Introduction to Object Oriented Programming

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

An object-oriented paradigm is to design the program using classes and objects. The object is related to real-word entities such as book, house, pencil, etc. The oops concept focuses on writing the reusable code. It is a widespread technique to solve the problem by creating objects.

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

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

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>
```

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 docstring defined in the function source code.

When we define a class, it needs to create an object to allocate the memory. Consider the following example.

For Example:

```
class car:
    def __init__(self, modelname, year):
        self.modelname = modelname
        self.year = year

    def display(self):
        print(self.modelname, self.year)


c1 = car("Toyota", 2000)
c1.display()
```

Output:

```
Toyota 2000
```

In the above example, we have created the class named car, and it has two attributes modelname and year. We have created a c1 object to access the class attribute. The c1 object will allocate memory for these values.

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 the re-usability of the code.

Polymorphism:

Polymorphism contains two words "poly" and "morphs". Poly means many, and morph means 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 a 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 essential 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 synonyms 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

The difference between object-oriented and procedure-oriented programming is given below:

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

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, rooms, 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 instantiation.

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:

```
class ClassName:
    #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.**

```
class Employee:
    id = 10
    name = "Nirajan"
    def display (self):
        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.

The self-parameter:

The self-parameter refers to the current instance of the class and accesses the class variables. We can use anything instead of self, but it must be the first parameter of any function which belongs to the class

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.

```
<object-name> = <class-name>(<arguments>)
```

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

```
class Employee:
    id = 7
    name = "Ramesh"
    Designation = "Cyber security Expert"
    Expr = "5 yrs"

    def display(self):
        print("ID: %d \nName: %s \nDesignation: %s \nExpr: %s" % (self.id, self.name, self.Designation, self.Expr))


# Creating a emp instance of Employee class
emp = Employee()
emp.display()
```

Output:

```
ID: 7
Name: Ramesh
Designation: Cyber security Expert
Expr: 5 yrs
```

In the above code, we have created the Employee class which has two attributes named id and name and assigned value to them. We can observe we have passed the self as parameter in display function. It is used to refer to the same class attribute.

We have created a new instance object named **emp.** By using it, we can access the attributes of the class.

Delete the Object:

We can delete the properties of the object or object itself by using the del keyword. Consider the following example.

```
class Employee:
    id = 10
    name = "Hari"

    def display(self):
        print("ID: %d \nName: %s" % (self.id, self.name))
        # Creating a emp instance of Employee class


emp = Employee()

# Deleting the property of object
del emp.id
# Deleting the object itself
del emp
emp.display()
```

Output:

```
Traceback (most recent call last):
  File "/home/nirajan/PycharmProjects/PythonExamples/DeletingTheObject.py", line 13, in <module>
    del emp.id
AttributeError: id
```

It will run through the Attribute error because we have deleted the object **emp**.