PYTHON LANGUAGE

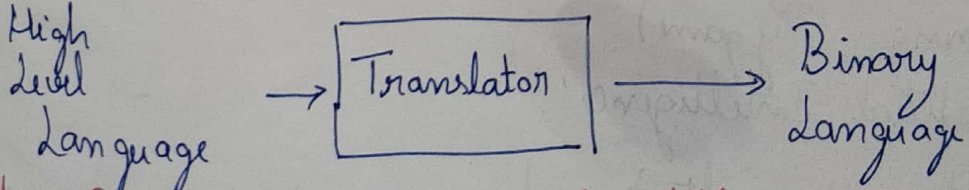
Language are of different type

1. Machine Language [Binary Language 0's & 1's]
2. Medium Level Language [Top to bottom]
3. High Level Language [Bottom to top]

Machine Language - It is the native language of a machine. Here we instruct the microprocessor to perform the activity. It is machine dependent.

Medium Level Language - Programming language like C is called medium level Language. This is because we can bind the gap between a machine level language and high level language. User can use C language to do system programming [For writing operating system] as well as application program.

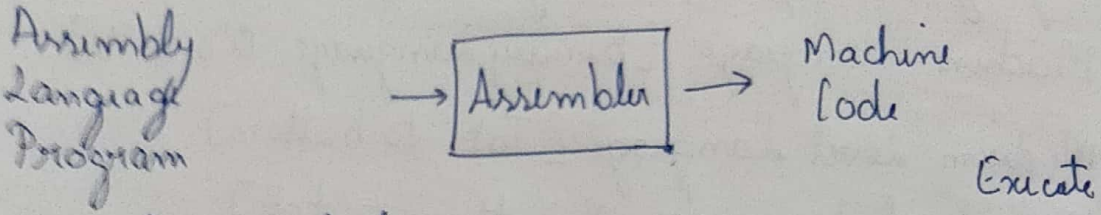
High Level Language Example of high level languag
C++, Java, Python, COBOL
- They are easy to learn and understand
- It is machine independent.
- Takes time to translate.



Why C is called medium / middle level language
It has least keyword, easy to convert and can be use for both OS & application.

Assembly Language - This is also low level language.

Here it will use assembler.



Execution is faster

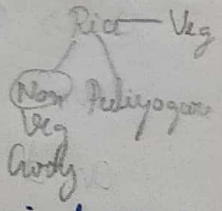
Purpose based language

Pascal -

Fortron

Cobolt - Business

Library are good in python



General Editors and IDE with python support

- Eclipse + PyDev, Sublime Text, Atom, VIM, Visual Studio Code, PyCharm, Spyder

Why is Python Used?

Python is a scripting language like PHP, Perl Ruby and so much more. It can be used for

1. Web programming (django, Zope, Google App Engine)
2. Desktop Application (Blender 3D)
3. Mobile testing (Appium)
4. Gaming (Pygame)
5. Artificial Intelligence

Python is multi paradigm

Hello World program can be written in one line.

History of Python

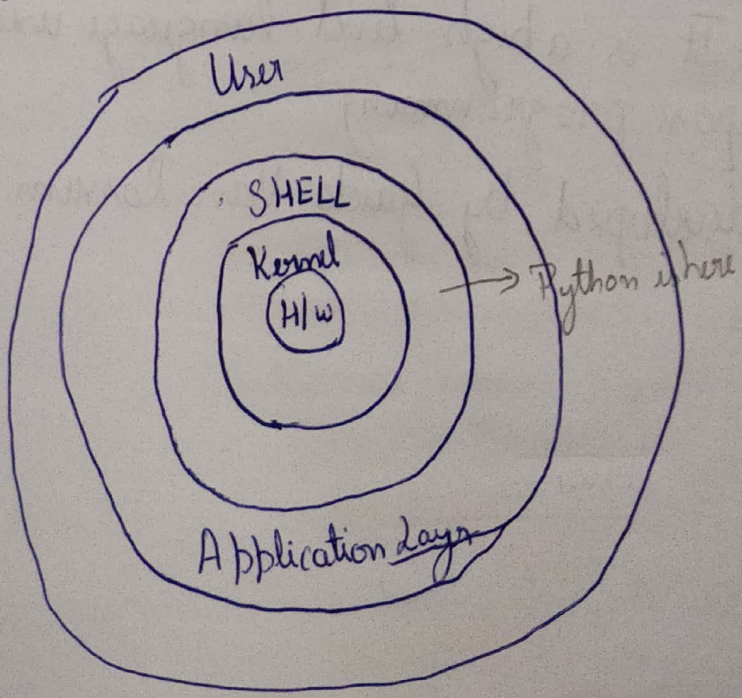
Python was developed by Guido Van Rossum in 1991.
It was done for programming for kids.

Installation

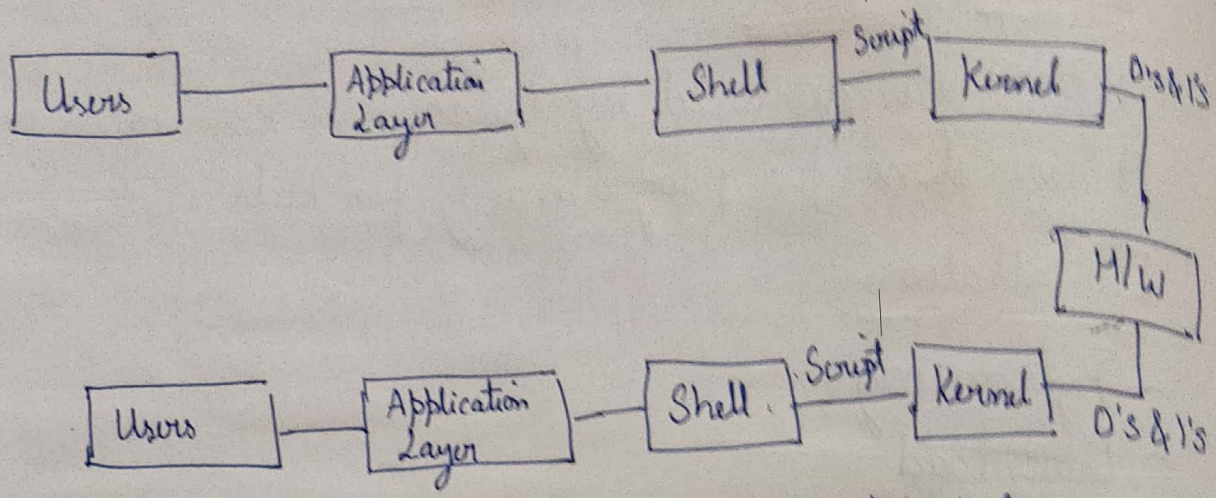
- Python.org
- Download
- 3.7
- Download - Window - Python 3.7.0
  3.6.6 → Window x 86 - executable installer
- Run as administrator
- Add python 3.6 to path
- Install

Search
- IDLE
- Taskbar

>>> 10+20 =30



OS Architecture



Kernel is responsible for the communication between hardware and user.

Basically kernel is having predefined compiler in it.

To work in python we need to add interpreter at kernel ie One installation of application compiler of respective application will be added to kernel

Python resides in shell layer of OS.

Shell is used for communication.

DEFINATION

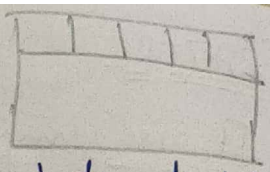
Python - It is a high level language used for general purpose programming.

It was developed by Guido Van Rossum. Released in 1991

FEATURE

- It is simple
- As there is no concept of braces and semicolon. Hence it is simple & also called programmer friendly language.
- It is cross platform language. Since python can run on multiple OS. Hence it is called as cross platform language.
- It is also called as interpreter language. Python makes use of interpreter which executes the code line by line where it makes the debugging easier
- It has huge built in features
- It is open source.

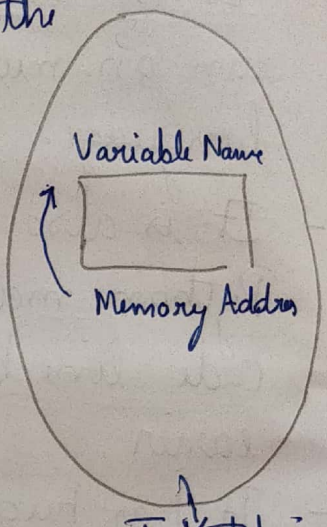
MEMORY



- Memory is divided into blocks where the blocks may contain bits of data or bytes of data.

VARIABLE

- Variable is named memory location which has address
- The address will be bendeled up with the variable name to identify the memory
  Variable Name = Value
- Each & every variable will store single value



IDENTIFIERS

- Variable Names are known as identifiers
- Identifiers are the name giving to identify the memory

RULES TO DEFINE IDENTIFIERS

① Identifiers must always start with an alphabet or underscore.

② Identifiers can't be a number because. One value can't identify another value or number.

```
>> a=2  ✓
>> 1=2  ×   SYNTAX ERROR : can't assign to literal
Syntax Error: Can't assign to literal.
>> _ = 20        >>> _
>> _ = 20        20
```

- No special symbol apart from underscore can be used as identifier.

- It can be a combination of alpha-numeric but should always start with alphabet or _

```
>> @ = 12
```
Syntax Error: Invalid Syntax

- Identifier can't be predefined keywords or words

KEYWORDS

Keywords are the reserved words which has some predefined meaning.

```
>>> import keyword
>>> keyword.kwlist
['False', 'None', 'True', 'and', 'as', 'assert', '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']
```

DATATYPES

- Datatypes specifies the type of data stored inside a variable and it is also used to convert the data from one data type to another datatype.

① INT - It specifies the data stored inside a memory with an integer value

```
>>> a = 10
>>> type(a)
  <class 'int'>
>>> b = int(10.3)
>>> b
  10
>>> int(13.40)
  13
```

② FLOAT - It specifies the data stored inside the memory is a decimal value.

```
>>> n = 12.54
>>> n
  12.54
>>> type(n)
  <class 'float'>
>>> float(10)
  10.0
```

Comment

#

Complex Number

```
>>> a = complex(7)
>>> a
  (7+0j)
```

Complex Number is a number which has both real & imaginary value.

```
>>> b = complex(2,4)
>>> b
  (2+4j)
```

Bool

- It is used to convert any value into either true or false.
- Boolean datatype has two kind of value ie True & false while converting the numberic value 0 is considered to be false and non zero value is considered as true.

In all the cases we will convert the value from numeric type like integer, float to a boolean.

```
var = bool (int/float)
>>> c = bool(4)
    true
>>> c = bool(-2)
    true
```

```
c = bool(0)
false
>> d1 = bool(10.2)
true
```

Grouped Datatype

Whenever there is collection of value or group of value to be stored, then we have to use grouped datatype.

