Write a Pgm to convert all the uppercase character present inside a input string to lower case character.

```
st = input('Enter the st:')
out = ''
for i in st:
    if 'A'<= i <='Z':
        out += chr(ord(i)+32)
    else:
        out = out + i
Print(out)
```

Write a Pgm to replace all the special characters by '*'

```
st= input('Enter the st:')
out = ''
for i in st:
    if not('a'<= i<='z' or 'A'<= i <= 'Z' or '0'<= i <= '9'):
        out = out+'*'
    else:
        out = out + i
Print(out)
```

Write a Pgm to Extract all the Even digits present inside a given string.

```
st = int(input('Enter the st:'))
out = ''
for i in st:
    if '0'<= i <= '9' and int(i)%2==0:
        out = out + i
Print(out)
```

Write a Prgm to find the square of all the Even numbers Present inside an homogeneous input list and store the o/p into o/p list.

```
l = [1, 2, 3, 4, 5]
Out = []
for i in l:
    if i%2 == 0:
        Out = Out + [i**2]
Print (Out)
```

if we have heterogeneous list we want do type cast for the given input

```
if type(i) == int and i%2 == 0
    out = out + [i**2]
```

Intermediate Termination in loop :

* Break :

Whenever we want to make the comout of looping statement we will make use of concept called intermediate termination in loop.

1. Break :

It is a keyword which is use to terminate the loop in between

→ As soon as control see a keyword called break it will comeout from the loop.

→ We can use break in both for loop and while loop.

→ We can write break without having any condition but which will be having no meaning

```
for i in range(1,5)
    Print (i)
    break
```

o/p : 1

Write a Pgm to run a while loop continuously unless until the user enters the proper password.

```
Opw = 'PySp123'
while True:
    Pw = input('Enter the pw:')
    if Opw == pw:
        break
    Print('Enter the proper pw')
```

Write a Pgm to print all the divisor of given num.

```
n = int(input('Enter the num:'))
for i in range(1,n+1):
    if n%i == 0:
        Print(i)
```

Write a Pgm to Print the smallest divisor of a given number

```
d = int(input('Enter the num:'))
for i in range(2, n+1):
    if d%i == 0 and d<=i:
        Print(i)
        break
```

Write a Pgm to check whether the given string is only alphabets are not

```
flag=0
st='hai'
for i in st:
    if not('a'<=i<='z' or 'A'<=i<='Z'):
        flag=1
        break
if flag==0
    Print('it is having only alphn')
else:
    Print('not only alphn')
```

if we don't use another flag the control doesn't understand where the break statement comes from either for or if but why if we give flag=0 as a var it's given str is not a alphabet flag=1 the given string is alphabet

Write a Prgm to check whether the given string is having only lower case alphabets are not

```
flag = 0
st = input ('Enter the str')
for i in st :
    if not ('a' <= i <= 'z') :
        flag = 1
        break
if flag == 0 :
    Print ('Only Lower case alpha')
Else :
    Print ('not only Lower alpha')
```

Write a Prgm to check whether the given number is Prime or not

```
n = int (input ('Enter the num :'))
f = 0
if n == 1 :
    Print ('neither Prime nor composite')
else :
    for i in range (2, n) :
        if n%i == 0 :
            f = 1
            break
    if f == 0 :
        Print ('Prime')
    else :
        Print ('not a Prime')
```

Write a Pgm to check whether the given list is having only integer num or not.

Wap to check whether the string is having only consonents are not
↓
other than vowels

```
l=[1,2,3,4.5,6,7.8]
f=0
for i in l:
    if type(l)==int:
        f=1
        Print(i)
        break
    else:
        Print('list is not having int num')
```

2. Write a Pgm to check whether the string is having only consonents are not

```
st=input('Enter the st:')
f=0
for i in st:
    if not('AEIOU aeiou')==st:
        f=1
        Print(i)
        break
    else:
        f=0
        Print('not consonents')
```

In-Built functions on string:

1. UPPER()

It's a function which is used to convert all the lowercase alpha present inside the original string into uppercase alpha.

syntax:

```
Var.upper()
```

Eg: * st = 'hello'

```
>>> st.upper()
>>> 'HELLO'
  * st
    'hello'
>>> ques = st.upper()
  * st
    'hello'
    ques
>>> 'HELLO'
  * st = 'hello12#'
    st.upper()
>>> 'HELLO12#'
```

2. LOWER()

Lower is a function it's used to convert all the uppercase character into lowercase character

syntax:

```
Var.lower()
```

Eg: st = 'HELLO'

```
st.lower()
>>> 'hello'
```

3. isupper()

It's a function which is used to check wether the given string is having only uppercase alpha or not

→ If it consists of only uppercase alpha result will be true else result will be false.

syntax: var.isupper()

Eg: st = 'HELLO'
st.isupper()
>>> True

st = 'HELLO12@'
st.isupper()
>>> True

st = 'HELLO'
st.isupper()
>>> False.

4. islower()

It's a function which is used to check whether the given string is having only lower-case alpha or not

→ If it's consist of only lowercase alpha result will be True Else result will be False.

syntax: var.islower()

Eg: st = 'hello'
st.islower()
>>> True

st = 'hello12@'
st.islower()
>>> True

st = 'Hello'
st.islower()
>>> False.

5. isnumeric()

isnumeric() is a function which is used to check whether the given string is having only numeric char or not

→ It will written the result as True it all the char has number.

Syntax: `var.isnumeric()`

Eg: n = '1234'

var.isnumeric() n.isnumeric()

>>> True.

n = '123.45'

~~var~~ n.isnumeric()

>>> False

6. isalnum()

It's a function which will written the result as true if the given string is having only alpha charaters or only ~~having~~ combination of alpha & numeric character.

syntax: `var.isalnum()`

st = 'abc123'
st.isalnum()

>>> True

st = 'abc'
st.isalnum()

>>> True

st = '123'
st.isnum()

>>> True

st = 'abc12@'
st.isalnum()

>>> False

7. isspace()

It's a function which will written the result as True if all the characters present inside the string or of spaces else it will written False.

syntax: `Var.isspace()`

Eg:

```
st = ' '                st = ' g'
var.isspace()           var.isspace()
>>> True                >>> False
```

8. Title() / capitilize() → first letter will be the capital remaining are lowercase

It's a function which is used to convert the first character of Each & Everyword present inside the given string into uppercase character. / in title

syntax: `Var.title()` `Varname/val.capitilize()`

Eg:

```
st = 'features of python'
var.title() / capitilize()
>>> 'Features Of Python'
```

```
st = 'features of python'
var.title() / capitilize()
>> 'Features Of Python'
```

eg def ghi jkl
Def Ghi JKL

9. istitle()

istitle() will written the result as true if the given string is in the form of title or not

syntax: st. `var.istitle()`

Eg:

```
st = 'Feature Of Python'
st.istitle()
>>> True

st = 'hai hello'
st.istitle()
>>> False
```

continue:

It is a keyword which is used skip / stop the current Execution to make the control to go for the next iteration

1. Prgm to skip all the odd numbers btw the range are do

```
for i in range(1, 21):
    if i%.21 ==0:
        continue
    Print(i)
```

2. Write a Prgm to skip all the Even num between the range 1 to 50

```
for i in range(1, 51):
    if i%.2 ==0:
        continue
    Print(i)
```

3. Write a Prgm to Extract all the integer no of present inside an heterogeous list collection.

```
l = [2, 2.3, 3, 3+1j, 4, 5.7]
res = []
For i in l:
    if type(i)!= int:
        continue
        rest = [i]
    rest += [i]
    Print(i)
```

4. Write a Pgm to Extract all the values present inside the given string

```
st = input('Enter the string')
out = ''
for i in st :
    if i not in 'AEIOU aeiou' :
        continue
    rest = i
Print(i) Prin'
```

5. Write a Pgm to print all the float numbers present inside given tuple.

```
a = (2, 2.3, 4, 4.8, 3+2j, 5.3)
b = ()
for i in a :
    if type(i) != float :
        continue
    b += i
Print(i)
```

6. Write a Pgm to Extract all the set data Present inside an input heterogniOus list collection.

```
l = [1, 2, {1, 2, 3, 4, 5}]
b = []
for i in l :
    if tuple(i) != set
        continue
    bt = [i]
Print(i)
```

7. WAP to Extract find the cube of all the odd num Print btw range (1 to 10) and save the result into an output tuple.

```
out = ()
for i in range (0, 11):
    if i%.2 == 0:
        continue
    out += (i**3)
Print (out)
```

PASS:

Pass is a keyword which is used to keep any Empty statement block as valid are block.

