**Inheritance in Python**

**Explanation of Python Inheritance Syntax**

1. **Parent Class**:
   * This is the base class from which other classes inherit.
   * It contains attributes and methods that the child class can reuse.
2. **Child Class:**
   * This is the derived class that inherits from the parent class.
   * The syntax for inheritance is class ChildClass(ParentClass).
   * The child class automatically gets all attributes and methods of the parent class unless overridden.

**Creating a Child Class**

A child class (also known as a subclass) is a class that inherits properties and methods from its parent class. The child class can also introduce additional attributes and methods, or even override the ones inherited from the parent.

**super() Function**

super() function is used to call the parent class’s methods. In particular, it is commonly used in the child class’s \_\_init\_\_() method to initialize inherited attributes.

**class** **Person**:

**def** \_\_init\_\_(self, name, idnumber):

self.name = name

self.idnumber = idnumber

*# Child Class: Employee*

**class** **Employee**(Person):

**def** \_\_init\_\_(self, name, idnumber, salary, post):

super().\_\_init\_\_(name, idnumber) *# Calls Person's \_\_init\_\_()*

self.salary = salary

self.post = post

**Types of Python Inheritance**

1. **Single Inheritance**: A child class inherits from one parent class.
2. **Multiple Inheritance**: A child class inherits from more than one parent class.

**class** **EmployeePersonJob**(Employee, Job):

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

Employee.\_\_init\_\_(self, name, salary)

Job.\_\_init\_\_(self, salary) *# Initialize Job*

1. **Multilevel Inheritance**: A class is derived from a class which is also derived from another class.

**class** **Manager**(EmployeePersonJob):

**def** \_\_init\_\_(self, name, salary, department):

EmployeePersonJob.\_\_init\_\_(self, name, salary)

self.department = department

1. **Hierarchical Inheritance**: Multiple classes inherit from a single parent class.

Here *Manager* and *AssistantManager* refer the hierarchical nature

**class** **Manager**(EmployeePersonJob):

**def** \_\_init\_\_(self, name, salary, department):

EmployeePersonJob.\_\_init\_\_(self, name, salary)

self.department = department

**class** **AssistantManager**(EmployeePersonJob):

**def** \_\_init\_\_(self, name, salary, team\_size):

EmployeePersonJob.\_\_init\_\_(self, name, salary)

self.team\_size = team\_size

1. **Hybrid Inheritance**: A combination of more than one type of inheritance.

SeniorManager inherits from both Manager (multilevel) and AssistantManager (hierarchical), combining two inheritance types.

**class** **SeniorManager**(Manager, AssistantManager):

**def** \_\_init\_\_(self, name, salary, department, team\_size):

Manager.\_\_init\_\_(self, name, salary, department)

AssistantManager.\_\_init\_\_(self, name, salary, team\_size)

**Method Overriding in Python**

Subclass or child class to provide a specific implementation of a method that is already provided by one of its super-classes or parent classes.

**class** **Parent**():

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

self.value = "Inside Parent"

**def** show(self):

print(self.value)

**class** **Child**(Parent):

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

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

self.value = "Inside Child"

**def** show(self):

print(self.value)

obj1 = Parent()

obj2 = Child()

obj1.show() *# Should print "Inside Parent"*

obj2.show() *# Should print "Inside Child"*

**Operator Overloading in Python**

Same built-in operator or function shows different behavior for objects of different classes, this is called ***Operator Overloading***.

Eg: + is used to add two integers as well as join two strings and merge two lists.

**Overloading binary + operator in Python:**

When we use an operator on user-defined data types then automatically a **special function or magic function** associated with that operator is invoked.

Eg: + operator, the magic method \_\_add\_\_

**class** **A**:

**def** \_\_init\_\_(self, a):

self.a = a

**def** \_\_add\_\_(self, o):

**return** self.a + o.a

ob1 = A(1)

ob2 = A(2)

ob3 = A("Geeks")

ob4 = A("For")

print(ob1 + ob2)

print(ob3 + ob4)

print(A.\_\_add\_\_(ob1 , ob2))

print(A.\_\_add\_\_(ob3,ob4))

print(ob1.\_\_add\_\_(ob2))

print(ob3.\_\_add\_\_(ob4))

3

GeeksFor

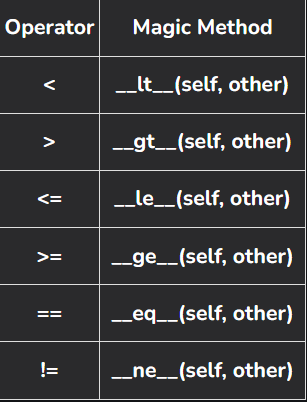
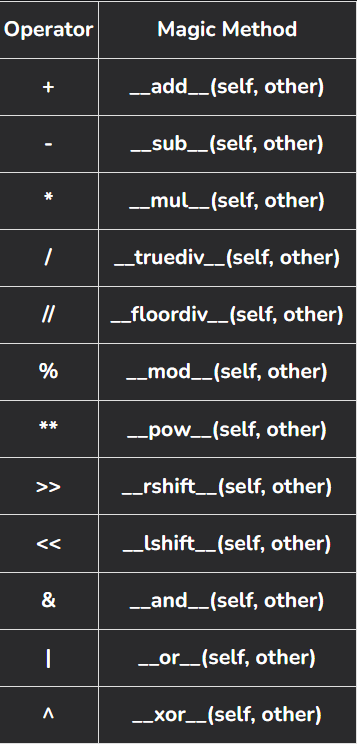
3

GeeksFor

3

GeeksFor

coded as "**ob1 + ob2**", the special function is automatically called as **ob1.\_\_add\_\_(ob2)**which simply means that ob1 calls the \_\_add\_\_( ) function with ob2 as an Argument and It actually means **A .\_\_add\_\_(ob1, ob2)**.



