```
1
2 2
3 3 3
4 4 4 4
5 5 5 5 5
```

```
n = 5
for i in range(1, n+1):
    for j in range(1, n+1):          or (1, i+1)
        if (i >= j):
            print(i, end=' ')
    print()
```

~~for~~

```
1
1 0
1 0 1
1 0 1 0 1
```

```
n = 5
for i in range(1, n+1):
    for j in range(1, n+1):
        if (i >= j):
            if (j % 2 == 0):
                print(0, end=' ')
            else:
                print(1, end=' ')
    print()
```

NOTE:

ORD - Ord is a function which is used to get the ASCII value of a character.

```
ord('A')
65
```

CHR - CHR is a function which is used to convert a ASCII value to a character

```
chr(65)
'A'
```

Print

```
A
B B
C C C
D D D D
E E E E E
```

```
n=5 k=ord('A')
for i in range(1, n+1):
    for j in range(1, n+1):
        if i>=j:
            print(chr(k), end=' ')
        else:
            print(' ', end=' ')
    print()
    k+=1
```

```
A
B C
D E F
G H I J
K L M N O
```

```
n = 5
k = ord('A')
for i in range(1, n+1):
    for j in range(1, n+1):
        if i >= j:
            print(chr(k), end=' ')
            k += 1
        else:
            print(' ', end=' ')
    print()
```

```
A
A B
A B C
A B C D
A B C D E
```

```
n = 5
for i in range(1, n+1):
    k = ord('A')
    for j in range(1, n+1):
        if i >= j:
            print(chr(k), end=' ')
            k += 1
        else:
            print(' ', end=' ')
    Print()
```

```
1
A A
1 2 3
B B B B
1 2 3 4 5
n=5    k=ord('A')
for i in range(1,n+1):
    for j in range(1,n+1):
        if i>=j:
            if i%2==1:
                print(j, end=' ')
            else
                print(chr(k), end=' ')
                k+=1
        else print(' ', end=' ')
    print()
    k+=1
```

```
1
A B
1 2 3
A B C D
1 2 3 4 5
n=5    k=ord('A')
for i in range(1,n+1)
    for j in range(1,n+1)
        if i>=j
            if i%2==1
                print(j, end=' ')
            else
                print(chr(k), end=' ')
                k+=1
        else print(' ', end=' ')
    print()
```

```
A
A 2
A 2 A
A 2 A 4
A 2 A 4 A
```

```
n=5
for i in range(1,n+1):
    k=ord('A')
    for j in range(1,n+1):
        if i>=j:
            if j%2==0
                print(j,end=' ')
            else
                print('A',end=' ')
        else
            print(' ',end=' ')
    print()
```

```
1
1 A
1 A 3
1 A 3 B
1 A 3 B 5
```

```
n=5
for i in range(1,n+1):
    k=ord('A')
    for j in range(1,n+1):
        if i>=j:
            if j%2==1
                print(j)
```

```
    1
  2 1
3 2 1
4 3 2 1
5 4 3 2 1
```

```
n = 5
for i in range (1, n+1):
    for k in range (1, n-i):
        print (' ', end = ' ')
    for j in range (1, n+1):
        if (i <= j):
            print (n-j), end = ' ')
        else
            print (' ', end = ' ')
```

```
n = 5
for i in range (1, n+1):
    for j in range (1, n+1):
        if i+j > n+1:
            print (n+1-j, end = ' ')
        else:
            print (' ', end = ' ')
    print()
```

```
n = 5   l = 1
for i in range (1, n+1):
    for j in range (1, n+1):
        if i+j > n+1:
            print (l, end = ' ')
            l += 1
        else:
            print (' ', end = ' ')
    print()
```

```
        1
      1 0 1
    0 1 0 1
  1 0 1 0 1
1 0 1 0 1
```

```
n=5
for i in range(1,n+1):
    for j in range(1,n+1):
        if i+j>=n+1:
            if (j%2==0):
                print(0,end=' ')
            else
                print(1,end=' ')
        else
            print(' ',end=' ')
    print()
```

---

```
n=5   k=ord('A')
for i in range(1,n+1):
    for j in range(1,n+1):
        if i+j>=n+1:
            print(chr(k),end=' ')
        else
            print(' ',end=' ')
    print()
```

```
        A
      A A
    A A A
  A A A A
```

```
        A
      B A
    C B A
  D C B A
E D C B A
```

```
n=5   k=ord('A')
for i in range(1,n+1):
    for j in range(1,n+1):
        if i+j>=n+1
            print(chr(k+n-j),end=' ')
```

```
        else:
            print(' ', end=' ')
    print()
```

```
1 2 3 4 5
1 2 3 4
1 2 3
1 2
1
```

```
n = 5
for i in range(1, n+1)
    for j in range(1, n+1)
        if (i+j <= n+1)
            print(j, end=' ')
        else:
            print(' ', end=' ')
    print()
```

```
5 4 3 2 1
5 4 3 2
5 4 3
5 4
5
```

```
n = 5
for i in range(1, n+1):
    for j in range(1, n+1):
        if (i+j <= n+1):
            print(n+1-j, end=' ')
        else:
            print(' ', end=' ')
    print()
```

```
n=5    k= chr ord('A')
for i in range(1, n+1):
    for j in range(1, n+1):
        if (i+j < n+1):
            print(chr(k), end=' ')
        k+=1
        else
            print(' ', end=' ')
    print()
```

A B C D E
F G H I
J K L
M N
O

```
n=5
for i in range(1, n+1):
    k = ord('A')
    for j in range(1, n+1):
        if (i+j < n+1):
            print(chr(k+n+1-j), end=' ')
        else:
            print(' ', end=' ')
    print()
```

E D C B A
E D C B
E D C
E D
E

```
n= 5
for i in range(1, n+1):
    k = ord('A')
    for j in range(1, n+1):
        if (i+j < n+1):
            print(chr(k), end=' ')
            k+=1
        else:
            print(' ', end=' ')
    print()
```

A B C D E
A B C D
A B C
A B
A

```
1 2 3 4 5
  1 2 3 4
    1 2 3
      1 2
        1
```

```
n = 5
for i in range(1, n+1):
    for j in range(1, n+1):
        if (i <= j):
            print(j+1-i, end=' ')
        else:
            print(' ', end=' ')
    print()
```

```
A B C D E
  1 2 3 4
    A B C
      1 2
        A
```

```
n = 5
for i in range(1, n+1):
    k = ord('A')
    for j in range(1, n+1):
        if (i <= j):
            if (i%2 == 0):
                print(j+1-i, end=' ')
            else:
                print(chr(k), end=' ')
                k += 1
        else:
            print(' ', end=' ')
    print()
```

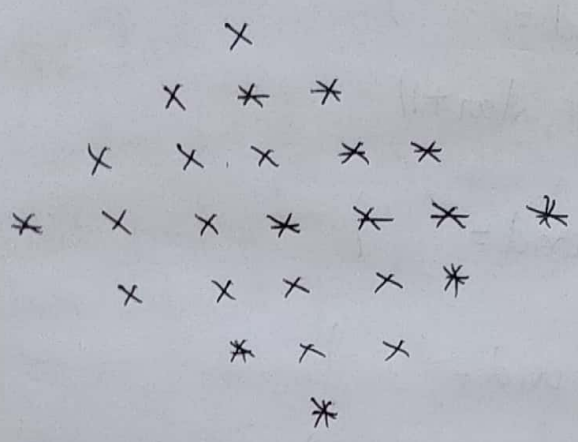
```
      *
    * * *
  * * * * *
* * * * * * *
```

```
n=4
space=n-1
star=1
for i in range(1,n+1):
    for j in range(1,space+1):
        print(' ',end=' ')
    for k in range(1,star+1):
        print('*',end=' ')
    space-=1
    star+=2
    print()
```

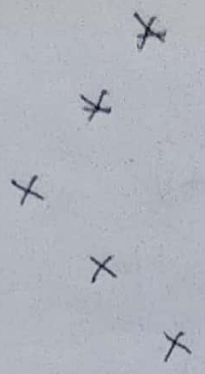
```
      1
    2 2 2
  3 3 3 3 3
4 4 4 4 4 4 4
```

```
n=4
space=n-1
star=1
for i in range(1,n+1)
    for j in range(1,space+1):
        print(' ',end=' ')
    for k in range(1,star+1)
        print(i,end=' ')
    space-=1
    star+=2
    print()
```

```
n = 4
for i in range(1, n+1):
    for j in range(1, n-i+1):
        print(' ', end=' ')
    for k in range(1, 2*i):
        print(i, end=' ')
    print()
```



```
n = 7   space = n-1   star = 1
for i in range(1, n+1):
    ~~if i <= n/2:~~
    for j in range(1, space+1):
        print(' ', end=' ')
    for k in range(1, star+1):
        print(i, end=' ')
    ~~space -= 1~~
    if i > (n/2):
        space += 1
        star -= 2
    else:
        space -= 1
        star += 2
    print()
```



```
n = 5   space = n - 1   star = 1
for i in range(1, n+1):
    for j in range(1, space+1):
        print(' ', end=' ')
    for k in range(1, star+1):
        if(k == 1):
            print('*', end=' ')
        elif(k == star):
            print('*', end=' ')
        else:
            print(' ', end=' ')
    print()
    if(i > n/2):
        space += 1
        star -= 2
    else:
        space -= 1
        star += 2
```

Factorial

```
n = 4
factorial = 1
for n in range(1, n+1):
    factorial = factorial * n
print factorial.
```

Prime

```
Number = 409    flag = 0
for n in range (2, n//2):
    if (number%.2 == 0):
        print ('Number is not prime')
        flag = 1
if (flag == 0)
    print ('Number is prime')
```

INCOMPLETE

## FUNCTION

It is a set of instruction designed to perform specific task.

Function are of two types built in function

① Built in function
② User defined function

Built in function are predefined function which are predefined in the software to perform specific task.

Eg `len('Hello')`
5

User defined function are the function that are created by the user to perform some task.

Syntax:

```
def functionName (arguements)
    <-->  Body of func
    -->
    return data1, data2   (return is optional)
```

arguements → Parameters to be passed to the function
Body of func → Set of instruction to be performed by func
Return → It is used to return data from the function back to the caller

## Return

Statement that is used to return the values or data from a function to a caller.

