Review of Python basics

CBSE Syllabus Based

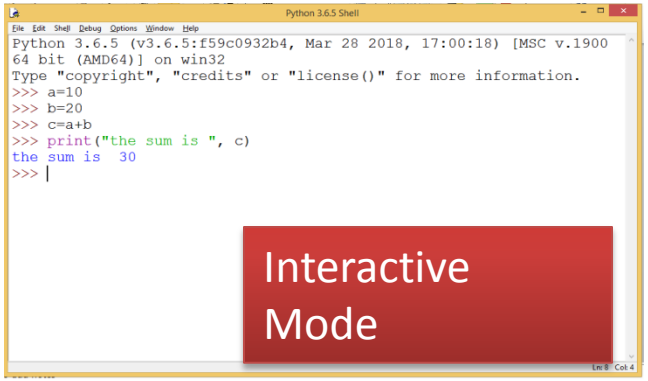
Class -12

CHAPTER -1

By-
Neha Tyagi
PGT CS
KV 5 Jaipur II Shift

Python (a computer language)

- In last class we have learned about Python. In this class we will learn Python with some new techniques.
- We know that Python is a powerful and high level language and it is an interpreted language.
- Python gives us two modes of working-
  - Interactive mode
  - Script mode



Python (a computer language)

- It is possible to develop various Apps with Python like–
  - GUI Apps
  - Web Apps
  - Games
  - DBMS Apps
  - Scripting etc.

Python (a computer language)- Limitations

There are few limitations in Python which can be neglected because of its vast usage.

It is not a Fast Language.

Libraries are very less.

It is week in Type binding.

It is not easy to convert in some other language.

Tokens

- Token- is the smallest unit of any programming language.

  It is also known as Lexical Unit. Types of token are-

  i. Keywords
  ii. Identifiers (Names)
  iii. Literals
  iv. Operators
  v. Punctuators

Keywords

Keywords are those words which provides a special meaning to interpreter.
These are reserved for specific functioning.
These can not be used as identifiers, variable name or any other purpose.
Available keywords in Python are-

False	class	finally	is	return
None	continue	for	lambda	try
True	def	from	nonlocal	while
and	del	global	not	with
as	elif	if	or	yield
assert	else	import	pass	
break	except	in	raise	

Identifiers

- These are building blocks of a program and are used to give names to different parts/blocks of a program like - variable, objects, classes, functions.
- An identifier may be a combination of letters and numbers.
- An identifier must begin with an alphabet or an underscore(_). Subsequent letters may be numbers(0-9).
- Python is case sensitive. Uppercase characters are distinct from lowercase characters (P and p are different for interpreter).
- Length of an Identifier is unlimited.
- Keywords can not be used as an identifier.
- Space and special symbols are not permitted in an identifier name except an underscore(_) sign.
- Some valid identifiers are –
  - Myfile, Date9_7_17, Z2T0Z9, _DS, _CHK FILE13.
- Some invald identifiers are –
  - DATA-REC, 29COLOR, break, My.File.

Literals / Values

- Literals are often called Constant Values.
- Python permits following types of literals -
  - String literals - “Pankaj”
  - Numeric literals – 10, 13.5, 3+5j
  - Boolean literals – True or False
  - Special Literal *None*
  - Literal collections

String Literals

String Literal is a sequence of characters that can be a combination of letters, numbers and special symbols, enclosed in quotation marks, single, double or triple(“ “ or ‘ ‘ or “’ ‘”).

In python, string is of 2 types-

Single line string

Text = “Hello World” or Text = ‘Hello World’

Multi line string

Text = ‘hello\
world’

or Text = ‘’’hello
word ‘’’

Numeric Literals

- Numeric values can be of three types -
  - int (signed integers)
    - Decimal Integer Literals – 10, 17, 210 etc.
    - Octal Integer Literals - 0o17, 0o217 etc.
    - Hexadecimal Integer Literals – 0x14, 0x2A4, 0xABD etc.
  - float (floating point real value)
    - Fractional Form – 2.0, 17.5 -13.5, -.00015 etc.
    - Exponent Form - -1.7E+8, .25E-4 etc.
  - complex (complex numbers)
    - 3+5i etc.

Boolean Literals

- It can contain either of only two values – True or False
  - A= True
  - B=False

Special Literals

- None, which means nothing (no value).
  - X = None

Operators

- An Operator is a symbol that trigger some action when applied to identifier (s)/ operand (s)
- Therefore, an operator requires operand (s) to compute upon. example :

c = a + b

Here, a, b, c are operands and operators are = and + which are performing differently.

Punctuators

- In Python, punctuators are used to construct the program and to make balance between instructions and statements. Punctuators have their own syntactic and semantic significance.
- Python has following Punctuators -

‘, ”, #, \, (,), [,], {, }, @. ,, :, .. `, =

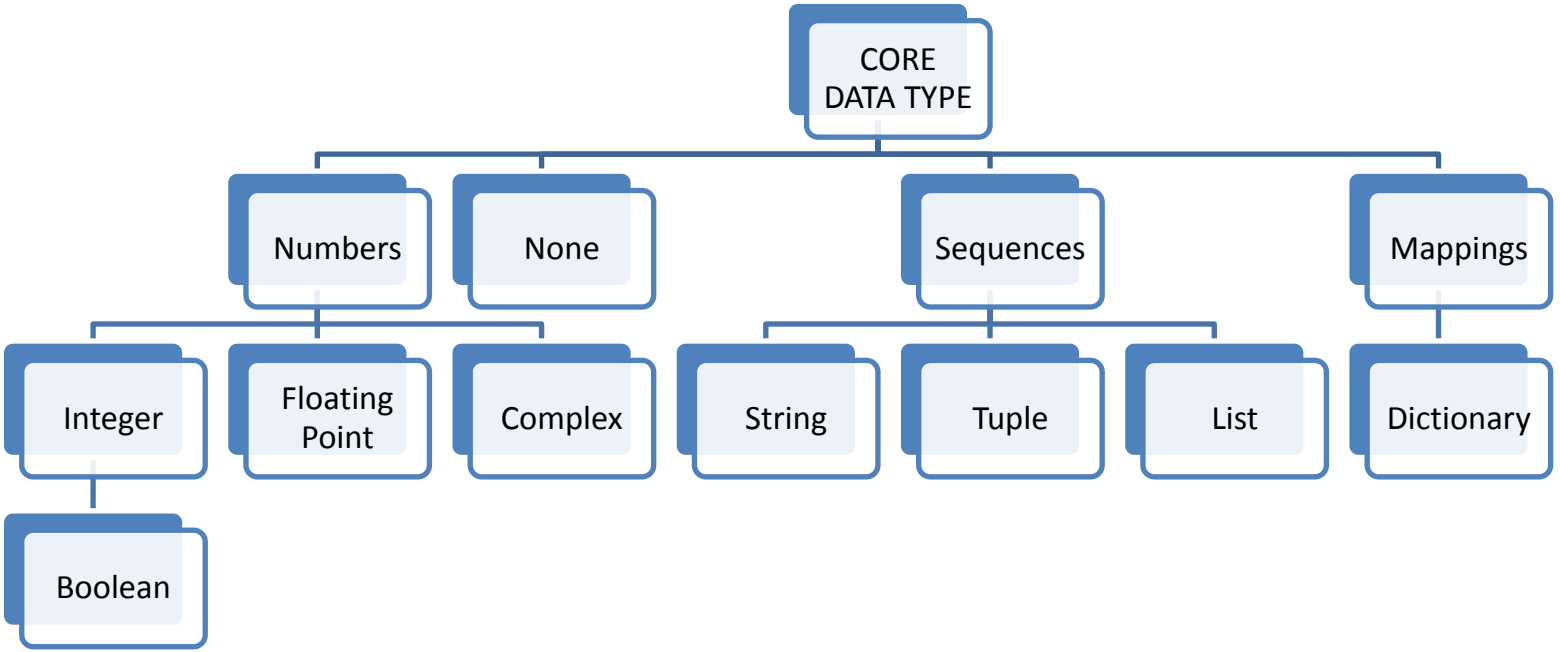
DATA TYPES

- Data can be of any type like- character, integer, real, string.
- Anything enclosed in " " is considered as string in Python.
- Any whole value is an integer value.
- Any value with fraction part is a real value.
- True or False value specifies boolean value.
- Python supports following core data types-

