Multilevel Inheritance
You can inherit a derived class from another derived class.
OR
Extending features from one derived class to another.

Outline of Multilevel Inheritance:
BaseClass #Existing Class
Feature1

DerievedClass1 #NewlyCreatedClass
Features+BaseClass
Feature2

DerievedClass2 #NewlyCreatedClass
Features+BaseClass
Features+DereivedClass1
Feature3

Syntax:

```
class BaseClass:
  Body of base class
class DerivedClass1(BaseClass):
  Body of derived class1
class DerivedClass2(DereivedClass1):
  Body of derived class2
```

Example:

```
class Student():#Super Class
    def getStudent(self):
        self.name=input("Name: ")
        self.age=int(input("Age: "))
        self.gender=input("Gender: ")

class Subjects(Student):#Derived Class-1
    def getMarks(self):
        self.StudentClass=input("Class: ")
        print("Enter the marks of the respective subjects:")
        self.Programming=int(input("Programming: "))
        self.math=int(input("Math: "))
        self.english=int(input("English: "))
        self.physics=int(input("Physics: "))

class Marks(Subjects):#Derived Class-2
    def Display(self):
        print("\n Name is: ",self.name)
        print("Age is: ",self.age)
        print("Gender is: ",self.gender)
        print("Class Name: ",self.StudentClass)
        print("Total Marks:
",self.Programming+self.math+self.english+self.physics)

MM=Marks()
MM.getStudent()
MM.getMarks()
MM.Display()
```

```
Example:
class Finacial_Services_India():
    def FSI(self):
        print("All Permissions from here")
        print("It is related Finance")
class ReserveBankofIndia(Finacial_Services_India):
    def RBI(self):
        print("It is leading All Bankers in INDIA")
        print("Thank u RBI")
class Bankers_India(ReserveBankofIndia):
    def Bank_INDIA(self):
        print("Welcome to Indian Banks")
        print("We are ready to serve")
        print("We deal Finanacial Matters")
BI=Bankers_India()
BI.FSI()
BI.RBI()
BI.Bank_INDIA()

Example:
class GrandFather():
    def Hello_GFather(self):
        print("I am in GrandFather Class")
class Father(GrandFather):
    def Hello_Father(self):
        print("I am in Father Class")
class Child(Father):
    def Hello_Child(self):
        print("I am in Child Class")

ch=Child()
ch.Hello_GFather()
ch.Hello_Father()
ch.Hello_Child()

Example:
class person():
    def display(self):
        print("Hello, This is class Person")
class employee(person):
    def printing(self):
        print("Hello This is derieved class Employee")
class programmer(employee):
    def show(self):
        print("Hello This is derieved class Programmer")

p1=programmer()
p1.display()
p1.printing()
p1.show()

Example:
class Animal:
    def eat(self):
      print('Eating...')
class Dog(Animal):
```

```
   def bark(self):
      print('Barking...')
class BabyDog(Dog):
    def weep(self):
        print('Weeping...')
d=BabyDog()
d.eat()
d.bark()
d.weep()
```

LIVE USE CASES:
Bank ==> CreditCard ==> Customer
College ==> Course ==> Student
Vehi ==> Car ==> SportsCar ==> Speed

Multiple Inheritance:-
You can derive a child class from more than one base (Parent) class.

Outline of Multiple Inheritance:

```
BaseClass1              BaseClass2

            DerievedClass
```

Syntax1:

```
class DerivedClassName(Base1, Base2...BaseN):
    <statement-1>
   ------------------------
    <statement-N>
```

Example:

```
class First():
    def A(self):
        print("I am in First PClass")
class Second():
    def B(self):
        print("I am in Second PClass")
class Third(First,Second):
    def C(self):
        print("I am in Child Class")

TT=Third()
TT.A()
TT.B()
TT.C()
```

Constructor or __init__ In Inheritance
Example:

```
class First():
    def __init__(self):
        print("init of First Class")
    def A(self):
        print("I am in First PClass")
class Second(First):
    def B(self):
        print("I am in Second PClass")
class Third(Second):
    def C(self):
```

```
        print("I am in Child Class")

FF=First()
```

Example:

```
class First():
    def __init__(self):
        print("init of First Class")
    def A(self):
        print("I am in First PClass")
class Second(First):
    def B(self):
        print("I am in Second PClass")
class Third(Second):
    def C(self):
        print("I am in Child Class")

SS=Second()
```

Example:

```
class First():
    def __init__(self):
        print("init of First Class")
    def A(self):
        print("I am in First PClass")
class Second(First):
    def B(self):
        print("I am in Second PClass")
class Third(Second):
    def C(self):
        print("I am in Child Class")

TT=Third()
```

Example:The priority for its own class

```
class First():
    def __init__(self):
        print("init of First Class")
    def A(self):
        print("I am in First PClass")
class Second(First):
    def __init__(self):
        print("init of Second Class")
    def B(self):
        print("I am in Second PClass")
class Third(Second):
    def C(self):
        print("I am in Child Class")

SS=Second()
```

Python super() function:

It always refer the immediate superclass. It allows you to call methods of the superclass in your subclass. It can refer the superclass implicitly. It eliminates the need to declare certain characteristics of a class.

In Python, super() built-in has two major use cases:
1 Allows us to avoid using base class explicitly
2 Working with Multiple Inheritance

Syntax:PY-3.x
super().methoName(args)

Example: If you want to execute both __init__ classes,super() function required

```
class First():
    def __init__(self):
        print("init of First Class")
    def A(self):
        print("I am in First PClass")
class Second(First):
    def __init__(self):
        super().__init__()
        print("init of Second Class")
    def B(self):
        print("I am in Second PClass")
class Third(Second):
    def C(self):
        print("I am in Child Class")

SS=Second()
```

Method Resolution Order (MRO)
In the multiple inheritance scenario, any specified attribute is searched first in the current class. If not found, the search continues into parent classes in depth-first, left-right fashion without searching same class twice. MRO must prevent local precedence ordering and also provide monotonicity.

Example:

```
class First():
    def __init__(self):
        print("init of First Class")
    def A(self):
        print("I am in First PClass")
class Second():
    def __init__(self):
        print("init of Second Class")
    def B(self):
        print("I am in Second PClass")
class Third(First,Second):
    def __init__(self):
        super().__init__()
        print("init of Third Class")
    def C(self):
        print("I am in Child Class")

TT=Third()
print(Third.mro())
```

Method Resolution Order (MRO) is the order in which Python looks for a method in a hierarchy of classes. Especially it plays vital role in the context of multiple inheritance as single method may be found in

multiple super classes.