Hybrid Inheritance

In [ ]:
```
Hybrid Inheritance: Hybrid inheritance is a combination of Single,Multilevel,multiple and hierarchial inheritance.
```

Implementation of Hybrid Inheritance

In [3]:
```
class A:
    def m1(self):
        print("Class A m1 Method")

class B:
    def m2(self):
        print("Class B m2 Method")

class C:
    def m3(self):
        print("Class C m3 Method")


class D(A,B,C):
    def m4(self):
        print("Class D m4 Method")

class E(D):
    def m5(self):
        print("Class E m5 Method")

class F(E):
    def m6(self):
        print("Class F m6 Method")

class G(F):
    def m7(self):
        print("Class G m7 Method")

class H(F):
    def m8(self):
        print("Class H m8 Method")

x=H()
x.m1()
x.m2()
x.m3()
```

```
Class A m1 Method
Class B m2 Method
Class C m3 Method
```

Cyclic Inheritance

In [ ]:
```
Cyclic Inheritance --> The concept of inheriting the properties and behaviour from one class class to itself into a cyclic
                       way. such type of inheritance are called cyclic inheritance.

Python does not support cyclic inheritance
```

Polymorphism

In [ ]:
```
--> Poly means many
--> Morphism means forms
--> polymorphism => many forms
--> Entity will be the same but the behaviour of that enitity is different in different conditions.
--> Polymorphism is the concept which means having many forms. In simple words, we can define polymorphism as the ability of
    a message to be displayed in more than one form.
--> A real-life example of polymorphism is a person who at the same time can have different characteristics

Example:

    * operator acts as concatenation and addtion
    * operator acts as repeatition and multiplication

Here the operator is same but the behaviour of both the operators are different in different
scenarios.

Example:

    "STring1"+"String2" == "STring1String2"
    10+20==30
```

Types of Polymorphism:

In [ ]:
```
Two Types of Polymorphism:

    1.Compile Time POlymorphism(Overloading)
    2.Runtime Polymorphism(Overriding)
```

Types of Compile Time Polymorphism(Overloading)

In [ ]:
```
Three Types of Overloading:
    1.Operator Overloading
    2.Method Overloading
    3.Constructor Overloading
```

Operator Overloading

In [ ]:
```
Operator Overloading --> we can use same operator for multiple purposes which is nothing  but operator overloading.
                         --> Python Supports Operator Overloading.

Example:
    * operator is acting as addition as well as concatenation operator.
    * operator is acting as multiplication as well as repetition operator

here the operator is same but the behaviour of both the operators got varied with respec to the operand.

"String1"+"String2"=String1String2
10+20=30
```

Special/Dunder/Magic Method

In [ ]:
```
--> There are many internal methods that are already present in our python interpretaor
--> Whenever we are required we can overload these methods for our use. These methods always
    prefix with double underscore and suffix with double underscore.

Example:
    __add__  --> for adding object
    __sub__  --> for subtracting two objects
    __div__  --> for dividing two objects
    __mul__  --> for multiplying two objects
```

Example of Operator Overloading and Special method

In [27]:
```
#Demo program to use + operator for a class object
class Book:
    def __init__(self,pages):
        self.pages=pages
        print("Constructor loaded")

    def __add__(self,x):
        return self.pages+x.pages
B1=Book(100)
B2=Book(1000)
print(B1)
print(B2)
print(B1+B2)


Note --> We can overload + operator to work with book object as well for doing this you need to overload
         + operator.
     --> Internally + operator is implemented vy using a special method named as __add__().
         this method is known as Magic method/Special Method/Dunder Method
```

```
Constructor loaded
Constructor loaded
<__main__.Book object at 0x000001AC1A048730>
<__main__.Book object at 0x000001AC1A048550>
1100
```

In [4]:
```
#Demo program to use - operator for a class object
class Book:
    def __init__(self,pages):
        self.pages=pages
        print("Constructor called")
    def __sub__(self,x):
        print("Magic method called")
        return self.pages-x.pages

b1=Book(200)
b2=Book(200)
print(b1-b2)
```

```
Constructor called
Constructor called
Magic method called
1800
```

In [7]:
```
#Demo program to use * operator for a class object
class Book:
    def __init__(self,pages):
        self.pages=pages
        print("Constructor loaded")

    def __mul__(self,x):
        return self.pages*x.pages
B1=Book(500)
B2=Book(1000)

print(B1)
print(B2)
print(B1*B2)
```

```
Constructor loaded
Constructor loaded
<__main__.Book object at 0x00000256EC06B490>
<__main__.Book object at 0x00000256EBF99F10>
500000
```

