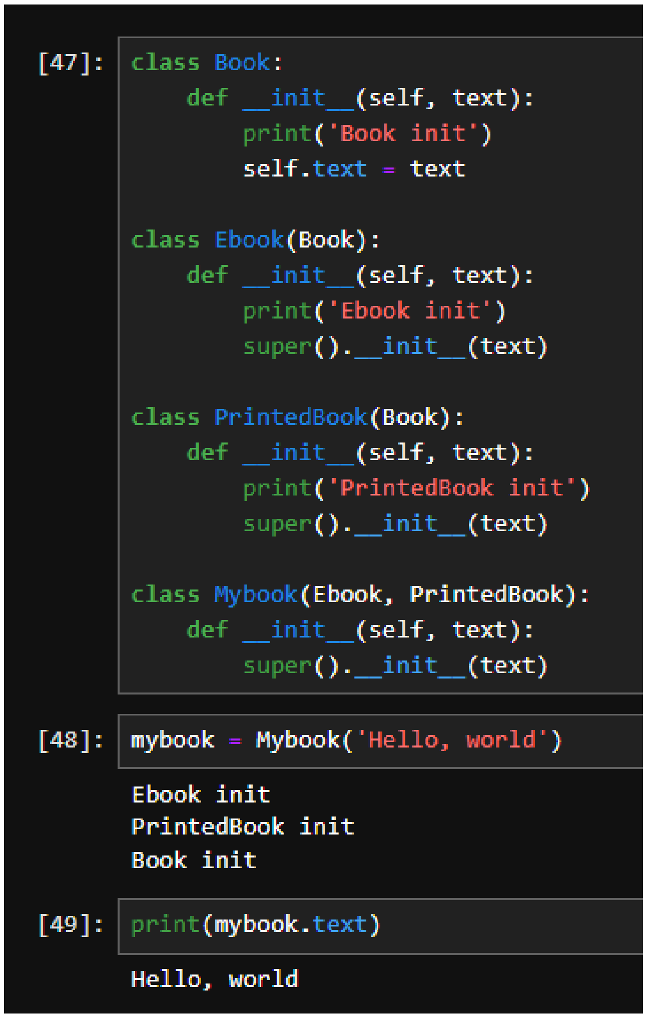
**Name:** Mohamed Mohamed Alsayed Hassan

---

## How super Function Handle Multiple Inheritance

The **super()** function is used to give access to methods and properties of a parent or sibling class. Let's look at some examples to see how super() function handle multiple inheritance.

### Example 1:

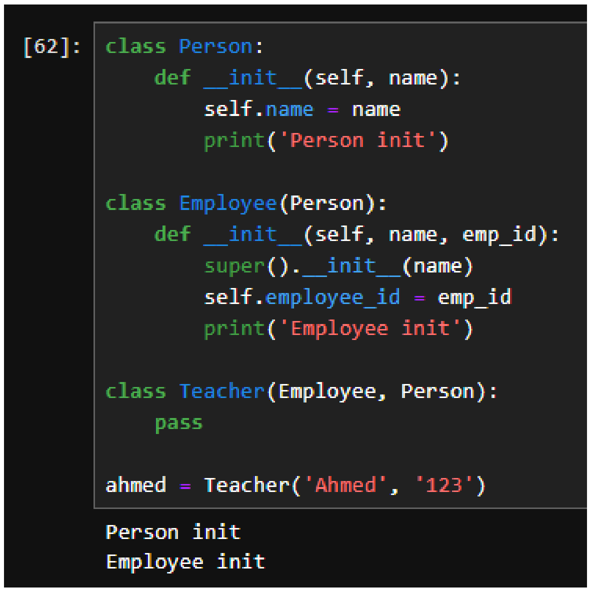


```
[47]: class Book:
          def __init__(self, text):
              print('Book init')
              self.text = text

      class Ebook(Book):
          def __init__(self, text):
              print('Ebook init')
              super().__init__(text)

      class PrintedBook(Book):
          def __init__(self, text):
              print('PrintedBook init')
              super().__init__(text)

      class Mybook(Ebook, PrintedBook):
          def __init__(self, text):
              super().__init__(text)

[48]: mybook = Mybook('Hello, world')

      Ebook init
      PrintedBook init
      Book init

[49]: print(mybook.text)

      Hello, world
```

When it comes to multiple inheritance, super() function is very helpful. In the Mybook class, you might think Book.__init__() will be called twice for both Ebook and PrintedBook, but super() function is smart enough to call it just once.

