Python Inheritance and OOPs Fundamentals with Examples

What Does Inheritance Mean In OOP?

Inheritance is the core feature of object-oriented programming which extends the functionality of an existing class by adding new features. You may compare it with real-life situations when a child inherits the property of his parents in addition to adding his own. He may even derive the surname (the second name) from his parents.

What Is The Purpose Of Inheritance?

By using the inheritance feature, we can have a new blueprint with old attributes but without making any changes to the original one. We refer to the new class as the derived or child class whereas the old one becomes the base or parent class.

How To Implement Inheritance In Python?

You can introduce inheritance by using the following syntax.

class ParentClass:

Parent class attributes

Parent class methods

class ChildClass(ParentClass):

Child class attributes

Child class methods

Inheritance automatically brings reusability to your code as the derived class has got everything from the base class.

Give An Example Of Python Inheritance

To understand the application of inheritance, let’s consider the following example.

Create A Base Class Taxi And A Subclass Vehicle

We have a **base class Taxi**, and it has a subclass (child) Vehicle.

class Taxi:

def \_\_init\_\_(self, model, capacity, variant):

self.\_\_model = model

# \_\_model is private to Taxi class

self.\_\_capacity = capacity

self.\_\_variant = variant

def getModel(self):

# getmodel() is accessible outside the class

return self.\_\_model

def getCapacity(self): # getCapacity() function is accessible to class Vehicle

return self.\_\_capacity

def setCapacity(self, capacity): # setCapacity() is accessible outside the class

self.\_\_capacity = capacity

def getVariant(self): # getVariant() function is accessible to class Vehicle

return self.\_\_variant

def setVariant(self, variant): # setVariant() is accessible outside the class

self.\_\_variant = variant

class Vehicle(Taxi):

def \_\_init\_\_(self, model, capacity, variant, color):

# call parent constructor to set model and color

super().\_\_init\_\_(model, capacity, variant)

self.\_\_color = color

def vehicleInfo(self):

return self.getModel() + " " + self.getVariant() + " in " + self.\_\_color + " with " + self.getCapacity() + " seats"

# In method getInfo we can call getmodel(), getCapacity() as they are

# accessible in the child class through inheritance

v1 = Vehicle("i20 Active", "4", "SX", "Bronze")

print(v1.vehicleInfo())

print(v1.getModel()) # Vehicle has no method getModel() but it is accessible via Vehicle class

v2 = Vehicle("Fortuner", "7", "MT2755", "White")

print(v2.vehicleInfo())

print(v2.getModel()) # Vehicle has no method getModel() but it is accessible via Vehicle class

Please note that we have not specified the getName() method in the Vehicle class, but we can access it. It is because the Vehicle class inherits it from the Taxi class.

The output of the above example is as follows.

# output

i20 Active SX in Bronze with 4 seats

i20 Active

Fortuner MT2755 in White with 7 seats

Fortuner

Python Inheritance’s UML Diagram

To bring more clarity, you can refer the below Python inheritance’s UML diagram of the example mentioned above.

The Super() Method

What Does The Super() Do In Python?

The **super()** method allows us to access the inherited methods that cascade to a class object.

In the earlier example, we’ve used the **super() method** in the constructor of the child class **<Vehicle>**. It is invoking the function of the base class <Taxi>.

How Does The Super() Method Work?

Just assume, if you have to invoke a method in the base class, i.e., **vehicleInfo()** defined in the child class, then you can use the following code.

super().vehicleInfo()

Likewise, you can invoke the base class constructor from the sub (child) class **\_\_init\_\_** using the below code.

super().\_\_init\_\_()

## **Hybrid Inheritance**

Hybrid inheritance involves multiple inheritance taking place in a single program.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | class Parent:       def func1(self):           print("this is function one")    class Child(Parent):       def func2(self):           print("this is function 2")    class Child1(Parent):       def func3(self):           print(" this is function 3"):    class Child3(Parent , Child1):       def func4(self):           print(" this is function 4")    ob = Child3()  ob.func1() |