Object-Oriented Programming (OOPS-1)

Introduction to OOPS

Object-oriented programming System(OOPs) is a programming paradigm based on the concept of *"objects"* and *"classes"* that contain data and methods. The primary purpose of OOP is to increase the flexibility and maintainability of programs. It is used to structure a software program into simple, reusable pieces of code **blueprints** (called *classes*) which are used to create individual instances of *objects*.

Python supports a variety of programming approaches. One of the most useful and popular programming approaches is OOPS.

What is an Object?

The object is an entity that has a state and a behavior associated with it. It may be any real-world object like the mouse, keyboard, chair, table, pen, etc.
Integers, strings, floating-point numbers, even arrays and dictionaries, are all objects. More specifically, any single integer or any single string is an object. The number **12** is an object, the string **"Hello, world"** is an object, a list is an object that can hold other objects, and so on. You've been using objects all along and may not even realize it.

What is a Class?

A class is a **blueprint** that defines the variables and the methods (Characteristics) common to all objects of a certain kind.

Example: If **Car** is a class, then **Maruti 800** is an object of the **Car** class. All cars share similar features like 4 wheels, 1 steering wheel, windows, breaks etc. Maruti 800 (The **Car** object) has all these features.

Classes vs Objects (Or Instances)

Classes are used to create user-defined data structures. Classes define functions called **methods**, which identify the behaviors and actions that an object created from the class can perform with its data.

In this module, you'll create a **Car** class that stores some information about the characteristics and behaviors that an individual **Car** can have.

A class is a blueprint for how something should be defined. It doesn't contain any data. The **Car** class specifies that a name and a top-speed are necessary for defining a **Car**, but it doesn't contain the name or top-speed of any specific **Car**.

While the class is the blueprint, an instance is an object that is built from a class and contains real data. An instance of the **Car** class is not a blueprint anymore. It's an actual car with a **name**, like Creta, and with a **top speed** of 200 Km/Hr.

Put another way, a class is like a form or questionnaire. An **instance** is like a form that has been filled out with information. Just like many people can fill out the same form with their unique information, many instances can be created from a single class.

Defining a Class in Python

All class definitions start with the `class` keyword, which is followed by the name of the class and a colon(**:**). Any code that is indented below the class definition is considered part of the class's body.
Here is an example of a **Car** class:

```
class Car:
    pass
```

The body of the **Car** class consists of a single statement: the `pass` keyword. As we have discussed earlier, `pass` is often used as a placeholder indicating where code will eventually go. It allows you to run this code without Python throwing an error.

Note: Python class names are written in *CapitalizedWords* notation by convention. **For example**, a class for a specific model of Car like the Bugatti Veyron would be written as **BugattiVeyron**. The first letter is capitalized. This is just a good programming practice.

The **Car** class isn't very interesting right now, so let's spruce it up a bit by defining some properties that all Car objects should have. There are several properties that we can choose from, including **color**, **brand**, and **top-speed**. To keep things simple, we'll just use **color** and **top-speed**.

Constructor

- Constructors are generally used for instantiating an object.
- The task of a constructor is to initialize(assign values) to the data members of the class when an object of the class is created.
- In Python, the `.__init__()` method is called the **constructor** and is always called when an object is created.

- **Note:** Names that have leading and trailing double underscores are reserved for special use like the **__init__** method for object constructors. These methods are known as **dunder methods**.

Syntax of Constructor Declaration

```
def __init__(self):
    # body of the constructor
```

Types of constructors

- **Default Constructor:** The default constructor is a simple constructor that doesn't accept any arguments. Its definition has only one argument which is a reference to the instance being constructed known as `self`.
- **Parameterized Constructor:** A constructor with parameters is known as a parameterized constructor. The parameterized constructor takes its first argument as a reference to the instance being constructed known as `self` and the rest of the arguments are provided by the programmer.

The properties that all **Car** objects must have been defined in `.__init__()`. Every time a new **Car** object is created, `.__init__()` sets the initial state of the object by assigning the values of the object's properties. That is, `.__init__()` initializes each new instance of the class.
When a new class instance is created, the instance is automatically passed to the self parameter in `.__init__()` so that new attributes can be defined on the object.

The `self` Parameter

- The `self` parameter is a reference to the current instance of the class and is used to access variables that belong to the class.
- It does not have to be named `self`, you can call it whatever you like, but it has to be the first parameter of any function in the class.