Eg: for i in 'hai':
    Pass

Functions :

Functions are the named memory block where the instructions stored.

→ Their are two types of functions.

1. In-Built function
2. user defined functions.

→ When Ever we want to Execute some task irrespective of its Order we will make use of functions.

1. In-Built functions :

These are the functions whose task is pre-defined by the developer, whenEver we want to access them we can Easly access but we can't modify the Original meaning inbuilt functions.

Eg : Len( ) - It is used get the length of the collection
Print( ) - It is used to print the output

2. user defined functions :

These are the functions which will get created based on the users requirment

def (definition) is used to define / create a function
Arguments are based on the user requirments

syntax :

```
def frame (args) :
    SB
```

(Tab)

return values

frame (args)

Where def : It is a keyword which is use to define the fun (or) which is use to create a the function

frame : It is a name given to the set of instruction to identify the Operation

arguments: These are the requirments to be pass to perform the required Operation.

Return : It is a keyword which is used to return the control from functions to outside the function.

Note:
Here, arguments and return values are not mandatory.
In case of fun Passing args & returning values is not mandatory

Function call:
It is a statement which is use to Execute the created function.

Based on the arguments and return value user define func got classified into (4) four diff types.

1. function without argument and without Return value.
2. function with argument and without return value.
3. function without argument and with return value.
4. function with argument and with Return value.

1. Function without argument and without Return value:
It is a type of function where it is not requir to pass arguments to the function and user will not Except any values in return

=> This type of fun will be used in root func

Eg: main & void func in c language.

```
def success()
    Print('Transaction successful')
def failure()
    Print('Transaction failure')
def main()
    pw = True
    if pw == True:
        success()
    else:
        failure()
main()
```

The Process of fun creation and Execution.

In case of function the memory will get segrigated in to two diff

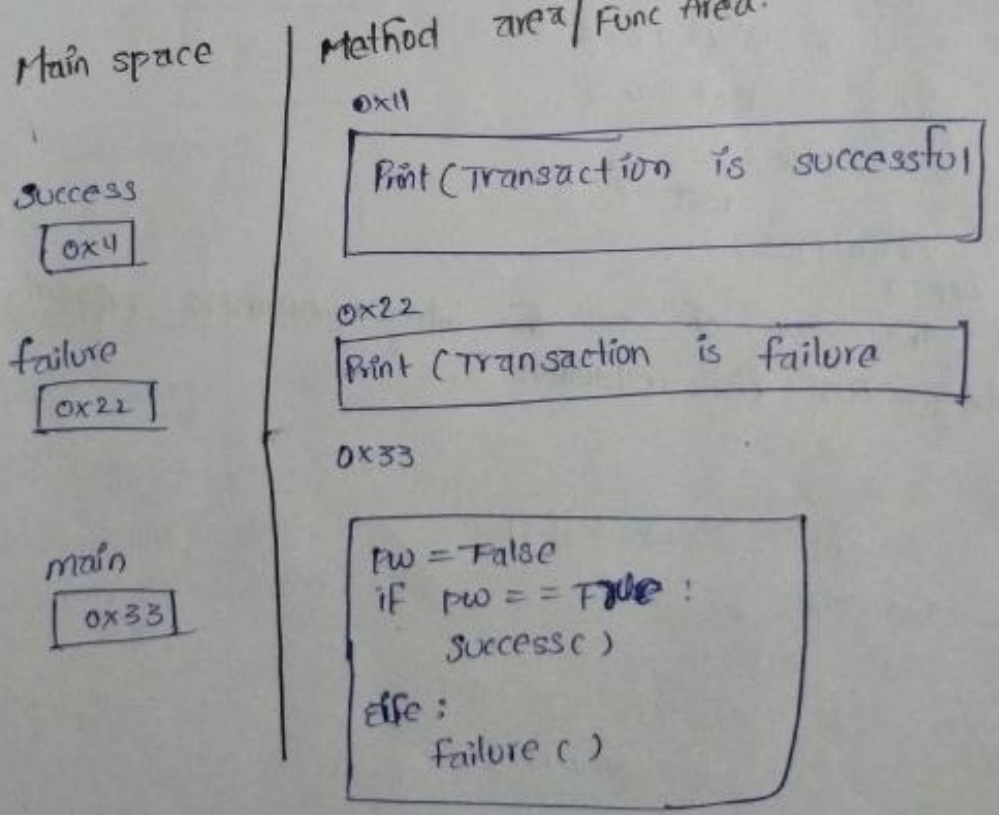
1. Main space
2. Method area / function area.

→ always controls will starts its Execution from main space.

→ as soon as control see a keyword called def it will go to method area and create a block of memory.

→ control will store all the instructions present inside the ~~contact~~ fun in to the created block space address will be given and that will get stored into main with func name.

→ as soon as we call the function the instruction which are stored inside the function it will get Executed.

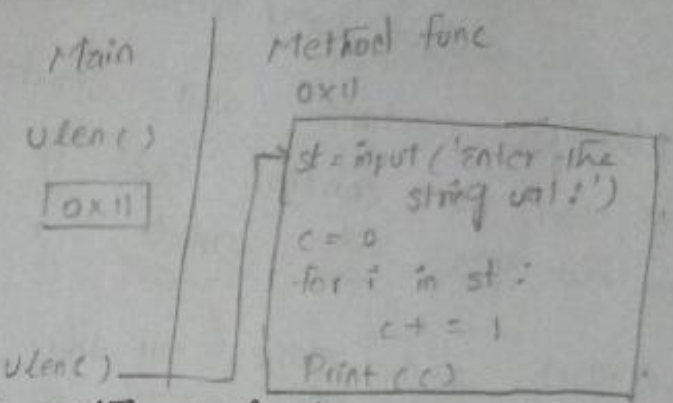


Note:

if we try to call the function before its creation control is going to throw an Error.

WAP to find the len() of the collection with out using len().

```
def ulen()
    st = input('Enter the string val:')
    c = 0
    for i in st:
        c += 1
    Print(c)
ulen()
```



WAP to replace all the uppercase alphabet with 0 inside the given string.

```
def upp()
    st = input('Enter the string:')
    res = ''
    for i in st:
        if 'A' <= i <= 'Z':
            res += 0
        else:
            rest = i
    Print(res)
upp()
```

WAP to find the sum of all the integer numbers present inside an hetrogeneous list collection.

```
def sum()
    l = [1, 2, 3.2, 6, 3+2j, 5]
    Sum = 0
    for i in l:
        if type(i) == int:
            Sum = sum + i[i]
    Print(Sum)
sum()
```

2. Function with argument and without Return Value :- setter() display()

It is a type of user-defined function, as a name tha its self idicates that it is require to pass arguments to the function but the user will not Expect any values in return

→ These kind of functions will be used in setter() function & display() function

setter() :

It is a function which is use to set the values

display()