Syntax: return data

or

return data1, data2

Return statement can return single or multiple values.

Always return statement of a function must be returned inside a function.

## How to call a function

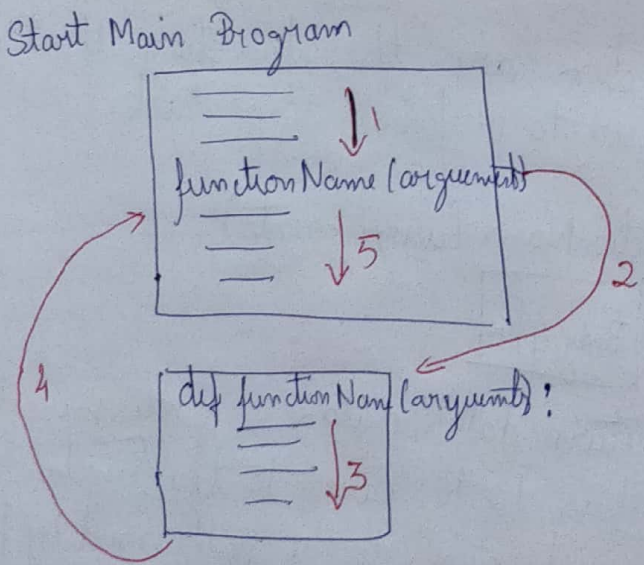
Syntax

var = functionName(arguments)

or

functionName(arguments)

## PROGRAM CALLING FUNCTION




- As long as function is not called function will not be executed.
- Function definition must be provided before calling

a function
- Return statement is not mandatory for each func
- Arguments of the function are also not mandatory.

Ex:
```
print ('control is outside')
print ('Hello')
def disp():
    print ('control inside the function')
print ('bye')
disp()
```

Ex2
```
print(1)
print(2)
def fadd(a,b):
    print ('inside add method')
    return a+b
print(add(4,3))
print(4)
print('End')
```

O/P
```
1
2
Inside add mt
7
4
End
```

User defined functions are broadly classified into 4 types

① Function without argument and without return type
② Function without argument & with return type
③ Function with argument & without return type
④ Function with argument & with return type.

Function without argument & without return type.

These kind of function are used whenever operation to be performed without the user input and not returning back anything to the user.

Ex: def disp():
    print('Inside disp function')

## FUNCTION WITHOUT ARGUEMENT WITH RETURN TYPE

These type of functions are used whenever we need not accept the input to perform the action or operation without returning the result.

Ex: def add():
    a = input()
    b = input()
    return a+b

## FUNCTION WITH ARGUEMENT AND WITHOUT RETURN TYPE

In these type of function are used whenever the operation need to be performed with user input and without returning the values

def add1(a,b):
    c = a+b
    print(c)

## FUNCTION WITH ARGUEMENT WITH RETURN TYPE

Here the function will take the arguement and will give the return value as well

Ex: def sub(a,b):
    c = a+b
    print(c)

**INPUT** – Statement of input is used to accept the input value from the user. The input taken from the user using input function is always in string format.

– In order to change the input value received type casting must be done explicitly.

Syntax var = Input('User Instruction')

or

var = Type(Input('User Instruction'))

Ex
>> a = [int(input('Enter the values')]
Enter the value1
>> a
[1]
>> a = [input()]
1,2,3,4
>> a
['1,2,3,4']
>> input()
1,2,3,4
>> input()
1,2,3,4
1,2,3,4

>>> 1,2,3,4
(1,2,3,4)
>> a = 1,2,3
>> a
(1,2,3)
>> name = input('Enter Name')
Enter Name Tarun
>> Name
Tarun

# DYNAMIC INSERTION OF ELEMENTS IN GROUPED DATA TYPES LIKE LIST, SET & DICTIONARY

## Adding element dynamically into list

```
def dynlist():
    a=[]
    n= int(input('Enter the no of elements to be in list'))
    for i in range(n):
        n1= input('Enter the list element')
        a.append(n1)
        print(a)
```

## Adding element dynamically into set.

```
def dynset
    a={}
n= int(input('Enter the last element in the set'))
    for i in range(n):
        s = input('Enter the list element')
        a.add(s)
        print(a)
```

## Adding element dynamically to dictionary

```
def dyndict():
    a={}
    n= int(input('Enter key value pair'))
    for is in range(n):
        key = input('Enter the key')
        value= input('Enter the value')
        a[key]=value
        print(a)
```

Send a = [1, [1, 2], 33, [4, 5, 6]]

```
def dylistlist():
    a=[]
n = int(input('Enter Number element in a list'))
for i in range(n):
    value = input('Does it contain multiple element')
    if(value == 'YES'):
        m= int(input('Enter value in second inner list'))
        for x in range(m):
            b=[]
            m1 = input('Enter the inner element')
            b.append(m1)
        print(a)
    else:
        n1 = input('Enter the single element')
        a.append(n1)
        print(a)
```

```
a=[]
n = int(input('Enter the number of element in a list'))
for i in range(n):
    n1= int(input('Enter the no of nested list element'))
    if n1==1:
        v1 = input('Enter the list element')
        a.append(v1)
        print(a)
    else:
        b=[]
        for i in range(n1):
            v2 = input('Enter the list element')
            b.append(v2)
        a.append(b)
```

```
print(a)
```

Enter the data

```
a=[1,2.5,[1,2],{1,2,3},{'eid':123,'ename':'Jaffy
```

```
def dynlist
    a=[ ]
    n= int(input('Enter the number of element in
                  the list'))
    for i in range(n):
        e= int(input('Enter number of element in list'))
        if n==1:
            v1= input('Enter the list element')
            a.append(v1)
            print(a)
        else:
            v2= input('Enter the type of element')
            if (v2=='LIST'):
                b[]
                n2= int(input('Enter the no of value of the lists
                    if n2==1:
                        v2= Input('Enter the list element')
                        a.append(v2)
                        print(a)
                else:
                    b[]
                    for(x in range(n2)
                        v2= input('Enter the list element')
                        b.append(v2)
                    a.append(b)
                    print(a)
```

ERROR in float type

```
elif (v2 == 'SET'):
    a1 = set()
    n3 = int(input('Enter the number of element in set'))
    for i in range(n3):
        s = input('Enter the list element')
        a1.add(s)
        print(a)
    a.append(a1)
elif (v2 == 'DICT')
    d = {}
    n4 = int(input('Enter number of key value pair'))
    for i in range(n4):
        key = input('Enter the key')
        value = input('Enter the value')
        d[key] = value
        print
    a.append(d)
    print(a)
```

```
a = {'a': 'apple', 'b': {'b1': 'ball', 'b2': 'bat'}, 'c': {
'c1': 'call', 'c2': 'cat', 'c3': 'character'}
```

Enter multiple element in dictionary

```
a={}
n=int(input('Enter the number of element key value pair'))
for i in range(n):
    key=input('Enter the key')
    m=int(input('Enter the number of dict element'))
    if m==1:
        value=input('Enter the value')
        a[key]=value
        print(a)
    else:
        b={}
        for j in range(m):
            key1=input('Enter the nested dict key')
            value1=input('Enter the nested dict value')
            b[key1]=value1
        a[key]=b
        print(a)
```

```
a=[]
n=int(input('Enter the no of list elements'))
for i in range(n)
    type=input('Enter the value type')
    if type=='int':
        value=int(input('Enter the element'))
        a.append(value)
        print(a)
    elif type=='float':
        value=float(input('Enter the element'
```

```
        a.append(value)
        print(a)
    elif type=='list':
        b=[]
        n1=int(input('Enter the number of nested
                      list elements'))
        for j in range(n1):
            value=int(input('Enter the element'))
            b.append(value)
        a.append(b)
        print a
    elif type=='set':
        b1=set()
        n2=int(input('Enter the no of next element
                      in set'))
        for j in range(n2):
            value=int(input('Enterthe element'))
            b1.add(value)
        a.append(b1)
        print(a)
    elif type=='dict':
        b2={}
        n3=int(input('No of element in dict'))
        for j in range(n3)
            key=input('Enter the key')
            value=input('Enter the value')
            b2[key]=value
        a.append(b2)
        print(a)
```

a={'a':[1,2,3], 'b'={4,5,6}, c:{'eid'=1234, 'ename': Jack}}

```
a={}
n= int(input('Enter number of element'))
for i in range(n)
    key = input('Enter the key')
    type = input('Enter the value type')
    if type=='list'
        b=[]
        n1= int(input('Enter number of element in list'))
        for j in range n1
            value = int(input('Enter the element'))
            b.append(value)
        a[key]=b
        print(a)
    elif type=='set'
        s=set()
        n2= int(input('Enter the number of set elemet
        for j in range(n2):
```

when two method or a function have the same name the implementation get overridden.

```
def add(a,b):
    print('Inside 1st add method')
    return a+b
def add(a,b,c):
    print('Inside 2nd add method')
    return a+b+c
```

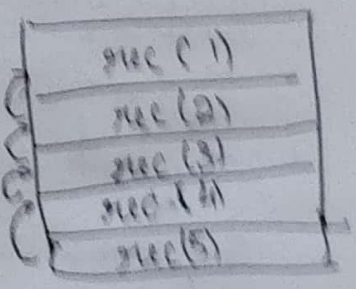
## RECURSIVE FUNCTION

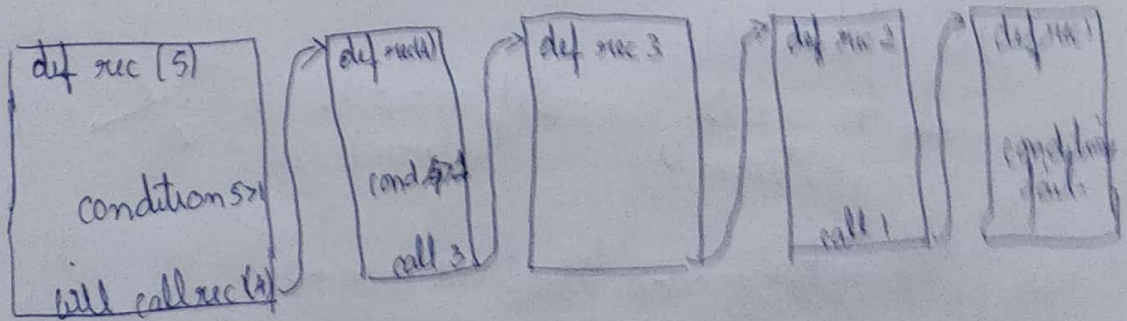
- Function which call itself until the given condition is satisfied is called recursive function. Recursive function can be used for avoid loops in some cases.
- It makes used of stack memory space.