I.	Numbers	(int like10, 5) (float like 3.5, 302.24) (complex like 3+5j)
II.	String	(like "pankaj", 'pankaj', 'a', "a")
III.	List	like [3,4,5,"pankaj"] its elements are Mutable.
IV.	Tuple	like(3,4,5,"pankaj") its elements are immutable.
V.	Dictionary	like {'a':1, 'e':2, 'I':3, 'o':4, 'u':5} where a,e,i,o,u are keys and 1,2,3,4,5 are their values.

CORE DATA TYPES

Graphical View



Variables and Values

An important fact to know is-

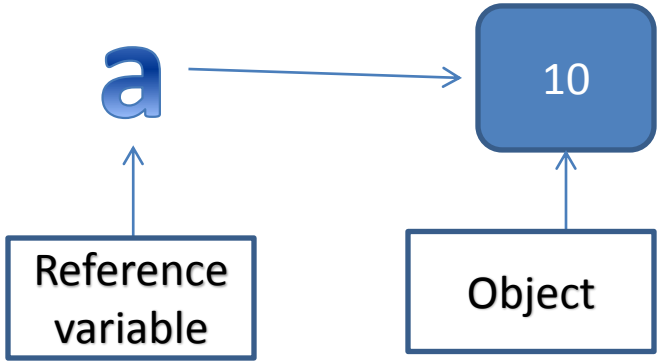
– In Python, values are actually objects.

– And their variable names are actually their reference names.

Suppose we assign 10 to a variable A.

A = 10

Here, value 10 is an object and A is its reference name.

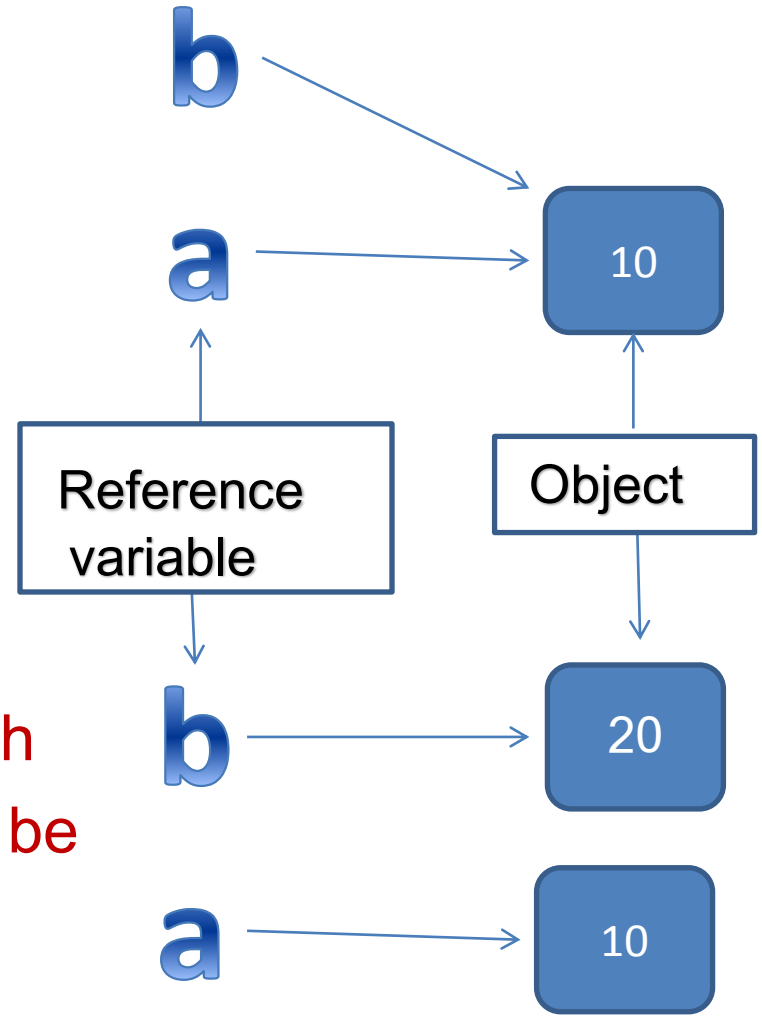


Variables and Values

If we assign 10 to a variable B,
B will refer to same object.

Here, we have two variables,
but with same location.

Now, if we change value of B like
B=20
Then a new object will be created with
a new location 20 and this object will be
referenced by B.



Mutable and Immutable Types

Following data types comes under mutable and immutable types-

- Mutable (Changeable)
  - lists, dictionaries and sets.
- Immutable (Non-Changeable)
  - integers, floats, Booleans, strings and tuples.

Operators

- The symbols that shows a special behavior or action when applied to operands are called operators. For ex- + , - , > , < etc.
- Python supports following operators-
  I. Arithmetic Operator
  II. Relation Operator
  III. Identity Operators
  IV. Logical Operators
  V. Bitwise Operators
  VI. Membership Operators

Operator Associativity

- In Python, if an expression or statement consists of multiple or more than one operator then operator associativity will be followed from left-to-right.

```
>>> 7*8/5//2
5.0
```

- In above given expression, first 7*8 will be calculated as 56, then 56 will be divided by 5 and will result into 11.2, then 11.2 again divided by 2 and will result into 5.0.

*Only in case of **, associativity will be followed from right-to-left.

```
>>> 3**3**2
19683
```

Above given example will be calculated as 3**(3**2).

Type Casting

- As we know, in Python, an expression may be consists of mixed datatypes. In such cases, python changes data types of operands internally. This process of internal data type conversion is called implicit type conversion.
- One other option is explicit type conversion which is like-

  <datatype> (identifier)

For ex-

a="4"

b=int(a)

Another ex-

If a=5 and b=10.5 then we can convert a to float.

Like d=float(a)

In python, following are the data conversion functions-

(1) int ( ) (2) float( ) (3) complex( ) (4) str( ) (5) bool( )

Taking Input in Python

- In Python, input () function is used to take input which takes input in the form of string. Then it will be type casted as per requirement. For ex- to calculate volume of a cylinder, program will be as-

```
#Program to calculate vloume of a cone
radius=int(input("Enter the radius of the Cylinder : "))
height=int(input("Enter the height of the Cylinder : "))
volume = 3.14* radius*radius*height
print("The Volume of the cylinder is : ",volume)
```

- Its output will be as-

```
>>>
 RESTART: C:/Users/KVBBKServer/AppData/Local/Programs/Python/Python36/VolOfCyl
inder.py
Enter the radius of the Cylinder : 10
Enter the height of the Cylinder : 5
The Volume of the cylinder is :  1570.0
>>>
```

Types of statements in Python

- In Python, statements are of 3 types-

» Empty Statements
  - pass

» Simple Statements (Single Statement)
  - name=input (“Enter your Name “)
  - print(name) etc.

