………………………………………………………………………………………………

>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Chapter-19\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Types Of Inheritance\*\*\*\*\*\*\*\*\*\*\*\*\*:-

………………………………………………………………………………………………

Types of Inheritance:

…………………………………..

1. Single Inheritance:

The concept of inheriting the properties from one class to another class is known as single inheritance.

………………………………………

Eg:

1) class P:

2) def m1(self):

3) print("Parent Method")

4) class C(P):

5) def m2(self):

6) print("Child Method")

7) c=C()

8) c.m1()

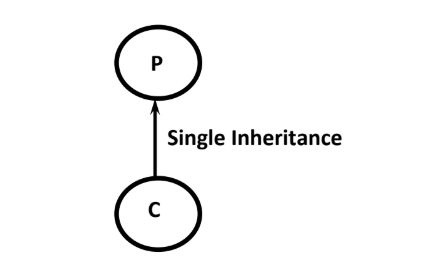
9) c.m2()

………………………………………….

Output:

Parent Method

Child Method



……………………………………………………………………………………………..2. Multi Level Inheritance:

…………………………………………

The concept of inheriting the properties from multiple classes to single class with the concept of one after another is known as multilevel inheritance.

………………………………….

Eg:

1) class P:

2) def m1(self):

3) print("Parent Method")

4) class C(P):

5) def m2(self):

6) print("Child Method")

7) class CC(C):

8) def m3(self):

9) print("Sub Child Method")

10) c=CC()

11) c.m1()

12) c.m2()

13) c.m3()

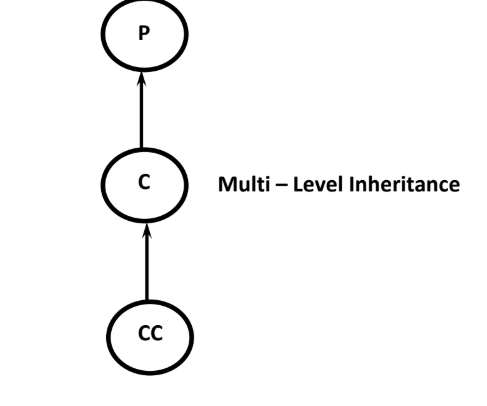
…………………………………………….

Output:

Parent Method

Child Method

Sub Child Method

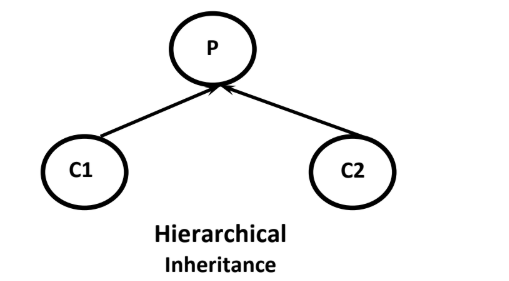


……………………………………………………………………………………..

3. Hierarchical Inheritance:

………………………………………………………………

The concept of inheriting properties from one class into multiple classes which are present at same level is known as Hierarchical Inheritance



………………………………………….

1) class P:

2) def m1(self):

3) print("Parent Method")

4) class C1(P):

5) def m2(self):

6) print("Child1 Method")

7) class C2(P):

8) def m3(self):

9) print("Child2 Method")

10) c1=C1()

11) c1.m1()

12) c1.m2()

13) c2=C2()

14) c2.m1()

15) c2.m3()

…………………………………………………………

Output:

Parent Method

Child1 Method

Parent Method

Child2 Method

………………………………………………………………….

4. Multiple Inheritance:

………………………….

The concept of inheriting the properties from multiple classes into a single class at a time, is known as multiple inheritance.

……………………………………

1) class P1:

2) def m1(self):

3) print("Parent1 Method")

4) class P2:

5) def m2(self):

6) print("Parent2 Method")

7) class C(P1,P2):

8) def m3(self):

9) print("Child2 Method")

10) c=C()

11) c.m1()

12) c.m2()

13) c.m3()

……………………………………….

Output:

Parent1 Method

Parent2 Method

Child2 Method

If the same method is inherited from both parent classes,then Python will always consider the order of Parent classes in the declaration of the child class.

class C(P1,P2): ===>P1 method will be considered

class C(P2,P1): ===>P2 method will be considered

………………………………………………………………….