**Python super()**

super() function is used to refer to the parent class or superclass

**Super with Single and Multiple Inheritances**

**class** **GFG1**:

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

print('HEY !!!!!! GfG I am initialised(Class GEG1)')

**def** sub\_GFG(self, b):

print('Printing from class GFG1:', b)

**class** **GFG2**(GFG1):

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

print('HEY !!!!!! GfG I am initialised(Class GEG2)')

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

**def** sub\_GFG(self, b):

print('Printing from class GFG2:', b)

super().sub\_GFG(b + 1)

**class** **GFG3**(GFG2):

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

print('HEY !!!!!! GfG I am initialised(Class GEG3)')

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

**def** sub\_GFG(self, b):

print('Printing from class GFG3:', b)

super().sub\_GFG(b + 1)

**if** \_\_name\_\_ == '\_\_main\_\_':

gfg = GFG3()

gfg.sub\_GFG(10)

HEY !!!!!! GfG I am initialised(Class GEG3)  
HEY !!!!!! GfG I am initialised(Class GEG2)  
HEY !!!!!! GfG I am initialised(Class GEG1)  
Printing from class GFG3: 10  
Printing from class GFG2: 11  
Printing from class GFG1: 12

**Super with Multi-Level Inheritance**

**class** **Mammal**():

**def** \_\_init\_\_(self, name):

print(name, "Is a mammal")

**class** **canFly**(Mammal):

**def** \_\_init\_\_(self, canFly\_name):

print(canFly\_name, "cannot fly")

super().\_\_init\_\_(canFly\_name)

**class** **canSwim**(canFly):

**def** \_\_init\_\_(self, canSwim\_name):

print(canSwim\_name, "cannot swim")

super().\_\_init\_\_(canSwim\_name)

**class** **Animal**(canSwim):

**def** \_\_init\_\_(self, name):

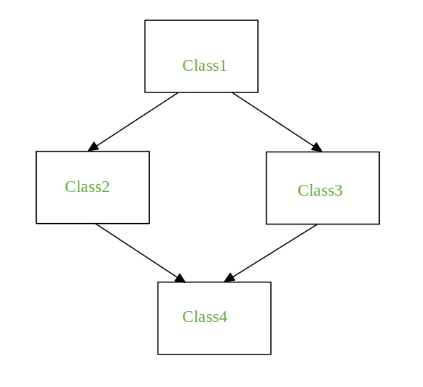
super().\_\_init\_\_(name)

Carol = Animal("Dog")

Dog cannot swim  
Dog cannot fly  
Dog Is a mammal

**Multiple Inheritance in Python**

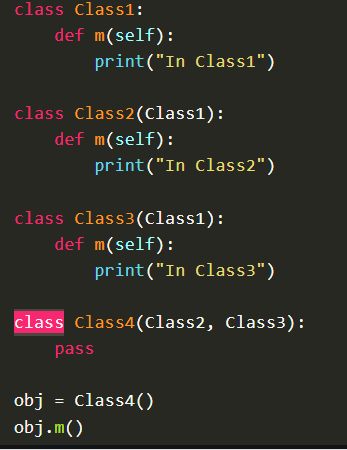
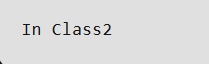
When a class is derived from more than one base class it is called multiple Inheritance.

**The Diamond Problem**

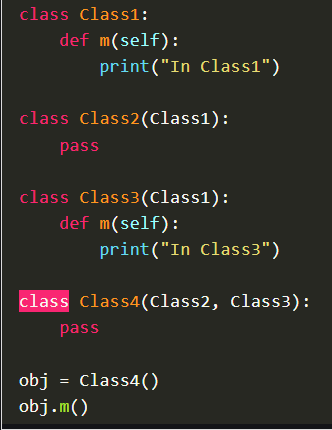
It refers to an ambiguity that arises when two classes Class2 and Class3 inherit from a superclass Class1 and class Class4 inherits from both Class2 and Class3.

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**

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

**Method resolution order:**

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).**

* Use the mro() method, it returns a list   
  Eg. Class4.mro()
* Use the \_mro\_ attribute, it returns a tuple   
  Eg. Class4.\_\_mro\_\_

**What Is Hybrid Inheritance In Python?**

 Hybrid inheritance is a combination of more than one type of inheritance.

**Multilevel Inheritance in Python**

Multilevel inheritance in Python is a type of Inheritance in which a class inherits from a class, which itself inherits from another class.