Inheritance

In [1]:

```
# A - Parent class
class A():
    def __init__(self):
        self.name = 'A'
        print ("I'm in", self.name)
a = A()
```

```
I'm in A
```

In [2]:

```
# B - Child class
class B(A):
    def __init__(self):
        self.name = "B"
        print ("I'm in", self.name)
        super().__init__()

b = B()
```

```
I'm in B
I'm in A
```

Changing the order

In [4]:

```
# B - Child class
# super().__init__() should be first
class B(A):
    def __init__(self):
        super().__init__()
        self.name = "B"
        print ("I'm in", self.name)

b = B()
```

```
I'm in A
I'm in B
```

Multiple Inheritance

In [3]:

```
# A - Parent class
# B - Parent class
class A():
    def __init__(self):
        print ("I'm in A")

class B():
    def __init__(self):
        print ("I'm in B")
```

In [4]:

```
# C - Child class (A,B)
class C(A,B):
    def __init__(self):
```

```
        print ("I'm in C")
        super().__init__()
c = C()
```

```
I'm in C
I'm in A
```

Order of inheritance

In [16]:

```
# C - Child class (B,A)
class C(B, A):
    def __init__(self):
        print ("I'm in C")
        super().__init__()
c = C()
```

```
I'm in C
I'm in B
```

Add another constructor

In [17]:

```
# C - Child class (A,B) + another constructor
class C(A,B):
    def __init__(self):
        print ("I'm in C")
        super().__init__()
        super().__init__()
c = C()
```

```
I'm in C
I'm in A
I'm in A
```

Initialize all base classes

In [19]:

```
# C - Child Class (A,B)
class C(A,B):
    def __init__(self):
        print ("I'm in C")
        A.__init__(self)
        B.__init__(self)

c = C()
```

```
I'm in C
I'm in A
I'm in B
```

Order of inheritance

In [20]:

```
# C - Child Class (A,B)
class C(A,B):
    def __init__(self):
        print ("I'm in C")
        B.__init__(self)
        A.__init__(self)

c = C()
```

```
I'm in C
I'm in B
I'm in A
```

Initilize all base classes with super(ParentClass, self).init()

super(second, self).init()

In [19]:

```
# A - Parent class
class A():
    def __init__(self):
        print ("I'm in A")
# B - Parent class
class B():
    def __init__(self):
        print ("I'm in B")
# C - Child Class (A,B)
class C(A,B):
    def __init__(self):
        print ("I'm in C")
        super().__init__()

c = C()
```

```
I'm in C
I'm in A
```

In [1]:

```
class A():
    def __init__(self):
        print ("I'm in A")
class D(A):
    def __init__(self):
        print ("I'm in D")
class B():
    def __init__(self):
        print ("I'm in B")
class C(B,D):
    def __init__(self):
        print ("I'm in C")
        super().__init__()
c = C()
```

```
I'm in C
I'm in B
```

Initialize 3 base classes

In [55]:

```
# Class First - Second - Third
class First():
    def __init__(self):
        print ("First")

class Second():
    def __init__(self):
        print ("Second")

class Third():
    def __init__(self):
        print ("Third")
```

In [58]:

```
# Class Child (First, Second, Third)
class Child(First, Second, Third):
    def __init__(self):
        print ('Child') # Child
        super().__init__() # First
        super(First, self).__init__() # Second
        super(Second, self).__init__()# Third
        super(Third, self).__init__()# Third

c = Child()
```

```
Child
```

if Without want to call the base Constructor

In [59]:

```
# Class First - Second - Third WITHOUT constructor
class First():
    def __init__(self):
        print ("First")

class Second():
    def __init__(self):
        print ("Second")

class Third():
    def __init__(self):
        print ("Third")
```

In [61]:

```
# Class Child (First, Second, Third)
class Child(First, Second, Third):
    def __init__(self):
        print ("Child")      # Child
        super(Second, self).__init__()   # First
c = Child()
```

```
Child
Third
```

