# Super Effect on Multiple Inheritance

## 1. Introduction: What Are Classes and Inheritance

In Python, a class is like a template for creating objects. Think of it as a blueprint for something you want to build. Each class can have attributes (like name, age, etc.) and methods (functions that belong to the class, like speak() or eat()).When one class inherits from another, we call that inheritance. It means the child class (subclass) gets all the features of the parent class (superclass) without having to rewrite them.

Example:

```python
class Person:
    def __init__(self, name):
        self.name = name

    def speak(self):
        print(f"My name is {self.name}")
```

You can now make a new class like '`Employee`' that inherits from '`Person`' and automatically gets its '`__init__`' and '`speak`' method.

## 2. How the super() Function Handles Multiple Inheritance

In Python, multiple inheritance allows a class to inherit from more than one parent class. The super() function is used to call methods from a parent class, and Python uses Method Resolution Order (MRO) to decide which method to run.
Example from inheritance.py:

F: > ITI Cyber Security > Python > report.py > ...

```python
class Person:
    def __init__(self, name):
        self.name = name
        print(f"Person initialized with name: {name}")

class Employee(Person):
    def __init__(self, name, employee_id):
        # super().__init__(name)
        self.employee_id = employee_id
        print(f"Employee initialized with name: {name} and ID: {employee_id}")

class teacher(Employee, Person):
    pass

x = teacher("omar", "Q852")
```

Output:

```
Employee initialized with name: omar and ID: Q852
PS F:\ITI Cyber Security\Python\project>
```

Note: Since 'Employee' didn't call 'super().__init__()', the Person constructor was skipped. So, only the 'Employee' message appears.

If we use super():

F: > ITI Cyber Security > Python > report.py > teacher

```python
class Person:
    def __init__(self, name):
        self.name = name
        print(f"Person initialized with name: {name}")

class Employee(Person):
    def __init__(self, name, employee_id):
        super().__init__(name)
        self.employee_id = employee_id
        print(f"Employee initialized with name: {name} and ID: {employee_id}")


class teacher(Employee, Person):
    pass

x = teacher("omar", "Q852")
```

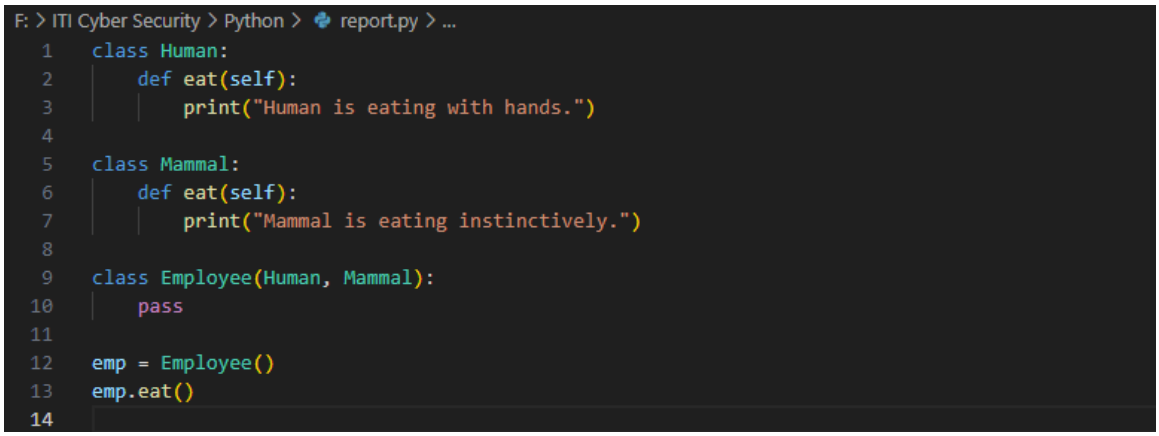
Output:

```
Person initialized with name: omar
Employee initialized with name: omar and ID: Q852
PS F:\ITI Cyber Security\Python\project>
```

This time, 'super()' ensures that 'Person' constructor runs first. Then 'Employee' constructor continues as normal.

## 3. What Happens When Two Parent Classes Have the Same Method Name

If two parent classes have the same method, Python uses the MRO to determine which one to run.
Example:




Output:

```
[Running] python -u "f:\ITI Cyber Security\Python\report.py"
Human is eating with hands.
```

If we reverse the order of inheritance:


Output:

```
[Running] python -u "f:\ITI Cyber Security\Python\report.py"
Mammal is eating instinctively.
```

Conclusion:

Multiple inheritance is a useful feature that allows code reuse across classes. However, when methods overlap, developers must be careful and use super() and understand the method resolution order. In the workplace, this helps avoid bugs and makes sure all necessary functionality runs as expected.