

Working with Methods: Examples

You have learned how to use objects, classes and the methods inside them. You have covered these both in cases where there is only one class, as well as when there are multiple classes. You also explored how multiple inheritance works in Python and the role Method Resolution Order(MRO) plays in determining the call for the method.

The following examples demonstrate how the function call is resolved in cases of multiple inheritance in different scenarios. Note that all the functions have the same names in all of the examples.

Example 1

```

1  # Example 1
2  class A:
3      def a(self):
4          return "Function inside A"
5
6  class B:
7      def a(self):
8          return "Function inside B"
9
10 class C(B,A):
11     pass
12
13 # Driver code
14 c = C()
15 print(c.a())

```

Output:

```

1  Function inside B

```

Class C inherits from classes B and A. When I don't find any function a() inside class C, I should search for classes B and A and its important that I do it in that order.

I will now add one more level to this and note the output.

Example 2

```

1  class A:
2      def b(self):
3          return "Function inside A"
4
5  class B:
6      def b(self):
7          return "Function inside B"
8
9  class C(A, B):
10     def b(self):
11         return "Function inside C"
12     pass
13
14 class D(C):
15     pass
16
17 d = D()
18 print(d.b())

```

Output:

```

1  Function inside C

```

Class D inherits from class C, which in turn inherits from classes A and B. Class D accesses the immediate superclass of class D, which is class C and resolves the value of the variable once it's found in that superclass.

Now let's say I comment out the declaration inside class C.

```
1  # def b(self):
2  #     return "Function inside C"
```

And replace it with the **pass** keyword to keep the code functional.

Since there was no value present inside class C either, the function call above would go to A. That is because class C will point to class A as having higher precedence while inheriting.

Now let's take another example of a similar scenario.

Example 3

```
1  class A:
2      def c(self):
3          return "Function inside A"
4
5  class B:
6      def c(self):
7          return "Function inside B"
8
9  class C(A, B):
10     def c(self):
11         return "Function inside C"
12
13  class D(A, C):
14     pass
15
16  d = D()
17  print(d.a)
```

The output is:

```
1  Traceback (most recent call last):
2      File "/Users/intropython/PycharmProjects/practicePython/inherit.py", line 10, in <module>
3          class D(A, C):
4      TypeError: Cannot create a consistent method resolution
5      order (MRO) for bases A, C
```

Note that this throws an error. In the code above, class D inherits from both class A and class C.

Class C is its immediate superclass, but since this is multiple inheritance, the rules are more complicated and it also has to check the classes passed to it for precedence.

In this particular case, class D is unable to resolve the order that should be followed, while resolving the value for the variable in cases where the variable is not present in the class of the given object.

It results in a `TypeError` because it's unable to create method resolution order (MRO). MRO is Python's way of resolving the order of precedence of classes while dealing with inheritance.

Let's examine one final example.

Example 4

```
1  class A:
2      def d(self):
3          return "Function inside A"
4
5  class B:
6      def d(self):
7          return "Function inside B"
```

```
8
9
10 class C:
11     def d(self):
12         return "Function inside C"
13
14
15 class D(A, B):
16     def d(self):
17         return "Function inside D"
18
19
20 class E(B, C):
21     def d(self):
22         return "Function inside E"
23
24
25 class F(E,D,C):
26     pass
27
28 f = F()
29 print(f.d())
30 print(F.mro())
```




Output:

```
1 Function inside E
2 [<class '__main__.F'>, <class '__main__.E'>, <class '__main__.D'>, <class '__main__.A'>, <class '__main__.B'>, <class 'object'>]
```

The code here is simple. class F directly inherits from its immediate superclass and the first class that is passed to it. The second line then demonstrates the return from the mro() function.

The examples in this reading demonstrate how code in which multiple inheritance is used, can get complicated and very messy, very fast. Multiple inheritance, with all the advantages and flexibility that it provides, should only be used once you have a strong command of Python as a language to avoid creating 'spaghetti code' that's difficult to understand and update.

Mark as completed

 Like  Dislike  Report an issue