CLASSES

In Object-Oriented Programming (OOP), a **class** is a fundamental construct that serves as a blueprint for creating objects. It defines a type of object by specifying its attributes (data) and methods (functions) that operate on that data. Here’s a more detailed look at what a class means in OOP:

**Key Concepts of Classes**

1. **Introduction to Classes**
   * A class acts as a blueprint for creating objects. It defines a structure that includes both attributes (data) and methods (functions), determining the properties and behaviors of the objects instantiated from it.
2. **Attributes: The Data Fields**
   * Attributes are variables defined within a class that store information about an object. They describe the state or characteristics of the object, such as its size, color, or any other specific details.
3. **Methods: Defining Behavior**
   * Methods are functions defined inside a class that dictate the actions or behaviors an object can perform. They operate on the object's attributes and enable the object to perform tasks or interact with other objects.
4. **Constructors (\_\_init\_\_ Method)**
   * The constructor, \_\_init\_\_, is a special method automatically called when a new instance of a class is created. It initializes the object’s attributes with provided values, setting up the object's initial state.
5. **Inheritance: Building on Existing Classes**
   * Inheritance allows one class (the child class) to inherit attributes and methods from another class (the parent class). This promotes code reuse and supports the creation of more specific classes based on existing ones.
6. **Encapsulation: Protecting Object Data**
   * Encapsulation involves grouping the data (attributes) and methods that operate on the data into a single unit (class). It also restricts direct access to some of the object's internal details, providing a controlled interface for interaction.
7. **Polymorphism: Flexibility in Method Usage**
   * Polymorphism allows objects of different classes to be treated as objects of a common superclass. This principle enables the same method to behave differently based on the object’s class, achieved through techniques like method overriding.
8. **Abstraction: Simplifying Complex Systems**
   * Abstraction hides the complex implementation details and exposes only the necessary parts of an object’s functionality. This helps in managing complexity by focusing on high-level interactions rather than low-level details.

EXAMPLE

### ****Smartphone****

#### **Overview of the Smartphone Class**

A **class** in programming is like a template that defines a category of objects sharing common attributes and behaviors. In this example, the Smartphone class represents the general properties and functions that all smartphones have. Each specific smartphone (object) created from this class will have its own unique values for these attributes.

#### **Attributes of the Smartphone Class**

* **Brand**: Identifies the manufacturer of the smartphone (e.g., Samsung, Apple).
* **Model**: Specifies the particular model of the smartphone (e.g., Galaxy S23, iPhone 14).
* **Battery Life**: Indicates the remaining battery percentage of the smartphone.
* **Storage Capacity**: Represents the total storage available on the smartphone, measured in gigabytes (GB).

#### **Methods to Interact with a Smartphone**

* **Make a Call**: Simulates the action of dialing a phone number and making a call.
* **Install an App**: Simulates the process of installing a new application on the smartphone.
* **Charge Battery**: Updates the battery life based on the amount of charge added, with a check to ensure it does not exceed 100%.
* **Show Details**: Displays comprehensive information about the smartphone's attributes.

OBJECTS

#### **Concept of an Object**

An **object** in Object-Oriented Programming (OOP) is a distinct instance created from a class. It embodies both data and functions that operate on that data, representing a concrete occurrence of the abstract blueprint defined by the class.

#### **Characteristics of an Object**

* **State**:
  + **Data**: The state of an object is defined by its attributes, which are specific variables holding data relevant to that object. For instance, in a Car class, attributes might include color, make, and model. Each object will have its own values for these attributes.
* **Behavior**:
  + **Methods**: The behavior of an object is determined by the methods defined in its class. These methods are functions that perform operations or computations using the object's attributes. For example, methods in a Car class might include start\_engine() and drive().

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

* **Instantiation**:
  + **Process**: The process of creating an object from a class is called instantiation. In Python, you create an object by calling the class name with any necessary arguments. This results in a new instance of the class with its own unique set of data.
* **Interaction**:
  + **Usage**: Once an object is instantiated, you can interact with it by accessing its attributes and invoking its methods using dot notation. This allows you to manipulate the object's state and perform actions.

EXAMPLE

### ****Coffee Mug****

#### **Concept of an Object**

In the real world, an **object** can be anything that has distinct characteristics and functions. For instance, consider a **coffee mug**. Each coffee mug can be viewed as an object with specific attributes and behaviors, similar to how objects in OOP work.

#### **Attributes of a Coffee Mug**

* **Color**: The color of the coffee mug (e.g., red, blue).
* **Size**: The volume capacity of the mug (e.g., 250 ml, 350 ml).
* **Material**: The material from which the mug is made (e.g., ceramic, glass).

**Behaviors of a Coffee Mug**

* **Hold Liquid**: The ability to hold a specific amount of liquid.
* **Heat Retention**: Keeps hot beverages warm for a period.
* **Wash**: Can be cleaned in a dishwasher or by hand.
* **Design**: The aesthetic or graphical design on the mug.