Classes In OOP

In Object-Oriented Programming (OOP), a class is a blueprint for creating objects. It encapsulates data for the object and methods to manipulate that data. Classes are fundamental to OOP because they enable the organization and reuse of code, encapsulation, and abstraction. Here's an overview of classes in OOP and their key concepts:

**Basic Concepts of Classes**

1. **Definition of a Class**:
   * A class defines a type of object, including its attributes (data) and behaviors (methods).
2. **Attributes**:
   * These are variables that belong to a class. They define the state or properties of an object.
3. **Methods**:
   * These are functions defined within a class that describe the behaviors or actions that the objects of the class can perform.
4. **Constructor**:
   * A special method called \_\_init\_\_ in Python, which is automatically called when a new object of the class is created. It initializes the object's attributes.
5. **Inheritance**:
   * This allows a class (child class) to inherit attributes and methods from another class (parent class). It promotes code reuse.
6. **Encapsulation**:
   * Encapsulation is the concept of hiding the internal state of an object and only exposing a controlled interface. This is typically achieved through private and public attributes and methods.
7. **Polymorphism**:
   * Polymorphism allows different classes to be treated as instances of the same class through a common interface. It often involves method overriding and method overloading.

**EXAMPLE**

### ****Library Book****

#### Concept:

In the context of a library, think of a **class** as a category of items with common features. For example, a LibraryBook class represents a template for all books in the library. Each book has attributes like title, author, and ISBN, and methods to perform actions related to the book, such as checking it out or returning it.

#### **Class Definition**

1. **Attributes**:
   * **Title**: The name of the book.
   * **Author**: The person who wrote the book.
   * **ISBN**: A unique identifier for the book.
   * **Status**: Whether the book is available or checked out.
2. **Methods**:
   * **Check Out**: Marks the book as checked out.
   * **Return**: Marks the book as available again.
   * **Display Info**: Shows details about the book.

**OBJECTS IN OOP**

In Object-Oriented Programming (OOP), **objects** are instances of classes. They are the fundamental entities in OOP that encapsulate data and behavior. To understand objects better, let’s break down the key concepts and attributes of objects in OOP:

### Understanding Objects in OOP

#### **What Is an Object?**

* **Definition**: An object is a specific instance of a class that contains both data and methods. While a class defines a blueprint for creating objects, each object represents a distinct, concrete entity created from that blueprint.
* **Example**: If Car is a class, then my\_car could be an object of the Car class.

#### **Attributes of an Object**

* **State**: The state of an object is defined by its attributes (also called properties or fields). These attributes hold the data related to the object. For example, for a Car object, attributes might include color, make, and model.
* **Instance Variables**: These are variables defined in the class and are specific to each object. They represent the unique data stored in an object.

#### **Methods of an Object**

* **Behavior**: The behavior of an object is defined by the methods of its class. Methods are functions that operate on the object’s attributes and can perform actions or computations. For instance, a Car object might have methods like start\_engine() and drive().
* **Instance Methods**: These are methods defined in the class that can access and modify the instance variables of the object.

#### **Creating and Using Objects**

* **Instantiation**: The process of creating an object from a class is known as instantiation. In Python, this is done by calling the class as if it were a function.
* **Usage**: Once an object is created, you can access its attributes and call its methods using the dot notation.

EXAMPLE

### ****Student****

#### **Introduction**

A **student** represents an individual with specific attributes and actions they can perform. Each student is an instance with unique characteristics and capabilities.

#### **Student Attributes**

* **Full Name**: The complete name of the student (e.g., "Alice Johnson").
* **Identification Number**: A unique ID assigned to the student (e.g., "S123456").
* **Academic Year**: The student's current academic level (e.g., "Sophomore").
* **Grade Point Average (GPA)**: The student's average academic performance (e.g., 3.8).

#### **Student Actions**

* **Enroll in New Course**: Ability to register for a new course.
* **Update Academic Performance**: Ability to revise the GPA based on new grades.
* **View Profile**: Display detailed information about the student.