking important rules that the class enforces. It also ensures that changes to internal implementation do not affect how other parts of the program interact with the class, keeping the design secure and consistent.

**2.Explain the concept of method resolution order (MRO) in multiple inheritance with an example.**

The Method Resolution Order (MRO) in Python defines the order in which methods and attributes are searched when a class uses multiple inheritance. It ensures a consistent and predictable path for method lookup, avoiding ambiguity in cases like the diamond problem. Python follows the C3 linearization algorithm to determine this order, which can be viewed using the .mro() method. For example, if a class D inherits from both B and C, Python will check methods in the order D → B → C → A → object, ensuring that the correct method is executed without confusion.

**3.How does polymorphism facilitate flexible function design across different classes?**

Polymorphism in object-oriented programming allows objects of different classes to be treated through a common interface, enabling the same function or method to work with different types of objects. This means a single function name can perform different tasks depending on the object that calls it, reducing the need for duplicate code. For example, a draw() method could be defined differently in Circle, Square, or Triangle classes, but a program can call shape.draw() without knowing the exact type of shape. This flexibility makes functions more general, easier to extend, and adaptable to new classes, promoting reusable and maintainable code.

**4. What are the benefits of utilizing super() in inheritance for code maintainability?**

Using **super()** in inheritance provides several benefits that improve code maintainability. It allows a child class to call methods from its parent class without directly naming the parent, making the code easier to modify if the class hierarchy changes. This avoids duplication and ensures that parent class initialization or overridden methods are executed properly, keeping the class behavior consistent. In multiple inheritance, super() also works with the Method Resolution Order (MRO), ensuring that each class in the hierarchy is called in the correct order without repetition. Overall, it makes the code cleaner, reduces errors, and simplifies long-term maintenance.

**5.** **How does operator overloading improve readability? Provide a real-world example in code.**

Operator overloading improves readability by allowing user-defined classes to use familiar operators like +, -, or \* instead of long method calls. This makes the code more intuitive and closer to real-world usage. For example, in a Complex number class, overloading the + operator allows us to add two objects using c1 + c2 instead of calling a method like c1.add(c2). Such natural expressions make programs easier to read, write, and maintain, especially when dealing with mathematical or domain-specific objects.