## Module 12 – Inheritance

Contents

[Module 12 – Inheritance 1](#_Toc216013554)

[Inheritance Overview 2](#_Toc216013555)

[Parent Class 2](#_Toc216013556)

[Child Class 2](#_Toc216013557)

[Advantages of Inheritance 2](#_Toc216013558)

[Multiple Inheritance 3](#_Toc216013559)

[Multiple inheritance example: 3](#_Toc216013560)

[Multilevel Inheritance 4](#_Toc216013561)

[Multilevel inheritance example: 4](#_Toc216013562)

[Hierarchical Inheritance 5](#_Toc216013563)

[Hierarchical Inheritance example: 5](#_Toc216013564)

[Hybrid Inheritance 6](#_Toc216013565)

[Hybrid Inheritance example: 7](#_Toc216013566)

[Method Overriding 7](#_Toc216013567)

[Method Overriding Example: 8](#_Toc216013568)

[Reference and Additional Materials: 10](#_Toc216013569)

## Inheritance Overview

[Inheritance](https://youtu.be/u1be7Vele5o?si=kGVtSd8Jxuv35r-d) allows you to create a new class based on an existing class, so the new class automatically gets all the attributes and methods of the original. The class that is being inherited from is called the [parent class](https://www.w3schools.com/python/python_inheritance.asp#:~:text=Create%20a%20Parent%20Class), and the class that inherits from it is the child class. The [child class](https://www.w3schools.com/python/python_inheritance.asp#:~:text=Create%20a%20Child%20Class) can access everything from the parent class and can also add its own unique features. You can even modify or override parts of the parent class if needed. This gives you the power to reuse existing functionality while still customizing the behavior for different use cases.

## Parent Class

The [parent](https://www.geeksforgeeks.org/inheritance-in-python/#creating-a-parent-class:~:text=class%20unless%20overridden.-,Creating%20a%20Parent%20Class,-In%20object%2Doriented) class, also called the base class, is the class being inherited from. It contains the common things that other classes can share. For example, in a store inventory system there could be a parent class named Item that stores things like name and quantity. These are basic details that all items in the store would have.

## Child Class

The [child](https://www.geeksforgeeks.org/inheritance-in-python/#creating-a-child-class:~:text=of%20the%20person.-,Creating%20a%20Child%20Class,-A%20child%20class) class, also called the derived class, is the class that inherits from the parent class. It gets all the things from the parent class and can also have features of its own. For example, a child class called Fruit or Vegetables. These classes would still have name and quantity from the Item class, but might also have an expiration\_date since fruit and vegetables can go bad.

## Advantages of Inheritance

There are many [advantages](https://trainings.internshala.com/blog/python-inheritance/#:~:text=Use%20Case%20of%20Inheritance%20in%20Python) in using inheritance. One main advantage of inheritance is code reusability. Instead of writing the same code again and again, a parent class with common features can be created and then the other classes inherit from it. This means the child classes automatically get the functions and attributes of the parent class, and more features can still be added to make it different as needed. Inheritance also reduces the programmer’s effort because once a feature is written in the parent class it doesn't have to be written in every new class that needs it. This saves time and makes coding faster. Another benefit is better [readability](https://www.datacamp.com/tutorial/python-inheritance#:~:text=and%20Athlete).-,Benefits%20of%20Inheritance%20in%20Python,-Now%2C%20it%27s%20time). When inheritance is used the script looks more clean and organized making it easier to understand and manage.

## Multiple Inheritance

[Multiple Inheritance](https://www.geeksforgeeks.org/multiple-inheritance-in-python/) is when a class inherits from more than one parent class. This means the child class can get features from multiple sources, combining them into one. This type of [inheritance](https://youtu.be/Q8YlYHjksLo?si=IESr5j8d8O1Z3PY4) can be useful when a class needs to bring together different sets of features from different places. [Multiple](https://www.programiz.com/python-programming/multiple-inheritance) inheritance helps make programs more flexible by allowing a mix and match of features from different classes to create more complex objects.

