```
>>> class Base1:
...     def print(self):
...         print("Base1")
...
>>> class Base2():
...     def print(self):
...         print("Base2")
...
>>> class Derived(Base1, Base2):
...     def print(self):
...         super().print()
...
>>> foo = Derived()
>>> foo.print()
Base1
>>> class Derived2(Base2, Base1):
...     def print(self):
...         super().print()
...
>>> foo2 = Derived2()
>>> foo2.print()
Base2
>>> class Base3():
...     def print(self):
...          print("Base3")
...
>>> class Derived3(Base3, Base1, Base2):
...     def print(self):
...          super().print()
...
>>> foo3 = Derived3()
>>> foo3.print()
Base3
>>> class Base2():
...     def print(self):
...          print("Base2")
...     def print_other(self):
...          print("FOO")
...
>>> class Derived(Base1, Base2):
...     def print(self):
...          super().print()
...     def print_other(self):
...          super().print_other()
```

```
...
>>> foo = Derived()
>>> foo.print()
Base1
>>> foo.print_other()
FOO
>>>
```

The super() method follows the order of Method Resolution Order in python. Therefore, when calling the super() method and getting an attribute from the base classes, Python uses the order found based on the MRO to look for attributes. Therefore, if Python does not find an attribute in the first base class, it will look through the next, and keep looking until either it finds the attribute or throws an attribute error.