» Compound Statements

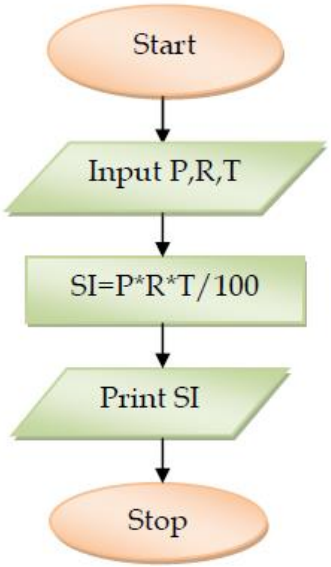
```
<Compound Statement Header>:
        <Indented Body containing multiple simple statements/compound statements>
```

- Here, Header line starts with the keyword and ends at colon (:).
- The body consists of more than one simple Python statements or compound statements.

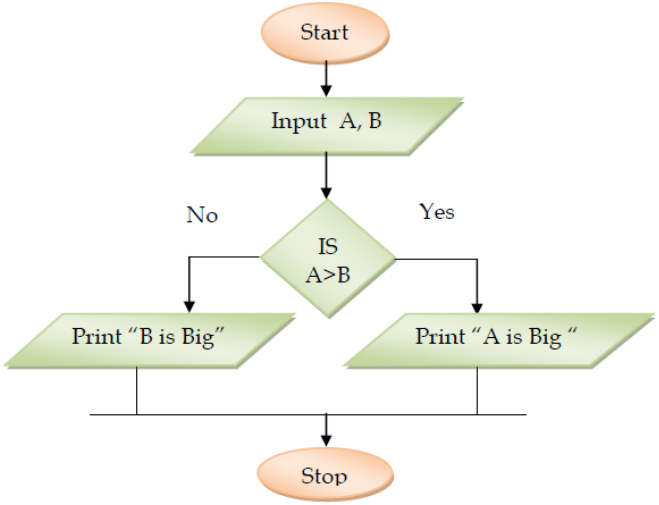
Statement Flow Control

- In a program, statements executes in sequential manner or in selective manner or in iterative manner.

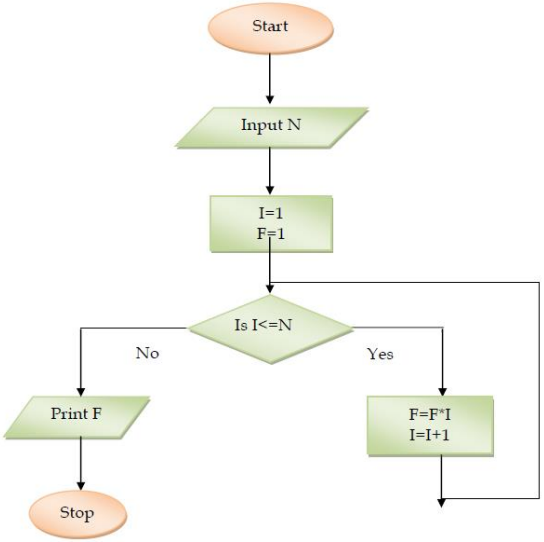
Sequential



Selective



Iterative

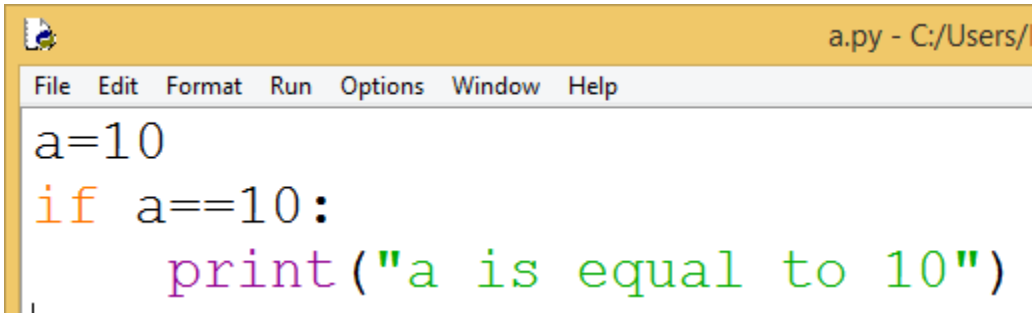


Python -----if Statements

- In Python, if statement is used to select statement for processing. If execution of a statement is to be done on the basis of a condition, if statement is to be used. Its syntax is-

```
if <condition>:
    statement(s)
```

like -



Python---if-else Statements

- If out of two statements, it is required to select one statement for processing on the basis of a condition, if-else statement is to be used. Its syntax is-

```
if <condition>:
        statement(s) when condition is true
else:
        statement(s) when condition is false
```

