

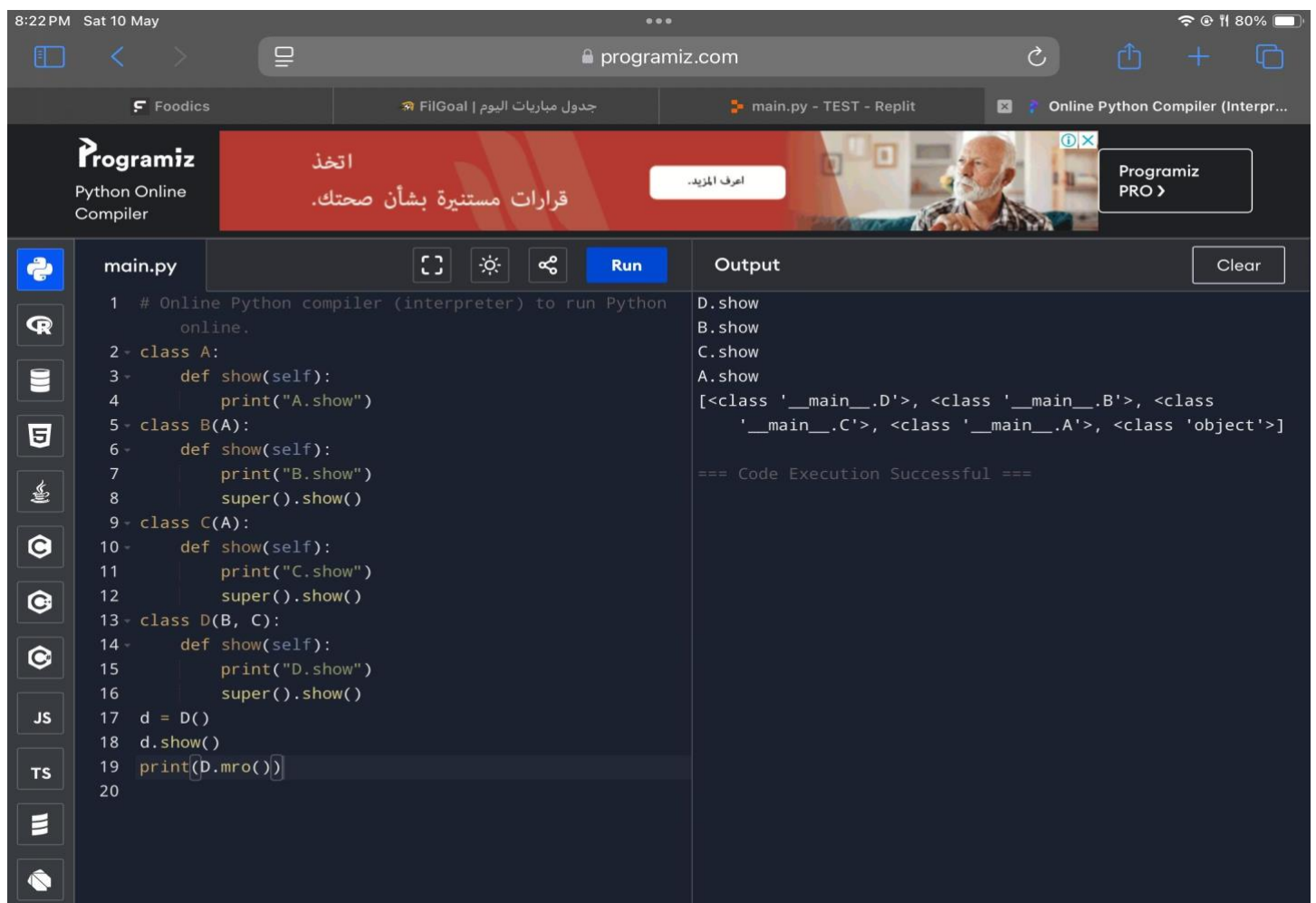
Abdall Khaled Alaassar

How super() Handles Multiple Inheritance

In multiple inheritance, the `super()` function follows the Method Resolution Order (MRO) to determine which class's method to call.

Python uses the C3 Linearization Algorithm to compute the MRO.

This ensures that `super()` doesn't just call the parent directly, but follows a predictable and consistent order across all inherited classes.



The screenshot shows the Programiz Online Python Compiler interface. The code editor contains the following Python code:

```
1 # Online Python compiler (interpreter) to run Python online.
2 class A:
3     def show(self):
4         print("A.show")
5 class B(A):
6     def show(self):
7         print("B.show")
8         super().show()
9 class C(A):
10    def show(self):
11        print("C.show")
12        super().show()
13 class D(B, C):
14    def show(self):
15        print("D.show")
16        super().show()
17 d = D()
18 d.show()
19 print(D.mro())
20
```

The output window displays the following results:

