Object Oriented Programming in Python

Python programming paradigms

Dear learners! Welcome you to yet another important module on **object-oriented programming**.

So far, we've covered the fundamentals of Python programming, including how to create functions, modules, and packages. I hope by this time you have become familiar with working with various Python files.

Python's unique selling feature is that it supports a wide range of programming paradigms, including functional programming, procedure-oriented programming, and object-oriented programming. That is why it is popular.

So for in this course, you have learned about,

1. Functional programming - This means writing a program using built-in functions. Python provides a rich set of functions and methods for various computations.
2. Procedure-oriented programming - It simply contains a series of computational steps to be carried out. Writing a program using functions is the best example of this kind.

Let us now move on to Python's next major programming paradigm, Object-Oriented Programming.

In some of the previous modules, I mentioned the word “object” right!!! But we haven’t discussed it yet. Now you will learn about what is an object? And how is it important?

In general, we strive to solve real-world problems with virtual world programming solutions. Right!!!

We all know that every object in this world will have specific qualities, features, attributes and behaviors.

Everything in the real world is represented as an object, such as a laptop, a phone, people, birds, gadgets, and so on.

Each object has its own set of characteristics and functionalities. In general, variables are used to record the attributes of any object, and methods or functions are used to describe the behavior or operations connected with variables.

Python is an object-oriented programming language. It allows us to develop applications using an Object Oriented Programming approach. All this while you have learned functional oriented programming. You know how to write functions, modules and packages which are part of functional oriented programming. Now we are going to discuss object oriented programming and in order to understand, you need to know about some concepts.

To help you understand this better let's look at an example.

Look at the example here. I have created a class called cars wherein I have defined the most common parameters that the cars will be having, such as the seating capacity, price and year of manufacture. Now I have defined the blueprint but I need to implement an instance which is basically an entity of the class that is referred to as an object. If you look at it here, there are 2 different types of cars that have been considered here. The first object is swift with 5 seating capacity, its price is 5 lakhs and the year of manufacturing is 2014. Similarly the second object is Xylo with the seating capacity of 8 people, price is 10 lakhs and the year of manufacturing is 2016.

Remember a class is the blueprint from which specific objects are created. In this case 'cars' can be said as a class, swift and xylo are said to be an object. It imitates the general structure of the cars class.

In simple terms, a class is basically a logical grouping which helps us reuse or regroup our data as part of the code. A class is like a blueprint or a template. Something similar to a blueprint used before we construct a house which defines what will be constructed where. Once I have defined my requirements, I can add more variations to it. For example in the house I can have varied tiles.

Now let us start with the formal definition of class.

- 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, et c.
- Class variable – A variable that is shared by all instances of a class. Class variables are defined within a class but outside any of the class's methods. Class variables are not used as frequently as instance variables are.

Now let us start with the formal definition of class and objects.

- Class – A user-defined prototype for an object that defines a set of attributes that characterize any object of the class. The attributes are data members (class variables and instance variables) and methods, accessed via dot notation.
- 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.

Almost everything in Python is an object, with its properties and methods. An object is simply a collection of data (variables) and methods (functions) that act on those data.

A Class is like an object constructor, or a "blueprint" for creating objects.

Python allows us to imitate almost everything available in the real world to the virtual or programming world.

Now let us look at how to implement the class in python?

Class ClassName:

Class definition starts with a keyword class followed by the name of the class and ends with a semicolon.

Every class is useful to represent three aspects. It will have class members, data attributes and methods.

A class member or a class variable is a variable that is shared by all the instances of a class. Here the instance is referring to the objects of a particular class. We refer to objects as instances alternatively. If there is a need of sharing a common variable by all the objects or instances.

The basic syntax of a class

'Optional class documentation string or comments'

class ClassName:

class members
data attributes
and methods #All must be given in first level indentation

All the methods are used to specify the behaviour of an object.

Let us look into detailed steps involved in creating a class or writing a code in class definition.

Step 1. class ClassName:

Step 2. A constructor creation using __init__ method

Step 3. Creating other methods

Step 4. Working on object properties

Step 5. Cr eating a class variables

Step 6. Write a Drive Code

Step 7: Create an object or instance variable

Step 8: Method Call

Python is an object-oriented programming language in which everything is an object. We can model the real world using Python classes and objects.