① String

It is a collection of characters where all the characters are enclosed with pair of single quotes or double quotes

```
>>> a = 'Hello'
>>> id(a)
    2348
>>> a[0]
    'H'
>>> id(a[0])
    9999
```

INDEXING

Indexing is a phenomenon of assuming a index

```
>>> a = 'hello world'
>>> a[5]
```

This is entity store

LIST

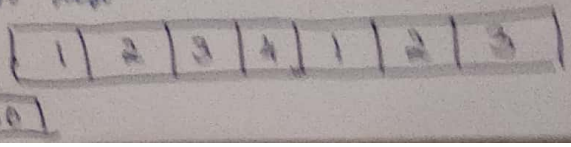
It is a collection of homogeneous & hetrogeneous object. Data item are separated by , (Comma Operator) and all the data item are enclosed with pair of [ ]

list var = [data1, data2, data3, data4]

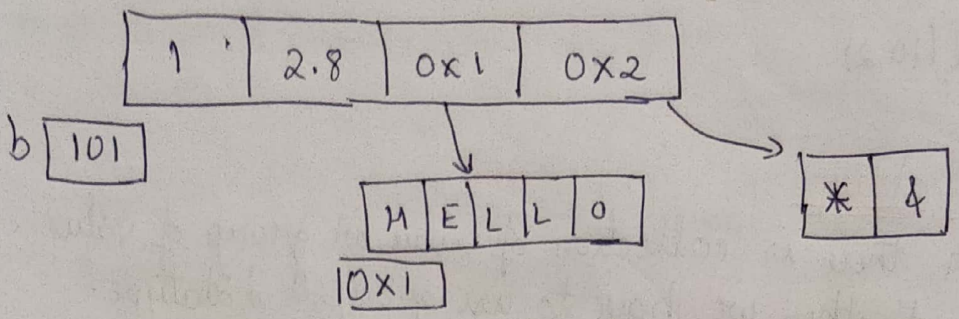
Homogeneous list a = [1, 2, 3, 4, 1, 2, 3]

Hetrogeneous list a = [1, 2, 3.4, 'Hello', 'Ad']

```
>>> type(a)
<class 'list'>
```



b = [1, 2.8, 'Hello', '*&']



If there is string then it will use pointer concept. String memory uses sub class

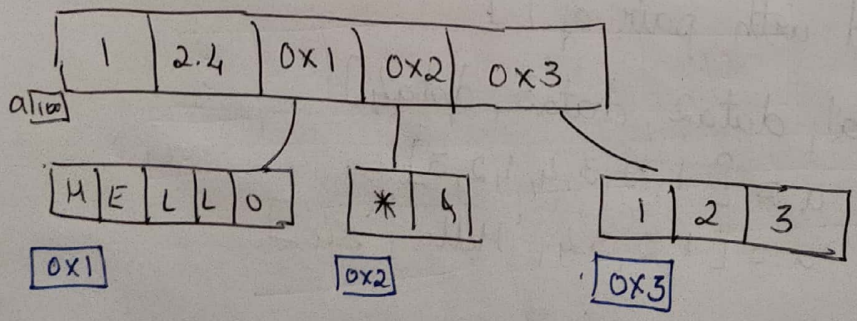
List is modifiable ie we can add an element to the existing list memory.

Ex:

```
a = [1, 2.4, 'Hello', '*&']
>>> id[a]
    21212
>>> a.append('Hi')
>>> a
 [1, 2.4, 'Hello', '*&', 'Hi']
>>> id(a)
    21212
>>> a = [1, 2.4, 'Hello', '*&', 'Hi', 2]
>>> id[a]
    99999
```

Indexing in list will work similar to the string.

```
>>> a = [1, 2.4, 'Hello', '*&', [1, 2, 3]]
>>> a
 [1, 2.4, 'Hello', '*&', [1, 2, 3]]
```



EMPTY LIST

```
a1=[]
>> type(a1)
<class 'list'>
```

To create an empty list we can use following methods

① list_var = []
② list_var = list()

To convert a string, set, tuple into list we have to use following syntax

List Var = list(var1)

var ⟶ string, set, tuple.

TUPLE

It is collection of homogeneous and hetrogeneous data items where each and every data item are separated by comma and all the data items are enclosed with pairs of paranthesis.

Tuplevar = (data1, data2, data3) or
Tuplevar = data1, data2, data3

```
>>> a = 1,2,3,4
>>> type(a)
>> b = (1,2,3,4.5)
type(b)
<class 'tuple'>
```

Tuple is immutable. Can't change or append tuple.

In python tuple is immutable ie we can't modify a tuple.

The operation which can be done on the tuple are the operations which doesn't affect the tuple variable.

* Alt+P will give you the previous data.

```
>>> c = (3, 4.5, 5, 'tarun', [1, 10, 100])
>>> type(c)
  <class 'tuple'>
>>> id(c)
  223311
>>> c = (3, 4.5, 5, 'tarun', [1, 10, 100], [1, 2, 3])
```

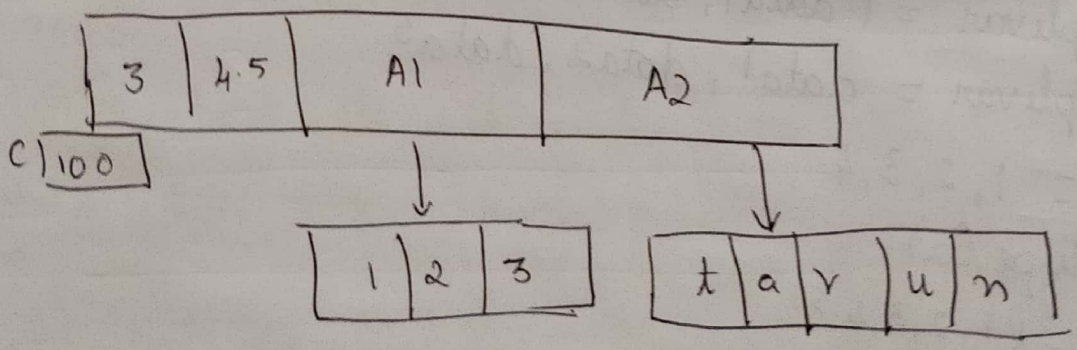
Change in memory.

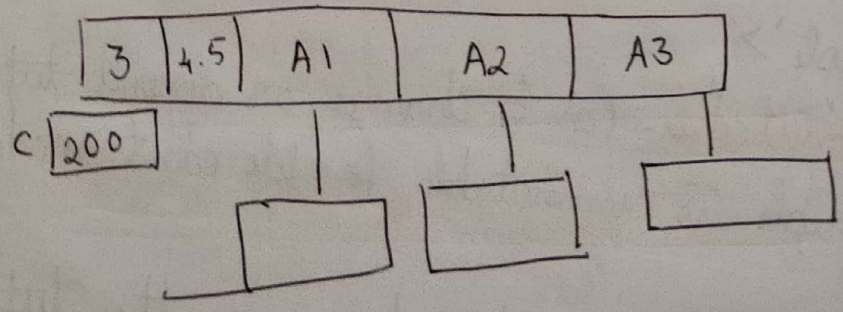
```
>>> type(c)
  <class 'tuple'>
```

Tuple change the memory

```
>>> id(c)
  334455
>>> name = 'tarun'
>>>  age = 28
>>> salary = 37442
>>> c = (name, age, salary)
>>> c
  ('tarun', 28, 37442)
>>d = (3, 4.5, (1, 2, 3), 'tarun')
```

→ Tuple has variable





SET

Set is rarely used in concept like in graph plotting.
It is a collection of non repeated homogeneous & heterogeneous data item which are separated by comma and enclosed with the pair of {}
Indexing is not possible on set ie. It is not possible to identify set element using index. Here the repeated values will be removed if any present during the initialization.

```
>>> a={1,2,3,1,2,3}
>>> a
    {1,2,3}
  a[1]
```

Error will occur
Type Error: 'set' object does not support indexing.
Set is mutable ie we can modify the set.

```
>>> a
    {1,2,3}
>>> id(a)
    1234
>>> a.add(4)
>>> a
    {1,2,3,4}
>>> a={}
    type(a)
    <class 'dict'>
```

} It will create dictionary.

To create an empty set

```
  a=set()
>>> type(a)
<class 'set'>
```

To convert the data from any group data type to set syntax is

variable = set(var1)

var = set(var1)

↳ Iterable

where var1 can be grouped datatype like Tuple or list or string

DICTIONARY

It is a collection of key & values where each and every key and value pairs are separated by comma(,) and each key and value are separated by colon(:)

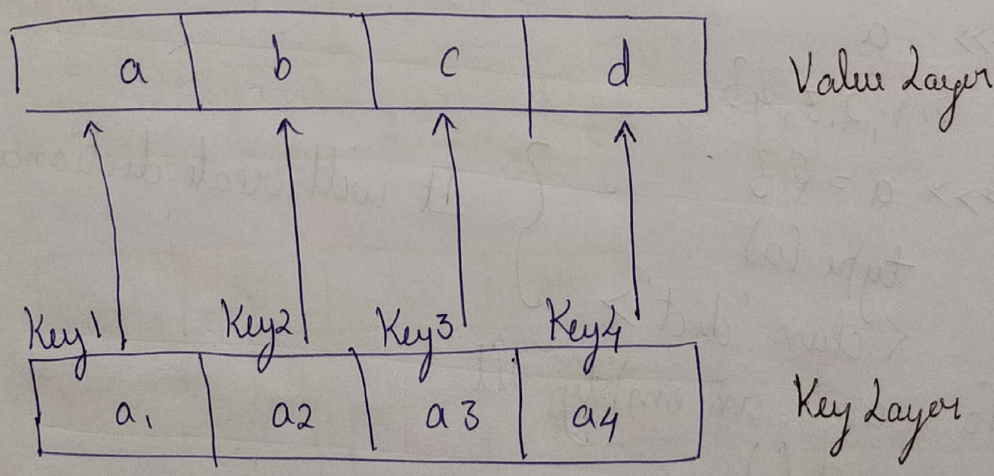
dict var = {'key1':'value1', 'key2':'value2', ... }

Collection of key value pair

'name': {'firstName': 'Tarun', 'LastName': 'Shetty'}

'name': ({['Ravi', 'Shastri']})