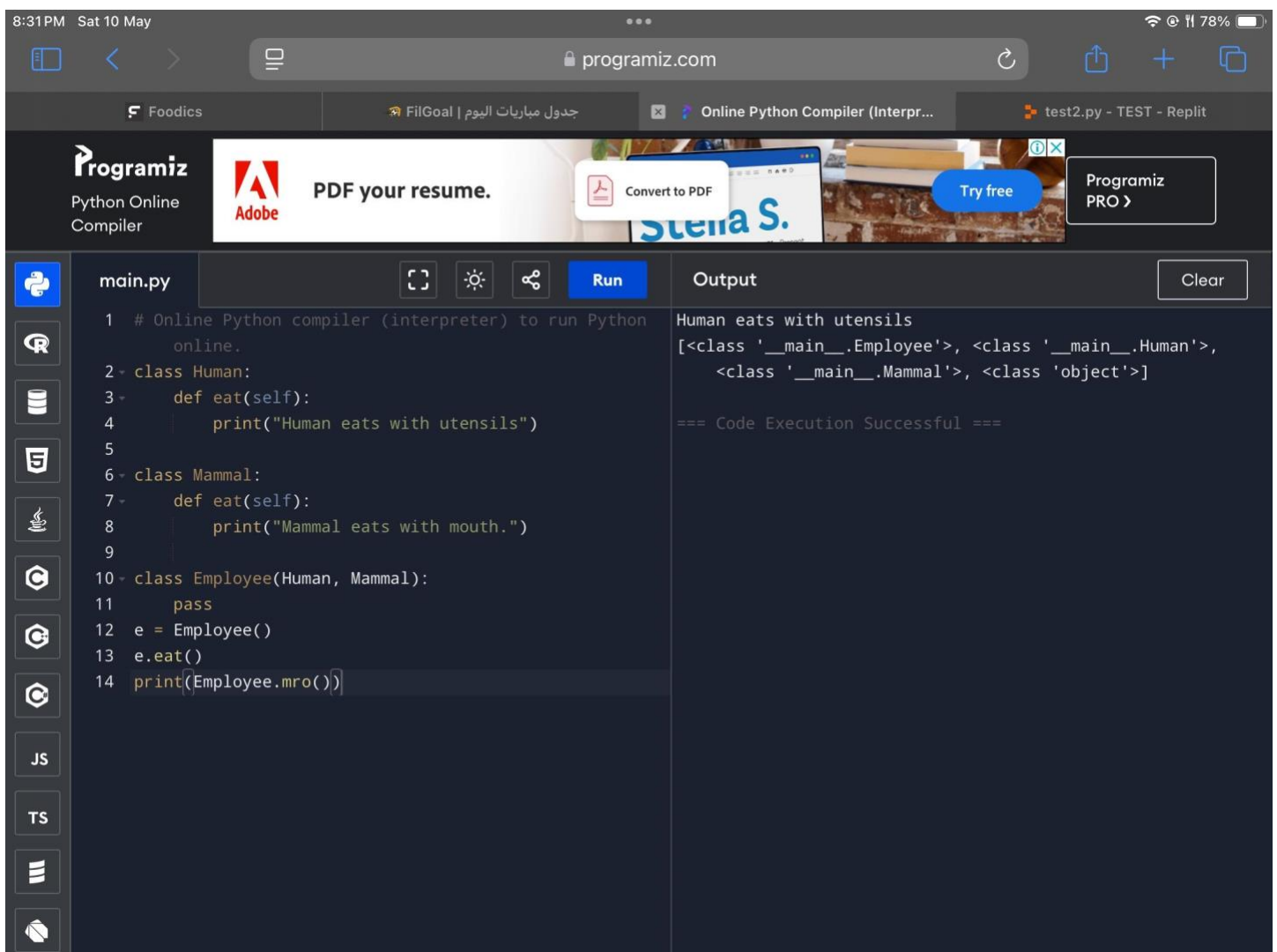
```
D.show
B.show
C.show
A.show
[<class '__main__.D'>, <class '__main__.B'>, <class '__main__.C'>, <class '__main__.A'>, <class 'object'>]
=== Code Execution Successful ===
```

Explanation:

- super() in class D refers to B, which in turn calls C, then A.
- This chain is based on the MRO: D -> B -> C -> A -> object.

2. Method Conflict: Same Method in Multiple Parents

If two parent classes define the same method (e.g., eat) with different implementations, the method that gets executed in the child class depends on the MRO.



The screenshot shows the Programiz Python Online Compiler interface. The code in main.py defines three classes: Human, Mammal, and Employee. Human has an eat method that prints "Human eats with utensils". Mammal has an eat method that prints "Mammal eats with mouth.". Employee inherits from both Human and Mammal and does not have its own eat method. The code creates an Employee instance 'e' and calls e.eat(). The output shows the result of the method call and the Method Resolution Order (MRO) for the Employee class.

```
1 # Online Python compiler (interpreter) to run Python online.
2 class Human:
3     def eat(self):
4         print("Human eats with utensils")
5
6 class Mammal:
7     def eat(self):
8         print("Mammal eats with mouth.")
9
10 class Employee(Human, Mammal):
11     pass
12 e = Employee()
13 e.eat()
14 print(Employee.mro())
```

Output:

```
Human eats with utensils
[<class '__main__.Employee'>, <class '__main__.Human'>,
 <class '__main__.Mammal'>, <class 'object'>]

=== Code Execution Successful ===
```

Explanation:

- Employee inherits from Human and Mammal, in that order.
- So, the MRO is: Employee -> Human -> Mammal -> object.
- Therefore, e.eat() calls Human.eat().

Conclusion

- Python uses MRO to resolve method calls in multiple inheritance.
- The super() function follows this MRO, not the immediate parent.
- Method conflicts are resolved by the order in which base classes are listed.

Resources

- <https://docs.python.org/3/library/functions.html#super>
- <https://www.python.org/download/releases/2.3/mro/>
- <https://www.geeksforgeeks.org/super-keyword/>