What is Object-Oriented Programming?

- Object Oriented Programming **is a way of computer programming** using **the idea of "objects" to represents data and methods**.
-
- It is also, an approach used for **creating neat and reusable code instead of a redundant one**.
-
- Every **Individual object represents a different part of the application having its own logic and data to communicate within themselves**.

Now, to get a clearer picture of why we use oops instead of pop, I have listed down the differences below.

OOP	**POP**
Object-oriented Programming is a programming language **that uses classes and objects to create models based on the real world environment**. In OOPs it makes it **easy to maintain and modify existing code** as new objects are created inheriting characteristics from existing ones.	On other hand Procedural Oriented Programming is a programming language that **follows a step-by-step approach to break down a task into a collection of variables and routines** (or subroutines) through a sequence of instructions. **Each step is carried out in order in a systematic manner** so that a computer can understand what to do.
In OOPs concept of **objects and classes is introduced and hence the program is divided into small chunks called objects which are instances of classes**.	On other hand in case of POP the main program is divided into small parts based on the functions and **is treated as separate program for individual smaller program**.
Program is divided into objects	Program is divided into functions
Makes use of Access modifiers: 'public', private', protected'	Doesn't use Access modifiers
It is more secure	It is less secure
Object can move freely within member functions	Data can move freely from function to function within programs
It supports inheritance	It does not support inheritance

What are Python OOPs Concepts?

Major OOP (object-oriented programming) concepts in Python include:

1. Class,
2. Object,
3. Method,
4. Inheritance,
5. Polymorphism,
6. Data Abstraction, and
7. Encapsulation.

What are Classes and Objects?

A class is a collection of objects or you can say it is a blueprint of objects defining the common attributes and behavior.

Objects are an instance of a class. It is an entity that has state and behavior. In a nutshell, it is an instance of a class that can access the data.

Now the question arises, how do you do that?

Well, it logically groups the data in such a way that code reusability becomes easy. I can give you a real-life example- think of an office going 'employee' as a class and all the attributes related to it like 'emp_name', 'emp_age', 'emp_salary', 'emp_id' as the objects in Python.

In real life, there are millions but just to name a couple of them:

a) "Human being" -> A class

"Man" -> An object of "Human being"

"Woman" -> An object of "Human being"

b) "Son" -> A class

"My eldest son" -> An object of "Son"

"My youngest son" -> An object of "Son"

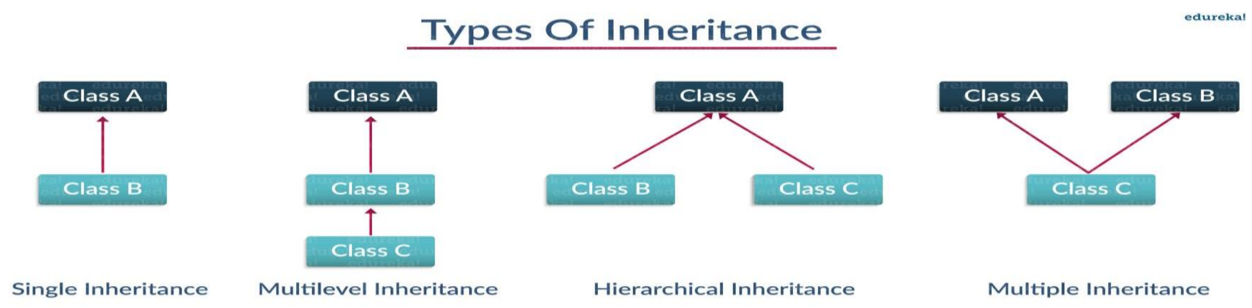
```
class employee():
    def __init__(self,name,age,id,salary):   //creating a function - Constructor
        self.name = name // self is an instance of a class
        self.age = age
        self.salary = salary
        self.id = id

emp1 = employee("harshit",22,1000,1234) //creating objects
emp2 = employee("arjun",23,2000,2234)
print(emp1.__dict__)//Prints dictionary
```

self is used to refer to the specific object that is calling that function

Inheritance

Ever heard of this dialogue from relatives **"you look exactly like your father/mother" the reason behind this is called 'inheritance'**. From the Programming aspect, It generally means **"inheriting or transfer of characteristics from parent to child class without any modification"**. **The new class is called the derived/child class and the one from which it is derived is called a parent/base class**.



