28-2-23.

Function: It is a named memory block to store the set of instructions.

-> It is a set of instructions, when we call the instruction then only those instructions are execute.

-> It is a block of code when we call the block then only it will perform the task.

-> It is used for to store the ~~repeti~~ repetative task (or code) for reduce the code.

-> It eliminates the duplicate code.

-> It is used for code resuability.
-> " " " for memory optimization.
-> The function it is defined in two ways

1. Pre-defined function.
2. User-defined function.

pre-defined function: These are the functions are developed by Python developers.

-> As a programmers we have to pass the values.

-> The function will return specific output.

→ pre-defined functions another name called in-built functions or built-in functions.

→ pre defined functions are classified into six types;

1. pre-defined utility functions
2. pre-defined string functions.
3. pre-defined list functions.
4. pre-defined tuple functions.
5. pre-defined set functions
6. pre-defined dict. functions.

pre-defined string function:

capitalize, center, count, endswith, find, format, index, isalnum, isalpha, isascii, isdecimal, lower, islower, isspace, istitle, isupper, join, lower, lstrip, replace, rfind, rindex, rsplit, rstrip, split, splitlines, startswith, Strip, swapcase, title, upper.

pre-defined list function:

append, clear, copy, count, extend, index, insert, pop, remove, reverse, sort.

Pre-defined tuple function:

Count, index

Pre-defined set function:

add, clear, copy, difference, difference_update, discard, intersection, intersection_update, pop, remove, symmetric_difference, symmetric_difference_update, union, update.

pre-defined dict function:

Clear, copy, fromkeys, get, items, keys, pop, popitem, update, values, setdefault.

Pre-defined utility function:

bin, bool, bytes, chr, classmethod, complex, dict, dir, enumerate, eval, filter, float, frozenset ~~set~~, getattr, globals, hasattr, help, hex, id, input, isinstance, iter, len, list, locals, map, max, min, next, object, oct, open, ord, pow, print, property, range, round, setattr, slice, sorted, ~~static method~~ staticmethod, str, sum, super, tuple, type, vars, zip.

user-defined function: These are the functions created by users based on the scenario and we can create the own function by the help of function signature. There are two types of function signature;

1. function definition
2. function call

function definition: Defining the function with the help of 'def' keyword. def is a definition of the function. It is used to creating a sub stack space (function area or method area).

substack space: is a subset of stack space. (main space)

- It consists of execution area, variable space, value space.
- substack space will have unique address, inside the execution area we can store set of instructions.

Syntax: keyword; identifier; Inputs/requirements/arguments/parameters; Indent

```
def functionname(args, args.....):
    ..... set of statements .....
    ..... return/print .....
    ..... output statement .....
```

function block / statement block

example :

mainspace

e.area	V.S	V.S
-> def sample(): Print("Hi") print("bye")	sample = 0X888	

0X888

e.area	VS	VS
print("Hi") print("bye")		

created all statements stored.

Function Call : Calling the function by the help of function name.

Internal process : whenever control see function address with ( ) then the intrepreter will consider function call.

-> when we call the function the controller will move on to function area then it starts execution of all the instructions / process.

-> After complete the execution process the control comeback to where the function call happend.

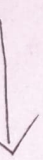
-> when controller is coming back defaultly it will return none.

-> The controller will come back to where Cal function will happen it will Call none

at the same time it destroy function area, variable space but all the instructions are present upto certain time.

1-3-23

```
def add( ):
    a = 100
    b = 200
    c = a+b
    print (`c value is', c)
    return c
print(add)
print(addC))
```



O/P:

<function add at 0x00.....>

C value is 300

300

```
a = 50
b = 60
def add( ):
    c = a+b
    print(`c value is', c)
    return c
Print (add)
print (addC))
```

O/P:

<function add at 0x00....>

C value is 110

110

```
def add (a, b):
    c = a+b
    Print ("c value is", c)
    return c

Print (add)
print (add(10, 20))
```

O/P :

```
<function add at 0x00...>
c value is 30
30
```

Case ① :

Stack Space

VS	VS
add = 0x121	

-> def add ( ):
    a = 100
    b = 200
    c = a+b
    print (c)

Sub Stack Space

0x121

	VS	VS
-> a = 100	a = 0x51	100 0x51
-> b = 200	b = 0x52	200 0x52
-> c = a+b	c = 0x53	300 0x53
-> print (c)		

shell

300

Case (2):

Stack Space

VS	VS
add = 0x121 V = 0x71	300 0x71

```
def add():
    a = 100
    b = 200
    c = a + b
    print(c)
    return c
V = add()
```

0x121

	VS	VS
a = 100 b = 200 c = a + b print(c) 300 return c 300	a = 0x51 b = 0x52 c = 0x53	100 0x51 200 0x52 300 0x53

Shell

300

Case (3):

VS	VS
a = 0x41 b = 0x51 add = 0x171 V = 0x61	50 0x41 60 0x51

```
a = 50
b = 60
def add():
    c = a + b
    print(c)
    return(c)
V = add()
```

