Introduction to OOPS

As the name suggests, Object-Oriented programming or OOPs refers to the language that uses the concept of class and object in programming. The main aim of OOPs is to implement real-world entities such as polymorphism, inheritance, encapsulation, abstraction, etc. The popular object-oriented programming languages are c++, java, python, PHP, c#, etc. ***Simula*** is the first object-oriented programming language.

Why do we use object-oriented programming?

- OOPs, make the development and maintenance of projects more manageable.
- OOPs, provide the feature of data hiding that is good for security concerns.
- We can provide the solution to real-world problems if we are using object-oriented programming.

OOPs Concepts: Let's look at the topics which we are going to discuss.

- Access modifiers
- Class
- Object
- Encapsulation
- Abstraction
- Polymorphism
- Inheritance
- Constructor
- Destructor

What is an Object?

- The object is an entity that has a state and 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** (a plan basically) that is used to define (bind together) a set of variables and methods (Characteristics) that are common to all objects of a particular kind.

Example: If **Car** is a class, then **Maruti 800** is an object of the **Car** class. All cars share similar features like wheels,1steeringwheel, windows, breaks, etc.Maruti800(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 a 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, the **pass** is often used as a placeholder indicating where the 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**.

Example:

```
class Car:
  pass

c = Car
c.name = "ferrari"
c.topspeed = 400
print(c.name)
print(c.topspeed)
```

output:

```
ferrari

400
```