Python OOPs Concepts

Object Oriented Programming is a fundamental concept in Python, empowering developers to build modular, maintainable, and scalable applications. By understanding the core OOP principles—classes, objects, inheritance, encapsulation, polymorphism, and abstraction—programmers can leverage the full potential of Python's OOP capabilities to design elegant and efficient solutions to complex problems.

What is Object-Oriented Programming in Python?

In Python object-oriented Programming (OOPs) is a programming paradigm that uses objects and classes in programming. It aims to implement real-world entities like inheritance, polymorphisms, encapsulation, etc. in the programming. The main concept of object-oriented Programming (OOPs) or oops concepts in Python is to bind the data and the functions that work together as a single unit so that no other part of the code can access this data.

Python Class

A class is a collection of objects. A class contains the blueprints or the prototype from which the objects are being created. It is a logical entity that contains some attributes and methods.

To understand the need for creating a class let's consider an example, let's say you wanted to track the number of dogs that may have different attributes like breed, and age. If a list is used, the first element could be the dog's breed while the second element could represent its age. Let's suppose there are 100 different dogs, then how would you know which element is supposed to be which? What if you wanted to add other properties to these dogs? This lacks organization and it's the exact need for classes.

Some points on Python class:

- Classes are created by keyword class.
- Attributes are the variables that belong to a class.
- Attributes are always public and can be accessed using the dot (.) operator. Eg.: Myclass.Myattribute

Python Objects

In object oriented programming Python, The object is an entity that has a state and behavior associated with it. It may be any real-world object like a 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.

An object consists of:

- State: It is represented by the attributes of an object. It also reflects the properties of an object.
- Behavior: It is represented by the methods of an object. It also reflects the response of an object to other objects.
- Identity: It gives a unique name to an object and enables one object to interact with other objects.

The Python self

- Class methods must have an extra first parameter in the method definition. We do not give a value for this parameter when we call the method, Python provides it
- If we have a method that takes no arguments, then we still have to have one argument.
- This is similar to this pointer in C++ and this reference in Java.

When we call a method of this object as myobject.method(arg1, arg2), this is automatically converted by Python into MyClass.method(myobject, arg1, arg2) – this is all the special self is about.