Now that you have understood the class let's look at an example of the simple class

```
class laptop: #Class definition
    name='Apple Mac'  #Class Attribute/instance with indentation
print(laptop.name)  #Accessing a class attribute
```

The first step is to define a class here it's called a class laptop.

Following by 4 spaces of indentation

Then the object is defined with class attributes or instances.

Here the value assigned to the class variable is "Apple Mac"

Now to access the class attribute, you will call the class name.attribute name.

The print statement: print(laptop.name) produces the output as "Apple Mac".

Hope you have learned about how to write a simple class.

Let us see some of the terminologies related to OOPs concepts.

- Method – A special kind of function that is defined in a class definition.
- Object – A unique instance of a data structure that's defined by its class. An object comprises both data members (class variables and instance variables) and methods.
- Instantiation – The creation of an instance of a class.
- Instance – An individual object of a certain class. An object obj that belongs to a class Circle, for example, is an instance of the class Circle.
- Data member – A class variable or instance variable that holds data associated with a class and its objects.
- Instance variable – A variable that is defined inside a method and belongs only to the current instance of a class.
- Function overloading – The assignment of more than one behavior to a particular function. The operation performed varies by the types of objects or arguments involved.
- Inheritanc e – The transfer of the characteristics of a class to other classes that are derived from it.

- Operator overloading – The assignment of more than one function to a particular operator.

Now you will learn about how to create a constructor. It is an important notion of a class definition.

- The method __init__ simulates the constructor of the class. This method is called when the class is instantiated.
- We can pass any number of arguments at the time of creating the class object, depending upon __init__ defin ition. It is mostly used to initialize the class attributes.
- Every class must have a constructor, even if it simply relies on the default constructor.
- The self parameter is a reference to the current instance of the class and is used to access variables that belong to the class.
- It does not have to be named self , you can call it whatever you like, but it has to be the first parameter of any function in the class.

The other important thing is creating an object/objects or instances in the driver code.

The creation of an instance of a class or an object is called instantiation. A class can be compared to a rough idea (prototype) of a house architectural drawing. It covers all of the information regarding the floors, doors, and windows, among other things. We construct the house based on these descriptions. The object is a house. We can make numerous objects from a class, just as we can make many houses from a house's plan. A class instance is sometimes known as an object, and the process of creating the object is known as instantiation (technical term).

In simple terms, When you create an object, you **are creating an instance of a class**, therefore "instantiating" a class.

Let us look at the anatomy of the program using constructor.

Class Definition

```
class Student:
    def __init__ (self , name, percentage) :
        self.name = name
        self.percentage = percentage

def show (self):
        print("Name is:", self.name, "and percentage is:", self.percentage)
```

Driver Code

```
stud = Student("Jyoti", 80)
```

stud.show()

Let us look into how to write a class definition.
Student is the class name.
'Def' keyword, __init__ and self parameters are being used to create a constructor.

Where, 'Self' is the special parameter, whereas Name and percentage are the parameters to the constructor.

self.name = name

self.percentage = percentage

These are instance values.
Def show self is the instance method.
Followed by the print function to display the name and percentage.

At this point, the class creation ends.

Now let us look into how to write the driver code.
It is important to create an object in a driver class to all the methods and class variables defined in a class.
Here the object Stud is the object of a class. This object creation is called instantiation.
It is done with the name of the class and all the parameters of the constructor.

stud = Student("Jyoti", 80)

In this case, Student is the name of the class. The parameters "Jyothi" and the number 80 are passed to the __init__ constructor.

Now the object stud is ready to call the method stud.show()

It produces the output as

Name is: Jyoti and percentage is:80

I hope you are clear with the autonomy of the constructor.

Moving on to variable types in OOP.

There are two types of variables used in Python OOP.

- Instance Variable - The values can be changed from object to object. It is defined within __init__.
- Class Variable or Static Variable - A Python class/static variable is shared by all object instances of a class. Class variables are declared when a class is being constructed. class/static variables are declared inside a class, but outside the method. It can be called directly from a class but not through the instances of a class.