Single Inheritance: Single inheritance enables a derived class to inherit properties from a single parent class, thus enabling code reusability and the addition of new features to existing code.

```
# A Python program to demonstrate
# inheritance



# Base class or Parent class
class Child:

    # Constructor
    def __init__(self, name):
        self.name = name

    # To get name
    def getName(self):
        return self.name

    # To check if this person is student
    def isStudent(self):
        return False

# Derived class or Child class
class Student(Child):

    # True is returned
    def isStudent(self):
        return True



# Driver code
# An Object of Child
std = Child("Ram")
print(std.getName(), std.isStudent())

# An Object of Student
std = Student("Shivam")
print(std.getName(), std.isStudent())
```

Multiple Inheritance: When a class can be derived from more than one base class this type of inheritance is called multiple inheritance. In multiple inheritance, all the features of the base classes are inherited into the derived class.

```
# Python program to demonstrate
# multiple inheritance



# Base class1
class Mother:
    mothername = ""
    def mother(self):
        print(self.mothername)

# Base class2
class Father:
    fathername = ""
    def father(self):
        print(self.fathername)

# Derived class
class Son(Mother, Father):
    def parents(self):
        print("Father :", self.fathername)
        print("Mother :", self.mothername)

# Driver's code
s1 = Son()
s1.fathername = "RAM"
s1.mothername = "SITA"
s1.parents()
```

Multilevel Inheritance:
In multilevel inheritance, features of the base class and the derived class are further inherited into the new derived class. This is similar to a relationship representing a child and grandfather.

```
# Python program to demonstrate
# multilevel inheritance

# Base class
class Grandfather:

    def __init__(self, grandfathername):
        self.grandfathername = grandfathername

# Intermediate class
class Father(Grandfather):
    def __init__(self, fathername, grandfathername):
        self.fathername = fathername

        # invoking constructor of Grandfather class
        Grandfather.__init__(self, grandfathername)

# Derived class
class Son(Father):
    def __init__(self,sonname, fathername, grandfathername):
        self.sonname = sonname

        # invoking constructor of Father class
        Father.__init__(self, fathername, grandfathername)

    def print_name(self):
        print('Grandfather name :', self.grandfathername)
        print("Father name :", self.fathername)
        print("Son name :", self.sonname)

#  Driver code
s1 = Son('Prince', 'Rampal', 'Lal mani')
print(s1.grandfathername)
s1.print_name()
```

Hierarchical Inheritance: When more than one derived classes are created from a single base this type of inheritance is called hierarchical inheritance. In this program, we have a parent (base) class and two child (derived) classes.

```
# Python program to demonstrate
# Hierarchical inheritance



# Base class
class Parent:
      def func1(self):
          print("This function is in parent class.")

# Derived class1
class Child1(Parent):
      def func2(self):
          print("This function is in child 1.")

# Derivied class2
class Child2(Parent):
      def func3(self):
          print("This function is in child 2.")

# Driver's code
object1 = Child1()
object2 = Child2()
object1.func1()
object1.func2()
object2.func1()
object2.func3()
```

Hybrid Inheritance: Inheritance consisting of multiple types of inheritance is called hybrid inheritance.

```
# Python program to demonstrate
# hybrid inheritance



class School:
     def func1(self):
         print("This function is in school.")

class Student1(School):
     def func2(self):
         print("This function is in student 1. ")

class Student2(School):
     def func3(self):
         print("This function is in student 2.")

class Student3(Student1, School):
     def func4(self):
         print("This function is in student 3.")

# Driver's code
object = Student3()
object.func1()
object.func2()
```

Polymorphism

You all must have used GPS for navigating the route, Isn't it amazing how many different routes you come across for the same destination depending on the traffic, from a programming point of view this is called 'polymorphism'. It is one such OOP methodology where one task can be performed in several different ways. To put it in simple words, it is a property of an object which allows it to take multiple forms.