Browse to all Parent Classes

In [62]:

```
# Class Child (First, Second, Third)
# Output : --> Child Second First
class Child(First, Second, Third):
    def __init__(self):
        print ("Child")
        super(First, self).__init__()
        super().__init__()
c = Child()
```

```
Child
Second
First
```

What if I don't want to use super(FirstParentClass, self).init()

In [68]:

```
class First():
```

```
    def __init__(self):
        print ("First")
        super().__init__() # Class Object
```

In [4]:

```
# Class First - Second - Third
class First():
    def __init__(self):
        print ("First")
        super().__init__()

class Second():
    def __init__(self):
        print ("Second")
        super().__init__()

class Third():
    def __init__(self):
        print ("Third")
        super().__init__()

class Child(First, Third):
    def __init__(self):
        print ("Child")
        super().__init__()

c = Child()
```

```
Child
First
Third
Second
```

In [ ]:

```
# Class Child (First, Second, Third)
```

How to handle Parameters

In [82]:

```
# Class First - Second - Third WITH constructor and PARAMETERS (attributes)
class First():
    def __init__(self, firstname):
        self.firstname = firstname
        print (self.firstname)
        super().__init__(firstname)

class Second():
    def __init__(self, secondname):
        self.secondname = secondname
        print (self.secondname)
        super().__init__()

class Third():
    def __init__(self, third):
        print ("Third")
        super().__init__()
        self.third = third
```

With 2 base classes

In [83]:

```
# Class Child(First, Second)
class Child (First, Third):
    def __init__(self, firstname, secondname):
```

```
        print ("Child")
        super().__init__(firstname)
c = Child("FirstName", "SecondName")
```

```
Child
FirstName
FirstName
```

keyword arguments -- > **kwargs

In [85]:

```
def Course(required, *args, **kwargs):
    print ('required:', required)
    if args: # Tuple
        print ("args:", args)
    if kwargs: # Dictionnary
        print ("kwargs:", kwargs)
```

In [87]:

```
Course("Learning by doing")
```

```
required: Learning by doing
```

In [88]:

```
Course("Learning by doing", "01-19-2022", "OOP")
```

```
required: Learning by doing
args: ('01-19-2022', 'OOP')
```

In [89]:

```
Course("Learning by doing", date="01-19-2022", chapter="OOP")
```

```
required: Learning by doing
kwargs: {'date': '01-19-2022', 'chapter': 'OOP'}
```

In [113]:

```
Course("Learning by doing", "UM6P-CS", date="01-19-2022", chapter="OOP")
```

```
required: Learning by doing
args: ('UM6P-CS',)
kwargs: {'date': '01-19-2022', 'chapter': 'OOP'}
```

in OOP

In [17]:

```
# Class First - Second - Third WITH constructor and PARAMETERS (attributes)
class First():
    def __init__(self, firstname, **kwargs):
        self.firstname = firstname
        print (self.firstname,kwargs)
        super().__init__(**kwargs)

class Second():
    def __init__(self, secondname, **kwargs):
        self.secondname = secondname
        print (self.secondname,kwargs)
        super().__init__(**kwargs)

class Third():
    def __init__(self, middlename, **kwargs):
        self.middlename = middlename
        print (self.middlename,kwargs)
```

```
# Class Child(First, Second)
class Child (First, Second, Third):
    def __init__(self, firstname, secondname, middlename):
        super().__init__(firstname = firstname, secondname = secondname, middlename = mi
ddlename)
c = Child("UM6P", "CS", "Computer-Science")
```

```
UM6P {'secondname': 'CS', 'middlename': 'Computer-Science'}
CS {'middlename': 'Computer-Science'}
Computer-Science {}
```

In [119]:

```
c.firstname
```

Out[119]:

```
'UM6P'
```

In [120]:

```
c.secondname
```

Out[120]:

```
'CS'
```