21 April 2025 07:25

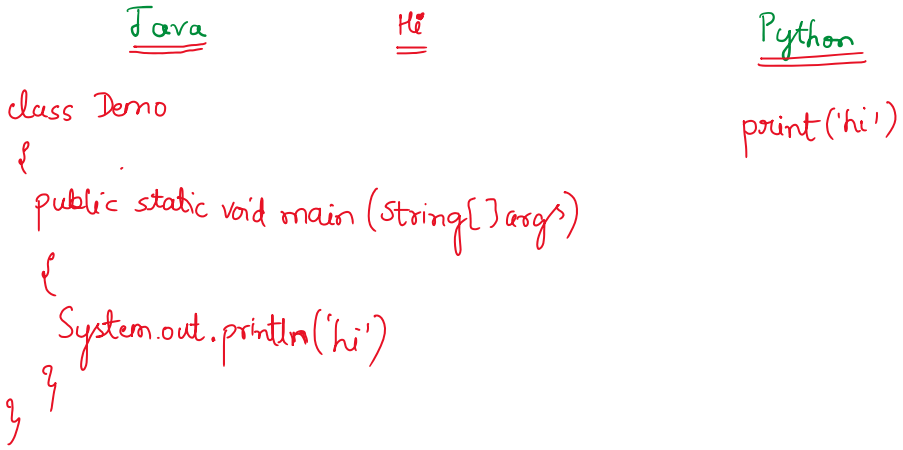
Day-1

Python
--- **Guido Van Rossum** is the person who created python programming language in the year 1991.

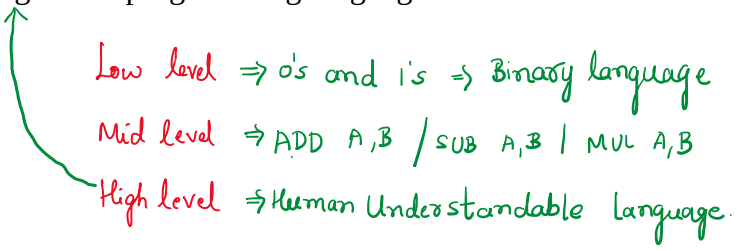
Programming Language : The language which is used communicate with the system.

Features of Python:

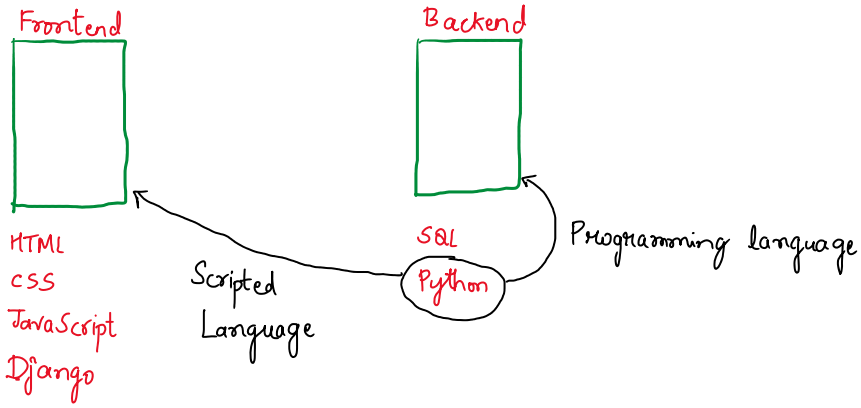
- It is easy to learn and analyse.



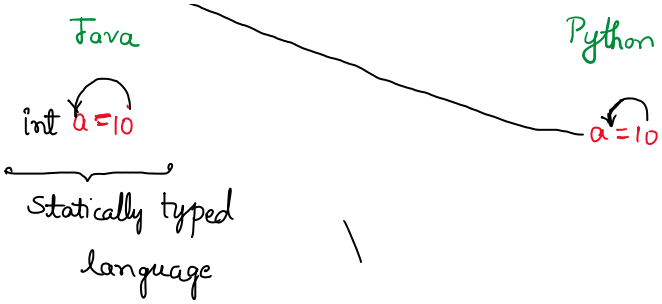
- It is high level programming language.



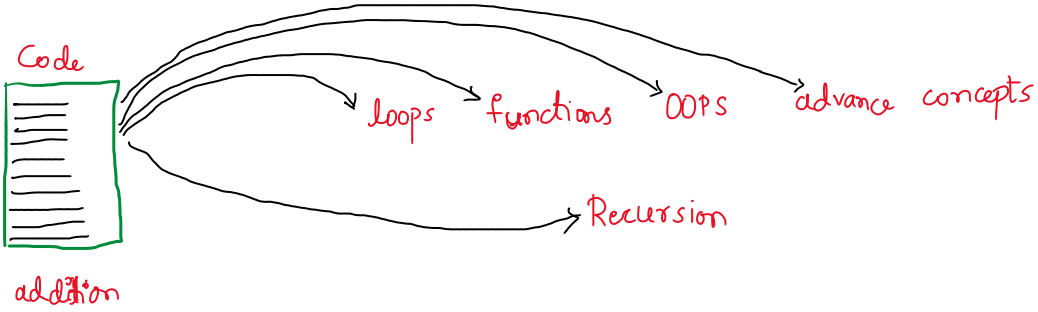
- It is scripted language.



- It is dynamically typed language.



- It is multi-paradigm - A code can be written in multiple ways.

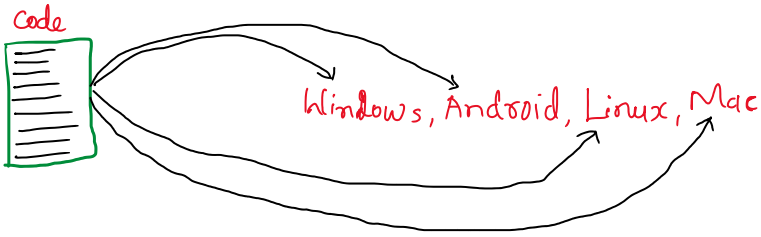


- It supports OOPS concept.

  OOPS - Object Oriented Programming System.

- It is open source.
  Free to download and we can contribute the code for python.
  numpy and pandas libraries are the examples of these.

- It is platform Independent.



- It has huge number of libraries.
  It has 7+ Crore libraries.

What is Python?

--- It is high level , general purpose programming language.

Day-2

<u>Library Functions:</u>

They are functions which are predefined by the developer to perform specific task.
We can only access them but we cannot modify them.

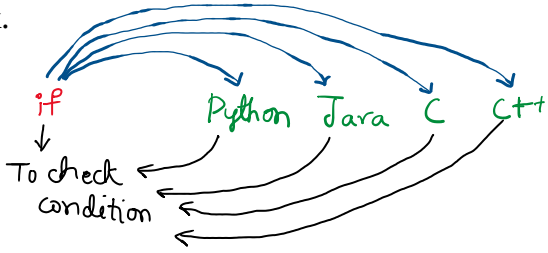
<u>Types</u>:

- Keywords
- Inbuilt functions

- Operator / Special symbols

1) <u>Keywords</u>

--- They are universally standard words which are predefined by the developer to perform specific task.



There are 35 keywords in python.

To print all 35 keywords we have to use the syntax,

```
import keyword
keyword.kwlist
['False', 'None', 'True', 'and', 'as', 'assert', 'async', 'await', 'break', 'class', 'continue',
'def', 'del', 'elif', 'else', 'except', 'finally', 'for', 'from', 'global', 'if', 'import', 'in', 'is',
'lambda', 'nonlocal', 'not', 'or', 'pass', 'raise', 'return', 'try', 'while', 'with', 'yield']
len(keyword.kwlist)
35
```

Special keywords - True, False, None
Reason -

- Starting with uppercase
- It can be used as keyword and value

Proof:

```
a=10
a
10
a = True
a
True
a = False
a
False
a = None
a = if
SyntaxError: invalid syntax
a = and
SyntaxError: invalid syntax
```

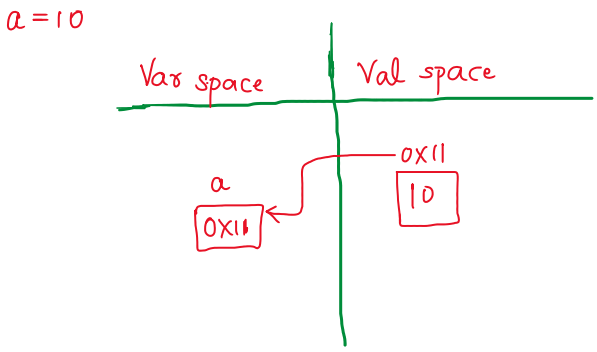
<u>Variable</u>:

--- It is a container which is used to store the address of the value.
Or
It is a name given to memory location where we store the value.

<u>Syntax</u>:

var = value

Memory allocation:



id():
--- It is used to get the actual address of the value stored in the memory.

Syntax:

id(var / val)

a
10
id(a)
140722490471496
id(10)
140722490471496

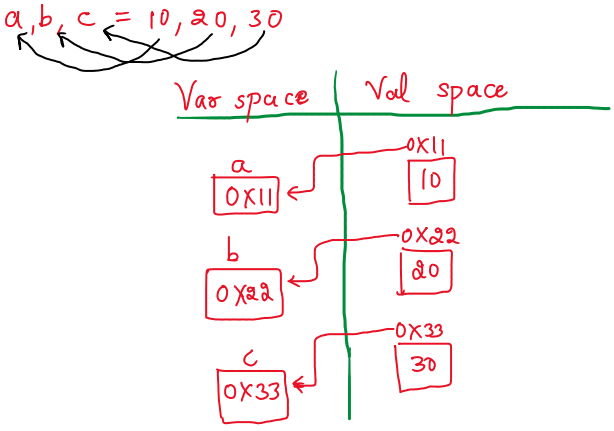
Multiple Variable Creation(MVC):
--- Creating multiple variable in a single line is known as MVC.

Syntax:

var1, var2, ..., varn = val1, val2, ..., valn

Rule: Number of variable must be equal to number of value.

Example 1:



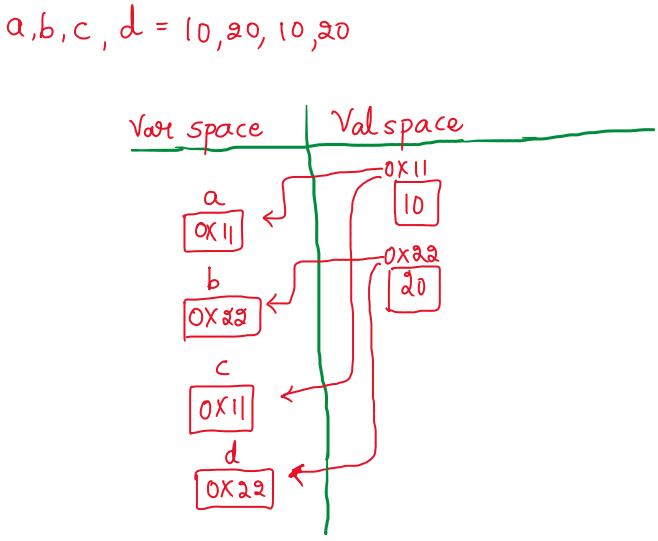
Proof:

a,b,c = 10,20,30
a

10
b
20
c
30

id(a)
140722490471496
id(b)
140722490471816
id(c)
140722490472136

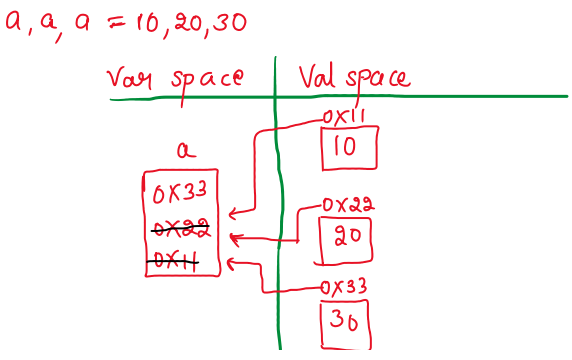
What will happen if multiple variable have same values?

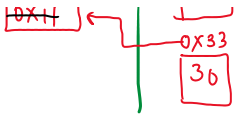


Proof:

a,b,c,d=10,20,10,20
id(a)
140722490471496
id(c)
140722490471496
id(b)
140722490471816
id(d)
140722490471816

What will happen if multiple variable have same variable name?

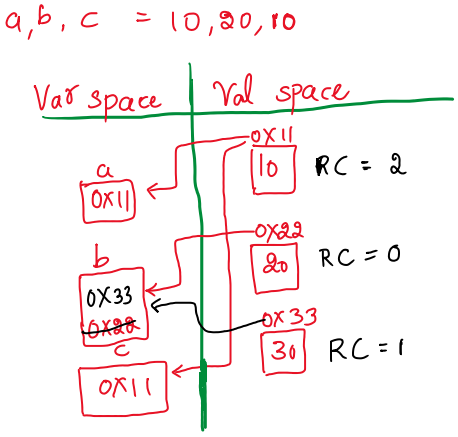




Proof:

a,a,a=10,20,30
a
30

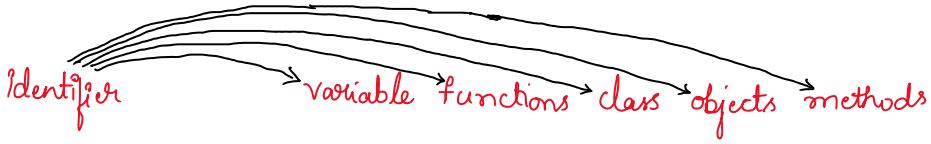
Reference count:
--- It will count the number of variables sharing the same value. When reference count becomes zero It will deleted from memory and collected by garbage collector.



b = 30

Identifier: Giving name uniquely to store the values.

All the variables are identifiers but all the identifiers are not variables.



Rules of Identifier:

1) **Identifier should not be a keyword.**
   Reason: Keywords are already predefined with some task.

   Proof:

   if = 10
   SyntaxError: invalid syntax
   and = 10
   SyntaxError: invalid syntax
   True = 10
   SyntaxError: cannot assign to True

2) **Identifier should not starts with number.**

   2a = 10
   a = 10

$2a = 10$

$a = \frac{10}{5}$

$a = 2$

3) Identifier should not contain any special character except underscore(_)
Special character = !@#$%^&*()_+{}:"<>,.?/|[]~`

Reason: All the special keywords are predefined with some task except underscore.

Proof

```
a_b = 10
a_b
10
ab_ = 10
ab_
10
_ = 10
_
10
a+b =  20
SyntaxError: cannot assign to expression here. Maybe you meant '==' instead of '='?
```

4) Identifiers should not contain spaces in between or at the beginning.

Proof:

```
 a =10

SyntaxError: unexpected indent
a b = 10
SyntaxError: invalid syntax
ab =10
ab
10
```

5) Identifiers can be alphabets, alphanumeric and underscore.

Proof:

```
a = 10
a
10
a2 = 20
a2
20
_ = 30
_
30
```

Industrial Standard Rule:

- Identifier name cannot exceed more than 79 characters.

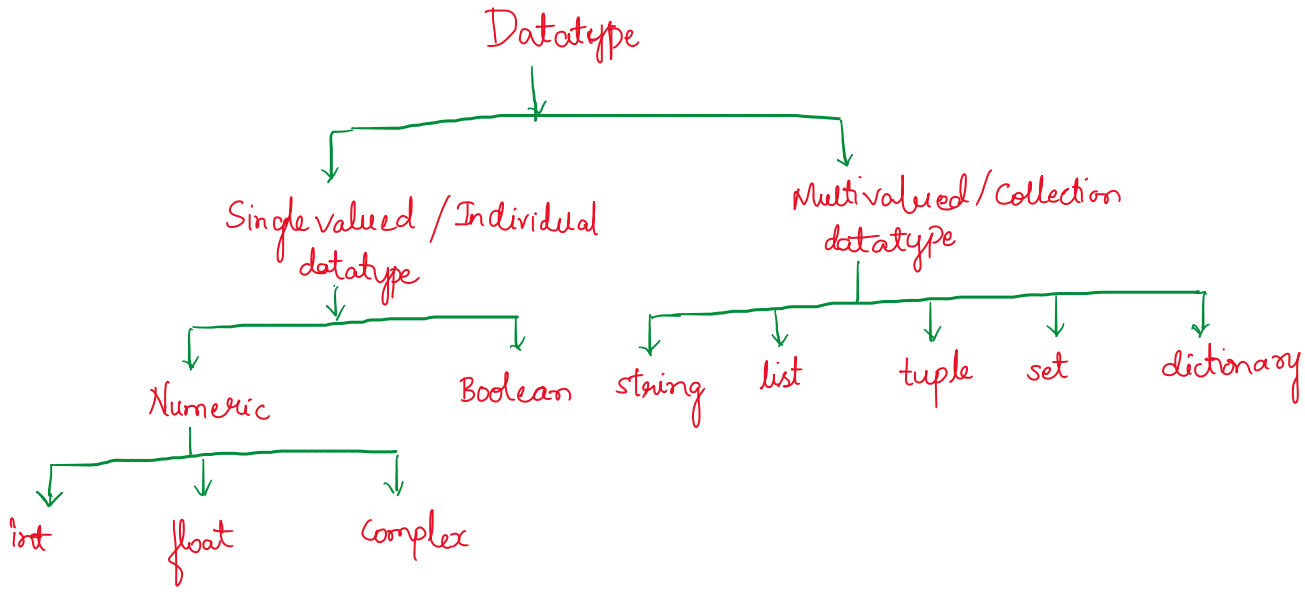
Proof:

```
fvmsnvjdsjfljdsnjfkhdsKFNDSFJHDHFJDSJFdbfjjDFHjfhdskflkdjvlks = 10
fvmsnvjdsjfljdsnjfkhdsKFNDSFJHDHFJDSJFdbfjjDFHjfhdskflkdjvlks
10
a = 10
a
10
```

Day-3

Datatypes:

- It is going to specify the size and type of the value stored inside the variable.



Single Valued Datatype:

1) **Integer(int):**
   --- It is a real number without decimal point.

$$-\infty \ldots\ldots -3, -2, -1, 0, +1, +2, +3 \ldots\ldots +\infty$$

Default value : It is an initial value and they are internally equal to False.

Non-default value: The values except default values. They are internally equal to True.

Default value of integer = Zero (0)

- Whenever we want to check what type of value stored inside variable we have to use type() function

type(var/val)

Whenever we want to check the value is default or not we use bool() function.

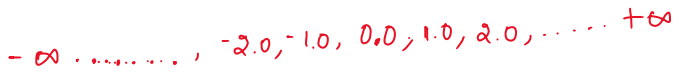
bool(var/val)

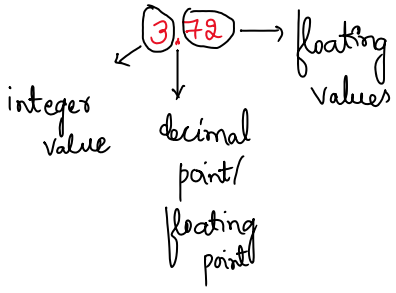
Proof:

```
a = 10
type(a)
<class 'int'>

bool(a)
True
b = 0
bool(b)
False
```

2) Float:

--- It is a real number with decimal point.





Default value of float = 0.0

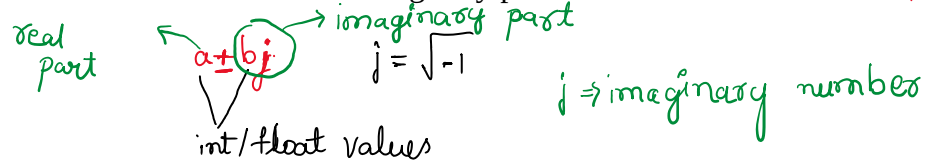
Proof:

```
a = 5.7
type(a)
<class 'float'>
bool(a)
True
b = 0.
c = 0.0
bool(c)
False
.
SyntaxError: invalid syntax
1.
1.0
.8
0.8
```

3) Complex:

--- It is combination of real and imaginary part.

(2+3i)



Default value of complex: 0j

Proof:

```
a = 2+3j
a
(2+3j)
type(a)
<class 'complex'>
bool(a)
True
b = 0j
```

```
bool(b)
False
c = 3j+2
c
(2+3j)
d = 2+j3
Traceback (most recent call last):
  File "<pyshell#8>", line 1, in <module>
    d = 2+j3
NameError: name 'j3' is not defined
e = 2+3J
e
(2+3j)
g = 2+4i
SyntaxError: invalid decimal literal
```

4) Boolean:

--- It consists of only 2 values.

- True
- False

Default value of boolean = False
Non default value of boolean = True

Boolean values are used in 2 scenarios

- It can be used as resultant

```
10>20
False
3>1
True
```

- It is having internal values
  False - 0
  True - 1

```
True + True
2
False + False
0
True + False
1
```

Multivalued Datatype:

5) String:

--- It is defined as the collection of characters enclosed by quotes(" ," " , ''' ''').

Uppercase (A - Z)
Lowercase (a - z)
digits (0 - 9)
special characters

Syntax:

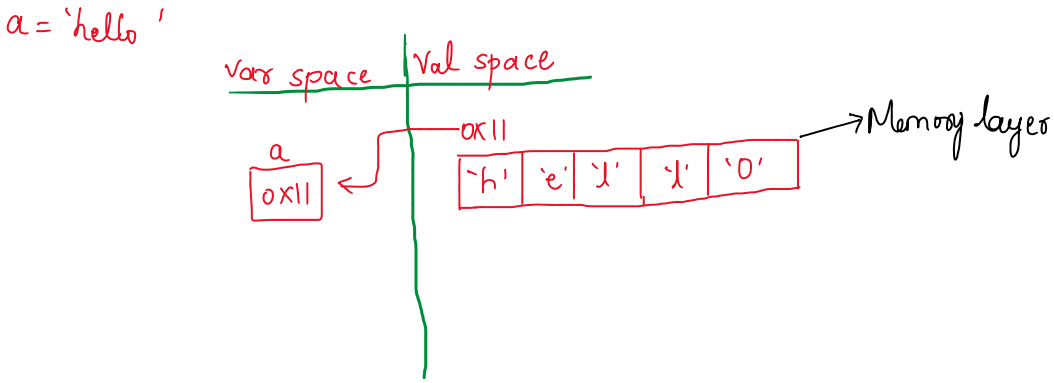
var = 'val1 val2 val3 ..... valn'

When to use 3 different quotes?

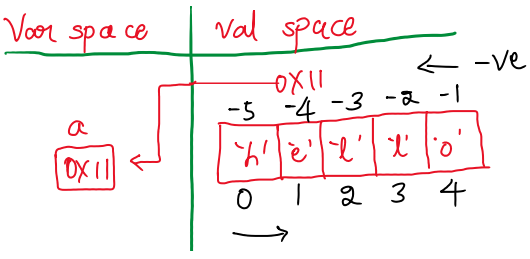
```
a = 'good morning'
a
'good morning'
a = "good morning"
a
'good morning'
a = '''good morning'''
a
'good morning'
b = "Python's Coders are smart"
b
"Python's Coders are smart"
c  = 'h1'
d = 'hi'
dfdsf
Traceback (most recent call last):
  File "<pyshell#10>", line 1, in <module>
    dfdsf
NameError: name 'dfdsf' is not defined
a = '''hi
hello
bye'''
a
'hi\nhello\nbye'
```

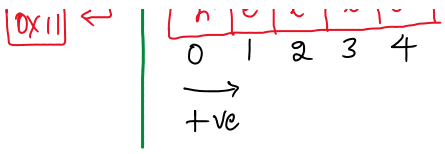
Default value of string: ''

Memory allocation:



Indexing: The process of passing subaddress to the memory block is called Indexing.





Types:

- +ve indexing (left to right)
- -ve indexing (right to left)

- Whenever we want to access the value we have to use the syntax,

var[index]

- Whenever we want to modify the value we have to use the syntax,

var[index]=new_value

If the datatype accepts modification it is called as Mutable Datatype.
If the datatype does not accepts modification it is called as Immutable Datatype.

```
a = 'hello'
a[4]
'o'
a[-1]
'o'
a[4]='e'
Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    a[4]='e'
TypeError: 'str' object does not support item assignment
```

Conclusion : String is immutable datatype.

6) List:

--- It is collection of homogeneous and heterogenous values enclosed by [].
homogeneous : [10,20,30,40]
heterogenous : [10,2.3,4+7j,True,'hello',[10,20]]

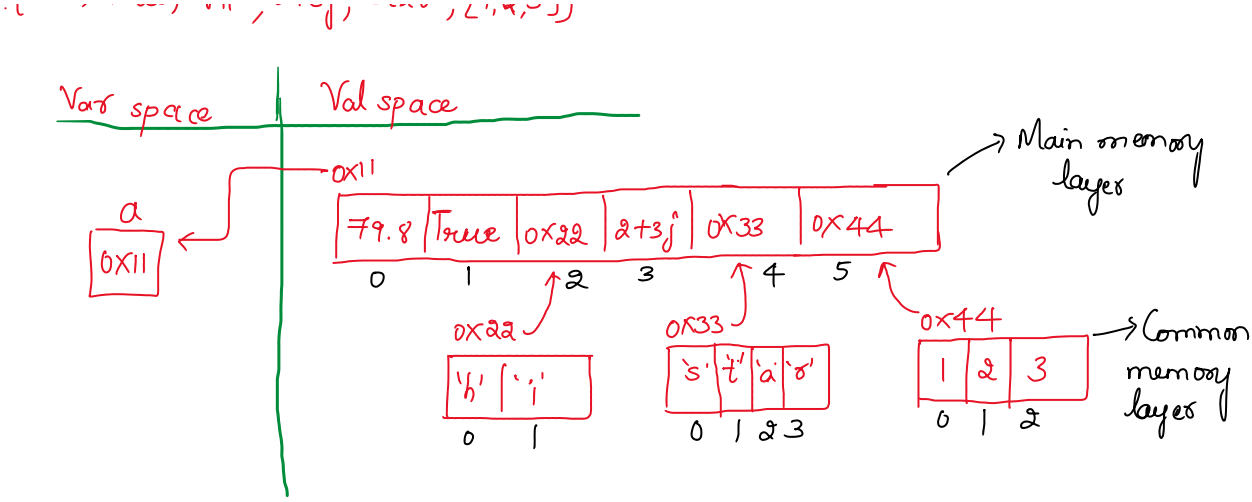
Syntax:

var = [val1, val2, val3, ..., valn]

Default value of list: []

Memory allocation of list:

a = [79.8, True, 'hi', 8+3j, 'Star', [1,2,3]]




```
a = [79.8,True,'hi',2+3j,'Star',[1,2,3]]
a[1]
True
a[4]
'Star'

a[4][3]
'r'
a[5]
[1, 2, 3]
a[5][1]
2

a = [79.8,True,'hi',2+3j,'Star',[1,2,3]]
a[1]
True
a[1]=False
a
[79.8, False, 'hi', (2+3j), 'Star', [1, 2, 3]]
```

Conclusion : List is mutable datatype.

Day-4

Inbuilt functions in list:

- append(): To add a new value to the list.
  syntax: var.append(val)

```
a=[10,20,30]
a.append(40)
a
[10, 20, 30, 40]
a.append(50)
a
[10, 20, 30, 40, 50]
```

- insert(): whenever we want to add the value to the particular index in the list.
  syntax: var.insert(index,val)

```
l=[10,20,30]
l.insert(1,40)
l
[10, 40, 20, 30]
l.insert(40)
Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    l.insert(40)
TypeError: insert expected 2 arguments, got 1


a=[10,20,30]
a.insert(10,40)
a
[10, 20, 30, 40]
```

- pop(): To eliminate a value from the list when we know the index.
  syntax: var.pop() ----- last value get removed
  var.pop(index) ---- remove value based on index position.

```
a=[10,20,30,40]
a.pop()
40
a
[10, 20, 30]
a.pop(1)
20
a
[10, 30]
```

- remove(): used to eliminate a value from the list when we don't know the index.
  syntax: var.remove(val)

```
a=[10,20,30,40]
a.remove(30)
a
[10, 20, 40]
a.remove(30)
Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    a.remove(30)
ValueError: list.remove(x): x not in list
a.remove(20)
a
[10, 40]
```

7) Tuple:

--- It is collection of homogeneous and heterogenous values enclosed by parenthesis ().

Syntax:

var = (val1, val2, ..., valn)

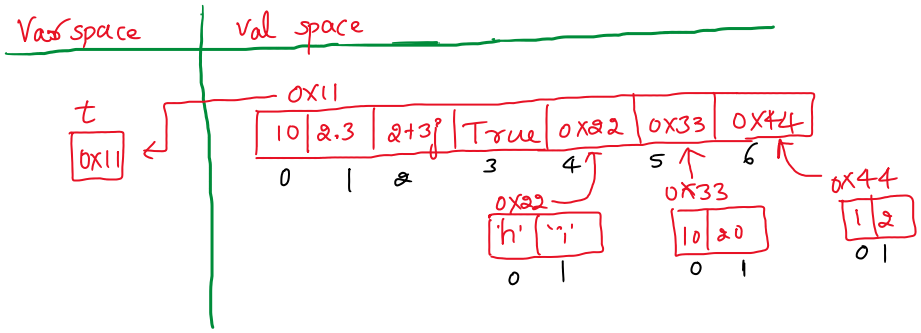
var = (val1,)

```
a = (10,)
type(a)
<class 'tuple'>
```

Default value of Tuple = ()

Memory allocation:

t = (10,2.3,2+3j,True,'hi',[10,20],(1,2))




```
t = (10,2.3,2+3j,True,'hi',[10,20],(1,2))
t[3]
True
t[6]
(1, 2)
t[3]=False
Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    t[3]=False
TypeError: 'tuple' object does not support item assignment
```

Conclusion : Tuple is Immutable datatype. Tuple is the most secured datatype.

8) <u>Set</u>:

--- It is collection of homogeneous and heterogenous values enclosed by parenthesis {}.

<u>Syntax</u>:

var = {val1, val2, ..., valn}

Default value of set = set()

<u>Unique qualities of Set:</u>

- Set is unordered - Order of input is not same as order of output

```
a = [10,20,30,40]
a
[10, 20, 30, 40]
a = {10,20,30,40}
a
{40, 10, 20, 30}
```

- Set will allow only immutable values.

```
s = {10,2.3,4+7j,True,'hi',(10,20)}
s
{'hi', True, 2.3, (4+7j), 10, (10, 20)}
s = {10,2.3,4+7j,True,'hi',[3,4],(10,20)}
Traceback (most recent call last):
  File "<pyshell#6>", line 1, in <module>
    s = {10,2.3,4+7j,True,'hi',[3,4],(10,20)}
TypeError: unhashable type: 'list'
```

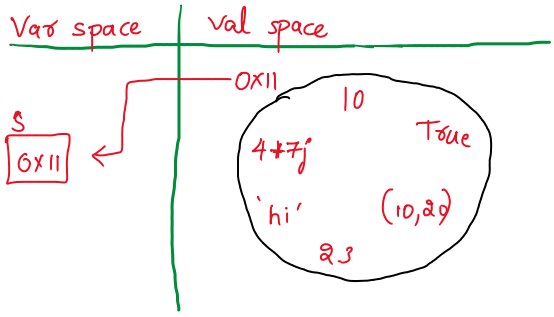
unhashable → mutable

- Set will not allow repeatition.

```
a = [1,2,1,2]
a
[1, 2, 1, 2]
a = (1,2,1,2)
a
(1, 2, 1, 2)
a = {1,2,1,2}
a
{1, 2}
```

Memory allocation:

s = {10,2.3,4+7j,True,'hi',(10,20)}



- Set will not support Indexing.
  Reason - It is unordered.

