Demonstrating `super()` Behavior in Python Multiple Inheritance

Objective

To demonstrate and understand the behavior of the `super()` function in Python's multiple inheritance context.

Python Code Example

```
class A:
    def process(self):
        print("A.process")
        super().process()

class B:
    def process(self):
        print("B.process")
        super().process()

class C:
    def process(self):
        print("C.process")

class D(A, B, C):
    def process(self):
        print("D.process")
        super().process()

d = D()
d.process()
```

Output

```
D.process
A.process
B.process
C.process
```

Explanation

Class Hierarchy

The inheritance hierarchy is as follows:

$$\mathtt{D} \rightarrow \mathtt{A} \rightarrow \mathtt{B} \rightarrow \mathtt{C}$$

Method Resolution Order (MRO)

The MRO determines the order in which Python looks for methods during method calls. For class `D`, the MRO is:

```
(<class '__main__.D'>, <class '__main__.A'>, <class '__main__.B'>, <class '__main__.C'>, <class 'object
```

How `super()` Works

- `d.process()` starts with class `D`.
- `super().process()` in `D` calls `A.process()` (next in MRO).
- `super().process()` in `A` calls `B.process()`.
- `super().process()` in `B` calls `C.process()`.
- `C.process()` does not call `super()`, so the chain stops.

Key Takeaways

- `super()` uses the MRO, not the direct parent class.
- It enables cooperative multiple inheritance.
- All classes must use `super()` for the chain to continue.
- MRO follows C3 linearization rules.

References

- Python documentation on `super()`: `https://docs.python.org/3/library/functions.html#super`
- Real Python guide: `https://realpython.com/python-super/`