1. How super() Function Handles Multiple Inheritance in Python?

In Python, the super() function handles multiple inheritance by following the Method Resolution Order (MRO), which is determined using the C3 linearization algorithm.

This ensures a consistent and predictable order for method resolution across complex inheritance hierarchies.

When super() is called (super().method()), it doesn't simply call the immediate parent class's method.

Instead, it returns a proxy object that delegates the method call to the next class in the MRO that implements the method.

Example:

```
class A:
    def show(self):
        print("A.show() called")

class B(A):
    def show(self):
        print("B.show() called")
        super().show()

class C(A):
    def show(self):
        print("C.show() called")
        super().show()

class D(B, C):  # Multiple inheritance
    def show(self):
        print("D.show() called")
        super().show()

# Check the MRO
print("Method Resolution Order (MRO):")
for cls in D.__mro__:
    print(cls.__name__)

# Run the method
print("\nCalling d.show():")
d = D()
d.show()
```

Output:

```
Method Resolution Order (MRO):
B
C
A
object

Calling d.show():
D.show() called
B.show() called
C.show() called
A.show() called
```

Explanation

1. `super().show()` in each class delegates to the next class in the MRO chain.
2. Python doesn't call the direct parent, but rather the next in line according to MRO.
3. This ensures that each class's method is called once and in the correct order.
4. It prevents duplication and resolves complex inheritance issues (like the diamond problem).

2. If Human and Mammal Have the Same Method (eat) but with different implementation. When Child [Employee] calls eat method how python handle this case?

When `Human` and `Mammal` both have a method named `eat()` but with different implementations, and a child class `Employee` inherits from both, Python uses the Method Resolution Order (MRO) to decide which `eat()` method will be called.

How Python Handles It:

- Python uses the C3 linearization algorithm to build the MRO.
- When `Employee` calls `eat()`, Python looks up the method in the MRO order:

```
Employee → Human → Mammal → object
```

- So the first class in MRO that defines `eat()` is the one that gets called.
- If you use `super().eat()` in the `Employee` class, Python continues the chain following the MRO.

Example:

```python
class Mammal:
    def eat(self):
        print("Mammal eats with tongue")

class Human:
    def eat(self):
        print("Human eats with spoon")

class Employee(Human, Mammal):# Order of inheritance matters
    def eat(self):
        print("Employee eats at office")
        super().eat()

e = Employee()
e.eat()
# Show MRO
print("Method Resolution Order (MRO):")
for cls in Employee.__mro__:
    print(cls.__name__)
```

Output:

```
Employee eats at office
Human eats with spoon
Method Resolution Order(MRO):
Employee
Human
Mammal
object
```

Conclusion:

- Python will call `Human.eat()` because it comes first in the MRO.
- The `super().eat()` in `Employee` triggers the next method in the MRO, not necessarily the direct parent.
- If you want `Mammal.eat()` to be called first → class Employee(Mammal, Human):