A namespace is a system that has a unique name for each and every object in Python. There

are two types of namespaces used in Python:

- Class Namespace
- Instance Namespace

Now you will learn one more program to create a class using a constructor. Constructors are useful in building templates or many objects.
The method __init__ simulates the constructor of the class. This method is called when the class is instantiated.
We can pass any number of arguments at the time of creating the class object, depending upon your problem requirement. It is mostly used to initialize the class attributes.
The self parameter is a reference to the current instance of the class, and is used to access the instance variables that belong to the class.
It does not have to be named self, you can call it whatever you like, but it has to be the first parameter of any function in the class.
It will be easier to understand with this example.

```
class ComplexNumber:
    def __init__(self, r, i): #Constructor with self and other parameter
        self.real = r  #Instance variable initiation
        self.imag = i #Instance variable initiation
    def get_data(self):   #Method
        print("The Complex Number:", self.real,'+', self.imag, 'j')
    def add_data(self):   #Method
        print("The addition is:", num1.real+num2.real, '+', num1.imag+num2.imag, 'j')
# Create a new ComplexNumber objects
num1 = ComplexNumber(2, 3)  #num1 Object instantiation
num2 = ComplexNumber(10, 20)  #num2 Object instantiation
# Call get_data() method & add_data()
num1.get_data()
num2.get_data()
num1.add_data()
num2.add_data()
```

Follow the code carefully.
Class Complex number, here complex name is the class name.
Whenever you see def__init__ in any code, then it's called a constructor. The def __init__ is the syntax for the constructor. The keyword self is used as a first parameter in the constructor followed by other parameters that can be defined.
The keyword **self** represents the instance of a class and binds the attributes with the given **arguments**.

The r and i parameters would be stored in variables on the stack and would be discarded when the init method goes out of scope. Setting those variables as self.real and self.imag sets those variables as members of the ComplexNumber object

So you will write self which is a keyword followed by a dot and any instance name, in this case it is self.real and self.imag equals to parameters r and i are respectively.

These two are attributes or data points that we have created. Real data point and imaginary data point.

In general, instance variables can have different values for each instance of the class, the class variables are the same across all instances created from that class.

What are we going to do with this data?
These instances are accessed by other methods within a class. Since at the time of defining the data attributes we used self parameter, therefore, we have access to the data attributes by using the keyword self.
We will get the data or access by writing. def get_data(self):
print("The Complex Number:", self.real,'+', self.imag, 'j')
Self is used to access variables that belong to the same class.
So through the get method I can access real and imag data.
print("The Complex Number:", self.real,'+', self.imag, 'j')
This is possible because I am using self.

Similarly
def add_data(self):
Using this method you can add the data accessed from the class using the self keyword.
print("The addition is:", num1.real+num2.real, '+', num1.imag+num2.imag, 'j')
Upto here it's called class definition.

Now we will look at the driver code.
Through the driver code the class data can be accessed.
So to do that, we create an object. which is instantiated or initialized by the class Complex Number.
In simple terms its object creation.
num1 = ComplexNumber(2, 3) #num1 Object instantiation
Num1 is the object that equals the Complex number which is the name of the class. The numbers 2 and 3 are passed as parameters to the constructor parameters r and i respectively.
This 2 & 3 will also be assigned to self.real and self.imag respectively.
Now we have assigned the values to the parameters.
Similarly let us create one more object called Num2.
Now this is assigned with the values 10 and 20 to r and i. It's similar to adding values to a

template.
In the memory, the value of r and i in num1 is 2 and 3 and similarly the value of r and i in num2 is 10 and 20.
Now we have the class and the values assigned to it by object instantiation, let us call the functions.
# Call get_data() method & add_data()
num1.get_data()
We will call the data method.
It executes the following code:
print("The Complex Number:", self.real,'+', self.imag, 'j')
The output is The complex number: 2+3j
Now we had assigned the values to the parameters in num2 and a method call
num2.get_data() produces the output as - The Complex Number: 10 + 20 j
Now in add data, since we have 2 separate objects num1 and num2, we can add the data.
num1.add_data()
What it does is it will add the value of num1.real and num2.real, in this case 2 and 10 for real and in the same way will add the value num1.imag and num2.imag, in this case 3 and 20 to get 23j.
Once it has added the values from both the objects, it will concatenate the two values using =plus sign and produces the output as
The addition is: 12 + 23 j

num2.add_data()

The addition is: 12 + 23 j

Since you have understood the self parameter, Now let us look at the same code block of self parameter with an alias name.