0x171

EA	VS	VS
c = a + b print(c) return c	c = 0x54	110 0x54

Shell

110

Arguments

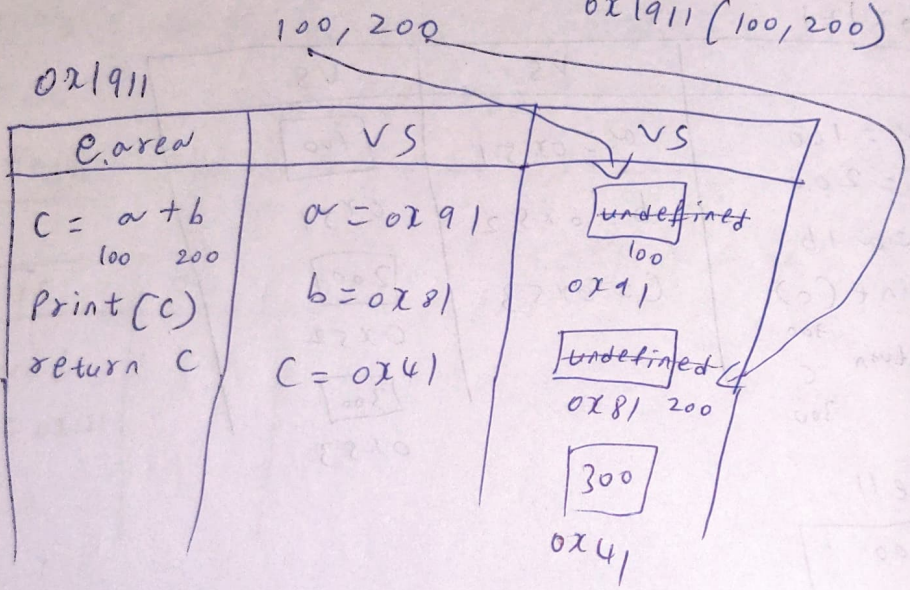
Case (4):

```
def add(a, b):
    c = a+b
    print(c)
    return c
```

VS	VS
add = 0x1911 V=0x76	300 0x76

v = add(100, 200)

0x1911(100, 200)



Shell

300

user - defined functions :-

1. Function without arguement, without return Statement
2. Function with ~~arg~~ arguement, with return Statement
3. Function without arg., with return statement.

4. Function with arg., with return Statement.

Function without arguement and without return Statement :-

> whenever we don't want to assign the values inside the function definition and we don't want to return any value from function. Then, we have to use this method.

> Directly we have to initialize the input inside the function statement.

> For display the output we have to use print statement.

Program :

```
def factorial ( ):
    n = eval (input())
    Fact = 1
    Start = 1
    While Start <= n :
        Fact = Fact * Start
        Start += 1
    print (fact)
factorial ( )
```

Function with arguments and without return Statement :-

-> whenever we have to assigning the values from function call to function definition. we don't want to return the output, that time we have to use this method.

-> In this method we have to declare the variables (arguements) inside the function definition. we have to pass the actual values from function call.

-> For display the output we have to use print Statement.

Note :-

-> we don't want to declare the input variables or arguements inside the function.

program :

```
def factorial (n):
    Fact = 1
    start = 1
    while start <= n:
        Fact = Fact * start
        start += 1
    print (fact)
factorial (5)
```

Factorial (eval (input()))

n = eval (input())

Factorial (n)

2-3-23.

return :- It is a keyword and termination statement. whenever controller see return keyword it will stop the function execution and carry the value from method area to where the call function is happened.

-> Defaultly the return statement will returns None value. If user assign any specific value then it returns that specific value.

-> return statement we can use within the function.

-> After return statement if any statements are present those statements are not executed.

Function without arguments and with return statement.

-> Whenever we don't want to assign the values inside the function and when we want to carry the output one space to another space then we have to use this method.

-> This method we call as 'setter method.

-> All the inputs we have to write inside the function statement.

-> Instead of print statement we have to use return statement.

```
def Factorial ( ):
    n = eval (input( ))
    Fact = 1
    start = 1
    while start <= n :
        Fact *= start
        start += 1
    return Fact

Factorial ( )

# V = Factorial( )
  print (V)
```

Input

logic

O/P

Function with arguments and with return statement :

-> Whenever we have to assign the values or inputs in function call when we have to return the value one space to another space, then we have to use this method.

-> In function def we have declare the Arguments / variables.

→ In function call we have to pass the actual values or inputs.

→ In function statement we have to use return keyword.

```
def Factorial(n):
    Fact=1
    Start=1
    While Start<=n:
        Fact*=Start
        Start+=1
    return Fact
print(Factorial(eval(inputC)))
```

logic

o/p

```
# V=Factorial(eval(inputC))
  print(V)
```

```
V=Factorial(5)
print(V)
```

Programs:

```
def add(a,b):
    c=a+b
    print("c value is", c)
    if c==30:
        return c
    print('havii')
print(add)
print(add(10,30))
```

o/p:

<function add at 0x00.....>

C value is 40

havii

None

~~For loop : It is~~

Looping Statements : It is Flow Control Statement will iterate the set of instructions 'n' no of times.

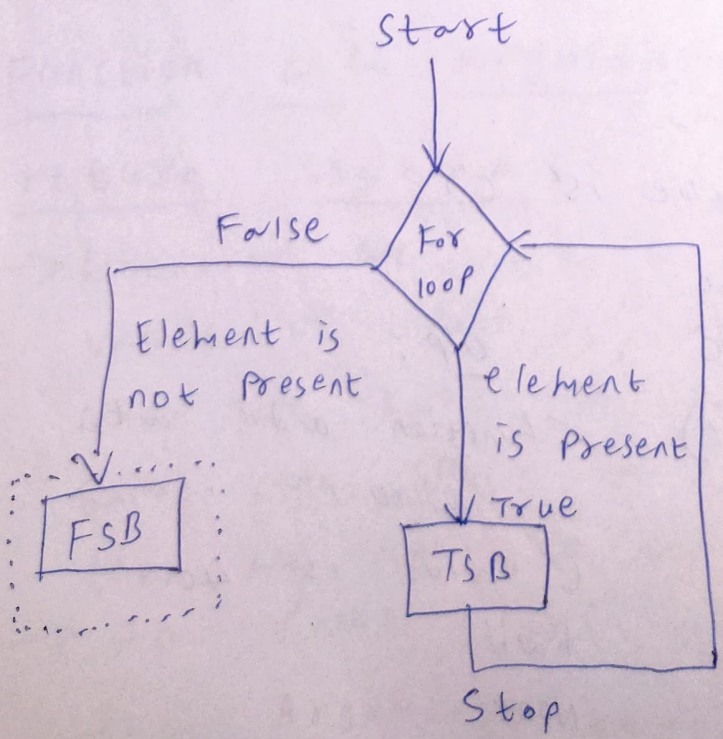
For loop : It is one of looping statement and iteration Statement.