Inbuilt Functions:

- **add():** It will add the value to the set.
  Syntax: var.add(val)

```
a={10,20,30,40,50}
a
{50, 20, 40, 10, 30}
a.add(60)
a
{50, 20, 40, 10, 60, 30}
```

- **remove():** It will remove the value from the set.
  Syntax: var.remove(val)

```
a
{50, 20, 40, 10, 60, 30}
a.remove(40)
a
{50, 20, 10, 60, 30}
```

- **pop():** It will remove the very first value in the set.
  Syntax: var.pop()

```
a
```

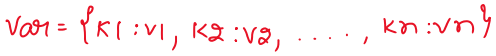
```
{50, 20, 10, 60, 30}
a.pop()
50
a
{20, 10, 60, 30}
a.pop()
20
a
{10, 60, 30}
a.pop()
10
a
{60, 30}
a.pop()
60
a
{30}
a.pop()
30
a
set()
```

Conclusion: Set is mutable datatype.

9) <u>Dictionary</u>:

--- It is a collection of key value pair which are enclosed by {}.

<u>Syntax</u>:



Default value of dictionary = {}

Unique qualities of dictionary :

- Keys should be immutable values and value can be of any datatype.

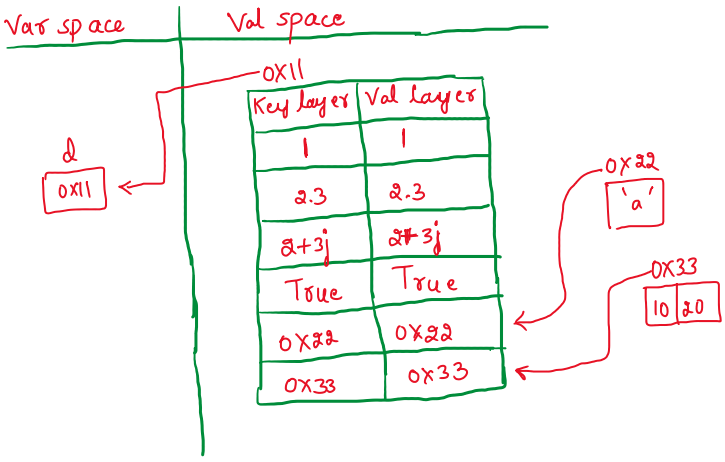
```
d = {1:1,2.3:2.3,2+3j:2+3j,True:True,'a':'a',(10,20):(10,20)}
d
{1: True, 2.3: 2.3, (2+3j): (2+3j), 'a': 'a', (10, 20): (10, 20)}
d = {1:1,2.3:2.3,2+3j:2+3j,True:True,'a':'a',(10,20):(10,20),[10]:[10]}
Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
    d = {1:1,2.3:2.3,2+3j:2+3j,True:True,'a':'a',(10,20):(10,20),[10]:[10]}
TypeError: unhashable type: 'list'
```

- Keys are the only visible layer of the dictionary.
- Keys should be unique. If we try to enter the same key then the old value of that key gets overrided with new value.

```
d={'a':1,'b':2,'c':3,'a':4}
d
{'a': 4, 'b': 2, 'c': 3}
```

Memory allocation:

d = {1:1,2.3:2.3,2+3j:2+3j,True:True,'a':'a',(10,20):(10,20)}



- Dictionary will not support for Indexing.

- if we want to access the value we have to use the syntax,
  var[key]

```
a={10:20,2.3:7,(2+4j):[10,20],True:'a','hi':{10,20},(10,20):{'a':10}}
a
{10: 20, 2.3: 7, (2+4j): [10, 20], True: 'a', 'hi': {10, 20}, (10, 20): {'a': 10}}
a[(2+4j)]
[10, 20]
a[True]
'a'
a['hi']
{10, 20}
```

If we want to modify or add the value, we can use the syntax
var[key]=new_value

```
a={10:20,2.3:7,(2+4j):[10,20],True:'a','hi':{10,20},(10,20):{'a':10}}
a[(10,20)]
{'a': 10}
a[(10,20)]={'b':20}
a
{10: 20, 2.3: 7, (2+4j): [10, 20], True: 'a', 'hi': {10, 20}, (10, 20): {'b': 20}}
a[3.5]='hello'
a
{10: 20, 2.3: 7, (2+4j): [10, 20], True: 'a', 'hi': {10, 20}, (10, 20): {'b': 20}, 3.5: 'hello'}
```

Conclusion: Dictionary is mutable datatype.

Slicing:
--- The process of extracting the group of values from the collections.

Syntax:

var = [SI : EI ± I : Updation]

Slicing supports only for string, list, tuple
Reason - It is depending on Indexing.

- Whenever we are traversing / extracting the values from left to right

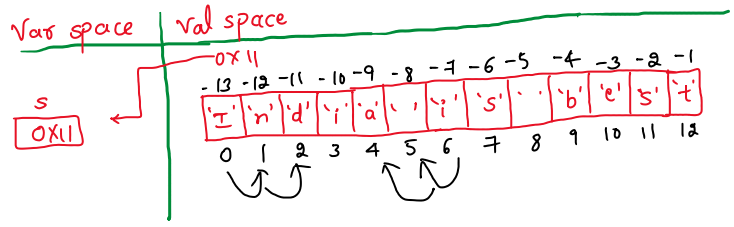
EI + I

- Whenever we are traversing / extracting the values from right to lx`eft

EI -1

Example:

s = 'India is best'



'India' ⇒ s[0:4+1:+1]

'best' ⇒ s[9:12+1:+1]

'Idai' ⇒ s[0:6+1,+2]

'sbs' ⇒ s[7:11+1:+2]

'aidnI' ⇒ s[-9:-13-1:-1]

'tseb' ⇒ s[-1:-4-1:-1]

'iadI' ⇒ s[-7:-13-1:-2]

'sbs' ⇒ s[-2:-6-1:-2]

Proof:

s = 'India is best'
s[0:5:1]
'India'
s[9:13:1]
'best'
s[0:7:2]
'Idai'
s[7:12:2]
'sbs'
s[-9:-14:-1]
'aidnI'
s[-1:-5:-1]
'tseb'
s[-7:-14:-1]
'i aidnI'
s[-7:-14:-2]
'iadI'
s[-2:-7:-2]
'sbs'

Slicing shortcuts:

- If SI is 0 or SI is -1
  var[:EI+-1:updation]
- If EI is the end of collection
  var[SI::updation]
- If updation is +1
  var[SI:EI:]

Proof:

```
s = 'India is best'
s[0:12+1:1]
'India is best'
s[::]
'India is best'
s[::-1]
'tseb si aidnI'
```

Day-5

Type Casting/ Type conversion:

--- The process of converting the data from one type to another.

Syntax:

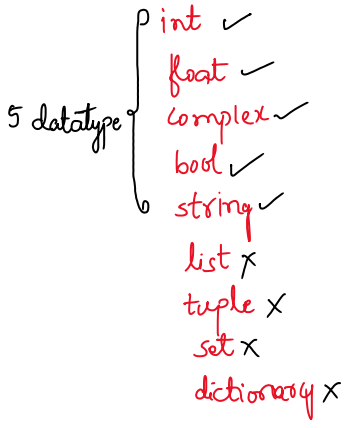
destinationtype(source-var)

Note:

- From SVDT to MVDT conversion only string is supported.
- From MVDT to SVDT conversion only bool is supported.

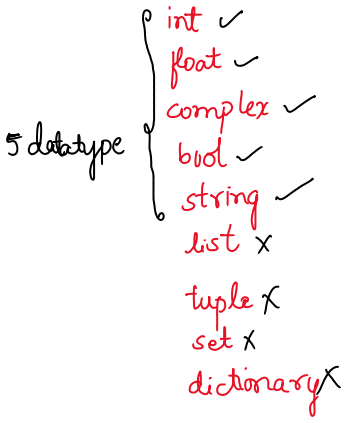
1) Converting int to other datatypes:

```
a = 10
float(a)
10.0
complex(a)
(10+0j)
complex(a,a)
(10+10j)
bool(a)
True
str(a)
'10'
list(a)
Traceback (most recent call last):
  File "<pyshell#6>", line 1, in <module>
    list(a)
TypeError: 'int' object is not iterable
tuple(a)
Traceback (most recent call last):
  File "<pyshell#7>", line 1, in <module>
    tuple(a)
TypeError: 'int' object is not iterable
set(a)
Traceback (most recent call last):
  File "<pyshell#8>", line 1, in <module>
    set(a)
TypeError: 'int' object is not iterable
dict(a)
Traceback (most recent call last):
  File "<pyshell#9>", line 1, in <module>
    dict(a)
TypeError: 'int' object is not iterable
```



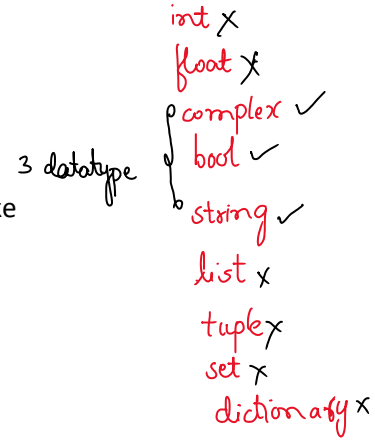
2) Converting float to other datatypes:

```
a = 2.3
int(a)
2
float(a)
2.3
complex(a)
(2.3+0j)
complex(a,a)
(2.3+2.3j)
bool(a)
True
str(a)
'2.3'
list(a)
Traceback (most recent call last):
  File "<pyshell#7>", line 1, in <module>
    list(a)
TypeError: 'float' object is not iterable
tuple(a)
Traceback (most recent call last):
  File "<pyshell#8>", line 1, in <module>
    tuple(a)
TypeError: 'float' object is not iterable
set(a)
Traceback (most recent call last):
  File "<pyshell#9>", line 1, in <module>
    set(a)
TypeError: 'float' object is not iterable
dict(a)
Traceback (most recent call last):
  File "<pyshell#10>", line 1, in <module>
    dict(a)
TypeError: 'float' object is not iterable
```



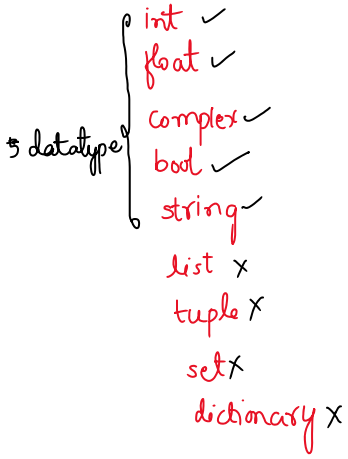
3) Converting complex to other datatypes:

```
a = 2+3j
int(a)
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    int(a)
TypeError: int() argument must be a string, a bytes-like
object or a real number, not 'complex'
float(a)
Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
    float(a)
TypeError: float() argument must be a string or a
real number, not 'complex'
bool(a)
True
str(a)
'(2+3j)'
list(a)
```



```
Traceback (most recent call last):
  File "<pyshell#5>", line 1, in <module>
    list(a)
TypeError: 'complex' object is not iterable
tuple(a)
Traceback (most recent call last):
  File "<pyshell#6>", line 1, in <module>
    tuple(a)
TypeError: 'complex' object is not iterable
set(a)
Traceback (most recent call last):
  File "<pyshell#7>", line 1, in <module>
    set(a)
TypeError: 'complex' object is not iterable
dict(a)
Traceback (most recent call last):
  File "<pyshell#8>", line 1, in <module>
    dict(a)
TypeError: 'complex' object is not iterable
```

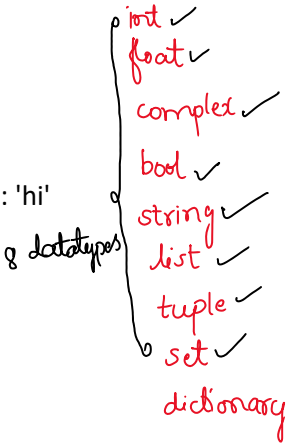
4) Converting bool to other datatypes:




```
a= True
int(a)
1
float(a)
1.0
complex(a)
(1+0j)
complex(a,a)
(1+1j)
bool(a)
True
str(a)
'True'
list(a)
Traceback (most recent call last):
  File "<pyshell#7>", line 1, in <module>
    list(a)
TypeError: 'bool' object is not iterable
tuple(a)
Traceback (most recent call last):
  File "<pyshell#8>", line 1, in <module>
    tuple(a)
TypeError: 'bool' object is not iterable
set(a)
Traceback (most recent call last):
  File "<pyshell#9>", line 1, in <module>
    set(a)
TypeError: 'bool' object is not iterable
dict(a)
Traceback (most recent call last):
  File "<pyshell#10>", line 1, in <module>
    dict(a)
TypeError: 'bool' object is not iterable
```

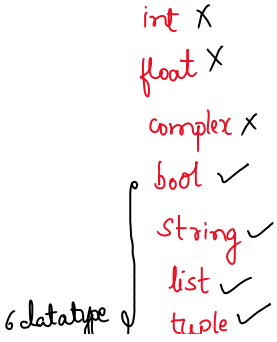
5) Converting string to other datatypes:

```
a = 'hi'
int(a)
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    int(a)
ValueError: invalid literal for int() with base 10: 'hi'
a = '78'
int(a)
78
float(a)
78.0
complex(a)
(78+0j)
complex(a,a)
Traceback (most recent call last):
  File "<pyshell#6>", line 1, in <module>
    complex(a,a)
TypeError: complex() can't take second arg if first is a string
complex(a,78)
Traceback (most recent call last):
  File "<pyshell#7>", line 1, in <module>
    complex(a,78)
TypeError: complex() can't take second arg if first is a string
complex(78,a)
Traceback (most recent call last):
  File "<pyshell#8>", line 1, in <module>
    complex(78,a)
TypeError: complex() second arg can't be a string
bool(a)
True
list(a)
['7', '8']
tuple(a)
('7', '8')
set(a)
{'7', '8'}
dict(a)
Traceback (most recent call last):
  File "<pyshell#13>", line 1, in <module>
    dict(a)
ValueError: dictionary update sequence element #0 has length 1; 2 is required
```

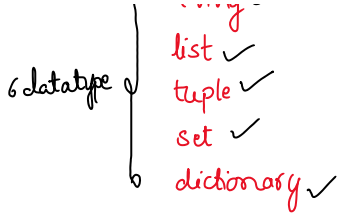


6) Converting list to other datatypes:

```
a = [10,20]
int(a)
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    int(a)
TypeError: int() argument must be a string,
a bytes-like object or a real number, not 'list'
float(a)
Traceback (most recent call last):
```

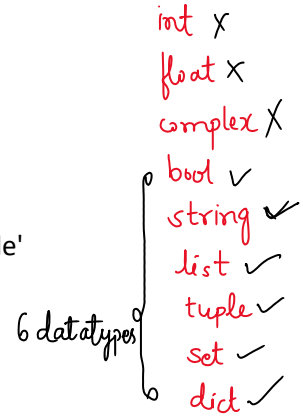


```
a bytes-like object or a real number, not 'list'
float(a)
Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
    float(a)
TypeError: float() argument must be a string
or a real number, not 'list'
complex(a)
Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    complex(a)
TypeError: complex() first argument must be a string or a number, not 'list'
bool(a)
True
str(a)
'[10, 20]'
complex(a)
Traceback (most recent call last):
  File "<pyshell#6>", line 1, in <module>
    complex(a)
TypeError: complex() first argument must be a string or a number, not 'list'
tuple(a)
(10, 20)
set(a)
{10, 20}
dict(a)
Traceback (most recent call last):
  File "<pyshell#9>", line 1, in <module>
    dict(a)
TypeError: cannot convert dictionary update sequence element #0 to a sequence
a = ['ab',[10,20],(1,2)]
dict(a)
{'a': 'b', 10: 20, 1: 2}
```



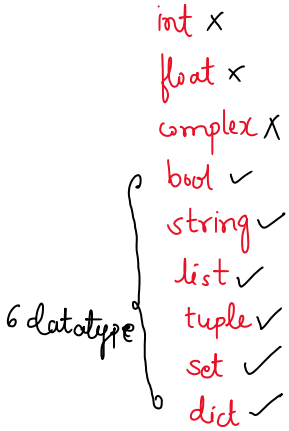
7) Converting tuple to other datatypes:

```
a = (1,2,3)
int(a)
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    int(a)
TypeError: int() argument must be a string,
a bytes-like object or a real number, not 'tuple'
float(a)
Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
    float(a)
TypeError: float() argument must be a string or
a real number, not 'tuple'
complex(a)
Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    complex(a)
TypeError: complex() first argument must be a string or a number, not 'tuple'
bool(a)
True
```



```
str(a)
'(1, 2, 3)'
list(a)
[1, 2, 3]
set(a)
{1, 2, 3}
dict(a)
Traceback (most recent call last):
  File "<pyshell#8>", line 1, in <module>
    dict(a)
TypeError: cannot convert dictionary update sequence element #0 to a sequence
a = ('hi',[1,2],(7,8))
dict(a)
{'h': 'i', 1: 2, 7: 8}
```

8) Converting set to other datatypes:

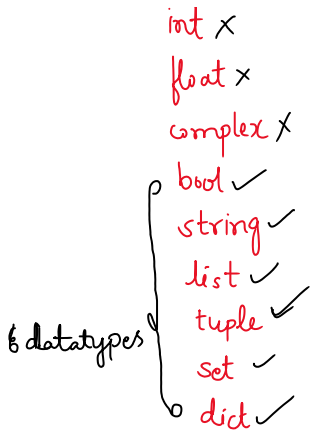



```
a = {1,2,3}
int(a)
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    int(a)
TypeError: int() argument must be a string,
a bytes-like object or a real number, not 'set'
float(a)
Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
    float(a)
TypeError: float() argument must be a string or
 a real number, not 'set'
complex(a)
Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    complex(a)
TypeError: complex() first argument must be a string or a number, not 'set'
bool(a)
True
str(a)
'{1, 2, 3}'
list(a)
[1, 2, 3]
tuple(a)
(1, 2, 3)
dict(a)
Traceback (most recent call last):
  File "<pyshell#8>", line 1, in <module>
    dict(a)
TypeError: cannot convert dictionary update sequence element #0 to a sequence
a = {'ab',(10,20)}
dict(a)
{'a': 'b', 10: 20}
```

9) Converting dictionary to other datatypes:


```
d={'a':10,'b':20,'c':30}
int(d)
Traceback (most recent call last):
```

```
d={'a':10,'b':20,'c':30}
int(d)
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    int(d)
TypeError: int() argument must be a string,
 a bytes-like object or a real number, not 'dict'
float(d)
Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
    float(d)
TypeError: float() argument must be a string
or a real number, not 'dict'
complex(d)
Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    complex(d)
TypeError: complex() first argument must be a string or a number, not 'dict'
bool(d)
True
str(d)
"{'a': 10, 'b': 20, 'c': 30}"
list(d)
['a', 'b', 'c']
list(d.values())
[10, 20, 30]
list(d.items())
[('a', 10), ('b', 20), ('c', 30)]
tuple(d)
('a', 'b', 'c')
tuple(d.values())
(10, 20, 30)
tuple(d.items())
(('a', 10), ('b', 20), ('c', 30))
set(d)
{'b', 'a', 'c'}
set(d.values())
{10, 20, 30}
set(d.items())
{('c', 30), ('b', 20), ('a', 10)}
dict(d)
{'a': 10, 'b': 20, 'c': 30}
```



Copy Operations:

--- The process of copying the content from one variable to another variable.

Objective : After copying the content we are doing modification for source variable and checking whether it is affecting the destination variable or not.

For modification we use - Indexing

Types:

- General copy / Normal copy
- Shallow copy
- Deep copy

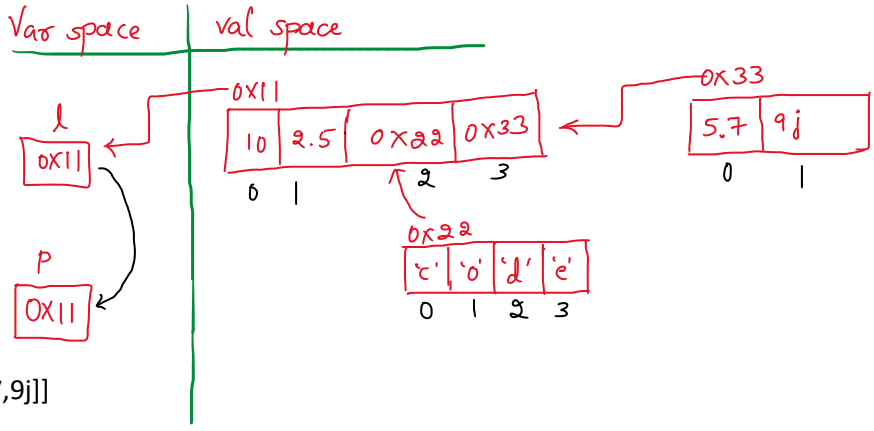
1) **General Copy**:
--- It will copy the variable space of one variable to another.

Syntax:

dest_var = source_var

Memory allocation:

l = [10,2.5,'Code',[5.7,9j]]
p = l




```
l = [10,2.5,'Code',[5.7,9j]]
p = l
l
[10, 2.5, 'Code', [5.7, 9j]]
p
[10, 2.5, 'Code', [5.7, 9j]]
l[0]
10
l[0] = 55
l
[55, 2.5, 'Code', [5.7, 9j]]
p
[55, 2.5, 'Code', [5.7, 9j]]
id(l)
2100232167296
id(p)
2100232167296
```

Conclusion : Modification done with respect to source variable will affect the destination variable.

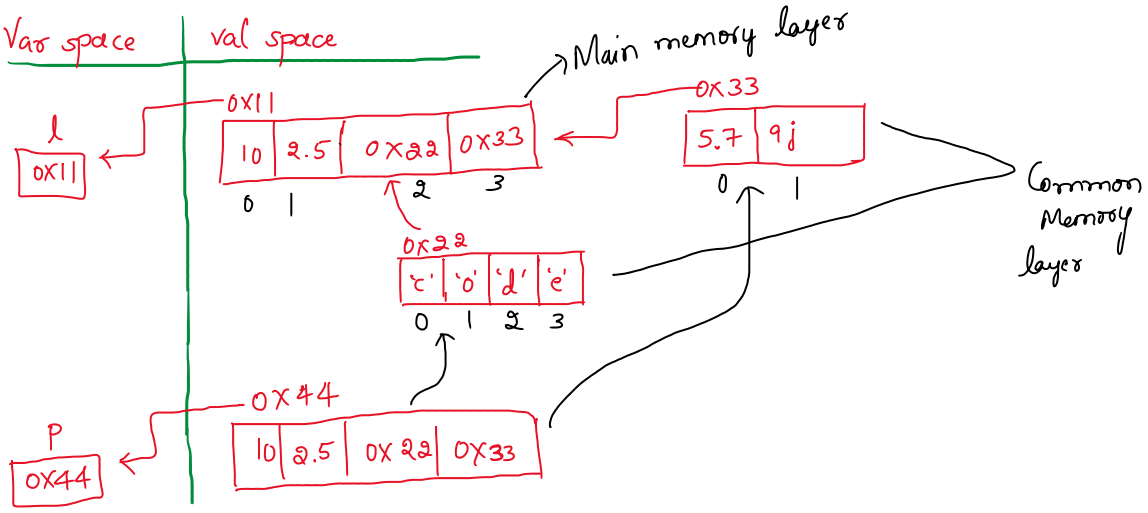
1) **Shallow Copy**:
--- It will copy the main memory layer of value space of one variable to another.

Syntax:

dest_var = source_var.copy()

Memory allocation:

l = [10,2.5,'Code',[5.7,9j]]
p = l.copy()




```
l = [10,2.5,'Code',[5.7,9j]]
p = l.copy()
l
[10, 2.5, 'Code', [5.7, 9j]]
p
[10, 2.5, 'Code', [5.7, 9j]]
id(l)
2971226709888
id(p)
2971278821696
l[1]
2.5
l[1]=15
l
[10, 15, 'Code', [5.7, 9j]]
p
[10, 2.5, 'Code', [5.7, 9j]]
l[3]
[5.7, 9j]
l[3][0]
5.7
l[3][0] = 6.0
l
[10, 15, 'Code', [6.0, 9j]]
p
[10, 2.5, 'Code', [6.0, 9j]]
id(l[3])
2971278690496
id(p[3])
2971278690496
```

Conclusion : Modification done with respect to linear collection of source variable will not affect the destination variable but Modification done with respect to nested collection of source variable will affect the destination variable.

2) <u>Deep Copy</u>:

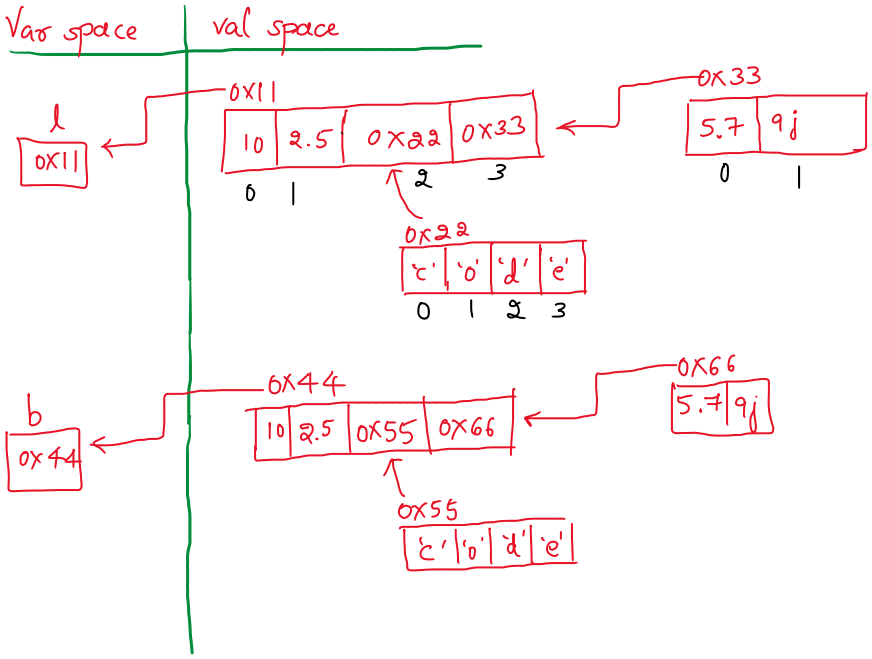
--- It will copy the entire content of value space of one variable to another.

Syntax:

```
import copy
dest_var =copy.deepcopy(source_var)
```

Memory allocation:

l = [10,2.5,'Code',[5.7,9j]]
import copy
b = copy.deepcopy(l)



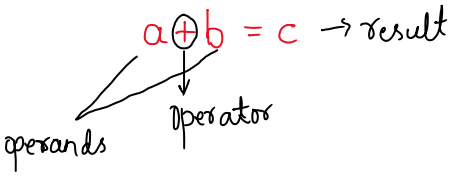

```
l = [10,2.5,'Code',[5.7,9j]]
import copy
b = copy.deepcopy(l)
l[1]
2.5
l[1]=5.9
l
[10, 5.9, 'Code', [5.7, 9j]]
b
[10, 2.5, 'Code', [5.7, 9j]]
id(l)
3025779802240
id(b)
3025779803200
l[3][1]
9j
l[3][1] = 2+3j
l
[10, 5.9, 'Code', [5.7, (2+3j)]]
b
[10, 2.5, 'Code', [5.7, 9j]]
id(l[3])
3025779524864
id(b[3])
```

3025779802944

Conclusion : Modification done with respect to linear collection of source variable will not affect the destination variable and Modification done with respect to nested collection of source variable will not affect the destination variable.

Day-6

Operators:
--- They are the special symbols which is used to perform operations.



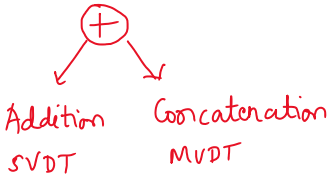
Types:

- Arithmetic Operator
- Logical Operator
- Relational Operator
- Bitwise Operator
- Assignment Operator
- Membership Operator
- Identity Operator

1) Arithmetic Operator :

- **Addition(+):**

OP1 + OP2



For SVDT:

```
10+20
30
2.3+4.7
7.0
(2+4j) + (7+3j)
(9+7j)
True + False
1
17+9.2+(2+3j)+True
(29.2+3j)
```

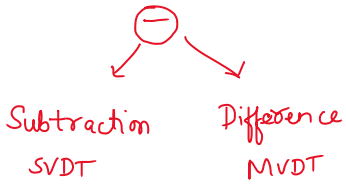
For MVDT:

```
'hi' +'hello'
'hihello'
[10,20,30] + [40, 50]
[10, 20, 30, 40, 50]
(1,2,3)+(4,5,6)
(1, 2, 3, 4, 5, 6)
{1,2,3}+{4,5}
Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    {1,2,3}+{4,5}
```

TypeError: unsupported operand type(s) for +: 'set' and 'set'
{'a':10}+{'b':20}
Traceback (most recent call last):
  File "<pyshell#4>", line 1, in <module>
    {'a':10}+{'b':20}
TypeError: unsupported operand type(s) for +: 'dict' and 'dict'

- **Subtraction(-):**

OP1 - OP2



For SVDT:

4-1
3
2.3 - 1.8
0.4999999999999998
(5+3j) - (3+7j)
(2-4j)
2.3 - 1.87
0.4299999999999997
2.3 - 1.8
0.4999999999999998
2.3 - 1.80
0.4999999999999998
True - False
1

For MVDT:

'hi'-'hello'
Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    'hi'-'hello'
TypeError: unsupported operand type(s) for -: 'str' and 'str'
[10,20,30]-[10,20]
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    [10,20,30]-[10,20]
TypeError: unsupported operand type(s) for -: 'list' and 'list'
(10,20,30)-(10,20)
Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
    (10,20,30)-(10,20)
TypeError: unsupported operand type(s) for -: 'tuple' and 'tuple'
{10,20,30}-{10,20}
{30}
{'a':10}-{'a':20}
Traceback (most recent call last):
  File "<pyshell#4>", line 1, in <module>
    {'a':10}-{'a':20}
TypeError: unsupported operand type(s) for -: 'dict' and 'dict'

- **Multiplication(*):**

OP1 * OP2

For SVDT:

2+2+2+2+2+2+2+2

For SVDT:

OP1 * OP2


```
2+2+2+2+2+2+2+2
16
2*8
16
2.3 * 6.5
14.95
(2+3j) * (4+7j)
(-13+26j)
True * False
0
```

$(2+3j) * (4+7j)$

$\Rightarrow 2*(4+7j) + 3j*(4+7j)$

$\Rightarrow 8+14j+12j+21j^2$

$\Rightarrow 8+26j+21(-1)$

$\Rightarrow 8+26j-21$

$\Rightarrow (-13+26j)$

$j = \sqrt{-1}$

$j^2 = (\sqrt{-1})^2$

$j^2 = -1$

For MVDT:

```
a = 'hi' * 'hello'
Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    a = 'hi' * 'hello'
TypeError: can't multiply sequence by non-int of type 'str'
'hi' * 3
'hihihi'
[10,20,30]*4
[10, 20, 30, 10, 20, 30, 10, 20, 30, 10, 20, 30]
(1,2,3)* 3
(1, 2, 3, 1, 2, 3, 1, 2, 3)
{1,2,3}*2
Traceback (most recent call last):
  File "<pyshell#4>", line 1, in <module>
    {1,2,3}*2
TypeError: unsupported operand type(s) for *: 'set' and 'int'
{'a':10}*2
Traceback (most recent call last):
  File "<pyshell#5>", line 1, in <module>
    {'a':10}*2
TypeError: unsupported operand type(s) for *: 'dict' and 'int'
```

- **Division(): --- It is applicable for only SVDT**

  - **True Division (/):**
    --- It will give quotient as the answer but output will be Displayed in the form of float values.