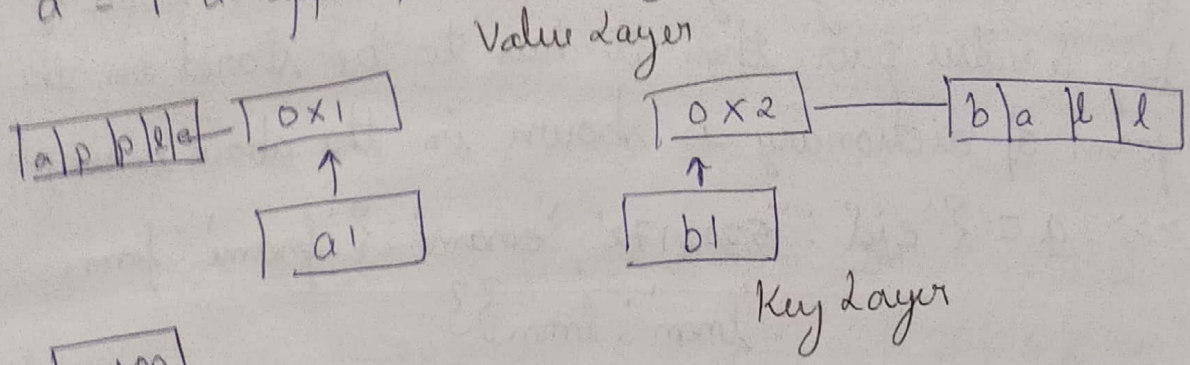
dictvar['key1']

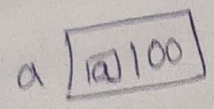


dict var1

100

a = {'a1':'apple', 'a2':'ball'}



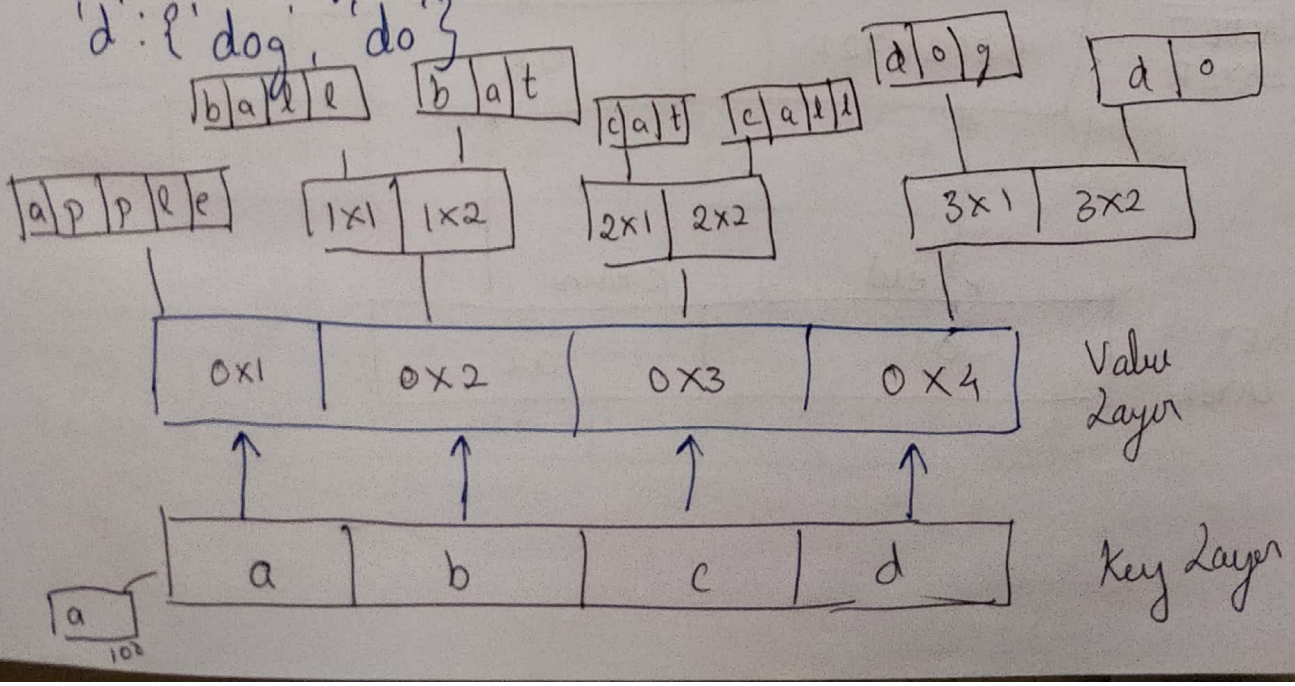



```
>>> a = {'a' : 'apple', 'b' : 'ball'}
>>> a['a']
'apple'
>>> a = {'a' : 'apple', 'b' : 'ball', 'b' : 'bat'}
>>> a
{'a' : 'apple', 'b' : 'ball'}
```

We can't store duplicate keys to the dictionary. Whenever we tried to store duplicate keys to the value will be updated as shown above

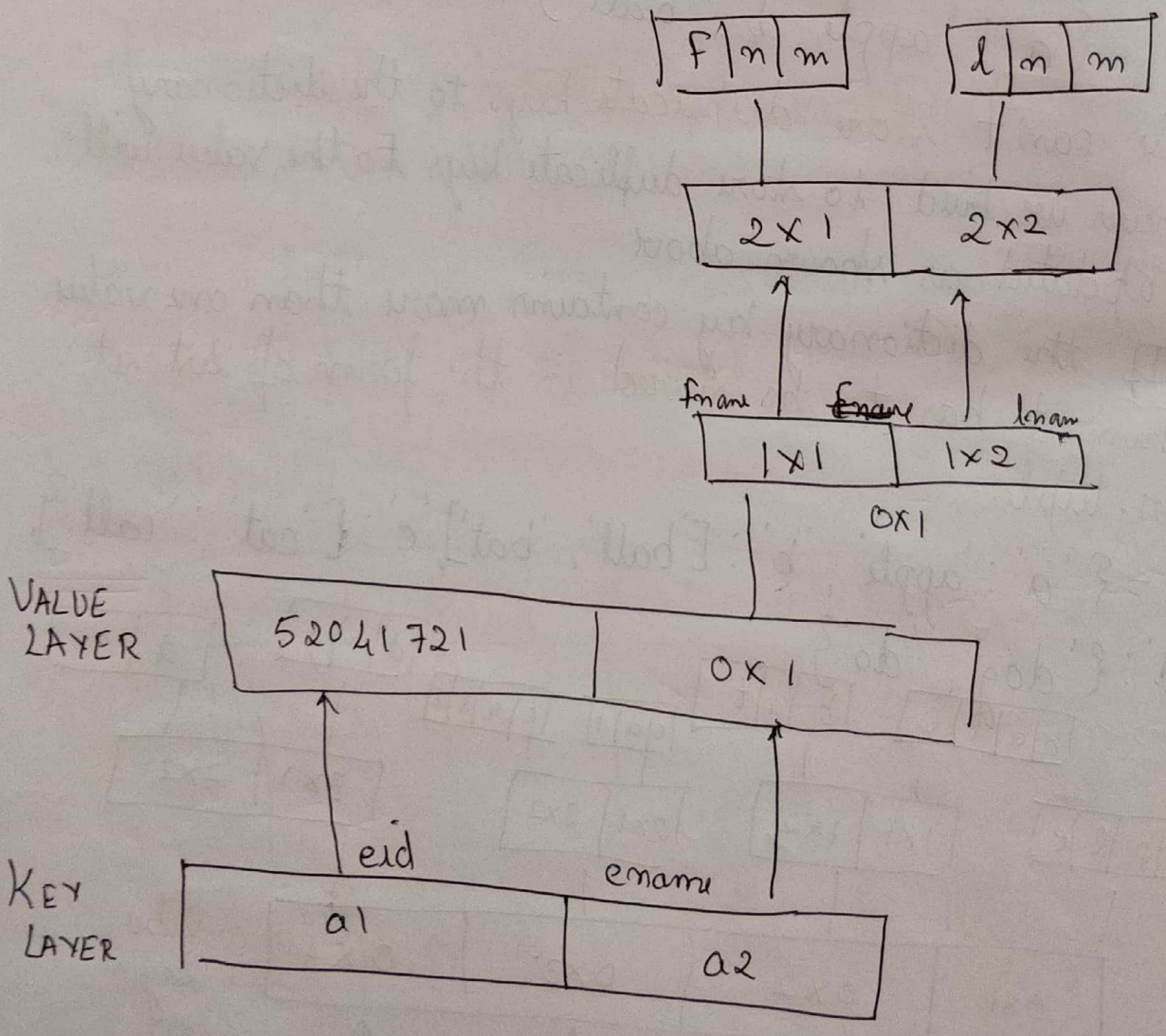
If the dictionary key contains more than one value then it has to be stored in the form of list, set or tuple.

a = {'a' : 'apple', 'b' : ['ball', 'bat'], 'c' : {'cat' : 'call'}, 'd' : {'dog', 'do'}}

