Python Programming

RGM College of Engineering & Technology (Autonomous)

Department of Computer Science & Engineering

AY:2021-2022

PYTHON'S OBJECT ORIENTED PROGRAMMING - 9

Guido Van Rossum

Agenda:

1. The Complete Story of super() function

1. The Complete Story of super() function

- Parent class members are by default available to the child class. In the child class we can access parent class members directly.

Eg:

I.

```
class P:

    def m1(self):
        print("Parent Method ")

class C(P):
    def m2(self):
        print('Child Method ')

c = C()
c.m2()
```

```
Child Method
```

II.

```
class P:

    def m1(self):
        print("Parent Method ")

class C(P):
    def m2(self):
        self.m1()  # We can use parent class methods in child class directly
        print('Child Method ')

c = C()
c.m2()
```

```
Parent Method
Child Method
```

III.

If parent class and child class contains a member with the same name (i.e.,Naming Conflicts), then to call explicitly parent class members from the child class we should use super() function.

```
class P:

    def m1(self):
        print("Parent Method ")

class C(P):
    def m1(self):
        self.m1()  # We can use parent class methods in child class directly
        print('Child Method ')

c = C()
c.m1()
```

```
---------------------------------------------------------------------------
RecursionError                            Traceback (most recent call last)
<ipython-input-2-45bad9ce83a3> in <module>
     10
     11 c = C()
---> 12 c.m1()

<ipython-input-2-45bad9ce83a3> in m1(self)
      6 class C(P):
      7     def m1(self):
----> 8         self.m1()  # We can use parent class methods in child class d
irectly
      9         print('Child Method ')
     10

... last 1 frames repeated, from the frame below ...

<ipython-input-2-45bad9ce83a3> in m1(self)
      6 class C(P):
      7     def m1(self):
----> 8         self.m1()  # We can use parent class methods in child class d
irectly
      9         print('Child Method ')
     10

RecursionError: maximum recursion depth exceeded
```

IV.

```
class P:

    def m1(self):
        print("Parent Method ")

class C(P):
    def m1(self):
        super().m1()  # We can use parent class methods in child class directl
y
        print('Child Method ')

c = C()
c.m1()
```

```
Parent Method
Child Method
```

super() function:

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

Demo Programs to describe the use of super() function

Demo Program 1:

I.

```
class P:
    a=10
    def __init__(self):
        print('Parent Constructor ')

    def m1(self):
        print('Parent instance method')

    @classmethod
    def m2(cls):
        print('Parent class method')

    @staticmethod
    def m3():
        print('Parent static method')

class C(P):

    def __init__(self):
        print('Child Constructor')


    def method1(self):
        print(super().a)
        super().m1()
        super().m2()
        super().m3()
        super().__init__()

c = C()
c.method1()
```

```
Child Constructor
10
Parent instance method
Parent class method
Parent static method
Parent Constructor
```

Now, My requirement is,

- In the previous example, child class and parent class not having any naming conflict.
- If naming conflict is not there, then super() function is not required. You can use 'self' also to call the members of the parent class.

II.

```
class P:
    a=10
    def __init__(self):
        print('Parent Constructor ')

    def m1(self):
        print('Parent instance method')

    @classmethod
    def m2(cls):
        print('Parent class method')

    @staticmethod
    def m3():
        print('Parent static method')

class C(P):

    def __init__(self):
        print('Child Constructor')


    def method1(self):
        print(super().a)
        self.m1()
        self.m2()
        self.m3()
        super().__init__()  # Naming conflict is there (i.e., 2 constructors a
re there)

c = C()
c.method1()
```

```
Child Constructor
10
Parent instance method
Parent class method
Parent static method
Parent Constructor
```

III.

```
class P:
    a=10
    def __init__(self):
        print('Parent Constructor ')

    def m1(self):
        print('Parent instance method')

    @classmethod
    def m2(cls):
        print('Parent class method')

    @staticmethod
    def m3():
        print('Parent static method')

class C(P):

    def __init__(self):
        print('Child Constructor')


    def method1(self):
        print(super().a)
        self.m1()
        self.m2()
        self.m3()
        self.__init__()  # child class constructor

c = C()
c.method1()
```

```
Child Constructor
10
Parent instance method
Parent class method
Parent static method
Child Constructor
```

