Multiple Inheritance in Python

1. How super() Handles Multiple Inheritance

In Python, super() is used to call methods in a parent class following the Method Resolution Order (MRO). In multiple inheritance, Python uses the C3 linearization algorithm to ensure a consistent and predictable order of method resolution.

```
class A:
    def show(self):
        print("A")

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

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

class D(B, C):
    def show(self):
        print("D")
        super().show()

d = D()
d.show()
```

2. Handling Method Conflicts with super() in Multiple Inheritance

When two parent classes (like Human and Mammal) have a method with the same name but different implementations, Python resolves this using MRO. The child class will invoke the first method found in the MRO chain.

```
class Human:
    def eat(self):
        print("Human is eating")

class Mammal:
    def eat(self):
        print("Mammal is eating")

class Employee(Human, Mammal):
    def eat(self):
        print("Employee starts eating:")
        super().eat()

e = Employee()
e.eat()
```

3. Example Based on Samy Story (Project Lab)

In the ITI story, Samy is an Employee who inherits from Person and owns a Car. Both Person and Car can have methods that affect Samy's actions. Below is how multiple inheritance can apply using super() to coordinate actions.

```
class Person:
    def move(self):
        print("Person is walking.")

class Car:
    def move(self):
        print("Car is driving.")

class Employee(Person, Car):
    def move(self):
        print("Employee starts moving...")
        super().move()

samy = Employee()
samy.move()
```

Output will show that Person.move() is called due to MRO (Person before Car). If we switch inheritance order (Car, Person), then Car.move() will be called instead.