**Constructor in inheritance:**

Let’s take a class and print a1 object it will call the default method which is init Check the output

class A:

    def \_\_init\_\_(self):

        print('in A init')

    def featureA1(self):

        print('feature A1 is working')

    def featureA2(self):

        print('feature A2 is working')

class B(A):

def featureB1(self):

        print('feature B1 is working')

    def featureB2(self):

        print('feature B2 is working')

a1 = A()

if we try a1. It will access/show features A1 and A2 which are features of the same class.

Remember a subclass can access all the features of super/parent/outer class but NOT vice versa

* Now if we create a object a1 with the help of class B, what do you think will happen?
* Can it access the init method in class A? - NO

Since we don’t have init inside Class B it will call the init in Class A

Now the question is what if we create a init method inside class B

Graphical user interface, text, application

Description automatically generated

It will call the init only from the same class and ignore the init from the super class.

* By default it will call only the init from it’s own class , what if we want to call the init from the outer class, that’s where we have a special keyword called ***super***  and below is shown how to mention it.
* Run the below code and check the output
* Graphical user interface, text, application

  Description automatically generated
* Now let’s separate class A, class B and create new class C(A,B) and mention init for print(‘init c’) and if we print it, we will get the output of **in C init** as expected.

Graphical user interface, text, application

Description automatically generated

What if you want to call the init method of a super class which is A/B and what do you think it will call A or B

Diagram

Description automatically generated

Once you run the below code it will print in **init A and init C**.

We are biased here which is unfair as we are ignoring B here.

Graphical user interface, text, application

Description automatically generated

The reason for ignoring B is we have a concept of MRO (method resolution order)—whenever you have multiple inheritance it will always start from **left to right.**

First it will execute the init of c as we had mentioned the init(self) for C , then we had mentioned super.init it will prefer the left one first which is A.

Diagram

Description automatically generated with medium confidence

The same thing can be done for methods as well. Example we have two methods which are same. Feature 1 in A and feature 1 in B

Graphical user interface, text, application

Description automatically generated

Now if we try to call a1.feature1() what do you think it will call, from A or B.

Since it is from left to right, it will call from 1 – A as shown below, run the code and check the output.

Graphical user interface, application, Word

Description automatically generated

Now with the help of super method can we call function? Yes, we can do it. Check the below code, run the code and observe the output.

Graphical user interface, text, application

Description automatically generated

**Polymorphism**

One thing can take multiple forms. Ex: we change as per the situations, when we are in office, school, ground, with friends etc..

Timeline

Description automatically generated with low confidence

There are 4 ways of implementing polymorphism

A person speaking into a microphone

Description automatically generated

<https://www.programiz.com/python-programming/examples/number-divisible>

<https://www.programiz.com/python-programming/examples/power-anonymous>

decimal – 0 1 2 3 4 5 6 7 8 9 (total 10)

binary – 1 and 0

hexadecimal – 16

octal - 8

60 – decimal num

Convert it to binary

2 power 0 2 power 1 2 power 2

1 2 4 8 16 32 64 128

0 0 1 1 1 1 0 0

+ 🡪 convert 🡪 1 10101001010

1 0

000001111100011100101010101001