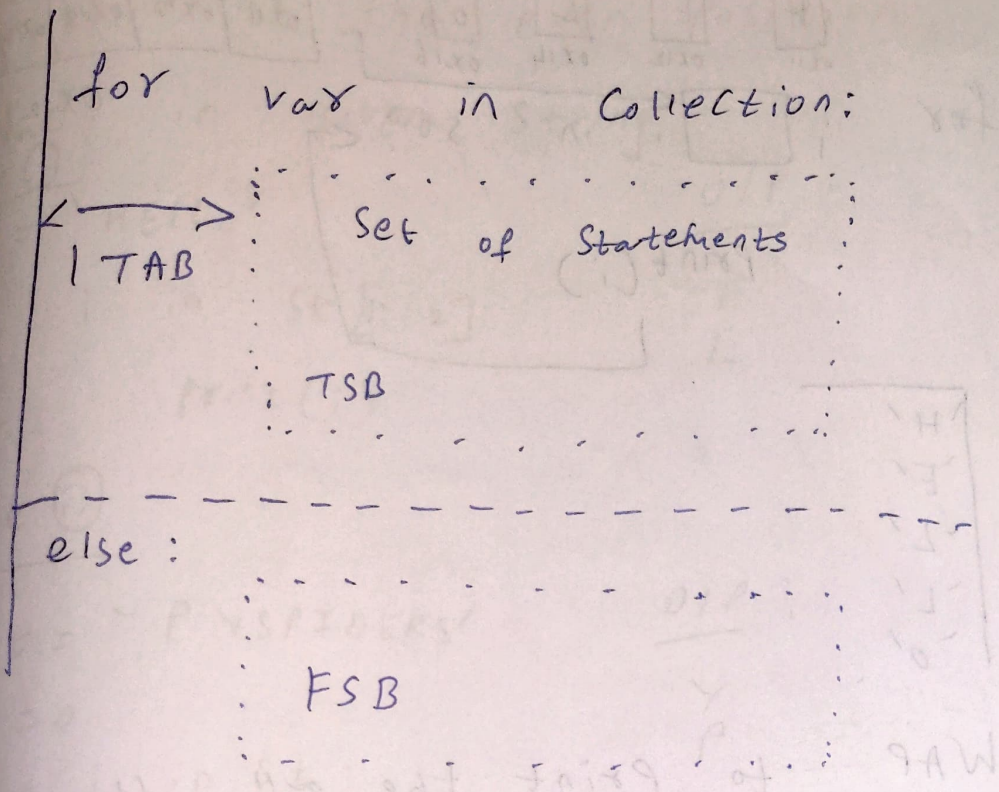
-> For loop will iterate set of statements upto length of collection.

~~for loop with~~

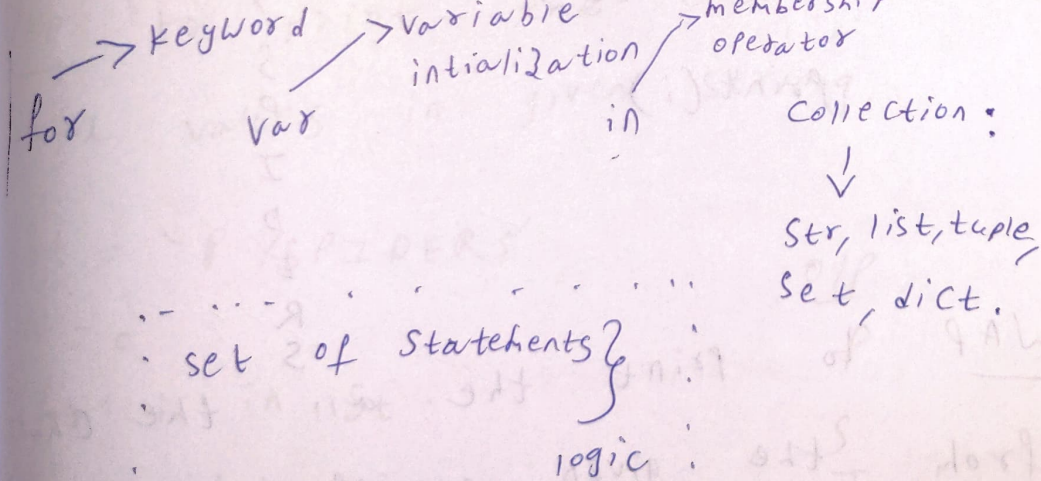
-> For loop will accept variable intilization and collection that collections are string, list, tuple, set, dict, range.

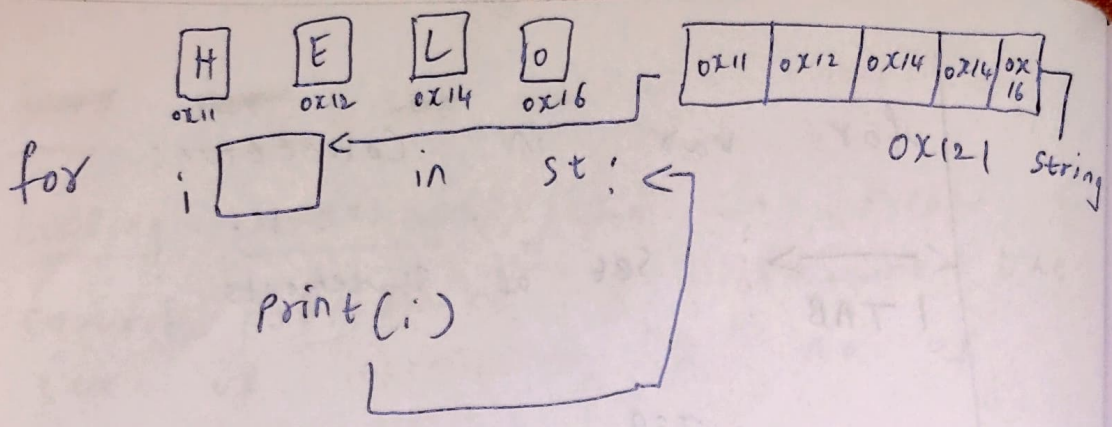
Flow Chart :





-> For statement will accept one or more ~~if~~ ~~b~~ variables and it will consider only one collection and else block is an optional.