Demo Program 2:

```
class P:
    a=10
    def __init__(self):
        self.b = 20
        print('Parent Class Constructor ')

    def m1(self):
        print('Parent instance method')

    @classmethod
    def m2(cls):
        print('Parent class method')

    @staticmethod
    def m3():
        print('Parent static method')

class C(P):

    a = 888

    def __init__(self):
        self.b = 999
        super().__init__()
        print(super().a)
        super().m1()
        super().m2()
        super().m3()

c = C()
```

```
Parent Class Constructor
10
Parent instance method
Parent class method
Parent static method
```

Demo Program 3:

```
class Person:

    def __init__(self,name,age):
        self.name=name
        self.age=age

    def display(self):
        print('Name:',self.name)
        print('Age:',self.age)

class Student(Person):

    def __init__(self,name,age,rollno,marks):
        super().__init__(name,age)
        self.rollno=rollno
        self.marks=marks


    def display(self):
        super().display()
        print('Roll No:',self.rollno)
        print('Marks:',self.marks)

s=Student('Karthikeya',7,111,90)
s.display()
```

```
Name: Karthikeya
Age: 7
Roll No: 111
Marks: 90
```

Demo Program 4:

```
class Person:

    def __init__(self,name,age,height,weight):
        self.name = name
        self.age = age
        self.height = height
        self.weight = weight


    def display(self):
        print('Name:',self.name)
        print('Age:',self.age)
        print('Height:',self.height)
        print('Weight:',self.weight)



class Student(Person):

    def __init__(self,name,age,height,weight,rollno,marks):
        super().__init__(name,age,height,weight)
        self.rollno=rollno
        self.marks=marks


    def display(self):
        super().display()
        print('Roll No:',self.rollno)
        print('Marks:',self.marks)

s=Student('Karthikeya',7,5.9,88,111,90)
s.display()
```

```
Name: Karthikeya
Age: 7
Height: 5.9
Weight: 88
Roll No: 111
Marks: 90
```

How to call method of a particular Super class in Multi Level Inheritance?

By using super() we can call immediate parent class methods. If we want a particular super class method then we have to use the following 2 approaches:

1.classname.methodname(self):

Eg:

A.m1(self) ➔ It will call m1() method of A class.

2. super(classname,self).m1()

Eg:

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

Eg:

I.

```
class A:

    def m1(self):
        print('A class Method')

class B(A):

    def m1(self):
        print('B class Method')

class C(B):

    def m1(self):
        print('C class Method')

class D(C):

    def m1(self):
        print('D class Method')

class E(D):

    def m1(self):
        print('E Class Method')
e=E()
e.m1()
```

```
E Class Method
```

II.

```
class A:

    def m1(self):
        print('A class Method')

class B(A):

    def m1(self):
        print('B class Method')

class C(B):

    def m1(self):
        print('C class Method')

class D(C):

    def m1(self):
        print('D class Method')

class E(D):

    def m1(self):
        super().m1()
e=E()
e.m1()
```

```
D class Method
```

III.

```
class A:

    def m1(self):
        print('A class Method')

class B(A):

    def m1(self):
        print('B class Method')

class C(B):

    def m1(self):
        print('C class Method')

class D(C):

    def m1(self):
        print('D class Method')

class E(D):

    def m1(self):
        B.m1(self)          # Approach 1
e=E()
e.m1()
```

```
B class Method
```

IV.

```
class A:

    def m1(self):
        print('A class Method')

class B(A):

    def m1(self):
        print('B class Method')

class C(B):

    def m1(self):
        print('C class Method')

class D(C):

    def m1(self):
        print('D class Method')

class E(D):

    def m1(self):
        super(B,self).m1()          # Approach 2. super class of B , i.e.,A
thod executes
e=E()
e.m1()
```

```
A class Method
```

Various Important Conclusions about super() function:

Case 1: super() vs Parent class Instance Variables

- 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:

I.

```
class P:
    a = 888
    def __init__(self):
        self.b = 999

class C(P):
    def m1(self):
        print(self.a)   # We can access, but we can't delete or modify
        print(self.b)
c = C()
c.m1()
```

```
888
999
```

In the child class instead of self, if we use super() function.

