Using super() in Multiple Inheritance in Python

What is super()?

The super() function in Python is used to call methods from a parent class. It's most commonly used in inheritance scenarios to allow subclasses to access and extend behavior from their parent classes.

Using super() in Multiple Inheritance

In multiple inheritance, a class inherits from more than one parent class. Python uses a system called MRO (Method Resolution Order) to decide the order in which base classes are initialized when super() is used. super() ensures that each parent class is only initialized once, and in the correct order — even in complex inheritance hierarchies.

Why is it useful?

- Avoids duplicate code

- Ensures all parent constructors are called

- Maintains clean and scalable code

- Prevents redundant or conflicting initializations

My Opinion

Using super() in multiple inheritance is a best practice, especially when all parent classes inherit from object and are designed to cooperate. IT keeps the code maintainable, especially in complex systems where multiple parent classes have shared logic.

Example

```
class A:

  def __init__(self):

    print("Init A")


class B(A):

  def __init__(self):

    super().__init__()

    print("Init B")
```

```
class C(A):
    def __init__(self):
        super().__init__()
        print("Init C")


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


obj = D()
```

Output

```
Init A
Init C
Init B
Init D
```

Even though class A is a parent of both B and C, it was called only once. That's because super() follows the MRO, preventing duplicate initializations.

Conclusion

Using super() in multiple inheritance is:

- Safe: avoids redundant calls
- Organized: respects the MRO
- Efficient: ensures every parent is initialized once

It's the recommended approach in cooperative multiple inheritance.