$2\overline{)10}(5$ → quotient

10

0 → remainder

OP1 / OP2

```
6 / 3
2.0
3.3 / 1.7
1.9411764705882353
(2+3j) / (5+7j)
(0.41891891891891897+0.013513513513513507j)
True / False
Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    True / False
ZeroDivisionError: division by zero
```

```
False / True
0.0
'sakshi' / 'sakshi'
Traceback (most recent call last):
  File "<pyshell#5>", line 1, in <module>
    'sakshi' / 'sakshi'
TypeError: unsupported operand type(s) for /: 'str' and 'str'
```

- **Floor Division (//):**
  --- It will give quotient as the answer but output will be
  Displayed in the form of int values But if float number is there output will be in the form of float.

OP1 // OP2

```
6//3
2
2.3//1.7
1.0
2.3//1
2.0
(2+3j)//(2+3j)
Traceback (most recent call last):
  File "<pyshell#4>", line 1, in <module>
    (2+3j)//(2+3j)
TypeError: unsupported operand type(s) for //: 'complex' and 'complex'
True//True
1
True/True
1.0
```

- **Modulus (%):**
  --- It will give remainder as the answer and output will be displayed as it is.

OP1 % OP2

```
10%3
1
2.3 % 4.5
2.3
(2+3j) % (2+3j)
Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
    (2+3j) % (2+3j)
TypeError: unsupported operand type(s)
for %: 'complex' and 'complex'
True % True
0
2.3%4.6
2.3
```

4.5) 0.3 (
↳ remainder

Day-7

- **Power(**): - It is used only for SVDT**
  --- It is used to find the power of the given value.

Proof:

2**3
8
4.5 **2
20.25
(1+2j)**3
(-11-2j)
True **10
1
5**True
5
7**False
1
3.2 ** (2+3j)
(-9.626669455748416-3.490678328027451j)

2) Logical Operator :

- **Logical AND(and):**When any one of the Operand is False, the result will be False.
  Syntax:

  op1 and op2

  Conditions to remember:

  * if op1 is False
    out = op1
  * if op1 is True
    out = op2

3 and 7 ⇒ 7

0.0 and 98 ⇒ 0.0

2 and 0j ⇒ 0j

[9.8] and 3.4 ⇒ 3.4

2 and 4 and 7 ⇒

Proof:

3 and 7
7
0.0 and 9.8
0.0
2 and 0j
0j
[9.8] and 3.4
3.4
2 and 4 and 7
7

- **Logical OR(or):**When any one of the Operand is True, the result will be True.
  Syntax:

  op1 or op2

Conditions to remember:

* if OP1 is False
  Out = OP2

* if OP1 is True
  out = OP1

'tea' or 'coffee' ⇒ 'tea'

6 or 9 ⇒ 6

0 or 123 ⇒ 123

[ ] or '' ⇒ ''

78 or 34 or 5 ⇒ 78

(67 and 23) or (89 and 45)
⇒ 23 or 45
⇒ 23

Proof:

'tea' or 'coffee'
'tea'
6 or 9
6
0 or 123
123
[] and ''
[]
[] or ''
''
78 or 34 or 5
78
67 and 23 or 89 and 45
23

- **Logical Not(not):** It gives negation(opposite) of values.
  Syntax:

not (OP)

Conditions to remember:

* if OP is False
  out = True

* if OP is True
  out = False

not(3) ⇒ False

not('') ⇒ True

not(5j) ⇒ False

not(0.0) ⇒ True

Proof:

```
not(3)
False
not('')
True
not(5j)
False
not(0.0)
True
```

3) **Relational Operator/Comparision Operator:**
--- Output will always be in the form of boolean.

- **Equal to (==):**
--- Used to check whether both the operands are same or not.

OP1 == OP2

For SVDT - It compares values,
For MVDT - It compares values and datatype.

```
2 == 2.0
True
True == 1
True
3.4 == 4.5
False
[1,2,3] == (1,2,3)
False
{'a':20} == {'a':30}
False
[1,2,3] == [3,2,1]
False
{1,2,3,4} == {4,3,2,1}
True
```

- **Not Equal to (!=):**
--- Used to check whether both the operands are Opposite or not.
Or It works exactly opposite or Equal to operator.

For SVDT - It compares values,
For MVDT - It compares values and datatype.

Proof:

```
45 != 45
False
4!=9
True
'' != ' '
True
(1) != (1,)
True
```

- **Greater than (>):** It will not support for complex and dictionary.
--- Used to check whether the operand1 is greater than operand2 or not.

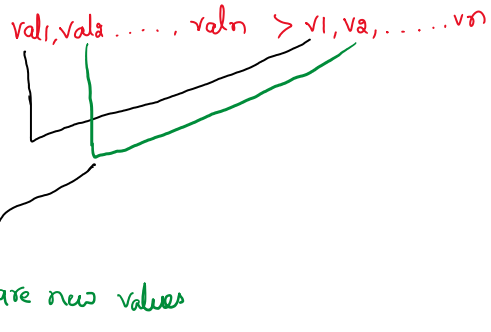
- **Greater than (>):** It will not support for complex and dictionary.
  --- Used to check whether the operand1 is greater than operand2 or not.

  op1 > op2

  For SVDT - It compares values,

  For MVDT :

  * if val1 > v1
    True
  * if val1 < v1
    False
  * if val1 == v1 ⇒ It will go and compare new values



  For string, ASCII values will be compared.

```
ord('A')
65
ord('B')
66
ord('Z')
90
ord('a')
97
ord('b')
98
ord('z')
122
```

  For Set : All the values of operand2 should be present in operand1, Then it will give True or it will give False.

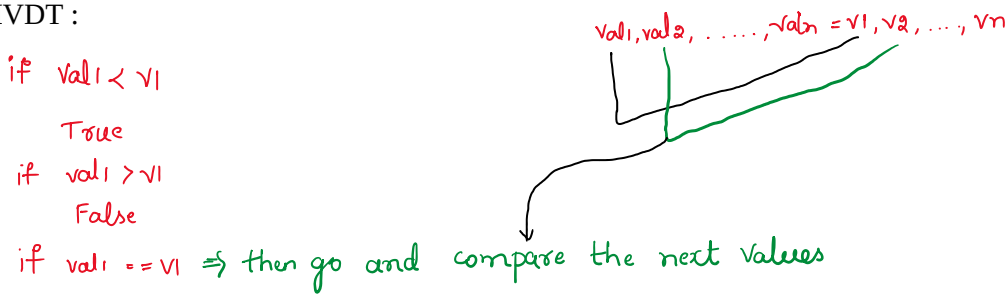
  Proof:

```
25 > 12
True
3.4 > 4.5
False
True > 1
False
'python' > 'abc'
True
ord('p')
112
ord('a')
97
'ABC' > 'abc'
False
'aBC' > 'abc'
False
[10,20,30] > [10,20,10]
True
[10,20,30] > [10,20,30]
False
(1,2,3,4) > (True,2,3,4.0)
False
```

{10,20,30,40}>{10,20}
True
{10,20}>{10,20}
False
{10,20,30}>{10,20}
True
[10,20,30] >[10,20]
True

- **Lesser than (<):** It will not support for complex and dictionary.
  --- Used to check whether the operand1 is lesser than operand2 or not.

For MVDT :



For string, ASCII values will be compared.

For Set : All the values of operand1 should be present in operand2, Then it will give True or it will give False.

Proof:

2 < 2.0
False
True < 1.7
True
'data' < 'engineer'
True
[10,20,30] < [10,20,10]
False
(12,24,56) < (12,335,456)
True
{12,True,'hello',0.0} <{'1','hello',3*4,False,78}
False
{12,True,'hello',0.0} <{1,'hello',3*4,False,78}
True

- **Greater than or Equal to (>=):** It will not support for complex and dictionary.
  --- Used to check whether the operand1 is greater than or equal to operand2 or not.

For SVDT - It compares values,
For string, ASCII values will be compared.
For Set : All the values of operand2 should be present in operand1, Then it will give True or it will give False.

Proof:

2>=2

True
2.3 >= 1.9
True
False >= 0.0
True
False >= 0.000000008
False
'great'>='great'
True
{1,2,3}>{1,2,3}
False
{1,2,3}>={1,2,3}
True
[10,20,30]>=[30,10,20]
False
(1,2,7)>=(1,2,2*4-True)
True

- **Lesser than or Equal to (<=):** It will not support for complex and dictionary.
  --- Used to check whether the operand1 is lesser than or equal to operand2 or not.

For SVDT - It compares values,
For string, ASCII values will be compared.
For Set : All the values of operand1 should be present in operand2, Then it will give True or it will give False.

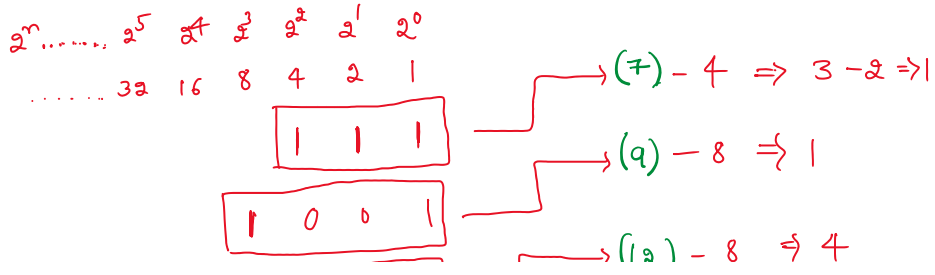
Proof:

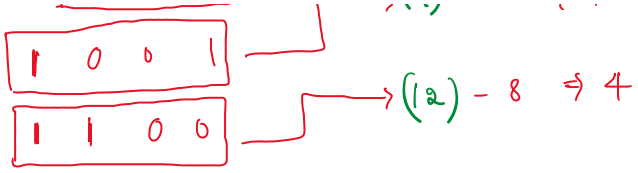
0.0 <= False
True
1 <= 4
True
True <= False
False
'python' <='java'
False
[5,4,3,2]<=[5,4,True+2,10%4]
True
(1,2,3) <= (False**0,True**2+1,123%10)
True
{3,'hello',8,5}<={24,56,'hello',5,1+2,8,True}
True

Day-8

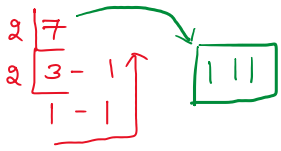
4) **<u>Bitwise Operator</u> : -- it is applicable only for integer**
   --- It will consider binary values perform operation bit by bit .

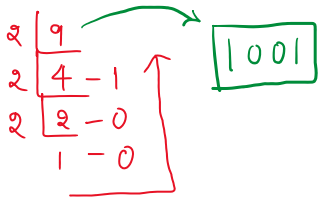
   We can get binary values using 3 ways,
   - **Binary scale method.**

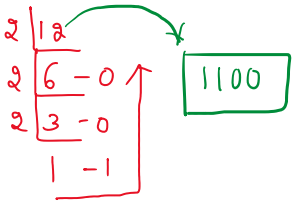




- **Divide by 2 method.**



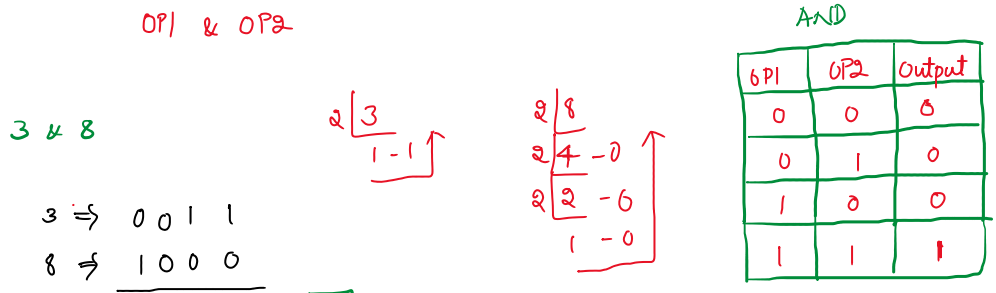


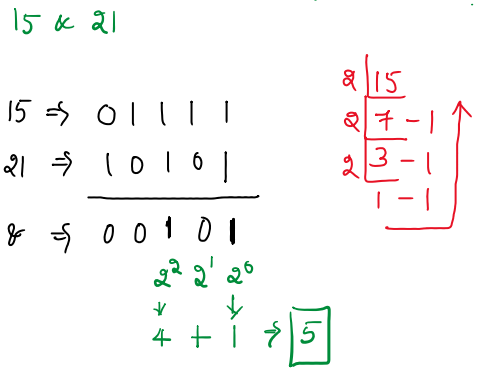


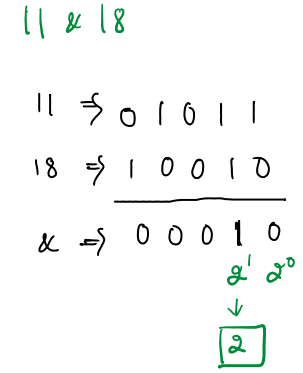
- **Using bin() function.**

```
bin(7)
'0b111'
bin(9)
'0b1001'
bin(12)
'0b1100'
```

- **Bitwise AND(&):**







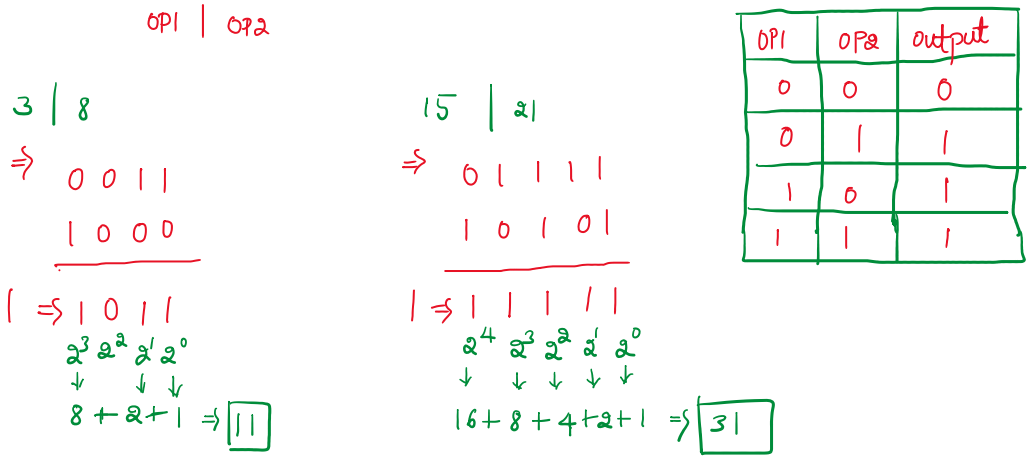
Proof:

3 & 8
0
15 & 21
5
11 & 18
2

Assignment : Get the bitwise and values of

- 17 & 19
- 32 & 27
- 14 & 21
- 13 & 24

- **Bitwise OR(|):**

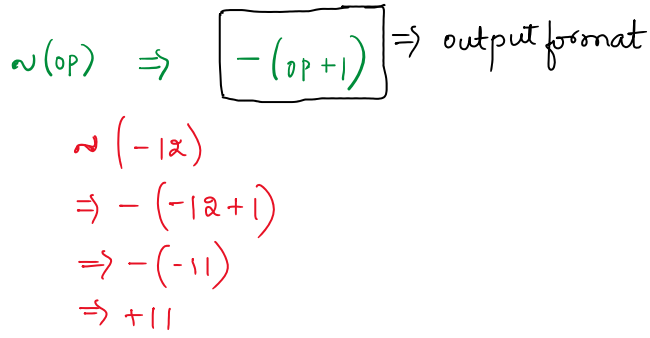


Program:

3 | 8
11
15 | 21
31

Assignment : Take the rest of the previous bitwise and questions and do it for bitwise or.

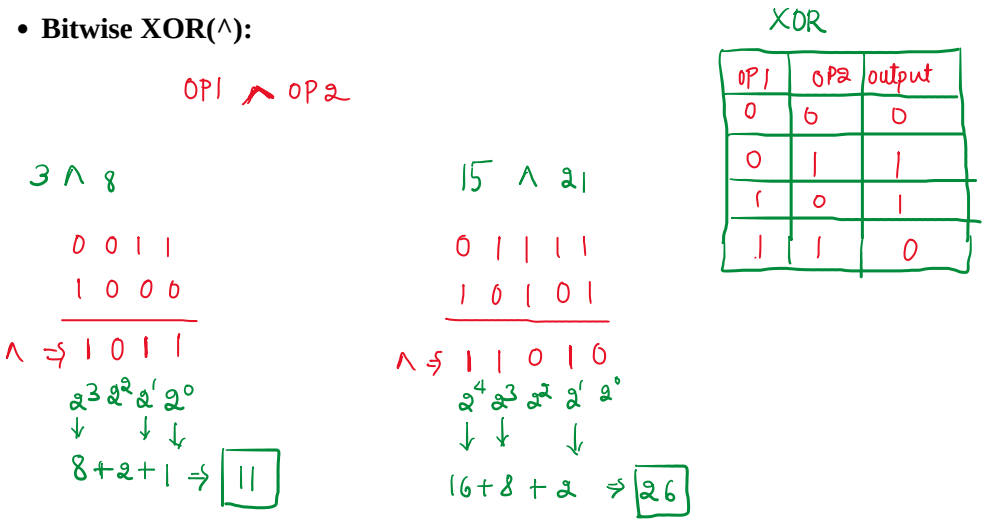
- **Bitwise NOT(~):**



Proof:

~3
-4
~-12
11

- **Bitwise XOR(^):**

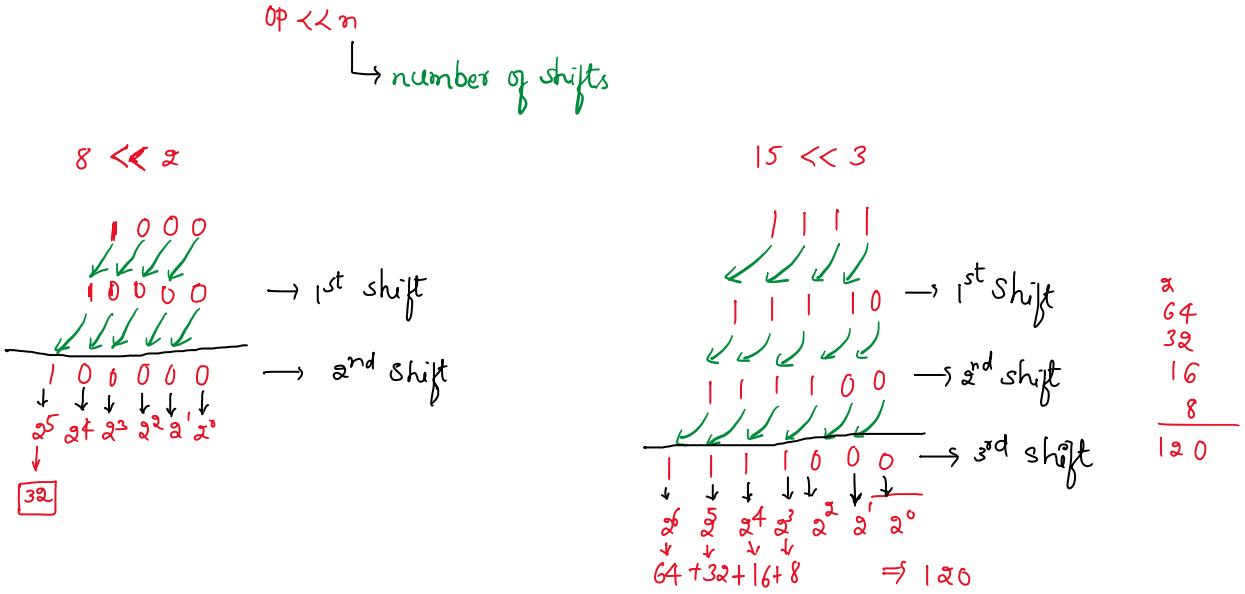


Program:

```
3 ^ 8
11
15 ^ 21
26
```

Assignment : Take the rest of the previous bitwise and questions and do it for bitwise XOR.

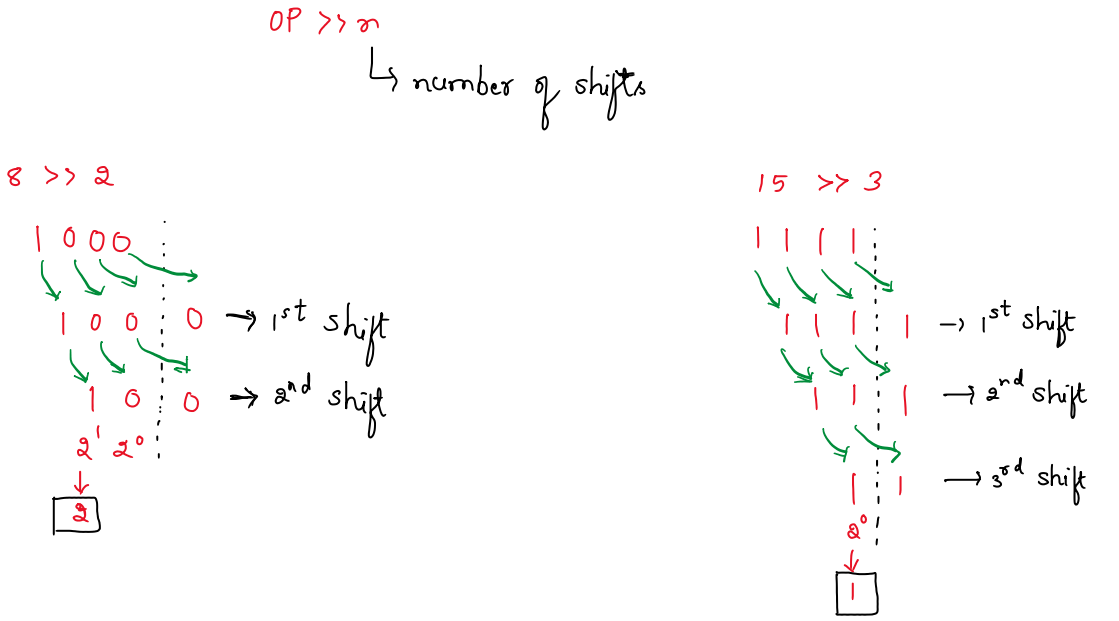
- **Bitwise left shift operator(<<):**



Proof:

```
8 << 2
32
15 << 3
120
```

- **Bitwise right shift operator(>>):**



Proof:

```
8>>2
2
15 >> 3
1
```

15 << 2

111100

15 >>2

11

5) Assignment Operator(=):

---- It is used to assign the value to a variable.

a=a+b	-----	**a+=b**
a=a-b	-----	**a-=b**
a=a*b	-----	**a*=b**
a=a/b	-----	**a/=b**
a=a//b	-----	**a//=b**
a=a%b	-----	**a%=b**
a=ab**	-----	**a**=b**
a=a&b	-----	**a&=b**
a=a|b	-----	**a|=b**
a=a^b	-----	**a^=b**
a=a<<b	-----	**a<<=b**
a=a>>b	-----	**a>>=b**

Proof:

```
a = 10
a = a + 3
a
13
a = a - 3
a
10
a += 3
a
13
a -= 3
a
10
```

7) Membership operator:

--- It is used to check if the value present inside the collection or not.

- **in** --- values present in collection. It returns True if the value is present inside the collection, else return False.

```
'hi' in  'hi hello'
True
10 in {'a':10,'b':20}
False
[1] in [1,2,3]
False
1 in [1,2,3]
True
[1] in [[1],2,3]
True
True in {1,2,3,4}
True
(1) in (1,2,3)
True
(1)
1
(1,) in (1,2,3)
False
```

- **not in** --- values not present in collection. It returns True if the value is not present inside the collection, else return False.

```
45 not in [12,34,56]
True
p not in 'python'
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    p not in 'python'
NameError: name 'p' is not defined
'p' not in 'python'
False
'hl' not in 'hello'
True
'' not in 'python'
False
[] not in [10,20]
True
() not in (10,20)
True
[1] not in [1,2,3]
True
```

8) Identity Operator:

--- It is used to check whether both the operands are sharing the same address or not.

Types:

- is --- It will return True if both the operands are sharing the same address or else return False.

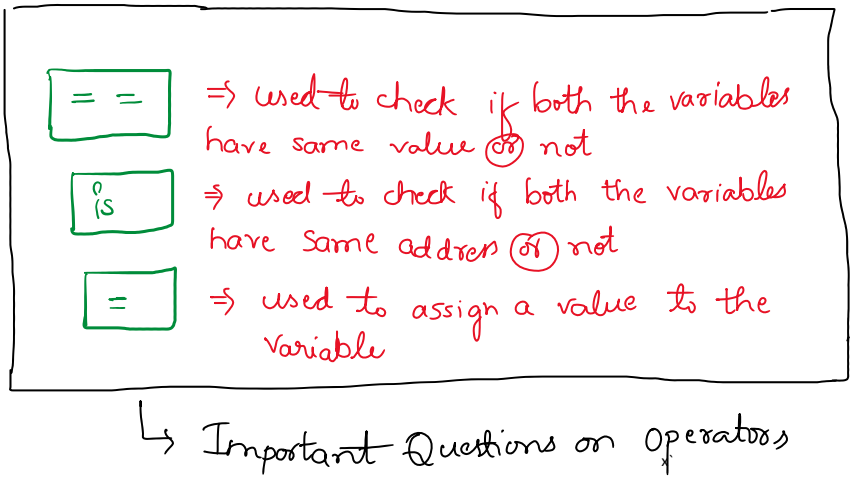
Syntax: OP1 is OP2

- is not --- It will return True if both the operands are not sharing the same address or else return False.

  Syntax: OP1 is not OP2

  Proof:

```
a=10
b=20
c=10
a is b
False
a is c
True
b is c
False
a is not b
True
a is not c
False
b is not c
True
```



Day-9

Input & Output Statements:

- **Input Statements:**
  --- We should never allow the user to modify the code instead just allow them to access the code.

  Syntax:

  var = input(' message ')

```
a = input('Enter the value: ')
print(type(a))
```

Note: input() function will take input by default in the form of string.

If we want other datatype values like int, float, complex then we have to use type casting.

d = dest-type(input('message'))

```
a = int(input('Enter the value: '))
print(type(a))
```

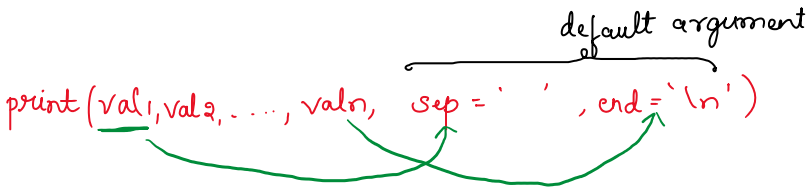
```
a = list(input('Enter the value: '))
print(a)
print(type(a))
'''
Enter the value: sakshi
['s', 'a', 'k', 's', 'h', 'i']
<class 'list'>'''
```

Whenever we want to typecast to collection datatype they will store each character of string as a value. In that condition use ,


```
a = eval(input('Enter the value: '))
print(a)
print(type(a))
```

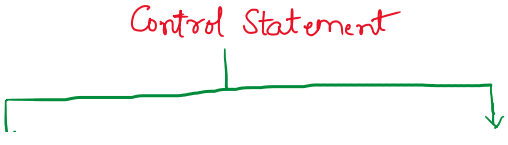
- **Output Statements:**
  --- Whenever we want to display the output in the screen we use this.

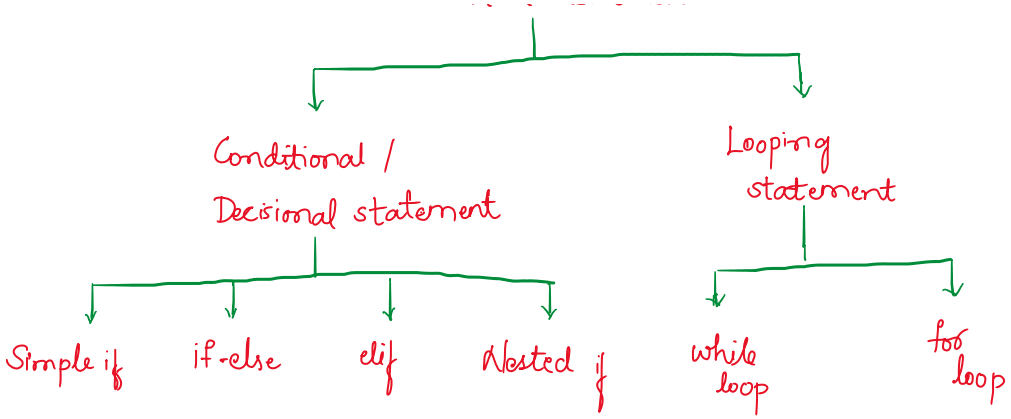
Syntax:




```
# Output Statement
'''
print(10,20,30,40)
print(10,20,30,40,sep='@')
print(10,20,30,40,sep='#',end='&')   '''


# programs on Input and Output Statements

# Program to add 2 numbers
'''
a = int(input('Enter the value1: '))
b = int(input('Enter the value2: '))
print(a+b)  '''

# Program to print the square of the number
'''
n = int(input('Enter the number: '))
print(n**2) '''

# Program to extract the last character from the string.
'''
a = input('Enter the string: ')
print(a[-1])   '''

# Program to reverse the string.
'''
a = input('Enter the string: ')
print(a[::-1])  '''
```

Control Statements:
--- It is used to control the flow of execution.

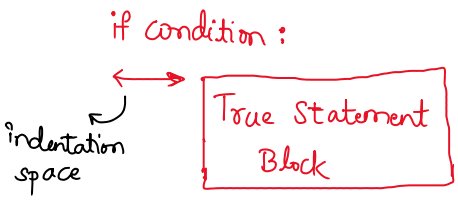




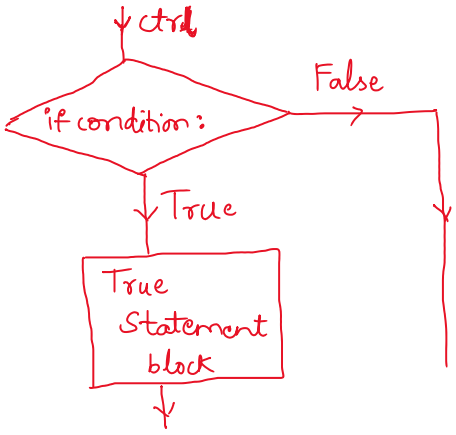
Conditional Statement:
--- It is used to control the flow of execution based on conditions.