'H'
'E'
'L'
'L'
'O'

WAP to print the all the Chars from the given String.

```
St = 'PYSPIDERS'
for i in st:
    print(i)
```

O/P :

P
Y
S
P
I
D
E
R
S

WAP to print the all the Chars from the given string in reverse order.

```
st = 'HELLO'
for i in st[::-1]:
    Print(i)
```

O/P :

O
L
L
E
H

WAP to print the even position charac. in given string.

case (i)

```
St = 'HELLO'
for i in St[1::2]
    print(i)
```

O/P :

E
L

case (ii)

```
St = 'PYSPIDERS'
a=0
for i in St:
    if a%.2!=0:
        Print(i)
    a+=1
```

O/P :

Y
P
D
R

WAP to print the even position index values in given string.

```
St = 'PYSPIDERS'
a=0
for i in St:
    if a%.2==0:
        Print(i)
    a+=1
```

O/P :

P
S
I
E
S

WAP to print uppercase charac. from given string.

```
st = 'HELLO'
for i in st:
    if 'A'<=i<='Z':
        print(i)
```

O/P:

H
L
L

3-3-23

WAP to count uppercase charac. in the string.

```
st = 'PYSPIDERS'
count = 0
for i in st:
    if 'A'<=i<='Z':
        count+=1
print(count)
```

WAP to extract and store the lowercase charac.

```
st = 'PYSPIDERS'
res = ''
for i in st:
    if 'a'<=i<='z':
        res+=i
print(res)
```

O/P:

spiers

IP to eliminate the duplicate Charac. in given string.

```
st = "PYspiDERs"
res = ""
or i in St:
    if i not in res:
        res += i
rint(res)
```

OIP:

PYspiDER

AP to eliminate the duplicate charac in string (don't consider case)

```
St = `PYspiDERs'
res = ` `
for i in st:
    if `A' <= i <= `Z' and i not in res
    and chr(ord(i)+32) not in res:
        res += i

    elif `a' <= i <= `z' and i not in res
    and Ch = Chr(ord(i)-32) not in res:
        ~~if Ch not in res:~~
        res+=i

Print(res)
```

OIP:

pYsideR.

WAP I/P = 'APPLE'

O/P = 'AIPPLE2'

Program:

```
st = 'APPLE'
res = ' '
count = 0
for i in st:
    if i in ('A', 'E', 'I', 'O', 'U'):
        count+=1 'a', 'e', 'i', 'o', u
        res = res + i + str(count)
    else:
        rest+=i
print(res)
```

O/P:

AIPPLE2

WAP to extract the values from given list

```
l = ['hai i', 'hello', 'bye', 'give me choco']
for i in l:
    print(i)
```

O/P:

hai i
hello
bye
give me choco.

I/P : "APPLE2" I/P : "APP345"

O/P : "APPLE3" O/P : "APP13"

WAP to concate all the String data items from given list.

```
l = ['haii', 'hello', 'bye']
res = ''
for i in l:
    if type(i) == str:
        res += i
print(res)
```

O/P

haiihellobye

WAP to product of all the numbers in given list.

```
l = [10, 10.05, 10+5j, True, 'str', [10, 20]]
prod = 1
for i in l:
    if type(i) in (int, float, complex):
        prod *= i
print(prod)
```

WAP to extract multivalue data types in given tuple.

```
t = (10, 10.05, 10+5j, True, 'str', [10,20], (neg)
     , {15}, {'a':10})
for i in t:
    if type(i) in (str, list, tuple, set, dict):
        print(i)
```

WAP to extract and store immutable data types in given set.

```
S = {True, 'str', 10, [1,2,3]}
l = [ ]
for i in s:
    if type(i) in (str, tuple):
        print(i)
```

program:

I/P : 'APPLE2' O/P : 'APPLE3'

I/P : 'APP3451' O/P : 'APP13'

program:

```
st = 'APP3451'
sum = 0
res = ''
for i in st:
```

```
if '0'<=i<='9':
    sum+=int(i)
else
    res+=i
res=res+str(sum)
print(res)
```

Op:
'APP13'

4-3-23.

append: It is a pre-defined list built-in function. This function will accept one input that input it will add inside the existing list.
->It will add the input ending of the list.

```
l=['hai', 'pyse', 'apple']
l.append('bye')
print(l)
l=['hai', 'pyse', 'apple']
l+=['bye']
print(l)
```

```
l = ['hai', 'pyse', 'apple']
l = l + ['bye']
print(l)

def uappend(coll, new):
    coll = coll + [new]
    return coll

res = uappend(["hai", "pyse", "apple"],
              "bye")
print(res)
```

Clear : It is a pre-defined list built-in function. This function destroy or clear the all data items inside the list and it returns empty list.

```
l = [10, 20, 30]
l.clear()
print(l)

def uclear(coll):
    start = 0
    while coll:
        del coll[start]
    return coll

v = uclear(['hai', 'helo', 'bye'])
print(v)
```

O/P :

[ ]

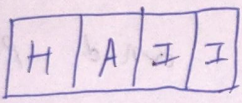
[ ]

Copy: It is a phenomenon of copying the data ~~from~~ one memory to another memory. This operation is called as copy operation.

Copy operations are classified into three types;

1. Normal copy
2. Shallow copy
3. deep copy

Normal copy: Copying the one variable reference address is assigned to new variable name.



s=0x11

st = s
0x11

a = 10
id(a)
>>> 7384
b = a
id(b)
>>> 7384

a
10
b
10

h = 'hello kaii'
n = 'hello kaii'
id(h)
>>> 7408
id(n)
>>> 8288

i = 'hello'
j = 'hello'
id(i)
>>> 9760

id(j)
>>> 9760

k = i
id(k)
>>> 9760

o = n
id(o)
>>> 8288

l = [10, 20, 30]
u = l
l[0] = 100
l >>> [100, 20, 30]
u >>> [100, 20, 30]

```
d = {'a': 10, 'b': 20}
w = d
d['a'] = 100
d
{'a': 100, 'b': 20}
w
{'a': 100, 'b': 20}
```

Shallow Copy: Copying the sub block address and paste it inside the new collection block.