Let's look at another example.

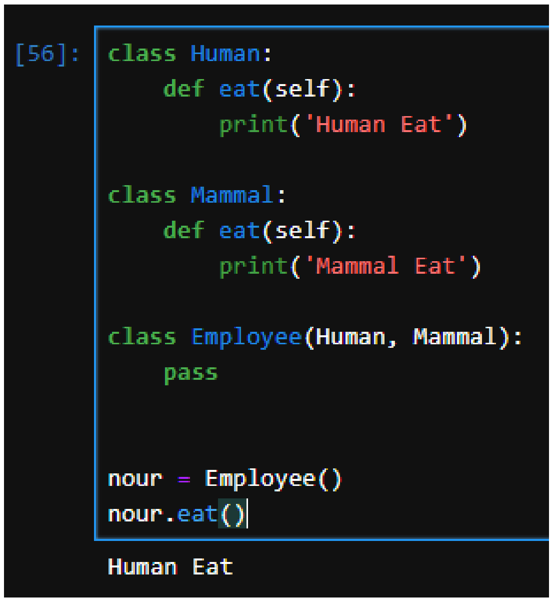
## Example 2:



```
[62]: class Person:
          def __init__(self, name):
              self.name = name
              print('Person init')

      class Employee(Person):
          def __init__(self, name, emp_id):
              super().__init__(name)
              self.employee_id = emp_id
              print('Employee init')

      class Teacher(Employee, Person):
          pass

      ahmed = Teacher('Ahmed', '123')

      Person init
      Employee init
```

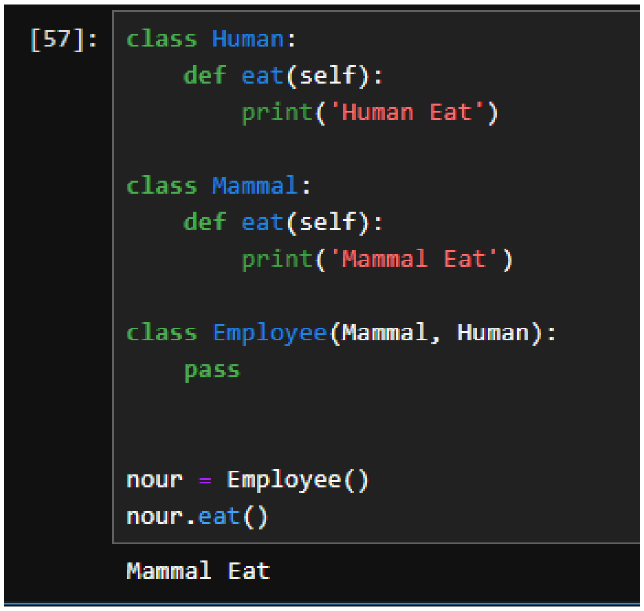
Here, it follows the method resolution order, it looks for __init__ at Employee class first. There, it will find the Person.__init__ function and execute it, then finishes the Employee.__init__ function. It does not execute the Person__init__ function again for the inherited Person class since it is already executed in the Employee.__init__ function.

**Example 3:** If Human and Mammal Have the same method like eat but with different Implementation. When Child[Employee] calls eat method how python handle this case.



```
[56]: class Human:
          def eat(self):
              print('Human Eat')

      class Mammal:
          def eat(self):
              print('Mammal Eat')

      class Employee(Human, Mammal):
          pass


      nour = Employee()
      nour.eat()

      Human Eat
```

To explain what happened here, let's first define the **MRO.** The **MRO** stands for Method Resolution Order. It is the order in which Python looks for a method in a hierarchy of classes. In multiple inheritance, methods are executed based on the order specified while inheriting the classes. In our example, it searches for the ‘eat’ function from the left to the right class. First, it looks for an ‘eat’ function in the Human class. Since Human class has ‘eat’ function, it will be executed. If we reverse the classes and put Mammal first, its ‘eat’ function will be executed instead.



```
[57]: class Human:
          def eat(self):
              print('Human Eat')

      class Mammal:
          def eat(self):
              print('Mammal Eat')

      class Employee(Mammal, Human):
          pass


      nour = Employee()
      nour.eat()

      Mammal Eat
```