Eg:

1) class P1:

2) def m1(self):

3) print("Parent1 Method")

4) class P2:

5) def m1(self):

6) print("Parent2 Method")

7) class C(P1,P2):

8) def m2(self):

9) print("Child Method")

10) c=C()

11) c.m1()

12) c.m2()

………………………………..

Output:

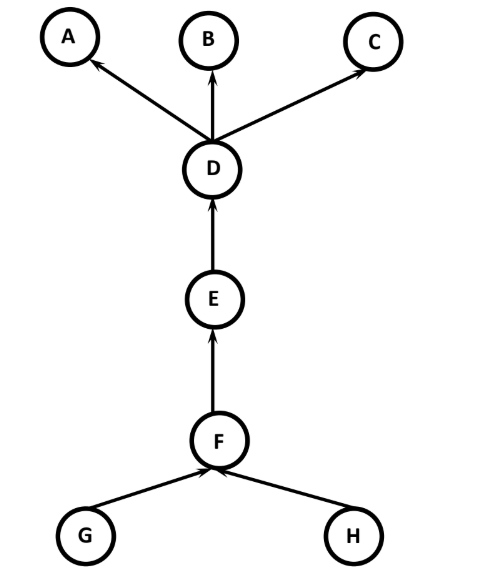
Parent1 Method

Child Method

…………………………………………………………………

5. Hybrid Inheritance:

Combination of Single, Multi level, multiple and Hierarchical inheritance is known as Hybrid Inheritance.



……………………………………………………………………………………….

6. Cyclic Inheritance:

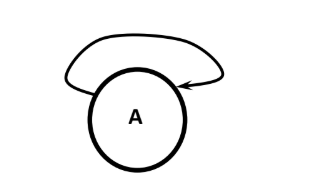
The concept of inheriting properties from one class to another class in cyclic way, is called Cyclic inheritance.Python won't support for Cyclic Inheritance of course it is really not required.

………………………………………..

Eg - 1:

class A(A):pass

NameError: name 'A' is not defined



………………………………………………………………………………….

Eg - 2:

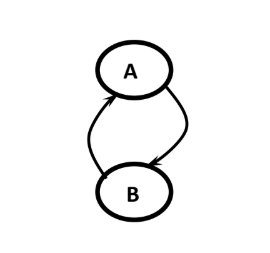
1) class A(B):

2) pass

3) class B(A):

4) pass

NameError: name 'B' is not defined



…………………………………………………………………………………….

\*\*\*\*\*\*\*\*\*\*\*Method Resolution Order (MRO):\*\*\*\*\*\*\*:-

……………………………………………………………………………………..

In Hybrid Inheritance the method resolution order is decided based on MRO algorithm.

This algorithm is also known as C3 algorithm.

Samuele Pedroni proposed this algorithm.

It follows DLR (Depth First Left to Right)

i.e Child will get more priority than Parent.

Left Parent will get more priority than Right Parent

MRO(X)=X+Merge(MRO(P1),MRO(P2),...,ParentList)

…………………………………………………………………………………….

\*\*\*\*\*\*\*\*\*Head Element vs Tail Terminology:\*\*\*\*\*\*\*

……………………………………………………………………………….

Assume C1,C2,C3,...are classes.

In the list : C1C2C3C4C5....

C1 is considered as Head Element and remaining is considered as Tail.

……………………………………………………………………………….

\*\*\*\*\*\*\*\*\*\*\*\*\*How to find Merge:\*\*\*\*\*\*\*\*\*:-

……………………………………………………………………………

Take the head of first list

If the head is not in the tail part of any other list,then add this head to the result and remove it

from the lists in the merge.

If the head is present in the tail part of any other list,then consider the head element of the next list and continue the same process.

………………………………………………….

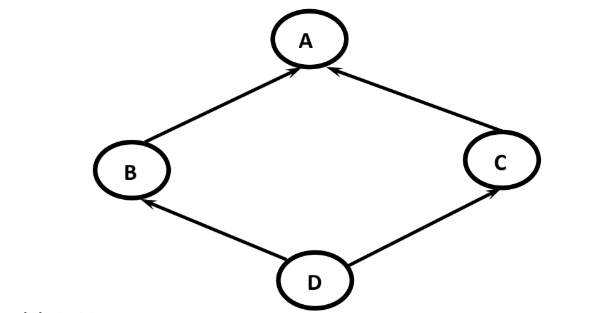
Note: We can find MRO of any class by using mro() function.

print(ClassName.mro())

…………………………………………….