II.

```
class P:
    a = 888
    def __init__(self):
        self.b = 999

class C(P):
    def m1(self):
        print(super().a)   # We can access parent class static variable by usi
ng super()
        print(super().b)  # We can't use parenet class instance variable by us
ing super()
c = C()
c.m1()
```

```
888
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-5-2f96230b1cde> in <module>
      9         print(super().b)
     10 c = C()
---> 11 c.m1()

<ipython-input-5-2f96230b1cde> in m1(self)
      7     def m1(self):
      8         print(super().a)   # We can access, but we can't delete or mo
dify
----> 9         print(super().b)
     10 c = C()
     11 c.m1()

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

III.

```
class P:
    a = 888
    def __init__(self):
        self.b = 999

class C(P):
    def m1(self):
        print(super().a)   # We can access parent class static variable by usi
ng super()
        print(self.b)  # We can use parenet class instance variable by using s
elf only
c = C()
c.m1()
```

```
888
999
```

IV. If the child class also contains an instance variable with the same name of parent class instance variable (i.e., in this case b). Then how you can differentiate them. See the below code:

```
class P:
    a = 888
    def __init__(self):
        self.b = 999

class C(P):
    def __init__(self):
        self.b = 20
    def m1(self):
        print(self.b)
c = C()
c.m1()
```

```
20
```

Important Conclusion:

- If the parent class and child class having an instance variable with the same name, then from child class you can't access parent class instance variable.

Case 2:

- From child class constructor and instance method, we can access parent class instance method, static method and class method by using super() function.

I.

```
class P:

    def __init__(self):
        print('Parent Constructor')

    def m1(self):
        print('Parent instance method')

    @classmethod
    def m2(cls):
        print('Parent class method')

    @staticmethod
    def m3():
        print('Parent static method')

class C(P):

    def __init__(self):
        super().__init__()
        super().m1()
        super().m2()
        super().m3()

    def m1(self):
        super().__init__()
        super().m1()
        super().m2()
        super().m3()

    @classmethod
    def m2(cls):
        super().__init__()
        super().m1()
        super().m2()
        super().m3()

    @staticmethod
    def m3():
        super().__init__()
        super().m1()
        super().m2()
        super().m3()

c = C()
```

```
Parent Constructor
Parent instance method
Parent class method
Parent static method
```

II. Let me call Instance method,

```
class P:

    def __init__(self):
        print('Parent Constructor')

    def m1(self):
        print('Parent instance method')

    @classmethod
    def m2(cls):
        print('Parent class method')

    @staticmethod
    def m3():
        print('Parent static method')

class C(P):

    def __init__(self):
        super().__init__()
        super().m1()
        super().m2()
        super().m3()

    def m1(self):
        super().__init__()
        super().m1()
        super().m2()
        super().m3()

    @classmethod
    def m2(cls):
        super().__init__()
        super().m1()
        super().m2()
        super().m3()

    @staticmethod
    def m3():
        super().__init__()
        super().m1()
        super().m2()
        super().m3()

c = C()
print()
c.m1()
```

```
Parent Constructor
Parent instance method
Parent class method
Parent static method

Parent Constructor
Parent instance method
Parent class method
Parent static method
```

Important Conclusion 1:

- From child class constructor and instance methods, we can call parent class constructors , instance methods , class methods and static methods by using super() [No Restrictions at all].

III.

```
class P:

    def __init__(self):
        print('Parent Constructor')

    def m1(self):
        print('Parent instance method')

    @classmethod
    def m2(cls):
        print('Parent class method')

    @staticmethod
    def m3():
        print('Parent static method')

class C(P):

    def __init__(self):
        super().__init__()
        super().m1()
        super().m2()
        super().m3()

    def m1(self):
        super().__init__()
        super().m1()
        super().m2()
        super().m3()

    @classmethod
    def m2(cls):
        super().__init__()
        super().m1()
        super().m2()
        super().m3()

    @staticmethod
    def m3():
        super().__init__()
        super().m1()
        super().m2()
        super().m3()

C.m2()  # calling child class class method
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-11-2f3487d70af0> in <module>
     43         super().m3()
     44
---> 45 C.m2()  # calling child class class method

<ipython-input-11-2f3487d70af0> in m2(cls)
     31     @classmethod
     32     def m2(cls):
---> 33         super().__init__()
     34         super().m1()
     35         super().m2()

TypeError: __init__() missing 1 required positional argument: 'self'
```

IV.