1) **Simple if**:
--- It is a keyword which is used to check the condition and it will execute the True Statement Block if the condition is True or else it will ignore the True statement block if the condition is False.

Syntax:



Flow diagram:




```
# Simple if

# Program to check whether the number is even or not
'''
n = int(input('Enter the number: '))
if n%2 ==0:
   print('even')  '''

#program to check whether the string has exactly 5 characters in it.
'''
s = input('Enter the string: ')
if len(s)==5:
   print('String has exactly 5 characters')
'''

# program to check whether the number is greater than 200
'''
n = int(input('Enter the number: '))
if n>200:
   print('Number is greater than 200')
'''

# Program to print the square of the number only if it is mutiple of 3
'''
```

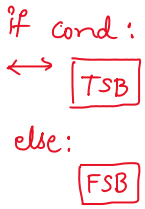
```
n = int(input('Enter the number: '))
if n%3 ==0:
   print(n**2)
'''
# program to check whether the number is 2 digit number
'''
n = int(input('Enter the number: '))
if n>9 and n<100:
   print('2 digit number')
'''
'''
n = int(input('Enter the number: '))
if 9<n<100:
   print('2 digit number')
'''
'''
n = int(input('Enter the number: '))
if 10<=n<=99:
   print('2 digit number')'''

# program to check whether the character is Uppercase
'''
ch = input('Enter the character: ')
if ch >='A' and ch<='Z':
   print('Uppercase')
'''
'''
ch = input('Enter the character: ')
if 'A'<=ch<='Z':
   print('Uppercase')
'''
```

Day-10

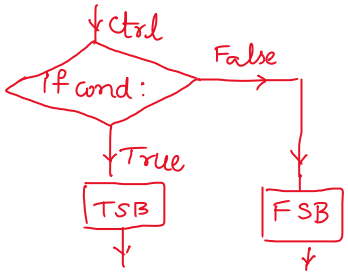
2) <u>if else</u>:

--- Advance version of if. It is used to check the condition, if the condition is True it will execute the True Statement block, or else It will execute the False Statement block.

<u>Syntax</u>:



<u>Flow diagram</u>:



Programs:

```
# elif

# Program to check whether the given data is float or not.
'''
data = eval(input('Enter the data: '))
if type(data) == float:
   print('Float datatype')
else:
```

```
    print('Not Float datatype')  '''

# Program to check whether the string is palindrome or not.
# madam, appa, amma, malayalam
'''
a = input('Enter the string: ')
if a == a[::-1]:
    print('Palindrome')
else:
    print('Not Palindrome')   '''

# Program to check whether the character is vowel or not
'''
ch = input('Enter the character: ')
if ch in 'AEIOUaeiou':
    print('Vowel')
else:
    print('Not vowel') '''

# Program to check whether the given data is SVDT or not
'''
data =  eval(input('Enter the data: '))
if type(data) in [int,float,complex,bool]:
    print('S V D T')
else:
    print('M V D T')   '''

# program to check whether the number is positive or not.
'''
n = int(input('Enter the number: '))
if n >= 0:
    print('positive number')
else:
    print('negative number') '''
```

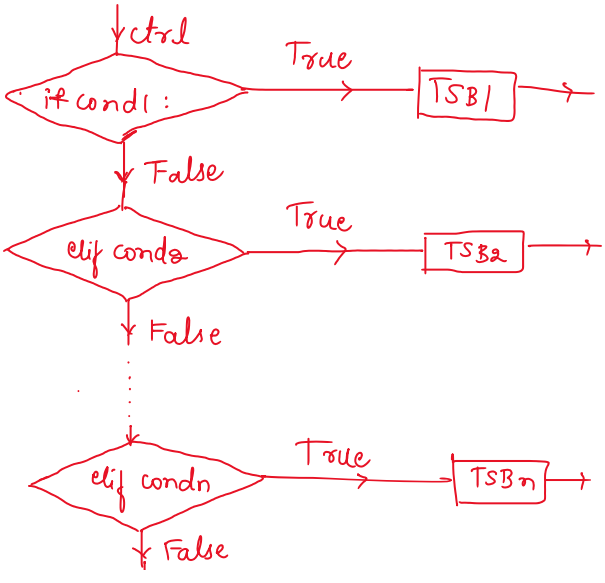
3) elif :

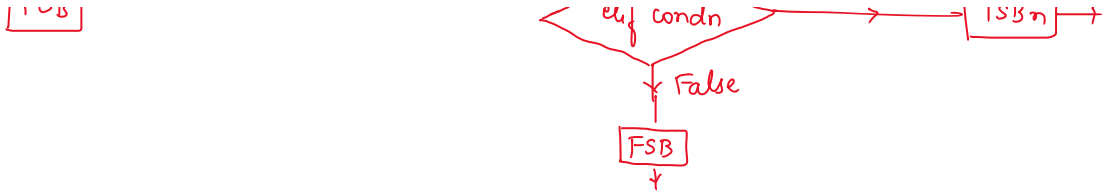
--- It is used to check the multiple conditions at a time. If the condition is True it will execute the Statement block of that particular condition.

Syntax:

```
if condl:
    TSB1
elif condz:
    TSB2
    :
elif condn:
    TSBn
else:
    FSB
```

Flow diagram:





Programs:

```
# elif

# Program to define the relation between 2 integers
'''
a = int(input('Enter the number: '))
b = int(input('Enter the number: '))
if a>b:
   print(a,'is greater')
elif a<b:
   print(a,'is smaller')
else:
   print('both are equal')  '''

# Program to check whether the character is uppercase, lowercase, digit or special character
'''
ch = input('Enter the number: ')
if 'A'<=ch<='Z':
   print(ch,' = Uppercase')
elif 'a'<=ch<='z':
   print(ch,' = Lowercase')
elif '0'<=ch<='9':
   print(ch,' = Digits')
else:
   print(ch,' = Special Character')  '''

# Find the greatest among 4 numbers
'''
a = int(input('Enter the number1: '))
b = int(input('Enter the number2: '))
c = int(input('Enter the number3: '))
d = int(input('Enter the number4: '))
if a>b and a>c and a>d:
   print(a,'is greatest')
elif b>a and b>c and b>d:
   print(b,'is greatest')
elif c>a and c>b and c>d:
   print(c,'is greatest')
elif d>a and d>b and d>c:
   print(d,'is greatest')'''
```

Assignment : Find the smallest among 4 numbers

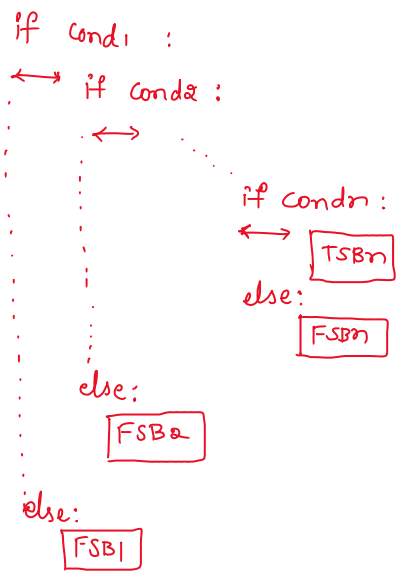
4) Nested if:

--- condition inside another condition.

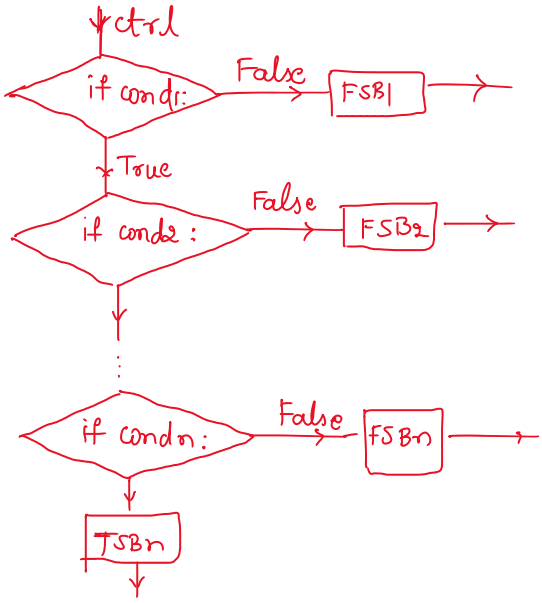
Syntax:

Flow diagram:

Syntax:



Flow diagram:



Programs:

```
# Nested if

# Program to check whether the character is vowel or not.
'''
ch = input('Enter the character: ')
if 'A'<=ch<='Z' or 'a'<=ch<='z':
   if ch in 'aeiouAEIOU':
      print('Vowel')
   else:
      print('Consonant')
else:
   print('Not an alphabet')   '''

# Instagram Login page
'''
# Signup
username = 'Sakshi_27'
password = 'Saku@123'

# Login
un = input('Enter the username: ')
if un == username:
   pw = input('Enter the password: ')
   if pw == password:
      print('Login Successful')
   else:
      print('Incorrect Password')
else:
   print('Invalid Username')'''

# Find the greatest among 3 numbers
'''
a = int(input('Enter the number1: '))
b = int(input('Enter the number2: '))
c = int(input('Enter the number3: '))
```

```
if a>b:
   if a>c:
      print(a,'is greatest')
   else:
      print(c,'is greatest')
else:
   if b>c:
      print(b,'is greatest')
   else:
      print(c,'is greatest') '''
```

Assignment :

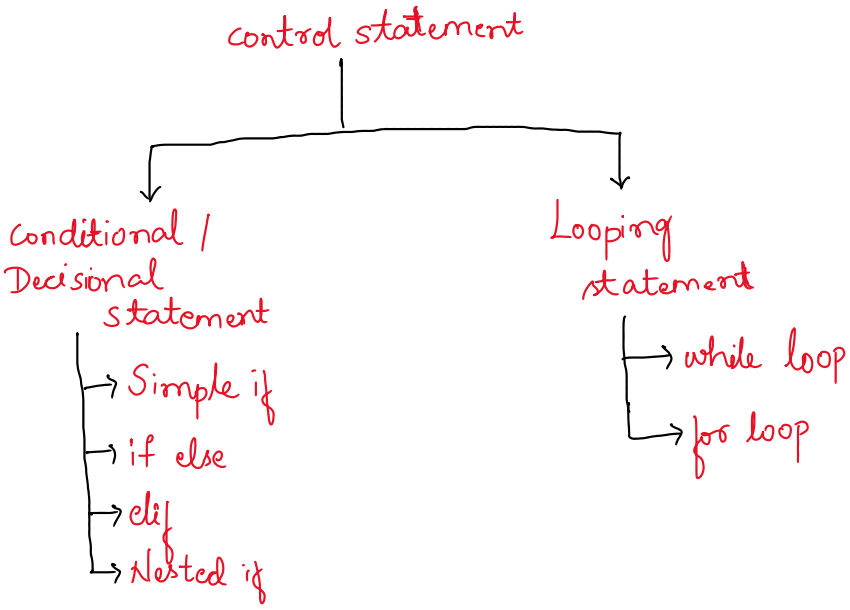
- Find the greatest among 4 numbers
- Find the smallest among 4 numbers

Day-16

Control Statement:

--- It is used to control the flow of execution.

Types:



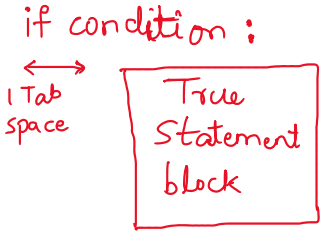
Conditional Statement:

--- It is used to control the flow of execution based on conditions.

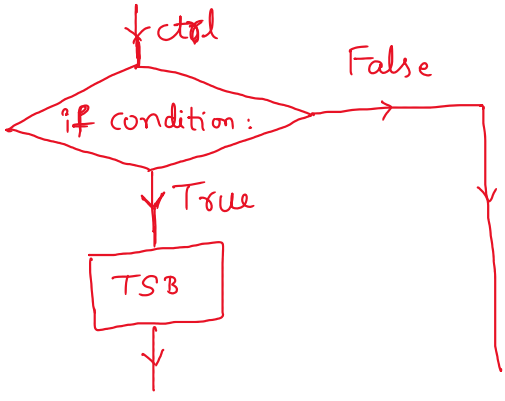
1) Simple if:

--- It is a keyword which is used to check the condition and it will execute the statement block if the condition is True or else it will ignore the statement block.

Syntax:



Flow diagram:



Programs:

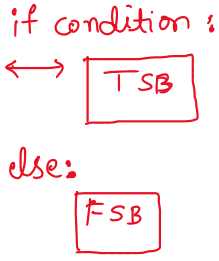
```
# Simple if

# WAP to check whether the number is even.
'''
n = int(input('Enter the number: '))
if  n%2 == 0:
```

```
   print('number is even') '''

# WAP to check whether the string has exactly 5 characters in it.
'''
s = input('Enter the string: ')
if len(s)==5:
   print('string has exactly 5 characters in it')'''

# WAP to check whether the number is greater than 200.
'''
n = int(input('Enter the number: '))
if  n>200:
   print('number is greater than 200')'''

# WAP to print the square of the number only if it is multiple of 3.
'''
n = int(input('Enter the number: '))
if  n%3==0:
   print('square of the number is: ',n**2)'''

# WAP to check whether the number is 2 digit number.
'''
n = int(input('Enter the number: '))
if  n>=10 and n<=99:
   print('number is 2 digit number')'''

# WAP to check if the character is Uppercase.
'''
ch = input('Enter a character: ')
if  'A'<= ch <= 'Z':
   print('character is Uppercase')'''
```

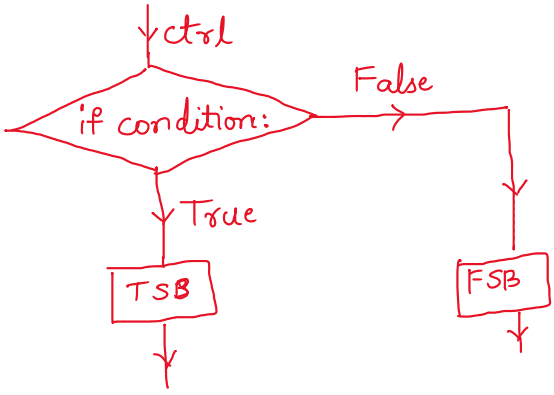
2) if else:

--- It is used to check the condition and it will execute the True Statement block if the condition is True else it will execute the False Statement block.

Syntax:



Flow diagram:



Programs:

```
# if else

# WAP to check the given data is float or not.
```

```
'''
data =  eval(input('Enter the data: '))
if  type(data)==float:
   print('given data is float')
else:
   print('given data is not float')'''


# WAP to check whether the string is palindrome or not.
'''
s = input('Enter the string: ')
if s==s[::-1]:
   print('string is palindrome')
else:
   print('string is not palindrome')'''

# WAP to check whether the given character is vowel or not.
'''
ch = input('Enter the character: ')
if ch in 'aeiouAEIOU':
   print('given character is vowel')
else:
   print('given character is not vowel')'''


# WAP to check whether the given data is SVDT or not.
'''
data = eval(input('Enter the data: '))
if  type(data) in [int, float, complex, bool]:
   print('given data is SVDT')
else:
   print('given data is not SVDT')'''


# WAP to check whether the given integer is 3 digit number or not.
'''
n = abs(int(input('Enter the number: ')))
if  100<=n<=999:
   print('given integer is 3 digit number')
else:
   print('given integer is not 3 digit number')'''
```

Note:

abs (absolute function) - It will convert the negative numbers into positive numbers. If we already have positive number it will keep as it is.

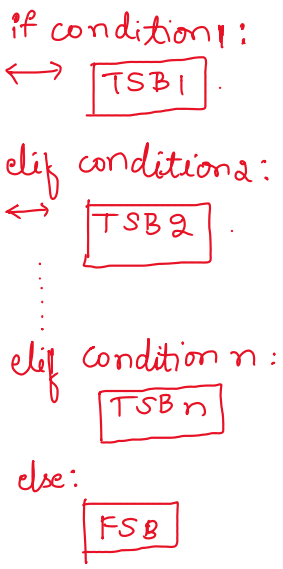
Day-17

3) elif:

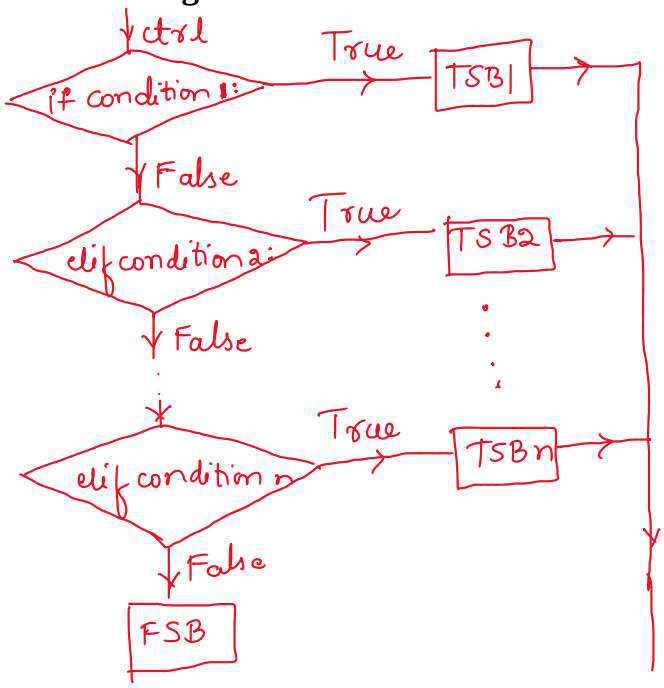
--- Whenever we want to check the multiple conditions and to execute statement blocks of each and every condition we use elif.

Syntax: **Flow Diagram:**

Syntax:



Flow Diagram:



Programs:

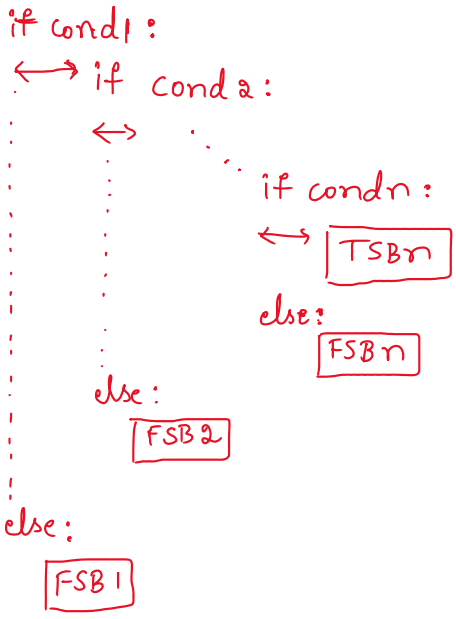
```
# elif

# WAP to find the relation between 2 numbers.
'''
a = int(input('Enter the number1: '))
b = int(input('Enter the number2: '))
if a > b:
   print(a, 'is greater')
elif a < b:
   print(a, 'is lesser')
else:
   print(a, b,'are equal')'''

# WAP to check whether the character is uppercase or lowercase or digits or special characters
'''
ch = input('Enter the character: ')
if 'A'<=ch<='Z':
   print('character is uppercase')
elif 'a'<=ch<='z':
   print('character is lowercase')
elif '0'<=ch<='9':
   print('character is digit')
else:
   print('character is special character')'''

# WAP to check whether the number is single digit or  two digit or three digit or more than 3 digit.
'''
n = abs(int(input('Enter the number: ')))
if 0<=n<=9:
   print('single digit')
elif 10<=n<=99:
   print('two digit')
elif 100<=n<=999:
   print('three digit')
else:
   print('more than three digit')'''
```

```
# WAP to find the greatest among four numbers
'''
a = int(input('Enter the number1: '))
b = int(input('Enter the number2: '))
c = int(input('Enter the number3: '))
d = int(input('Enter the number4: '))
if a>b and a>c and a>d:
   print(a,'is greatest')
elif b>a and b>c and b>d:
   print(b,'is greatest')
elif c>a and c>b and c>d:
   print(c,'is greatest')
else:
   print(d,'is greatest')'''

# Assignment:  WAP to find the smallest among four numbers

# WAP to predict the student result based on the obtained percentage.
'''
per = float(input('Enter the percentage: '))
if per < 0 or per>100:
   print('Invalid result')
elif 70<=per<=100:
   print('Distinction')
elif 60<=per<70:
   print('First Class')
elif 45<=per<60:
   print('Second Class')
elif 35<=per<45:
   print('Just pass')
elif per<35:
   print('Fail')'''
```

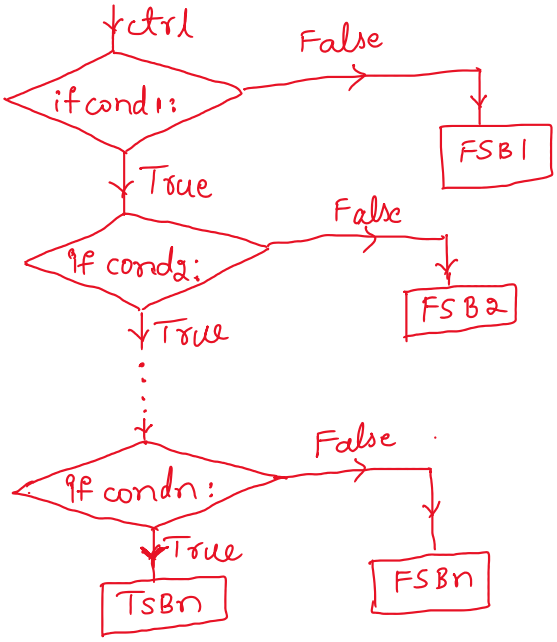
4) Nested if:

--- Whenever it is necessary to check a condition before checking another condition we use Nested if.

Syntax:



Flow diagram:



Programs:

```
# Nested if

# WAP to check whether the given character is vowel or consonant.
'''
s = input('Enter the character: ')
if 'A'<=s<='Z' or 'a'<=s<='z':
   if s in 'aeiouAEIOU':
      print('Vowels')
   else:
      print('Consonants')
else:
   print('character is not alphabet')'''

# WAP to login to Instagram by entering the proper username and password.
'''
username = 'python'
password = 'coders@123'
un = input('Enter the username: ')
pw = input('Enter the password: ')
if un == username:
   if pw == password:
      print('Login Successful')
   else:
      print('Invalid password')
else:
   print('Incorrect username')'''

# WAP to print the greatest among 3 numbers
'''
a = int(input('Enter the number1: '))
b = int(input('Enter the number2: '))
c = int(input('Enter the number3: '))
if a>b:
   if a>c:
      print(a,'is greatest')
   else:
      print(c,'is greatest')
elif b>a:
   if b>c:
      print(b,'is greatest')
   else:
      print(c,'is greatest')'''
```

Day-18

Looping Statement:

--- It is a control statement which will control the flow of execution by repeating the same task again and again.

Types:

- While loop
- For loop

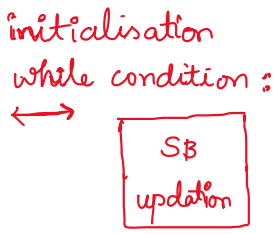
1) While loop:

--- It is used to execute the same set of instructions again and again until the condition become False.

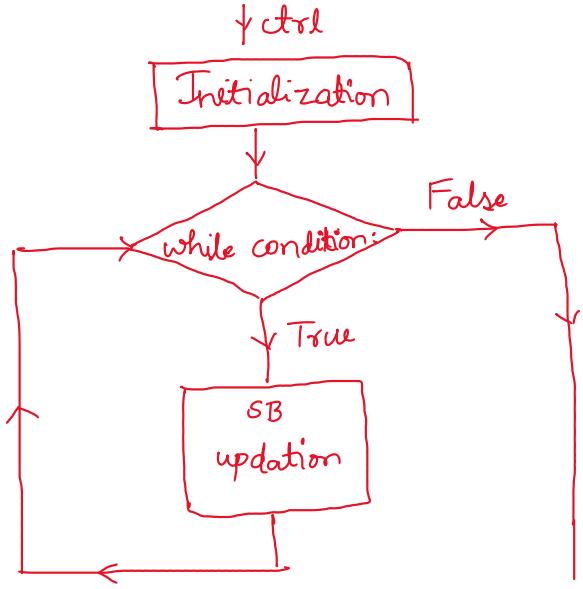
Note:

- **Initialization**
- **Updation**

Syntax:



Flow Diagram:

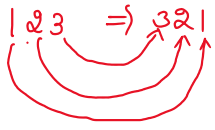


Programs:

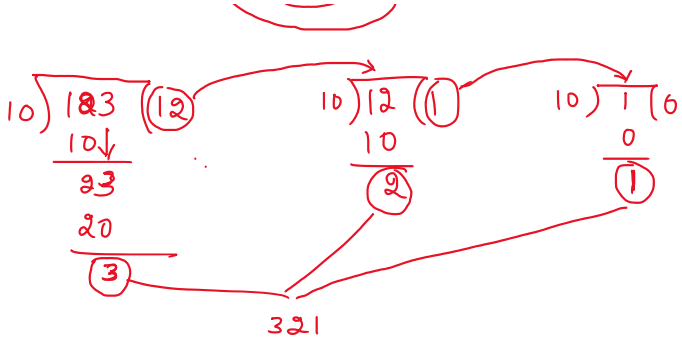
```
# while loop

# WAP to print hello world for 5 times
'''
i = 1
while i<=5:
   print('hello world')
   i = i + 1'''
```


```
# WAP to print first 10 natural numbers.
'''
num = int(input('Enter the number: '))
i = 1
while i <= num:
   print(i)
   i = i + 1 '''
```

```
# WAP to print the  first 10 natural numbers in reverse order
'''
n = int(input('Enter the number: '))
i = n
while i >0:
   print(i)
   i = i - 1'''
```

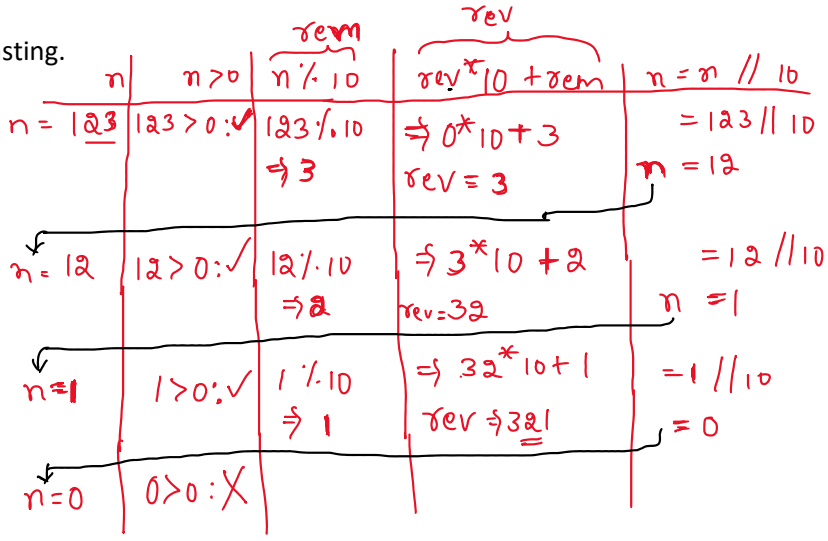


```
    print(i)
    i = i - 1'''

# WAP to print all the even numbers from 1 to 50
'''
n = int(input('Enter the number: '))
i = 1
while i <= 50:
    if i%2==0:
        print(i)
    i = i + 1'''
```




```
'''
n = int(input('Enter the number: '))
i = 2
while i <= 50:
    print(i)
    i = i + 2'''

# WAP to print the sum of n natural numbers
'''
n = int(input('Enter the number: '))
i = 1
res = 0
while i<=n:
    res = res + i
    i = i + 1
    print(res)'''

# WAP to reverse the number without using typecasting.
'''
n = int(input('Enter the number: '))
rev = 0
while n > 0:
    rem = n % 10
    rev = rev*10 + rem
    n = n // 10
print(rev)'''

# WAP to print the product of individual digit
from the number.
'''
n = int(input('Enter the number: '))
prod = 1
while n > 0:
    rem = n%10
    prod = prod * rem
    n = n // 10
print(prod)'''
```

rev=0



143

1*4*3 ⇒ 12

3*4*1 ⇒ 12

Assignment: Do tracing for this program

Day-19

While loop programs

```
# WAP to find the factorial of a given integer / to find the prod of n natural numbers.
# Example:  5!   --- 5*4*3*2*1
# Example: product upto 5 --- 1*2*3*4*5
'''
n = 5
prod = 1
while n>0:
   prod = prod * n
   n = n - 1
print(prod)'''

'''
i = 1
prod = 1
while i<=5:
   prod = prod * i
   i = i + 1
print(prod)'''

# WAP to extract the uppercase character from the string.
'''
st = input('Enter the string: ')
i = 0
out = ''
while i<len(st):
   if 'A'<=st[i]<='Z':
      out = out + st[i]
   i +=1
print(out)'''
```

To toggle the string.

```
ord('A')
65
ord('a')
97
chr(65)
'A'
chr(ord('A')+32)
'a'
chr(ord('B')+32)
'b'
chr(ord('C')+32)
'c'
chr(ord('a')-32)
'A'
chr(ord('b')-32)
'B'
```

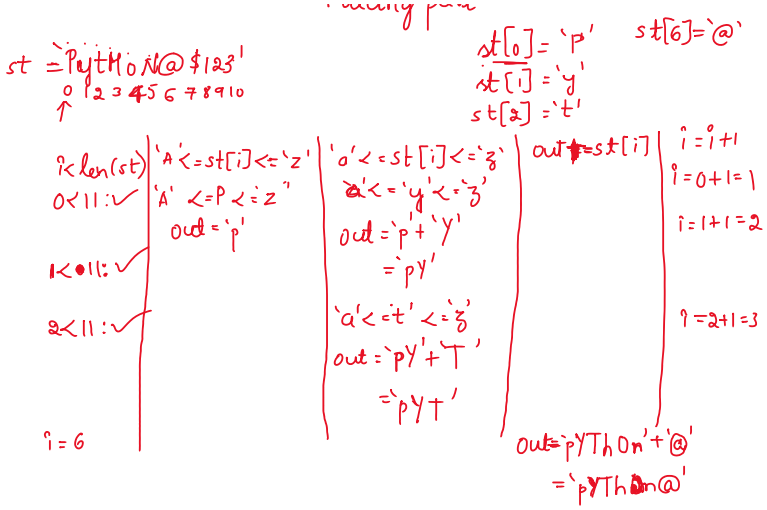
Tracing part

```
# WAP to toggle the string.
'''
st = input('Enter the string: ')
```

st = 'PythON@$123'
0 1 2 3 4 5 6 7 8 9 10

st[0] = 'P' st[6] = '@'
st[1] = 'y'

```
# WAP to toggle the string.
'''
st = input('Enter the string: ')
i = 0
out = ''
while i<len(st):
    if 'A'<=st[i]<='Z':
        out += chr(ord(st[i])+32)
    elif 'a'<=st[i]<='z':
        out += chr(ord(st[i])-32)
    else:
        out += st[i]
    i += 1
print(out)'''
```




```
# WAP to find the sum of all the integers in a list.
'''
l = eval(input('Enter the list: '))
sum = 0
i = 0
while i<len(l):
    if type(l[i]) == int:
        sum += l[i]
    i += 1
print(sum)'''
```

Day-20

For loop:
--- It is self-iterative loop.