Multiple Inheritance syntax would look something like this:

class parent\_1:

pass

class parent\_2:

pass

class child(parent\_1,parent\_2):

pass

obj = child()

### Multiple inheritance example:

class Brands: #parent class 1

brand\_name\_1 = "Amazon"

brand\_name\_2 = "Ebay"

brand\_name\_3 = "OLX"

class Products: # parent class 2

prod\_1 = "Online Ecommerce Store"

prod\_2 = "Online Store"

prod\_3 = "Online Buy Sell Store"

class Popularity(Brands,Products): # child class that is inherited from 2 parent

prod\_1\_popularity = 100 # attributes of child class

prod\_2\_popularity = 70

prod\_3\_popularity = 60

obj\_1 = Popularity() #Object\_creation

print(obj\_1.brand\_name\_1+" is an "+obj\_1.prod\_1 + "is the popularity of " + str(obj\_1.prod\_1\_popularity))

print(obj\_1.brand\_name\_2+" is an "+obj\_1.prod\_2 + "is the popularity of " + str(obj\_1.prod\_2\_popularity))

print(obj\_1.brand\_name\_3+" is an "+obj\_1.prod\_3 + "is the popularity of " + str(obj\_1.prod\_3\_popularity))

## Multilevel Inheritance

[Multilevel](https://www.geeksforgeeks.org/multilevel-inheritance-in-python/) inheritance is when a class is created by inheriting from a class that is already a child class. In other words, it’s like a chain of inheritance. For example, class A is the parent class, class B inherits from A, and then class C inherits from B. This forms a multilevel structure where class C ends up with all the features of both A and B. [Multilevel inheritance](https://youtu.be/J2_DHndMW9s?si=RwVxqjXqXQHaeFEJ) can go as deep as needed, meaning you can keep building on top of previous classes. The child class at the bottom of the chain will have access to everything from all the classes above it.

Multilevel syntax would look something like this:

class A:

pass

class B(A):

pass

class C(B):

pass

obj = C()

### Multilevel inheritance example:

class Brands: # Parent class

brand\_name\_1 = "Amazon"

brand\_name\_2 = "Ebay"

brand\_name\_3 = "OLX"

class Products(Brands): # Child class

prod\_1 = "Online Ecommerce Store"

prod\_2 = "Online Store"

prod\_3 = "Online Buy Sell Store"

class Popularity(Products): # Grandchild class

prod\_1\_popularity = 100

prod\_2\_popularity = 70

prod\_3\_popularity = 60

obj\_1 = Popularity() # Object creation

print(obj\_1.brand\_name\_1 + " is an " + obj\_1.prod\_1 + " with popularity of " + str(obj\_1.prod\_1\_popularity))

print(obj\_1.brand\_name\_2 + " is an " + obj\_1.prod\_2 + " with popularity of " + str(obj\_1.prod\_2\_popularity))

print(obj\_1.brand\_name\_3 + " is an " + obj\_1.prod\_3 + " with popularity of " + str(obj\_1.prod\_3\_popularity))

## Hierarchical Inheritance

[Hierarchical](https://www.geeksforgeeks.org/hierarchical-inheritance-with-examples-in-python/) inheritance happens when more than one child class inherits from the same parent class. In this type of inheritance, a single parent class shares its features with multiple child classes. Each child class gets access to the common attributes and methods of the parent class, but it can also have its own special behavior. For example, a parent class A, and then child classes B, C, and D all inherit from A. This means B, C, and D will all have whatever is in A, but they can also do different things on their own. [Hierarchical inheritan](https://openstax.org/books/introduction-python-programming/pages/13-4-hierarchical-inheritance)ce is useful when you want to create several different classes that all have some shared functionality from one main class, without having to repeat code.

Hierarchical syntax would look something like this:

class A: #parent\_class

pass

class B(A): #child\_class

pass

class C(A): #child\_class

pass

class D(A): #child\_class

pass

obj\_1 = B() #Object\_creation

obj\_2 = C()

obj\_3 = D()