```
class P:

    def __init__(self):
        print('Parent Constructor')

    def m1(self):
        print('Parent instance method')

    @classmethod
    def m2(cls):
        print('Parent class method')

    @staticmethod
    def m3():
        print('Parent static method')

class C(P):

    def __init__(self):
        super().__init__()
        super().m1()
        super().m2()
        super().m3()

    def m1(self):
        super().__init__()
        super().m1()
        super().m2()
        super().m3()

    @classmethod
    def m2(cls):
       # super().__init__()
        super().m1()
        super().m2()
        super().m3()

    @staticmethod
    def m3():
        super().__init__()
        super().m1()
        super().m2()
        super().m3()

C.m2()  # calling child class class method
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-12-645a58c1d7dc> in <module>
     43         super().m3()
     44
---> 45 C.m2()  # calling child class class method

<ipython-input-12-645a58c1d7dc> in m2(cls)
     32     def m2(cls):
     33        # super().__init__()
---> 34         super().m1()
     35         super().m2()
     36         super().m3()

TypeError: m1() missing 1 required positional argument: 'self'
```

V.

```
class P:

    def __init__(self):
        print('Parent Constructor')

    def m1(self):
        print('Parent instance method')

    @classmethod
    def m2(cls):
        print('Parent class method')

    @staticmethod
    def m3():
        print('Parent static method')

class C(P):

    def __init__(self):
        super().__init__()
        super().m1()
        super().m2()
        super().m3()

    def m1(self):
        super().__init__()
        super().m1()
        super().m2()
        super().m3()

    @classmethod
    def m2(cls):
       # super().__init__()
      #  super().m1()
        super().m2()
        super().m3()

    @staticmethod
    def m3():
        super().__init__()
        super().m1()
        super().m2()
        super().m3()

C.m2()  # calling child class class method
```

```
Parent class method
Parent static method
```

Important Conclusion 2:

- From child class , class method we cannot access parent class constructor and instance methods directly by using super().But we can access parent class static and class methods.

Reason:

- Class method no way related to object. Without object also we can call class method. But constructor and instance methods are always associated with object.

Another important Conclusion 3:

- From child class static method, we cannot use super() to call parent class members. But indirectly we can call parent class static and class methods.

VI.

```
class P:

    def __init__(self):
        print('Parent Constructor')

    def m1(self):
        print('Parent instance method')

    @classmethod
    def m2(cls):
        print('Parent class method')

    @staticmethod
    def m3():
        print('Parent static method')

class C(P):

    def __init__(self):
        super().__init__()
        super().m1()
        super().m2()
        super().m3()

    def m1(self):
        super().__init__()
        super().m1()
        super().m2()
        super().m3()

    @classmethod
    def m2(cls):
        super().__init__()
        super().m1()
        super().m2()
        super().m3()

    @staticmethod
    def m3():
        super().__init__()
        super().m1()
        super().m2()
        super().m3()

C.m3()  # calling child class static method
```

```
---------------------------------------------------------------------------
RuntimeError                              Traceback (most recent call last)
<ipython-input-14-f40085319c2e> in <module>
     43         super().m3()
     44
---> 45 C.m3()  # calling child class static method

<ipython-input-14-f40085319c2e> in m3()
     38     @staticmethod
     39     def m3():
---> 40         super().__init__()
     41         super().m1()
     42         super().m2()

RuntimeError: super(): no arguments
```

VII.

```
class P:

    def __init__(self):
        print('Parent Constructor')

    def m1(self):
        print('Parent instance method')

    @classmethod
    def m2(cls):
        print('Parent class method')

    @staticmethod
    def m3():
        print('Parent static method')

class C(P):

    def __init__(self):
        super().__init__()
        super().m1()
        super().m2()
        super().m3()

    def m1(self):
        super().__init__()
        super().m1()
        super().m2()
        super().m3()

    @classmethod
    def m2(cls):
        super().__init__()
        super().m1()
        super().m2()
        super().m3()

    @staticmethod
    def m3():
        # super().__init__()
        super().m1()
        super().m2()
        super().m3()

C.m3()  # calling child class class method
```

```
---------------------------------------------------------------------------
RuntimeError                              Traceback (most recent call last)
<ipython-input-16-d72e8789a768> in <module>
     43         super().m3()
     44
---> 45 C.m3()  # calling child class class method

<ipython-input-16-d72e8789a768> in m3()
     39     def m3():
     40         # super().__init__()
---> 41         super().m1()
     42         super().m2()
     43         super().m3()

RuntimeError: super(): no arguments
```

VIII.