Advantage:

- It will allow us to use all the MVDT but in while loop it considers only string, list and tuple.
- No need of initialization and updation.

Range():
--- It is used to create a sequence of integers between the given value.
Syntax:

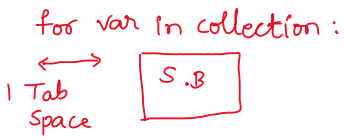
```
range(SV, EV+1,updation)
range(SV, EV-1,updation)
```

- If updation == +1
  range(SV, EV+-1)

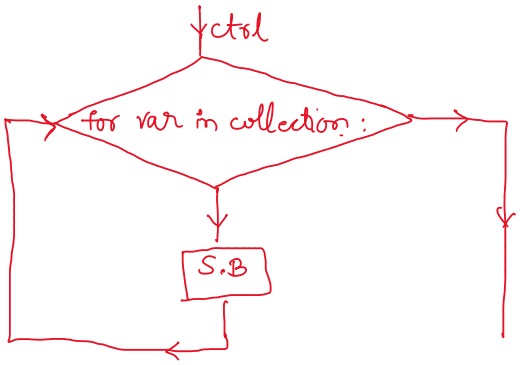
- If SV==0
  range(EV+-1, updation)

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

Syntax:



Flow Diagram:



Programs:

```
# For loop

# Practice programs
'''
for i in [10,2.3,6+7j,78]:
   print(i)'''

'''
for i in (12,34):
   print(i)'''

'''
for i in {17,3.4,78,32}:
   print(i)'''

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

'''
for i in 'sakshi':
   print(i,end=' ')'''

'''
for i in range(1,6):
   print(i) '''

# Actual programs of loop programs

# WAP to find the length of the collection without using len function.
'''
c = eval(input('Enter the collection: '))
count = 0
for i in c:
   count+=1
print(count)'''

# WAP to extract the vowels from the given string.
'''
s = input('Enter the string: ')
out = ''
for i in s:
   if i in 'aeiouAEIOU':
      out += i
print(out)'''

# WAP to replace space by and underscore in a given string.
'''
s = input('Enter the string: ')
out = ''
```

```
for i in s:
   if i == ' ':
      out += '_'
   else:
      out += i
print(out)'''

# WAP to check whether the string is palindrome or not without using slicing.
'''
s = input('Enter the string: ')
rev = ''
for i in s:
   rev = i + rev
if rev == s:
   print('palindrome')
else:
   print('not palindrome')'''

# WAP to remove the duplicates values from the list.
'''
l = eval(input('Enter the list: '))
out = []
for i in l:
   if i not in out:
      out += [i]
print(out)'''
```

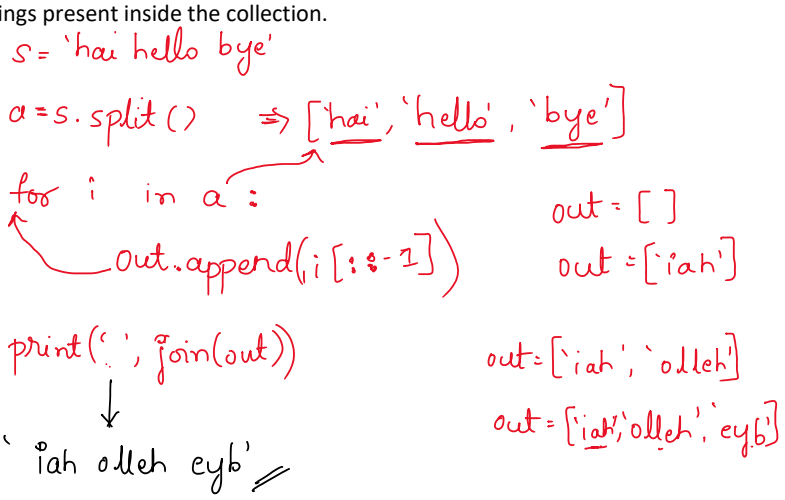
Day-21

```
# Get the following output.
'''
Input : (12,3.4,'hello',2+3j,'python','bye',False)
Output : {'hello': 5, 'python': 6,'bye':3} '''

'''
t = eval(input('Enter the tuple: '))
out = {}
for i in t:
   if type(i) == str:
      out[i] = len(i)
print(out)'''

# Get the following output.
'''
Input : [12,3.4,'hello',2+3j,'python','bye',False]
Output : {'hello': 'ho', 'python': 'pn','bye':'be'} '''

'''
l = eval(input('Enter the list: '))
out = {}
for i in l:
   if type(i) == str:
      out[i] = i[0]+i[-1]
print(out)'''

# Get the following output.
'''
Input : 'aPpLe#123'
Output : {'a':'A', 'P':'p', 'p':'P',  'L':'l', 'e':'E'} '''

'''
s = input('Enter the string: ')
out = {}
for i in s:
   if 'a'<=i<='z':
      out[i] = chr(ord(i)-32)
   elif 'A'<=i<='Z':
```

```
    out[i] = chr(ord(i)+32)
print(out) '''
```

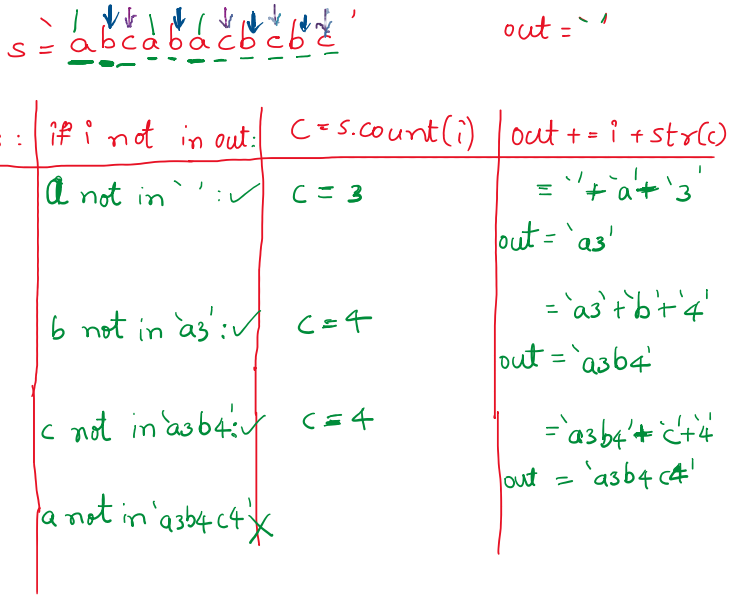
Note:

- Split() --- it is used to split each word present in the string
- Join() --- it is used to join/merge the strings present inside the collection.




```
# Get the following output.
'''
Input : 'hai hello bye'
Output : 'iah olleh eyb'      '''

'''
s = input('Enter the string: ')
out = []
a = s.split()
for i in a:
   out.append(i[::-1])
print(' '.join(out)) '''
```

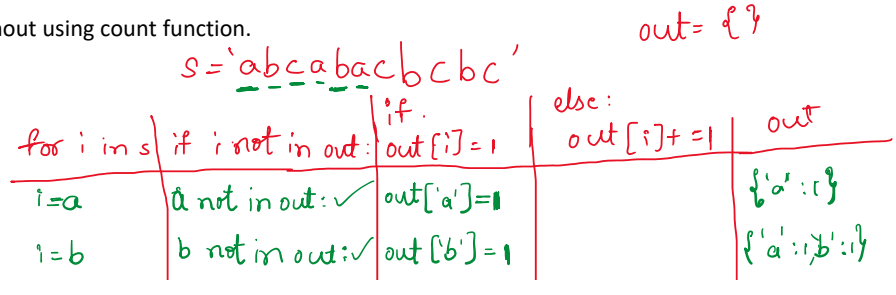
```
# Get the following output.
'''
Input : 'Everyone Loves python'
Output : 'Ee Ls pn'      '''

'''
s = input('Enter the string: ')
out = []
a = s.split()
for i in a:
   out.append(i[0]+i[-1])
print(' '.join(out)) '''
```



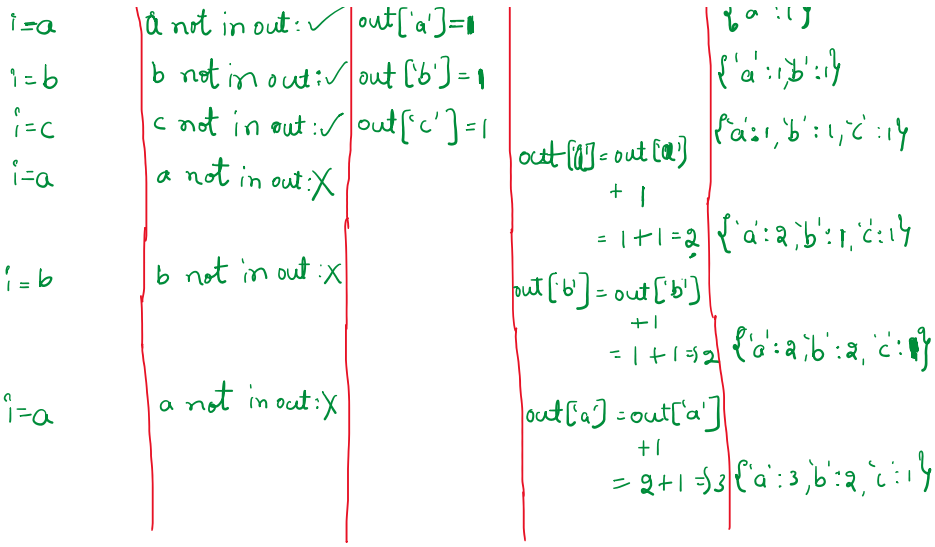

```
# Get the following output.
'''
Input : 'abcabacbcbc'
Output : 'a3b4c4'      '''

s = input('Enter the string: ')
out = ''
for i in s:
   if i not in out:
      c = s.count(i)
      out += i + str(c)
print(out)
```

Count() --- It will count the number of occurrence of character in a string.

```
# Get the following output without using count function.
'''
Input : 'abcabacbcbc'
Output : {'a':3,'b':4,'c':4}     '''

s =  input('Enter the string: ')
out = {}
for i in s:
   if i not in out:
```



```
out = {}
for i in s:
    if i not in out:
        out[i] = 1
    else:
        out[i] += 1
print(out)
```




```
# WAP to print all the divisors of a given number.
'''
n = int(input('Enter the number: '))
for i in range(1,n+1):
    if n % i == 0:
        print(i)'''
```

Day - 22:

Nested for loop:

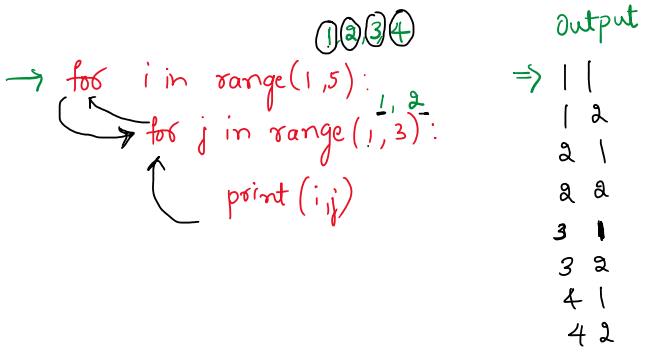
--- It is a phenomenon where we write a for loop inside another for loop.

Syntax:

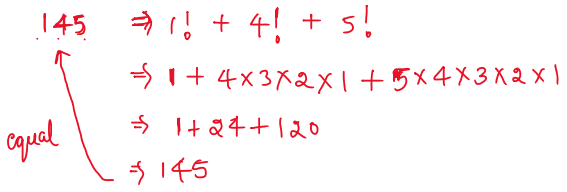
```
for var1 in collection :
    for var2 in collection :
        SB
```

Example:

```
# Nested for loop

for i in range(1,5):
    for j in range(1,3):
        print(i,j)
```



Strong Number: If the number is equal to the sum of the factorial of individual digits, then we can call that number as Strong Number.



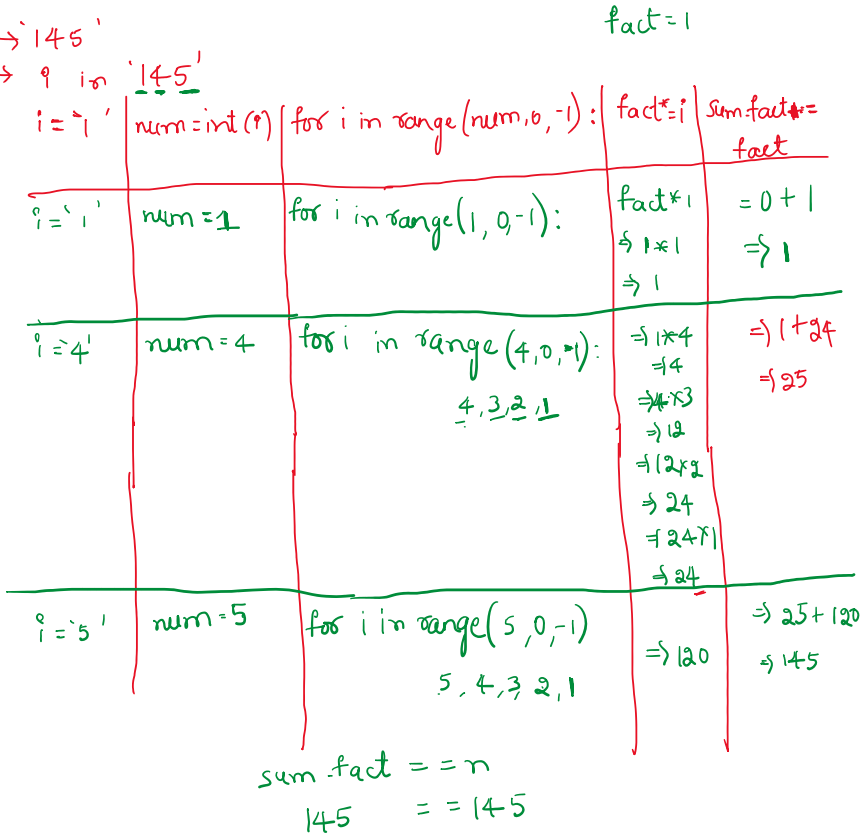
1st for loop ⇒ extract individual digits

2nd for loop ⇒ finding factorial

```
# WAP to check whether the number is strong number or not.

n =  int(input('Enter the number: '))
sum_fact = 0
```

fact = 1

```
n =  int(input('Enter the number: '))
sum_fact = 0
a = str(n)
for i in '145':
    num = int(i)
    fact = 1
    for i in range(num,0,-1):
        fact *= i
    sum_fact += fact
if sum_fact == n:
    print('Strong number')
else:
    print('Not strong number')
```




```
# Get the following output.
'''
Input : [12, 'program',4+2j, False,'holiday']
Output : {'program' : 'oa', 'holiday' : 'oia'}  '''

'''
l = eval(input('Enter the list: '))
out = {}
for i in l:
    if type(i) == str:
        vow = ''
        for j in i:
            if j in 'AEIOUaeiou':
                vow += j
        out[i] = vow
print(out)  '''

#Assignment
# Get the following output.
'''
Input : [12, 'program',4+2j, False,'holiday']
Output : {'program' : 'prgrm', 'holiday' : 'hldy'}  '''


#Assignment
# Get the following output.
'''
Input : [12, 'program',4+2j, False,'holiday']
Output : {'program' : 'PROGRAM', 'holiday' : 'HOLIDAY'}  '''
```

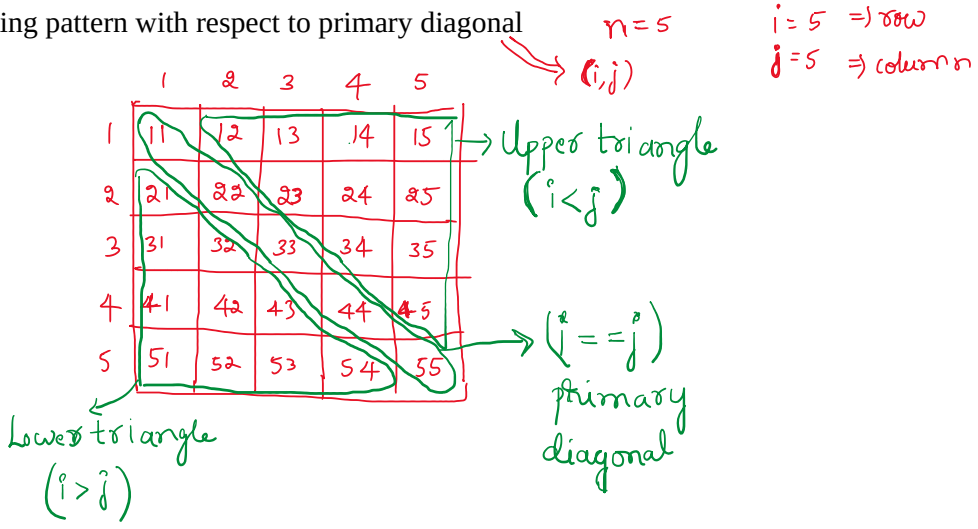
Patterns:

--- Using Nested for loop to print some unique structure or pattern.

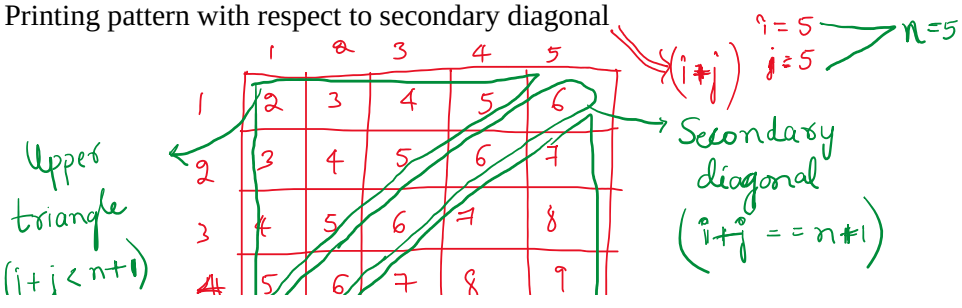
Programs:

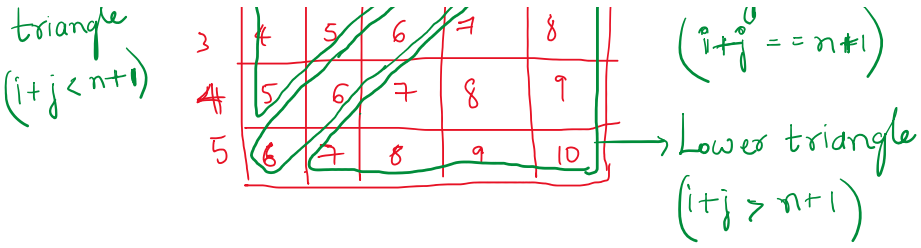
```
# Pattern
'''
* * *

for i in range(1,4):
   print('*',end = ' ') '''


'''
* * *
* * *
* * *

for i in range(1,4):
   for j in range(1,4):
      print('*',end = ' ')
   print() '''

'''
* * *
* * *
* * *
* * *
* * *

for i in range(1,6):
   for j in range(1,4):
      print('*',end = ' ')
   print()'''

'''
* * * * *
* * * * *

for i in range(1,3):
   for j in range(1,6):
      print('*',end = ' ')
   print() '''
```

- Printing pattern with respect to primary diagonal



Printing pattern with respect to secondary diagonal





Patterns:

```
'''
*
 *
  *
   *
    *
'''

n = int(input('Enter the number: '))
for i in range(1,n+1):
    for j in range(1,n+1):
        if i == j:
            print('*', end = ' ')
        else:
            print(' ', end = ' ')
    print()
```

```
'''
@
* @
* * @
* * * @
* * * * @
'''

n = int(input('Enter the number: '))
for i in range(1,n+1):
    for j in range(1,n+1):
        if i == j:
            print('@', end = ' ')
        elif i > j:
            print('*', end = ' ')
        else:
            print(' ', end = ' ')
    print()
```

```
'''
# # # # $
# # # $ &
# # $ & &
# $ & & &
$ & & & &
'''

n = int(input('Enter the number: '))
for i in range(1,n+1):
    for j in range(1,n+1):
        if i+j == n+1:
            print('$', end = ' ')
        elif i+j > n+1:
            print('&', end = ' ')
        elif i+j < n+1:
            print('#', end = ' ')
```

```
    print()

'''
* * * * *
*     *
*     *
*     *
* * * * *
'''
n = int(input('Enter the num: '))
for i in range(1,n+1):
    for j in range(1,n+1):
        if i == 1 or j == 1 or i == n or j ==n:
            print('*',end = ' ')
        else:
            print(' ',end = ' ')
    print()

'''
1 0 0 0 0
0 1 0 0 0
0 0 1 0 0
0 0 0 1 0
0 0 0 0 1
'''
n = int(input('Enter the num: '))
for i in range(1,n+1):
    for j in range(1,n+1):
        if i ==j:
            print('1',end = ' ')
        else:
            print('0',end = ' ')
    print()

'''



*     *
 *   *
  *
 *   *
*     *
'''
n = int(input('Enter the num: '))
for i in range(1,n+1):
    for j in range(1,n+1):
        if i == j or i+j == n+1:
            print('*',end = ' ')
        else:
            print(' ',end = ' ')
    print()


'''
* * * * *
*     *
*     *
*     *
* * * * *
'''
n = int(input('Enter the num: '))
for i in range(1,n+1):
    for j in range(1,n+1):
        if i == 1 or j==1 or i==n or j==n:
            print('*',end = ' ')
        else:
            print(' ',end = ' ')
    print()
```

```
'''
  *
  *
* * * * *
  *
  *
'''
n = int(input('Enter the num: '))
for i in range(1,n+1):
    for j in range(1,n+1):
        if i == n//2+1 or j == n//2+1:
            print('*',end = ' ')
        else:
            print(' ',end = ' ')
    print()


'''
* * * * * * * * *
*  *  *  *  *
*   *  *  *   *
*    ***    *
* * * * * * * * *
*    ***    *
*   *  *  *   *
*  *  *  *  *
* * * * * * * * *

'''
n = int(input('Enter the num: '))
for i in range(1,n+1):
    for j in range(1,n+1):
        if i==1 or j==1 or i==n or j==n or i==j or i+j==n+1 or i==n//2+1 or j==n//2+1:
            print('*',end = ' ')
        else:
            print(' ',end = ' ')
    print()
```

Day-23

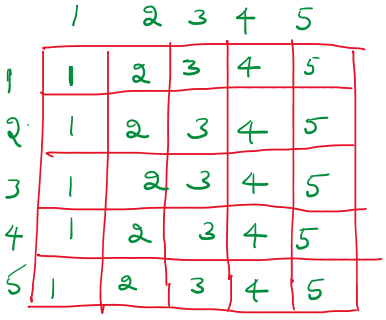
```
'''
1 2 3 4 5
1 2 3 4 5
1 2 3 4 5
1 2 3 4 5
1 2 3 4 5

n = int(input('Enter the number: '))
for i in range(1,n+1):
    for j in range(1,n+1):
        print(j,end = ' ')
    print()  ''
'''
```

n=5

(1,2,3,4,5) → row

(1,2,3,4,5) → column

end = '\n'



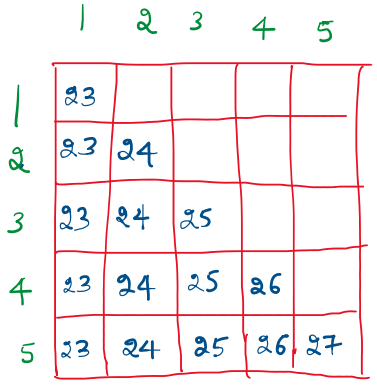

```
'''
1 1 1 1 1
2 2 2 2 2
3 3 3 3 3
4 4 4 4 4
5 5 5 5 5

n = int(input('Enter the number: '))
for i in range(1,n+1):
```

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


'''
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
'''

n = int(input('Enter the number: '))
for i in range(1,n+1):
    for j in range(1,n+1):
        if i == j or i > j:
            print(j,end = ' ')
        else:
            print(' ',end = ' ')
    print()

'''
23
23 24
23 24 25
23 24 25 26
23 24 25 26 27
'''

n = int(input('Enter the number: '))
for i in range(1,n+1):
    k = 23
    for j in range(1,n+1):
        if i == j or i > j:
            print(k,end = ' ')
            k += 1
        else:
            print(' ',end = ' ')
    print()
```

n=5

(1,2,3,4,5)

(1,2,3,4,5)




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

n = int(input('Enter the number: '))
for i in range(1,n+1):
    k = 5
    for j in range(1,n+1):
        if i+j == n+1 or i+j > n+1:
            print(k,end = ' ')
            k -= 1
        else:
            print(' ',end = ' ')
    print()


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

```
'''

n = int(input('Enter the number: '))
for i in range(1,n+1):
   k = 5
   for j in range(1,n+1):
      if i+j == n+1 or i+j > n+1:
         print(k,end = ' ')
         k -= 1
      else:
         print(' ',end = ' ')
   print()

'''
*     *
* *   * *
*   *   *
*     *
*     *
'''

n = int(input('Enter the number: '))
for i in range(1,n+1):
   for j in range(1,n+1):
      if j == 1 or j == n or (i == j and i <= n//2+1 ) or (i+j == n+1 and i <= n//2+1) :
         print('*',end = ' ')
      else:
         print(' ',end = ' ')
   print()

'''
*     *
*     *
*   *   *
* *   * *
*     *
'''

n = int(input('Enter the number: '))
for i in range(1,n+1):
   for j in range(1,n+1):
      if j == 1 or j == n or (i == j and i >= n//2+1 ) or (i+j == n+1 and i >= n//2+1) :
         print('*',end = ' ')
      else:
         print(' ',end = ' ')
   print()


'''
* * * *   *     *
*   *   *     *
*   * *   *   *   *
* * * *   * *   * *
        *   *     *
'''

n = int(input('Enter the number: '))
for i in range(1,n+1):
   for j in range(1,n+1):
      if (j == 1 and i != n) or (i == 1 and j != n) or (i == n-1 and j != n) or (j == n-1 and i != n) or (i == j and i >= n//2+
1 ) :
         print('*',end = ' ')
      else:
         print(' ',end = ' ')

   print(end = '  ')

   for j in range(1,n+1):
      if j == 1 or j == n or (i == j and i >= n//2+1 ) or (i+j==n+1 and i >= n//2+1):
```

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

'''
 * * * *
*
 * * *
     *
* * * *
'''


n = int(input('Enter the num: '))
for i in range(1,n+1):
    for j in range(1,n+1):
        if (i == 1 and j!=1) or (i == n//2+1 and  1<j<n) or (i == n and j!=n ) or (j == 1 and 1<i<n//2+1) or (j == n and n//2+1<i<n):
            print('*',end = ' ')
        else:
            print(' ',end = ' ')
    print()
```

Assignment:
--- Print the first 4 letters in your name using patterns.

Day-24

Intermediate terminations in looping:

--- Making the loops to stop the execution in between and restricting them to go all the iterations.

Types:

- Break
- Continue
- Pass

1) **Break:** It is used to perform intermediate terminations in looping. This can be used only for looping statement.
   Task: Whenever the controller sees the keyword break it will immediately terminate the loop.

   Example:

```
'''
num = int(input('Enter the number: '))
for i in range(1,6):
    print(i)
    if i == num:
        break '''

# Actual programs
# WAP to demonstrate guess the number game.
'''
sup = 784
while True:
    n = int(input('Enter the number: '))
    if n == sup:
        print('Congrats you have guessed it')
        break
    elif n > sup:
        print('Your guessed number is greater')
    elif n < sup:
        print('Your guessed number is lesser') '''

# WAP to check whether the given string is having only lowercase characters or not.
'''
s  = input('Enter the string: ')
for i in s:
```

```
    if not('a'<=i<='z'):
        print('It has other characters than lowercase')
        break
else:
    print('It has lowercase character') '''
```

2) Continue : Whenever we want to stop the particular step/Iteration we use continue keyword.

```
# Example
'''
num = 3
for i in range(1,6):
    if i == num:
        continue
    print(i)  '''

# Actual programs
# WAP to extract all the integers from the list.
'''
l = eval(input('Enter the list: '))
out = []
for i in l:
    if type(i) != int:
        continue
    else:
        out.append(i)
print(out)  '''

# WAP to print all the even number from 1 to n.
'''
n = int(input('Enter the number: '))
for i in range(1,n+1):
    if i % 2 != 0:
        continue
    else:
        print(i) '''
```

3) Pass : It is a keyword used to make any empty block as valid block.

```
# Example
'''
a=10
b=20
if a>b:
    pass
print('hello coders') '''
```

Functions:

--- It is a name given to a memory block, where the set of intructions are stored and performing some particular task.
By creating a function for one time, we can use it for n number of times.

Why we need Functions? (Important)

- We can reduce the number of instructions.
- We can increase the efficiency of the code.
- We can avoid code repitition.
- Reuse the code for n number of times.

Types:

- **Inbuilt Functions**

- **User defined functions.**

Inbuilt Functions:

--- They are predefined functions.