\*\*\*\*Demo Program-1 for Method Resolution Order:\*\*\*:-

…………………………………………………………………………………



mro(A)=A,object

mro(B)=B,A,object

mro(C)=C,A,object

mro(D)=D,B,C,A,object

………………………………………………………………..

test.py:

1) class A:pass

2) class B(A):pass

3) class C(A):pass

4) class D(B,C):pass

5) print(A.mro())

6) print(B.mro())

7) print(C.mro())

8) print(D.mro())

Output:

[<class '\_\_main\_\_.A'>, <class 'object'>]

[<class '\_\_main\_\_.B'>, <class '\_\_main\_\_.A'>, <class 'object'>]

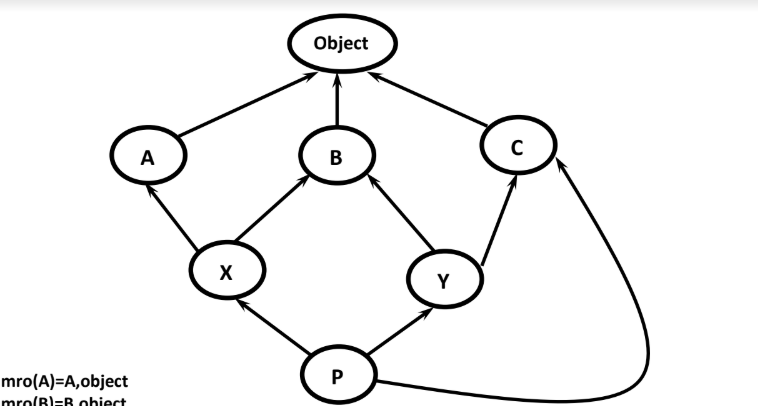
[<class '\_\_main\_\_.C'>, <class '\_\_main\_\_.A'>, <class 'object'>]

[<class '\_\_main\_\_.D'>, <class '\_\_main\_\_.B'>, <class '\_\_main\_\_.C'>, <class '\_\_main\_\_.A'>, <class 'object'>]

………………………………………………………………………………………

\*\*\*\*\*\*Demo Program-2 for Method Resolution Order:\*\*\*

…………………………………………………………………………………..



mro(A)=A,object

mro(B)=B,object

mro(C)=C,object

mro(X)=X,A,B,object

mro(Y)=Y,B,C,object

mro(P)=P,X,A,Y,B,C,object

……………………………………………………………………………………………..\*\*\*Finding mro(P) by using C3 algorithm:\*\*\*\*\*\*\*

………………………………………………………………………………………..

Formula: MRO(X)=X+Merge(MRO(P1),MRO(P2),...,ParentList)

mro(p)= P+Merge(mro(X),mro(Y),mro(C),XYC)

= P+Merge(XABO,YBCO,CO,XYC)

= P+X+Merge(ABO,YBCO,CO,YC)

= P+X+A+Merge(BO,YBCO,CO,YC)

= P+X+A+Y+Merge(BO,BCO,CO,C)

= P+X+A+Y+B+Merge(O,CO,CO,C)

= P+X+A+Y+B+C+Merge(O,O,O)

= P+X+A+Y+B+C+O

………………………………………………

test.py:

1) class A:pass

2) class B:pass

3) class C:pass

4) class X(A,B):pass

5) class Y(B,C):pass

6) class P(X,Y,C):pass

7) print(A.mro())#AO

8) print(X.mro())#XABO

9) print(Y.mro())#YBCO

10) print(P.mro())#PXAYBCO

…………………………………

Output:

[<class '\_\_main\_\_.A'>, <class 'object'>]

[<class '\_\_main\_\_.X'>, <class '\_\_main\_\_.A'>, <class '\_\_main\_\_.B'>, <class 'object'>]

[<class '\_\_main\_\_.Y'>, <class '\_\_main\_\_.B'>, <class '\_\_main\_\_.C'>, <class 'object'>]

[<class '\_\_main\_\_.P'>, <class '\_\_main\_\_.X'>, <class '\_\_main\_\_.A'>, <class '\_\_main\_\_.Y'>, <class '\_\_main\_\_.B'>,

<class '\_\_main\_\_.C'>, <class 'object'>]

…………………………………….

test.py:

1) class A:

2) def m1(self):

3) print('A class Method')

4) class B:

5) def m1(self):

6) print('B class Method')

7) class C:

8) def m1(self):

9) print('C class Method')

10) class X(A,B):

11) def m1(self):

12) print('X class Method')

13) class Y(B,C):

14) def m1(self):

15) print('Y class Method')

16) class P(X,Y,C):

17) def m1(self):

18) print('P class Method')

19) p=P()

20) p.m1()

……………………………..

