

[Free Python 3 Course](#) [Data Types](#) [Control Flow](#) [Functions](#) [List](#) [String](#) [Set](#) [Tuple](#) [Dictionary](#)

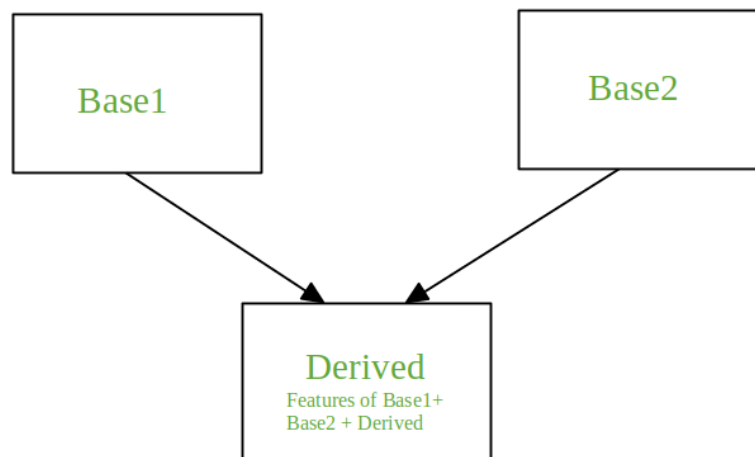
# Multiple Inheritance in Python

[Read](#)[Discuss](#)[Courses](#)[Practice](#)

Inheritance is the mechanism to achieve the re-usability of code as one class(child class) can derive the properties of another class(parent class). It also provides transitivity ie. if class C inherits from P then all the sub-classes of C would also inherit from P.

## Multiple Inheritance

When a class is derived from more than one base class it is called multiple Inheritance. The derived class inherits all the features of the base case.



### Syntax:

Class Base1:

    Body of the class

Class Base2:

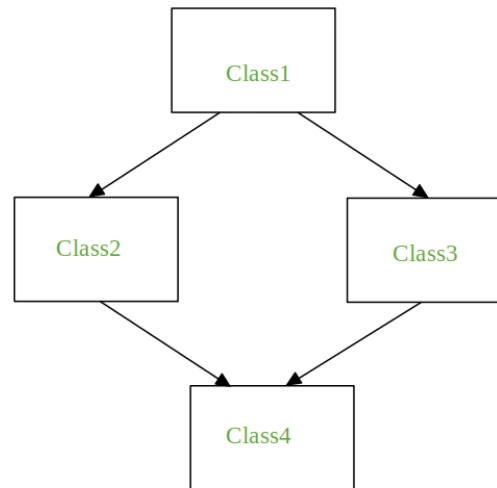
    Body of the class



```
Class Derived(Base1, Base2):  
    Body of the class
```

In the coming section, we will see the problem faced during multiple inheritance and how to tackle it with the help of examples.

## The Diamond Problem



It refers to an ambiguity that arises when two classes Class2 and Class3 inherit from a superclass Class1 and class Class4 inherits from both Class2 and Class3. If there is a method “m” which is an overridden method in one of Class2 and Class3 or both then the ambiguity arises which of the method “m” Class4 should inherit.

### When the method is overridden in both classes

---

## Python3

```
# Python Program to depict multiple inheritance  
# when method is overridden in both classes
```

```
class Class1:  
    def m(self):  
        print("In Class1")  
  
class Class2(Class1):  
    def m(self):
```



```
        print("In Class2")

class Class3(Class1):
    def m(self):
        print("In Class3")

class Class4(Class2, Class3):
    pass

obj = Class4()
obj.m()
```

### Output:

In Class2

**Note:** When you call obj.m() (m on the instance of Class4) the output is In Class2. If Class4 is declared as Class4(Class3, Class2) then the output of obj.m() will be In Class3.

### When the method is overridden in one of the classes

---

## Python3

```
# Python Program to depict multiple inheritance
# when method is overridden in one of the classes
```

```
class Class1:
    def m(self):
        print("In Class1")

class Class2(Class1):
    pass

class Class3(Class1):
    def m(self):
        print("In Class3")

class Class4(Class2, Class3):
    pass

obj = Class4()
obj.m()
```



## Output:

In Class3

## When every class defines the same method

---

### Python3

```
# Python Program to depict multiple inheritance  
# when every class defines the same method
```

```
class Class1:  
    def m(self):  
        print("In Class1")  
  
class Class2(Class1):  
    def m(self):  
        print("In Class2")  
  
class Class3(Class1):  
    def m(self):  
        print("In Class3")  
  
class Class4(Class2, Class3):  
    def m(self):  
        print("In Class4")  
  
obj = Class4()  
obj.m()  
  
Class2.m(obj)  
Class3.m(obj)  
Class1.m(obj)
```

## Output:

In Class4  
In Class2  
In Class3  
In Class1



The output of the method `obj.m()` in the above code is **In Class4**. The

method “m” of Class4 is executed. To execute the method “m” of the other classes it can be done using the class names.