It is a function which is use to display the values

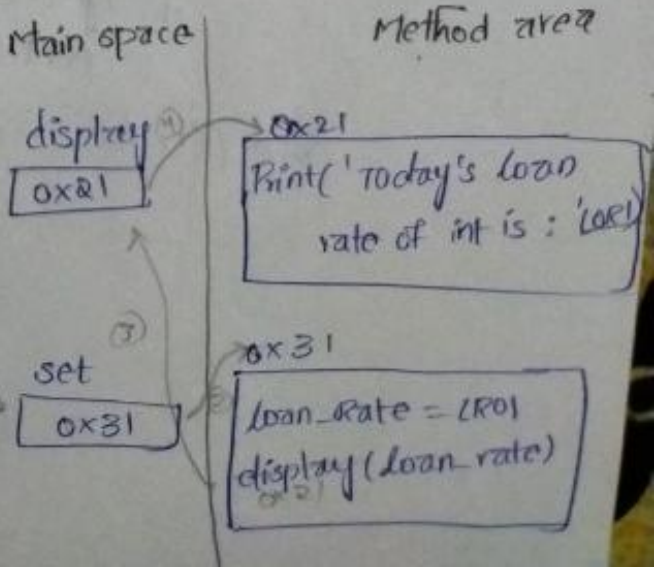
Syntax :

```
def fname (args)
    SB
fname(args)
```

Tab

Eg:

```
def display (LROI) :
    Print ('Today's Loan rate of intrest is :', LOR I)
def set (LROI) :
    loan_Rate = LROI
    display (loan_rate)
set (8)
```



WAP to Print all the float numbers present inside a tuple collection.

```
def disp(tu):
    for i in tu:
        if type(i)==float:
            Print(i)
disp(1,3.4,5.2,8,9,6.7)
```

```
a=(1,3.4,5.2,8,9,6.7)
disp(a)
```

WAP to Extract all the string present inside the list collection which are having more than 5 characters in it.

```
def disp(l):
    res=[]
    for i in l:
        if type(i)==st and len(i)>5:
            gest+=[i]
    ← Print(res)
disp(['hello',23,'hai',12])
```

WAP to find the cube of all the integer values present inside an input tuple collection and print it on the screen

```
def cube(a):
    for i in a:
        if type(i)==int:
            Print(i**3)
cube((1,'hai',3,2.3,5))
```

3. Function without argument with return value :- getter()

This is the type of user-define function where its not require Pass arguments to the function but users will expect some values in return.

→ This kind of functions will be used in getter()

getter() :

It is a function which is use to get the input from the user and to return those values to outside of the function

→ We can return n-values

Syntax:

```
def fname():
    SB
    return val1, val2 --- valn
```

(1 tab)

res.var = fname() → we have to use variable to store the return values because return values itself doesn't have variable to display the output

or

Print(fname()) → if we no need to use var at that time we can directly print statement by using fn name.

Eg:

```
def sam():
    a = int(input('Enter the val :'))
    b = int(input('Enter the val :'))
    return a,b, a+b
var = sam()
Print(var)
```

Note:

If we returning n-no: of values from the function then we should make use of n no: of variables to store those values. (OR) we can make use single variable to store all the multiple values in the form of tuple.

WAP to find the sum and difference between two values.

```
def st():
    a = int(input('Enter the value:'))
    b = int(input('Enter the value:'))
return a+b, a-b
add, diff = st()
Print('sum is:', add)
Print('diff is:', diff)
```

o/p:
10
20
sum is: 30
diff is: -10

WAP to Extract all the Even numbers present inside a given string.

```
def Extract():
    st = input('Enter the string:')
    Out = ''
    for i in st:
        if '0'<=i<='9' and int(i)%2==0:
            Out = Out + i
return Out
Var = Extract()
Print(Var)
```

* if the numbers are present in string collection at that we have to check even numbers present or not not possible to do operation even if not we have type cast the string number to integer

WAP to calculate the sum of all the floating values present inside the given tuple.

```
def tup():
    a = (1, 2.3, 5, 7.2, 8+3j, 'hai')
    sum = 0
    for i in a:
        if type(i) == float:
            sum = sum + i
    return sum
res = tup()
Print(res)
```

WAP to find or Extract by finding the cube of all the odd numbers present at Even position in a given tuple if len of the tuple is more than 1 Equals to 5

```
def sam():
    tu = (7, 28, 43, 29, 420, 69)
    lies = ()
    if len(t) >= 5:
        for i in range(0, len(tu)):
            if i%.2 == 0 and tu[i]%.2 != 0:
                lies = lies + (tu[i] ** 3,)
    else:
        Print('Len of a given collection is less than 5')
    return out
Print(sam())
```

4. function with argument and with Return value:

It is a type of user defined function where it is require pass the arguments to the function, and user will Expect some values in return

-> This type of fonction we will be used in all the real time applications

Syntax:

```
def fname(args):
    ↔ 1 tab
        SB
    return values
var = fname(args)
    (or)
Print(fname(args))
```

Eg: WAP to create four diff functions to perform addition, sub, multiplication & division operation respectively.

```
def add(a,b):
    return a+b
def sub(a,b):
    return a-b
def mul(a,b):
    return a*b
def div(a,b):
    return a/b
Print(add(12,2))
```

```
def OP(a,b):
    a = int(input('Enter the num'))
    b = int(input('Enter the num'))
    return a+b, a-b, a*b, a/b
Print(OP(10,20))
```

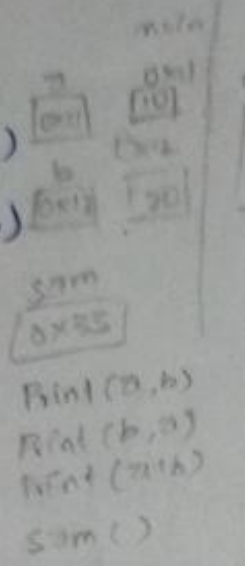
WAP to create a basic calculator with the help of any statement

```
def add(a,b):
    return a+b
def sub(a,b):
    return a-b
def mul(a,b):
    return a*b
def div(a,b):
    return a/b
def get():
    a = int(input('Enter the value :'))
    b = int(input('Enter the value :'))
    return a,b
while True
    Print('Enter 1 for add, 2 for sub, 3 for mul, 4 for div and
                                        9 to exit)
    ch = int(input('Enter the choice :'))
    if ch=1:
        a,b = get()
        Print('sum is :', add(a,b))
    elif ch=2:
        a,b = get()
        Print('diff is :', sub(a,b))
    elif ch=3:
        a,b = get()
        Print('prod is :' mul(a,b))
    elif ch==4:
        a,b = get()
        Print('div is :' div(a,b))
    elif ch==9:
        break
    else:
        Print('Enter valid num')
```

Global Variable, local Variables and non-local Keyword:

Eg:

```
a,b = 10,20
def sam():
    Print(a,b)
    Print(a+b)
Print(a,b)
Print(b,a)
Print(a+b)
sam()
```

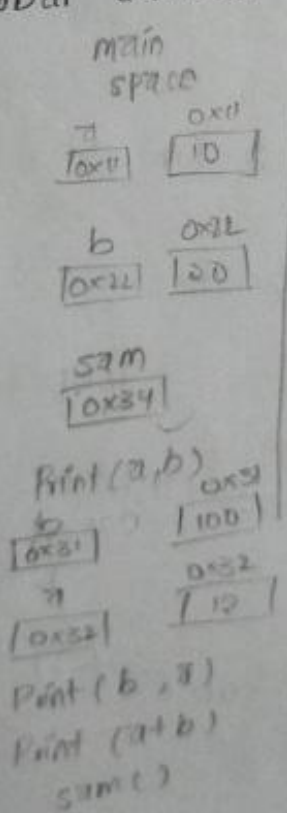


10, 20
20, 10
30
10 20
30

first main method will Execute after func method will Execute

Eg:

```
a,b = 10,20 # global variable.
def sam():
    Print(a,b)
    Print(a+b)
    a=50
    Print(a,b)
Print(a,b)
b=100
a=10
Print(b,a)
Print(a+b)
sam()
```



o/p:
10 20
100 10
110
10 20
30
50, 10

Global Variable:

These are the variables which will get created inside main space.