Example

5 4 3 2 1

```
def rec(n):
    print(n)
    if n>1:
        rec(n-1)
```



3 4 3 2 1 2 3 4 5

```
def rec(n)
    if (n ==
    print(n, end=' ')
    if n>1:
        rec(n-1)
    elif n==1:
        rec(n+1)
        if (n -
    print(n, end=' ')
```

```
def rec(n)
    print(n, end=' ')
        if n>1:
            rec(n-1)
            print(n, end=' ')
```

O/p 5 4 3 2 1 2 3 4 5

## Factorial of number

```
def recfac(n)
    fact = 1    result = 1
    if (n==0):
        fact result = fact
    elif (n==1):
        result = n * fact(n-1)
```

```
def fact(n)
    if n in [0,1]:
        return 1
    else:
        return n * fact(n-
```

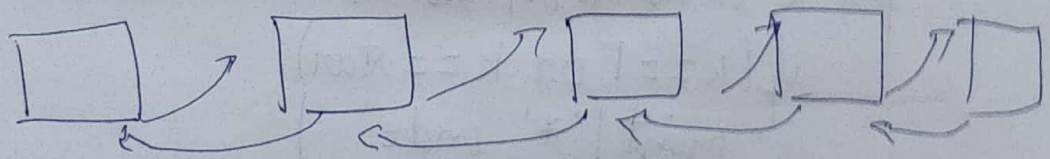
```
def fibo(n)
    if n ==0
        return 0
    def n==1:
        return 1
    else
        return n-1 + fibo(n-1) n-2
```

```
def fibo(n)
    result = 0
    print(result)
    for i in range(n)
def fibo(n)
    if n<=1 :
        return n
    else
    returnfibo(n-1)+fibo(n-2)
```

---

KARMAMRAK

```
a= KARMA    l=a.length()
rev def fibostr(a)
    def s1(a,b):
        print(b[a], end=' ')
        if a< len(b)-1:
            s1(a+1,b)
            print(b[a], end=' ')
```



## ALIAS

Copying a address of a function to a another function is known as aliasing.

Ex:

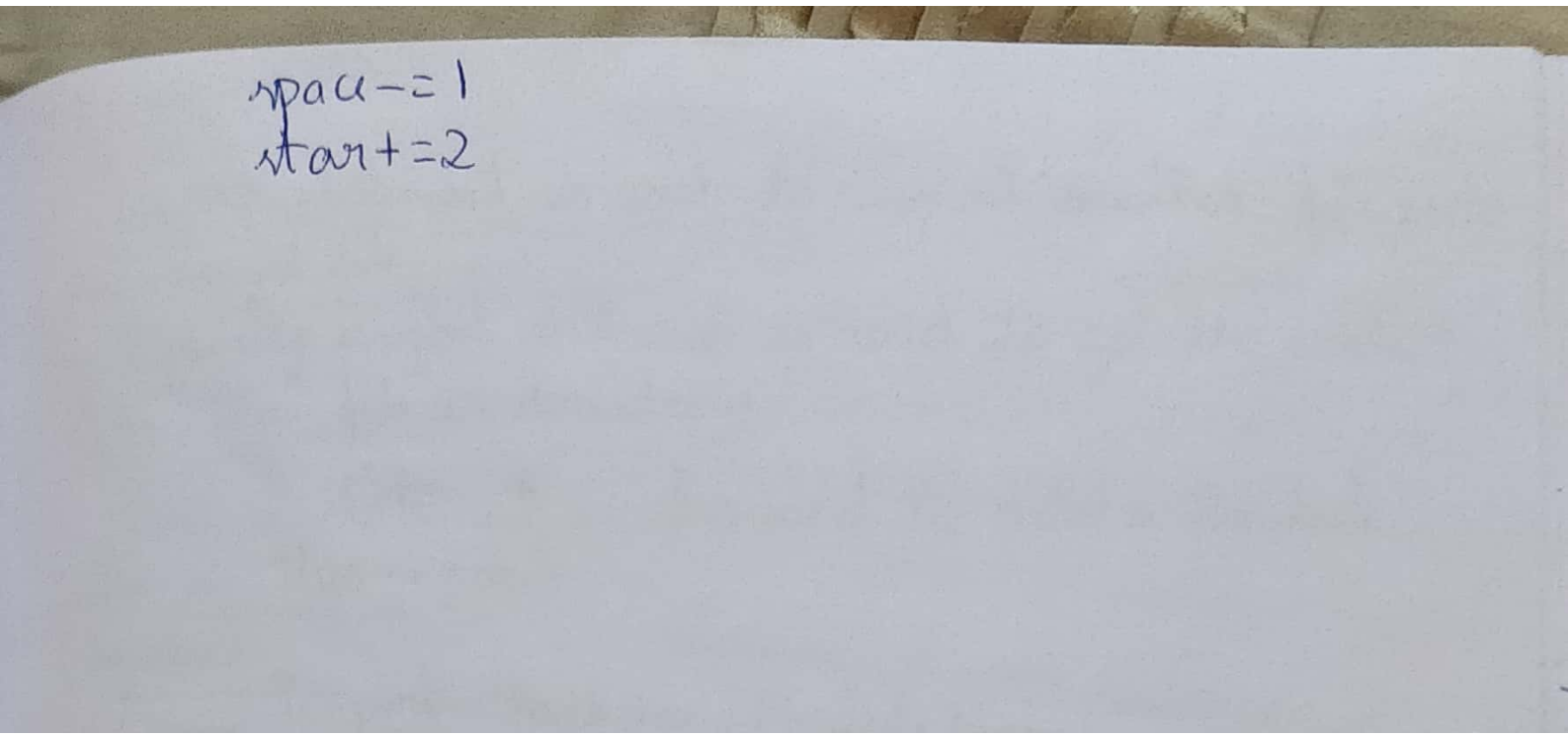
```
def fun()
    print('Inside fun')

a = fun
>> a()
o/p Inside fun
```

Id of both will be same as they are pointing to same memory space

NOTE: Here both fun() & a() are same because the addrss of a function fun is assigned to a

```
      *
    * A *
  * A B A *
* A B C B A *
  * A B A *
    * A *
      *
```

```
import math
n=7
space=n-1
star=1
for i in range(1,n+1):
    x=ord('A')
    for j in range(1,space+1):
        print(' ',end=' ')
    for k in range(1,star+1):
        if(k==1 or k==star):
            print('*',end=' ')
        else:
            print(chr(x),end=' ')
            if(math.ceil(star/2)>k):
                x+=1
            else:
                x-=1
    print()
    if(i>n/2):
        space+=1
        star-=2
    else:
```



## IMPORT

Import statement is used to import another file into the current file.

Basically import statement is used to get the content of another file or directory

Import statement is also used to add a module into another module.

Syntax: Import Package / Module Name

## MODULES

Modules are nothing but the source file which expoess class function and variables.

## PACKAGE

Package is collection of modules and it is also called as directories

Modules are classified into two types

① Built-in Modules
② User Defined Modules

## BUILT-IN MODULES

They are predefined with-in the software

## USER DEFINED MODULES

User defined modules are the modules which are created by the user

Modules helps in code reusability

N Examples of built-in modules

```
import math
dir(math)
['__doc
```

```
>> factomath.factorial(5)
120
```

→ Suppose if entire module is imported then the function is called as

```
ModuleName.functionName
```

Eg Math.ceil(4.5)

→ Suppose if the specific functionality or a function of a module is imported then function is called as

```
FunctionName()
```

Example to import specific single function from a module.

```
from ModuleName Import FunctionName
Calling
FunctionName()
```

```
from math import sqrt, log
>> sqrt(9)
   3.0
>> log(10,2)
   3.32
```

```
>> from math import *
>> factorial(5)
120
>> sqrt(8)
```

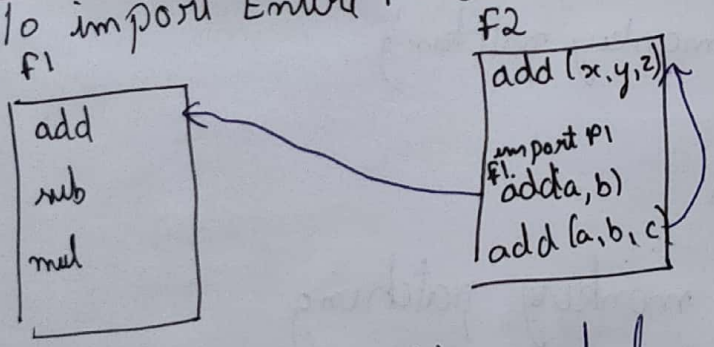
FF2.py

```
def add(a,b,c):
    print('Inside add in ff2')
    return a+b+c
# To import the entire user defined module
import ff1
print(add(1,2,3))
# To print add of ff1
print(add(1,2))
```

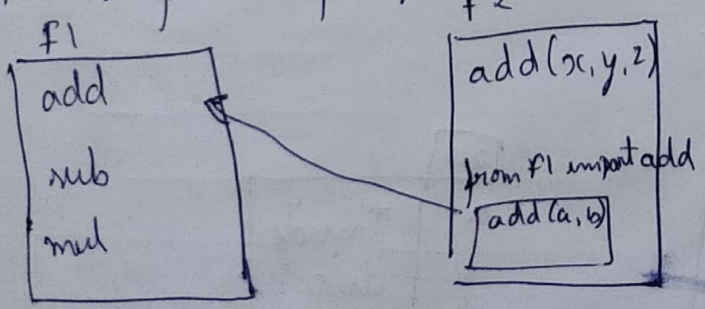
FF1

```
def add(a,b):
    print('Inside Add')
    return a+b
def sub(a,b):
    print('Inside Sub')
    return a-b
def mul(a,b)
    print('Inside Mul')
    return a*b
def div(a,b)
    print('Inside Div')
    return a/b
```

