Python Object Oriented Programming Language (OOPL) Concepts:

- Generally all Programming languages are basically classified into two types.
  1. Functional Oriented Programming.
  2. Object Oriented Programming.
- If you want to develop one-to-one applications then use functional oriented programming.
  For example, C-language
- To develop operating system, drivers, compilers we have to use functional programming.
- If you want to develop one-to-many applications then we have to use object oriented programming.
  For example, Python, Java, C#.net, ...etc
- To develop web applications like Amazon, Swiggy, Irctc, etc we have to use object oriented programming.
- In 'OOPL' we have to develop programs by using **class** and **objects**.
- Like other general-purpose programming languages, Python is also an object-oriented programming 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.

Some general words which we are using in oops concepts,

Class
Object
Method
Variables
self
constructor

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

- Inheritance
- Polymorphism
- Encapsulation
- Data Abstraction

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.

```
class ClassName:
        <statement-1>
         --------------
        <statement-N>
object1 = ClassName()
```

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.

Example:

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

    def display(self):
        print("The Car modelname is :", self.modelname)
        print("The Car launch year :",self.year)

c1 = car("Toyota", 2016)
c1.display()
```

Output:

```
Toyota 2016
```

--->> 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

--->> A method is a set of statements to perform specific operations on data.

--->> 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.

Example:

```
def  display(self):
        print('Employee id:101)
```

Examples:

Q. Create a class with some variables and methods. How to display all members of class?

```
class  Sample:
    a = 10
    b = 20
    def message(self):
        print('hello')
        return 'ok'
obj = Sample()
print('a value is :',obj.a)
print('b value is :',obj.b)
print(obj.message())
```

Q. Ho to create a Employee data and display employee data?

```
class Employee:
    def create(self):
        self.eid = 101
        self.ename = 'Rumi'
        self.salary = 10000

    def show(self):
        print('Employee id :',self.eid)
        print('Employee Name :',self.ename)
        print('Employee Salary:', self.salary)

emp = Employee()  #  object creation
emp.create()      #  create the data using object
emp.show()        #  access the data from object
```

Q. How to create default constructor and execute with some data?

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

    def display(self):
        print(self.modelname,self.year)
c1 = Car('Benz', 2020)
c2 = Car('BMW', 2019)
c1.display()
c2.display()
```

Q. How to create the default concstructor and execute automatically ?

```
class Car:
    def __init__(self):
       print('iam from constructor')

    def display(self):
        print('hello')
c1 = Car()
c1.display()
```

Output:

```
iam from constructor
hello
```