What is Object-Oriented Programming? What are the benefits of OOP?

Object-Oriented Programming (OOP) is a way of writing code by grouping related data and actions together into **objects**. These objects are created from **classes**, which serve as blueprints. Each object contains information called **attributes** and functions called **methods** that act on that data.

Benefits of OOP:

- **Modularity:** Code is divided into small, manageable parts (objects), making it easier to understand and maintain.
- **Reusability:** You can reuse existing classes in other programs or create new classes based on them (inheritance).
- **Scalability:** OOP makes it easier to add new features by creating or changing objects without affecting the whole program.
- **Encapsulation:** Objects protect their data from outside interference, helping to keep it safe and secure.
- **Abstraction:** It hides complex implementation details, allowing users to work with simple interfaces.

What are objects and classes in Python? Give a real-world example.

In Python, a **class** is like a blueprint that defines what attributes (data) and methods (functions) an object will have. An **object** is a specific instance of that class with real values.

Example:
Imagine a class called **Car**. It defines attributes like `color`, `brand`, and `max_speed`, and methods like `accelerate()` and `brake()`. When you create an object like `my_car = Car("Red", "Toyota", 180)`, this object represents a specific red Toyota car that can go up to 180 km/h. You can create many different car objects from the Car class, each with its own unique details.

Inheritance

Inheritance is a way to create a new class (called a **child** or **subclass**) that takes attributes and methods from an existing class (called a **parent** or **superclass**). It helps reuse code and create logical relationships between classes.

For example, if you have a base class called **Recipe**, you can create a subclass called **BakingRecipe** that inherits everything from Recipe but adds new features specific to baking, like an `oven_temperature` attribute. This means BakingRecipe can use the methods from Recipe but also have its own special properties.

Polymorphism

Polymorphism means "many forms" and lets different classes be used through the same interface. It allows a method to work differently depending on which class's object calls it.

For instance, you might have two subclasses of Recipe: **CakeRecipe** and **DrinkRecipe**, both with a method called `prepare()`. When you call `prepare()` on a CakeRecipe or a DrinkRecipe object, Python knows which version of the method to use based on the object's class. This lets you write flexible and easy-to-extend code.

Operator Overloading

Operator overloading lets you change how standard Python operators like `+`, `-`, or `==` work with objects of your class. You do this by defining special methods such as `__add__` for `+`, `__eq__` for `==`, or `__str__` for printing.

For example, in a **Recipe** class, you could overload the `+` operator to combine ingredients from two recipes. When you do `recipe1 + recipe2`, Python will call the `__add__` method you defined and return a new recipe that includes ingredients from both.