To import Entire Module




To import specific module



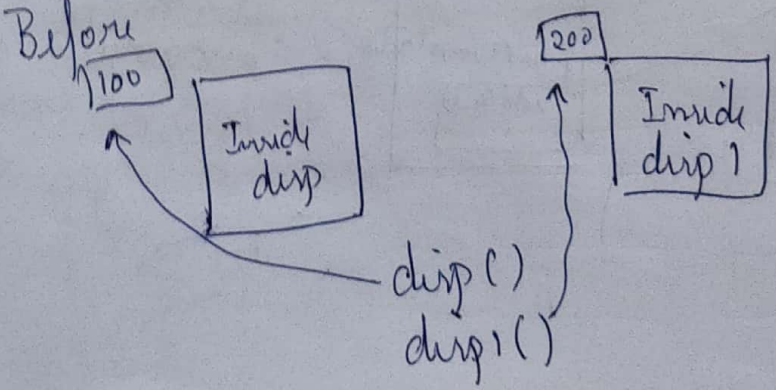

It has overwritten add

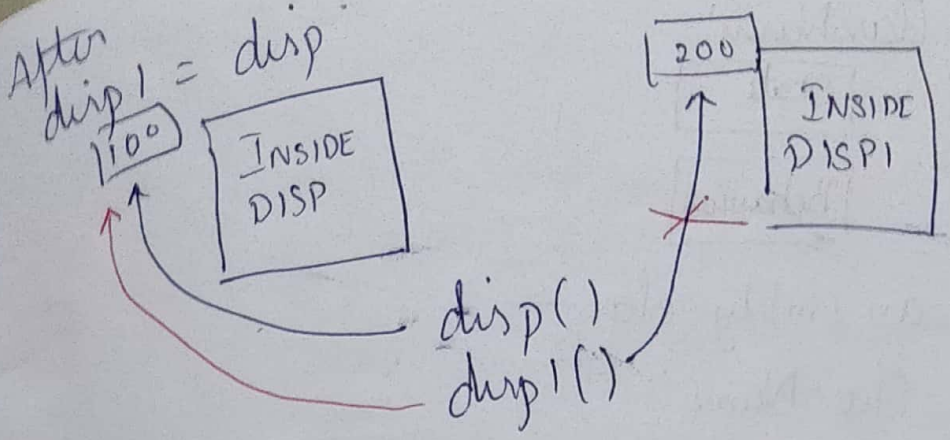
# MONKEY PATCHING

Phenomenon of overriding the address of one function with another function is known as monkey patching.

```
def disp():
    print('Inside disp')

def disp1()
    print('Inside disp1')
print('Result before monkey patching')
disp()
disp1()
disp1 = disp
print('Result after monkey patching')
disp()
disp1()
```

O/P – Result before monkey patching
Inside disp
Inside Disp1
Result after monkey patching
Inside disp
Inside disp






# OBJECT ORIENTED PROGRAMING

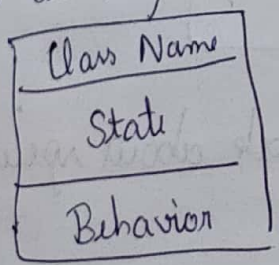
## CLASS

Class is a user defined data type or it is a area which consist of state & behavior of a real world entity.

Where,

State is a data members and the behavior is a functions or the methods

Whenever we want to represent a class then it has to be represented in the following way




The following representation is known as class diagram.

## OBJECT

Object is a variable created using the user defined data type called class. It is also called as instance of a class.

Syntax:

```
Class ClassName:
    -- State
    -- Behavior
```

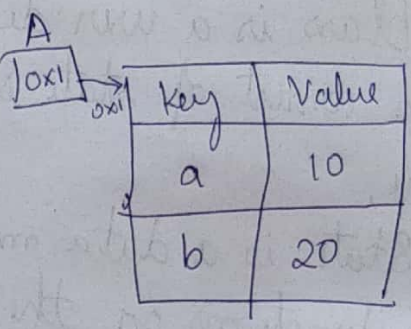
To create an empty class

```
class ClassName:
    Pass
```

Pass is a keyword used to represent class as empty

```
class A:
    pass
    a=10
    b=20
print(A.a, A.b)
```

o/p: 10 20




To create an object.

```
objName = className(arguments)
```

Only if we speak about specific data

```
class A:
    a=10
    b=20
ob1 =A()
ob =A()
print(A.a, A.b)
print(ob.a, ob.b)
print(ob1.a, ob.b)
```

| Key | Value |
|---|---|
| a | 10 |
| b | 20 |

| Key | Value |
|---|---|
| a | a1 |
| b | a2 |

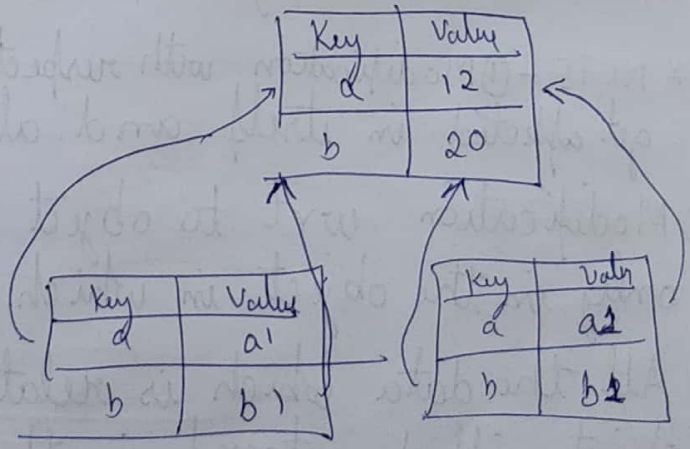
| Key | Value |
|---|---|
| a | a1 |
| b | a2 |

whenever object is created automatically dictionary will be created.

Look into document of key value pair in class

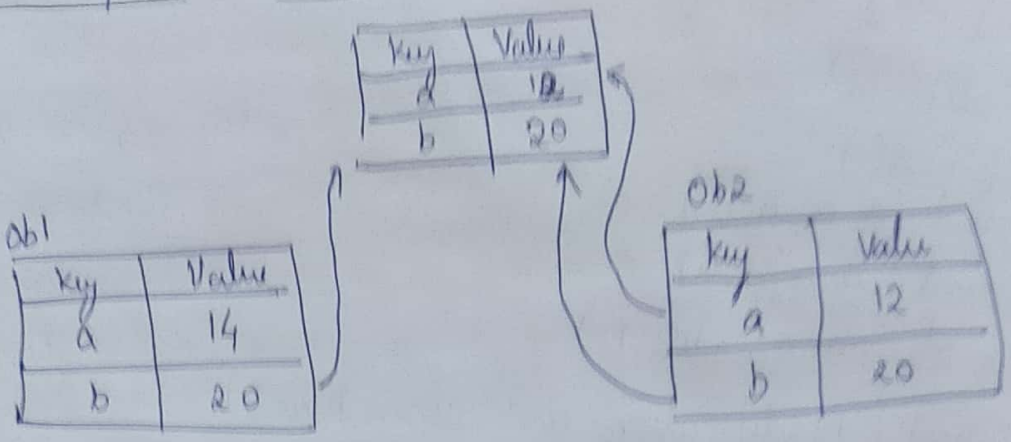
```
class A:
    a = 10
    b = 20
ob1 = A()
ob = A()
print (A.a, A.b)
print (ob.a, ob.b)
print (ob1.a, ob2.b)
print ('Modification wrt class Name')
A.a = 12
print (A.a, A.b)
print (ob.a, ob.b)
print (ob1.a, ob.b)
print ('Modification wrt object ob')
ob.a = 14
print (A.a, A.b)
print (ob.a, ob.b)
print (ob1.a, ob1.b)
```

o/p
10 20
10 20
10 20
12 20
12 20
12 20
12 20
14 20
12 20



Whenever the modification is done on class name changes get effected in itself & all its objects

Modification wrt objid




Future change in class

```
A.a = 18
print(A.a, A.b)
print(ob.a, ob.b)
print(ob1.a, ob1.b)
```

18, 20
14, 20
16, 20

Since a is already dereferenced.

* NOTE - ① Modification with respect to class Name changes get affected in itself and all its decendents (object)

② Modification wrt to object changes get affected only in the object in which we have modified

③ All the data which is related to a class or an object will be stored in the form of dictionary where dictionary keys are nothing but name of the state or name of the function or values are

nothing but the data or the address of the behaviour or function which is stored in the method area.

When we talk about data, data is of two types

① Generic Data
② Specific Data

Generic Data: It is a data which is generally designed or general or common for all the objects. We call this kind of data as class data and a variable which has a class data are known as class variable. Thus this variables are defined inside a class or stored inside a class.

SPECIFIC DATA: It is a data which is specific for each individual object. As the data is specific we dont store it inside a class, we store it only inside an object.

- As this data is stored only inside an object we call it as object members.
- These object members are been initialized inside the special method with the name

__init__

- It must take atleast one arguement ie. address of object in which we wanted to insert the data
- In other programming language we call this kind of methods as a constructors or this method is also called as initialization method.

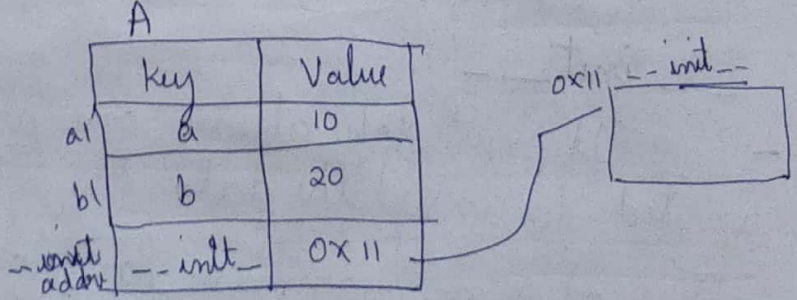
Constructor will be invoked at the time of object creation by the application itself.
We don't have to call externally to this particular function
That one argument which holds the address of an object inside the constructor is named as self as per industrial standards.