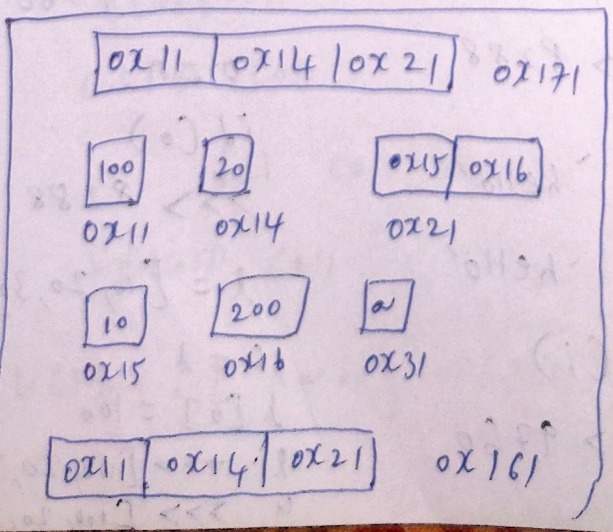
-> The old collection sub block address are copied inside and paste it inside new collection block.

-> New collection block ref. add. are assigned to new variable.

Syntax:

```
newvariablename = oldvariablename.copy()
```

```
ab = [100, 20, [10, 200]]
cd = ab.copy()
```



ab = 0x171

cd = 0x161

```
ab= [100, 20, [200, 500]]

ab
>>> [100, 20, [200, 500]]
cd = ab.copy()
cd
>>> [100, 20, [200, 500]]
id(ab)
>>> 0x171
id(cd)
>>> 0x161
id(ab[2])
>>> 2656
id(cd[2])
>>> 2656
ab[0] = 1000
ab
[1000, 20, [200, 500]]
cd
[100, 20, [200, 500]]
```

Normal copy (modification or deletion):

If we manipulate the data any of one list the effect will be happen in both the list.

6-3-23.

L = [10, 20, 30]

T = L

L[1] = 10

T[-1] = 50

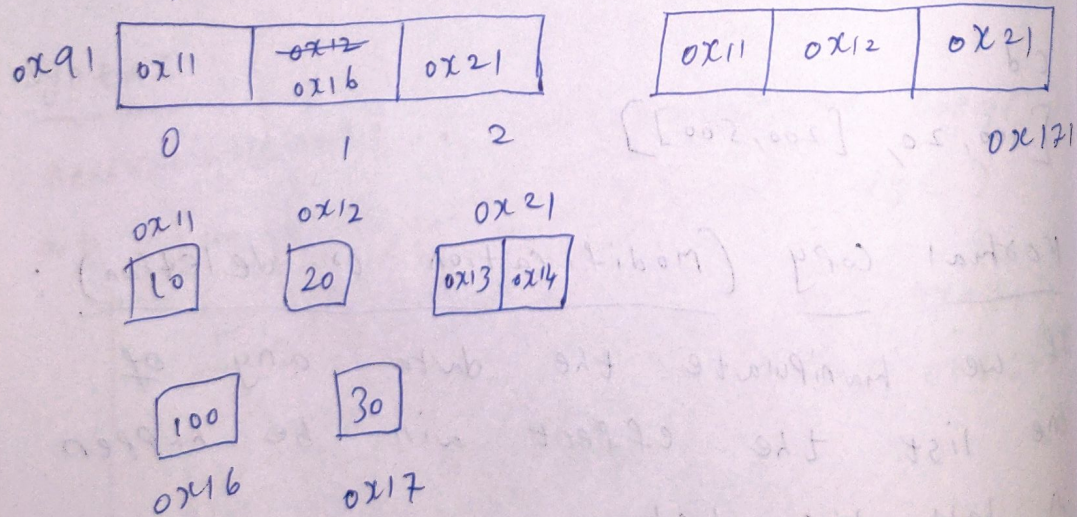
L 50
[10, 10, ~~30~~]

T 50
[10, 10, ~~30~~]

Modification in Shallow copy: If we manipulate the data in one list, that manipulation will happen in the main memory block.

-> If we modify any value in main list it will not effect on copy list.

-> If we manipulate the data in sub list inside the main list it will effect on original list and copy list.



U = [10, 20, [50, 40]]

```
V = U.copy()
U[1] = 100
U[2][0] = 30
```

U [10, 100, [30, 40]] V [10, 20, [30, 40]]

Deep Copy: If we copy the data one collection to another collection all the values are same but for each value memory address is changed in the deep copy.

- for non-primitive and mutable data types memory block addresses are changed
- for primitive, memory block addresses are not changed.

Syntax:

```
import copy
newvariablename = copy.deepcopy(existing variable name)
```

- Inside the list if any mutable data types are available if we apply deep copy coming to the copied list, the mutable memory blocks are re created and the recreated memory block address is assigned to copied list.

-> If we manipulate any data in main collection or sub collection it will not effected on copied collection.

```
import copy
ab = [10, 'hai', [50, 60]]
cd = copy.deepcopy(ab)
ab
[10, 'hai', [50, 60]]
cd
[10, 'hai', [50, 60]]
ab[0] = 100
ab
[100, 'hai', [50, 60]]
cd
[10, 'hai', [50, 60]]
cd[1] = 'hello'
ab
[100, 'hai', [50, 60]]
cd
[10, 'hello', [50, 60]]
ab[-1][0] = 'a'
ab
[100, 'hai', ['a', 60]]
```

cd

'hello', [50, 60]]

d[-1][-1] = 'bye'

b 'hai', ['a', 60]]

d

[10, 'hello', [50, 'bye']]

Normal copy :

	original collection	duplicate collec.
in Collection	Modify/delete	affect
	affect	Modify/delete
b Collection	Modify/delete	affect
	affect	Modify/delete

Shallow copy :

	original collection	duplicate collection
in Collection	Modify/delete	not affect
	not affect	Modify/delete
b Collection	Modify/delete	affect
	affect	Modify/delete

Deep copy :

	original coll.	Duplicate coll.
Main Collection	Modify / delete	not affect
	not affect	Modify/delete
Sub Collection	Modify/delete	not affect
	not affect	Modify/delete

Note: Modify/delete we can do in the list/dict.

