Q1: How super function handle multiple inheritance?

Consider next example :

super() is usually used in multiple inheritance to call parent classes functions in child class.

```python
class Father:
    def print_fname(self, name):
        print(f'father name is : {name}')


class Mother:
    def print_mname(self, name):
        print(f'mother name is : {name}')


class Child(Father, Mother):
    def print_parent_names(self, fname, mname):
        super(Child, self).print_fname(fname)
        super(Child, self).print_mname(mname)


new_child = Child()
new_child.print_parent_names("Mikel", "Anne")
```

```
father name is : Mikel
mother name is : Anne

Process finished with exit code 0
```

Q2: How python handles same function name in different parent classes in multiple inheritance ?

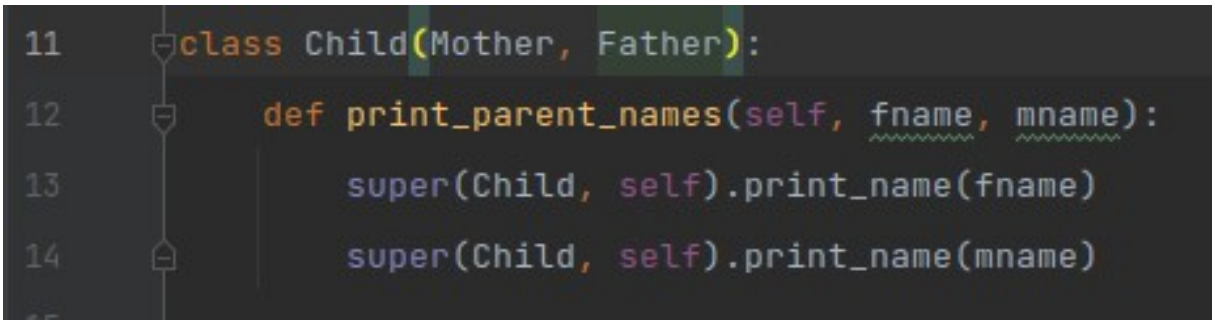
Python interpreter search for the called function – in the child class – in its parents class depending on something called Method Resolution Order (MRO). The method resolution order (or **MRO**) tells Python how to search for inherited methods. This comes in handy when you're using `super()` because the MRO tells you exactly where Python will look for a method you're calling with `super()` and in what order.

In next example we have 2 function with the same name in both parent classes and we print mro of child class. let's what will happen depending on that mro.

```python
class Father:
    def print_name(self, name):
        print(f'father name is : {name}')


class Mother:
    def print_name(self, name):
        print(f'mother name is : {name}')


class Child(Father, Mother):
    def print_parent_names(self, fname, mname):
        super(Child, self).print_name(fname)
        super(Child, self).print_name(mname)

new_child = Child()
new_child.print_parent_names("Mikel", "Anne")
print(Child.__mro__)
```

```
father name is : Mikel
father name is : Anne
(<class '__main__.Child'>, <class '__main__.Father'>, <class '__main__.Mother'>, <class 'object'>)
```

MRO of child class say that any called function will search for it in Child class then Father class than Mother Class.
When we look at the output we will find that every time we call name function , interpreter searches for it in Father class first , finds it and executes it. So it prints “father name” twice.
we can control mro and force interpreter to search in mother class first by change order of inherited classes in Child class signature as follows:



```
class Child(Mother, Father):
    def print_parent_names(self, fname, mname):
        super(Child, self).print_name(fname)
        super(Child, self).print_name(mname)
```

```
mother name is : Mikel
mother name is : Anne
(<class '__main__.Child'>, <class '__main__.Mother'>, <class '__main__.Father'>, <class 'object'>)
```

Q3: Can python implement overloading ?

Overloading can be implemented in python using Multiple Dispatch Decorator. Multiple Dispatch Decorator Can be installed by:

```
pip3 install multipledispatch
```

Example:

```
from multipledispatch import dispatch

#passing one parameter
@dispatch(int,int)
def product(first,second):
    result = first*second
    print(result);

#passing two parameters
@dispatch(int,int,int)
def product(first,second,third):
    result  = first * second * third
    print(result);

#you can also pass data type of any value as per requirement
@dispatch(float,float,float)
def product(first,second,third):
    result  = first * second * third
    print(result);


#calling product method with 2 arguments
product(2,3,2) #this will give output of 12
product(2.2,3.4,2.3) # this will give output of 17.98599999999999
```

Output:

```
12
17.985999999999997
```