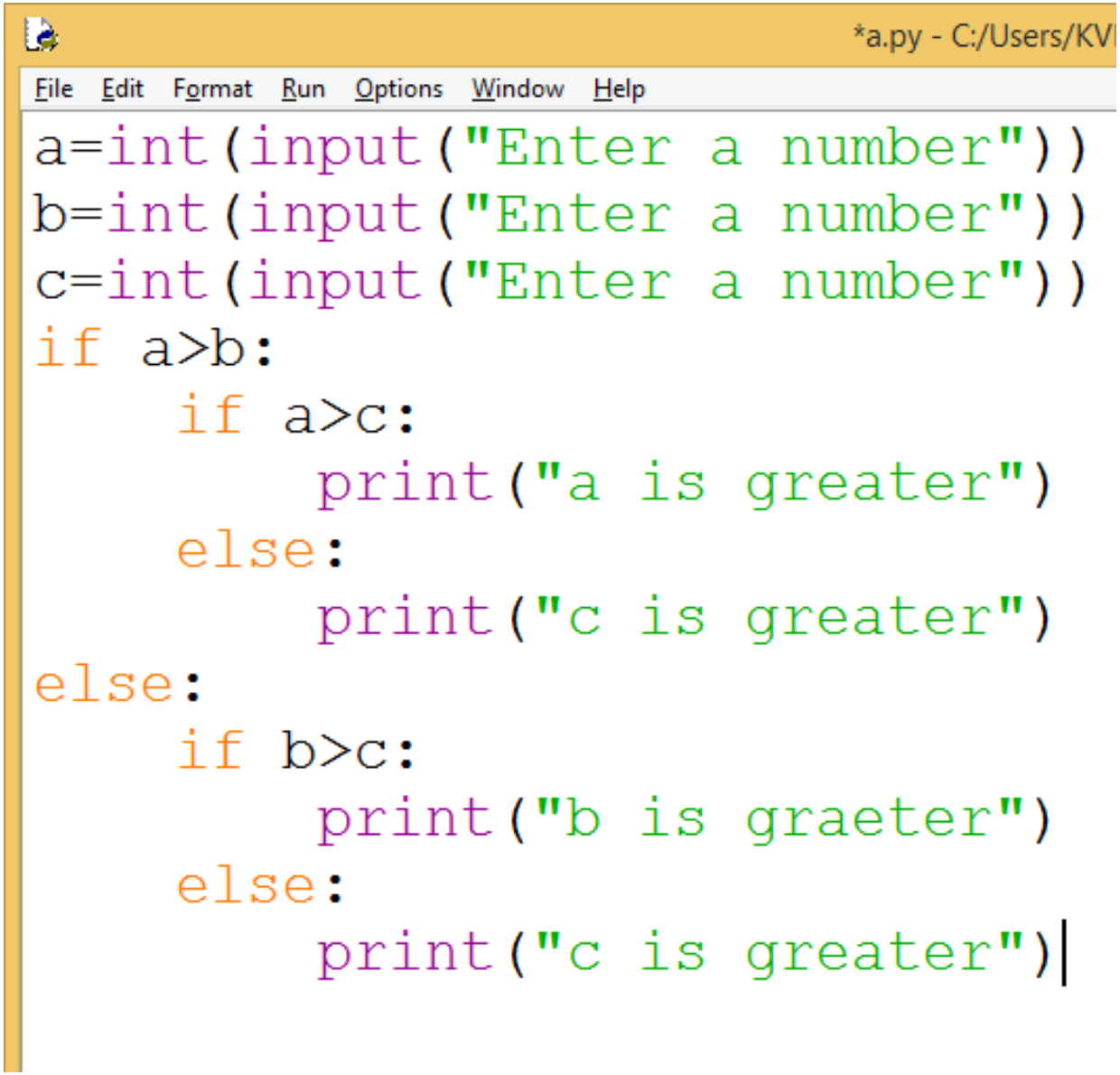
like -

a.py - C:/Users/KVBBKSer

File Edit Format Run Options Window Help

```
a=10
if a==10:
    print("a is equal to 10")
else:
    print("a is not equal to 10")
```

Nested If -else

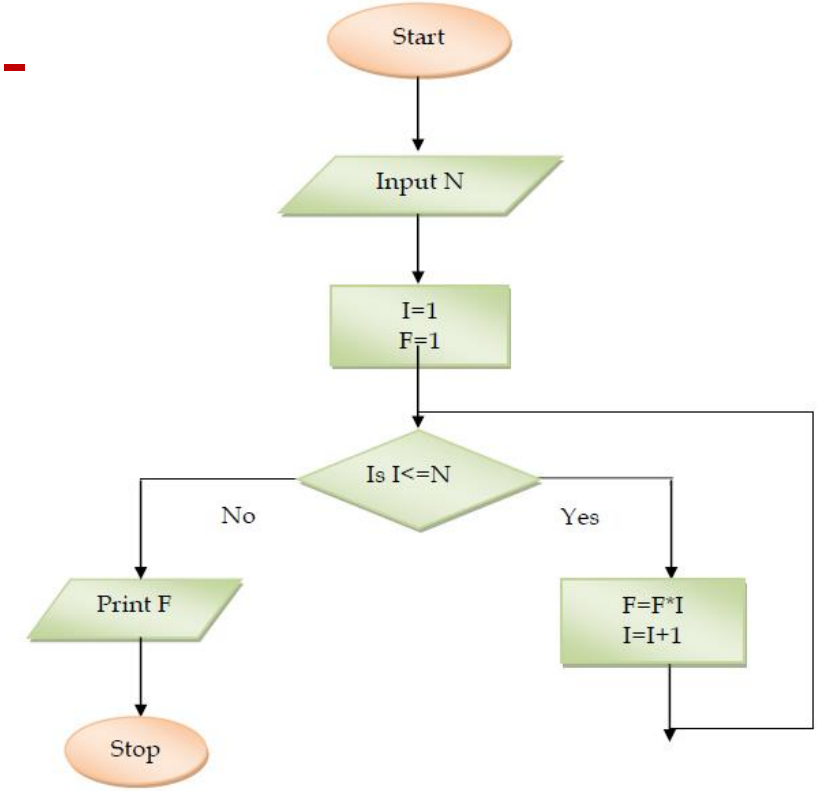


Loop/Repetitive Task/Iteration

These control structures are used for repeated execution of statement(s) on the basis of a condition.

Loop has 3 main components-

1. Start (initialization of loop)
2. Step (moving forward in loop)
3. Stop (ending of loop)



Python has following loops-

- for loop
- while loop

range () Function

- **In Python**, an important function is range( ). its syntax is-

range (<lower limit>,<upper limit>)

If we write - range (0,5)

Then a list will be created with the values [0,1,2,3,4] i.e. from lower limit to the value one less than ending limit.

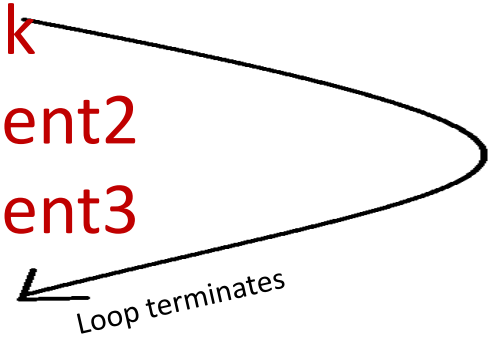
range (0,10,2) will have the list [0,2,4,6,8].

range (5,0,-1) will have the list [5,4,3,2,1].

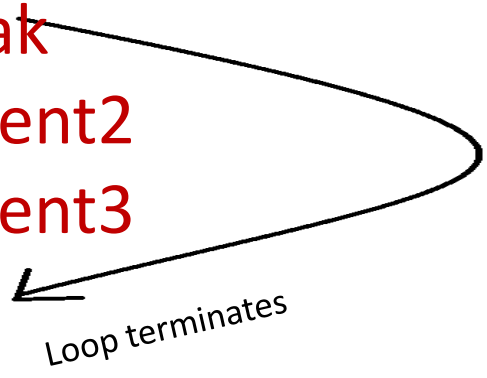
Jump Statements

break Statement

```
while <test-condition>:
        statement1
        if <condition>:
            break
        statement2
        statement3
Statement4
statement5
```




```
for <var> in <sequence>:
        statement1
        if <condition>:
            break
        statement2
        statement3
Statement4
statement5
```



Jump Statements

break Statement

```
n=int(input("Enter a number"))
c=1
while c<11:
    if c==5:
        break
    print(n,"x",c,"=",c*n)
    c=c+1
```

Output

```
Enter a number4
4 x 1 = 4
4 x 2 = 8
4 x 3 = 12
4 x 4 = 16
```

```
n=int(input("Enter a number"))
c=1
for c in range(1,11):
    if c==5:
        break
    print(n,"x",c,"=",c*n)
```

Output

```
Enter a number5
5 x 1 = 5
5 x 2 = 10
5 x 3 = 15
5 x 4 = 20
>>>
```

in and not in operator

- *in* operator-

  3 in [1,2,3,4] will return ***True***.

  5 in [1,2,3,4] will return ***False***.

– *not in* operator-

  5 not in [1,2,3,4] will return ***True.***

```
for a in [1,2,3]:
    print(a)
    print(a*a)
```

Jump Statements

continue Statement

```
n=int(input("Enter a number"))
for c in range(1,11):
    if c==5:
        continue
    print(n,"x",c,"=",c*n)
```

```
n=int(input("Enter a number"))
c=0
while c<11:
    c=c+1
    if c==5:
        continue
    print(n,"x",c,"=",c*n)
```

Output of both the program---

```
Enter a number5
5 x 1 = 5
5 x 2 = 10
5 x 3 = 15
5 x 4 = 20
5 x 6 = 30
5 x 7 = 35
5 x 8 = 40
5 x 9 = 45
5 x 10 = 50
5 x 11 = 55
>>>
```

Nested Loop

```
n=int(input("Enter the number"))
for r in range(1,n+1):
    for c in range(1,r+1):
        print("*", end="")
    print("")
```

OUTPUT

```
Enter the number5
*
**
***
****
*****
```

String Creation

- String can be created in following ways-
  1. By assigning value directly to the variable

```
>>> str="I love my india"
>>> str
'I love my india'
```

String Literal

  2. By taking Input

```
>>> str1=input("Enter a string")
Enter a stringThis is python
>>> str1
'This is python'
```

Input ( ) always return input in the form of a string.

