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 behaviour. 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.

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", 2020)                    #c1 is an object of class car
c1.display()
```

➢ 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 behaviours of the parent object.

By using inheritance, we can create a class which uses all the properties and behaviour 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 defines the ability to take different forms. Polymorphism in Python allows us to define methods in the child class with the same name as defined in their parent class.

➢ 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.

Python Class and Objects

➢ Creating classes in Python

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.**

Example:

```
class Employee:
    id = 10
    name = "Anjaly"
    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.

Example:

```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age
p1 = Person("Anjaly", 25)
print(p1.name)
print(p1.age)
```

Output:

```
Anjaly
25
```

- **Delete the Object**

Example:

```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age
```

```
    def myfunc(self):
      print("Hello my name is " + self.name)
p1 = Person("Anjaly", 25)
del p1.age
print(p1.age)
```

Output:

```
Traceback (most recent call last):
  File "C:\Users\lumin\PycharmProjects\Batch3\test.py", line 10, in <module>
    print(p1.age)
AttributeError: 'Person' object has no attribute 'age'
```