As a developer or a programmer we can give any name for it but whenever we are working with realtime environment or a team of people we must follow industrial standards.

```
def __init__(self, arguments):
    self.argument = value
```

CONSTRUCTOR WITHOUT ANY ARGUEMENT

```
class A:
    a=10
    b=20
    def __init__(self):
        print('INIT is difined')
        print('Address of the object')

ob=A()
print(ob)
ob1=A()
print(ob1)
```




| Key | Value |
|---|---|
| a | a1 |
| b | b1 |
| __init__ | __init__ addr |

| Key | Value |
|---|---|
| a | a1 |
| b | b1 |
| __init__ | __init__ addr |

Calling of constructor

In an object

ob.__init__(arguments)

IN AN CLASS OUTSIDE

A.__init__(self, arguments)

>> A.__init(obj)

INIT is invoked

Address of the object <__main__.A object at 0x00022A8>

ob.__init()

INIT is invoked

Address of the object <__main__.A object at 0x00022A8

Write a class where there is bank name, IFSC, Manager Name, Phone number.

```
class Customer:
    bankName = 'SBI, Chamarajpet'
    IFSC = '988SBI'
    BMGR = 'Raghuram'
    def __init__(self, Name, PhoneNumber):
        print('INIT is started')
        self.Name = input('Enter the name')//Name
        self.PhoneNumber = input('Enter the number')//PhoneNumber
ob = Customer('Rama', 100)
print(Customer.BMGR, Customer.bankName, Customer.IFSC)
print(ob.BMGR, ob.bankName, ob.IFSC, ob.Name, ob.PhoneNumber)

ob1 = Customer('Dinga', 420)
print(Customer.BMGR, Customer.bankName, Customer.IFSC)
print(ob1.BMGR, ob1.bankName, ob1.IFSC, ob1.Name, ob1.PhoneNumber)
```

O/P Prashanth, SBI Chamarajpet, 988SI

Raghuram,

, Rama, 100

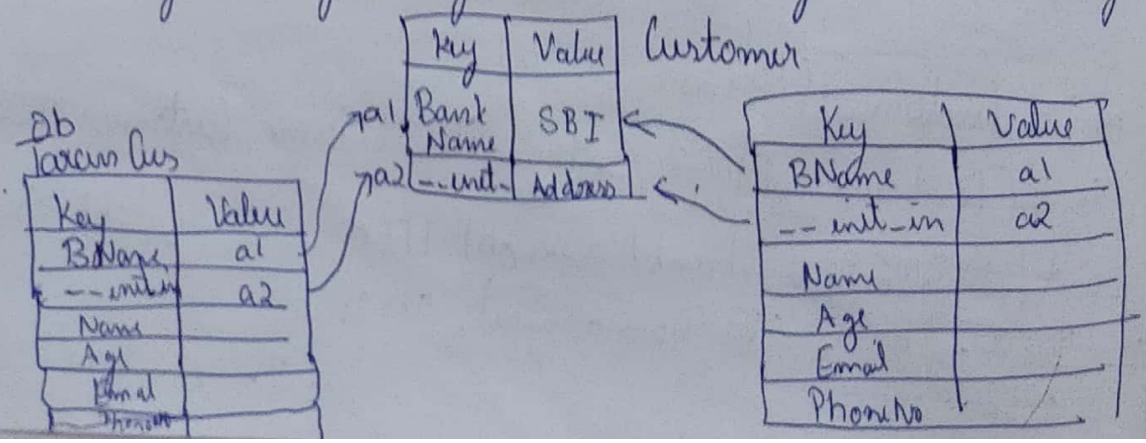
NOTE: Wherever constructor is not defined in a class the controller automatically insert the following code during the execution.

```
__init__(self):
    pass
```

But it cannot be externally invoked.

* Write a program for adding 3 customer age, name, email,

```
class Customer:
    BankName = 'SBI'
    def __init__(self, age, name, email, phoneNo):
        self.age = age
        self.Name = Name
        self.email = email
        self.phoneNo = phoneNo
TarunCus = Customer(27, Tarun, tarun@gmail.com, 988)
print(Customer.BankName)
print(TarunCus.age, TarunCus.Name, TarunCus.email, Tarun.pho
SantCus = Customer(27, Santosh, S@yahoo.com, 777)
NagaCus = Customer(27, Naga, N@yahoo.com 888)
print(SantCus.age, SantCus.Name, SantCus.email, Sant.phon
print(NagaCus.age, NagaCus.Name, NagaCus.email, NagaCus.ph
```

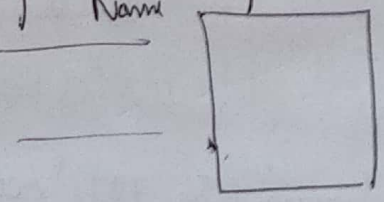


There are 3 types of methods or function that can be defined inside a class based on the usage of variable or data

1. Object Method / Function
2. Class Method / Function
3. Static Method / Function

## Object Method / Function

Syntax: def functionName(self):



How to call a function: className.functionName(~~self~~,objectName)

or

objectName.functionName()

These are the function that works with object data for all its operation. or object method that are used to access and modify members of the object.

```
class Customer:
    BName = 'SBI'
    BManager = 'Raghuram'
    IFSC = SBI1999
    def __init__(self, name, phno, age, mail):
        self.name = name
        self.phno = phno
        self.age = age
        self.mail = mail
```

```
def disp(self):
    print('Name of the bank is', self.BName)
    print('Bank Manager name is', self.BMgr)
    print('Ifsc code of branch is', self.IFSC)
    print('Name of branch customer', self.Name)
    print('Phone No of customer', self.PhoneNo)
    print('Age of customer', self.age)
    print('Mail of customer is', self.Mail)
c1 = Customer('yash', 123, 456, 'yash@yahoo.com')
c1.disp()
Customer.disp(c1)
c2 = Customer('salman', 654, 789, 'salman@kat.com')
c2.disp()
Customer.disp(c2)
```

## Dictionary Diagram

Customer

| | Key | Value |
|---|---|---|
| a1 | BName | Raghuram |
| a2 | MName | SBI |
| a3 | IFSC | SB1999 |
| a4 | __init__ | address of __init__ |
| a5 | disp | address of disp |

Salman

| Key | Values |
|---|---|
| BName | a1 |
| MName | a2 |
| IFSC | a3 |
| __init__ | a4 |
| disp | a5 |
| Name | Salman |
| PhoneNo | 5678 |
| Age | 51 |
| Mail | salman@gmail.com |

Yash

| Key | Value |
|---|---|
| BName | ~~SBI~~ a1 |
| mName | a2 ~~Raghuram~~ |
| IFSC | ~~SBI999~~ a3 |
| --init-- | ~~Add~~ a4 |
| disp | a5 |
| Name | Yash |
| PhoneNo | 1234 |
| Age | 31 |
| Mail | yash0 gmail |
| | |

Give address not value

Write a program to modify the name of salman using previous method

```
def modify(self):
    data = input('Enter the field name to modify')
    if data == 'name':
        self.Name = input('Enter the name')
    elif data == 'phoneNo'
        self.PhoneNo = int(input('Enter the phoneNo'))
    elif data == 'age':
        self.age = int(input('Enter the age')
    elif email == 'email':
        self.email = input('Enter the email')
    else:
        self.Name = input('Enter the name')
        self.PhoneNo = int(input('Enter the phone')
        self.age = int(input('Enter the age')
        self.email = input('Enter the email')
```

```
c1 = Customer('Tarun', 88864, 27, 'tarun@3')
c1.disp()
c1.modify()
```

---

Write a program for atm assignment

```
class Customer:
    BName = 'SBI'
    BManager = 'Raghuram'
    IFSC = 'SBI999'
    def __init__(self, name, phoneno, age, atm=fal
        self.name = name
        self.phoneNo = phoneNo
        self.age = age
        self.mail = mail
        self.atm = atm
    def disp(self):
        print('Name of bank is', self.BName)
        print('Bank Manager is', self.BManager)
        print('IFSC code of branch is', self.IFSC)
        print('Name of customer', self.customer)
        print('PhoneNo of customer', self.PhoneNo)
        print('Age of customer', self.Age)
        print('Mail of customer', self.Mail)
        print('AtmCard is', self.AtmCard)
    def GiveAtmCard(self):
        if(self.atm == False):
            self.atm == True
            print('Koodi atm card')
```

```
        else:
            print ('Manne alli nodi Sir')
C1 = Customer ('Yash', 123, 45, yash@yahoo, True)
C1.disp()
C1.GiveAtmCard()
C1.disp()
C2 = Customer ('Salman, 467, 63, salu@allu.com, false)
C2.disp()
C2.GiveAtmCard()
C2.disp()
```

o/p:

## CLASS METHOD

Class method are used to access and modify the class data

Python class method must always starts with:

"@ classMethod" keyword

```
@ classMethod
def functionName (cls):
    ____
    ____
```

Body.

The class methods can be accessed either by using class name or by using object name.

ClassName.classMethodName (arguements)

or

objectName.classMethodName (arguements)

CLS: It is used to hold the address of the class in which the class member have to be modified.

Continue previous program

```
@classMethod
def cModify(cls):
    n = input('Enter Modified data')
    if n == 'BName':
        cls.BName = Input('Enter the BName')
    elif n == 'BManager':
        cls.BManager = Input('Enter the BM
    elif n == 'IFSC':
        cls.IFSC = Input('Enter IFSC')
    else:
        cls.BName = Input('Enter Mana
        cls.BManager = Input('Enter Bank
        cls.IFSC = Input('Enter IFSC'
print('
# disp method of class
@classmethod
def disp1(cls):
    print('BName is', cls.BName)
    print('BankManager name is', cls.BmgrNa
    print('Bank IFSC Code is', cls.IFSC)
Customer.modify()
Customer.display()
```

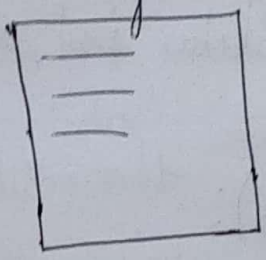
## STATIC METHOD

Static methods always start with @staticmethod keyword.

- Static method neither deal with class data (members) nor with object data members. It is used to perform some common operation. As it performs the common operation it is also called as generalized method.