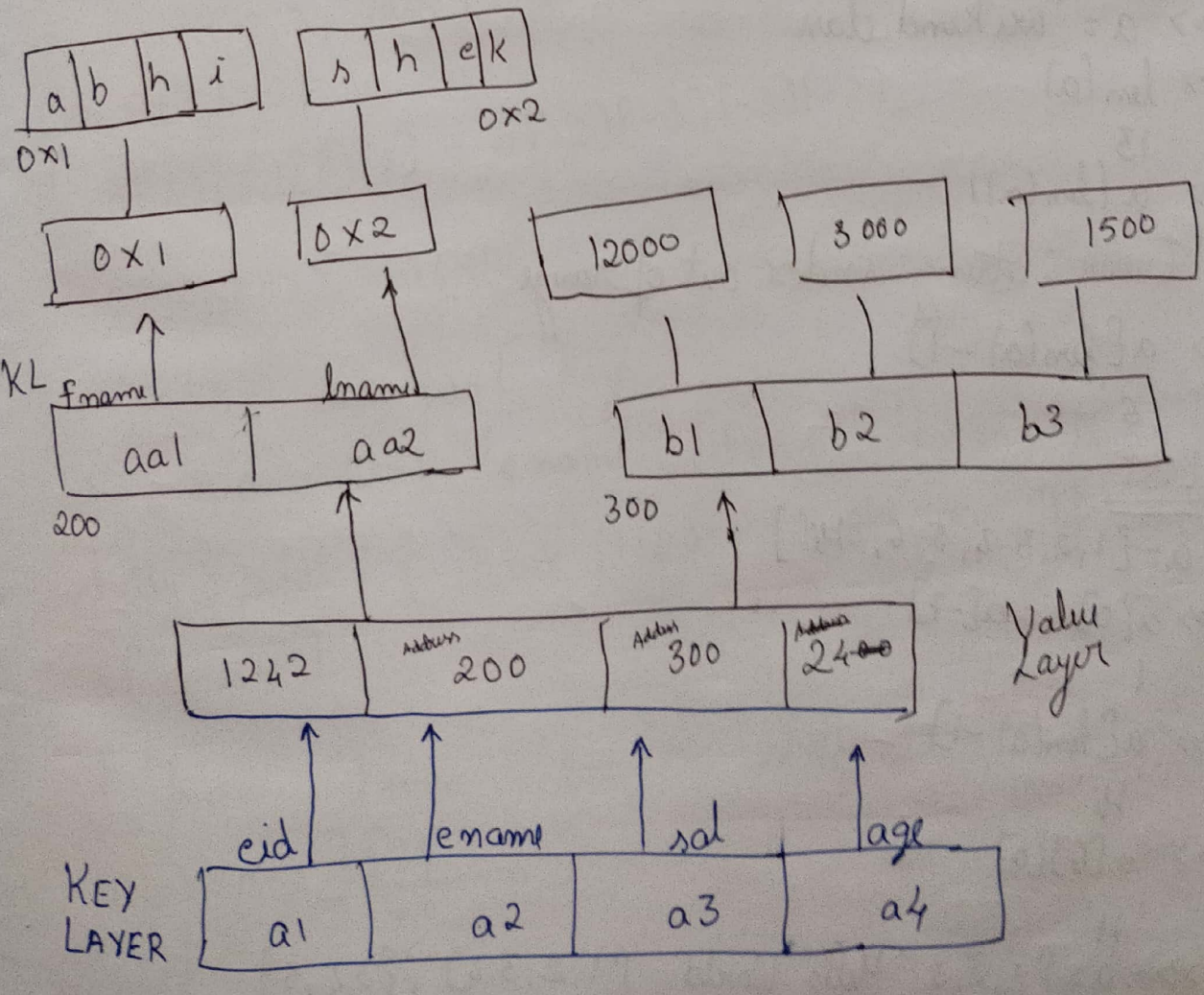


If the dictionary key has a value in the form of key & value pair then it has to be stored in the form of dictionary as shown in the above example

```
>>> a = {'eid' : '5204172', 'ename' : {'fname' : fnm,
                                       'lnm' : lnm}}
>>> a
{'eid' : '52041721', 'ename' : {'fname' : 'tarfnm', 'lname' :
                                'shetty'}}
```



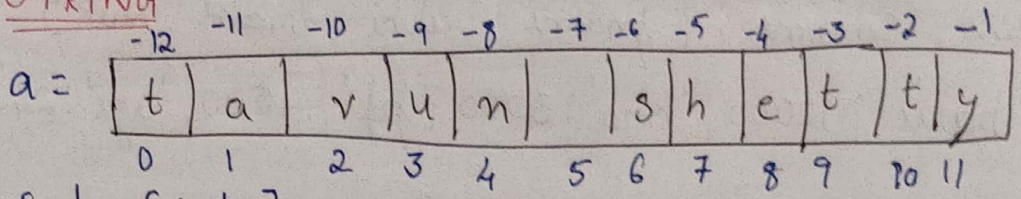
```
>>> a={'eid':1242,'ename':{'fname':'tarun','lname':
'shuk'},'sal':{'basic':12000,'hra':300,'pf':1500},
'age':28}
>>> a
{'eid':1242,'ename':{'fname':'abhi','lname':'shuk'}
'sal':{'basic':12000,'hra':3000,'pf':1500,'age':24}
```



SLICING

Identification of individual data items in the group data items is knowing as slicing.

STRING



Syntax [index]

↳ Sub Address

```
a[2] = a[-10] = r
a[5] = a[-7] = ' '
a[10] = a[-2] = t
a[11] = a[-1] = y
a[len(a)-1] = y
>> a = 'weekend class'
>> len(a)
   13
>> a(len(a))
   Error : String index out of range
>> a[len(a)-1]
   's'
```

LIST

```
a = [1,2,3,4,5,6,'Hi']
>> a[0] = a[-7]
   1
>> a[len(a)-1]
   'Hi'
>> a[6][0]
   'H'
>> a = [1,2,3 'Hello World', [1,2,3,4], {3,4,5}
   a[-3][-6] = ' '
   a[-3][5] = ' '
```

a[3][6] = '
a[4][2] = 3
a[4][-2] = 3
a[-2][2] = 3
a[-2][-2] = 3
a[-2][-2] = 3

1 2 3 Hello World [1, 2, 3, 4], [3, 4, 5]
0 1 2 3 4

TUPLE

a = (1, 2, 3, 'tarun shetty', 'low', [1, 2, 3, 4])
a[3][2] = r
a[4][0] = l
a[5][2] = 3
a[-1][2] = 3

>> a = (1, 2, 3, 'hello world', [1, 2, 3 [1, 2, 3, (3, 4), 5), 6, 7), 5, 6]
, [3, 4, 5])

>> a[4][3][3][1] = a[-2][-3][-3][-2]

DICTIONARY SLICING

SYNTAX: data var['key']

a = {'eid' = 1234, 'ename': {'fname': 'tarun', 'lname':
'shetty'}, 'sal': {'basic': 12000, 'hra': 2000, 'pf': 2000}}

a['ename']

{'fname': 'tarun', 'lname': 'shetty'}

a['ename']['fname']

'shetty'

a['ename']['fname'][2]

'e'

a={'eid':123, 'sal':{356, 1233, 67, 13255}

>>> a['sal'][1]

Error 'set' object doesn't support indexing.

>> a={'eid':123, 'sal'={1233, 132, 3456, 67}}

a['sal'][1]

132

>> a={'eid':123, 'sal':{1233, '132', 3456, 67}}

a['sal'][1][1]

'3'

Identification of sequence of data items in String, list & tuple.

Syntax:

strvar[startindex : end index]

or

strvar[start index : end index + 1

```
>>> a = 'Weekend Class'
    0 1 2 3 4 5 6 7 8 9 10 11 12
   13 12 11 10 9 8 7 6 5 4 3 2 1

a[4:6]
EN
>>> a[11:length]
SS
>> a[11:]
SS
>> a[5:10+1]
'ND CLA'
>> a[3,7+1]
'KEND'
>> a[3:8+1]
'KEND C'
```

```
a[3:-4]
'KEND C'
a[-10:-4]
>> a[-1:-(len(a))+1]
   → Nothing will come
>> a[-1:-(len(a))+1:-1]
"SSALC DNEKEEW"

>> a = 'Weekend Class'
>> a[-2:-1+1]
>> a[-2:len(a)]
'ss'
```

LIST

a = [5, 10, 15, 20]

a[0:3] = a[0 : -3+1] = [5, 10]

a[1:] = [10, 15, 20]

Tuple

a = (5, 10, 15, 20, 25)

a = (0, 4) = a(0, -1+1) = (5, 10, 15, 20, 25

a(-4:) = a(0 : 3+1)

Identification of sequence of data item in list string and tuple if the start index is 0 or -len

STRING

Syntax:

strvar[: endindex + 1]

a =

0	1	2	3	4	5	6	7	8	9	10
H	E	L	L	O		W	O	R	L	D
-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

a[0:4+1] = Hello

a[:4+1] = Hello

```
>> a[0:4+1] = Hello
>> a[-(len(a)):-7+1]
   'Hello'
>> a[:-7+1]
   'Hello'
>> a[:len(a)]
   'Hello World'
```

LIST

```
a=[1,2,3,'def',(1,2),[2,3,4,5]]
>>a
 [1,2,3,'def',(1,2),[2,3,4,5]]
>> a[:4+1]
 [1,2,3,'def',(1,2),[2,3,4,5]]
>> a[:-2+1]
 [1,2,3,'def',(1,2)]
```

TUPLE

```
a=(1,2,3,'def',(1,2),[2,3,4,5])
a[:4+1]
 [1,2,3,'def',(1,2),[2,3,4,5])
>> a[:-2+1]
 [1,2,3,'def',(1,2)]
>> a[0:1] = a[:1] = a[-6:-5+1]
 [1,2]
>> a[3][:1+1]
  da
>> a=(1,2,3,'daf',(1,2,[40,20,10]),[2,3,45])
>> a[4][2][:2+1]
 [40,20]
>> a[-3][:-2+1]
 'da'
>> a = 'Hello World'
```

Identification of sequence of data item in string, list and tuple if end index +1 = length

```
>> a[6:]
   World
```

LIST

```
a=[1,2,3,'daf',(1,2),[2,3,4,5]]

a
[1,2,3,'daf',(1,2),[2,3,4,5]]
>> a[2:]
   3,'daf',(1,2),[2,3,4,5])
>> a[:-2+1]
   [1,2,3,'daf',(1,2)]
```

Tuple

```
a=[1,2,3,'daf',(1,2),[2,3,4,5])
>>a[:4+1]
   (1,2,3,'daf',(1,2),[2,3,4,5])
```

[Repeat of previous page]

```
>> a[:-2+1]
   [1,2,3,'daf',(1,2))
>> a[0:1] = a[:1] = a[-6:-5+1]
   [1,2]
>> a[3][:1+1]
   da
```

Note: If both the start index and end index is skipped [a[:] by default it return all the data items of string list and tuple. ie a[:]

```
>> a=[1,2,3,4]
   a[:]
   [1,2,3,4]
```

Update Value

String

Syntax :-

strvar [start index : endindex ± 1 : update value]

If update value is +ve → End index + 1

If update value is -ve → End Index - 1

```
>> a = 'Hello World'
```

HELLO WORLD

```
a[0 : len(a) : 1] = Hello World
a[0 : len(a) : 2] = HloWrd
a[0 : len(a) : 4] = Hor
a[0 : len(a) : 6] = Hl
a[0 : len(a) : 8] = Hr
```

Identification of data items in string list tuple using user defined update value.

```
>> a[0 : len(a) : 2] = HloWrd
>> a[-11 : len(a) : 8] = Hr
```

Odd position

```
>> a[1 : len(a) : 2]
   el_ol
   a[1 : len(a) : 4] = e_l
```

To print the element of group data items the updat value must be negative.

```
a = 'Hello World'
>> a[10 : 1-1 : -1]
'dlrow olle'
>> a[10 : -11-1 : -1]
dlrow olleh
```

```
>> a[-1:-11-1:-1]
'dlrow olleh'
>> a[::]
'hello world'
>> a[::1]
hello world
>> a[5::2]
_ol
>> a[:-6-1:-2]
drw
```

```
>> a[10:-11-1:-2]
drwoleh
>> a[::-1]
'dlrow olleh
>> a[::-2]
drwolh
>> a[5::-2]
_le
```

+ Operator work for both addition & concatenation

```
>> a[-1:-5-1:-1]
dlrow
>> a[0:5+1]
hello_
>> a[0:5+1] + a[-1:-5-1:-1]
'hello dlrow'
>> a[10] + a[1:5+1] + a[6:9+1] + a[0]
dello worlh
>> a[10] + a[1:9+1] : a[0]
```

o/p : wollo hedlr

```
a[6:7+1] + a[2:5+1] + a[0:1+1] : a[-1:-3-1:-1]
```

o/p: olleh dorlw

```
a[-7:-11-1:-1] + a[5:len(a):5] + a[7:9+1] + a[6]
```

OPERATORS

Operators are special symbols which performs operation on the operands.

