**https://www.edureka.co/blog/object-oriented-programming-python/**

**https://medium.com/@manjuladube/encapsulation-abstraction-35999b0a3911**

**https://www.javatpoint.com/how-long-does-it-take-to-learn-python**

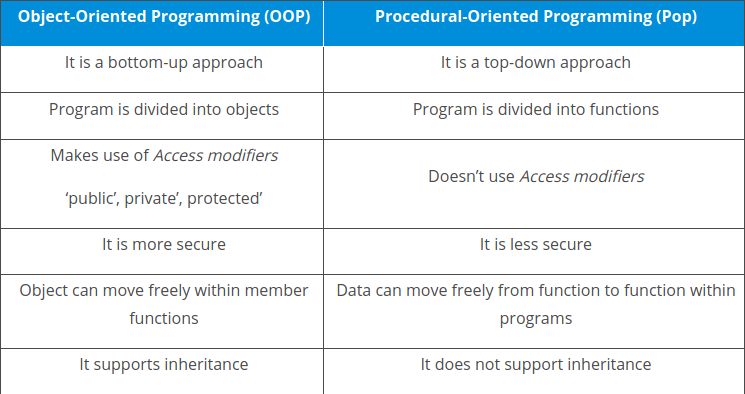
**https://medium.com/@manjuladube/encapsulation-abstraction-35999b0a3911**

1. **What is Object-Oriented Programming? (OOPs concepts in Python)**

Object Oriented Programming is a way of computer programming using the idea of “[objects](https://www.edureka.co/blog/python-class/" \l "Objects)” to represents data and methods. It is also, an approach used for creating neat and reusable code instead of a redundant one. the program is divided into self-contained objects or several mini-programs. Every Individual object represents a different part of the application having its own logic and data to communicate within themselves.

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

1. Difference between Object-Oriented and Procedural Oriented Programming



1. What are Python OOPs Concepts?

Major OOP (object-oriented programming) concepts in Python include Class, Object, Method, Inheritance, Polymorphism, Data Abstraction, and Encapsulation.

1. **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. 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 [P](https://www.edureka.co/blog/python-programming-language)y

Example:

**class class1(): // class 1 is the name of the class**

Note: Python is not case-sensitive.

### **Objects:**

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.

Syntax:

obj = class1()

Here obj is the “object “ of class1.

### **Creating an Object and Class in python:**

Example:

class employee():

    def \_\_init\_\_(self,name,age,id,salary):   //creating a function

        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

’employee’.Here, the word (\_\_dict\_\_) is a “dictionary” which prints all the values of object ‘emp1’ against the given parameter (name, age, salary).(\_\_init\_\_) acts like a constructor that is invoked whenever an object is created.

I hope now you guys won’t face any problem while dealing with ‘classes’ and ‘objects’ in the future.

With this, let me take you through a ride of [Object Oriented Programming](https://www.edureka.co/blog/python-class/) methodologies:

[thon](https://www.edureka.co/blog/python-programming-language). Let us see from the coding perspective that how do you instantiate a class and an object.

Class is defined under a “Class” Keyword.

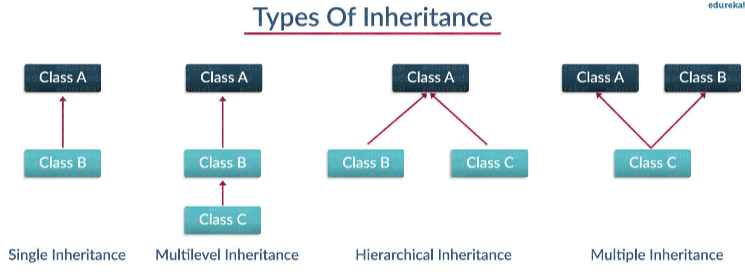
## **Object-Oriented Programming methodologies:**

Object-Oriented Programming methodologies deal with the following concepts.

* 1. Inheritance
  2. Polymorphism
  3. Encapsulation
  4. Abstraction

### **Inheritance:**

Ever heard of this dialogue from relatives “you look exactly like your father/mother” the reason behind this is called ‘[inheritance](_blank)’. 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.



Let us understand each of the subtopics in detail.

### **a) Single Inheritance:**

Single level inheritance enables a derived class to inherit characteristics from a single parent class.

Example:

**class employee1()://This is a parent class**

def \_\_init\_\_(self, name, age, salary):

self.name = name

self.age = age

self.salary = salary

**class childemployee(employee1)://This is a child class**

def \_\_init\_\_(self, name, age, salary,id):

self.name = name

self.age = age

self.salary = salary

self.id = id

**emp1 = employee1('harshit',22,1000)**

print(emp1.age)

Output**: 22**

Explanation:

* + I am taking the parent class and created a constructor (\_\_init\_\_), class itself is initializing the attributes with parameters(‘name’, ‘age’ and ‘salary’).
  + Created a child class ‘childemployee’ which is inheriting the properties from a parent class and finally instantiated objects ’emp1′ and ’emp2′ against the parameters.
  + Finally, I have printed the age of emp1. Well, you can do a hell lot of things like print the whole dictionary or name or salary.

