Python OOPs Concepts

Like other general purpose languages, python is also an object-oriented language since its beginning. Python is an object-oriented programming language. It allows us to develop applications using an Object Oriented approach. In Python, we can easily create and use classes and objects.

Major principles of object-oriented programming system are given below.

- Object
- Class
- Method
- Inheritance
- Polymorphism
- Data Abstraction
- Encapsulation

Object

The object is an entity that has state and behavior. It may be any real-world object like the mouse, keyboard, chair, table, pen, etc.

Everything in Python is an object, and almost everything has attributes and methods. All functions have a built-in attribute __doc__, which returns the doc string defined in the function source code.

Class

The class can be defined as a collection of objects. It is a logical entity that has some specific attributes and methods. For example: if you have an employee class then it should contain an attribute and method, i.e. an email id, name, age, salary, etc.

Syntax

```
class ClassName:
    <statement-1>
    .
    .
    <statement-N>
```

Method

The method is a function that is associated with an object. In Python, a method is not unique to class instances. Any object type can have methods.

Inheritance

Inheritance is the most important aspect of object-oriented programming which simulates the real world concept of inheritance. It specifies that the child object acquires all the properties and behaviors of the parent object.

By using inheritance, we can create a class which uses all the properties and behavior of another class. The new class is known as a derived class or child class, and the one whose properties are acquired is known as a base class or parent class.

It provides re-usability of the code.

Polymorphism

Polymorphism contains two words "poly" and "morphs". Poly means many and Morphs means form, shape. By polymorphism, we understand that one task can be performed in different ways. For example You have a class animal, and all animals speak. But they speak differently. Here, the "speak" behavior is polymorphic in the sense and depends on the animal. So, the abstract "animal" concept does not actually "speak", but specific animals (like dogs and cats) have a concrete implementation of the action "speak".

Encapsulation

Encapsulation is also an important aspect of object-oriented programming. It is used to restrict access to methods and variables. In encapsulation, code and data are wrapped together within a single unit from being modified by accident.

Data Abstraction

Data abstraction and encapsulation both are often used as synonyms. Both are nearly synonym because data abstraction is achieved through encapsulation.

Abstraction is used to hide internal details and show only functionalities. Abstracting something means to give names to things so that the name captures the core of what a function or a whole program does.

Object-oriented vs Procedure-oriented Programming languages

Index	Object-oriented Programming	Procedural Programming
1.	Object-oriented programming is the problem-solving approach and used where computation is done by using objects.	Procedural programming uses a list of instructions to do computation step by step.
2.	It makes the development and maintenance easier.	In procedural programming, It is not easy to maintain the codes when the project becomes lengthy.
3.	It simulates the real world entity. So real-world problems can be easily solved through oops.	It doesn't simulate the real world. It works on step by step instructions divided into small parts called

		functions.
4.	It provides data hiding. So it is more secure than procedural languages. You cannot access private data from anywhere.	Procedural language doesn't provide any proper way for data binding, so it is less secure.
5.	Example of object-oriented programming languages is C++, Java, .Net, Python, C#, etc.	Example of procedural languages are: C, Fortran, Pascal, VB etc.

Python Class and Objects

As we have already discussed, a class is a virtual entity and can be seen as a blueprint of an object. The class came into existence when it instantiated. Let's understand it by an example.

Suppose a class is a prototype of a building. A building contains all the details about the floor, doors, windows, etc. we can make as many buildings as we want, based on these details. Hence, the building can be seen as a class, and we can create as many objects of this class.

On the other hand, the object is the instance of a class. The process of creating an object can be called as instantiation.

In this section of the tutorial, we will discuss creating classes and objects in python. We will also talk about how an attribute is accessed by using the class object.

Creating classes in python

In python, a class can be created by using the keyword class followed by the class name. The syntax to create a class is given below.

Syntax

```
1. class ClassName:
2.     #statement_suite
```

In python, we must notice that each class is associated with a documentation string which can be accessed by using **<class-name>.__doc__**. A class contains a statement suite including fields, constructor, function, etc. definition.

Consider the following example to create a class Employee which contains two fields as Employee id, and name.

The class also contains a function display() which is used to display the information of the Employee.

Example

```
1. class Employee:
2.     id = 10;
3.     name = "ayush"
4.     def display (self):
5.         print(self.id,self.name)
```

Here, the self is used as a reference variable which refers to the current class object. It is always the first argument in the function definition. However, using self is optional in the function call.

Creating an instance of the class

A class needs to be instantiated if we want to use the class attributes in another class or method. A class can be instantiated by calling the class using the class name.

The syntax to create the instance of the class is given below.

1. <object-name> = <**class**-name>(<arguments>)

The following example creates the instance of the class Employee defined in the above example.

Example

```
1. class Employee:
2.     id = 10;
3.     name = "John"
4.     def display (self):
5.         print("ID: %d \nName: %s"%(self.id,self.name))
6. emp = Employee()
7. emp.display()
```

Output:

```
ID: 10
Name: ayush
```