Handling Multiple Inheritance in Python

# 1. Introduction

In Python, multiple inheritance allows a class to inherit from more than one parent class. While powerful, it can introduce complexities when different parent classes define methods with the same name. Python handles this complexity using a well-defined mechanism called the Method Resolution Order (MRO), and the built-in super() function helps manage this hierarchy effectively.

# 2. Understanding super() in Multiple Inheritance

The super() function in Python is used to call a method from a parent class. In multiple inheritance scenarios, super() works with the MRO to determine the next class’s method to invoke, rather than just calling the immediate parent. This helps avoid redundant method calls and supports cooperative multiple inheritance.

# 3. The Role of Method Resolution Order (MRO)

Python uses the C3 linearization algorithm to determine the MRO. This ensures that every class appears before its parents and that shared base classes are only called once. The super() function respects this order when chaining method calls.

# 4. Code Example

Here’s a simple illustration:  
  
class A:  
 def show(self):  
 print("A")  
  
class B(A):  
 def show(self):  
 print("B")  
 super().show()  
  
class C(A):  
 def show(self):  
 print("C")  
 super().show()  
  
class D(B, C):  
 def show(self):  
 print("D")  
 super().show()  
  
d = D()  
d.show()

# 5. Common Scenario: Conflicting Method Names

If classes Human and Mammal both define a method named eat(), and a Child class inherits from both, the method resolution depends on the MRO. If both parent classes use super(), the method calls will chain in the MRO order, avoiding duplication.

# 6. Conclusion

Using super() with multiple inheritance enables clean and maintainable code. It respects the MRO and ensures that methods are not invoked multiple times. This is especially useful in complex hierarchies where multiple base classes define similar methods.