



EMAN AHMED HAMDY

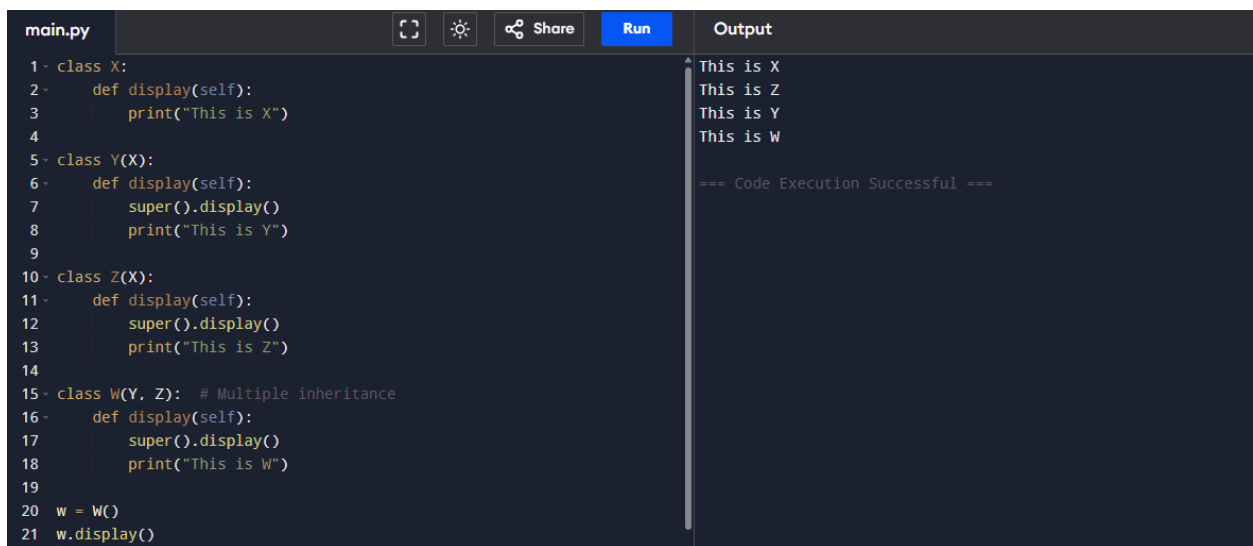
OOP Report



1. How super Function handle Multiple Inheritance.

- `super()` is a powerful tool in Python, particularly useful in complex multiple inheritance scenarios. It helps to avoid direct base class references, which can make the code less flexible and harder to maintain. By correctly using `super()`, developers ensure a smooth method resolution path that respects the order and logic of the base classes

An example:




```
main.py  [Icons]  Share  Run  Output
1 - class X:
2 -     def display(self):
3 -         print("This is X")
4
5 - class Y(X):
6 -     def display(self):
7 -         super().display()
8 -         print("This is Y")
9
10 - class Z(X):
11 -     def display(self):
12 -         super().display()
13 -         print("This is Z")
14
15 - class W(Y, Z): # Multiple inheritance
16 -     def display(self):
17 -         super().display()
18 -         print("This is W")
19
20 w = W()
21 w.display()
```

This is X
This is Z
This is Y
This is W

=== Code Execution Successful ===

2. Human and Mammal Have the same method as eat but with different Implementation. When Child [Employee] calls eat method how python handle this case.

- The first eat() it finds is in Human, so that's the one it uses.

main.py	Run	Output
<pre>1 ~ class Human: 2 ~ def eat(self): 3 ~ print("Human eats with utensils.") 4 ~ 5 ~ class Mammal: 6 ~ def eat(self): 7 ~ print("Mammal eats instinctively.") 8 ~ 9 ~ class Employee(Human, Mammal): 10 ~ pass 11 ~ 12 ~ e = Employee() 13 ~ e.eat() 14 ~</pre>		<pre>Human eats with utensils. === Code Execution Successful ===</pre>