SYNTAX:

```
@staticmethod
def function(arguments):
```



Example

From the previous program

```
@static
class Customer:
    BName = 'ICICI'
    BMgrNam = 'Vandini'
    IFSC = 'ICIC123'

    def __init__(self, name, age, sal, phno=None
                 mail=None, AtmCard=False):

        self.Name = name
        self.phone = phno
        self.age = age
        self.sal = sal
        self.mail = mail
        self.AtmCard = AtmCard
```

```
# display method
    def disp(self):
        print('Name of bank', self.BName)
        print('Bank Manager', self.BMgrName)
        print('Ifsc Code of branch', self.IFSC)
        print('Name of Customer', self.Name)
        print('PhoneNo', self.PhoneNo)
        print('Age of Customer', self.age)
        print('Salary is', self.salary)
        print('Mail of customer', self.mail)
        print('Atm Status is', self.AtmCard)

    @staticmethod
    def calci(a, b):
        n = input('Enter the type of operation performed')
        if n == 'add':
            return a+b
        elif n == 'sub':
            return a-b
        elif n == 'mul':
            return a*b
        elif n == 'div':
            return a/b

C1 = Customer('Yash', 31, 12009, 431243, 'tarun@gmail', True)
C1.disp()
a = C.calci(C1.sal, 1000)
```

NOTE: Static method can be accessed using class Name or object

objectName.StaticMethodName(arguments)

or

className.StaticMethodName(arguments)

```
@staticmethod
def cal(i):
    n = input('Enter the type of operation to be performed')
    a = int(input('Enter the salary'))
    b = int(input('Enter the EMI'))
    if n == 'add':
        return a+b
    elif n == 'sub':
        return a-b
    elif n == 'mul':
        return a*b
    elif n == 'div':
        return a/b

c1 = Customer('Yash', 31, 12000, 43212, 'taru@gmail')
c1.disp()
n1 = c1.cal()
print('Total outstanding salary after emi', n1)
```

Dictionary diagram method dictionary

Customer

| | Key | Value |
|---|---|---|
| a1 | BName | Rajhu |
| a2 | BMgrName | SBI |
| a3 | IFSC | SBI222 |
| a4 | init | address |
| a5 | Cal i | Address |

c)

| Key | Value |
|---|---|
| BManag | a1 |
| BName | a2 |
| IFSC | a3 |
| -init | a4 |
| Name | Yash |
| Phone No | 4342 |
| Age | 31 |
| Mail | tarun@gmail |
| Calci | |

~~INHERITANCE~~

NOTE

Object Method

```
class A:
    def disp(self):
        print('Inside disp object method without argu
    def disp1(self, a, b):
        print('Inside disp1 object method with argument
oa = A()
A.disp(oa) # calling obj method using classname without arg
A.disp1(oa, 3, 4)#                                  with arg

# calling the object method using object Name
oa.disp()  # without argument
oa.disp(3, 4) # with argument
```

## Class Method

```
class A:
    @classmethod
    def disp(cls):
        print('Inside disp class method without arguments')
    @classmethod
    def disp1(cls,a):
        print('Inside disp1 class method with argument')

oa=A()
A.disp()
A.disp1(3)      # Calling using class without argument
                with argument

oa.disp()
oa.disp1(3)     # Calling using object without argument
                with argument
```

## STATIC METHOD

```
class A:
    @staticmethod
    def disp1():
        print('Inside disp class of static without
               argument')

    @staticmethod
    def disp1(a):
        print('Inside disp1 class of static with
               argument')

oa=A()
A.disp()
A.disp1(3)      Using ClassName

oa.disp()
oa.disp1(3)     Using object
```

# INHERITENCE

It is object oriented programming language in which property one class will be used in another class

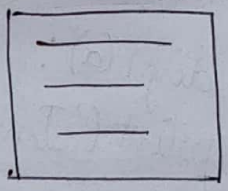
Types of inheritence
1. Single inheritence
2. Multiple inheritence
3. Multi-Level Inheritence
4. Heriarchical Inheritence
5. Hybrid Inheritence

## SINGLE INHERITENCE

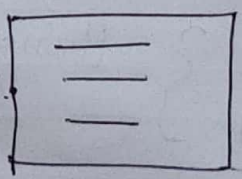
In single inheritence the derived class has the properties of single parent class. or when a class is derived using single parent class then it is called as single inheritence.

Syntax

class A:



class B:



```
class A:
    a = 10
    b = 20
class B(A)
    c = 30
    d = 40
```

a1, a2

| Key | Value |
|---|---|
| a | 10 |
| b | 20 |

| Key | Value |
|---|---|
| a | a1 |
| b | a2 |
| c | 30 |
| d | 40 |

NOTE: Base / Parent / Super Class are the class from which properties are been adopted.

Child / Derived / Sub Class are the classes which has the properties of another class.

Step 1: When the inheritence is adopted, child will get the properties of its parent class.

| Key | Value |
|---|---|
| a | 10 |
| b | 20 |

| Key | Value | |
|---|---|---|
| a | a1 | address of a |
| b | a2 | address of b |
| c | 30 | |
| d | 40 | |

STEP2: It is possible to modify parent class member using parent class

| Key | Value |
|---|---|
| a | 10 / 20 |
| b | 25 |

Override

| Key | Value |
|---|---|
| a | a1 |
| b | a2 |
| c | 30 |
| d | 40 |

Here the changes are affected on the child class members as well

STEP3

It is possible to modify child class members. Whenever the modification is done at the child class changes get affected only in child class and its objects

Here parent class remains unaffected

| | Key | Value |
|---|---|---|
| a1 | a | 10 |
| a2 | b | 20 |

| Key | Value |
|---|---|
| a | 12 |
| b | a2 |
| c | 30 |
| d | 40 |

STEP4: When the child class number is modified once any more parent class number modification, the change will not affected the child

Example

```
class A:
    a=10
    b=20
class B(A):
    c=30
    d=40
print(A.a, A.b)                                  10 20
print(B.a, B.b, B.c, B.d)                        10 20 30 4
print('Modification wrt parent class')
A.a=12
print(A.a, A.b)                                  12 20
print(B.a, B.b, B.c, B.d)                        12 20 30 4
print('Modification wrt child class')
B.a=14
print(A.a, A.b)                                  12 20
print(B.a, B.b, B.c, B.d)                        14 20 30 4
print('Further modification on the same number in
                the parent class
A.a=16
print(A.a, A.b)                                  16 20
print(B.a, B.b, B.c, B.d)                        14 20 30 4
```

When the constructor is defined in parent class

— In the child class always we store the derived members first followed by non derived members

— child class members (Non derived) are not accessable for parent class.

— When the parent class members is modified the changes get affected in the child and its objects

- When the constructor is defined in parent class the child class inherites the constructor of the parent class

Class A

| | Key | Value |
|---|---|---|
| a1 | a | 10 |
| a2 | b | 20 |
| a3 | __init__ | address of class A init |

Class B

| | Key | Value |
|---|---|---|
| b1 | a | a1 |
| b2 | b | a2 |
| b3 | __init | a3 |
| b4 | c | 30 |
| b5 | d | 40 |

Object of A

| | Key | Value |
|---|---|---|
| a1 | a | a1 |
| a2 | b | a2 |
| a3 | __init__ | address of a3 class A __init |

| Key | Value |
|---|---|
| a | b1 |
| b | b2 |
| __init | b3 |
| c | b4 |
| d | b5 |

Object of B

---

class A:
    a=10
    b=20
    def __init__(self):
        print('init in parent class')

class B:
    c=30
    d=40

```
print (A.a, A.b)
print (B.a, B.b, B.c, B.d)
print ('When the object is created of both the classes')
oa = A()
ob = B()
print (A.a, A.b)
print (oa.a, oa.b)
print (B.a, B.b, B.c, B.d)
print (ob.a, ob.b, ob.c, ob.d)
print ('Modification wrt parent class')
A.a = 12
print (A.a, A.b)
print (oa.a, oa.b)
print (B.a, B.b, B.c, B.d)
print (ob.a, ob.b, ob.c, ob.d)
print ('Modification wrt child class
B.a = 14
print (A.a, A.b)
print (oa.a, oa.b)
print (B.a, B.b, B.c, B.d)
print (ob.a, ob.b, ob.c, ob.d)
print ('Modification wrt parent class object')
oa.a = 16
print (A.a, A.b)
print (oa.a, oa.b)
print (B.a, B.b, B.c, B.d)
print (ob.a, ob.b, ob.c, ob.d)
print ('Modification wrt child class object
ob.a = 18
print (A.a, A.b)
print (oa.a, oa.b)
```

print (B.a, B.b, B.c, B.d)
print (ob.a, ob.b, ob.c, ob.d)

When the constructor is defined in both parent class and child class.

```
class A:
    a = 10
    b = 20
    def __init__(self):
        print('init in parent class')
class B:
    c = 30
    d = 40
    def __init__(self):
        print('init in child class')
```

ob = B

| | Key | Value |
|---|---|---|
| a1 | a | 10 |
| a2 | b | 20 |
| a3 | __init__ | Address of init |

Class A

| | Key | Value |
|---|---|---|
| b1 | a | a1 |
| | b | a2 |
| | init | Address of B |
| | c | 30 |
| | d | 40 |

object of A

| Key | Value |
|---|---|
| b1 | a1 |
| b | a2 |
| __init__ | a3 |

object of B

| Key | Value |
|---|---|
| a | b1 |
| b | b2 |
| __init__ | b3 |
| c | b4 |
| d | b5 |

NOTE: When the constructor is defined only in the parent class the child class and its objects contains parent class constructor.

When the constructor is defined in both parent and child class, then the child class & its objects

contain child class constructor

## CONSTRACTOR CHAINING

It is a phenomenon of calling the parent class constructor inside the child class.

```
class A:
    a=10
    b=20
    def __init__(self,x):
    self.x=x
    print('init in parent class')
class B(A):
    c=30
    d=20
    def __init__(self,x):
    print('init in child class')
    A.__init__(self,x) # calling the parent class inside
                         the child class using the
                         class name
    super().__init__(x) # calling the parent class inside
                          the child class using super
                          keyword / function
ob=B()
```

o/p :
init in child class
init in parent class
init in parent class

- It is possible to call parent class constructor for the child class (inside the child class constructor)

- The object of child class can call both parent and child class constructor provided the child class contains the call to it parent class constructor within it.