```
class P:

    def __init__(self):
        print('Parent Constructor')

    def m1(self):
        print('Parent instance method')

    @classmethod
    def m2(cls):
        print('Parent class method')

    @staticmethod
    def m3():
        print('Parent static method')

class C(P):

    def __init__(self):
        super().__init__()
        super().m1()
        super().m2()
        super().m3()

    def m1(self):
        super().__init__()
        super().m1()
        super().m2()
        super().m3()

    @classmethod
    def m2(cls):
        super().__init__()
        super().m1()
        super().m2()
        super().m3()

    @staticmethod
    def m3():
       # super().__init__()
        #super().m1()
        super().m2()
        super().m3()

C.m3()  # calling child class class method
```

```
---------------------------------------------------------------------------
RuntimeError                              Traceback (most recent call last)
<ipython-input-17-aa6910609f58> in <module>
     43         super().m3()
     44
---> 45 C.m3()  # calling child class class method

<ipython-input-17-aa6910609f58> in m3()
     40         # super().__init__()
     41          #super().m1()
---> 42         super().m2()
     43         super().m3()
     44

RuntimeError: super(): no arguments
```

IX.

```
class P:

    def __init__(self):
        print('Parent Constructor')

    def m1(self):
        print('Parent instance method')

    @classmethod
    def m2(cls):
        print('Parent class method')

    @staticmethod
    def m3():
        print('Parent static method')

class C(P):

    def __init__(self):
        super().__init__()
        super().m1()
        super().m2()
        super().m3()

    def m1(self):
        super().__init__()
        super().m1()
        super().m2()
        super().m3()

    @classmethod
    def m2(cls):
        super().__init__()
        super().m1()
        super().m2()
        super().m3()

    @staticmethod
    def m3():
        # super().__init__()
        #super().m1()
        #super().m2()
        super().m3()

C.m3()  # calling child class class method
```

```
---------------------------------------------------------------------------
RuntimeError                              Traceback (most recent call last)
<ipython-input-18-f48143240dd2> in <module>
     43         super().m3()
     44
---> 45 C.m3()  # calling child class class method

<ipython-input-18-f48143240dd2> in m3()
     41         #super().m1()
     42         #super().m2()
---> 43         super().m3()
     44
     45 C.m3()  # calling child class class method

RuntimeError: super(): no arguments
```

X.

```
class P:

    def __init__(self):
        print('Parent Constructor')

    def m1(self):
        print('Parent instance method')

    @classmethod
    def m2(cls):
        print('Parent class method')

    @staticmethod
    def m3():
        print('Parent static method')

class C(P):

    def __init__(self):
        super().__init__()
        super().m1()
        super().m2()
        super().m3()

    def m1(self):
            super().__init__()
            super().m1()
            super().m2()
            super().m3()

    @classmethod
    def m2(cls):
        super().__init__()
        super().m1()
        super().m2()
        super().m3()

    @staticmethod
    def m3():
       # super().__init__()
        #super().m1()
        #super().m2()
        #super().m3()
        pass

C.m3()  # calling child class class method
```

Q. From class method of child class, how to call parent class constructor and instance methods indirectly???

I.

```
class P:

    def __init__(self):
        print('Parent class constructor ')

    def m1(self):
        print('Parent Instance Method ')

class C(P):
    @classmethod
    def m2(cls):
        super(C,cls).__init__(cls)
        super(C,cls).m1(cls)

c = C()
c.m2()          # Calling through reference variable
```

```
Parent class constructor
Parent class constructor
Parent Instance Method
```

II.

```
class P:

    def __init__(self):
        print('Parent class constructor ')

    def m1(self):
        print('Parent Instance Method ')

class C(P):
    @classmethod
    def m2(cls):
        super(C,cls).__init__(cls)
        super(C,cls).m1(cls)


C.m2()   # calling through class name
```

```
Parent class constructor
Parent Instance Method
```

Q. How to call Parent class static and class methods from child class static method ???

```
class P:

    @classmethod
    def m2(cls):
        print('Parent Class Method')

    @staticmethod
    def m3():
        print('Parent Static Method ')

class C(P):
    @staticmethod
    def m2():
        super(C,C).m2()
        super(C,C).m3()


C.m2()
```

```
Parent Class Method
Parent Static Method
```

Any question?

If you try to practice programs yourself, then you will learn many things automatically

Spend few minutes and then enjoy the study

Thank You