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.

CASE:1
This is a simple case where we have class C derived from both A and B. When method process() is called with object of class C then process() method in class A is called.

Example:

```
class A:
    def process(self):
        print('A process()')

class B:
    pass

class C(A, B):
    pass

obj = C()
obj.process()
print(C.mro())
```

Example2:Case-2

```
class A:
    def process(self):
        print('A process()')


class B:
    def process(self):
        print('B process()')

class C(A, B):
    pass

obj = C()
obj.process()
```

Case 3
In this case, we create D from C and B. Classes C and B have process() method and as expected MRO chooses method from C. Remember it goes from left to right. So it searches C first and all its super classes of C and then B and all its super classes.

Example:

```
class A:
    def process(self):
        print('A process()')
class B:
    def process(self):
        print('B process()')
class C(A, B):
    def process(self):
        print('C process()')
class D(C,B):
    pass
```

```
obj = D()
obj.process()
print(D.mro())
```

Case 4
Lets change the hierarchy. We create B and C from A and then D from B and C. Method process() is present in both A and C.

Example:
```
class A:
    def process(self):
        print('A process()')

class B(A):
    pass

class C(A):
    def process(self):
        print('C process()')

class D(B,C):
    pass

obj = D()
obj.process()
```

Case 5
There are cases when Python cannot construct MRO owing to complexity of hierarchy.

Example:
```
class A:
    def process(self):
        print('A process()')

class B(A):
    def process(self):
        print('B process()')

class C(A, B):
    pass

obj = C()
obj.process()
```

NOTE: TypeError: Cannot create a consistent method resolution order (MRO) for bases A, B

Example:
```
class Employee():
    def Set1(self,empid,name,salary):
        self.empid=empid
        self.name=name
        self.salary=salary

class Fitness():
    def Set2(self,height,weight, bloodGrp):
```

```
        self.height=height
        self.weight=weight
        self.bloodGrp=bloodGrp

class Company(Employee,Fitness):
    def Set3(self,Company,dep):
        self.Company=Company
        self.dep=dep

    def Display(self):
        print("ID is: ",self.empid)
        print("Emp Name is: ",self.name)
        print("Salary is: ",self.salary)
        print("Height is: ",self.height)
        print("Weight is: ",self.weight)
        print("Blood Group is: ",self.bloodGrp)
        print()
        print("Company is: ",self.Company)
        print("Department is: ",self.dep)

CC=Company()
print(Company.mro())
CC.Set1(101,"Raju",70000)
CC.Set2(145,50,"B-")
CC.Set3("IBM","Software")
CC.Display()
```

Hierarchical Inheritance:-
The concept of inheriting properties from one class into multiple classes is known as a hierarchical inheritance

Syntax:

```
class A:
  pass
class B(A):
  pass
class C(A):
  pass
```

Example:

```
class Details():
    def __init__(self):
        self.Id=""
        self.Name=""
    def SetData(self,Id,Name):
        self.Id=Id
        self.Name=Name
    def ShowData(self):
        print("ID: ",self.Id)
        print("Name: ",self.Name)

class Employee(Details):
    def __init__(self):
        self.Company=""
        self.Dept=""
    def SetEmployee(self,Id,Name,Comp,Dept):
        self.SetData(Id,Name)
```

```
        self.Company=Comp
        self.Dept=Dept
    def ShowEmployee(self):
        self.ShowData()
        print("Company: ",self.Company)
        print("Department: ",self.Dept)

class Doctor(Details):
    def __init__(self):
        self.Hospital=""
        self.Dept=""
    def SetEmployee(self,Id,Name,Hos,Dept):
        self.SetData(Id,Name)
        self.Hospital=Hos
        self.Dept=Dept
    def ShowEmployee(self):
        self.ShowData()
        print("Hospital: ",self.Hospital)
        print("Department: ",self.Dept)

print("Employee Class Object")
EE=Employee()
EE.SetEmployee(101,"Raju","TCS","IT")
EE.ShowEmployee()

print("\n Doctor Class Object: ")
DD=Doctor()
DD.SetEmployee(101,'RaviKumar',"Prime","Dental")
DD.ShowEmployee()

Example:
class Banker():
    def Bank_Services(self):
        print("Welcome to Banking Services")
class EBanking(Banker):
    def Email(self):
        print("Welcome to Email Banking")
class Banking_Mobile(Banker):
    def Mobile(self):
        print("Welcome to Mobile Banking")
class Banking_Phone(Banker):
    def Phone(self):
        print("Welcome to Phone Banking")
BP=Banking_Phone()
BP.Bank_Services()
BP.Phone()
BM=Banking_Mobile()
BM.Bank_Services()
BM.Mobile()
BE=EBanking()
BE.Bank_Services()
BE.Email()

Example:
class Hello():
  def MyOne(self):
      print("I am in Hello Class")
```

```
class Hei(Hello):
  def MyTwo(self):
    print("I am in Hei Class")
class Bye(Hello):
  def MyThree(self):
    print("I a, in Bye Class")

HH=Hei()
HH.MyOne()
HH.MyTwo()
BB=Bye()
BB.MyOne()
BB.MyThree()
```