### **b) Multilevel Inheritance: (hair problem’baalo ka girna’)**

Multi-level inheritance enables a derived class to inherit properties from an immediate parent class which in turn inherits properties from his parent class.

Example:

**class employee()://Super class**

def \_\_init\_\_(self,name,age,salary):

self.name = name

self.age = age

self.salary = salary

**class childemployee1(employee)://First child class**

def \_\_init\_\_(self,name,age,salary):

self.name = name

self.age = age

self.salary = salary

**class childemployee2(childemployee1)://Second child class**

def \_\_init\_\_(self, name, age, salary):

self.name = name

self.age = age

self.salary = salary

**emp1 = employee('harshit',22,1000)**

**emp2 = childemployee1('arjun',23,2000)**

**print(emp1.age)**

**print(emp2.age)**

**Output: 22,23**

Explanation:

* It is clearly explained in the code written above, Here I have defined the superclass as employee and child class as childemployee1. Now, childemployee1 acts as a parent for childemployee2.
* I have instantiated two objects ’emp1′ and ’emp2′ where I am  passing the parameters “name”, “age”, “salary” for emp1 from superclass  “employee” and “name”, “age, “salary” and “id” from the parent class “childemployee1”

### **Hierarchical Inheritance:**

Hierarchical level inheritance enables more than one derived class to inherit properties from a parent class.

**Example:**

**class employee():**

def \_\_init\_\_(self, name, age, salary): //Hierarchical Inheritance

self.name = name

self.age = age

self.salary = salary

**class childemployee1(employee):**

def \_\_init\_\_(self,name,age,salary):

self.name = name

self.age = age

self.salary = salary

**class childemployee2(employee):**

def \_\_init\_\_(self, name, age, salary):

self.name = name

self.age = age

self.salary = salary

**emp1 = employee('harshit',22,1000)**

**emp2 = employee('arjun',23,2000)**

**print(emp1.age)**

**print(emp2.age)**

**Output: 22,23**

### **Explanation:**

* In the above example, you can clearly see there are two child class “childemployee1” and “childemployee2”. They are inheriting functionalities from a common parent class that is “employee”.
* Objects ’emp1′ and ’emp2′ are instantiated against the parameters ‘name’, ‘age’, ‘salary’.

### **Multiple Inheritance:**

Multiple level inheritance enables one derived class to inherit properties from more than one base class.

Example:

**class employee1()://Parent class**

def \_\_init\_\_(self, name, age, salary):

self.name = name

self.age = age

self.salary = salary

**class employee2()://Parent class**

def \_\_init\_\_(self,name,age,salary,id):

self.name = name

self.age = age

self.salary = salary

self.id = id

**class childemployee(employee1,employee2):**

def \_\_init\_\_(self, name, age, salary,id):

self.name = name

self.age = age

self.salary = salary

self.id = id

**emp1 = employee1('harshit',22,1000)**

**emp2 = employee2('arjun',23,2000,1234)**

**print(emp1.age)**

**print(emp2.id)**

**Output: 22,1234**

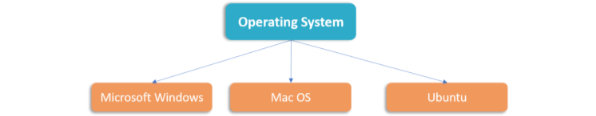
**Explanation**:

In the above example, I have taken two parent class “employee1” and “employee2”.And a child class “childemployee”, which is inheriting both parent class by instantiating the objects ’emp1′ and ’emp2′ against the parameters of parent classes.

This was all about inheritance, moving ahead in Object-Oriented Programming Python, let’s take a deep dive in ‘[polymorphism](_blank)‘.

1. 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.



Polymorphism is of two types:

* Compile-time Polymorphism
* Run-time Polymorphism

#### **Compile-time Polymorphism:**

A compile-time polymorphism also called as static polymorphism which gets resolved during the compilation time of the program. One common example is “method overloading”. Let me show you a quick example of the same.

Example:

**class** **employee1():**

def name(self):

print("Harshit is his name")

def salary(self):

print("3000 is his salary")

def age(self):

print("22 is his age")

**class employee2():**

def name(self):

print("Rahul is his name")

def salary(self):

print("4000 is his salary")

def age(self):

print("23 is his age")

**def func(obj)://Method Overloading**

**obj.name()**

**obj.salary()**

**obj.age()**

**obj\_emp1 = employee1()**

**obj\_emp2 = employee2()**

**func(obj\_emp1)**

**func(obj\_emp2)**

Output:

Harshit is his name  
3000 is his salary  
22 is his age  
Rahul is his name  
4000 is his salary  
23 is his age

1. Dfd
2. fd
3. fdf
4. df
5. df
6. df
7. d
8. fdf
9. df
10. ddf