In this program, the self parameter is named as com_add.
Everywhere in the code, self has been replaced by com_add. It's a normal function as the previous example with the only specialty of com_add being used in place of the self parameter.

It does not have to be named self, you can call it whatever you like, but it has to be the first parameter of any function in the class.

The output remains the same as the previous program using the keyword 'self'.

Instances are individual objects of a certain class. Two types of instances are used in a class

- Accessor Methods - Only accessing or fetching values of the variables - you can access the data from the class constructor.
- Mutator Methods - Allows to modify the values of the variables - You can edit or modify

the data or the attribute value.

To better understand the concepts let us look at the Sample Program to demonstrate the Accessor & Mutator

```
 #Class Definition
class OnlineStudent:    #Class namespace
 platform ="vityarthi"  #Class or Static variable
  #It is common to all the objects
 def __init__(self, name, m1, m2, m3): #Constructor
    self.name = name #Instance Variable
    self.m1 = m1     #Instance Variable
    self.m2 = m2     #Instance Variable
    self.m3 = m3     #Instance Variable
 def average(self):   #A Method to calculate an average
     return((self.m1+self.m2+self.m3)/3)
 def get_data(self):   #Accessor to fetch the data
   return (self.name, self.m1, self.m2, self.m3)
 def modify_data(self, name, mark):  #Mutator to modify the data
   self.name = name
   self.m1 = mark
```

Here Onlinestudent is the class name or namespace.
Platform is initialized with the string " vityarthi", now this is the class or static variable.
This class variable is common to all the objects.
Now let us look into the class constructor.
`def __init__(self, name, m1, m2, m3): #Constructor`

The constructor is created using the syntax **def __init__(self,** followed by the other parameters.
It is suggested that self is used as it is in the constructor instead of any other name in its place.
Name, m1, m2, m3 are defined as instance variables using self.name, self.m1, self.m2, self.m3.
This will make the data available and common to all the objects.
Now let's reinforce the definition. Within the class outside the method whatever you are creating is called static method and whatever objects you define within the constructor is called instance variable.
Now we will look at one of the methods to calculate the average.
The data to this method is provided by a self parameter.
Now the average will be calculated by fetching data from the instance variable and returning the value.
Similarly, get_data will fetch the data from the accessor. The accessor data will only get the data whereas the mutator will modify the data in case of **def modify_data(self, name, mark):**

This will change the name and mark, only two parameters will be changed.
Now let us look into the driver code and the object creation.

```
#Object Creation
student1 = OnlineStudent('Jesse',56,67,78)
student2 = OnlineStudent('Jose',67,78,89)
print(OnlineStudent.platform, "Students Score Card")
print("The Student 1 Average is", student1.average())
print("The Student 2 Average is", student2.average())
print("The Student1 details are accessed",student1.get_data())
print("The Student2 details before modification",student2.get_data())
student2.modify_data('David', 100)
print("The Student2 details after modification",student2.get_data())
```

Now I have created one object called student1 with the name Jesse and her marks.
Student 2 Jose and his marks.
To access the data we will have to first access the class followed by a class variable or static variable.
Output:
vityarthi Students Scorecard

We are calling the average of student1 and student2 by passing the values from jesse and jose marks. The formula to calculate the average is called the front he constructor.
Output is The Student 1 Average is 67.0
The Student 2 Average is 78.0
Now we want to access the student1 data. We can do that through the get_data method.
The output will be The Student1 details are accessed ('Jesse', 56, 67, 78)
The Student2 details before modification ('Jose', 67, 78, 89)
Now we want to modify data. This can be done through the modify data which is a method of mutator. All this while we were accessing the data so it is called accessor.

```
student2.modify_data('David', 100)
```

With this code I on;y want to change the name and first data.
In the class definition, this can be done through

```
self.name = name
self.m1 = mark
```

NowJose has been modified to david and his first mark is changed to 100.
The output of the modification is
The Student2 details are displayed after the modification ('David', 100, 78, 89)