Now, to call the method m for Class1, Class2, Class3 directly from the method “m” of the Class4 see the below example

---

## Python3

```
# Python Program to depict multiple inheritance
# when we try to call the method m for Class1,
# Class2, Class3 from the method m of Class4
```

```
class Class1:
    def m(self):
        print("In Class1")

class Class2(Class1):
    def m(self):
        print("In Class2")

class Class3(Class1):
    def m(self):
        print("In Class3")

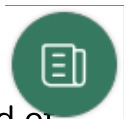
class Class4(Class2, Class3):
    def m(self):
        print("In Class4")
        Class2.m(self)
        Class3.m(self)
        Class1.m(self)

obj = Class4()
obj.m()
```

### Output:

```
In Class4
In Class2
In Class3
In Class1
```

To call “m” of Class1 from both “m” of Class2 and “m” of Class3 instead of Class4 is shown below:



## Python3

```
# Python Program to depict multiple inheritance
# when we try to call m of Class1 from both m of
# Class2 and m of Class3
```

```
class Class1:
    def m(self):
        print("In Class1")

class Class2(Class1):
    def m(self):
        print("In Class2")
        Class1.m(self)

class Class3(Class1):
    def m(self):
        print("In Class3")
        Class1.m(self)

class Class4(Class2, Class3):
    def m(self):
        print("In Class4")
        Class2.m(self)
        Class3.m(self)

obj = Class4()
obj.m()
```

### Output:

```
In Class4
In Class2
In Class1
In Class3
In Class1
```

The output of the above code has one problem associated with it, the method `m` of `Class1` is called twice. Python provides a solution to the above problem with the help of the `super()` function. Let's see how it works.



### The super Function

## Python3

```
# Python program to demonstrate  
# super()
```

```
class Class1:  
    def m(self):  
        print("In Class1")  
  
class Class2(Class1):  
    def m(self):  
        print("In Class2")  
        super().m()  
  
class Class3(Class1):  
    def m(self):  
        print("In Class3")  
        super().m()  
  
class Class4(Class2, Class3):  
    def m(self):  
        print("In Class4")  
        super().m()  
  
obj = Class4()  
obj.m()
```

### Output:

```
In Class4  
In Class2  
In Class3  
In Class1
```

Super() is generally used with the `__init__` function when the instances are initialized. The super function comes to a conclusion, on which method to call with the help of the **method resolution order (MRO)**.

### Method resolution order:



In Python, every class whether built-in or user-defined is derived from the object class and all the objects are instances of the class object. Hence,

the object class is the base class for all the other classes.

In the case of multiple inheritance, a given attribute is first searched in the current class if it's not found then it's searched in the parent classes. The parent classes are searched in a left-right fashion and each class is searched once.

If we see the above example then the order of search for the attributes will be Derived, Base1, Base2, object. The order that is followed is known as a linearization of the class Derived and this order is found out using a set of rules **called Method Resolution Order (MRO)**.

To view the MRO of a class:

- Use the `mro()` method, it returns a list  
Eg. `Class4.mro()`
- Use the `__mro__` attribute, it returns a tuple  
Eg. `Class4.__mro__`

### Example:

---

## Python3

```
# Python program to demonstrate  
# super()
```

```
class Class1:  
    def m(self):  
        print("In Class1")  
  
class Class2(Class1):  
    def m(self):  
        print("In Class2")  
        super().m()  
  
class Class3(Class1):  
    def m(self):  
        print("In Class3")  
        super().m()  
  
class Class4(Class2, Class3):  
    def m(self):  
        print("In Class4")  
        super().m()
```





```
print(Class4.mro())          #This will print list
print(Class4.__mro__)       #This will print tuple
```

### Output:




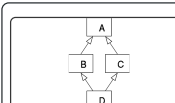



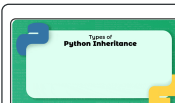
```
[<class '__main__.Class4'>, <class '__main__.Class2'>, <class
 '__main__.Class3'>, <class '__main__.Class1'>, <class 'object'>]
(<class '__main__.Class4'>, <class '__main__.Class2'>, <class
 '__main__.Class3'>, <class '__main__.Class1'>, <class 'object'>)
```

Whether you're preparing for your first job interview or aiming to upskill in this ever-evolving tech landscape, [GeeksforGeeks Courses](#) are your key to success. We provide top-quality content at affordable prices, all geared towards accelerating your growth in a time-bound manner. Join the millions we've already empowered, and we're here to do the same for you. Don't miss out - [check it out now!](#)