- **Utility function: It is used for every datatype / common for more than one datatype.**

  Example: id(), type(), bool()

- **Inbuilt functions on String**:

  Example: upper(), lower(), swapcase(), capitalize(), title(), split(), join(),replace(), index(),count() , ord(), chr()

  Practical proof:

```
s = 'Coders'
s.upper()
'CODERS'
s = 'PytHoN'
s.lower()
'python'
s = 'GoOd EvEnInG'
s.swapcase()
'gOoD eVeNiNg'
s = 'PytHoN'
s.capitalize()
'Python'
s = 'Python is easy to learn and analyze'
s.title()
'Python Is Easy To Learn And Analyze'
s.capitalize()
'Python is easy to learn and analyze'
s = 'Python is easy'
s.split()
['Python', 'is', 'easy']
s = 'Coders'
'@'.join(s)
'C@o@d@e@r@s'
print('c','o','d','e','r','s', sep = '@')
c@o@d@e@r@s
s = 'Coders'
s.replace('o','a')
'Caders'
s.replace('Python','Java')
'Coders'
s = 'Python'
s.replace('Python','Java')
'Java'
s = 'Python'
s.index('t')
2
s = 'appa'
s.count('a')
2
ord('A')
65
chr(65)
'A'
s = 'appa'
'a'.join(s)
'aapapaa'
'a'.join(' appa')
' aaapapaa'
```

- **Inbuilt functions on List**:

Example: append(), insert(), pop(), remove(), sort(), reverse(), count(), index()

```
l = [30,20,10,50,40]
l.sort()
l
[10, 20, 30, 40, 50]
l.sort(reverse = True)
l
[50, 40, 30, 20, 10]
l = [30,20,'10',[50,20],40]
l.sort()
Traceback (most recent call last):
  File "<pyshell#6>", line 1, in <module>
    l.sort()
TypeError: '<' not supported between instances of 'str' and 'int'
l = [10,2.3,2+3j,True,'hi']
l.reverse()
l
['hi', True, (2+3j), 2.3, 10]
l = [10,2.3,2+3j,10,True,'hi',10]
l.count(10)
3
l.index(2+3j)
2
```

- **Inbuilt functions on Tuple**:

Example: count(), index()

```
t = (10,20,30,40)
t.count(30)
1
t.index(20)
1
```

- **Inbuilt functions on Set**:

Example: add(), pop(), remove(), union(), intersection(), difference(), clear()

```
s1 = {12,3.4,2+4j,7,True}
s2 = {22,3.4,3+4j,9,False}
s1.union(s2)
{False, True, 3.4, 7, 9, 12, (2+4j), (3+4j), 22}
s1.intersection(s2)
{3.4}
s1
{True, 3.4, 7, 12, (2+4j)}
s1.difference(s2)
{True, 12, (2+4j), 7}
s1.clear()
s1
set()
```

- **Inbuilt functions on Dictionary**:

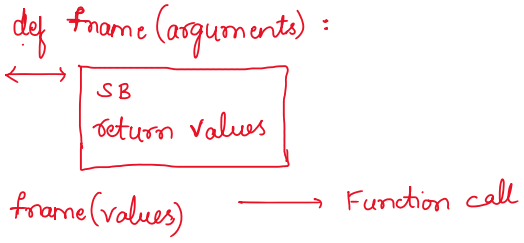
Example: keys(),values(), items(), pop(), get(), popitem(), update(), clear()

```
a = {'a':10,'b':20,'c':30,'d':40,'m':'monkey'}
a.keys()
dict_keys(['a', 'b', 'c', 'd', 'm'])
a.values()
dict_values([10, 20, 30, 40, 'monkey'])
a.items()
dict_items([('a', 10), ('b', 20), ('c', 30), ('d', 40), ('m', 'monkey')])
a.pop('m')
'monkey'
a
{'a': 10, 'b': 20, 'c': 30, 'd': 40}
```

```
a.get('d')
40
a.get('c','b')
30
a
{'a': 10, 'b': 20, 'c': 30, 'd': 40}
a = {'a': 10, 'b': 20, 'c': 30, 'd': 40}
a.popitem()
('d', 40)
a = {'a': 10, 'b': 20, 'c': 30, 'd': 40}
a = {'a': 10, 'b': 20, 'c': 30, 'd': 40}
b = {'e': 10, 'f': 20}
a.update(b)
a
{'a': 10, 'b': 20, 'c': 30, 'd': 40, 'e': 10, 'f': 20}
b = {'e': 10, 'f': 20}
b.update({'a':10})
b
{'e': 10, 'f': 20, 'a': 10}
b = {'e': 10, 'f': 20}
b.clear()
b
{}
```

- **User defined functions:**
  --- The functions which are defined by the user based on their user requirement.

Syntax:



Note:

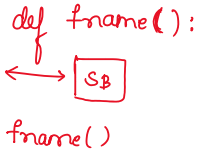
- Passing the arguments and the return value are not mandatory.
- Arguments --- It consists of required values to perform the operations
- def --- It is a keyword used to define / create a function.
- To execute the function, we have to call the function.
- return keyword is used to stop the execution.

Types:

- **Function without arguments and without return value**
- **Function with arguments and without return value**
- **Function without arguments and with return value**
- **Function with arguments and with return value**

1) Function without arguments and without return value:

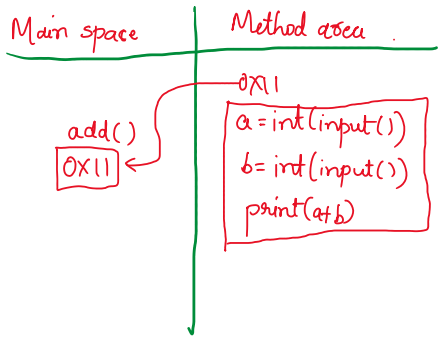
Syntax:




```
# Example:
'''
def add():
```

```
    a = int(input('Enter the number: '))
    b = int(input('Enter the number: '))
    print(a+b)
add() '''
```

Memory allocation:




```
# Example:
'''
def add():
    a = int(input('Enter the number: '))
    b = int(input('Enter the number: '))
    print(a+b)
add() '''

# Program
# WAP to convert the string to uppercase.
'''
def convert_up():
    s = input('Enter the string: ')
    out = ''
    for i in s:
        if 'a'<=i<='z':
            out += chr(ord(i)-32)
        else:
            out += i
    print(out)
convert_up() '''

# WAP to count the number of occurence of character in a string.
'''
def count_char():
    s = input('Enter the string: ')
    ch = input('Enter the character: ')
    count = 0
    for i in s:
        if i == ch:
            count += 1
    print(count)
count_char() '''
```

2) Function with arguments and without return value:

Syntax:

```
def fname(var1, var2, ...., varn):
    SB

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

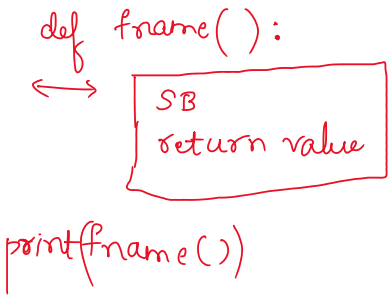
Programs:

```
# Function with arguments and without return value
```

```
# Example:
'''
def add(a,b):
    print(a+b)
add(10,20) '''

# Program
# WAP to convert the string to uppercase.
'''
def convert_up(s,out):
    for i in s:
        if 'a'<=i<='z':
            out += chr(ord(i)-32)
        else:
            out += i
    print(out)
convert_up(input('Enter the string: '),'') '''

# WAP to count the number of occurence of character in a string.
'''
def count_char(s,ch,count):
    for i in s:
        if i == ch:
            count += 1
    print(count)
count_char(input('Enter the string: '),input('Enter the character: '),0)
'''
```

3) Function without arguments and with return value:

Syntax:



Program

```
# Example:
'''
def add():
    a = int(input('Enter the number: '))
    b = int(input('Enter the number: '))
    return a+b
print(add()) '''

# Program
# WAP to convert the string to uppercase.
'''
def convert_up():
    s = input('Enter the string: ')
    out = ''
    for i in s:
        if 'a'<=i<='z':
            out += chr(ord(i)-32)
        else:
            out += i
    return out
print(convert_up()) '''

# WAP to count the number of occurence of character in a string.
'''
def count_char():
    s = input('Enter the string: ')
    ch = input('Enter the character: ')
    count = 0
    for i in s:
        if i == ch:
            count += 1
    return count
print(count_char()) '''
```

4) Function with arguments and with return value:

Syntax:

def fname(var1, var2, ... ,varn) :
    SB
    return values
print(fname(val1, val2 ..... ,valn))

Programs:

```
# Example:
'''
def add(a,b):
   return a+b
print(add(10,20)) '''

# Program
# WAP to convert the string to uppercase.
'''
def convert_up(s,out):
   for i in s:
      if 'a'<=i<='z':
         out += chr(ord(i)-32)
      else:
         out += i
   return out
print(convert_up(input('Enter the string: '),''))'''

# WAP to count the number of occurence of character in a string.
'''
def count_char(s,ch,count):
   for i in s:
      if i == ch:
         count += 1
   return count
print(count_char(input('Enter the string: '),input('Enter the character: '),0))
'''
```

Global Variable:

--- These variables are created in main space and this can be accessed and modified in the main space but cannot be modified in method area.

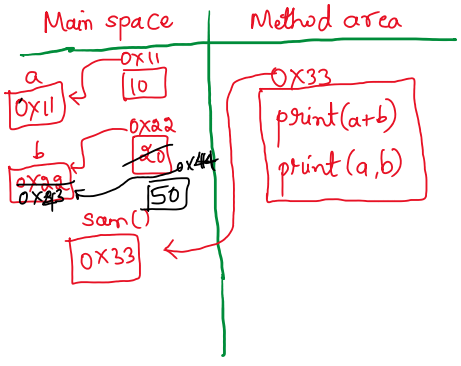
Example:

```
a = 10
b = 20
def sam():
   print(a+b)
   print(a,b)
print(a,b)
sam()
b = 50
print(a,b)
```

output

```
10  20
30
10  20
10  50
```

Memory allocation:



To overcome the problem(cannot be modified in method area**) of this, we use a keyword 'global' inside the function. After using the global keyword we can do modification for global variables inside the function.**

```
a = 10
b = 20
def sam():
    global a
    print(a+b)
    a = 30
    print(a,b)
print(a,b)
sam()
b = 50
print(a,b)
```

output

10	20
30	
30	20
30	50

Local Variable:

--- These variables are created in method area and this can be accessed and modified in the same function but cannot be modified in the nested function.

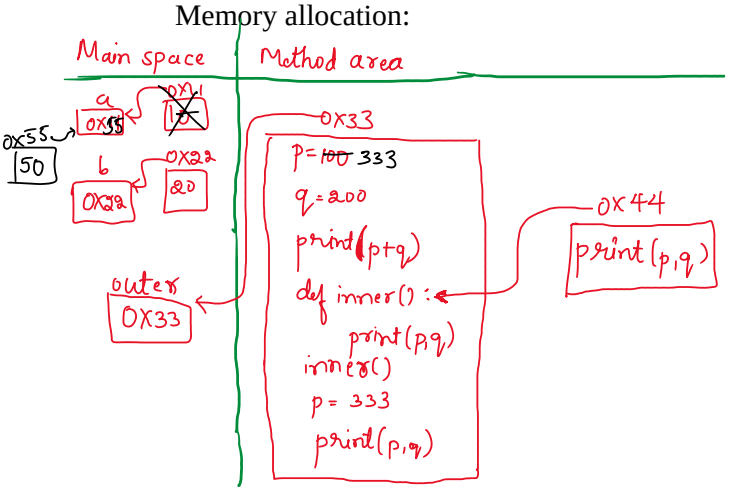
Example:

```
a = 10
b = 20
def outer():
    p = 100
    q = 200
    print(p+q)
    def inner():
        print(p,q)
    inner()
    p = 333
    print(p,q)
a = 50
print(a,b)
outer()
```

output

50	20
300	
100	200
333	200

Memory allocation:



To overcome the problem(cannot be modified in nested function**) of this, we use a keyword ' nonlocal ' inside the nested function. After using the nonlocal keyword we can do modification for local variables inside the nested function.**

```
#Using nonlocal keyword
'''
a = 10
b = 20
def outer():
    p = 100
    q = 200
    print(p+q)
    def inner():
        nonlocal p
        print(p,q)
        p = 500
        print(p,q)
    inner()
    p = 333
    print(p,q)
a = 50
print(a,b)
outer()  '''
```

Day-28

Packing and Unpacking:

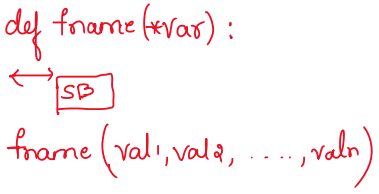
- **Packing:**
--- It is a phenomenon of grouping the individual values in the form of collections to provide security.

Note: We can do packing in all the collection, but our system will prefer to do packing in only tuple datatype.
Reason: Tuple is the most secured datatype.

Types of Packing:
- Tuple/Single packing
- Dictionary/Double packing

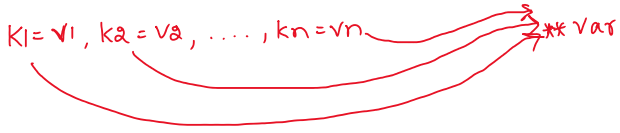
1) **Tuple/Single packing:** It is a phenomenon of grouping the individual values in the form of tuple collections to provide security.

Syntax:

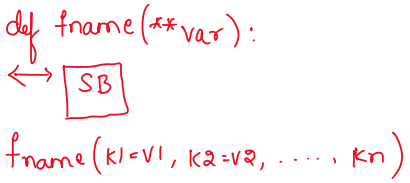



```
# Tuple packing
'''
def pack(*t):
   print(type(t))
   print(t)
pack(10,20,30,40,50) '''
```

2) **Dictionary/Double packing:** It is a phenomenon of grouping the individual key-value pair in the form of dictionary to provide security.



Syntax:



Note:

- Keys should follow the rules of identifiers
- Keys should be of string datatype only
- Keys shouldn't enclose with quotes but values must be enclosed with quotes if it is string.

```
# Dictionary packing
'''
def pack(**d):
   print(type(d))
   print(d)
pack(a=10,b='20',c=30,d=40)  '''
```

What is the advantage of Packing over Collection datatype?

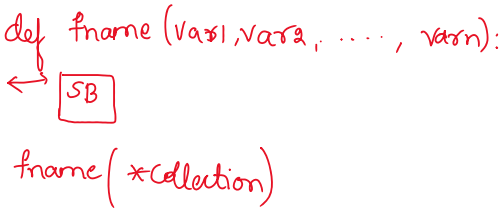
- Memory efficiency
- Increases performance
- Lighter and faster

When to use collection datatypes and when to use packing?

- **Use Nested collection :** When developing applications where clarity and maintainability required.
- **Use packing :** For systems which has less memory, less processing efficiency.

- **<u>Unpacking:</u>**
  --- The phenomenon of dividing the collection and storing each and every value present in the collection to a unique variable.

<u>Syntax</u>:




```
# Unpacking
'''
def unpack(v1,v2,v3,v4):
   print(v1,v2,v3,v4)
unpack(*'abcd') '''

'''
def unpack(v1,v2,v3,v4):
   print(v1,v2,v3,v4)
unpack(*[10,20,30,40]) '''

'''
def unpack(v1,v2,v3,v4):
   print(v1,v2,v3,v4)
unpack(*(10,20,30,40)) '''

'''
def unpack(v1,v2,v3,v4):
   print(v1,v2,v3,v4)
unpack(*{10,20,30,40})  '''

'''
def unpack(v1,v2,v3,v4):
   print(v1,v2,v3,v4)
unpack(*{'a':10,'b':20,'c':30,'d':40})
unpack(*{'a':10,'b':20,'c':30,'d':40}.values())
```

```
unpack(*{'a':10,'b':20,'c':30,'d':40}.items()) '''

'''
def unpack(v1,v2,v3,v4):
   print(v1,v2,v3,v4)
unpack(*range(1,5)) '''
```

Types of arguments:

- Positional argument
- Default argument
- Keyword argument
- Variable length argument

1) Positional argument :

--- The arguments which are created in function declaration are called as Positional argument.

Note:

- It is compulsory to pass the value
- Follow the same order.

```
def add(a,b):
   print(a+b)
add(10,20)
```

2) Default argument :

--- They are present in function declaration, if the user is passing the default value then by default it will take the value. If I pass the value then default value of the variable will be replaced with new value.

```
# Default argument
'''
def add(a=0,b=0):
   print(a+b)
add(10,20)  '''
```

3) Keyword argument :

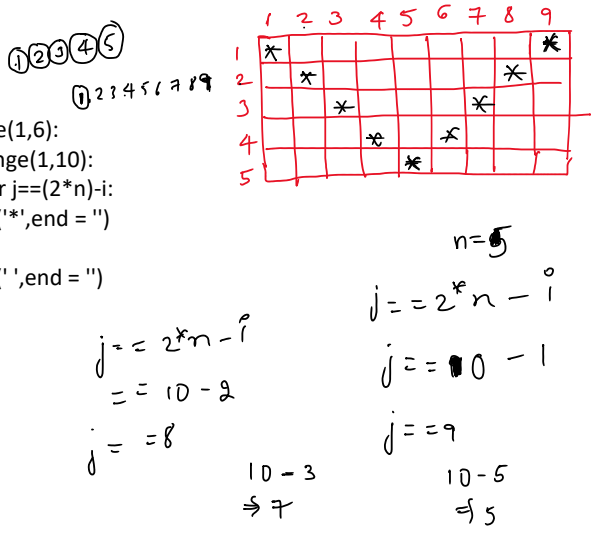
--- Passing the key-value pair as the value to the arguments. They are present in function call.

```
# Keyword argument
'''
def pack(**d):
   print(type(d))
   print(d)
pack(a=10,b='20',c=30,d=40) '''
```

4) Variable length argument :

--- These arguments are capable of taking values from 1 to n number of inputs or values. They are present in function declaration.

```
# Variable length argument
'''
def pack(*t,**d):
   print(type(t))
   print(t)
   print(type(d))
   print(d)
pack(10,20,30,40,50,a=10,b=20,c=30) '''
```



```
n = 5
for i in range(1,6):
    for j in range(1,10):
        if i==j or j==(2*n)-i:
            print('*',end = '')
        else:
            print(' ',end = '')
    print()
```

Day-29

Recursion:

--- The phenomenon of calling the function by itself until the termination condition becomes True.

Syntax:

Without return Value

```
def fname(args):
    if termination_cond:
        return
    fname(args)
fname(values)
```

With return value.

```
def fname(args):
    if termination_cond:
        return value
    return fname(args)
print(fname(values))
```

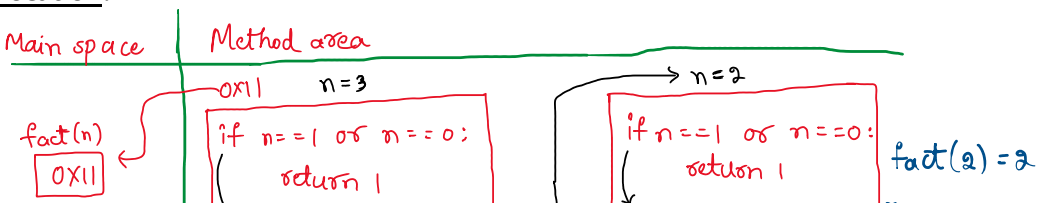
Note: return keyword is used to stop the execution of the function. If it is having any result in front of it it will display that or else it will just simply terminate the function.

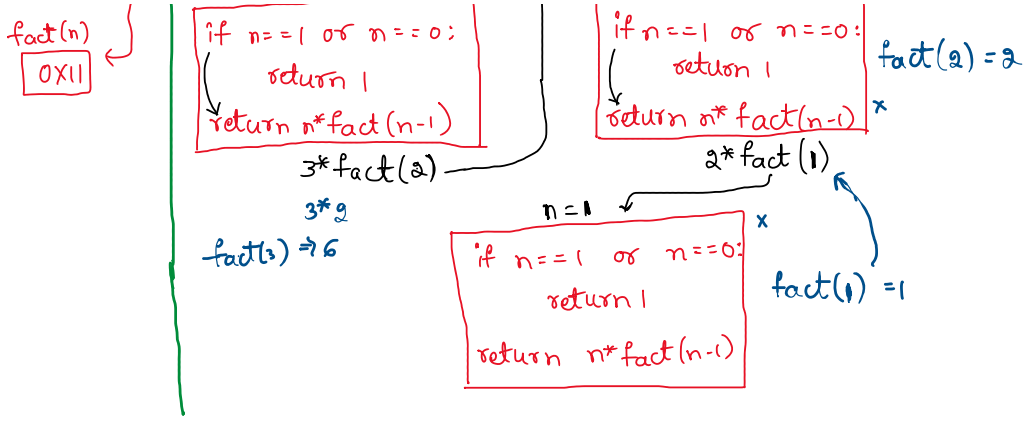
Programs:

WAP to find the factorial of a given number.

```
def fact(n):
    if n==1 or n==0:
        return 1
    return n*fact(n-1)
print(fact(3))
```

Memory allocation:

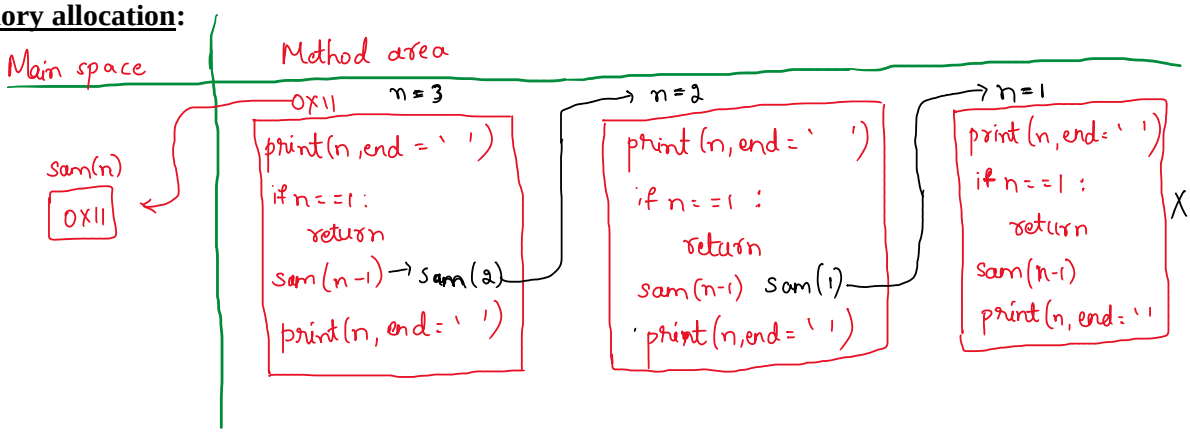





```
# WAP to print the sum of n natural numbers
'''
def sum_nat(n):
  if n == 1:
    return 1
  return n+sum_nat(n-1)
print(sum_nat(10))  '''


# WAP to print
#  3 2 1 2 3
'''
def sam(n):
  print(n,end = ' ')
  if n == 1:
    return
  sam(n-1)
  print(n,end = ' ')
sam(3) '''
```

3 2 1 2 3

Memory allocation:



Steps to convert the looping program in the form of recursion:

- Initialisation of all the required variables of looping should be done in function declaration.
- The termination condition should be written exactly opposite to the looping condition in the form of if statement.
- Return the total result inside the termination condition.
- Logic of the program should be written as it is.
- Updation of looping variable should be done in recursive call.

> ⇒ <=
< ⇒ >=

- Updation of looping variable should be done in recursive call.

Extract all the lowercase character from the string.

```
def sam(s,out='',i=0):
    if i>=len(s):
        return out
    if 'a'<=s[i]<='z':
        out+=s[i]
    return sam(s,out,i+1)
print(sam(input('enter the string:')))
```

```
s=input('enter the string:')
out=''
i=0
while i<len(s):
    if 'a'<=s[i]<='z':
        out+=s[i]
    i+=1
print(out)
```

Get the following output
Input : ['hai',45,2+4j,'bye']
output : ['iahhai','eybbye']

```
'''
l = eval(input('Enter the list: '))
out = []
i = 0
while i<len(l):
    if type(l[i]) == str:
        out.append(l[i][::-1] + l[i])
    i+=1
print(out) '''

'''
def sam(l,out=[],i=0):
    if i >= len(l):
        return out
    if type(l[i]) == str:
        out.append(l[i][::-1] + l[i])
    return sam(l,out,i+1)
print(sam(eval(input('Enter the list : '))))  '''
```

Assignment:
--- Take any 5 while loop program and convert them to recursion program

Day-30

Introduction to OOPS:
--- The concept which deals with class and objects are called OOPS.

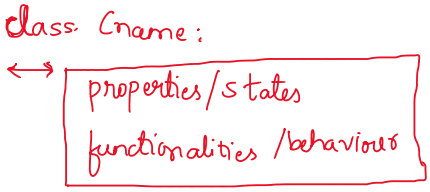
OOPS -- Object Oriented Programming System.

Class: It is a container which is used to store the properties and functionalities of real-time entity.

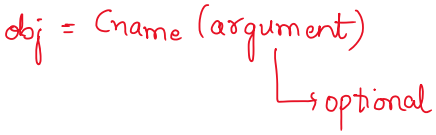
Object: It is an instance or copy of the class.

Class and Object Creation:

- **Class creation:**
  Syntax:



- **Object creation:**
  Syntax:




```
#Class and Object Creation
'''
class Creation:
   a = 10
   b = 20
demo = Creation() '''
```

Types of Class:

- **Inbuilt class --- All the datatypes are inbuilt class because they are predefined.**
- **User-defined class.**

```
class Creation:
   a = 10
   b = 20
demo = Creation()
print(type(demo))
```

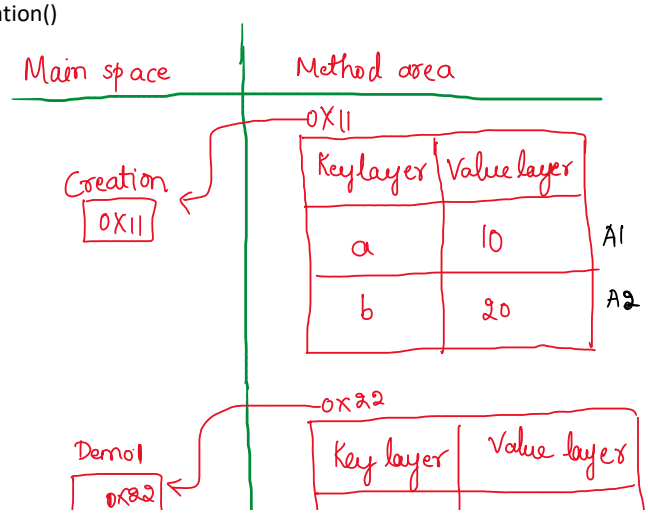
Output:

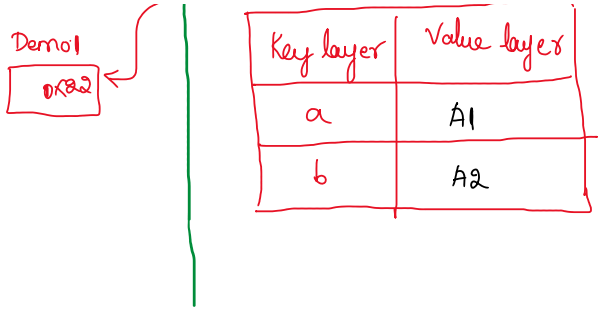
```
<class '__main__.Creation'>
```

Note: __main__ is representing that class is created by user / User Defined class.

Memory allocation:

```
class Creation:
   a = 10
   b = 20
Demo1 = Creation()
```

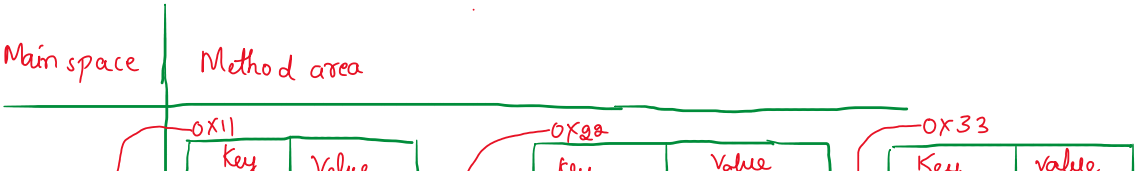


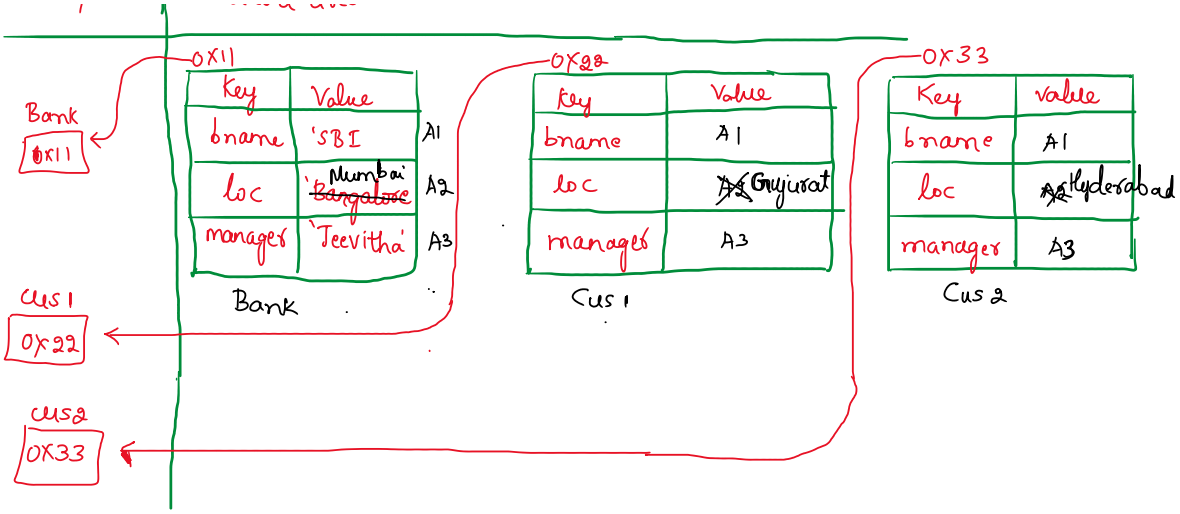


Whenever we want to access the values from class or object we have to use the syntax,

For class ----- Cname.var
For object ---- obj.var

```
# Access the value of class and object
'''
class Creation:
   a = 10
   b = 20
demo1 = Creation()
demo2 = Creation()
print(Creation.a,Creation.b)
print(demo1.a,demo1.b)
print(demo2.a,demo2.b) '''

# Actual program on accessing the value of class and object
'''
class Bank:
   bname = 'SBI'
   loc = 'bangalore'
   manager = 'Jeevitha'
cus1 = Bank()
cus2 = Bank()
print(Bank.bname,Bank.loc,Bank.manager)
print(cus1.bname,cus1.loc,cus1.manager)
print(cus2.bname,cus2.loc,cus2.manager)
print('*'*50)

Bank.loc = 'Mumbai'
print(Bank.bname,Bank.loc,Bank.manager)
print(cus1.bname,cus1.loc,cus1.manager)
print(cus2.bname,cus2.loc,cus2.manager)
print('*'*50)

cus1.loc = 'Gujurat'
print(Bank.bname,Bank.loc,Bank.manager)
print(cus1.bname,cus1.loc,cus1.manager)
print(cus2.bname,cus2.loc,cus2.manager)
print('*'*50)

cus2.loc = 'Hyderabad'
print(Bank.bname,Bank.loc,Bank.manager)
print(cus1.bname,cus1.loc,cus1.manager)
print(cus2.bname,cus2.loc,cus2.manager) '''
```





Conclusions:

- Modification done with respect to class will affect all the objects and also the class.
  Reason: Objects are instance / copy of the class .

- Modification done with respect to one object will not affect the class and other objects.
  Reason: Class are not depending on objects.

Day-31

States:
--- The properties or functionalities storing inside the class are called as States.

Types:

- Generic / static / class members
- Specific / base / object members

1) **Generic State:**
   --- The properties or functionalities which will be common for each and every object we create
   is called as Generic State.

   Example:

```
# Generic State.
'''
class School:
    sname = 'JES'
    loc = 'Hampi'
    principal = 'Nandakumar'
    timing = '9am - 4:30pm'
st1 = School()
print(School.sname,School.loc,School.principal,School.timing)
print(st1.sname,st1.loc,st1.principal,st1.timing)
'''
```

2) **Specific State:**
   --- The properties or functionalities which will we create outside the class after the object creation is called as Generic State.

   Example:

