MRO

MRO `(Method Resolution Order)` is the order in which Python searches for a method or attribute in a class hierarchy—especially when multiple inheritance is used.

Simple words me:
`When you call a method on an object, MRO decides which class's method should run first`

```
class A:
    def home(self):
        print("Home in A")

class B(A):
    def home(self):
        print("Home in B")
        super().home()

obj = B()
obj.home()
print(B.mro())


# O/P:--
Home in B
Home in A
[<class '__main__.B'>, <class '__main__.A'>, <class 'object'>]
```

```
class A:
    def home(self):
        print("Home in A")

class B(A):
    def home(self):
        print("Home in B")
        super().home()

class C(B):
    def home(self):
        print("Home in C")
        super().home()

obj = C()
```

```
obj.home()
print(C.mro())

# O/P:--
Home in C
Home in B
Home in A
[<class '__main__.C'>, <class '__main__.B'>, <class '__main__.A'>, <class
'object'>]
```

```
class A:
    def home(self):
        print("Class A")

class B(A):
    def home(self):
        print("Class B")
        super().home()

class C(A):
    def home(self):
        print("Class C")
        super().home()

class D(A):
    def home(self):
        print("Class D")
        super().home()

class E(B,C,D):
    def home(self):
        print("Class E")
        super().home()

obj = E()
obj.home()
print(E.mro())

# O/P:--
Class E
Class B
Class C
Class D
```

```
Class A
[<class '__main__.E'>, <class '__main__.B'>, <class '__main__.C'>, <class
'__main__.D'>, <class '__main__.A'>, <class 'object'>]
```

```
class A:
    def home(self):
        print("Class A")

class B(A):
    def home(self):
        print("Class B")
        super().home()

class C(A):
    def home(self):
        print("Class C")
        super().home()

obj1 = B()
obj1.home()

print(B.mro())

obj2 = C()
obj2.home()
print(C.mro())

# O/P:--

Class B
Class A
[<class '__main__.B'>, <class '__main__.A'>, <class 'object'>]
Class C
Class A
[<class '__main__.C'>, <class '__main__.A'>, <class 'object'>]
```