Output:

P class Method

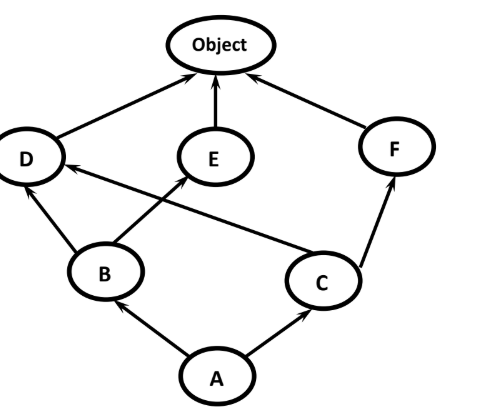
In the above example P class m1() method will be considered.If P class does not contain m1() method then as per MRO, X class method will be considered. If X class does not contain then A class method will be considered and this process will be continued.

The method resolution in the following order:PXAYBCO

………………………………………………………………………………….

\*\*\*\*\*Demo Program-3 for Method Resolution Order:\*\*\*

………………………………………………………………………………………

mro(o)=object

mro(D)=D,object

mro(E)=E,object

mro(F)=F,object

mro(B)=B,D,E,object

mro(C)=C,D,F,object

mro(A)=A+Merge(mro(B),mro(C),BC)

=A+Merge(BDEO,CDFO,BC)

=A+B+Merge(DEO,CDFO,C)

=A+B+C+Merge(DEO,DFO)

=A+B+C+D+Merge(EO,FO)

=A+B+C+D+E+Merge(O,FO)

=A+B+C+D+E+F+Merge(O,O)

=A+B+C+D+E+F+O

…………………………………………………..

1) class D:pass

2) class E:pass

3) class F:pass

4) class B(D,E):pass

5) class C(D,F):pass

6) class A(B,C):pass

7) print(D.mro())

8) print(B.mro())

9) print(C.mro())

10) print(A.mro())

…………………………………………

Output:

[<class '\_\_main\_\_.D'>, <class 'object'>]

[<class '\_\_main\_\_.B'>, <class '\_\_main\_\_.D'>, <class '\_\_main\_\_.E'>, <class 'object'>]

[<class '\_\_main\_\_.C'>, <class '\_\_main\_\_.D'>, <class '\_\_main\_\_.F'>, <class 'object'>]

[<class '\_\_main\_\_.A'>, <class '\_\_main\_\_.B'>, <class '\_\_main\_\_.C'>, <class '\_\_main\_\_.D'>, <class '\_\_main\_\_.E'>,

<class '\_\_main\_\_.F'>, <class 'object'>]

………………………………………………………………………………………….

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*super() Method:\*\*\*\*\*\*\*\*\*\*\*\*\*

…………………………………………………………………………………….

super() is a built-in method which is useful to call the super class constructors,variables and methods from the child class.

……………………………………………………………………………………………..\*\*\*\*Demo Program-1 for super():\*\*\*\*\*\*\*\*\*:-

…………………………………………………………………………………………..

1) class Person:

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

3) self.name=name

4) self.age=age

5) def display(self):

6) print('Name:',self.name)

7) print('Age:',self.age)

8)

9) class Student(Person):

10) def \_\_init\_\_(self,name,age,rollno,marks):

11) super().\_\_init\_\_(name,age)

12) self.rollno=rollno

13) self.marks=marks

14)

15) def display(self):

16) super().display()

17) print('Roll No:',self.rollno)

18) print('Marks:',self.marks)

19)

20) s1=Student(' Prasanna',22,101,90)

21) s1.display()

………………………….

Output:

Name: Prasanna

Age: 22

Roll No: 101

Marks: 90

…………………………………………

In the above program we are using super() method to call parent class constructor and display() method

………………………………………………………………………………………………\*\*\*\*\*Demo Program-2 for super():\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*:-

…………………………………………………………………………………….

1) class P:

2) a=10

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

4) self.b=10

5) def m1(self):

6) print('Parent instance method')

7) @classmethod

8) def m2(cls):

9) print('Parent class method')

10) @staticmethod

11) def m3():

12) print('Parent static method')

13)

14) class C(P):

15) a=888

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

17) self.b=999

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

19) print(super().a)

20) super().m1()

21) super().m2()

22) super().m3()

23)

24) c=C()

………………………………………………

Output:

10

Parent instance method

Parent class method

Parent static method

In the above example we are using super() to call various members of Parent class.