Traversal of a string

- Process to access each and every character of a string for the purpose of display or for some other purpose is called string traversal.

```
name="superb"
for ch in name:
    print(ch,"-",end="")
```

Output

```
s -u -p -e -r -b -
```

Program to print a String after reverse -

```
str=input("Enter a String")
print("The string ", str," in reverse order is: ")
length=len(str)
for a in range(-1,(-length-1),-1):
    print(str[a],end="")
```

Output

```
Enter a Stringsanjeev
The string  sanjeev  in reverse order is:
veejnas
```

String Operators

- There are 2 operators that can be used to work upon strings + and *.

» + (it is used to join two strings)

- Like - “tea” + “pot” will result into “teapot”
- Like- “1” + “2” will result into “12”
- Like – “123” + “abc” will result into “123abc”

» * (it is used to replicate the string)

- like - 5*”@” will result into “@@@@@”
- Like - “go!” * 3 will result “go!go!go!”

note : - “5” * “6” expression is invalid.

String Slicing

- Look at following examples carefully-

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13
Word	R	E	S	P	O	N	S	I	B	I	L	I	T	Y
Reverse index	-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

word = “RESPONSIBILITY”

word[0 : 14] will result into‘RESPONSIBILITY’

word[0 : 3] will result into‘RES’

word[2 : 5] will result into‘SPO’

word[-7 : -3] will result into‘IBIL’

word[: 14] will result into‘RESPONSIBILITY’

word[: 5] will result into ‘RESPO’

word[3 :] will result into ‘PONSIBILITY’

String Functions

String.capitalize()	Converts first character to Capital Letter
String.find()	Returns the Lowest Index of Substring
String.index()	Returns Index of Substring
String.isalnum()	Checks Alphanumeric Character
String.isalpha()	Checks if All Characters are Alphabets
String.isdigit()	Checks Digit Characters
String.islower()	Checks if all Alphabets in a String.are Lowercase
String.isupper()	returns if all characters are uppercase characters
String.join()	Returns a Concatenated String
String.lower()	returns lowercased string
String.upper()	returns uppercased string
len()	Returns Length of an Object
ord()	returns Unicode code point for Unicode character
reversed()	returns reversed iterator of a sequence
slice()	creates a slice object specified by range()

List Creation

- List is a standard data type of Python. It is a sequence which can store values of any kind.
- List is represented by square brackets " [ ] "

For ex -

- [ ] Empty list
- [1, 2, 3] integers list
- [1, 2.5, 5.6, 9] numbers list (integer and float)
- ['a', 'b', 'c'] characters list
- ['a', 1, 'b', 3.5, 'zero'] mixed values list
- ['one', 'two', 'three'] string list

- In Python, only list and dictionary are mutable data types, rest of all the data types are immutable data types.

List Creation

- List can be created in following ways-
  - Empty list -

    L = [ ]
  - list can also be created with the following statement-

    L = list( )

```
>>> Mylist = list('hello')
>>> Mylist
['h', 'e', 'l', 'l', 'o']
>>>
```

```
>>> L = ('p','a','n','k','j')
>>> NewList = list(L)
>>> NewList
['p', 'a', 'n', 'k', 'j']
>>>
```

This is a Tuple

  - Long lists-

    even = [0, 2, 4, 6, 8, 10 ,12 ,14 ,16 ,18 ,20]
  - Nested list -

    L = [3, 4, [5, 6], 7]

Another method

```
>>> L1 = list(input("Enter List Elements"))
Enter List Elements12345
>>> L1
['1', '2', '3', '4', '5']
```

List Creation

-As we have seen in the example
That when we have supplied
values as numbers to a list even then
They have automatically converted to string.

```
>>> L1 = list(input("Enter List Elements"))
Enter List Elements12345
>>> L1
['1', '2', '3', '4', '5']
```

– If we want to pass values to a list in numeric form then we have to write following function -

eval(input())

L=eval(input("Enter list to be added "))

eval ( ) function identifies type of the passed string and then return it.

```
>>> a="15"
>>> b="25"
>>> print(a+b)
1525
>>> print(eval(a)+eval(b))
40
```

String Values

Another example

```
>>> a="15"
>>> b=eval(a)
>>> type(a)
<class 'str'>
>>> type(b)
<class 'int'>
>>>
```

Accessing a List

- First we will see the similarities between a List and a String.
- List is a sequence like a string.
- List also has index of each of its element.
- Like string, list also has 2 index, one for forward indexing (from 0, 1, 2, 3, ….to n-1) and one for backward indexing(from -n to -1).
- In a list, values can be accessed like string.

Forward index	0	1	2	3	4	5	6	7	8	9	10	11	12	13
List	R	E	S	P	O	N	S	I	B	I	L	I	T	Y
Backward index	-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

```
>>> vowels=['a','e','i','o','u']
>>> vowels[4]
'u'
>>> vowels[-5]
'a'
>>> vowels[-1]
'u'
```

Accessing a List

- len( ) function is used to get the length of a list.

```
>>> name=list("Pankaj")
>>> name
['P', 'a', 'n', 'k', 'a', 'j']
>>> len(name)
6
>>>
```

Important 1: membership operator (*in, not in*) works in list similarly as they work in other sequence.

- L[i] will return the values exists at i index.
- L [i : j] will return a new list with the values from i index to j index excluding j index.

```
>>> name=list("Pankaj")
>>> name[3]
'k'
>>> nm=name[2:4]
>>> nm
['n', 'k']
```

Important 2: + operator adds a list at the end of other list whereas * operator repeats a list.

Difference between a List and a String

- Main difference between a List and a string is that string is immutable whereas list is mutable.
- Individual values in string can't be change whereas it is possible with list.

```
>>> string="aeiou"
>>> string[2]
'i'
>>> string[2]='I'
Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
    string[2]='I'
TypeError: 'str' object does not support item assignment
```

Value didn't change in string. Error shown.

Value got changed in list specifying list is mutable

```
>>> st=list("aeiou")
>>> st
['a', 'e', 'i', 'o', 'u']
>>> st[2]='I'
>>> st
['a', 'e', 'I', 'o', 'u']
>>>
```

Traversal of a list

- Traversal of a list means to access and process each and every element of that list.
- Traversal of a list is very simple with for loop –

for <item> in <list>:

```
L=['P','Y','T','H','O','N']
for a in L:
    print(a)
```

```
 RESTART: C:/
P
Y
T
H
O
N
```

```
L=['P','Y','T','H','O','N']
length=len(L)
for a in range(length):
    print("Index ",a," और ",(a-length)," पर आइटम है : ",L[a])
```

```
 RESTART: C:/Users/KVBBKServer/
Index  0  और  -6  पर आइटम है :  P
Index  1  और  -5  पर आइटम है :  Y
Index  2  और  -4  पर आइटम है :  T
Index  3  और  -3  पर आइटम है :  H
Index  4  और  -2  पर आइटम है :  O
Index  5  और  -1  पर आइटम है :  N
```

*Python supports UNICODE therefore output in Hindi is also possible

List Operations (+, *)

