1. How super() Function Handles Multiple Inheritance

In Python, the super() function is used to call methods from a parent or superclass without explicitly naming them. This becomes especially powerful in multiple inheritance scenarios, where it helps manage the **Method Resolution Order (MRO)** to ensure that each parent class is called only once and in a predictable order.

Python uses the **C3 Linearization algorithm** to determine the MRO. This means that when a method is called via super(), Python will:

- Look up the class hierarchy following the MRO sequence.
- Call the **next method** in the MRO that matches the method name.
- Ensure each class in the inheritance tree is called only once, preventing duplication and infinite recursion.

```python
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()
```

Output :

```
D
B
C
A
```

Explanation: The MRO for class D(B, C) is: D → B → C → A. Each class uses super() to call the next one in line.

2. Handling Method Name Conflicts in Multiple Inheritance

If two parent classes (e.g., Human and Mammal) have methods with the same name (e.g., eat()), but with different implementations, Python uses the **MRO** to resolve which method gets called when the child class (e.g., Employee) calls eat().

```python
class Human:
    def eat(self):
        print("Human eats with utensils")

class Mammal:
    def eat(self):
        print("Mammal eats raw food")

class Employee(Human, Mammal):
    pass

e = Employee()
e.eat()
```

Output:

```
Human eats with utensils
```

Explanation:
The method resolution follows the order of inheritance: Employee(Human, Mammal). Since Human is listed first, Python looks for eat() in Human before Mammal.