Example:

L = [10, 20, [10, 20]]

→ T = L ——————→ Normal Copy

→ T = L.copy( ) ——————→ Shallow Copy.

→ import copy

T = copy.deepcopy(L) ——→ Deep Copy.

program : user defined Shallow Copy

```
L = [10, 20, ['hai', 50]]
def ShallowCopy(coll):
    h = []
    for i in coll:
        h += [i]
    return h
L = ShallowCopy(L)
print(L)
print(t)
```

o/p :

[10, 20, ['hai', 50]]

[10, 20, ['hai', 50]]

program : user defined deep copy

```
def usercopyd(coll):
    temp = { }
    for j in coll:
        if type(coll[j]) == dict:
            temp[j] = usercopyd(coll[j])
        elif type(coll[j]) == list:
            temp[j] = usercopyd[coll[j]]
        else:
            temp[j] = coll[j]
    return temp.
```

```
def usercopyl(coll):
    temp = [ ]
    for j in coll:
        if type(j) == list:
            temp += [usercopyl(j)]
        elif type(j) == dict:
            temp += [usercopyl[j]]
        else:
            temp += [j]
    return temp

l = [10, 20, [10, 20, [50], [15, 10]], {'a': {'d': [15]}, 'b': [10, {'z': 30}]}]

def usercopy(l):
    res = [ ]
    for i in l:
        if type(i) == list:
            res += [usercopy l(i)]
        elif type(i) == dict:
            res += [usercopy d(i)]
        else:
            res += [i]
    return res

res = usercopy(l)
print(l)
print(res).
```

7-3-23

WAP to print keys from dict.

```
d= {'a':10, 'b':20, 'c':30}
for i in d:
    print(i)
```

```
a
b
c
```

WAP to print the keys and value from the dict

```
d = {'a':10, 'b':20, 'c':30}
for i in d:
    print(i, d[i])
```

```
a 10
b 20
c 30
```

WAP to count the number of times values present in given list.

```
L= [10, 20, 30, 'count', 'value', 'keys', 10]
print(L.count('count'))
```

o/p:

```
1
```

Count : It is a pre-defined list function it will count the number of occurences of given value in collection and it will accept one input and it returns count the no. of times value is occured in integer format or integer value.

program :

```
L = [10, 20, 30, `count', `value', `keys', 10, `count']

def ucount(val, coll):
    count = 0
    for i in coll:
        if val == i:
            count += 1
    return count

res = ucount(`count', L)
print(res)

res = ucount(`c', `choco')
print(res)
```

o/p

2
2

~~to find out the maximum occurance~~

~~in a given collection.~~

~~program~~ :

```
st = 'choco'
def unique(st):
    res = []
    for i in st:
        if i not in res:
            res += [i]
    return res
uni = unique(st)
d = {}
for i in uni:
    count = 0
    for j in st:
        if i == j:
            count += 1
    d[i] = count
print(d)
```

O/P :

{'c': 2, 'h': 1, 'o': 2}

~~WAP~~ to eliminate charac. or values in given collection (string, list, tuple)

```
st = (10, 20, 30, 40, 50, 10, 20)
def unique(s):
    res = []
    for i in s:
        if i not in res:
            res += [i]
    return res          ---> list output
    return ''.join(res) -> string  "
    return tuple(res)   -> Tuple   "
unique(st)
```

WAP to find max occurance of value in given collection.

```
St = 'choco'
def unique(s):
    res = []
    for i in s:
        if i not in res:
            res += [i]
    return res

def maxOccur(st):
    uni = unique(st)
    max = 0
    ch = ' '
    for i in uni:
        count = 0
        for j in st:
            if i == j:
                count += 1
        if max < count:
            max = count
            ch = i
    return ch, max
res = maxOccur(st)
Print(res)
```

O/P :

C 2

WAP to find min occurance of value in given collection.

st = (10, 20, 30, 40, 40, 50, 10, 20, 40, 40)

```
def unique(s):
    res = []
    for i in s:
        if i not in res:
            res+ = [i]
    return res

def minoccur(st):
    uni = unique(st)
    min = len(st)
    ch = ' '
    for i in uni:
        count = 0
        for j in st:
            if i == j:
                count += 1
        if min > count:
            min = count
            ch = i
    return ch, min

res = minoccur(st)
print(res)
```

O/P:
(30, 1)

8-3-23.

WAP to find out min element and max element in given collection.

```
l = [55, 99, 89, 72, 36, 67, 18, 10]
max = 0
for i in l:
    if max < i:
        max = i
    if min > i:
        min = i
print(min, max)
```

o/p:

10 99

Program:

```
d = {'ratna': 55, 'seetha': 99,
     'raja': 89, 'rani': 72, 'dinga': 36,
     'dingi': 67, 'satn': 18, 'sai': 10}
minname = ''
maxname = ''
min = 10000000
max = 0
```

```
for i in d:
    if max < d[i]:
        max = d[i]
        maxname = i
    if min > d[i]:
        min = d[i]
        minname = i
print(minname, min)
print(maxname, max)
```

sai 10

eetha 99

Program:

```
l = [10, 20, 10, 30, 40, 20, 10]
u = set(l)
final = 0
t = [ ]
for i in u:
    count = 0
    for j in l:
        if i == j:
            count += 1
    if count % 2 != 0:
        final += 1
        t += [i]
print(final)
```

O/P

3

[40, 10, 30]

Program:

```
l = ['red', 'green', 'yellow', 'red', 'pink',
     'green', 'yellow', 'violet', 'red']
u = set(l)
final = 0
t = []
for i in u:
    count = 0
    for j in l:
        if i == j:
            count += 1
    if count % 2 != 0:
        final += 1
        t += [i]
print(final, t)
```

O/p:

3 ['red', 'pink', 'violet'].