In [8]:
```
class Book:
    def __init__(self,pages):
        self.pages=pages
        print("Constructor loaded")

    def __floordiv__(self,x):
        return self.pages//x.pages
B1=Book(500)
B2=Book(1000)

print(B1)
print(B2)
print(B1//B2)
```

```
Constructor loaded
Constructor loaded
<__main__.Book object at 0x00000256EC195190>
<__main__.Book object at 0x00000256EBF84A60>
0
```

Method Overloading

In [ ]:
```
Method Overloading --> if multiple methods are having same name but different arguments then such type of overloading
                       is known as method overloading.
                   --> In python method overloading is not possible.
                   --> if you are providing multiple methods with same name with different arguments then
                       python will automatically consider the last one .


Example:
    fun(int a, int b)
    fun(double x)
    fun(float a)
```

Demostration of Method Overloading

In [8]:
```
class Test:
    def m1(self):
        print("No Argument")
    def m1(self,name):
        print("One Argument")
    def m1(self,name,classes):
        print("Two Argument")
    def m1(self,name,classes,roll):
        print("Three Argument")

t=Test()
t.m1()  #error
t.m1(10)  #error
t.m1(10,20)  #error
t.m1(10,20,30)
```

```
Three Argument
```

Demo program to overcome the issue of Method Overloading(Variable length argument)

In [10]:
```
#Demo program to overcome the issue of Method Overloading(Variable length argument)
class Test:
    def m1(self,*a):
        print("Method executed")
t=Test()
t.m1()
t.m1(10)
t.m1(10,20)
```

```
Method executed
Method executed
Method executed
```

Constructor overloading

In [ ]:
```
Constructor overloading --> If we are having multiple constructors with different arguments
                            then that type of overloading is known as Constructor overloading.
                        --> constructor overloading is not possible.
                        --> if you are giving multiple constructor with same name and different argument then in Python
                            only the last one will considered.
```

Demostration of Constructor Overloading

In [11]:
```
class test:
    def __init__(self):
        print("No - Arg Constructor")
    def __init__(self,name):
        print("One Arg Constructor")
    def __init__(self,name,roll):
        print("Two Arg Constructor")
    def __init__(self,name,roll,classes):
        print("Three Arg Constructor")

t=test(10,20,30)
```

```
Three Arg Constructor
```

Demo program to overcome the issue of Constructor Overloading(Variable length argument)

In [13]:
```
class test:
    def __init__(*A):
        print("No - Arg Constructor")

t=test()
t1=test(10,20)
t2=test(10,20,30,40,50,60,70)
```

```
No - Arg Constructor
No - Arg Constructor
No - Arg Constructor
```

Overriding

In [ ]:
```
Overriding --> whatever  memebers are present in the parent class are by default available to the child class
               This concept is known as Inheritance.If the child class is not satisfied with the parent class functionality
               then child class is allowed to implement or redefine its own functionality based on certain requirement ,
               this concept is knowns as overriding.

Type of Overriding:
    1.Method overriding
    2.Constructor Overriding
```

Method Overriding

In [ ]:
```
Method Overriding  : if two methods are having same name and same number of argument then that
                     concept is known as method overriding

Example :
    fun(int a, int b)  --->> Parent class
    fun(int a, int b)  --->> class class
```

Demostration of Method Overriding

In [594]:
```
class Parent:
    def study(self):
        print("B_Tech is good")
    def property(self):
        print("Flat")
class Child(Parent):
    def study(self):
        print("BSC IS GOOD")
x=Child()
x.study()
```

```
B_Tech is good
```

Super() Method

In [ ]:
```
--> For overriding the method of child class we can also call parent class method with the help
    of super() method
--> It is a method that is used to access the parent class method , variables and constructor
```

Demostration of Method Overriding and Super Method

In [598]:
```
class Parent:
    def study(self):
        print("B_Tech is good")
    def property(self):
        print("Flat+Car")
class Child(Parent):
    def property(self):
        print("Iphone+Gold")

class Grand(Child):
    def property(self):
        super().property()
        print("Laptop")
x=Grand()
x.property()
```

```
Iphone+Gold
Laptop
```