- Main operations that can be performed on lists are joining list, replicating list and list slicing.
- To join Lists,+ ***operator ,*** is used which joins a list at the end of other list. With + operator, both the operands should be of list type otherwise error will be generated.

```
>>> L1=[1,2,3]
>>> L2=[4,5,6,7]
>>> L3=L1+L2
>>> L3
[1, 2, 3, 4, 5, 6, 7]
```

- To replicate a list, **** operator*** , is used.

```
>>> L1=[1,2,3]
>>> L2=L1*3
>>> L2
[1, 2, 3, 1, 2, 3, 1, 2, 3]
```

List Operations (Slicing)

- To slice a List, syntax is

seq = list [start : stop]

```
>>> LST=[10,12,14,20,22,24,30,32,34]
>>> SEQ=LST[3:-3]
>>> SEQ
[20, 22, 24]
>>> SEQ=LST[2:4]
>>> SEQ
[14, 20]
```

- Another syntax for List slicing is –

seq=list[start:stop:step]

```
>>> LST=[10,12,14,20,22,24,30,32,34]
>>> SEQ=LST[0:10:2]
>>> SEQ
[10, 14, 22, 30, 34]
>>> LST[2:10:3]
[14, 24, 34]
>>> LST[::3]
[10, 20, 30]
```

```
>>> LST[::-1]
[34, 32, 30, 24, 22, 20, 14, 12, 10]
```

Use of slicing for list Modification

- Look carefully at following examples-

```
>>> L=["one","two","three"]
>>> L
['one', 'two', 'three']
>>> L[0:2]=[0,1]
>>> L
[0, 1, 'three']
>>> L=["one","two","three"]
>>> L[0:2]="a"
>>> L
['a', 'three']
```

New value is being assigned here.

Here also, new value is being assigned.

See the difference between both the results.

```
>>> l=[1,2,3]
>>> l[2:]="604"
>>> l
[1, 2, '6', '0', '4']
>>> l[2:]=144
Traceback (most recent call last):
  File "<pyshell#12>", line 1, in <module>
    l[2:]=144
TypeError: can only assign an iterable
```

144 is a value and not a sequence.

List Functions and Methods

- Python provides some built-in functions for list manipulation
- Syntax is like

<list-object>.<method-name>

Function	Details
List.index(<item>)	Returns the index of passed items.
List.append(<item>)	Adds the passed item at the end of list.
List.extend(<list>)	Append the list (passed in the form of argument) at the end of list with which function is called.
List.insert(<pos>,<item>)	Insert the passed element at the passed position.
List.pop(<index>)	Delete and return the element of passed index. Index passing is optional, if not passed, element from last will be deleted.
List.remove(<value>)	It will delete the first occurrence of passed value but does not return the deleted value.

List Functions and Methods

Function	Details
List.clear ( )	It will delete all values of list and gives an empty list.
List.count (<item>)	It will count and return number of occurrences of the passed element.
List.reverse ( )	It will reverse the list and it does not create a new list.
List.sort ( )	It will sort the list in ascending order. To sort the list in descending order, we need to write----- list.sort(reverse =True).

Creation of Tuple

- In Python, “( )” parenthesis are used for tuple creation.

( )	empty tuple
(1, 2, 3)	integers tuple
(1, 2.5, 3.7, 7)	numbers tuple
(‘a’, ’b’, ’c’)	characters tuple
(‘a’, 1, ‘b’, 3.5, ‘zero’)	mixed values tuple
(‘one’, ’two’, ’three’, ’four’)	string tuple

***Tuple is an immutable sequence whose values can not be changed.**

Creation of Tuple

Look at following examples of tuple creation carefully-

- Empty tuple:

```
>>> t=()
>>> t
()
```

- Single element tuple:

```
>>> t=(1)
>>> t
1
```

- Long tuple:

```
>>> t=(0,1,4,9,16,25,36,49,64,81,100,121)
>>> t
(0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100, 121)
```

- Nested tuple:

```
>>> t=(1,2,3,(4,5))
>>> t
(1, 2, 3, (4, 5))
```

Creation of Tuple

tuple() function is used to create a tuple from other sequences.
See examples-

Tuple creation from string

```
>>> t=tuple("Hello")
>>> t
('H', 'e', 'l', 'l', 'o')
```

Tuple creation from list

```
>>> L=['a', 'e', 'i', 'o', 'u']
>>> T=tuple(L)
>>> T
('a', 'e', 'i', 'o', 'u')
```

Tuple creation from input

```
>>> t1=tuple(input("Enter element"))
Enter element123456
>>> t1
('1', '2', '3', '4', '5', '6')
```

All these elements are of character type. To have these in different types, need to write following statement.-

Tuple=eval(input("Enter elements"))

```
>>> t1=eval(input("Enter the elements"))
Enter the elements (2,4.5,"hello")
>>> t1
(2, 4.5, 'hello')
```

Accessing a Tuple

- In Python, the process of tuple accessing is same as with list. Like a list, we can access each and every element of a tuple.
- **Similarity with List-** like list, tuple also has index. All functionality of a list and a tuple is same except mutability.

Forward index	0	1	2	3	4	5	6	7	8	9	10	11	12	13
Tuple	R	E	S	P	O	N	S	I	B	I	L	I	T	Y
Backward index	-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

- len ( ) function is used to get the length of tuple.

```
>>> t=tuple("Hello")
>>> len(t)
5
```

Accessing a Tuple

- Indexing and Slicing:
  - T[i] returns the item present at index i.
  - T[i : j] returns a new tuple having all the items of T from index i to j.
  - T [i : j : n] returns a new tuple having difference of n elements of T from index i to j.

```
>>> T=(1,2,3,4,5,6,7,8,9,10)
>>> T[1:10:3]
(2, 5, 8)
```

- Membership operator:
  - Working of membership operator “in” and “not in” is same as in a list. (for details see the chapter- list manipulation).
  - Concatenation and Replication operators:
    - + operator adds second tuple at the end of first tuple. * operator repeats elements of tuple.

Accessing a Tuple

- Accessing Individual elements-

```
>>> L=['a', 'e', 'i', 'o', 'u']
>>> L[0]
'a'
>>> L[3]
'o'
```

- Traversal of a Tuple –

```
for <item> in <tuple>:
        #to process every element.
```

```
T=tuple("Python")
print(T,end="")
print()
for a in T:
    print(a)
```

```
('P', 'y', 't', 'h', 'o', 'n')
P
y
t
h
o
n
```

OUTPUT

Tuple Operations

- Tuple joining
  - Both the tuples should be there to add with +.

```
>>> tp1=(1,3,5)
>>> tp2=(6,7,8)
>>> tp1+tp2
(1, 3, 5, 6, 7, 8)
>>> tp3=tp1+tp2
>>> tp3
(1, 3, 5, 6, 7, 8)
```

- Some errors in tuple joining-
  - In Tuple + number
  - In Tuple + complex number
  - In Tuple + string
  - In Tuple + list
  - **Tuple + (5) will also generate error because when adding a tuple with a single value, tuple will also be considered as a value and not a tuple..**

- Tuple Replication-

```
>>> tp1=(1,3,5)
>>> tp1*3
(1, 3, 5, 1, 3, 5, 1, 3, 5)
>>> tp1
(1, 3, 5)
>>> tp2=tp1*3
>>> tp2
(1, 3, 5, 1, 3, 5, 1, 3, 5)
```

Tuple Slicing

```
>>> tpl=(10,12,14,20,22,24,30,32,34)
>>> seq=tpl[3:-3]
>>> seq
(20, 22, 24)
>>> tpl[3:30]
(20, 22, 24, 30, 32, 34)
>>> tpl[-15:7]
(10, 12, 14, 20, 22, 24, 30)
>>> tpl[0:10:2]
(10, 14, 22, 30, 34)
>>> tpl[0:10:3]
(10, 20, 30)
>>> tpl[::3]
(10, 20, 30)
```

Tuple will show till last element of list irrespective of upper limit.