Last Updated : 22 Feb, 2022

49

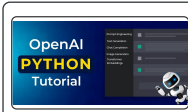
### Similar Reads

 OOP in Python   Set 3 (Inheritance, examples of object, subclass and...	 Inheritance in Python   Set 2
 Inheritance in Python	 Method resolution order in Python Inheritance
 Python   super() in single inheritance	 Data Classes in Python   Set 4 (Inheritance)
 Python   super() function with multilevel inheritance	 Types of inheritance Python
Inheritance in Python Inner Class	Conditional Inheritance in Python

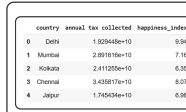




## Related Tutorials



OpenAI Python API -  
Complete Guide



Pandas AI: The Generative  
AI Python Library



Python for Kids - Fun  
Tutorial to Learn Python  
Programming



Data Analysis Tutorial



Flask Tutorial

Previous

self in Python class

Next

Python - Remove nested records from  
tuple

### Article Contributed By :

[soumya08](#)

S

soumya08

Follow

### Vote for difficulty

Current difficulty : [Easy](#)

Easy

Normal

Medium

Hard

Expert

Improved By : [anushkalaharia](#), [trdeshmukh2012](#)

Article Tags : [python-inheritance](#) , [Python-OOP](#) , [Python](#)

Practice Tags : [python](#)

Improve Article

Report Issue



Tower, Sector-136, Noida, Uttar Pradesh -  
201305

feedback@geeksforgeeks.org



## Company

About Us  
Legal  
Terms & Conditions  
Careers  
In Media  
Contact Us  
Advertise with us  
GFG Corporate Solution  
Placement Training Program  
Apply for Mentor

## Languages

Python  
Java  
C++  
PHP  
GoLang  
SQL  
R Language  
Android Tutorial

## DSA Roadmaps

DSA for Beginners  
Basic DSA Coding Problems  
DSA Roadmap by Sandeep Jain  
DSA with JavaScript

## Explore

Job-A-Thon Hiring Challenge  
Hack-A-Thon  
GfG Weekly Contest  
Offline Classes (Delhi/NCR)  
DSA in JAVA/C++  
Master System Design  
Master CP  
GeeksforGeeks Videos

## DSA Concepts

Data Structures  
Arrays  
Strings  
Linked List  
Algorithms  
Searching  
Sorting  
Mathematical  
Dynamic Programming

## Web Development

HTML  
CSS  
JavaScript  
Bootstrap



Top 100 DSA Interview Problems

All Cheat Sheets

## Computer Science

GATE CS Notes

Operating Systems

Computer Network

Database Management System

Software Engineering

Digital Logic Design

Engineering Maths

## Data Science & ML

Data Science With Python

Data Science For Beginner

Machine Learning Tutorial

Maths For Machine Learning

Pandas Tutorial

NumPy Tutorial

NLP Tutorial

Deep Learning Tutorial

## Competitive Programming

Top DSA for CP

Top 50 Tree Problems

Top 50 Graph Problems

Top 50 Array Problems

Top 50 String Problems

Top 50 DP Problems

Top 15 Websites for CP

## Interview Corner

Company Wise Preparation

ReactJS

AngularJS

NodeJS

Express.js

Lodash

## Python

Python Programming Examples

Django Tutorial

Python Projects

Python Tkinter

OpenCV Python Tutorial

Python Interview Question

## DevOps

Git

AWS

Docker

Kubernetes

Azure

GCP

## System Design

What is System Design

Monolithic and Distributed SD

Scalability in SD

Databases in SD

High Level Design or HLD

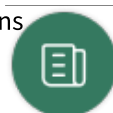
Low Level Design or LLD

Crack System Design Round

System Design Interview Questions

## GfG School

CBSE Notes for Class 8



Preparation for SDE

Experienced Interviews

Internship Interviews

Competitive Programming

Aptitude Preparation

### Commerce

Accountancy

Business Studies

Economics

Human Resource Management (HRM)

Management

Income Tax

Finance

Statistics for Economics

### SSC/ BANKING

SSC CGL Syllabus

SBI PO Syllabus

SBI Clerk Syllabus

IBPS PO Syllabus

IBPS Clerk Syllabus

Aptitude Questions

SSC CGL Practice Papers

CBSE Notes for Class 9

CBSE Notes for Class 10

CBSE Notes for Class 11

CBSE Notes for Class 12

English Grammar

### UPSC

Polity Notes

Geography Notes

History Notes

Science and Technology Notes

Economics Notes

Important Topics in Ethics

UPSC Previous Year Papers

### Write & Earn

Write an Article

Improve an Article

Pick Topics to Write

Share your Experiences

Internships

@GeeksforGeeks, Sanchhaya Education Private Limited, All rights reserved