→ We can access and modify global variables inside main space

→ we can access global variables inside method area as well but we can't modify the value of global variable inside method area.

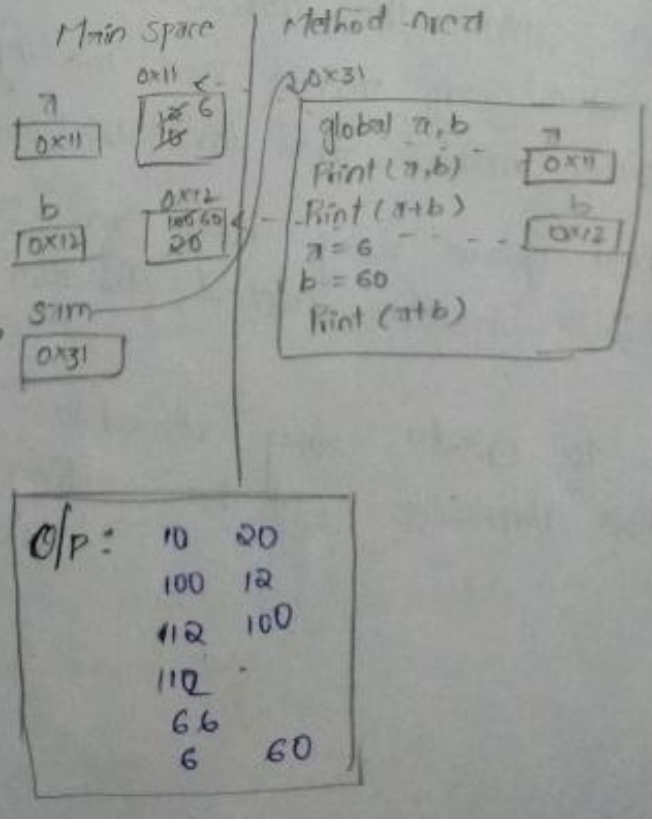
→ whenever it is require to modify the values of global variab inside method area then we will make use of a keyword called global and we need to specify the variables to which we are going to do the modification.

Syntax:

```
global var1, var2 ---- varn
```

→ If we want to used global keyword inside method area then that should be the first instruction of the function

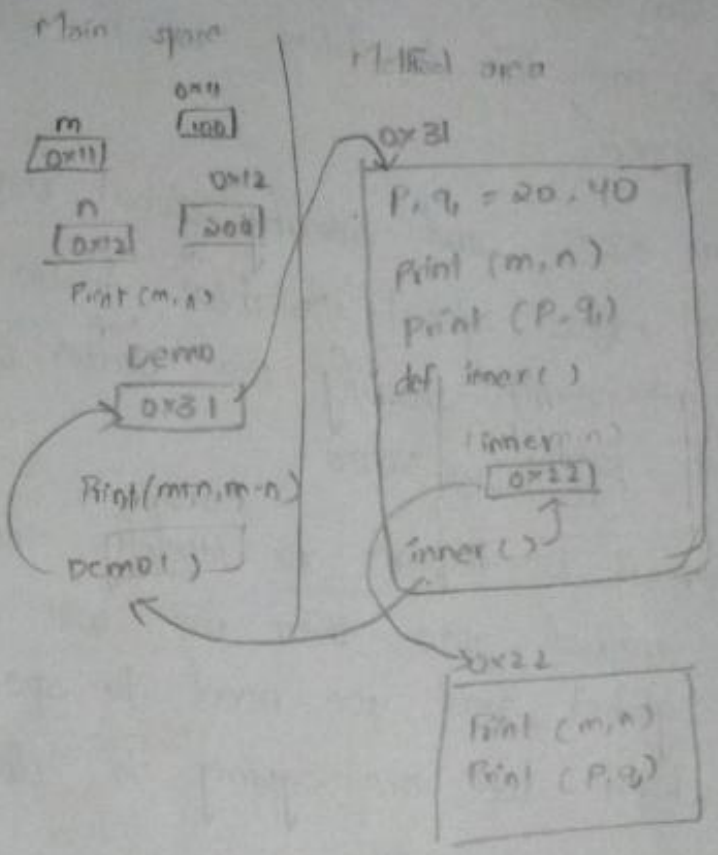
```
a,b = 10,20
def samc)
    global a,b
    Print(a,b)
    Print(a+b)
    a = 6
    b = 60
    Print(a+b)
Print(a,b)
b = 100
a = 12
Print(b,a)
sam()
Print(a,b)
```



~~Local Variable~~: Local Variable:

Eg:

```
m, n = 100, 200
Print (m, n)
def demo():
    P, q = 20, 40
    Print (m, n)
    Print (P, q)
    def inner():
        Print (m, n)
        Print (P, q)
    inner()
Print (m+n, n-m)
demo()
```



o/p:-
100 200
300 100
100 200
20 40
100 200
20 40

Local variables are the variables which will get created in method area we can access and modify the value of local variables inside method area (or) the same function.

→ In the same ~~place~~ way we can access and modify the local variables inside nested function.

→ if we try to create any variables inside nested function then those variables will be acting as non-local variables

```
m, n = 100, 200 # global variables
Print (m, n)
def Demo():
    P, q = 20, 40 # local variables
    Print(m, n)
    Print(P, q)
    P = 1000
    Print (P)
    def inner():
        c, d = 34, 89 # non-local variables.
        Print (c, d)
        Print (m, n)
        Print (P, q)
    inner()
Print(m+n, m-m)
Demo()
```

Note:
Whenever we want to modify the local variable inside the nested function we were making use of keyword called nonlocal in old version of IDLE software.

Formal and actual parameters:

Formal arguments:
These are the arguments which will get created while defining a function.
(or)
These are the variables which will be present in receiving End.

Actual parameters:
These are the parameters which will be present in function call (or) which will be present in sending End.

Eg:
```
def sam(a,b): # formal args
    Print (a,b)
sam(3,4)# actual args
```

WAP (or) create a function to find the total number special characters present in a given string.

```
def special():
    st = input('Enter the string:')
    out = 0
    for i in st:
        if not ('a'<= i <='z' or 'A'<= i <='Z' or '0'<= i <='9'):
            out = out + i
            Print(out)
special()
```

(or)

```
def special(st):
    out = 0
    for i in special:
        if not ('a'<= i <='z' or 'A'<= i <='Z' or '0'<= i <='
            out = out + i
        Print(out)
special(Rupa@/sree)
```

(or)

```
def special(st):
```

WAP to create a fun to convert all the uppercase to lowercase alphabet and lower case alpha to uppercase without changing numbers and special characters present in given string.

```
def Lower(st):
    out = ''
    for i in st:
        if ('A'<= i <='Z'):
            out = out + chr(ord[i]+32)
            Print(out)
        elif ('a'<= i <='z'):
            out = out + chr(ord - 32)
            Print(out)
        else:
            out = out+i
            Print(out)
Lower('hai@ HELLO/')
```

Passing Default values to the parameter :

To Execute any function, the no: of formal arguments should be Equals to the no: of actual arguments else control will through an Error

→ To over come the above problem we are going to make use of a concept called passing values to the parament

→ In this concept we are going to pass the mandatory arguments as it is and non-mandatory arguments with its default value.