```
# Specific state
'''
class School:
   sname = 'JES'
   loc = 'Hampi'
   principal = 'Nandakumar'
   timing = '9am - 4:30pm'
st1 = School()
st1.name = 'A'
st1.id = 21
st1.age = 22
st1.bg = 'B+ve'
st2 = School()
st2.name = 'B'
st2.id = 22
st2.age = 22
st2.bg = 'A+ve'
print(st1.name,st1.id, st1.age, st1.bg )
print(st2.name,st2.id, st2.age, st2.bg ) '''
```

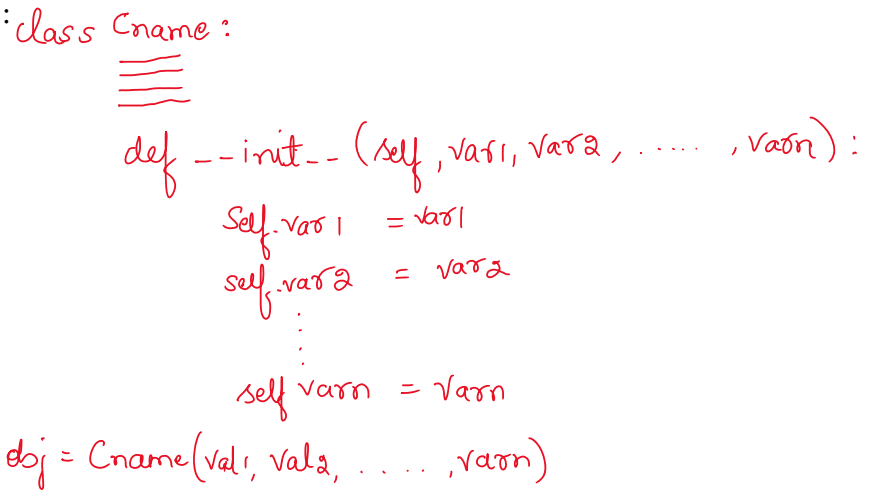
Difference between method and function:

--- Functions which we declare inside the class is called as Methods, If we declare the function outside the class it is called as Function.

Constructor method / __init__ / Initialisation :

- It is used to initialize the members of the object.
- No need of calling the __init__ method , by default it will execute when we create an object.
- self is the manditory argument which we have to pass for the __init__ method.
  self --- it is used to store the address of the object.
- We can pass arguments in the object creation only if there is __init__ method inside the class.

Syntax :



Program:

```
# Constructor method
'''
class School:
   sname = 'JES'
   loc = 'Hampi'
   principal = 'Nandakumar'
   timing = '9am - 4:30pm'
   def __init__(self,name,sid,age,bg):
      self.name = name
```

```
        self.sid = sid
        self.age = age
        self.bg = bg
st1 = School('A',21,23,'B+ve')
print(st1.name,st1.sid, st1.age, st1.bg )
st2 = School('B',22,21,'A+ve')
print(st2.name,st2.sid, st2.age, st2.bg ) '''
```

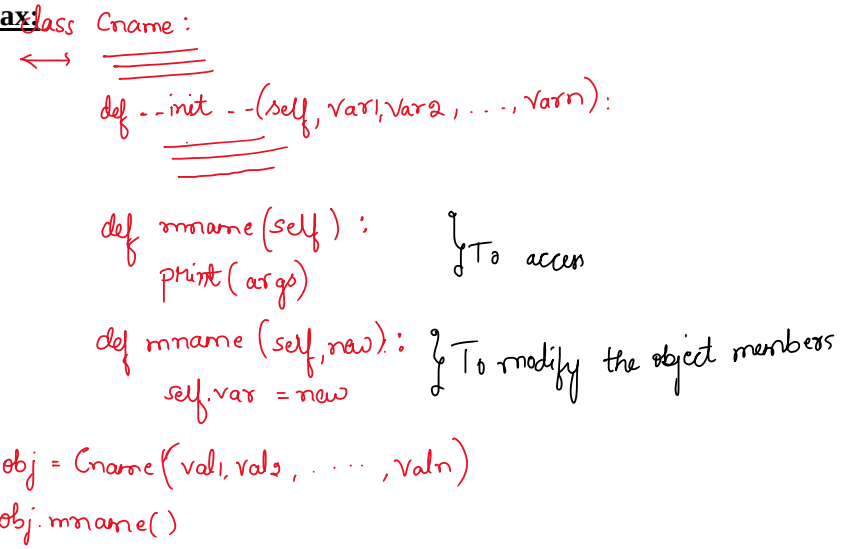
Methods:
--- The function which we declare inside the class.

Types:

- Object method
- Class method
- Static method

1) **Object method** :
--- They are used to access and modify the object members

Syntax:



Programs

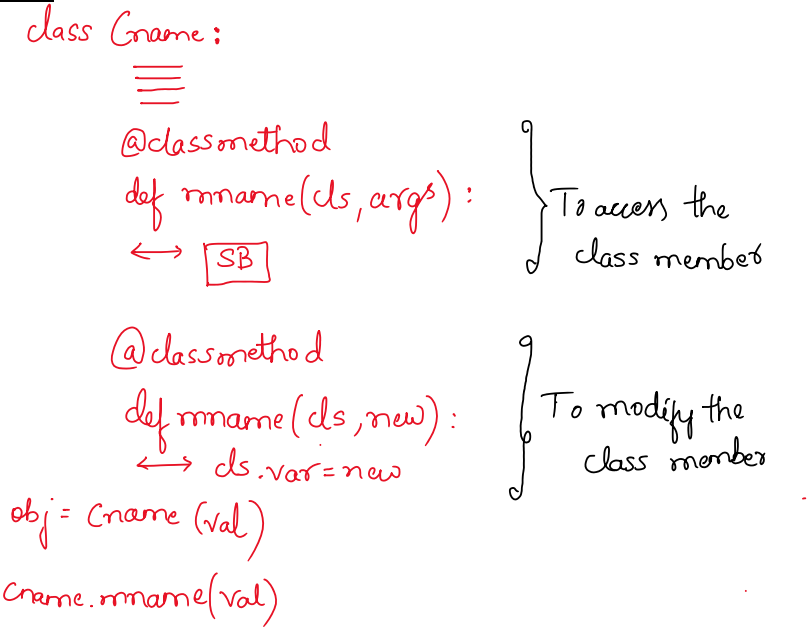
```
# Object method
'''
class School:
    sname = 'JES'
    loc = 'Hampi'
    principal = 'Nandakumar'
    timing = '9am - 4:30pm'
    def __init__(self,name,sid,age,bg):
        self.name = name
        self.sid = sid
        self.age = age
        self.bg = bg
    def display(self):
        print(self.name,self.sid,self.age,self.bg)
    def ch_age(self,new):
        self.age = new
st1 = School('A',21,23,'B+ve')
st1.display()
st1.ch_age(25)
st1.display() '''
```

Day-32

2) Class method :

--- They are used to access and modify the class members. We need to use 'cls' as an argument to store the address of the class members and it is compulsory to use @classmethod

Syntax:



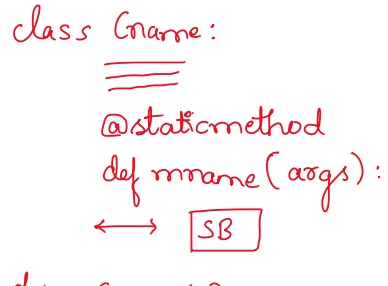
Program

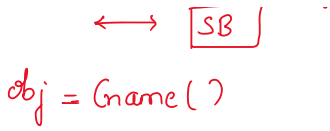
```
class School:
   sname = 'JES'
   loc = 'Hampi'
   principal = 'Nandakumar'
   timing = '9am - 4:30pm'
   @classmethod
   def display(cls):
      print(cls.sname,cls.loc,cls.principal,cls.timing)
   @classmethod
   def ch_time(cls,new,change):
      cls.timing = new
      cls.loc = change
st1 = School()
School.display()
School.ch_time('9am - 5pm','mausmi')
School.display()
```

3) Static method :

--- It is neither belongs to class members nor object members but it will acts as supportive method for both class and objects.

Syntax:



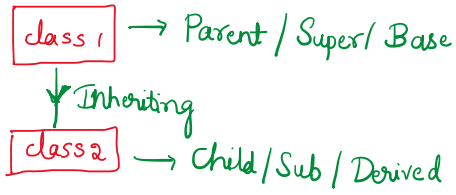


Programs:

```
class Boring:
   name = 'Sakshi'
   role = 'Irritator'
   @staticmethod
   def nonsense(a,b):
      print(a+b)
st1 = Boring()
Boring.nonsense(10,20)
st1.nonsense(10,20)
```

OOPS concepts:

Inheritance:
--- The phenomenon of deriving the properties from one class to another class.



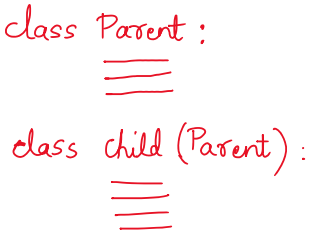
Parent class : The class from where we inherit the properties.
Child class : The class which inherits the properties from parent class.

Types of Inheritance:

- Single level
- Multi-level
- Multiple
- Hierarchical
- Hybrid

1) **Single level Inheritance:**
   --- Inheriting the properties from one parent class to one child class is called Single level Inheritance.

   Syntax:



   Program :

```
#Single level
'''
class School:
```

```
    sname = 'JES'
    loc = 'Hampi'
    principal = 'Nandakumar'

class College(School):
    clgname = 'Sai Vidya'
    HOD = 'Ranganath'
    @classmethod
    def display(cls):
        print(cls.clgname,cls.HOD,cls.sname,cls.loc,cls.principal)
College.display()
'''

'''
class Bank:
    bname = 'SBI'
    loc = 'Koppa'
    def __init__(self,name,phno,bal):
        self.name = name
        self.phno = phno
        self.bal = bal

cus1 = Bank('A',947349534,500)

class Updated_Bank(Bank):
    Founder = 'Sairam'
    def __init__(self,addr,pan,name,phno,bal):
        self.addr = addr
        self.pan = pan
        self.name = name
        self.phno =  phno
        self.bal = bal
    def display(self):
        print(self.addr,self.pan,self.name,self.phno,self.bal)
cus2 = Updated_Bank('Gokarna',97374,'A',947349534,500)
cus2.display()
'''
```

Constructor chaining and method chaining:

The process of calling parent __init__ method inside the child __init__ method is called Constructor chaining.

Syntax:

super().__init__(args)

The process of calling the parent method inside the child method is called Method chaining.

Syntax:

super().mname(args)

Program :

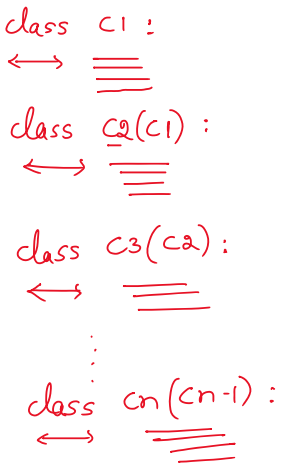
```
class Bank:
    bname = 'SBI'
    loc = 'Koppa'
    def __init__(self,name,phno,bal):
        self.name = name
        self.phno = phno
        self.bal = bal
    def disp(self):
        print(self.name,self.phno,self.bal)
```

```
cus1 = Bank('A',947349534,500)

class Updated_Bank(Bank):
   Founder = 'Sairam'
   def __init__(self,name,phno,bal,addr,pan):
      super().__init__(name,phno,bal)
      self.addr = addr
      self.pan = pan
   def display(self):
      super().disp()
      print(self.addr,self.pan)
cus2 = Updated_Bank('Gokarna',97374,'A',947349534,500)
cus2.display()
```

2) Multi-level Inheritance:

--- Inheriting the properties from parent class to child class occurs in multiple level.

Syntax:



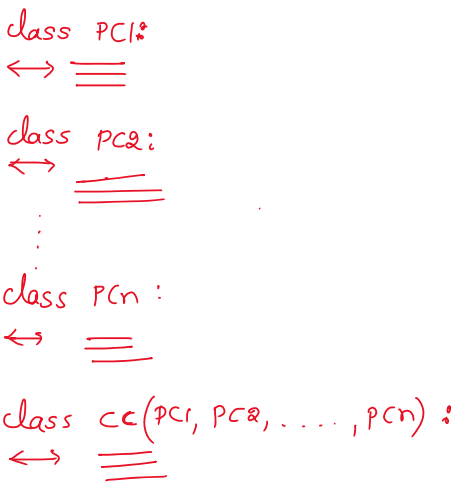
Proof:

```
# Multi-level Inheritance.
'''
class School:
   sname = 'JES'
   loc = 'Hampi'
   princi = 'Nanda'

class College(School):
   clgname = 'Sai'
   HOD = 'Ranga'

class Masters(College):
   uname = 'VTU'
   founder = 'Bala'
   @classmethod
   def display(cls):
      print(cls.uname,cls.founder,cls.clgname,cls.HOD,cls.sname,cls.loc,cls.princi)
Masters.display() '''
```

3) Multiple Inheritance:

--- Inheriting properties from multiple parents to a single child.

Syntax:



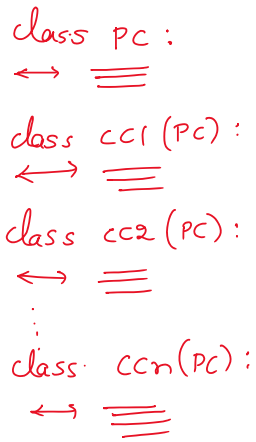
Programs:

```
# Multiple Inheritance.
'''
class School:
   sname = 'JES'
   loc = 'Hampi'
   princi = 'Nanda'

class College:
   clgname = 'Sai'
   HOD = 'Ranga'

class Masters(College,School):
   uname = 'VTU'
   founder = 'Bala'
   @classmethod
   def display(cls):
      print(cls.uname,cls.founder,cls.clgname,cls.HOD,cls.sname,cls.loc,cls.princi)
Masters.display()
'''
```

4) Hierarchical Inheritance:

--- Inheriting properties from single parent to a multiple child.

Syntax:



```
# Hierarchical Inheritance
'''
class School:
    sname = 'JES'
    loc = 'Hampi'
    princi = 'Nanda'


class College(School):
    clgname = 'Sai'
    HOD = 'Ranga'
    @classmethod
    def disp(cls):
        print(cls.clgname,cls.HOD,cls.sname,cls.loc,cls.princi)
class Masters(School):
    uname = 'VTU'
    founder = 'Bala'
    @classmethod
    def display(cls):
        print(cls.uname,cls.founder,cls.sname,cls.loc,cls.princi)
College.disp
Masters.display()
'''
```

5) Hybrid Inheritance:

--- Combination of more than one type of inheritance.

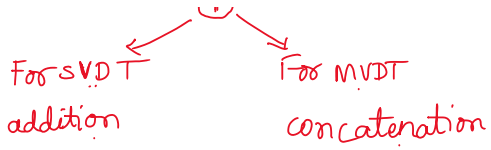
Programs:

```
# Hybrid Inheritance.

class School:
    sname = 'JES'
    loc = 'Hampi'
    princi = 'Nanda'

class College(School):
    clgname = 'Sai'
    HOD = 'Ranga'
    @classmethod
    def disp(cls):
        print(cls.clgname,cls.HOD,cls.sname,cls.loc,cls.princi)

class Degree:
    dname = 'BTech'
    sub_code = '17DAA45'

class Masters(School,Degree):
    uname = 'VTU'
    founder = 'Bala'
    @classmethod
    def display(cls):
        print(cls.uname,cls.founder,cls.sname,cls.loc,cls.princi,cls.dname,cls.sub_code)

College.disp()
Masters.display()
```

Day-33

Polymorphism:

--- It is a phenomenon of making the operator or method to perform two or more functionalities.



Types:

- Method Overloading
- Operator Overloading

1) **Method overloading**.
--- It is a phenomenon of making the method to perform two or more functionalities.

Note: In python method overloading is not supported, if we try to perform it , it will by default acts as method overriding.

Example:

```
# Method Overloading.
'''
class Demo:
   @staticmethod
   def sam():
      print('Good evening')
   @staticmethod
   def sam(a,b):
      print(a+b)
   @staticmethod
   def sam(a,b,c):
      print(a*b*c)
user = Demo()
user.sam(1,2,3)
user.sam(10,20) '''
```

Monkey Patching: The process of storing the address of the method in a variable so that the variable can be used as a method. To execute that method we have to call that variable.

```
#Monkey Patching

'''
class Demo:
   @staticmethod
   def sam():
      print('Good evening')
   x = sam
   @staticmethod
   def sam(a,b):
      print(a+b)
   y = sam
   @staticmethod
   def sam(a,b,c):
      print(a*b*c)
   z = sam
user = Demo()
user.x()
user.y(10,20)
user.z(1,2,3) '''
```

2) **Operator overloading**.
--- It is a phenomenon of making the operator to perform two or more functionalities by invoking the magic method.

Operation	Operator	Internal Operation
Addition	ob1 + ob2	ob1.__add__(ob2)
Subtraction	ob1 - ob2	ob1.__sub__(ob2)
Multiplication	ob1 * ob2	ob1.__mul__(ob2)
True Division	ob1 / ob2	ob1.__truediv__(ob2)
Floor Division	ob1 // ob2	ob1.__floordiv__(ob2)
Modulus	ob1 % ob2	ob1.__mod__(ob2)
Power	ob1 ** ob2	ob1.__pow__(ob2)

Encapsulation:
--- The phenomenon of providing security to the data members or variables or methods stored inside the class by restricting the user to access.

They are also called as Access Specifiers.

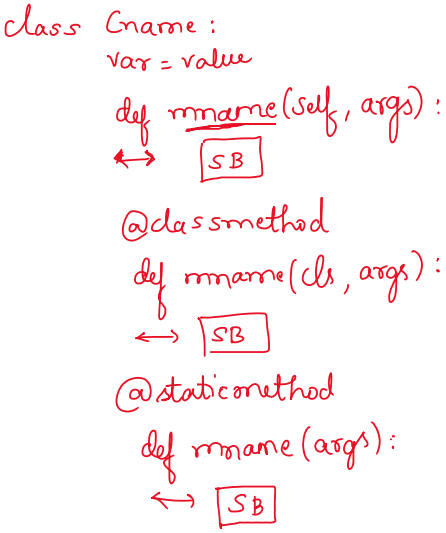
Access Specifiers : Members of the class which will inform the user whether they can access them outside the class or not.

Types:

- Public
- Protected
- Private

1) **Public**: Members of the class that can be accessed and modified outside the class.

Syntax:




```
#Encapsulation
'''
# Public

class Demo:
   a = 10
   b = 20
   def __init__(self,c ,d):
      self.c = c
      self.d = d
   def display(self):
      print(self.c,self.d)
   def ch_c(self,new):
      self.c = new
```

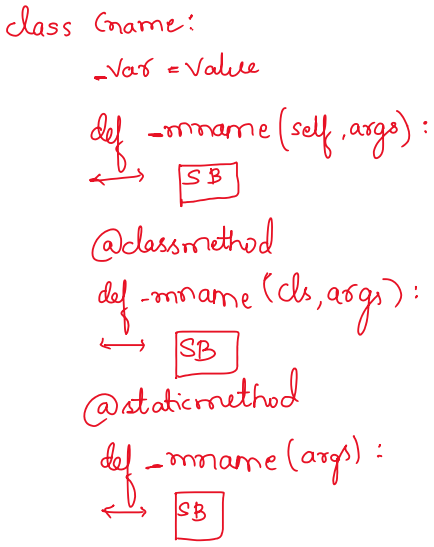
```
    @classmethod
    def disp(cls):
        print(cls.a,cls.b)
    @staticmethod
    def msg():
        print('Public Access Specifiers')
user = Demo(10,20)
user.display()
user.ch_c(40)
user.display()
'''

# Operator overloading.
'''
class A:
    a = 10
    b = 20
    def __init__(self, c,d):
        self.c = c
        self.d = d
    def display(self):
        print(self.c + self.d)
ob1 = A(30,15)
ob2 = A('hi','hello')
ob1.display()
ob2.display() '''
```

Day-34

1) Protected Access Specifier:

--- It should provide security to the data members present inside the class but by default it acts as public access specifier. The only difference between public and protected is the syntax.

Syntax:



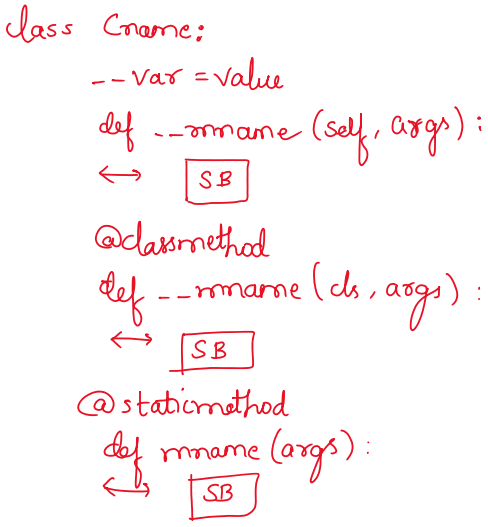
Program:

```
# Protected

class Demo:
    _a = 10
    _b = 20
    def __init__(self,c ,d):
        self._c = c
        self._d = d
    def _display(self):
        print(self._c,self._d)
    def _ch_c(self,new):
```

```
        self._c = new
    @classmethod
    def _disp(cls):
        print(cls._a,cls._b)
    @staticmethod
    def _msg():
        print('Protected Access Specifiers')
user = Demo(10,20)
user._display()
Demo._disp()
user._msg()
```

2) **Private Access Specifier**:
--- Data members which cannot be accessed and modified outside the class.

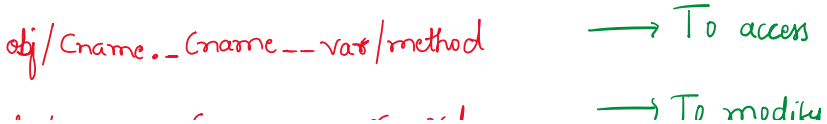
Syntax:




```
# Private
'''
class Demo:
    __a = 10
    __b = 20
    def __init__(self,c ,d):
        self.__c = c
        self.__d = d
    def __display(self):
        print(self.__c,self.__d)
    def __ch_c(self,new):
        self.__c = new
    @classmethod
    def __disp(cls):
        print(cls.__a,cls.__b)
    @staticmethod
    def __msg():
        print('Protected Access Specifiers')
user = Demo(10,20)
user.__display()
'''
```

There are 3 ways to access and modify the private data members outside the class

1) **Syntax method:**



obj/Cname._Cname__var/method ⟶ To access

obj/Cname._Cname__var = value ⟶ To modify

Programs:

```
# Syntax method
'''
class Demo:
   __a = 10
   __b = 20
   def __init__(self,c ,d):
      self.__c = c
      self.__d = d
   def __display(self):
      print(self.__c,self.__d)
   def __ch_c(self,new):
      self.__c = new
   @classmethod
   def __disp(cls):
      print(cls.__a,cls.__b)
   @staticmethod
   def __msg():
      print('Protected Access Specifiers')
user = Demo(30,40)
user._Demo__display()
Demo._Demo__disp()
user._Demo__c = 50
user._Demo__display()
'''
```

2) Using getter and setter method:

Syntax:

```
class Cname :
    def __init__ (self , var ) :
        self.__var = val
    def getter (self):
        return self.__var
    def setter (self,new):
        self.__var = new
obj = Cname (val)
print (obj.getter( ))
obj .setter (new-value)
```

To access

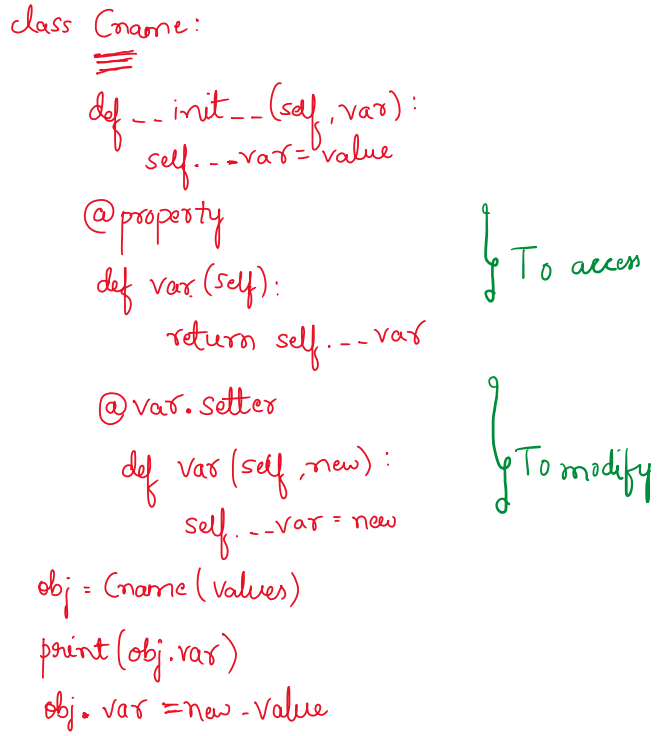
To modify

```
# Using getter and setter method
'''
class Demo:
   __a = 10
   __b = 20
   def __init__(self,c ,d):
      self.__c = c
      self.__d = d
   def getter(self):
```

```
        return self.__c,self.__d
    def setter(self,new):
        self.__d = new
user = Demo(30,40)
print(user.getter())
user.setter(50)
print(user.getter())
'''
```

3) Property decorator :

Syntax:




```
# Property decorator
'''
class Demo:
    __a = 10
    __b = 20
    def __init__(self,c ,d):
        self.__c = c
        self.__d = d
    @property
    def c(self):
        return self.__c
    @c.setter
    def c(self,new):
        self.__c = new
user = Demo(10,20)
print(user.c)
user.c = 70
print(user.c)

'''
```

Day-35

Abstraction:
--- It is used to hide the implementation from the user by making them to use the functionality .

Abstract method : Method which has only function declaration but not function definition.

Syntax:

```
class Cname :
    def mname(args) :
        pass
```

Abstract class : A class which consists of atleast one abstract method is called Abstract class.

Concrete class : A class consists of method which are not having any abstract method are considered as Concrete class

How to create Abstract class and Abstract method?

```
from abc import ABC, abstractmethod
class Cname(ABC):
    @abstractmethod
    def mname(args):
        pass
```

Abstract class

```
class Cname1(Cname):
    def mname(args):
        SB
obj = Cname()
```

Abstract class to Concrete class

Advance Concepts:

Lambda : It is a keyword which is used to perform small operation.

The programs which can be solved using simple functions, if and if-else concepts all those can be solved using lambda.

Note: Lambda is an anonymous function which is not having any name. So we will store the complete address of the function into a variable so that variable can act as function.

Syntax:

```
var = lambda args : expression
var(values)
```

```
# Abstraction
'''
from abc import ABC,abstractmethod
class Demo(ABC):
  @abstractmethod
  def sum(a,b):
    pass
  @abstractmethod
  def sub(a,b):
    pass
```

```
class Dude(Demo):
   @staticmethod
   def sum(a,b):
      print(a+b)
   @staticmethod
   def sub(a,b):
      print(a-b)
user = Dude()
user.sum(10,20)

'''


# Lambda
'''
def even(n):
   return n%2==0
print(even(72))

poo = lambda n:n%2==0
print(poo(71))
'''

# Program to check whether the string starting
# with vowel or not
'''
def check(s):
   return s[0] in 'AEIOUaeiou'
print(check('Apple'))

sam = lambda s:s[0] in 'AEIOUaeiou'
print(sam('Apple'))   '''

# program to add 2 numbers
'''
def add(a,b):
   return a+b
print(add(2,3))

demo = lambda a,b:a+b
print(demo(2,3))  '''
```

Day-36

To convert if-else program to lambda we have to use,

var = lambda args : TSB if cond else FSB

```
# From if-else to lambda conversion

# Program to check whether the string is
# palindrome or not. If it is palindrome then
# return the string as it is else return the
# reversed string.
'''
def sam(s):
   if s ==  s[::-1]:
      return s
   else:
      return s[::-1]
print(sam('python'))

sam = lambda s: s if s == s[::-1] else s[::-1]
print(sam('python')) '''

# Program to add minimum 3 numbers and maximum 5 numbers
'''
sam = lambda a,b,c,d=0,e=0 : a+b+c+d+e
```

```
print(sam(10,20,30,40,50)) '''

# Program to return concatenated list if both
# the list has same length else return list1
'''
sam = lambda l1,l2: l1+l2 if len(l1) == len(l2) else l1
print(sam([1,2,3],[4,5,6])) '''
```

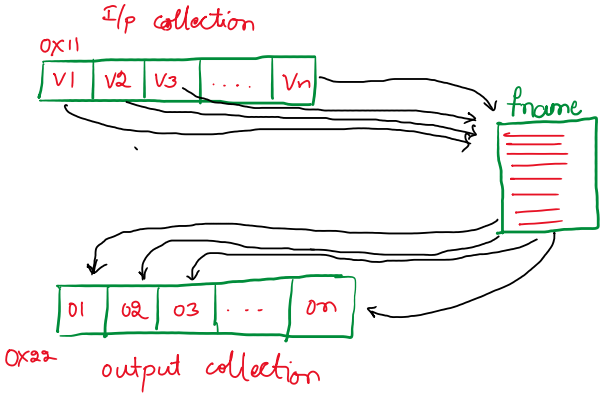
Assignment : Take any 10 if-else based programs and convert it to lambda

map():
--- It will consider a collection and it applies the same functionality to each and every value Present inside the collection.

Syntax:

```
var = map (fname, collection)
print(list(var))
```

Memory allocation:



Programs:

```
# Mapping

# Program to find the square of all the integer from 1 to 10
'''
a = lambda i: i**2
b = map(a,range(1,11))
print(list(b))   '''

# Get the following output
# input = ['abcd','start','data','python']
# output = [4,5,4,6]
'''
l = ['abcd','start','data','python']
a = lambda  i : len(i)
b = map(a,l)
print(list(b))   '''

# Get the following output
# input = 'hi hello how are you'
# output = ['hi','ho','hw','ae','yu']
'''
s = 'hi hello how are you'
a = s.split()
b = lambda i : i[0] + i[-1]
c = map(b,a)
print(list(c))   '''
```

```
# Get the following output
# output = {1:1,2:8,3:27,4:64,5:125}
'''
a = lambda  i : (i,i**3)
b = map(a,range(1,6))
print(dict(b))   '''

# Program to get the factorial of all the integers
# present inside the tuple
'''
import math
t = (5,10,15,20,25)
a = lambda i : math.factorial(i)
b = map(a,t)
print(tuple(b))   '''