Extend: It is a pre-defined list function, it is used for concating the set of values from any collection data type to existing list. Extend function will accept only collection data type. it will collect all the date items from given collection an

will concate all the values ending the list.

It returns existing list collection.

Example:

```
l = [10, 20, 30]
l.extend('10')        [10, 20, 30, '1', '0']
print(l)
```

user defined extend function:

```
def extend(coll1, coll2):
    if type(coll1) == list:
        if type(coll2) in
            [str, list, tuple, set, dict]:
            for i in coll2:
                coll1 += [i]
            return coll1
        else:
            raise typeerror('coll2
                value should be
                iterable data type')
```

```
else:
        raise typeerror('coll1
                value should be list
                data type')
res = uextend([10,20,30],[50:60,70:80])
print(res)
```

O/p

[10, 20, 30, 50, 70]

Program:

```
L = [10,20,30,'Hari']
T = [70,80,97,'Hello',10+2j,[10,20]]
def uextend(coll1,coll2):
    if type(coll1)==list:
        if type(coll2) in [str,list,tuple,
                           set,dict]:
            for i in coll2:
                if type(i)==int and
                   i%2==0:
                    coll1+=[i]
```

```
elif type(i) == str:
    coll1 += [i]
return coll1
```

s = extend(L, T)

int(res)

Output:

res = [10, 20, 30, 70, 'Haii', 80, 'Hello']

Program:

```
l = [10, 20, 30, 'haii']
t = [70, 80, 92, 'hello', 10+2j, [10,20], {'s': 'sah'}]
def extend(coll1, coll2):
    if type(coll1) == list:
        if type(coll2) in [str, list, tuple, set, dict]:
            for i in coll2:
                if type(i) in [str, list, tuple, set, dict]:
                    coll1 += [i]
            return coll1
```

```
else:
    raise typeerror("coll2 value should be iterable datatype")
else:
    raise type error("coll1 value should be list data type")
res = l.extend(l, t)
print(res).
```

O/P:

[10, 20, 30, 'hai', 'hello', [10, 20], {'s': 'satu'}]

insert: insert is a pre-defined list built-in function by using insert function we can append the new value in specific position inside the list. This function will accept two values, 1. index value

2. actual value.

-> it returns existing list.

```
l = [10, 20, 30]
l.insert(-3, [100, 200, 300])
print(l)
```

Doubt.

1-3-23.

user def

defined insert function:

```
def insert(ind, new, coll):
    coll = list(coll)
    u = []
    if ind < 0:
        ind = len(coll) + (ind)
        print(ind)
    if ind >= len(coll) or ind < 0:
        coll += [new]
        return coll
    else:
        start = 0
        while start < len(coll):
            if start == ind:
                u += [new]
                u += [coll[start]]
            else:
                u += [coll[start]]
            start += 1
        print(u)
```

insert(18, [150, 250], [100, 200, 500, 600])

[100, 200, 500, 600, [150, 250]].

Program:

```
def vinsert(ind, new, coll):
    if type(coll) in [int, float, complex,
                      bytes, bool, type(None)]:
        return "invalid collection"
    temp = type(coll)
    coll = list(coll)
    u = []
    if ind < 0:
        ind = len(coll) + (ind)
    if ind >= len(coll) or ind < 0:
        coll += [new]
        return coll
    else:
        start = 0
        while start < len(coll):
            if start == ind:
                u += [new]
                u += [coll[start]]
            else:
                u += [coll[start]]
            start += 1
        if temp == str:
            return ''.join(u)
        elif temp == tuple:
            return tuple(u)
        else:
            return u.
vinsert(3, 500, [10, 20])
```

O/P:

[10, 20, 500]

pop : It is a pre-defined list built-in function it is used for to remove the specific data item or remove the end of the list data item. (using index value)

-> It will accept one index value.

-> based on the index value it will remove the index position value.

-> If in case we are not assigned any index value defaultly it eliminates last data item inside the list.

```
l = [10, 20, 30, 40, 50, 60, 80]
Print (l.POP(-2))
Print (l)
```

O/p :

```
60
[10, 20, 30, 40, 50, 80]
```

Program: User defined pop function:

```
def upop (coll, ind=-1):
    if ind == -1:
        res = coll [ind]
        del coll [ind]
        return res
    if ind < 0:
        ind = len (coll) + (ind)
    if ind >= len (coll) or ind < 0:
        return 'out of index'
    res = coll [ind]
    del coll [ind]
    return res
l = [10, 20, 40, 50, 60, 70]
res = upop (l, -2)
print (res)
print (l)
```

O/P:

```
60
[10, 20, 40, 50, 70]
```

program:

```
def upop(coll, ind=-1):
    temp = type(coll)
    coll = list(coll)
    if ind == -1:
        res = coll[ind]
        del coll[ind]
        return res
    if ind < 0:
        ind = len(coll) + (ind)
    if ind >= len(coll) or ind < 0:
        return "out of index"
    res = coll[ind]
    del coll[ind]
    if temp == str:
        return res, ''.join(coll)
    if temp == tuple:
        return res, tuple(coll)
    else:
        return res, coll
l = [10, 20, 40, 50, 60, 70]
res = upop(l, -2)
print(res)
print(l)
```

o/p:

```
60
[10, 20, 40, 50, 70]
```

10-3-23

Remove: It is a pre-defined built-in list function. This function is used for removing the specific element inside the Collection.

-> This function accepts only one arguement and it remove the element inside the collection and it returns None.

-> It will remove the first occurance.

Program: User defined remove function.

```
def uremove(ele, coll):
    if ele not in coll:
        return "invalid"
    indexvalue = 0
    for i in coll:
        if ele == i:
            del coll[indexvalue]
            break
        indexvalue += 1
    return coll

l = [10, 50, [10,20], 60, 10]
uremove([10,20], l)
```

O/P

[10, 50, 60, 10]

example:

```
# l = [10, 50, [10, 20], 60, 10]
# l.remove([10, 20])
# print(l)
```

[10, 50, 60, 10].