Example:

```
class Animal():
  def __init__(self,name):
    self.name=name
  def talk(self):
    pass

class Cat(Animal):
  def talk(self):
    print("Cat is MewMewMewMewMewMew..!!")

class Dog(Animal):
  def talk(self):
    print("Dog is Barkingngngngngngngng..!!")

a=Animal('Sound')
c=Cat("KITTY")
c.talk()

d=Dog("TOMMY")
d.talk()
```

LIVE USE CASE
Library ==> Students ==> Faculty ==> Management ==>SrCitizen

Hybrid Inheritance
It is combining two or more types of inheritance(s).
OR
It is the combination of Hirarchical+Multiple+Multilevel
Inheritance(s) are called Hybrid Inheritance.

PIC: HybridInheritance

Syntax:

```
class A:
  pass
class B(A):
  pass
class C(A):
  pass
class D(B, C):
  pass
```

```
def main():
  obj1=D()

if __name__=="__main__":
  main()
```

Example:

```
class A:
  def m(self):
    print("method from Class A....")

class B(A):
  def m(self):
    print("method from Class B....")

class C(A):
  def m(self):
    print("method from Class C....")

class D(B, C):
  def m(self):
    print("method from Class D....")
    B.m(self)
    C.m(self)
    A.m(self)

def main():
  obj1=D()
  obj1.m()

if __name__=="__main__":
  main()
```

Example: (Diamond Death Problem)

```
class A:
  def m(self):
    print("method from Class A....")

class B(A):
  def m(self):
    print("method from Class B....")
    A.m(self)

class C(A):
  def m(self):
    print("method from Class C....")
    A.m(self)

class D(B, C):
  def m(self):
    print("method from Class D....")
    B.m(self)
    C.m(self)
    A.m(self)

def main():
  obj1=D()
```

```
  obj1.m()

if __name__=="__main__":
  main()
```

Example: (with super())

```
class A:
  def m(self):
    print("method from Class A....")

class B(A):
  def m(self):
    print("method from Class B....")
    super().m()

class C(A):
  def m(self):
    print("method from Class C....")
    super().m()

class D(B, C):
  def m(self):
    print("method from Class D....")
    super().m()

def main():
  obj1=D()
  obj1.m()

if __name__=="__main__":
  main()
```

```
 if __name__ =='__main__':
```

__name__ (A Special variable) in Python

__name__ is one such special variable. If the source file is executed as the main program, the interpreter sets the __name__ variable to have a value “__main__”. If this file is being imported from another module, __name__ will be set to the module’s name.

__name__ is a built-in variable which evaluates to the name of the current module. Thus it can be used to check whether the current script is being run on its own or being imported somewhere else by combining it with if statement.

```
>>>__name__
'__main__'
```

Example:

```
def Mul(a,b):
  return a*b
print(Mul(2,3))
print(__name__)
```

Example:

```
import multiply
```

```
print(multiply.Mul(2.3,1.1))
print(__name__)

Example:
def Mul(a,b):
  return a*b
if __name__=="__main__":
  print(Mul(2,3))
  print(__name__)
```

Consider two separate files File1 and File2.

```
# File1.py
print("File1 __name__ = %s" %__name__)
if __name__ == "__main__":
    print("File1 is being run directly")
else:
    print("File1 is being imported")

# File2.py
import File1
print("File2 __name__ = %s" %__name__)
if __name__ == "__main__":
    print("File2 is being run directly")
else:
    print("File2 is being imported")
```

Above, when File1.py is run directly, the interpreter sets the __name__ variable as __main__ and when it is run through File2.py by importing, the __name__ variable is set as the name of the python script, i.e. File1. Thus, it can be said that if __name__ == "__main__" is the part of the program that runs when the script is run from the command line using a command like python File1.py.