Every alternate element will be shown.

Every third element will be shown.

Dictionary Creation

- To create a dictionary, it is needed to collect pairs of key:value in “{ }”.

<dictionary-name>={ <key1>:<value1>,<key2>:<value2>,<key3>:<value3>. . . }

Example:

teachers={“Rajeev”:”Math”, “APA”:”Physics”,”APS”:”Chemistry:”SB”:”CS”}

In above given example :

Key-value pair	Key	Value
“Rajeev”:”Math”	“Rajeev”	“Math”
“APA”:”Physics”	“APA”	“Physics”
“APS”:”Chemistry”	“APA”	“Chemistry”
“SB”:”CS”	“SB”	“CS”

Dictionary Creation

- Some examples of Dictionary are-

Dict1= { } # this is an empty dictionary without any element.

DayofMonth= { January":31, "February":28, "March":31, "April":30, "May":31, "June":30,
"July":31, "August":31, "September":30, "October":31, "November":30,
"December":31}

FurnitureCount = { "Table":10, "Chair":13, "Desk":16, "Stool":15, "Rack":15 }

– By above examples you can easily understand about the keys and their values.
– One thing to be taken care of is that keys should always be of immutable type.

Note: Dictionary is also known as associative array or mapping or hashes .

Dictionary Creation

- Keys should always be of immutable type.

- If you try to make keys as mutable, python shown error in it. For example-

```
>>> dict = {[2,3]:"MyRoom"}
Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    dict = {[2,3]:"MyRoom"}
TypeError: unhashable type: 'list'
```

Here key is a list which is of mutable type.

Here error shows that you are trying to create a key of mutable type which is not permitted.

Accessing a Dictionary

- To access a value from dictionary, we need to use key similarly as we use an index to access a value from a list.
- We get the key from the pair of Key: value.

teachers={“Rajeev”:”Math”, “APA”:”Physics”,”APS”:”Chemistry:”SB”:”CS”}

- If we execute following statement from above example-

```
>>> teachers={"Rajeev":"Math", "APA":"Physics", "APS":"Chemistry", "SB":"CS"}
>>> teachers["Rajeev"]
'Math'
>>> print("Rajeev Teaches ", teachers["Rajeev"])
Rajeev Teaches  Math
```

- We have selected key “Rajeev” and on printing it, Math got printed. Another example-

```
>>> d={"Vowel1":'a', "Vowel2":'e', "Vowel3":'i', "Vowel4":'o', "Vowel5":'u'}
>>> print(d["Vowel2"])
e
>>> print(d["Vowel5"])
u
```

If we access a non-key, error will come.

Traversal of a Dictionary

- To traverse a Dictionary, we use for loop. Syntax is-

for <item> in <dictionary>:

```
>>> d={5:"number","a":"String",(1,2):"tuple"}
>>> for k in d:
        print(k," : ",d[k])
```

```
5   :   number
a   :   String
(1, 2)   :   tuple
```

Here, notable thing is that every key of each pair of dictionary d is coming in k variable of loop. After this we can take output with the given format and with print statement.

Assignment : Develop a dictionary of your friends in which key will be your friend's name and his number will be its value.

Traversal of Dictionary

- To access key and value we need to use keys() and values().for example-

```
>>> d={"Vowel1":'a', "Vowel2":'e', "Vowel3":'i', "Vowel4":'o', "Vowel5":'u'}
>>> d.keys()
dict_keys(['Vowel1', 'Vowel2', 'Vowel3', 'Vowel4', 'Vowel5'])
>>> d.values()
dict_values(['a', 'e', 'i', 'o', 'u'])
```

- d.keys( ) function will display only key.
- d.values ( ) function will display value only.

Features of Dictionary

1. **Unordered set:** dictionary is a unordered collection of key:value pairs.
2. **Not a sequence:** like list, string and tuple , it is not a sequence because it is a collection of unordered elements whereas a sequence is a collection of indexed numbers to keep them in order.
3. Keys are used for its indexing because according to Python key can be of immutable type. String and numbers are of immutable type and therefore can be used as a key. Example of keys are as under-

```
>>> d={0:"Key0",1:"Key1","3":"KeyAsString",(4,5):"KeyAsTuple","Hello":6}
>>> d[0]
'Key0'
>>> d[1]
'Key1'
>>> d["3"]
'KeyAsString'
>>> d[(4,5)]
'KeyAsTuple'
>>> d["Hello"]
6
```

Key of a Dictionary should always be of immutable type like number, string or tuple whereas value of a dictionary can be of any type.

Features of Dictionary

4. **Keys should be unique :** Because keys are used to identify values so they should be unique.
5. Values of two unique keys can be same.
6. Dictionary is mutable hence we can change value of a certain key. For this, syntax is-

<dictionary>[<key>] = <value>

```
>>> d={0:"Key0",1:"Key1","3":"KeyAsString",(4,5):"KeyAsTuple","Hello":6}
>>> d["3"]="This is String"
>>> d
{0: 'Key0', 1: 'Key1', '3': 'This is String', (4, 5): 'KeyAsTuple', 'Hello': 6}
```

4. Internally it is stored as a mapping. Its key:value are connected to each other via an internal function called ***hash-function**.*** Such process of linking is knows as mapping.

**Hash-function is an internal algorithm to link a and its value.

Working with Dictionary

- Here we will discuss about various operation of dictionary like element adding, updation, deletion of an element etc. but first we will learn creation of a dictionary.
- **Dictionary initialization-** For this we keep collection of pairs of key:value separated by comma (,) and then place this collection inside "{ }".

```
>>> Employee={'name':'suresh','salary':15000,'age':34}
>>> Employee
{'name': 'suresh', 'salary': 15000, 'age': 34}
```

- **Addition of key:value pair to an empty dictionary.** There are two ways to create an empty dictionary-
  1. Employee = { }
  2. Employee = dict( )

After that use following syntax-

<dictionary>[<key>] = <value>

```
>>> Employee = {}
>>> Employee['name']='Pankaj'
>>> Employee['salary']=20000
>>> Employee
{'name': 'Pankaj', 'salary': 20000}
```

Working with Dictionary

3. **Creation of a Dictionary with the pair of name and value**: dict( ) constructor is used to create dictionary with the pairs of key and value. There are various methods for this-

I. By passing Key:value pair as an argument:

```
>>> Employee=dict(name='Ramesh',salary=10000,age=24)
>>> Employee
{'name': 'Ramesh', 'salary': 10000, 'age': 24}
```

The point to be noted is that here no inverted commas were placed in argument but they came automatically in dictionary.