Syntax :

Mandatory Non-mandatory

```
def fname (a1, a2 --- an, k1 = v1, ---- kn = vn) :
    SB
fname (v1, v2 --- vn)
```

Note :

Whenever we don't want to follow the order of optional argument in actual argument section then we need to pass that value as [Key name = value] or (Var_name = value)

```
def Reg (name, phno, Email, allphno = None, allemail = None) :
    Print ('name is' :, name)
    Print ('phno is' :, phno)
    Print ('Email is :', Email)
    if allphno != None :
        Print ('allphno is :', allphno)
    if allemail != None :
        Print ('allEmail is :', allEmail)
Reg ('Rupa', 9949232420, 'R@gmail.com')
Print '_' * 50
Reg ('sree', 9182364151, 's@gmail.com', 9121212121)
Print '_' * 50
Reg ('Harsha', 9507645521, 'h@gmail.com', altemail = 'sh@gmail.com')
```

WAP to find the sum of min 3 numbers or max 8 numbers. with the help of passing default values

```
def add(a,b, c,d=0, e=0,f=0, g=0, h=0)
    return( a+b+c+d+e+f+g+h
Print(add(10,20,30))
Print(add(10,20,30,60,705))
```

WAP to find the product of min 2 numbers and max 5 numbers.

```
def Prod(a,b, c=1, d=1, e=1) :
    return a*b*c*d*e
Print (Prod(10,20)
Print (Prod(10,5,6,7)
```

WAP to Print all the numbers Present inside the string between starting and Ending limit.

```
def string(st, si=0, End=None) :
    if Ei==None :
        ei=len(st)
    for i in range(si, Ei) :
        if '0'<=st[i]<='9' :
            Print(st[i])
string('hai123hello9023', 2, 7)
string('hai123hello9023')
```

WAP to Extract all the integer values Present inside the list collection btw the specified starting index and Ending index.

```
def Eiv(a, si=0, Ei=None) :
    if Ei==None :
        Ei=len(a)
    Out=[]
    for i in range(si, Ei) :
        if type(a[i])==int :
            Out+=[a[i]]
    return Out
Print(Eiv([10,'hai',39,3.6,45,'hello']))
```

~~special character~~

WAP to Print all the ~~numbers~~ even number present inside the string between the specified starting and ending index.

```
def string(st, si=0, ei=None)
    if ei==None:
        ei=len(st)
    for i in range(si,ei):
        if 'a'<=st[i]<='z' and int(st[i])%2==0:
            print(i)
string('hi this is 12345 62', 'hello', 5, 6, 10)
```

Packing:

Packing is a finamizan gouping an individual data items in the form of collection to keep them securly

→ user can do packing in all the five collection data type, but the system will do packing the form of 'tuple' (because tuple is most secured data type that we have in python).

→ To make the system to pack the all the individual data items in the form of tuple '*' operator is used.

```
val1, val2 --- valn = *var
```

By using * operator it will acts as a tuple data type to store the values

Syntax:

```
def fname(*var)
```

SB

```
fname(val1, val2 ---- valn)
```

Eg:

```
def sam(*a):
    Print(a)
    Print(type(a))
sam(1,2,3,4,5)
sam(2)
```

o/p: (1,2,3,4,5)
<class tuple>
(2,)
<class tuple>

Whenever we want to pass the data in the form key = value then we will make use of '**' to store the data in the form of dictionary (because we have only single data type in Python to store key and values.

Syntax:

```
def fname(**var):
    SB
```

fname(K1=V1, K2=V2 --- Kn=Vn) → key should either alphabet (or) group of alphabet

where key should be of an alphabet (or) group of alphabet

Eg: ~~def sam(**a):~~

~~SB~~

```
def sam(**a):
    Print(a)
    Print(type(a))
sam(abcd=10, b=2, c=3)
```

o/p:

WAP to print the string data item present inside the tuple the len of that string is more than or equals to 3

```
def string(*st):
    for i in a:
        if type(i)==str and len(i)>=3:
            Print(i)
string(12,'hello','bye',80,'hai')
```

WAP to Extract all the collection data items present inside the tuple with the help of packing concept.

```
def sam(*a):
    out=()
    for i in a:
        if type(i) in [list, str, tuple, set, dict]:
            out+=(i,)
    Print(Out)
sam('hai',[1,2],'ok bye', 11, 2.3, 2+3j, {'a':1})
```

WAP to Extract all the list data items present at even index if length of that list is odd.

```
def Ext(*a):
    out=()
    for i in range(0, len(a)):
        if type(i)%2==0 and type(a[i])==list and
                len(a[i])%2!=0:
            out+=(a[i])
    Print(Out)
Ext('hai', [1,2], 2, [3,4,5], 'hello')
```

WAP to Extract the key values from the dictionary only if the len of key is more than or Equals to key.

```
def dict(**d):
    out={}
    for i in d:
        if len(i)>=3:
            out[i] = d[i]
    Print(Out)
dict('a':1, 'abc':2, 'bc':3, 'hai':4)
```

WAP to Extract the value key & value pair from the dictionary only if the value is of integer data.

```
def dict(**d):
    out = {}
    for i in d:
        if type(d[i]) == int:
            out[i] = d[i]
    Print(out)
dict('a':103, 'b':'hai', 'c':5.2, 'd':10)
```

WAP to Extract key value pairs from the dictionary only if the value is of collection data-type.

```
def dict(**d):
    out = {}
    for i in d:
        if type(d[i]) in [list, str, tuple, set, dict]:
            out[i] = d[i]
    Print(out)
dict('a':(1,2), 'b':[2,3], 'c':2, 'd':'80,'o')
```

Unpacking:

It is a fenamenan of dividing the collection into an individual data items as an independent values.

→ In unpacking the no: of formal arguments should be equal to the length of collection which is present at actual argument section.

→ unpacking can be applied for all the five types of collection data types.

Syntax:

```
def fname (var1, var2 --- varn)
    <-> 1 tab    SB

fname (* col)
```

Eg: 1. def sam(a,b,c):
        Print (a,b,c)
    sam (* 'hai')

2. def sam (a,b):
        Print (a,b)
    sam (* {'m':1, 'n':23)
    sam (* {'m':1, 'n':23}.values())
    sam (* {'m':1, 'n':23}.items())

Variable arguments :

If we want to pass individual values (or) positional argum in the form of tuple collection will make use of packing (*)

→ If we want to pass key and value place store in the form of dictionary and we will make use of packing (**).

→ If we want to pass the combination of individual values and key values pair then we will make use of concept called var arguments.

Syntax :

```
def fname (* args, ** kwargs)
↔ 1 tab
        SB
```

fname ($v_1, v_2 \cdots v_n$, $k_1 = val_1, k_2 = val_2, \cdots k_n = val_n$)

Eg:

```
def sam(* args, ** kwargs)
    Print (args)
    Print (type (args))
    Print (kwargs)
    Print (type (kwargs))
sam (10, 20, 30, a = 1, b = 2, c = 3)
```

Recursion :

It is a finamion of calling the function by its self until the given termination condition is true.

syntax :

With return val

```
def fname(args):
    if <term cond>:
        return val
    ==========
    return fname(args)
Print(fname(args))
```

Without return val

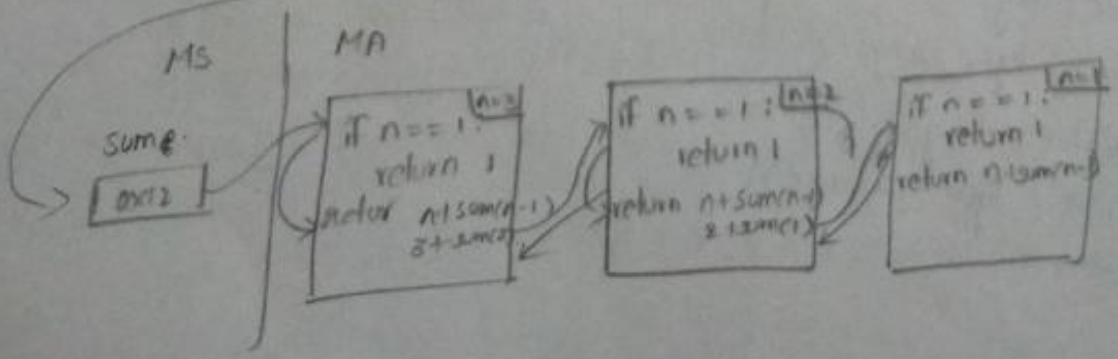
```
def fname(args):
    if <term cond>:
        return
    ==========
    fname(args)
fname(args)
```