Operands are the data or the variables on which the operation has to be performed.

Operators are classified into different categories

1. Arithmatic Operators
2. Logical Operators
3. Relational Operators
4. Bitwise Operators
5. Assignment Operators
6. Membership Operators
7. Identity Operators

Arithmatic Operators

	Symbol	Operation
1)	+	Addition / Concatination
2)	-	Substraction
3)	*	Multiplication / Replication
4)	/	True Division
5)	%	Modulus
6)	**	Power
7)	//	Floor Division

ADDITION

```
>> a=2
>> b=3
>> a+b
   5
>> 3+5
   8
>> 3+'hi'
Traceback (most recent call last);
File <pyshell5>, line1 in <module>
```

* To perform concatenation both operand can be of same datatype

ie str-str, list-list, tuple-tuple

```
>> [1,2] +[2,3,4]
   [1,2,2,3,4]
>> (1,2) +(1,2,3) ⟶ (1,2,1,2,3)
```

Only list and tuple in multiple value element can be concatenated.

Set & Dictionary give an error.

SUBRACTION

Work with only an numberical value.

Subraction operation is used to perform substraction only between the numeric value and return signed or unsigned result.

```
>> 4-2
   2
>> 3-9
   -6
>> 4-2.5
   1.5
```

MULTIPLICATION (*) / REPLICATION

```
>> 2* [1,2,3]
   [1,2,3,1,2,3]
>>[2,7,9] *2
   [2,7,9,2,7,9]
```

Same set of value repeating again & again is called replication.

```
>> 'hi' x4
   'hihihihi'
```

One of the value must be positive integer, we get error otherwise like

- Traceback (most recent call last)
- Unsupported operand type(s) for * 'int and dict'

TRUE DIVISION

In python if a integer value is divided by another integer value, the result will be always floating result

```
>> 2/2
   1.0
>> -5/2
   -2.5
```

```
>> -10/5
   -2.0
```

MODULUS

It is used to return reminder

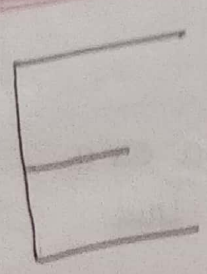
```
>> 10/2
   0
>> 10/3
   1
```

POWER

$a^b = a**b$

```
>> 2**3
8
>> 2**4
16
>> 2**9
512
>> 2**3.5
11.313
```

FLOOR DIVISION



Ciel	4	-3
Decimal	3.5	-3.5
Floor	3	-4

A decimal value which lies between two set of integers. Floor division will consider the lower end value.

LOGICAL OPERATORS

Logical operators result will be true or false

AND

OP1	OP2	OP1 AND OP2
0	0	0
0	1	0
1	0	0
1	1	1

```
>> a = True
>> b = False
>> b and b
False
>> a and b
False
>> b and a
False
>> a and a
True
>> 1 and 1
True (1) ✓
>> 1 and 0
False (0) ✓
>> 0 and 1
0
>> 0 and 0
0
>> 1 and 2
2
>> 2 and 1
1
>> 2 and 3.5
3.5
```

And logical operator is used to check whether given both the operator are true.

OR (Logical Operator)

OP1	OP2	OP1 or OP2
0	0	0
0	1	1
1	0	1
1	1	1

>> a = True
b = False

>> b or b
False

>> a or b
True

>> b or a
True

>> a or a
True

>> 0 or 0
0

>> 0 or 1
1

>> 1 or 0
1

>> 1 or 1
1

>> 1 or 2
1

>> 2.5 or 1
2.5

>> 0 or 3.5
3.5

Or logical operator is used to check given either of the condition is true.

NOT

Not logical operation is used to modify the result from true to false and vice versa.

NAND

Nand is negation of and

Not (op1 and op2)

op1	op2	not(op1 & op2)
0	0	1
0	1	1
1	0	1
1	1	0

```
>> a = True
>> b = False
>> not (a and b)
   True
```

NOR

Nor is negation of or

Not (op1 or op2)

op1	op2	Not(op1 or op2)
0	0	1
0	1	0
1	0	0
1	1	0

BITWISE OPERATOR

Symbol	Operation
&	Bitwise And
|	Bitwise OR
^	Bitwise X-OR
>>	Bitwise Right Shift

```
>> 4 &6
   4
```

```
 0100
 0110
------
 0100
```

BITWISE AND (&)

```
>> 8&12
   8
```

```
>>> 14 &15
    14
```

```
 1110
 1111
------
 1110
```

```
>>> 2 &4
    0
```

```
 0010
 0100
```

```
>>> -2 &13
    12
>>> -8&6
    0
>> -2 &13
```

Step 1) -2 = Complement (2's)

```
 0010
 1101
    1
------
 1110
```

Add both

```
 1110
 1101
------
 1100   [12]
```

>>> -15 & -8 → -16

>>> -2 & -4 → -4

>>> -2&-4

-2 → 0010
1101
1
1110

-4 → 0100
1011
1
1100

-2&-4
1110
1100
1100
0011
0100

>> -17 & 13
13
>> -2&13
12
>> -9 &11
3
>> -15&8
0

-15&-8

1111
0000
1
0001

1000
0111
1
1000

-15 & -8
0001
1000
0000
1111
10000 = -16

BITWISE OR

>>> 8|12 → 12
1000
1100
1100

>>> -8|12 → -4
-8
1000
0111 +1
1000

1000
1100
1100

1100
0011
1
0100

>>-18 & -23 → -24

```
10010   [-18]
01101
    1
-------
01110

01110
01001
-------
01000
00111   1's comp
    1
-------
11000
```

```
10111
01000
    1
-------
01001
```

>>> 23 >
5

BIT

N

2

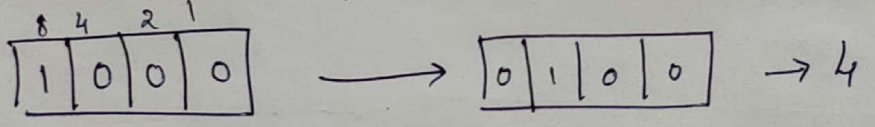
1) E

2)

Bitwise Right Shift

Number to be shifted >> Number of times the number to be shifted.

8 >> 1 → 4

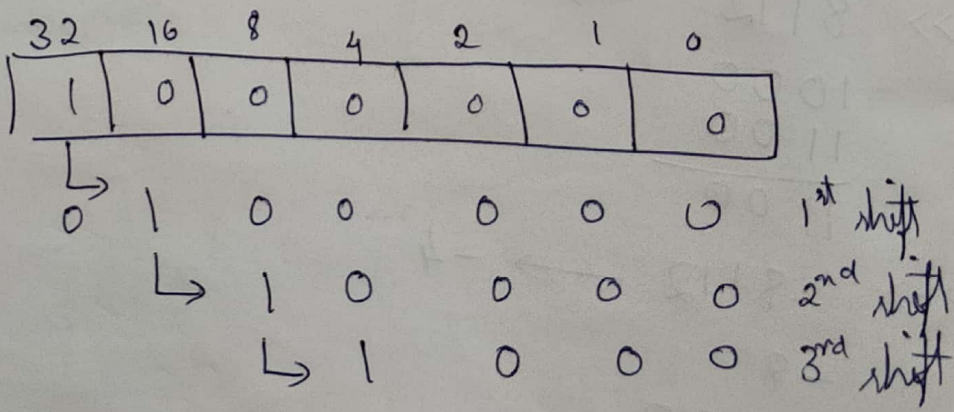


→ 4

Notes: Always right shift means floor divided by 2

In right shift operation for every shift the given number will be floor divided by 2

Eg: 32 >>>3



Ans: 4

29 >> 3 → 3

32 >> 2

Ans: 8

>> 23 >> 2
5

1	0	1	1	1

↳ 1 0 1

BITWISE LEFT SHIFT OPERATOR

$2^n \rightarrow$ n stands for number of shift

Number to be shifted << Number of times the number to be shifted

Left shift Result = given Number $\times 2^n$

where n = Number of shift

4 << 2

→ 16

0	1	0	0

4

1 0 0 0

1 0 0 0 0

BITWISE NOT

Result = ~(n+1)

= −(n+1)

1) Eg : ~−4 → −(−4+1)

→ −(−3)

Ans → 3

2) ~12 → −(+12+1)

−(+13)

−13

RELATIONAL OPERATOR (>, <, >=, <=, !=, ==)

Relational Operator result will be of boolean result

Syntax: Operand1 Relational Operator Operand2

We use relational operator when we are going to use for condition

: Eg: 2>3 -2>3 -2>-3 2==2
False False True True

>> 10>=5 or (10>5) or (10==5)
True True

4<=3 3>=4 2==5
False False False

a=2 >>a >> a= a>=2
a>=4 2 >> a
False True

ASSIGNMENT OPERATOR

=, +=, -=, *=, %=, /=, //=, **=, >>=, <<=

Assignment operator are used to perform the operation and stores or assign the value to a variable.

```
>>> a=10
>>> b=5
>>> a+=b
>>> a
15
>> a-=b
>> a
10
>>> a*=b
>> a
50
```

```
>> a%b
>> a
0
```

```
>> a=10
>> a/=b
>> a
2.0
```

```
>> a//=b
>> a
0.0
```

```
>> a**=b
>> a
27
```

a<<2

```
>> a>>=2
a
6
```

MEMBERSHIP OPERATOR

It is used to check whether the value or variable is present in group or not IN, NOT IN

IN → Variable / Value in group

IN membership operator is used to check the value or variable present in the group

If the value is present it returns boolean result TRUE otherwise FALSE

NOTE: For dictionary it will check for the keys not value.

```
>>> 1 in [1,2,3,4]
    True
>>> 5 in [1,2,3,4]
    False
>>> 'ddf' in {'a':'adf', 'b':'XYZ'}
    False
>>> 'a' in (1,2,'hi',a)
    True
```

NOT IN: It is used to check the value or variable is not present in the group

Not in → variable / value not in group

If the value is not present it returns TRUE otherwise FALSE.