# Get the following output
# input = 'programs on map function .....'
# output = {'programs':'smargorp','on':'no',.....}
'''
s = 'programs on map function'
a = s.split()
b = lambda i:(i,i[::-1])
c = map(b,a)
print(dict(c)) '''
```

Disadvantage in mapping:

- It requires only homogeneous values.
- It will take each and every value and applies the same functionality.

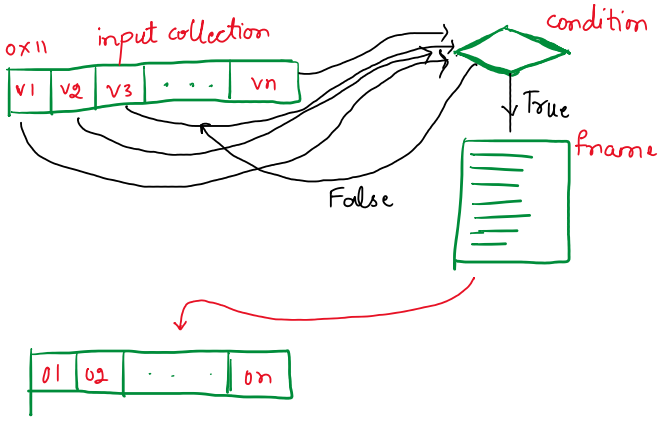
Day-37

filter():
--- It is used to remove unwanted values present inside the collection.

Syntax:

```
var = filter(fname, collection)
print(list(var))
```

Memory allocation:



Programs:

```
# Filter

# Program to extract even numbers between the
# range 1 to 10
'''
a = lambda i : i%2==0
b = filter(a,range(1,11))
```

```
print(list(b))   '''

# Extract all the string values from the tuple
# only if it is starting with uppercase and
# ending with lowercase
'''
t = (10,2.3,'Supritha','home','pythoN','Ugadi')
a = lambda i : type(i)==str  and 'A'<=i[0]<='Z' and 'a'<=i[-1]<='z'
b = filter(a,t)
print(tuple(b))  '''

# program to find the square of all the even numbers
# from the given list
'''
l = [1.0,2,3.0,4,5.0,6]
even = lambda i : type(i)==int and i%2==0
a = filter(even,l)
b = lambda i : i**2
c = map(b,a)
print(list(c))   '''

# Program to extract all the collection values
# present in a list which has even length
'''
l = [10,2.3,'sakshi',[10,20,30],(7,8),{1,2,3,4}]
a = lambda i : type(i) in [str,list,tuple,set,dict] and len(i)%2==0
b = filter(a,l)
print(list(b))   '''
```

Day-38

Comprehension:

-- The phenomenon of creating a new collection by using less number of instructions.

Note: We can use only mutable collections.

Types:

- List comprehension
- Set comprehension
- Dictionary comprehension

1) **List comprehension**:

--- The phenomenon of creating a new list.

Syntax:

* var = [val for var in collection if cond]
* var = [TSB if cond else FSB for var in collection]
* var = [var1, var2 for var1 in collection for var2 in collection]

Practical proof:

```
# List Comprehension

# program to create a list consists of 1 to 10 numbers.
'''
out = []
for i in range(1,11):
   out.append(i)
print(out)   '''

'''
a = [i for i in range(1,11)]
```

```
print(a)

print([i for i in range(1,11)])
'''
# Program to create a list consists of squares
# of each and every integer between the range(1,21)
# only if it is multiple of 3
'''
out = []
for i in range(1,21):
   if i%3==0:
      out.append(i**2)
print(out)   '''

'''
a = [i**2 for i in range(1,21) if i%3==0]
print(a)    '''

# program to extract string from the list only if it is palindrome
'''
l = ['hi',100,3.2,'madam','appa','bye']
a = [i for i in l if type(i)==str and i == i[::-1]]
print(a)   '''

# Program to store the squares of integers if it
# is even else store cube of the integers from
# the range(1,11)
'''
a = [i**2 if i%2==0 else i**3 for i in range(1,11)]
print(a)  '''

# Get the following output
# s = 'programs based open comprehension happy'
# out = ['programs','bd','open','cn','hp']
'''
s =  'programs based open comprehension happy'
a = s.split()
a = [i if len(i)%2==0 else i[0]+i[-1] for i in a]
print(a)   '''

# Get the following output
# [('A',1),('A',2),('A',3),('B',1)('B',2),('B',3)]
'''
a = [(i,j) for i in 'AB' for j in range(1,4)]
print(a) '''

# Assignment

# Get the following output
# [('a',1),('a',4),('a',9),('b',1)('b',4),('b',9)]
```

2) Set Comprehension:

--- The phenomenon of creating a new list.

Syntax:

* var = { i for var in collection if cond }
* var = { TSB if cond else FSB for var in collection }
* var = { var1, var2 for var1 in collection for var2 in collection }

Programs:

```
# Set Comprehesion

# Program to find the square root of the numbers
# from the range 1 to 10
'''
import math
a = {math.sqrt(i) for i in range(1,11)}
print(a) '''

# Get the following output
# s = 'Hey Python coders are Intelligent'
# out = {'HEY','PYTHON', 'sredoc','era','INTELLIGENT'}
'''
s = 'Hey Python coders are Intelligent'
a = s.split()
b = [i.upper() if  'A'<=i[0]<='Z' else i[::-1] for i in a]
print(b)

s = 'Hey Python coders are Intelligent'
a = s.split()
out = []
for i in a:
   if 'A'<=i[0]<='Z' :
      out.append(i.upper())
   else:
      out.append(i[::-1])
print(out)   '''

# Get the following output
# stud = ['Vineetha','Ashish','Ayan','Poornima']
# subj = ['Full-Stack','Data Analysis','Testing']
'''
stud = ['Vineetha','Ashish','Ayan','Poornima']
subj = ['Full-Stack','Data Analysis','Testing']
a = [(i,j) for i in stud for j in subj]
print(a)   '''

# Assignment
# Convert List comprehension programs to Set
# Comprehension
```

Day-39

zip(): Whenever we want to traverse through multiple collection in a single for loop we will go with zip() function.

Syntax:

for var1, var2 . . . varn in zip(col1, col2, . . . , coln) :

⟷ SB

Dictionary Comprehension:

--- It is a phenomenon of creating a new dictionary.

Syntax:

* var = { k:v for var in collection if cond }

* var = { k:v for var1, var2 in zip(col1, col2) if condition }

* var = {k:v1 if condition else v2 for var1, var2 in zip(col1, col2) }

Programs:

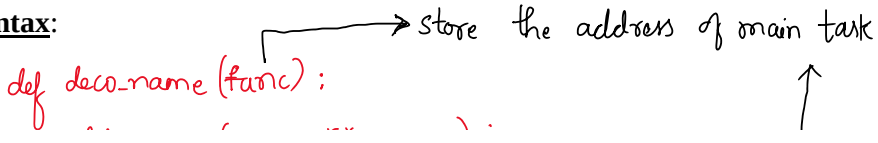
```
# zip()
'''
for i,j in zip('hii','bye'):
   print(i,j)   '''

# Dictionary Comprehension

# Get the following output
# out = {1:1,2:4,3:9,4:16,5:25}
'''
a = {i:i**2 for i in range(1,6)}
print(a)  '''

# Get the following output
# input = 'hello baby are you fine'
# out ={'hello':5,'are':3,'you':3}
'''
s = 'hello baby are you fine'
a = s.split()
b = {i:len(i)  for i in a    if len(i)%2==1}
print(b) '''


# Get the following output
# input = ['nayan','abcd','data','appa']
# out ={'nayan':5,'abcd':'dcba','data':'atad','appa':4}
'''
l = ['nayan','abcd','data','appa']
a = {i:len(i) if i == i[::-1] else  i[::-1]  for i in l}
print(a)   '''


# Get the following output
# input = range(1,11)
# out = {1:1,2:4,3:27,4:16,5:125,.....}
'''
a = {i:i**2 if i%2==0 else i**3 for i in range(1,11)}
print(a)     '''


# Get the following output
# a =[1,2,3]
# b = [4,5,6]
# out = {1:4,2:5,3:6}
'''
a =[1,2,3]
b = [4,5,6]
c = {i:j for i,j in zip(a,b)}
print(c)    '''
```

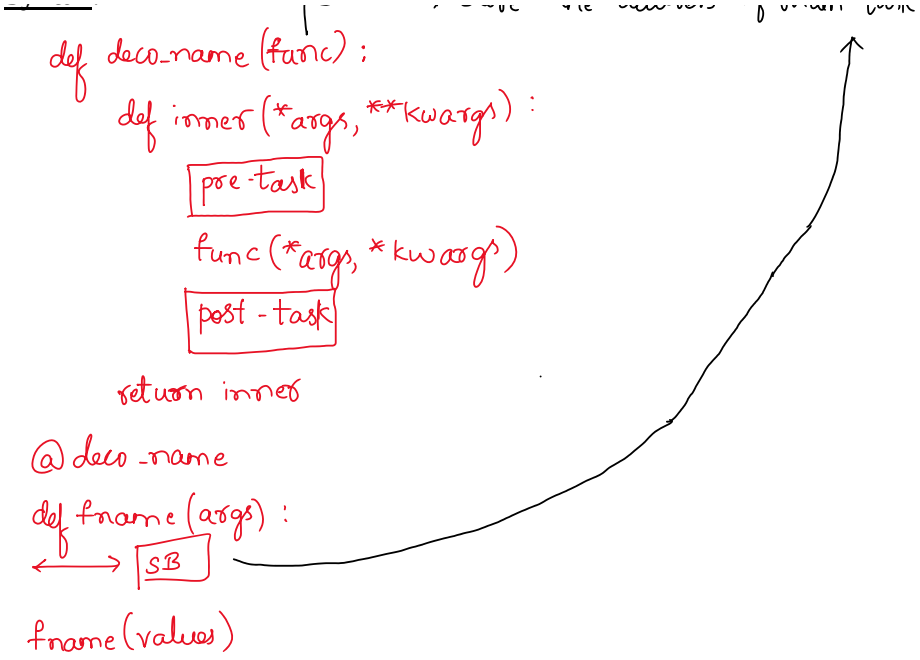
Decorator:
--- It is a function which is used to add more functionalities to the existing function.

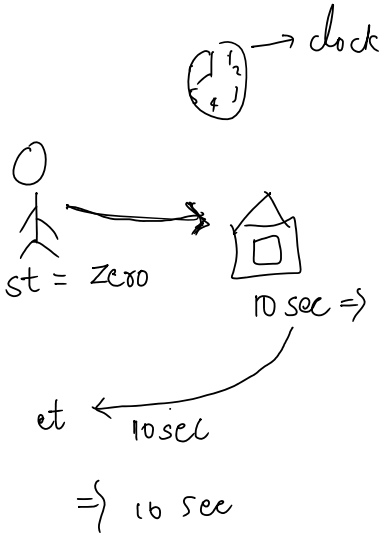
Types:

- **Inbuilt decorator:** @property, @classmethod, @staticmethod, @abstractmethod
- **User-defined decorator** : The decorators which are defined by the user.

Syntax:







Programs:

```
# Decorator
'''
def insta(func):
    def inner(*args,**kwargs):
        print('www.insta.com')
        print('Signup')
        print('Login')
        func(*args,*kwargs)
        print('Logout')
    return inner
@insta
def demo():
    print('adding posts')
    print('likes and comments')
    print('blocking some accounts')
    print('hiding stories and posts')
    print('making account private')
demo()   '''

# Program to keep the track of office spending time
'''
import time
def office(func):
    def inner(*args,**kwargs):
        print('Login')
        st = time.time()
        func(*args,*kwargs)
        print('Logout')
        et = time.time()
        print('Working hours = ', et-st)
    return inner
@office
def demo():
    print('Working .....')
    time.sleep(15)
demo()    '''

#program to wish Happy Ugadi to all the employees
'''
def fest(func):
    def inner(*args,**kwargs):
```

```
      print('Hi')
      print(input('Enter the name: '))
      func(*args,*kwargs)
      print('Thank You')
      print('with regards HR')
   return inner
@fest
def demo():
    print('Happy Ugadi')
demo()    '''

# program to return only positive results
'''
def task(func):
   def inner(*args,**kwargs):
      print('take the input')
      c = func(*args,*kwargs)
      print('Done')
      return abs(c)
   return inner
@task
def add(a,b):
    return a+b
print(add(-10,-20))    '''
```

Day-40

Generator:

--- The phenomenon of creating/generating a new collection by using yield keyword.

Yield --- It is going to pause the execution to store the value and it comes back to further instruction execution.

```
# Generator
'''
def sam():
   print('hello')
   yield 1
   print('hi')
   yield 2
   print('bye')
   yield 3
print(list(sam()))   '''

# Program to create a square of all the integers
# from the range  1 to 10
'''
def square(out = []):
   for i in range(1,11):
      out.append(i**2)
   print(out)
square()   '''

'''
def square():
   for i in range(1,11):
      yield i**2
print(list(square()))   '''

# Program to extract all the string from the
# list only if it is palindrome.
'''
def extract(l):
   for i in l:
      if type(i)==str and  i==i[::-1]:
         yield i
```

```
print(list(extract([10,2.3,'ada','happy','appa','bye'])))
'''

# Get the following output
# out = {'A':65,'B':66,'C':67,.....,'Z':90}
'''
def sample():
   for i in range(65,91):
      yield (chr(i),i)
print(dict(sample()))   '''

# Get the following output
# input = 'GenERat@r2'
# out = {'G':'g','e':'E',....'@':'@','r':'R','2':'2'}
'''
def sam(s):
   for i in s:
      if 'A'<=i<='Z':
         yield (i,i.lower())
      elif 'a'<=i<='z':
         yield (i,i.upper())
      else:
         yield (i,i)
print(dict(sam('GenERat@r2')))   '''

# Assignment

# Get the following output
# input = range(1,11)
# output = {1:1,2:8,3:9,4:64,5:25.....10:1000}
```

Iterator:
--- The process of traversing through each and every value in the collection is called Iteration. The function which is Used for iteration is called as Iterator.

Types:

- Built-in iterator --- for loop
- User defined iterator --- while loop, iter() and next()

Example:

```
# Iterator
'''
l = [1,2,3,4,5,6,7]
for i in l:
   print(i)

l = [1,2,3,4,5,6,7]
i = iter(l)
print(next(i))
print(next(i))
print(next(i))
print(next(i))
print(next(i))
print(next(i))
print(next(i))
print(next(i))   '''

# Task given

# b = {'a',1,2.3,True,(10,20),2+3j}
```

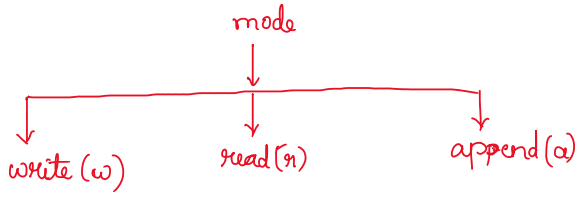
Day-41

File Handling:
--- The phenomenon of writing a data into file and reading the data from the file.

First step is to open the file,

Syntax:

var = open('file.ext' / loc ', 'mode')



Working with respect to text file.

1) write(w):

---

* write()

Syntax: var.write(data) → It is used to write single data.

* writelines()

Syntax:- var.writelines(data) → It is used to write multiple data

Programs:

```
'''
a = open('file1.txt','w')
data = "I'm not interested in you"
a = a.write(data)  '''

'''a = open('file1.txt','w')
data = ['hi ','beautiful ','tell ','me ','brother']
a = a.writelines(data)  '''
```

Alternate syntax for opening the file:

with open('file.ext'/loc,'mode') as var :
    var.write(data)

```
'''
with open(r"C:\Users\saksh\OneDrive\Desktop\file\file1.txt",'w') as a:
   data = 'Hi Samrudh'
   a = a.write(data) '''

with open('file1.txt','w') as a:
   data = ['hi ','hello']
   a = a.writelines(data)
```

2) **Read Operation:**

---

read()
⇒ var.read()

readline()
⇒ var.readline()

readlines()
⇒ var.readlines()

Programs:

```
'''
with open('file1.txt','w') as a:
   data = ['hi\n','hello\n','how\n','are\n','you'  ]
   a = a.writelines(data) '''

'''
with open('file1.txt','r') as a:
   a = a.read()
   print(a)

with open('file1.txt','r') as a:
   a = a.readline()
   print(a)

with open('file1.txt','r') as a:
   a = a.readlines()
   print(a) '''
```

3) append operation(a):

---

* write()
⇒ var.write(data)

* writelines()
⇒ var.writelines(data)

Program:

```
with open('file1.txt','a') as a:
   data = '\ngood-bye'
   a = a.write(data)

with open('file1.txt','a') as a:
   data = ['\ngood boy\n','Bad sister\n','Best brother\n']
   a = a.writelines(data)
```

Practical and real time application of file handling.

```
with open('file1.txt') as a:
   a = a.read()
   b = a.split()
   print(len(b))
```

Note: If u don't mention the mode of operation by default it will take read operation.

Programs:

```
# Program to count number of words present
# in a given text file
'''
with open('file1.txt','w') as a:
   data = 'hi hello how are you'
   a = a.write(data)'''
'''
with open('file1.txt','r') as a:
   a = a.read()
   b = a.split()
   print(len(b))   '''
```

```
# Program to create dictionary where key
# will be the word and value will be the number
# of times of occurence of that word
'''
with open('file1.txt','r') as a:
   a = a.read()
   b = a.split()
   out = {}
   for i in b:
      out[i]=b.count(i)
   print(out)   '''
```

Day-42

File handling with respect to csv file.

Csv file : File which is consisting of comma seperated values.

1) **Writing(w) the data w.r.t csv file:**

```
import csv
with open('file.csv','w') as var:
    var1 = csv.writer(var)
```

* writerow ⇒ var1.writerow([val1, val2, ..., valn])
* writerows ⇒ var1.writerows([[val1, val2, ... valn], [val1, val2, ..., valn] ....])

Example:

```
# write operation
'''
import csv
with open('file1.csv','w') as a:
   b = csv.writer(a)
   b.writerow(['Supritha',35000,25])
   b.writerows([['Akash',25000,30],['Rahul',27000,28],['Megha',30000,27]])
'''
```

2) **Read(r) with respect to csv file:**

```
import csv
with open('file.csv/loc','r') as var:
    var1 = csv.reader(var)
    print(var1)
```

Example:

```
# Read operation

'''
import csv
with open('file1.csv','r') as a:
   b = csv.reader(a)
   print(list(b))   '''

# Program to get the details of all the employees
# whose salary is greater than 25000
'''
import csv
with open('file1.csv','r') as a:
   b = csv.reader(a)
```

```
    c = [i for i in b if i!=[]]
    for i in c:
        if eval(i[1])>25000:
            print(i)   '''

# Program to get the details of all the employees
# whose name starts with vowel
"""
import csv
with open('file1.csv','r') as a:
    b = csv.reader(a)
    c = [i for i in b if i!=[]]
    for i in c:
        if i[0][0] in 'AEIOUaeiou':
            print(i)    '''

# task
# Program to get the details of all the employees
# whose age is even
```

3) Append(a) with respect to csv file:

```
import csv
with open('file.csv/loc', 'a') as var:
    var1 = csv.writer(var)
    var1.writerow([val1, ....valn])
    var1.writerows([[val1 ....valn], [val1, ....valn], .....])
```

Example:

```
# Append Operation

'''
import csv
with open('file1.csv','a') as a:
    b = csv.writer(a)
    b.writerow(['Rudresh',35000,25])
    b.writerows([['Spoorthi',25000,30],['Nidhi',27000,28],['Parvathi',40000,27]])
'''
```

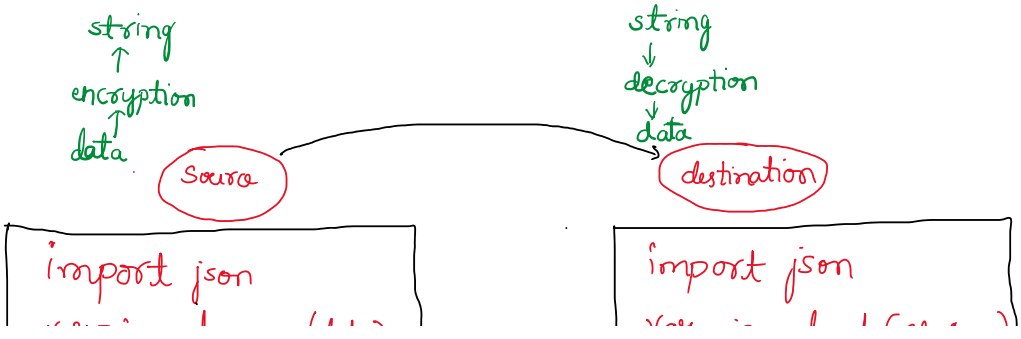
Parsing Technique:

--- The process of providing security to the data while sending from source to destination

Types:

- **Json**
- **Pickle**

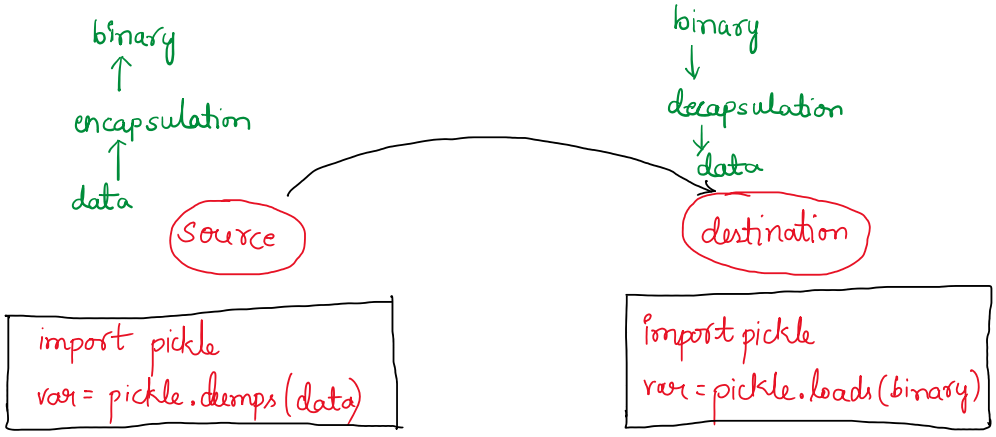
1) Json parsing technique:



Example:

```
# json
'''
import json

data = [10,20,30,40,50]
b = json.dumps(data)
print(type(data),data)
print(type(b),b)

with open('file1.txt','w') as a:
   a = a.write(b)

with open('file1.txt','r') as a:
   a = a.read()
   b = json.loads(a)
   print(type(b),b)   '''
```

2) Pickle parsing technique:



Example:

```
# pickle parsing
'''
import pickle

data = [10,20,30,40,50]
b = pickle.dumps(data)
print(data)
print(b)

with open('file1.txt','wb') as a:
   a  = a.write(b)

with open('file1.txt','rb') as a:
   a = a.read()
   b = pickle.loads(a)
   print(b)   '''
```

Day-43

Modules:

1) math module:

```
import math
math.sqrt(8)
2.8284271247461903
math.factorial(5)
120
math.lcm(2,8,16)
16
math.gcd(5,10)
5
math.pi
3.141592653589793
```

2) random module:

```
import random
random.random()
0.32204350635189194
random.random()
0.5353827657444836
random.randint(1,100)
90
random.randint(1,100)
85
random.randint(1,100)
11
random.randint(1,100)
50
random.choice('Modules')
'M'
random.choice('Modules')
's'
random.choice('Modules')
'u'
random.shuffle([10,20,30,40])
a = [10,20,30,40]
random.shuffle([10,20,30,40])
a
[10, 20, 30, 40]
random.shuffle(a)
a
[10, 20, 40, 30]
a
[10, 20, 40, 30]
random.shuffle(a)
a
[20, 10, 30, 40]
random.shuffle(a)
a
[20, 10, 40, 30]
```

3) time module:

```
import time
time.time()
1744036083.114272
time.sleep(5)
```

4) calendar module:

```
import calendar
calendar.calendar(2025)
```

```
calendar.month(2025,12)
'   December 2025\nMo Tu We Th Fr Sa Su\n 1  2  3  4  5  6  7\n 8  9 10 11 12 13 14\n15 16 17 18 19 20 21\n22
23 24 25 26 27 28\n29 30 31\n'
```

Exception Handling:

Exception: It is an unauthorized event which will occur during program execution which will stop the flow of execution.

The process of handling exceptions using syntax is called as Exception handling.

Note: Syntax error cannot be called as Exceptions.

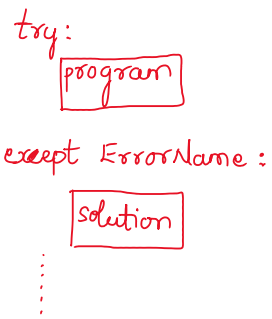
Types:

- Specific
- Generic
- Default

} try & except keywords are used

1) **Specific Exception handling:**
   --- When we know what type of error we will get.

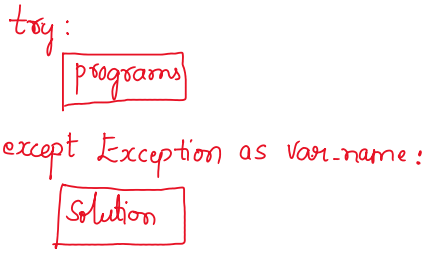
   Syntax:




```
# Specific

try:
   a = int(input('Enter the value for a: '))
   b = int(input('Enter the value for b: '))

   print(a/b)
except ZeroDivisionError:
   print('b value should not be 0')

except ValueError:
   print('a and b must be int')
```

2) **Generic Exception handling:**
   --- When we don't know the type of error that our program going to throw we will use Generic Exception handling.

   Syntax:



Example:

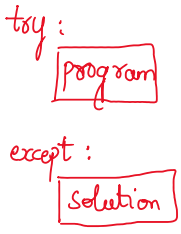
```
# Generic Exception Handling
'''
try:
   a = int(input('Enter the value for a: '))
   b = int(input('Enter the value for b: '))
   print(a/b)
except Exception as e:
   print(e)  '''
```

Disadvantage: Generic Exception Handling will not support for the Keyboard Interrupt error for the python version which is less than 3.92

3) Default Exception handling:

--- To overcome the drawback of generic exception handling we use default exception handling.

Syntax:

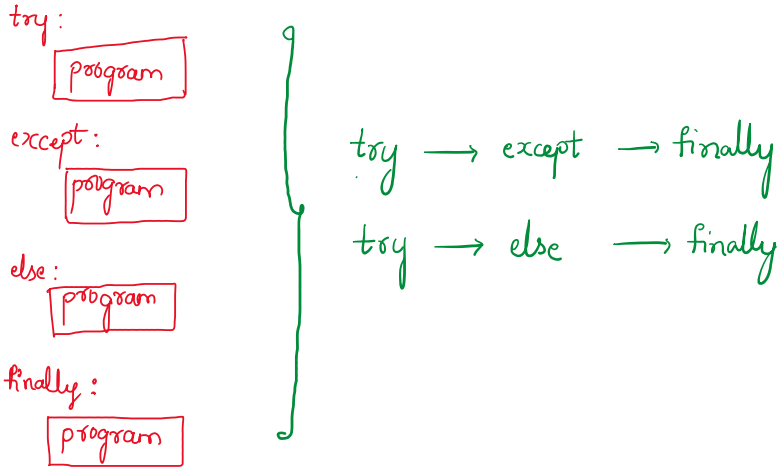


Example:

```
# Default exception handling
'''
try:
   a = int(input('Enter the value for a: '))
   b = int(input('Enter the value for b: '))
   print(a/b)
except:
   print('Exception is Handled') '''
```

Note: We can handle exception using try, except, else , finally

Syntax:



Example:

```
# IPL match  (RCB vs MUMBAI)
'''
try:
   import random
   rcb_score = random.randint(100,180)
   mi_score = random.randint(100,180)
   a = rcb_score / 0
   b = mi_score / 0
except ZeroDivisionError:
   print('Team has not scored anything')
else:
   print('******* Team score *******')
   print('RCB Score=',rcb_score)
   print('MI Score=',mi_score)
finally:
   print('Total score=',rcb_score + mi_score)
'''
```

Day-44

Package architecture:
--- The process of dividing the project to number of modules.

Package ---- Folder
Modules --- Python File

Create a folder(package) and create a python file inside it , save it as file1 and write the below code,

```
def valid_un(s):
   if '_' in s:
      return True
   else:
      return False

print(valid_un('Saku_r'))

def valid_pw(s):
   if len(s)>=8:
      u,l,d,sc = 0,0,0,0
      for i in s:
         if 'A'<=i<='Z':
            u+=1
         elif 'a'<=i<='z':
            l+=1
         elif '0'<=i<='9':
            d+=1
         elif i in '@_$':
            sc+=1
      if u>=2 and l>=2 and d>=2 and sc>=2:
         return True
      else:
         return False
   else:
     return False
print(valid_pw('Saku_R@123'))
```

Create one more python file in the same folder, save it as file2 and run this code where we use the functionality of file1 in file2 by importing it.

```
import file1
un = 'Sakshi_27'
pw = 'Sakshi_R@123'
print(file1.valid_un(un))
print(file1.valid_pw(pw))
```

Create one python file outside the folder, save it as file3 and try to use the functionality of file1 in it by using this below code,

```
# Don't save inside the folder

from package import file1
un = 'Suppi_27'
pw = 'Suppi_R@123'
print(file1.valid_un(un))
print(file1.valid_pw(pw))
```

SQL Connection:

--- To connect python with the database we use SQL Connection.

- connect() -- To establish connection between the python file and database file.
  connect() will be present in sqlite3

Syntax:

```
import sqlite3
var = sqlite3.connect('data.db')
```

- cursor() --- It is used to point to the database file.

```
var1 = var.cursor()
```

- execute() --- It will take string input where we are going to mention all the queries which we want to execute.

```
var1 = execute('Query')
```

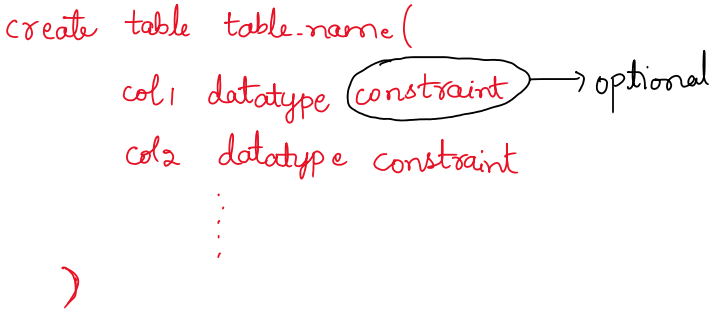
- commit() -- It is used to save all the changes which you are going to do for the database file.

```
var.commit()
```

- close() --- To remove the established connection .

Day-45

Table creation:



To insert values:

insert into table_name values (val1, val2, ...)

To extract data:

* Select * from tablename => all records | details
* Select colname from tablename => all records of one column
* Select * from tablename where condition

```
# install the extension Sqlite Viewer

import sqlite3
a = sqlite3.connect('data.db')
b = a.cursor()
'''
b.execute("create table Flipkart(prod_id  num, prod_name char)")
b.execute("insert into Flipkart values(1,'iphone')")
b.execute("insert into Flipkart values(2,'Oppo Reno7')")
b.execute("insert into Flipkart values(3,'Nokia')")

res = b.execute("Select * from Flipkart")
print(list(res))

res = b.execute("Select prod_id from Flipkart")
print(list(res))

res = b.execute("Select * from Flipkart where prod_id==1")
print(list(res))   '''
```

Assignment :

Create your own table of Students, which includes student_id, student_name, student_addr,student_marks as columns