Eg:
Wap to find the sum of n-natural numbers. with the help of recursion.

logic

sum(1) ⇒ 1
sum(2) ⇒ 2+1
2+sum(1)
sum(3) ⇒ 3+sum(2)
sum(4) ⇒ 4+sum(3)
⋮
sum(n) ⇒ n+sum(n-1)

```
def sum(n):
    if n==1:
        return 1
    return n+sum(n-1)
Print(sum(3))
```



WAP to find factorial of a given number.

```
def fact(f):
    if f==1 or f==0:
        return 1
    return n * fact(n-1)
Print fact(5)
```

$5 * 4 * 3 * 2 * 1$

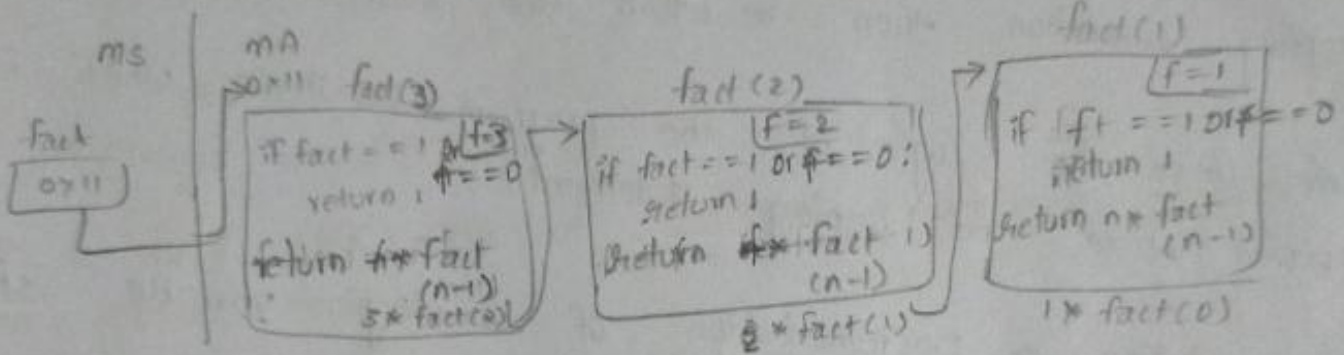
fact(0) => 1

fact(1) => 1

fact(2) => 2 * fact(1)

fact(3) => 3 * fact(2)

fact(n) => n * fact(n-1)



WAP to print following output with the help of recursion 9 to 0

```
def Out(n):
    if n==-1:
        return
    Print(n, End=' ')
    out(n-1)
Out(9)
```

WAP to reverse a given string with the help of recursion

```
st = input('Enter the string:')
rev = ''
i = 0
while i < len(st):
    rev = st[i] + rev
    i += 1
Print(rev
```

```
def rev_st(st, rev='', i=0):
    if i >= len(st):
        return rev
    rev = st[i] + rev
    return rev_st(st, rev, i+=1)
Print(rev_st('hello'))
```

Steps to convert any looping Program in the form of Recursion.

- Intialization of all the looping variables should be done while by creating formal argument

Write down the termination condition Exactly opposite to looping condition, then we return the total result

Write down the logic of the program as it is (which is present inside the looping statement)

Incrementation or Decrementation of looping variable should be done recursive call.

P to Extract all the Even numbers present insid an mogeneous list collect.

```
= [          ]
ut = []
= 0
while i< len(l):
    if l[i]%2==0:
        Out += l[i]
    i+=1
Print(Out)
```

```
def Ext(l, Out=[], i=0):
    if i>=len(l):
        return Out
    if l[i]%2==0:
        Out+=[l[i]]
    return Ext(l, Out, i+1)
Print(Ext([2, 10, 18, 15, 9, 7]))
```

P to Print all the special characters present inside the string

```
st = input()
es = ''
i = 0
while i< len(st):
    if not('a'<= st[i]<='z' or 'A'<= st[i]<='Z' or '0'<= st[i]<='9'):
        res = res + st[i]
    i+=1
```

```
def special(st, i=0):
    if i >= len(st):
        return
    if not('A'<=st[i]<='Z' or 'a'<=st[i]<='z' or '0'<=st[i]<='9'):
        print(st[i])
    special(st, i+1)
sp('Hello World*')
```

WAP to find the sum of all the numbers present inside the given string.

```
st = 'I0 hello 3'
sum = 0
i = 0
while i<len(st):
    if '0'<=st[i]<='9':
        sum += int(st[i])
    i += 1
print(sum)
```

```
def num(st, sum=0, i=0)
    if i >= len(st):
        return sum
    if '0'<=st[i]<='9':
        sum += int(st[i])
    return num(st, sum, i+1)
print(num('I0 hello 3'))
```

WAP to seperate all the types of characters present inside the string into n different output collections.

```
st = input()
up = ''
low = ''
num = ''
sp = ''
i = 0
while i<len(st):
    if 'A'<=st[i]<='Z':
        up += st[i]
    elif 'a'<=st[i]<='z':
        low += st[i]
    elif '0'<=st[i]<='9':
        num += st[i]
    else:
        sp += st[i]
    i += 1
print(up)
print(low)
print(num)
print(sp)
```

```
def str(st, up='', low='', num='', sp='', i=0):
    if i >= len(st):
        return up, low, num, sp
    if 'A'<=st[i]<='Z':
        up += st[i]
    elif 'a'<=st[i]<='z':
        low += st[i]
    elif '0'<=st[i]<='9':
        num += st[i]
    else:
        sp += st[i]
    return str(st, up, low, num, sp, i+1)
```

11. Startswith():

This is the string inbuilt function it is use to check wheather the given string is starts with the user given input string if it is true it will return true, else it will return false.

Syntax:

Var.startswith('input string') / var.startswith('string', ind

Eg: var.startswith('input string') / var.startswith('string', start inde index

a = 'Test yentra'

a.startswith('Te')

→ True

a.startswith('ye')

→ False.

b = 'hii hello how are you rahul'

b.startswith('h', 5)

→ false

b.startswith('el', 5)

→ True.

b.startswith('el', 5, 10)

→ True

12. Endswith():

Endswith() is a string inbuilt function it is use to check wheather the given string is Endswith the user given input string, if it is true it will return true, else it will return false.

Syntax:

var.Endswith('string') / var.Endswith('string', index) / var.Endswith('string', start indx, index+1)

Eg: a = 'hii hello how are you rahul'

a.Endswith('ul')

→ true

a.Endswith('ul', 6)

→ true

a.Endswith('ul', 3, 8)

→ False.

13. index() :

It is a inbuilt function which will return you index of given string. this function is return the first occurence of letter in a string.

syntax :

var.index(string)/var.index(string, index)/
var.index(string, sindex, Endindex+1)

Eg: b = 'hii hello how are you'

b.index('i')

→ 6

b.index('z')

→ Error. → z is not present in given string

b.index('ei')

→ 5

b.index('h', 3)

→ 4

14. find() :

The function will return index value of given substring if the user string is present then it will given index number, if the user given string is not present in the input string then it will give the index of '-1'

syntax :

var/val.find('substring')

var/val.find('substring', SI)

var/val.find('substring', SI, EI+1)

Eg: a = 'hii hello'

a.find('i')

→ 1

a.find('z')

→ -1

a.find('') → Empty string.