Eg:

```
>>> 5 not in [1,2,3,4]
    True
>>> 'a' not in {1,2,'hi','a'}
    False
>>> 'a' not in {'a':'abc','b':'XYZ'}
    False
```

IDENTITY OPERATORS

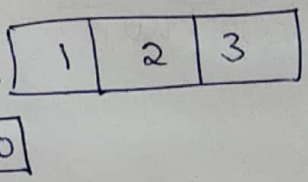
It is used to check whether both the variable pointing to same memory or not.

If it is pointing to same memory the result will be true, if it is pointing to different memory the result will be false

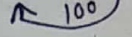
1) is → is operator is used to check whether both the operator are pointing to same memory.

If it is pointing to same memory the result will be true otherwise FALSE.

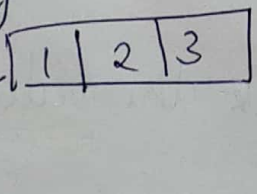
a = [1, 2, 3]



b = a



c = a.copy()




```
>>> a=[1,2,3]
>>> id(a)
    100
>>> b=a
>>> id(a)
    100
>>> id(b)
    100
>>> c=a.copy()
>>> id(c)
    200
>>> c
    [1,2,3]
```

IS NOT : Is not operator is used to check whether both the variables are not pointing to same memory If it is not pointing to same memory it returns the boolean result TRUE otherwise FALSE

Same as above example continue

```
>>> id(a)          >> id(b)          >> id(c)
    100                100               200
>> a isnot b       >> a is not c
    False              True
```

BITWISE XOR OPERATOR

In bitwise XOR operator if both the bits are zeros or one it returns zero.

ie >>> 10^12
6

Eg
```
  1010
  1100
 ------
  0110  → 6
```

```
>>> 12^16
    28
```

```
  01100
  10000
 ------   → 28
  11100
```

```
>>> -12^25
    19
```

```
  1100
  0011
     1
 ------
   100
```

```
  00100
  11001
 ------
  11101
```

```
>>> -12^-18  → 26
      1100
```

When both the number are -ve then no need of two's compliment for the result. This is for only XOR operation If both the operands are negative then 2's compliment of the result is not required (For OR, AND & XOR) If either of the operand is -ve then 2's compliment of the result is necessary to get the actual result

OPERATIONS ON GROUP DATA ITEMS

1) Operations on string

```
>>> a = 'Welcome'
```

Count

Count function is used to find occurance of the substring in a string between the boundaries Here the boundaries are not mandatory. If the count function is given only with substring it returns the count from the entire string

SYNTAX : string variable/value . count (substring)
str variable/value . count (substring , start point, End point

* If the count function is given with substring along with the start point and end point then it returns the count between the boundaries
Excluding the count of the substring in the end point

Syntax

String variable/value . count (substring, start point)
↓
By default end point is length of string.

String variable/value . count (substring)
↓
By default start point and end point is 1 & length of string respectively.

```
>> a='WEEKEND BATCH'
>> a.count('E')
   3
>> a.count('e')
   0
>> a.count('E',0,3)
   2
>> a.count('E',4)
   2
>>'WEEKEND PATIL'.count('E',0,4)
   2
>>'WEEKEND TARUN'.count('E',0,5)
   3
>> 'eriesson'.count('e',0,3)
   1
>> 'eriesson.count('e',0)
   1
```

STARTS WITH

Syntax:

str var/value . startswith(given)

```
a='hello world'
→ a.startswith('h')
  True
→ a.startswith('H')
  False
→ a.startswith('w')
  False
→ a.startswith('w')
  True
```

```
→ a.startswith('w',6,len(a))
  True
→ a='hello World
  len(a)
  11
```

ENDS WITH

Ends with function is used to check whether string ends with given substring or not. It is case sensitive
- If the string is ending with given substring it returns value True, false otherwise.

```
>> 'audtime'.endswith('E')
   False
>> 'audtime'.endswith('e')
   True
>> 'audetime'.endswith('e',0,3)
   False
>> 'audetime'.endswith('e',0,4)
   True
>> 'taran'.endswith('n',3,6)
   True
```

FIND FUNCTION

It is used to find the position of the substring within the boundary. If boundary are not provided it always return the first occurence position.

NOTE: If a substring is not present it returns result as -1.

```
>> 'audetime'.find('e')
   3
>> 'audetime'.find('e',4)
   7
```

```
>> 'audetime'.find('e',4)
   7
>> 'audetime'.find('e',0,3)
   -1
>> 'audetime'.find('mi',3,8)
   6
```

INDEX

Index function is similar to find but the difference is, if the element is not found it will throw an error

value.Index(Substring) or
value.Index(Substring, start index, end index)

IS ALPHA

>> Isalpha function is used to check whether a string is having only alphabets or not. Return TRUE if all the char of string contains only alphabets and return false otherwise.

```
>> 'audetime'.isalpha()
   True
>> 'audetime1'.isalpha()
   False
```

ISDIGIT

```
>> '123'.isdigit()  → True
```

Is digit function is used to check whether the entered string has only number of digit.

```
>> '123a'.isdigit()
   False
>> '123.5'.isdigit()
   False
>> '-1'.isdigit()
   False
```

IsAlNum()

IsAlNum function is used to check whether the entered string is collection of characters or only the collection of numbers or collection of both character and number.

It returns boolean result true if the string is having only character number (Positive) or collection of both character & numbers

```
>> 'auditime123'.isalnum()
   True
>> '123'.isalnum()
   True
>> '123asd-1'.isalnum()
   False
>> '123asr@'.isalnum()
   False
```

LOWER

It is used to convert the data into lower case if any data is in upper case.

It can take alpha numeric data as input. It converts only the character and returns the result.

```
>> '123HAT@'.lower()
>> '123hat@'
>> 'PYTHON'.lower()
   python
>> '123@@##'.lower()
   '123@@##
```

ISLOWER()

```
>>> '-123@eee'.islower()
    True
>> '-12Ae'
    False
```

Is lower function is used to check all the data (character) in the string is in lower case or not. and returns the boolean result TRUE if all the characters are in lower case

UPPER

It converts the data into upper case if any input data is in lower case

```
>> 'tarun0304'.Upper
   TARUN0304.Upper
```

IS UPPER

It checks whether the entire string character is in uppercase or not and returns boolean. result TRUE or FALSE

```
>> '0304A'.isupper()
   True
>> '0304a'.isupper()
   False
```

Title

Title function is used to convert the starting character of all the words into uppercase

```
>> 'java python'.title()
   Java Python
>> 'java @python'.title()
   Java @Python
```

Is Title

Is title function checks whether the starting character of all the words in the string is an uppercase or not.

```
>> 'Java @python'.istitle()
   False
>> 'Java @Python'.istitle()
   True
>> 'JAVA @PYTHON'.istitle()
   False
```

CAPITALIZE

It is used to display the first character of a string in uppercase

The operation which provides the output value will not change the original value

```
>> 'Hello_World'.capitalize()
   'Helloworld
>> 'helloworld'.capitalize()
   'Helloworld
>> a = 'helloworld'
   a.capitalize()
   Helloworld
>> a
   'helloworld'
```

REPLACE

Replace function is used to replace character or substring to a older substring in a string with a new string

It is also possible to mention the replacement count ie If the older substring is repeated more than once user can specify how many times the older string has to be replaced with the string by providing count as third argument in the replace function.

Syntax

```
str var/value.replace(older substring, new substring)
```

or

```
str var/value.replace(older substring, new substring, count)
```

```
>> 'hello Tarun'.replace('hello', 'hey')
   'hey Tarun'
>> 'hello hello Tarun'.replace('hello', 'hey')
   'hey hey Tarun
```

```
>> 'hello hello hello Tarun'.replace('hello', 'hey', 1)
   hey hello hello Tarun
```

Isidentifier

Isidentifier function is used to check the content present inside the string can be a name given to any memory location or not

```
>> 'abc'.isidentifier()
   True
>> '1'.isidentifier()
   False
>> '_*'.isidentifier()
   False
>> '_'.isidentifier()
   True
>> 'if'.isidentifier()
   True
```

True

Split

Split method is used to break to string into smaller peaces, whenever given substring occur in the main string.

Syntax
Str Var/Value. Split (substring / split char)

```
>>> 'object has no attribute'.split('t')
    ['objec', ' has no a', 'ribu', 'e']
>> 'hello'.split('e')
    ['h', 'llo']
>> 'hello'.split('l')
    ['he', '', 'o']
>> 'hello world'.split('l')
    ['he', '', 'o wor', 'd']
>> 'gspiders is a test engineers weekend adda'.
['gsid', 'rs is a t', 'st', 'ngin', 'rs w', '
    'nd adda']
>> 'java java java avaj'.split('j')
    ['', 'ava', 'ava', 'ava ava', '']
```

NOTE :- If count is not provided number of split is equal to number of split character + 1.
If number of split is provided number of split is equal to count + 1.

```
>> 'hello'.split('l',1)
   ['he', 'lo']
>> 'hello world'.split('l',2)
   ['he', '', 'o world']
```

JOIN

Join function is used to join collection of string as a single string.

Syntax:

```
'glue String'.join(arg)
                ↳ Iterable type
a=['', 'ava', 'ava' 'ava ava', '']
→ j.join(a)
>> a={
   [javajava java avaj]
```

OPERATION ON LIST

Append - Append function is used to add the element at the end of the list.

Syntax: List Var/Value.append(data)

```
>>> a=[1,2,3,4]
    a.append('hi')
    a
    [1,2,3,4,'hi']
>>> a.append(['hi','hey'])
    a
    [1,2,3,4,'hi',['hi','hey']]
>> a.append((2,3))
    a
    [1,2,3,4,'hi',['hi','hey'],(2,3)]
```

NOTE: Here the original value will be modified in the list.

INSERT

Insert function is used to insert or add an element at the position specified by the user. When the input position is mentioned then the element present in that position will be shift towards right by one bit.

Syntax:

List Var/value Insert (Position, data)

```
a=[1,2,3,4]
a.insert(2,2.5)
>> a
   a=[1,2,2.5,3,4]
-> a.insert(len(a),'xyz')
   a
   a=[1,2,2.5,3,4,xyz]
```

POP

Pop function is used to remove an element from a list. If a input argument(index) is not provided then it always removes the last element in list. If the index is provided then it removes a element which is present in the given index.

Syntax

List var.Pop()

or

List var.Pop(index)

```
>>> a
[1,2,3,4,'hi']
>> a.pop()
→ a
[1,2,3,4]
>> a.pop(2)
[1,2,4]
```

REMOVE

~~List var~~ = ~~Syntax~~

Remove function is used to remove the user specified element from the list providing element as an input argument to the remove function.