II. By specifying Comma-separated key:value pair-

```
>>> Employee=dict({'name':'Rahul','age':24})
>>> Employee
{'name': 'Rahul', 'age': 24}
```

Working with Dictionary

III. By specifying Keys and values separately:

For this, we use zip() function in dict ( ) constructor-

```
>>> Employee=dict(zip(('name','salary','age'),('Mukesh',12000,26)))
>>> Employee
{'name': 'Mukesh', 'salary': 12000, 'age': 26}
```

IV. By giving Key:value pair in the form of separate sequence:

```
>>> Employee=dict([['name','anand'],['salary',14000],['age',23]])
>>> Employee
{'name': 'anand', 'salary': 14000, 'age': 23}
```

By passing List

```
>>> Employee=dict((['name','Angad'],['salary',11000],['age',29]))
>>> Employee
{'name': 'Angad', 'salary': 11000, 'age': 29}
```

By passing tuple of a list

```
>>> Employee=dict((('name','Suman'),('salary',17000),('age',21)))
>>> Employee
{'name': 'Suman', 'salary': 17000, 'age': 21}
```

By passing tuple of tuple

Adding an element in Dictionary

following syntax is used to add an element in Dictionary-

```
>>> Employee=dict((('name','Suman'),('salary',17000),('age',21)))
>>> Employee
{'name': 'Suman', 'salary': 17000, 'age': 21}
>>> Employee['Dept']='Sales'
>>> Employee
{'name': 'Suman', 'salary': 17000, 'age': 21, 'Dept': 'Sales'}
```

Nesting in Dictionary

look at the following example carefully in which element of a dictionary is a dictionary itself.

```
Employee={'mukesh':{'age':23,'salary':34000},'Meena':{'age':27,'salary':24000}}
for key in Employee:
    print("Employee",key,':')
    print('Age: ',str(Employee[key]['age']))
    print('Salary: ',str(Employee[key]['salary']))
```

```
Employee mukesh :
Age:  23
Salary:  34000
Employee Meena :
Age:  27
Salary:  24000
```

Updation in a Dictionary

following syntax is used to update an element in Dictionary-

<dictionary>[<ExistingKey>]=<value>

```
>>> Employee={'name':'Sudha','Salary':10000,'age':24}
>>> Employee['Salary']=15000
>>> Employee
{'name': 'Sudha', 'Salary': 15000, 'age': 24}
```

WAP to create a dictionary containing names of employee as key and their salary as value.

File Edit Format Run Options Window Help

```
n=int(input("Enter the Number of Employees"))
Employee={}    #Empty Dictionary
for a in range(n):
    key=input("Enter Name of the Employee")
    value=int(input("Enter Salary of Employee"))
    Employee[key]=value
print("The Dictionary is now is :")
print(Employee)
```

Output

```
Enter the Number of Employees3
Enter Name of the EmployeePavan
Enter Salary of Employee30000
Enter Name of the EmployeeRakesh
Enter Salary of Employee12000
Enter Name of the EmployeeMukesh
Enter Salary of Employee20000
The Dictionary is now is :
{'Pavan': 30000, 'Rakesh': 12000, 'Mukesh': 20000}
```

Deletion of an element from a Dictionary

following two syntaxes can be used to delete an element form a Dictionary. For deletion, key should be there otherwise python will give error.

1. del <dictionary>[<key>]- it only deletes the value and does not return deleted value.

```
>>> emp={'Pavan': 30000, 'Rakesh': 12000, 'Mukesh': 20000}
>>> del emp['Pavan']
>>> emp
{'Rakesh': 12000, 'Mukesh': 20000}
```

Value did not return after deletion.

2. <dictionary>.pop(<key>) it returns the deleted value after deletion.

```
>>> emp={'Pavan': 30000, 'Rakesh': 12000, 'Mukesh': 20000}
>>> emp.pop('Rakesh')
12000
>>> emp.pop('Aman',"Not Found")
'Not Found'
```

Value returned after deletion.

If key does not match, given message will be printed.

Detection of an element from a Dictionary

Membership operator is used to detect presence of an element in a Dictionary.

<key> in <dictionary> it gives true on finding the key otherwise gives false.

<key> not in <dictionary> it gives true on not finding the key otherwise gives false.

```
>>> emp={'Pavan': 30000, 'Rakesh': 12000, 'Mukesh': 20000}
>>> 'Pavan' in emp
True
>>> 'Hari' in emp
False
>>> 'Hari' not in emp
True
>>> 'Pavan' not in emp
False
```

*** in and not in does not apply on values, they can only work with keys.**

Pretty Printing of a Dictionary

To print a Dictionary in a beautify manner, we need to import ***json module.*** After that following syntax of dumps ( ) will be used.

json.dumps(<>,indent=<n>)

```
>>> import json
>>> emp={'Pavan': 30000, 'Rakesh': 12000, 'Mukesh': 20000}
>>> print(json.dumps(emp,indent=2))
{
  "Pavan": 30000,
  "Rakesh": 12000,
  "Mukesh": 20000
}
```

Program to create a dictionary by counting words in a line

```
import json
statement="His Name is Pankaj. \
His father is a teacher. His \
father is a good person"
w=statement.split()
d={}
for c in w:
    key=c
    if key not in d:
        count=w.count(key)
        d[key]=count
print("Counting frequencies in list\n",w)
print(json.dumps(d,indent=1))
```

```
Counting frequencies in list
 ['His', 'Name', 'is', 'Pankaj.', '
His', 'father', 'is', 'a', 'teacher
.', 'His', 'father', 'is', 'a', 'go
od', 'person']
{
 "His": 3,
 "Name": 1,
 "is": 3,
 "Pankaj.": 1,
 "father": 2,
 "a": 2,
 "teacher.": 1,
 "good": 1,
 "person": 1
}
```

Here a dictionary is created of words and their frequency.

Dictionary Function and Method

1. *len( )* Method : it tells the length of dictionary.

```
>>> emp={'Pavan': 30000, 'Rakesh': 12000, 'Mukesh': 20000}
>>> len(emp)
3
```

2. *clear( )* Method : it empties the dictionary.

```
>>> emp={'Pavan': 30000, 'Rakesh': 12000, 'Mukesh': 20000}
>>> emp.clear()
>>> emp
{}
```

3. *get( )* Method : it returns value of the given key.

```
>>> emp={'Pavan': 30000, 'Rakesh': 12000, 'Mukesh': 20000}
>>> emp.get('Rakesh')
12000
>>> emp.get('Ankit',"Not Found")
'Not Found'
```

It works similarly as <dictionary>[<key>]

On non finding of a key, default message can be given.

Dictionary Function and Method

4. ***items( )*** Method : it returns all items of a dictionary in the form of tuple of (key:value).

```
>>> emp={'Pavan': 30000, 'Rakesh': 12000, 'Mukesh': 20000}
>>> mylist=emp.items()
>>> mylist
dict_items([('Pavan', 30000), ('Rakesh', 12000), ('Mukesh', 20000)])
```

5. ***keys( )*** Method : it returns list of dictionary keys.

```
>>> emp={'Pavan': 30000, 'Rakesh': 12000, 'Mukesh': 20000}
>>> emp.keys()
dict_keys(['Pavan', 'Rakesh', 'Mukesh'])
```

6. ***values( )*** Method : it returns list of dictionary values.

```
>>> emp={'Pavan': 30000, 'Rakesh': 12000, 'Mukesh': 20000}
>>> emp.values()
dict_values([30000, 12000, 20000])
```

Dictionary Function and Method

7. *Update ( )* Method: This function merge the pair of key:value of a dictionary into other dictionary. Change and addition in this is possible as per need. Example-

```
>>> emp1={'name':'Suresh','salary':10000,'age':24}
>>> emp2={'name':'siya','salary':45000,'dept':'sales'}
>>> emp1.update(emp2)
>>> emp1
{'name': 'siya', 'salary': 45000, 'age': 24, 'dept': 'sales'}
>>> emp2
{'name': 'siya', 'salary': 45000, 'dept': 'sales'}
```

In the above given example, you can see that change is done in the values of similar keys whereas dissimilar keys got joined with their values.

Thank you

please follow us on our blog-

www.pythontrends.wordpress.com