### Hierarchical Inheritance example:

class Brands: #parent\_class

brand\_name\_1 = "Amazon"

brand\_name\_2 = "Ebay"

brand\_name\_3 = "OLX"

class Products(Brands): #child\_class

prod\_1 = "Online Ecommerce Store"

prod\_2 = "Online Store"

prod\_3 = "Online Buy Sell Store"

class Popularity(Brands): #grand\_child\_class

prod\_1\_popularity = 100

prod\_2\_popularity = 70

prod\_3\_popularity = 60

class Value(Brands):

prod\_1\_value = "Excellent Value"

prod\_2\_value = "Better Value"

prod\_3\_value = "Good Value"

obj\_1 = Products() #Object\_creation

obj\_2 = Popularity()

obj\_3 = Value()

## Hybrid Inheritance

[Hybrid](https://www.codechef.com/learn/course/oops-concepts-in-python/CPOPPY03/problems/ADVPPY19) inheritance is a type of inheritance that combines two or more other types of inheritance in a single program. It can include any mix of single, multiple, multilevel, or hierarchical inheritance. This approach allows for more complex and flexible class structures by enabling classes to share and organize features in different ways. [Hybrid](https://youtu.be/3HlxsEGlHFA?si=z3K_JfXko7W-i9hM) inheritance is especially useful when building systems that require overlapping or related features across several classes. It provides the ability to design more advanced relationships between classes without being restricted to one specific inheritance pattern.

Hybrid syntax would look something like this:

class PC:

pass

class Laptop(PC):

pass

class Mouse(Laptop):

pass

class Student3(Mouse, Laptop):

pass

obj = Student3()

### Hybrid Inheritance example:

class PC:

def fun1(self):

print("This is PC class")

class Laptop(PC):

def fun2(self):

print("This is Laptop class inheriting PC class")

class Mouse(Laptop):

def fun3(self):

print("This is Mouse class inheriting Laptop class")

class Student(Mouse, Laptop):

def fun4(self):

print("This is Student access class inheriting PC and Laptop")

# Driver’s code

obj = Student()

obj1 = Mouse()

obj.fun4()

obj.fun3()

## Method Overriding

[Method overriding](https://www.geeksforgeeks.org/method-overriding-in-python/) occurs when a child class defines a method with the same name and parameters as a method in its parent class. The method in the child class replaces, or overrides, the one inherited from the parent class. This allows the child class to provide its own specific implementation while keeping the method name consistent. When the method is called on an object of the child class, the version defined in the child class is used instead of the one from the parent. [Method overriding](https://youtu.be/4Y83cUbDKZ8?si=hK__YIO4GJ833h0W) is especially useful when multiple classes share common behavior but require different outcomes for the same action. It supports code reuse while allowing customization, which is a key principle of object-oriented programming.

### Method Overriding Example:

class AnimalData:

def \_\_init\_\_(self):

self.full\_name = ''

self.age\_years = 0

def set\_name(self, given\_name):

self.full\_name = given\_name

def set\_age(self, num\_years):

self.age\_years = num\_years

# Other parts omitted

def print\_all(self):

print('Name:', self.full\_name)

print('Age:', self.age\_years)

class PetData(AnimalData):

def \_\_init\_\_(self):

AnimalData.\_\_init\_\_(self)

self.id\_num = 0

def set\_id(self, pet\_id):

self.id\_num = pet\_id

def print\_all(self): # print\_all of child class over ride the print\_all metho

AnimalData.print\_all(self)

print('ID:', self.id\_num)

user\_pet = PetData()

user\_pet.set\_name(input())

user\_pet.set\_age(int(input()))

user\_pet.set\_id(int(input()))

user\_pet.print\_all()

## Reference and Additional Materials:

<https://youtu.be/u1be7Vele5o?si=kGVtSd8Jxuv35r-d>

<https://trainings.internshala.com/blog/python-inheritance/>

<https://youtu.be/Q8YlYHjksLo?si=IESr5j8d8O1Z3PY4>

<https://www.programiz.com/python-programming/multiple-inheritance>

<https://youtu.be/J2_DHndMW9s?si=RwVxqjXqXQHaeFEJ>

<https://openstax.org/books/introduction-python-programming/pages/13-4-hierarchical-inheritance>

<https://youtu.be/3HlxsEGlHFA?si=z3K_JfXko7W-i9hM>

<https://www.codechef.com/learn/course/oops-concepts-in-python/CPOPPY03/problems/ADVPPY19>

<https://youtu.be/4Y83cUbDKZ8?si=hK__YIO4GJ833h0W>