………………………………………………………………………………………..

\*\*\*\*How to call method of a particular Super class:\*\*\*:-

…………………………………………………………………………………………..

We can use the following approaches

1. super(D,self).m1()

It will call m1() method of super class of D.

……………………………………..

2. A.m1(self)

It will call A class m1() method

1) class A:

2) def m1(self):

3) print('A class Method')

4) class B(A):

5) def m1(self):

6) print('B class Method')

7) class C(B):

8) def m1(self):

9) print('C class Method')

10) class D(C):

11) def m1(self):

12) print('D class Method')

13) class E(D):

14) def m1(self):

15) A.m1(self)

16)

17) e=E()

18) e.m1()

……………………………………….

Output:

A class Method

…………………………………………………………

\*\*\*\*Various Important Points about super():\*\*\*\*:-

………………………………………………………………………………..

Case-1: From child class we are not allowed to access parent class instance variables by using super(),Compulsory we should use self only.

But we can access parent class static variables by using super().

…………………………………

Eg:

1) class P:

2) a=10

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

4) self.b=20

5)

6) class C(P):

7) def m1(self):

8) print(super().a)#valid

9) print(self.b)#valid

10) print(super().b)#invalid

11) c=C()

12) c.m1()

………………………………….

Output:

10

20

AttributeError: 'super' object has no attribute 'b'

…………………………………

Case-2: From child class constructor and instance method, we can access parent class instance method,static method and class method by using super()

1) class P:

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

3) print('Parent Constructor')

4) def m1(self):

5) print('Parent instance method')

6) @classmethod

7) def m2(cls):

8) print('Parent class method')

9) @staticmethod

10) def m3():

11) print('Parent static method')

12)

13) class C(P):

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

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

16) super().m1()

17) super().m2()

18) super().m3()

19)

20) def m1(self):

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

22) super().m1()

23) super().m2()

24) super().m3()

25)

26) c=C()

27) c.m1()

…………………………………………

Output:

Parent Constructor

Parent instance method

Parent class method

Parent static method

Parent Constructor

Parent instance method

Parent class method

Parent static method

……………………………………………………

Case-3: From child class, class method we cannot access parent class instance methods and constructors by using super() directly(but indirectly possible). But we can access parent class static and class methods.

……………………………………………..

1) class P:

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

3) print('Parent Constructor')

4) def m1(self):

5) print('Parent instance method')

6) @classmethod

7) def m2(cls):

8) print('Parent class method')

9) @staticmethod

10) def m3():

11) print('Parent static method')

12)

13) class C(P):

14) @classmethod

15) def m1(cls):

16) #super().\_\_init\_\_()--->invalid

17) #super().m1()--->invalid

18) super().m2()

19) super().m3()

20)

21) C.m1()

………………………………………

Output:

Parent class method

Parent static method

From Class Method of Child class,how to call parent class instance methods and constructors:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

1) class A:

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

3) print('Parent constructor')

4)

5) def m1(self):

6) print('Parent instance method')

7)

8) class B(A):

9) @classmethod

10) def m2(cls):

11) super(B,cls).\_\_init\_\_(cls)

12) super(B,cls).m1(cls)

13)

14) B.m2()

…………………………..

Output:

Parent constructor

Parent instance method

……………………………..

Case-4: In child class static method we are not allowed to use super() generally (But in special way we can use)

1) class P:

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

3) print('Parent Constructor')

4) def m1(self):

5) print('Parent instance method')

6) @classmethod

7) def m2(cls):

8) print('Parent class method')

9) @staticmethod

10) def m3():

11) print('Parent static method')

12)

13) class C(P):

14) @staticmethod

15) def m1():

16) super().m1()-->invalid

17) super().m2()--->invalid

18) super().m3()--->invalid

19)

20) C.m1()

RuntimeError: super(): no arguments

How to call parent class static method from child class static method by using super():

………………………………………………………………….

1) class A:

2)

3) @staticmethod

4) def m1():

5) print('Parent static method')

6)

7) class B(A):

8) @staticmethod

9) def m2():

10) super(B,B).m1()

11)

12) B.m2()

…………………………………………………………………………………

Output: Parent static method

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*END\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

………………………………………………………………………………………………