Advanced Programming Concepts

**Decorators -**

In Python, **decorators** are functions that modify the behavior of another function or method. They are often used to **wrap** functions to add functionality without changing their code. Decorators are a key feature of Python and help make code more modular and readable.

Decorators are applied to functions or methods using the @decorator\_name syntax.

### **How Do Decorators Work?**

A **decorator** is essentially a higher-order function, meaning it takes a function as an argument and returns a new function. Here’s the step-by-step process:

1. **Define the Decorator Function:** The decorator takes a function as input and returns a new function that typically extends or modifies the behavior of the input function.
2. **Wrap the Target Function:** Inside the decorator, you define a wrapper function that is executed instead of the original function. The wrapper can run the original function and add functionality before or after it.
3. **Apply the Decorator:** Using the @decorator\_name syntax, you apply the decorator to a function. Python automatically passes the function to the decorator.

**Decorator Chaining**

When multiple decorators are applied to a single function, the priority or order in which they are applied follows a **bottom-up** approach, meaning the decorator closest to the function definition is applied first, and the outer decorators are applied afterward.

### **How Multiple Decorators Work**

1. **Decorator Application Order:**
   * The decorators are applied in the **reverse order** they appear, starting from the one closest to the function.
2. **Function Wrapping:**
   * Each decorator wraps the function it decorates, meaning the innermost decorator wraps the function first, and then the outer decorators wrap the result of the previous wrapping.

**MetaClasses -**

A metaclass in Python is a class that defines how other classes are constructed. In Python, everything is an object, including classes themselves. A metaclass is the "class of a class," meaning it defines how classes themselves behave. By default, Python uses the type class as the metaclass for all classes, but you can create your own metaclasses to modify class creation behavior.

When you create a new class in Python, it is an instance of a metaclass. The metaclass defines the structure and behavior of the class. You can customize this behavior by defining your own metaclass.

### **Key Methods in Metaclasses**

1. **\_\_new\_\_(cls, name, bases, dct)**: This method is called to create a new class. We can override this method to control how the objects are created. It receives:
   * cls: the metaclass.
   * name: the name of the class being created.
   * bases: the base classes of the class.
   * dct: the dictionary of class attributes and methods.

This method returns the class object.

1. **\_\_init\_\_(cls, name, bases, dct)**: This method is called after the class is created, but it is less commonly overridden than \_\_new\_\_. It's useful if you need to perform some additional initialization in the class.