Calling parent class constructor from child class

- We can class parent class constructor either by using super keyword or function or by using the parent class name.

When super keyword is used to call or invoke parent class constructor we need not to pass self as an argument.

SYNTAX:

super().__init__(arguments)

Super keyword by default calls or invokes the immediate parent.

In Python Super keyword is used to call the base class but it is used occationally since super specifies only one parent class which fails in case of MULTIPLE INHERITENCE. As we will have more that one parent class in multiple inheritance.

- When the parent class constructor is called using the className we must pass 'self' as well as the arguments.

where self contains the object reference from which class it is called.

SYNTAX: className.__init__(self, arguments)

# CONSTRACTOR CHAINING EXAMPLE WITH ARGUEMENTS

```
class A:
    a=10
    b=20
    def __init__(self,x)
    self.x=x
    print('init in present class')
class B(A):
    c=30
    d=40
    def __init__(self,x):
    print('init in the child class')
      A.__init__(self,x) # calling the parent class init
                           the child class using the
                           class Nam
    super().__init_(x)
ob.B()
```

Class A

| | Key | Value |
|---|---|---|
| a1 | a | 10 |
| a2 | b | 20 |
| a3 | init | Address of A |

Class B

| | Key | Value |
|---|---|---|
| b1 | a | a1 |
| b2 | b | a2 |
| b3 | --init-- | Address of B |
| b4 | c | 30 |
| b5 | d | 40 |

object of B

| Key | Value |
|---|---|
| a | b1 |
| b | b2 |
| --init-- | b3 |
| c | b4 |
| d | b5 |

When the object method is defined in parent class then the child class and all its objects will inherit the parent class method. Thus it is possible to call the parent class object method from the child class and its objects.

```
class A:
    a=10
    b=20
    def __init__(self,x):
        self.x=x
        print('init in parent class')
    def disp(self):
        print('Object method disp in parent class')
class B(A):
    c=30
    d=40
    def __init__(self,x):
    print('init in child class')
    A.__init__(self,x) # calling the parent class
                       inside the child class using
                       the class Name
    super().__init__(x) # calling the parent class
                        inside the child class using
                        super keyword / function
ob=B(4)
# invoking the parent class object method using parent class Name
A.disp(ob) # parent class object method is invoked
# invoking the parent class object
B.disp(ob) # parent class object method is invoked
# invoking the parent class object method using child class object
ob.disp() # parent class object method is invoked
```

When object method is defined in both parent and child class with the same name

```
class A:
    a = 10
    b = 20
    def __init__(self, x):
        self.x = x
        print('Init in parent class')
    def disp(self):
        print('Object method disp in parent class')

class B:
    c = 30
    d = 40
    def __init__(self, x):
        print('Init in child class')          ①
        A.__init__(self, x)
        super().__init__(x)                   ②
    def disp(self):
        print('object method disp in child class')

ob = B(4)
A.disp(ob)  ⟶
B.disp(ob)
ob.disp()
```

A

| | Key | Value |
|---|---|---|
| a1 | a | 10 |
| a2 | b | 20 |
| a3 | __init__ | Address of init in A |
| a4 | disp | Address of disp in A |

B(4)

| | Key | Value |
|---|---|---|
| b1 | a | a1 |
| b2 | b | a2 |
| b3 | __init__ | Address of init in B |
| b4 | c | 30 |
| b5 | d | 40 |
| b6 | disp | a4 / Address of disp in B |

| Key | Value |
|---|---|
| a | b1 |
| b | b2 |
| __init__ | b3 |
| disp | b6 |
| c | b4 |
| d | b5 |

op
init in immediate/child class
init in parent class
Object method disp in parent
" "
child
child

## CLASS METHOD INHERITANCE

```
class A:
    a=10
    b=20
    def __init__(self,x):
        self.x=x
        print('init in parent class')
    @classmethod
    def disp(cls):
        print('class method disp in parent class')

class B(A):
    c=30
    d=40
    def __init__(self,x,y):
        self.y=y
        print('init in child class')
        A.__init__(self,x)

ob=B(4,3)
A.disp() # invokes class method in parent class
B.disp() # invokes class method in parent class
ob.disp() # invokes class method in parent class
```

o/p: init in child class
init in parent class
class method disp in parent class
class method disp in parent class
class method in parent class

Class A

| | Key | Value |
|---|---|---|
| a1 | a | 10 |
| a2 | b | 20 |
| a3 | --init-- | Address of init in A |
| a4 | disp | Address of disp in A |

Class B

| | Key | Value |
|---|---|---|
| b1 | a | ~~a1~~ |
| b2 | b | ~~20~~ a2 |
| b3 | --init-- | init in B address ~~A3~~ |
| b4 | --disp-- | a4 |
| b5 | c | 30 |
| b6 | d | 40 |
| | ~~X~~ | ~~3~~ |

Object of B

| Key | Value |
|---|---|
| a | b1 |
| b | b2 |
| --init-- | b3 |
| disp | b4 |
| c | b5 |
| d | b6 |
| Y | 3 |

Example: Class with classmethod disp

```
Class A:
    a=10
    b=20
    def __init__(self,x)
        self.x = x
        print("init in parent class")
    @classmethod
    def disp(cls):
```

print('class method disp in parent class')

```
class B(A):
    c=30
    d=40
    def __init__(self,x,y):
        self.y=y
        print('init in child class')
        A.__init__(self,x)
    @classmethod
    def disp(cls):
        print('class method disp in child class')

ob=B(4,3)
A.disp()  # invokes class method in parent class
B.disp()  # invokes class method in child class
ob.disp()  # invokes class method in child class
```

Class A

| | Key | Value |
|---|---|---|
| a1 | a | 10 |
| a2 | b | 20 |
| a3 | __init__ | Adrus of init in A |
| | disp | Address of disp in A |

Object of B

Class B

| | Key | Value |
|---|---|---|
| b1 | a | a1 |
| b2 | b | a2 |
| b3 | __init__ | init in B address |
| b4 | disp | Address of disp in B |
| b5 | c | 30 |
| b6 | d | 40 |

# STATIC METHOD INHERITENCE

```
class A:
    a=10
    b=20
    def __init__(self,x)
        self.x=
        print('Init in parent class')
    @staticmethod
    def disp(a,b)
        print('The value are',a,b)
class B(A):
    c=40
    d=50
    def __init(self):
        print('Init in child class')
ob=B()
A.disp()
B.disp()
ob.disp()
```

---

```
class A:
    a=10
    b=20
    def __init__(self):
        print('init in parent class')
    @staticmethod
    def disp():
        print('static method disp in parent class')
        return A.a+A.b
```

```
class B(A):
    c=30
    d=50
    def __init__(self):
        print('init in child class')

    @staticmethod
    def disp():
        print('static method disp in child class')
        return B.c-B.d

ob=B()
print(A.disp())
print(B.disp())
print(ob.disp())
```

o/p: 30
-20
-20

When both parent and child class are defined with two different types of method with the same name

```
class A:
    a=10
    b=20
    def __init__(self):
        print("init in parent class")
    def disp(self):
        print("Object method disp in parent class")
        return A.a+A.b

class B:
    c=30
    d=50
    def __init__(self):
        print('init in child class')
```

```
@staticmethod
def disp():
    print("Static method disp in child class")
    return B.c - B.d

oa = A()
ob = B()
print(A.disp(ob))      — 30
print(B.disp())        — -20
print(oa.disp())       — 30
print(ob.disp())       — -30
```

o/p :
-20
-30
20
-30

## MULTILEVEL INHERITANCE

Inheritance that takes place in difference level ie a new class is created using another derived cla

Syntax:

Parent Class
↓
Child Class
↓
SubChild / child1 class

In multi-level inheritance the derived member are inherited first

When the derived members are inherited the child will [ subchild of child1 ] will refer to the immediate parent

- When modification is done at parent level, all the child class will get effected hierarchy.
- When modification is done at immediate parent, the changes are limited to itself and its child. (sub child)
- Here the parent class will be uneffected
- When modification is done at sub child class the changes get effected in itself.
- Here parent and immediate parent remains uneffected.
-

STEP1 - Multilevel inheritance is created.

STEP2 - When modification is done at parent level the child class will get effected

STEP3 - When modification is done at the immediate parent. changes will get effected in itself and all its childs and also in their objects. Here the parent and parents object remains uneffect

STEP4

When modification is made at sub child class the changes get effected only in itself and its object. Here parent and immediate parent (child) includes their object remain uneffected.

Class A

| | Key | Value |
|---|---|---|
| a1 | a | 10 |
| a2 | b | 20 |

Class B

| | Key | Value |
|---|---|---|
| b1 | a | a1 |
| b2 | b | a2 |
| b3 | c | 30 |

Class C

| | Key | Value |
|---|---|---|
| c1 | a | b1 |
| | b | b2 |
| | c | b3 |
| | d | 40 |

STEP 1

STEP2

Class A

| | Key | Value |
|---|---|---|
| a1 | a | ~~10~~ 12 |
| a2 | b | 20 |

Class B

| Key | Value |
|---|---|
| a | a1 |
| b | a2 |
| c | 30 |

| Key | Value |
|---|---|
| a | b1 |
| b | b2 |
| c | b3 |
| d | 40 |

STEP3

Class A

| | Key | Value |
|---|---|---|
| a1 | a | 12 |
| a2 | b | 20 |

Class B

| | Key | Value |
|---|---|---|
| b1 | a | ~~12~~ 14 |
| b2 | b | a2 |
| b3 | c | 30 |