→ 0

15. split()

split() is a inboilt -function and it will split the given string (Each and Every word) in to list of words it will split automaticrilly if there any space is present

Syntax:

```
var/val . split()
var/val . split (substring, no: of splits)
```

Eg:

```
a = 'hii hello'
a . split()
⇒ ['hii', 'hello']

c = 'abc'
c . split()
⇒ ['abc']

c . split ('b')
⇒ ['a', 'c']

d = 'let me give big string'
d . split ('i')
⇒ ['let me g', 've b', 'g str', 'ng']

d . split ('i', 1)
⇒ ['let me g', 've big string']
```

16. count()

count() is a inbuilt function and it is use to count the no: of occurances of substring in a given string

syntax:

```
var/val . count('substring')
var/val . count ('substring', sI)
var/val . count ('substring', sI, EI+1)
```

Eg:

```
a = 'any string'
a . count ('a')
→ 1
a . count (' ')
→ 1

a . count ('n')
→ 2
a . count ('n', 2) → position of index
→ 1

a . count ('n', 0, -1)
→ 2
```

17. Replace()

Replace() is a inbuilt function which will use to replace the specific substring to the given another specific substring.

Syntax:

```
var/val . replace ('substring', 'new string')
var/val . replace(substring, replacing string no of replacement)
```

Eg:

```
a = 'Qspiders'
a.replace ('Q', 'Py')
→ 'Pyspiders'

b = 'qspiders'
b.replace ('s', 'n')
→ 'qnpidern'

c = 'qspideresq'
c.replace ('q', 'pq', 1)
→ 'pqspideresq'
```

18. join()

join() is a inbuilt function which is use to join the list of string.

Syntax:

```
cluster . join(var/val)
```

Eg:

```
a = 'hii hello how are you'
b = a.split()
b
→ ['hii', 'hello', 'how', 'are', 'you']

''.join(b)
→ 'hiihellohowareyou'

" ".join(b)
→ 'hii hello how are you'

'@'.join(b)
→ 'hii@hello@how@are@you'

a = ['hii', 6, 'hello', 'how', 'are', 'you']
''.join(a)
→ Error
```

List in buit -function :

1. append() :

append() is use to add the element in to list at the last, append is a list inbuilt function.

syntax :

var/val . append(element)

Eg: a = [1, 2, 3]

a . append ('rajo')

→ [1, 2, 3, 'rajo']

a . append ([1, 2, 3, 4])

⟹ [1, 2, 3, 'rajo', [1, 2, 3, 4]]

2. Insert() :

Insert is a list in built function which is used to insert (or) add the new elements into the list at specific index.

syntax :

var/val . insert (index, element)

Eg: a = [1, 2, 3, 'raju', [1, 2, 3, 4]]

a . insert (1, 'hii')

→ [1, 'hii', 2, 3, 'raju', [1, 2, 3, 4]]

a . insert (0, '2.3')

→ [2.3, 1, 'hii', 2, 3, 'raju', [1, 2, 3, 4]]

3. count():

count is a inbuilt function which will count the no of Occurances of Element in a list (given list)

Syntax:

Var/val . count (Element)

Eg:

```
a=[1,'hii',2,3,'raju',[1,2,3,4]]
a.count(1)
⇒ 1
a.coont([1])
⇒ 0
a[6].count(1)
⇒ 1
```

4. Index()

It is a inbuilt function which will return the index of a given element in a given list

Syntax:

Var/val . index (Element)

Eg:

```
a=[1,'hii',2,3,'raju',[1,2,3,4]]
a.index('hii')
⇒ 1
a.index('raju')
→ 4.
```

5. Remove ( )

Remove ( ) is a inbuilt function which is used to remove a particular element in a given list.

syntax :

```
Var/val . remove( element )
```

Remove inbuilt function it will take only one argument to remove . If in the list we have duplicate value/ elements it will remove the first accurence of the value.

Eg :

```
a = [1, 2, 3, 4, 5]          b = [1, 2, 3, 4, 3, 5]
a . remove (3)               b . remove (3)
=> a                         => [1, 2, 4, 3, 5]
=> [1, 2, 4, 5]
```

6. Pop :

It is a list inbuilt function , it is used to remove the last element from the given list

syntax :

```
Var/val . pop( )
```

Eg :

```
a = [1, 2, 4, 5 , 3]        b = a . pop( )
a . pop( )                  b
=> 3                        => 5
a = [1, 2, 4, 5]
a . pop( )
=> 5
```

7. clear() :

clear() is a list in built function it will remove all the elements in the list

syntax :

Var/val . clear()

Eg:

```
a = [1, 2, 3, 4]
a.clear()
--> []
```

8. sort() :

sort() is a list in built function it is used to sort the element in the form of accending order based on their ascii value.

syntax :

Var/val = sort()

when we are sorting the list elements the list elements should be homogeneous / same data types.

Eg:

```
a = ['hai', 'are', 15, 'you']
a.sort()
--> Error

a = [0123, 123]
a.sort()
--> Error

b = [12, 0, 1, 2, 3, 4, 5]
b.sort()
--> [0, 1, 2, 3, 4, 5, 12]
```

```
a = ['raja', 'rani', 'are']
a.sort()
b = a
b
--> ['are', 'raja', 'rani']

c = ['01', '005']
c.sort()
c
--> ['005', '01']
```

9. Extend():

Extend() is a list inbuilt function it is used to Extend the given ~~argument~~ / string Each End Every word into the List

syntax:

. val/var . Extend (~~Element~~/'string')

Eg:

a = ['1','2',3']

a. Extend ('hai')

⇒ [1, 2, 3, 'h', 'a', 'i']

Tuple inbuilt functions:

1. count()
2. Index()

1. count():

count() is a tuple inbuilt function which will get the no: of accurence of element present in the tuple collection.

Syntax:

val/var . count (Element)

Eg: a = (1, 2, 3, 4, 2, 4,2)

a. count (2)

a

→ 3

2. Index():

index() is a tuple inbuilt function which will return the index of the element in the given tuple collection

syntax:

```
var/val . index ( element )
```

Eg:

```
a = (1, 2, 3)
a . index (2)
a
→ 1
```

Set in built functions:

1. add():

add() is a inbuilt function which will add elements inside the set which are immutable data types.

syntax:

```
var/val . add (Elements)
```

Eg:

```
a = {1, 23}
a
⇒ {1, 2, 3}
a . add ('hii')
⇒ {1, 'hii', 23}
a . add (55)
⇒ {1, 'hii', 55, 23}
a . add (C1, 2])
⇒ Error
```

```
a . add ((1, 2))
⇒ {1, 'hii', 23, (1, 2), 55)
a . add ({'a' : 2})
⇒ Error
```

2. Remove ():

remove () is a inbuilt function which will remove the particular element present in the given set collection. It will give error if the specific element is not present in the set collection.

Syntax:

var/val . remove (element)

Eg:

a = {1, 2, 3, 3, 4}

a . remove (3)

⇒ {1, 2, 4}

a . remove (5)

a

⇒ Error

If we have duplicate values in the set collection it, s will not accept duplicate value change as a single value.

3. discard ():

discard() is a inbuilt function which is use to remove the specific element from the set. If the user given the element if is not in the set collection it will not return error.

Syntax:

var/val . discard (element)

Eg: a = {1, 2, 3, 4, 5, 6}

a . discard (6)

⇒ {1, 2, 3, 4, 5}

a . discard (0) → If 0 is not present in the set it will not give any error

a

⇒ {1, 2, 3, 4, 5}

4. Pop():

Pop() is a inbuilt function which will remove the first element in the set and return the element which is removed.

syntax:

var/val . pop()

Eg:

a = {1, 2, 3, 'hii'}

a.pop() {1, 2, 3, 'hii'}

⇒ 1

b = a.pop() {2, 3, 'hii'}

⇒ b

→ 2

5. clear():

clear() is a inbuilt function which is use to remove the all elements in the set and return the empty set

syntax:

var/val . clear()

Eg:

a = {1, 2, 3, 'hii'}

a.clear()

⇒ set() → default value of Empty set

6. union():

union() is a inbuilt function which is unseal to add two sets and it will return in one set.