WAP to eliminate the n^{th} occurance of the value from the given collection.

```
def uremove(n, ele, coll):
    if ele not in coll:
        return 'invalid'
    count = 0
    indexvalue = 0
    for i in coll:
        if ele == i:
            count += 1
        if count == n:
            del coll[indexvalue]
            break
        indexvalue += 1
    return coll
```

l = [10, 50, [10, 20], 60, 10]

uremove (2, 10, l)

o/p :

[10, 50, [10, 20], 60].

program :

```
def uremove(n, ele, coll):
    temp = type(coll)
    if temp not in [str, list, tuple]:
        return 'invalid collection'
    coll = list(coll)
    if ele not in coll:
        return 'invalid'
    count = 0
    indexvalue = 0
    for i in coll:
        if ele == i:
            count += 1
        if count == n:
            del coll[indexvalue]
            break
        indexvalue += 1
```

```
if temp == str:
    return ''.join(coll)
elif temp == tuple:
    return tuple(coll)
else:
    return coll
```

remove(3,10,l).
-> l = [10,20,10,50,30,10,90]

O/P:

[10,20,10,50,30,90]

remove(3,'l',l).
-> l = 'helloworld'

O/P:

helloword

remove(3,10,l).
-> l = (10,20,10,50,10,90)

O/P:

(10,20,10,50,90).

range: It is a pre-defined utility function it generates the sequence of number and stores the numbers in collection block.

-> It generates the value based on starting index, ending index, updation values.

-> The range function returns collection block address.

-> range is a pre-defined function as well as data type.

Syntax:

range (si, ei+1, up)

-> It is classified into two types.

1. positive range
2. negative range

Positive range: It will travel from left to right and updation value should be positive.

rule:

si < ei

-> we can eliminate the si and up value but we cannot eliminate the ei value.

-> si default value is zero and up default value is one.

Negative range : It will travel from right to left and updation value should be negative.

rule :

Si > ei

⇒ we cannot eliminate the starting index, ending index, updation.

example :

range (0, 10, 1) -> collection address.

```
>>> range(0, 5, 1)
    range(0, 5)
>>> e = range(0, 5, 1)
>>> type(e)
    <class 'range'>
>>> list(e)
    [0, 1, 2, 3, 4]
>>> list(range(6))
    [0, 1, 2, 3, 4, 5]
>>> list(range(6, 1))
    [ ]
```

```
>>> list(range(1,6))
[1,2,3,4,5]
>>> list(range(0,10,2))
[0,2,4,6,8]
>>> list(range(1,11,2))
[1,3,5,7,9]
>>> list(range(1,12,3))
[1,4,7,10]
```

~~WAP~~

```
for var in collection :
    print(var)
```

collection → str, list, tuple, set, dict, range

WAP to print the series of values by using given range.

```
for i in range(1,10)
    Print(i)
```

O/P

1
2
3
4
5
6
7
8
9

WAP to find out the ~~factorial~~ factors of given value:

```
n = 15
for i in range (1, n+1):
    if n % i == 0:
        print(i)
```

o/p

1
3
5
15

WAP to print the series of upper case characters in given range.

```
n = 65
n = 90
for i in range (65, 90+1):
    print(chr(i))
```

o/p:

A
B
C
.
.
.
Z

11-3-23.

reverse : It is a pre-defined list built-in function. It is used for reversed the values within the same list.

-> It returns none value.

-> It will not accept any arguements

```
l = [10, 500, 60, 80]
r = l.reverse()
print(r)
print(l)
```

program : user defined reverse function.

```
def ureverse(coll):
    n = len(coll)
    for i in range(0, n//2):
        coll[i], coll[(n-1)-i]
            = coll[(n-1)-i], coll[i]
l = [10, 20, 30, 40, 50, 60]
print(l)
print(ureverse(l))
print(l)
```

O/P:

10, 20, 30, 50, 60]

one

[60, 50, 30, 20, 10]

Program:

```
def reverse(coll):
    temp = type(coll)
    coll = list(coll)
    n = len(coll)
    for i in range(0, n//2):
        coll[i], coll[(n-1)-i]
        = coll[(n-1)-i], coll[i]
    if temp == str:
        return ''.join(coll)
    elif temp == tuple:
        return tuple(coll)
    else:
        return coll

# l = [10, 20, 30, 40, 50, 60]
# l = (10, 20, 30, 40, 50, 60)
l = 'reverse'
```

```
print (l)
print (ureverse(l))
```

o/p :-

reverse

esrever.

Sort : It is a pre-defined list built-in function. It is used for Sorting the values either ascending order or descending order.

-> Sorting operation will happen inside the same list.

-> It returns none value.

To perform the Sorting for ascending order :

Syntax :

```
varname.Sort()        (or)
varname.sort(reverse = False)
```

To perform the sorting for descending order :

Syntax :

```
varname.sort(reverse = True)
```

program : user defined sort function.

Case 1 :

```
l = [50, 80, 90, 40, 10, 5, 25]
for i in range (0, len(l)-1):
    for j in range (0, (len(l)-1)-i):
        if l[j] > l[j+1]:
            l[j], l[j+1] = l[j+1], l[j]
print(l)
```

o/p :

[5, 10, 25, 40, 50, 80, 90]

Case 2 :

```
l = [50, 80, 90, 40, 10, 5, 25]
for i in range (0, len(l)):
    for j in range (0, len(l)):
        if l[i] < l[j]:
            l[i], l[j] = l[j], l[i]
print(l)
```

o/p :

[5, 10, 25, 40, 50, 80, 90]

Program : User defined ascending and descending sort.

```
l = [50, 80, 90, 40, 10, 5, 25]
def usort(col, reverse=False):
    if reverse:
        for i in range(0, len(l)-1):
            for j in range(0, (len(l)-1)
                -i):
                if l[j] < l[j+1]:
                    l[j], l[j+1]
                    = l[j+1], l[j]
    else:
        for i in range(0, len(l)-1):
            for j in range(0, (len(l)-1)
                -i):
                if l[j] > l[j+1]:
                    l[j], l[j+1]
                    = l[j+1], l[j]
usort(l)
usort(l, True)
print(l)
```

[5, 10, 25, 40, 50, 80, 90]