

define a simple base class with a method we can trace

```
class A:
    def foo(self):
        print("A")
```

define another simple base class with a method we can trace

```
class B:
    def foo(self):
        print("B")
```

Now define a class C that inherits from both A and B, in that order

```
class C(A,B):
    def foo(self):
        print("C")
        super().foo()
```

C.mro()

```
[<class '__main__.C'>, <class '__main__.A'>, <class '__main__.B'>, <class 'object'>]
```

New class D, now with base classes in the opposite order

```
class D(B,A):
    def foo(self):
        print("D")
        super().foo()
```

D.mro()

```
[<class '__main__.D'>, <class '__main__.B'>, <class '__main__.A'>, <class 'object'>]
```

In multiple inheritance, `super()` doesn't call a class's direct parent but instead calls the next class in the method resolution order (MRO). Because the MRO is built from the order in which classes are inherited, changing that order also changes which method `super()` chooses. And the call chain continues only if each class in the hierarchy also uses `super()`.