Class C

| Key | Value |
|---|---|
| a | b1 |
| b | b2 |
| c | b3 |
| d | 40 |

STEP4

Class A

| Key | Value |
|---|---|
| | |
| | |

Class B

| Key | Value |
|---|---|
| | |
| | |
| | |

Class C

| Key | Value |
|---|---|
| | |
| | |
| | |
| | |

```
class A:
    a = 10
    b = 20
class B(A):
    c = 30
    D = 40
class C(B):
    D = 40
print (A.a, A.b)
print (B.a, B.b, B.c)
print (C.a, C.b, C.c, C.d)
print ('Modification in parent class')
    a = 12
print (A.a, A.b)
print (B.a, B.b, B.c, B.d)
print (C.a, C.b, C.c, C.d)
print ('Modification wrt to immediate parent')
print (A.a, A.b)
print (B.a, B.b, B.c)
print (C.a, C.b, C.c, C.d)
print ('Modification wrt to sub child class')
    C.a = 16
print (A.a, A.b)
print (B.a, B.b, B.c)
print (C.a, C.b, C.c, C.d)
```

o/p

```
10 20
10 20 30
10 20 30 40
```

Modification in parent

```
12 20
12 20 30
12 20 30 40
```

Modification wrt immediate

```
12 20
14 20 30
14 20 30 40
```

Modication of child

```
12 20
14 20 30
16 20 30 40
```

## --init-- in MULTIPLE LEVEL INHERITANCE:

Defining the constructor only in parent class

- When the constructor is defined only in the parent class
- The immediate parent (child) and sub child (child) also refers to parent class constructor.
- The object created for any of the 3 levels of class will refer to same parent class constructor.

```
class A:
    a=10
    b=20
    def __init__(self,x):
        print('init in parent class')
class B(A):
    c=30
class C(B)
    d=40
oa = A(4)
ob = B(4)
ob = C(4)
```

Class A

| | Key | Value |
|---|---|---|
| a1 | a | 10 |
| a2 | b | 20 |
| a3 | __init__ | Address of init in A |

Object of A

| Key | Value |
|---|---|
| a | a1 |
| b | a2 |
| __init__ | a3 |
| x | 4 |

Class B

| | Key | Value |
|---|---|---|
| b1 | a | a1 |
| b2 | b | a2 |
| b3 | -init- | a3 |
| b4 | c | 30 |

Object of B

| Key | Value |
|---|---|
| a | b1 |
| b | b2 |
| -init-- | b3 |
| c | b4 |
| | |

Class C

| | Key | Value |
|---|---|---|
| c1 | a | b1 |
| c2 | b | b2 |
| | -init | b3 |
| | c | b4 |
| | d | 40 |

When constructor is defined at immediate parent or child.

Sub child will call its immediate parent class constructor.

The object of immediate parent or sub child will also refer to immediate parents constructor

Parent class constructor will be henceforth called by its object.

```
class A:
    a = 10
    b = 20
    def --init--(self):
        print('init in parent class')
class B:
    c = 30
    def --init--(self)
        print('init in immediate parent')
```

```
class C(B):
    d=40
oa=A()
ob=B()
oc=C()
```

o/p:
init in parent class
init in child class
init in child class

Class A

| | Key | Value |
|---|---|---|
| a1 | a | 10 |
| a2 | b | 20 |
| a3 | __init__ | Address of __init__ in A |

Class B

| | Key | Value |
|---|---|---|
| b1 | a | a1 |
| b2 | b | a2 |
| b3 | __init__ | Address of __init__ in B |
| b4 | c | 30 |

Class C

| Key | Value |
|---|---|
| a | b1 |
| b | b2 |
| __init__ | b3 |
| c | b4 |
| d | 40 |

- When the constructor is defined in sub child or child

① The object of each class will call its own construct

```
class A:
    a=10
    b=20
    def __init__(self):
        print('init in parent class')
class B(A):
    c=30
    def __init__(self):
        print('init in immediate parent')
class C(B):
    d=40
    def __init__(self):
        print('init in sub child / child 1 class')
oa=A()
```

```
ob = B()
oc = C()
```

o/p:
init in parent
init in child of immediate class
init in child or sub child.

Constructor of one class can be called inside of another class ie Constructor of parent class can be called within its child class constructor

- It is not possible to call the parent class constructor outside of child class constructor

```
class A:
    a = 10
    b = 20
    def __init(self):
        print('init in parent class')

class B(A)
    C = 30
    D = 40
    A.__init__(self)
    super().__init()

class C(B):
    E = 50
    B.__init__(self)
    super().__init()

oc = C()
```

Try this

```
class A:
    a = 10
    b = 20
    def __init__(self):
        print('init in parent class')
class B(A):
    c = 30
    def __init__(self):
        print('init in immediate class of parent')
        super().__init__()
class C(B):
    d = 40
    def __init__(self):
        print('init in sub child')
        super().__init()
OC = C()
```

O/P: init in sub child
immediate
parent

```
class A:
    a = 10
    b = 20
    def __init(self, x):
        print('Parent class init')
class B(A):
    c = 30
    def __init__(self, x, y):
        self.y = y
        print('Immediate class')
        super().__init__(x)
```

```
class C(B):
    d=40
    def __init__(self,x,y,z)
      self.z=z
      print('Sub Child Class')
      super().__init__(x,y)
```

## MULTILEVEL INHERITANCE FOR OBJECT METHOD

MULTILEVEL INHERITANCE FOR OBJECT METHOD without argument and with method in parent class

① Object method only in parent class
- The immediate parent (child) and sub child (child1) also refers to parent class ~~constructor~~ object method
- Object created for any of the 3 level of class will refer to same parent class object method

```
class A:
    a=10
    b=20
    def __init__(self):
        print('Init in parent class')

    def disp(self):
        print('Object method in parent class')

class B(A):
    C=30
    D=40
    def __init__(self):
        print('Init method disp in immediate parent class')

class C(B):
    E=50
    def __init__(self):
        print('Init method in child(child1) class')
```

```
oc = C()
A.disp(oc)
B.disp(oc)
oc.disp()
```

O/p:- Init in child class
Object method in parent class
Object method in parent class
Object method in parent class

---

Object Method multilevel inheritance with arguements with method present in parent class

```
Class A:
    a=10
    def __init__(self):
        print('Init of parent')
    def disp(self,x):
        print('Object Method disp in parent class')

Class B(A):
    b=20
    def __init__(self):
        print('Init of immediate parent')
    def disp(.

Class C(B):
    c=30
    def __init__(self):
        print('Init in child class')

oc=C(4)
A.disp(oc,3)
B.disp(oc,4)
oc.disp(5)
```

o/p :- Init in child class
Object method in parent class
Object method in parent class
Object method in parent class

② Multilevel inheritance with object method defined at immediate parent or child class

ⓐ Sub child will call its immediate parent class ~~constructor~~ object method

ⓑ The object of immediate parent and sub child will refer to immediate parent object method

ⓒ Parent class object method will be called by its own object.

```
Class A:
    a=10
    def __init__(self):
        print('Init of parent')
    def disp(self)
        print('Object Method disp in parent class')
Class B(A):
    b=20
    def __init__(self):
        print('Init in immediate parent')
    def disp(self):
        print('Object method disp in immediate parent')
Class C(B):
    c=30
    def __init__(self):
        print('Init in child class')
ob=B(2)
oc=C(4)  A.disp(oc)  B.disp(oc)  oc(disp(o)
```

o/p: Init in immediate parent
init in child class
Object method disp in parent class
Object method disp in immediate class

o/p Object method disp in immediate class

Multilevel Inheritance with argument including object method in parent and immediate class.

```
Class A:
    a=10
    def __init__(self,x):
        self.x=x
        print('Init in parent class')

    disp disp(self,a,b):
        print('Object method a1 in class A')
        return a+b

Class B(A)
        c=10
        def __init__(self,y):
            self.y=y
            print('Init in immediate parent (child) class')
        def a1(self,a,b):
            print('Object method a1 in class B')
            return A-B

Class C(B):
            d=40
            def __init__(self,x):
                self.z=2
                print('Init in subchild class')
oc = C(5)
print(A.disp(oc,2,3))
```

```
print (B.al(oc,3,4))
#print (C.al(oc,5,6))
print (oc.al(2,30))
```

o/p:

```
init in child1 class
object method al in classA
5
object method al in classB
-1
object method al in classB
-1
Objd method al in classB
-28
```

Methods can be overridden class, static or object method in multilevel inheritance

- When both parent class have same member then the child class always contains members or values from lastly inherited parent class as shown in the above example
- Any modification done to the first inherited class will not affect the child
- But any modification done at lastly inherited class will affect the child class.

Constructor in multiple inheritance

When only one of the ~~eta~~ parent class has constructor

At the time of object creation in child class, it calls constructor of parent class in which constructor is defined.

# Class Method for Multilevel Inheritance

Multilevel Inheritance for ~~object~~ class method without argument and with method in class.

① Class method only in parent class

```
class A:
    a=10
    b=20
    def __init__(self,x):
        self.x=x
        print('init in parent class')

    @classmethod
    def disp(cls,x):
        print('Object method disp in parent class')

class B(A):
    c=30
    d=40
    def __init__(self,x):
        print('init in immediate parent')

class C(B):
    e=30
    def __init__(self,x):
        print('init in child class')

ob=B(3)
oc=C(4)
A.disp(3)
B.disp(4)
ob.disp(2)
oc.disp(3)
```

o/p: init in immediate parent

init in child class
Object method disp in parent class
Object method disp in parent class
Object method disp in parent class
Object method disp in parent class

} Have class method in the print statement

(5) Class Method only in parent and immediate class

```
class A:
    a=10
    b=20
    def __init__(self,x):
        self.x=x
        print('init in parent class')

    @classmethod
    def disp(cls,x):
        print('Class method disp in parent class')

class B(A):
    c=30
    d=40
    def __init__(self,x):
        print('Init in immediate parent')

    @classmethod
    def disp(cls,y)
        print('Class method in immediate parent')

class C(B):
    e=50
    def __init__(self,x):
        print('init in child class')

ob=B(3)
oc=C(4)
A.disp(3)
B.disp(4)
ob.disp(2)
oc.disp(1)
```

O/p:- init in immediate parent
init in child class
Class method in disp in parent class
Class method disp in child class
Class method disp in child class
Class method disp in child class.

## STATIC METHOD IN MULTILEVEL INHERITANCE

class A: With only parent static method

```
a=10
def __init__(self,x):
    self.x=x
    print('init in parent class')

@staticmethod
def disp(x):
    print('Static method disp in parent method')

class B(A):
    c=30
    def __init__(self,x):
        print('init in immediate parent')
class C(B):
    c=50
    def __init__(self,x):
        print('init in child class')

ob=B(3)
oc=C(4)
A.disp(3)
B.disp(4)
ob.disp(2)
oc.disp(3)
```

O/p: init in immediate parent
init in child class
Static method disp in parent class
Static method disp in parent class