Introduction to Object-Oriented Programming (OOP) in Python

Classes • Attributes • Methods • Objects

Introduction

- OOP organizes code to mirror real-life concepts.
- Main building blocks: Classes, Attributes, Methods, Objects.
- Helps write organized, reusable, and scalable programs.

Understanding Classes

- A class is a blueprint for creating objects.
- Defines attributes (properties) and methods (functions).
- Example: class Car with color and speed.

Attributes in OOP

- Attributes describe an object’s characteristics.
- They can differ between objects from the same class.
- Example: my_car.color = 'Red', my_car.speed = 120

Methods in OOP

- Methods are functions inside a class defining behavior.
- Example: accelerate() increases speed.
- Objects can share methods but have different attribute values.

Objects and Instances

- An object is a specific realization of a class.
- Each has its own attribute values but shares methods.
- Example: Car('Red', 200) and Car('Blue', 180)

Real-Life Examples

- Car Example: Attributes - color, top speed; Methods - start, stop, accelerate.
- Student Example: Attributes - name, age; Methods - take_exam, attend_classes.

How OOP Works: Step-by-Step

- 1. Define the Class (blueprint).
- 2. Define the Attributes (properties).
- 3. Define the Methods (actions).
- 4. Create Objects (instances).

Why OOP is Important

- Reusability: Create many objects from one class.
- Modularity: Break problems into smaller parts.
- Maintainability: Update class, changes apply everywhere.
- Scalability: Grow program easily.

Summary Table

- Class: Blueprint → Example: class Car:
- Attribute: Property → Example: color, speed
- Method: Action → Example: accelerate()
- Object: Instance → Example: my_car = Car('Red', 120)