Syntax:

var/val . union(b)

Eg:

a = {1, 2, 3, 4, 5}

b = {7, 8, 9}

a . union(b)

⇒ {1, 2, 3, 4, 5, 7, 8, 9}

c = a . union(b)

⇒ c {1, 2, 3, 4, 5, 7, 8, 9}

7. Intersection():

intersection() is a inbuilt function which is used to return the common element present in the both sets.

syntax:

var/val . intersection(var/val)

Eg:

a = {1, 2, 'hello', 3}

b = {'hai', 4, 3, 'hello'}

c = a . intersection(b) / b . intersection(a)

⇒ c {'hello', 3}

8. Intersection_update():

It is a inbuilt function which is use to ~~remove~~ find the comman element blw sets and it will store in the used var and update the set common element.

Syntax:

~~var/val . intersection (var/val)~~ var . intersection_update (var)

Eg:

a = {1, 2, 3, 4}
b = {4, 5, 6}
a . intersection_update (b)

a
⇒ {4}
b
⇒ {4, 5, 6}

9. difference():

It is a inbuilt function which will return you the difference of one set to the another set (from the two sets)

Syntax:

var . difference (var)

Eg:

a = {'abcdef', 1, 2, 3, 4}
flag = {3, 4, 5, 6, 1, 2}
a . difference (flag)
⇒ {'abcdef'}

flag . difference (a)
flag
⇒ {5, 6}

10. difference_update() :

It is a inbuilt function which is use to find difference of one set to another set and it will ~~modify~~ update ~~the~~ inside the variable which we have mentioned.

syntax :

var . difference_update (var)

Eg :

a = {1, 2, 'hai', 4}

b = {4, 'hello', 'a'}

a . difference_update (b)

a

⇒ {4}

b . difference_update (a)

b

⇒ {4}

IN-BUILT FUNCTION ON DICTIONARY :

1. get() :

It is a dictionary inbuilt function which will return the value which is accosiated with the keys

syntax :

var . get (argument)

Eg :

d = {'a' : 'vinnu', 'b' : 'oupa', 'c' : 3}

d . get (keys)

⇒ {'vinnu', 'oupa', 3}

d . get ('a')

⇒ {'vinnu'}

d . get ('a', 'b')

⇒ 'vinnu'

10. difference_update() :

It is a inbuilt function which is use to find difference of one set to another set and it will ~~modify~~ update ~~the~~ inside the variable which we have mentioned.

Syntax :

var . difference_update (var)

Eg :

```
a = {1, 2, 'hai', 4}
b = {4, 'hello', 'a'}
a.difference_update(b)
a
⇒ {4}
```

```
b.difference_update(a)
b
⇒ {4}
```

IN-BUILT FUNCTION ON DICTIONARY :

1. get() :

It is a dictionary inbuilt function which will return the value which is accosiated with the keys

Syntax :

var . get (argument)

Eg :

```
d = {'a' : 'vinnu', 'b' : 'Rupa', 'c' : 3}
d.get(keys)
⇒ {'vinnu', 'Rupa', 3}
d.get('a')
⇒ {'vinnu'}
d.get('a', 'b')
→ 'vinnu'
```

2. Pop item ( ) :

It is a dictionary inbuilt function which will remove the last key value pair in the dictionary collection and it will return the key removed key value. in the form of tuple.

Syntax :

```
var . pop item ( )
```

Eg:

```
a = { 'a' : 'vinnu' , 'b' : 'kowmya' , 'c' : 'Rupa' }
a . popitem( )
⇒ ('c', 'Rupa')
```

3. pop ( ) :

It is a dictionary inbuilt function which will remove the value which is associated with key. And it will return the value which is Removed

Syntax :

```
var . pop (key)
```

Eg:

```
a = { 'a' : 1 , 'b' : 2 , 'c' : 3 }
a . pop (b)
⇒ 2
```

4. clear ( ) :

It is a inbuilt function which is use remove all the key values from the dictionary.

Syntax :

```
var . clear ( )
```

Eg:

```
a = {'a' : 1 , 'b' : 2}
a . clear ( )
⇒ {}
```

5. update():

It is a dictionary inbuilt function which is use to merge the two dictionaries.

syntax:

var.update(var) var/val(dict).update(var/val(dict))

Eg:

```
a = {'a':1}
b = {'b':'hai'}
a.update(b)
⇒ {'a':1, 'b':'hai'}
```

6. keys()

It is a inbuilt function which is use to extract keys from the dictionary.

syntax:

var/val.keys()

Eg:

```
d = {'a':1, 'b':'hai'}
d.keys()
⇒ dict_keys(['a', 'b'])
  list(d.keys()) → typecasting into list
  ⇒ ['a', 'b']
```

values()

It is a inbuilt function which is use to Extract the values from the given dictionary.

syntax:

var/val . values()

Eg: d = {'a' : 1, 'b' : 'apple'}

d . values()

~~→ {1, 'app~~

→ dict_values([1, 'apple'])

d list(d . values())

→ [1, 'apple']

converting for loop Pgms to Recursion:

Write a Program to find the count of integer numbers Present inside an hetrogeneous list collection.

```
li = ['hai', 2, 5.2, 3+2j, 5, 7, 10]
c=0, index
for i in range (0, len(li)) :
    if type (li[i]) == int :
        c+=1
Print (c)
```

```
def sam( li, c=0 , i=0 )
    if i == len( li)
        return c
    if type (li[i]) == int
        c+=1
    return sam( li, c, i+1)
Print (sam(['hai', 2, 5.2, 5, 7
```

If we want to convert for loop to recursion we have to use range function in the for loop
When we use for loop in recursion 'i' is act as index. because it will follows the range values.
Write a Pgm to Print the divisor of a given num

```
n=int (input ('enter the num :'))
for i in range (1, n+1) :
    if n%i == 0 :
        Print(i)
```

```
def div( n, i=1) :
    if i == n+1 :
        return 0
    if n%i == 0 :
        Print(i)
    return div (n, i+1)
Print (div (7)
div(7)
```

List comprehension :

WAP to find the square of all the numbers btw the range 1 to 10 and store the result in the form of list

```
n = 1
m = 10
a = []
for i in range (1, m+1) :
    a += [i*i]  / a.append(i*i)
Print(a)
```

- When Ever we want perform some small operations in the form of list we are going to make use of the concept called list comprehension.
- ~~with~~ with the help of this concept we can reduce the no of instructions and we can increase the Effiency of the code.

Syntax :

```
var = [var-to-be-stored for var in collection IF cond]
Print (var)
```

Eg :

```
Out = [i*i for i in range(1,11)]
Print(Out)
```

ist to tuple (type casting) :

```
Out = ([i*i for i in range(1,11)]) → ( )
Print(Out)
```

WAP to find the cube of all the Even numbers btw the range 1 to 10

```
a = [i**3 for i in range(1, 11) if i%2==0]
Print(a)
```

WAP to Extract all the string ~~of~~ data items present inside an hetrogeneous list collection only if len of string is more than or Equal to 3.

```
l = ['kai', 'ab', 3, '2.4', 'python']
out = []
for i in l:
    if type(i)==str and len(i)>=3:
        out+=[i]
Print(out)
```

In list comp -

```
Print([i for i in ['kai', 3, 'ab', 2.4, 'Python'] if type(i)==str
                                                  and len(i)>=3])
```

(or)

```
a = ['kai', 3, 'ab', 2.4, 'python']
Print([i for i in a if type(i)==st and len(i)>=3])
```

WAP to Extract an integer data items present at odd index in a given hetrogeneous list.

```
a = ['hai', 4, 3, 2.3, 10, 11]
print([a[i] for i in range(0, len(a)) if i%2 != 0 and type(a[i]) == int])
```

Input from user in the list:

```
a = input()
b = a.split()
Print ([int(i) for in b])
```

```
a = [int(i) for i in input().split()]
```

(or)

```
Print ([int(i) for i in input().split()])
```