If the element is not present then it throws an error.

Syntax: List Var.Remove(value)

>> a = [1, 2, 3, 4, ['hi', 'hey'], 5, 5]

a.remove(5)

a = [1, 2, 3, 4 ['hi', 'hey'], 5]

a.remove(2)

a = [1, 3, 4, ['hi', 'hey'], 5]

CLEAR

Clear function is used to remove all the ele present in the list.

Syntax

List var.clear()

→ a = [1, 2, 3, 4]

a.clear()

>> a

[]

If want to del a variable al we need to ha del function

EXTEND

Extend function is used to add list of element individually at the end of the list.

Syntax:

List Var.Extend (arg)
↳ Iterable

```
>> a=[1,2,3,4]
   a.extend([5,6,7])
o/p a=[1,2,3,4,5,6,7]
   a.extend('Hello')
   [1,2,3,4,5,6,7,'H','e','l','l','o']
```

REVERSE

Reverse function is used to reverse the list values. Here the actual values will be modified.

```
>> a=[1,2,3,4,'H','I']
   a.reverse()
o/p - a=['I','H',4,3,2,1]
```

Syntax

List Var.Reverse()

COUNT

Count function is used to count the number of occurance of particular value or element in the list.

```
>> a='Hello'
   a.count('l')
   2
>> a.count('z')
   0
```

Syntax

List Var.Count('Value')

INDEX

Syntax

List Var. Index (Value)

or

ListVar. Index (value , startpoint)

or

List Var. Index (value , startpoint , end point)

Index is a function used to find the index of an element between the boundaries.

```
>> a = ['h','i','h','e','l','l','o']
>> a.index('l')
   4
>> a.index('l',0,4)
```

Error:

< Value Error > l is not in list

```
>> a.index('l',0,5)
   4
>> a.index('l',6,len(a))
   5
```

SORT

Sort function is used to sort or arrange the homogenious elements in the ascending order.

Syntax

List Var. Sort()

```
>> b=[5,2,7,3,1]
   b.sort()
   [1,2,3,5,7]
>> c=['t','a','r','u','n']
   c.sort()
```

o/p c = ['a', 'n', 'r', 't', 'u']

COPY

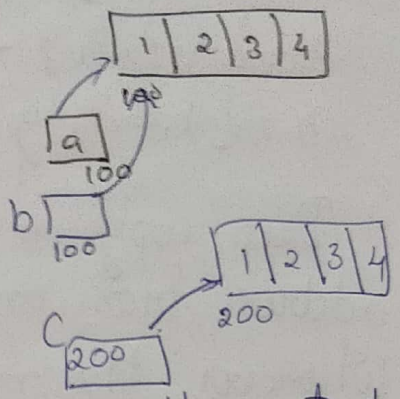
Syntax:

var = list var.copy()

a = [1,2,3,4]

b = a → Deep copy

c = a.copy() → Shallow copy

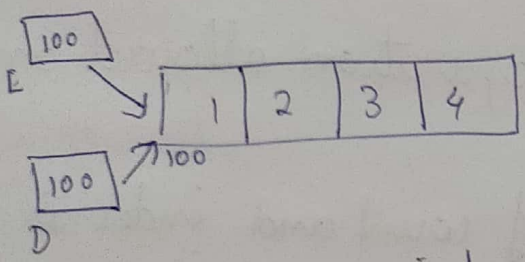


Copy is a function is used to copy the content present in the list into another list memory. It is also called as shallow copy.

DEEP COPY

D = [1,2,3,4] ← Step 1

E = D ← Step 2



When a variable is copied to another variable at step 2 address is copied

Step 3 : D.append(A)

o/p D = [1,2,3,4,a]

Thus variable is modified at step 3 also modifies the new variable.

E = [1,2,3,4,a]

```
>>a=[1,2,3]
  b=a
>>a.append('g')
  a=[1,2,3,'y']
  b=[1,2,3,'y']
```

Deep copy is the phynomenon of copying the address from one variable to another variable. Whenever the modification is done to any of the variable the changes get reflected also in another variable.

OPERATIONS ON TUPLE

Insertion and deletion can't be performed on tuple. Thus the operation that can be performed on tuple are the operation that doesn't affect the tuple variable

Thus the only operation allowed on tuple are count, index

NOTE : Operation of count and index is similar to operation of count and index in list.

COUNT

Syntax

```
TupleVar.count('Value')
a=(1,2,3,4,5,'a')
a.count(2)
1
```

INDEX

Syntax

```
TupleVar.Index('Value')
a.Index(4)
3
```

OPERATIONS ON SET

ADD

Syntax:

SetVar.Add(value)

Add function is used to add or insert an element into the set.

```
>> a = {1,2,3,4,5,12}
>> a.add(6)
   a = {1,2,6,3,4,5,12}
>> a.add('Hi')
   a = {1,2,6,3,4,5,12,'Hi'}
```

POP

Syntax:

SetVar.Pop() → Pop the first value

~~or~~

Pop function is used to remove the first element present in the set.

```
>> a = {1,2,3}
   a.pop()
   1
>> a.pop()
   2
```

REMOVE

Remove function is used to remove the user specified element in the set.

Syntax:

SetVar.Remove(Value)

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

a.remove

op a = {1,2,3}

CLEAR

Clear function is used to clear all the elements inside a set and return an empty set

Syntax:

```
a={1,2,3}
a.clear()
Set()
```

Here the memory of the set is not deleted.

DEL

Del function is used to delete the memory containing the data as well as the memory where it is storing

Syntax

```
del setvar
   or
del (setvar)
```

```
>> a={1,2,3}
   b=a
   del b
>> a
   {1,2,3}
```

Note: Del function can delete the memory of any of the group data items.

COPY

Copy function is used to reperform shallow copy operation.

Syntax:

var = Setvar.copy()

```
>> m = {1,2,3}
   n = m.copy
   n = {1,2,3}
```

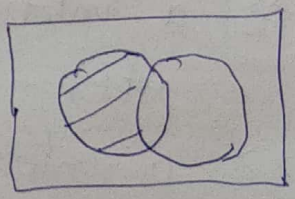
DIFFERENCE

Difference set function is used to identify the element of the set one which is not present in set 2.
It work similar to - set operator.

Syntax

Setvar1.difference(Setvar2)

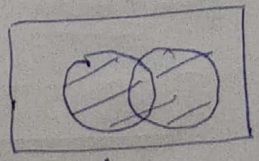
```
>> a = {1,2,3}
   b = {3,4,5}
>> a.difference(b)
   {1,2}
>> b.difference(a)
   {4,5}
>> c = {4,5,6}
   a.difference(c)
   {1,2,3}
```



UNION

Syntax:

Set1.union(set2)



Union function is used to return unique elements from both the sets.

>> a = {1,2,3}
b = {3,4,5}
c = {4,5,6}
d = {1,2,3}

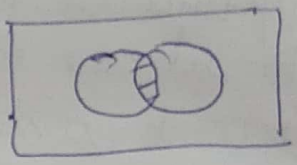
>> a.union(b)
{1,2,3,4,5}

a.union(c)
{1,2,3,4,5,6}

>> a.union(d)
{1,2,3}

INTERSECTION

Syntax
Set1.intersection(Set2)



>> a.intersection(b)
{3}

>> a.intersection(c)
set()

>> a.intersection(d)
{1,2,3}

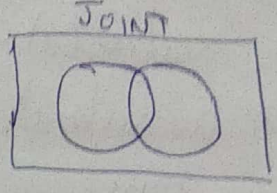
It is used to return the common elements from both the set.

If there is no common elements then the function returns empty set.

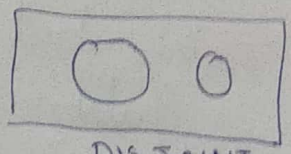
ISDISJOINT

It is used to check whether both the set are having any common elements or not

If there is no common element it returns boolean result true or false otherwise.



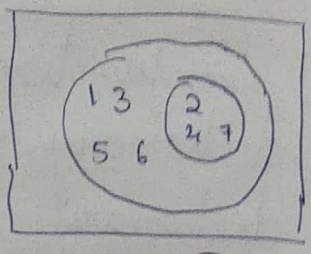
>> a.isdisjoint(c)
True
>> a.isdisjoint(b)
False



ISSUPERSET

Issuperset function checks whether set1 is super set of set2.

> syntax
Set1.Issuperset(set2)



>> a.Issuperset(d)
True
>> a.Issuperset(b)
False

Suppose the set d contains any additional element which is not present in set a then a is not called as superset of d

ISSUBSET

Syntax:
Set1.Issubset(set2)

a = {1,2,3,4,5,6,7}
b = {2,4,7} c = {1,2,3}

Is subset is used to check a setB is derived from setA or not

ie Used to check all element of set B is some of the element of set A. Then the set B is called as subset A.

```
>> b.subset(A)
    True
>> b.subset(c)
    False
```

OPERATIONS ON DICTIONARY

LEN

Len function is used to return the length of the dictionary. ie Number of unique keys present in the dictionary.

```
>> a = {'eid': 123, 'firstName': 'Tarun', 'salary': 29999}
   len(a)
   3
```

COPY

It is used to copy the content of one dictionary into another. ie It performs shallow copy.

```
>> b = a.copy()
   b
```

o/p : {'eid': 123, 'firstName': 'Tarun', 'salary': 2999

>>>

POP

Pop function is used to remove user specified data item in the dictionary.

```
>> a.pop('salary')
   29999
>> a.eid('eid', 123)
     123
>>  a
o/p  {'firstName': 'Tarun'}
```

POPITEM

It is used to remove key value pairs(items) which is present at the end of dictionary.

Syntax

dictvar.popitem()

```
a = {'eid': 123, 'ename': 'Tarun'}
>> a.popitem()
   {'ename', 'Tarun'}
```

KEYS

Keys functions is used to return the list of keys from the dictionary.

```
a = {'eid': 123, 'en': 'Tarun'}
dict a.keys()
dict.keys(['eid', 'en'])
```

Syntax

dictvar.keys()

Values

Syntax :

dictvar.values()

Values is a function used to return list of values from the dictionary.

```
a.values
→ dict_values (['123', 'tarun'])
```

Get

Syntax

```
dictvar.get(key)
```

Get function is used to get the dictionary key's value by providing the keys as an argument or by providing both key and values as an argument.

```
>> a.get('eid')
   123
>> a.get('eid', '123')
   123
```

SETDEFAULT

Set default function is used to create a key and value pair. But the value can be default value or the actual value.

Set default function is basically used whenever we need to store the null values to the dictionary keys.

If the key is already present it displaces its value else it adds the new key value pair to the dictionary

Set default function will not override the

dictionary keys.

```
>> a ={'eid':123}
   a.setdefault('sal')
>> a
>>a = {'eid':123,'sal':None}
>>> a.setdefault('panno','EEUPS')
      EEUPS
>>> a
    a={'eid':123,'sal':None,'panno':'EEUPS'}
```

CONTROL STATEMENTS

Control statements are used to change the flow of execution of a program.

Type
1) Decisional Statement
2) Looping or iterative statement.

Decisional Statement

1) If
2) if else
3) elif
4) Nested if

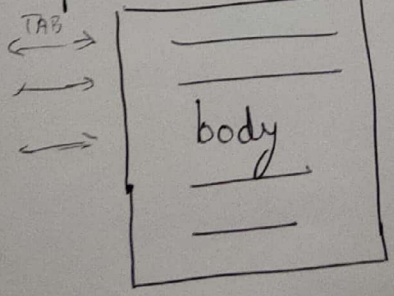
Looping Statement

1) For
2) While

If

Sytax

```
if Condition:
    body
```



Simple if statement is used whenever single condition has to be checked.

Ex:

```
time = 12 / 11.55        → Give one value
print ('application started')
print ('before it started')
if time == 12:
    print ('credit the daily datalimits')   } If is this much
print ('control out of if block
```

Output:

If time = 12
application started
before it started
Credited the daily
Control out of block

If else

If else is used whenever two conditions has to be checked.

Syntax:

```
if condition:
    =
    =
    body
    =
else:
    =
    =
    body
    =
    =
```

Ex

```
a = 10
b = 20
if a > b:
    print ('a is greater')
```

```
else:
    print('b is greater')
```

o/p b is greater.

Write a program to check whether the given number is even or odd

```
a = 10
b = a % 2
if (b==0):
    print('a is even')
else:
    print('a is odd')
```

o/p: a is even

Write a program to login to a application and display the authentication message.

```
authenticated_Password = True
authenticated_Userword = True
if (authentication_Userword and authentication_Password):
    print('Login is successful')
else
    print('Authentication error: Either username or
          password is wrong')
```

```
UN = 'tarun'
PW = 'Shetty'
if UN=='tarun and PW='Shetty':
    print('Login Successful')
else:
    print('Invalid syntax')
```

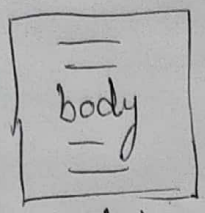
o/p: Login Successful.

Elif

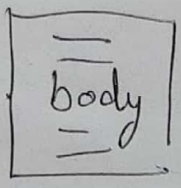
Whenever multiple conditions to be checked we use elif.

Syntax:

If condition:



Elif condition:



Else:


```
a = 10
b = 20
c = 30
If a > b and a > c:
    print('a is greatest')
elif b > a and b > c:
    print('b is greatest')
else:
    print('c is greatest')
```

a, b, c = 3, 2, 1

(b > c is enough)

Greater or equal

```
a = 10    b = 20
If (a > b):
    print('a is greater')
elif (a == b):
    print('a & b are equal')
```

if a > b is enough

```
else:
    print('b is greater')
```

Program to conselling

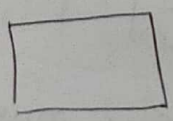
```
a ~~interest = 'EC'~~ intelligent = 100
if (intelligent > 90):
    branch == 'EC'
elif (intelligent > 70 and intelligent <= 90)
    branch == 'Comp'
elif (intelligent > 50 and intelligent <= 70)
    branch == 'Instrumental'
else
    brach == 'civil
print ('Joined' + branch + 'branch enjoy!')
```

Nested if

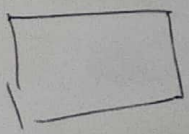
If side if is know as nested if whenever multiple conditions are written in the if statement, such statement are called nested if.

Syntax

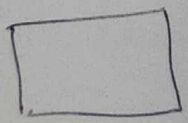
```
if Condition:
    if Condition
```



```
else
```



```
else
```



WAP for checking username password

```
UserName = 'tarun'
Password = 'Shetty'
if (UserName == 'tarun'):
    print('Tarun is valid username')
    if (Password == 'Shetty'):
        print('Login is successful')
    else:
        print('Password is incorrect')
else:
    print('Username is wrong')
```

Greatest of 3 numbers

```
a, b, c = 3, 2, 1
if (a > b and a > c):
    print('a is greatest')
else:
    if (b > c):
        print('b is greatest')
    else
        print('c is greatest')
```

Try this

Write a program to c greatest of 4 numbers

```
a = 10
b = 20
c = 30
d = 40
```

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

LOOPING STATEMENT

Iterative Statement, are the statement where the starting point and ending point are same, we call that type of statement as looping statement.

It is also called as iterative statements as it executes set of instruction again & again

Type
1) While
2) For Loop

While

It is basically used when the number of iteration are unknown ie It executes set of statements again & again until the given condition is satisfied.

Syntax:

```
While condition:
    Body
```

Write a program to write a program from 1 to 10

```
a = 1
while ( a < 11):
    print (a, end = ' ')
    a += 1
```

or a = a+1

Write a program to write a program of even number

```
n = 1
while n <= 10:
    if n % 2 == 0:
        print (n, end = ' ')
    n += 1
```

For

Whenever the number of iteration are known we use for loops. It is a statement which will be executed for each and every item present in the group.

Range

Range function is used to generate the sequence of values, but we cannot save the values until and unless we store the values.

Syntax :-

range (start point, end point, update value)

or

range (start point, end point) → Update value by default is 1

or

range (end point) → Start value by default is 0
Update value by default is 1

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

```
>> range(1,5)
   range(1,5
>> range(1,10,2)
   range(1,10)
>> list(1,10,2))
   [1,3,5,7,9]
```

Group : Group can be a string, list, tuple, set dictionary or range function.

For Loop (DICTIONARY)

a = {'a': 'abc', 'b': 'xyz', 'c': 'pqr'}

(i) for i in a:
    print(i)

o/p
a
b
c
d

Try concatenation of string

(ii) for i in a
    print(i, ':', a[i])

o/p
a: abc
b: xyz
c: pqz

All the above looping statements get terminated after the condition is false

STATEMENTS USED TO BREAK OR INTERRUPT THE FLOW OF EXECUTION

1. Break
2. Continue

BREAK

Break statement which is used to break the loop & make the control to come out of the loop.

Find the element in the loop

a = [1, 2, 3, 4 5]
element = 4
for i in a:
    if i == element
        print ('element found')
        break

(ii) a = [1, 2, 15, 20, 25]

element = 15

```
for i in a
    while (i == element):
        print ('Element is', i)
        print ('Element found as 15 so terminated')
        break
print ('Element is', i)
```

iii) Print till 14

```
a = 1
while a <= 20:
    if a == 15:
        break
    print (a)
    a += 1
```

O/P:

1
2
3
4
:
:
13
14

iv)

```
i = 1
while range (1, 10)
    if i == 5
        break
    print (i)
    i += 1
```

```
i = 1
a = range (1, 10)
while [a[i] == 5]
```

CONTINUE

Continue statement which is used to come out of the current execution and work normally for other execution

i) a = [1, 6, 5, 3, 4, 2]

element = 4

for i in a

```
        if i == element
            print ('element found')
            continue
    print (i)
```

(ii)
```
i = 1
while i <= 10
    if (a >= 5 and a <= 7)        if i in [5,6,7]
        continue                      i += 1
    print (i)
    i += 1
```

o/p : 1, 2, 3, 4 8, 9, 10

(iii)
```
for i in range (1,11):
    if i in [5,6,7]:
        continue
    print (i)
```

Print statement is used to print a variable value or text in the output console.

Syntax to print values in new line:

```
print (var)
```

Syntax to print value in same line:

```
print (var, end = ' ')
```

x x x x x

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

j = column

i = row

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

j = column

i = row

(1,1)	(1,2)	(1,3)	(1,4)	(1,5)
(2,1)	(2,2)	(2,3)	(2,3)	(2,3)
(3,1)	(3,2)	(3,3)	(3,4)	(3,5)
(4,1)	(4,2)	(4,3)	(4,4)	(4,5)
(5,1)	(5,2)	(5,3)	(5,4)	(5,5)

(i)

```
n = 5
for i in range (1, n+1)
    for j in range (1, n+1)
        print ('*', end = ' ')
    print ()
```

(ii)

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

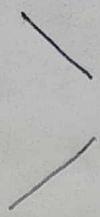
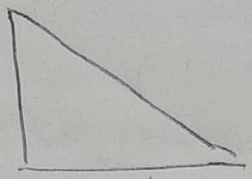
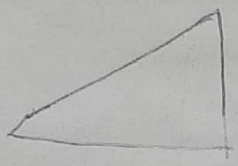
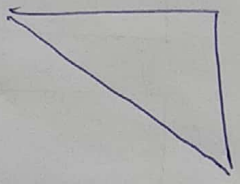
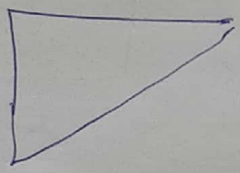
```
1 1 1 1 1
2 2 2 2 2
3 3 3 3 3
4 4 4 4 4
5 5 5 5 5
```

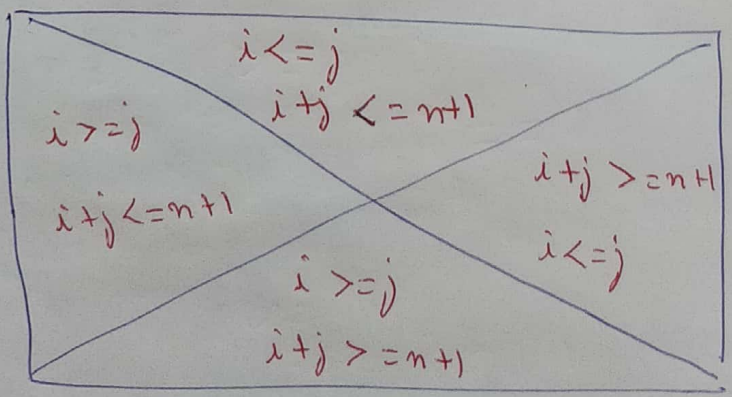
(iii)

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

```
* 1 1 1 1
1 * 1 1 1
1 1 * 1 1
1 1 1 * 1
1 1 1 1 *
```

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

	FORMATION	FORMULA
i:		$i == j$
		$i + j = n + 1$
i)		$i >= j$
		$i + j >= n + 1$
(		$i <= j$
(i		$i + j <= n + 1$



$ * * * *
$ * * *
$ * *
$ *
$

```
n=5
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()
```

```
|
| |
| | |
| | | |
```

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

(1, i+1)