Python Basics

> Comments

> Variables

> Standard Data Types.

Numbers

→ int

→ float

→ Complex.

String

> Type Conversions.

→ int ()

→ float ()

→ str()

python ~~w~~.py.
file name

* python doesn't Supports curly braces.
* to avoid complexity we use indentation.

* Operators in python
* Control statements

1) Operators :

An Operator is a Symbol, which is responsible to perform Operation b/w the Symbols.

1. Arithmetic Operator :-

Used to perform Mathematical Operations

(i) +
(ii) −
(iii) *
(iv) / (divison)
(v) % (Modulous)
(vi) // (floor division)

Ex a=10 b=3

- a//b = 3 (It returns integer type values)
- a/b = 3.33

(vii) Exponent (**)

Ex a=2 b=3.

$a**b = 2^3 = 8.$

2. Comparison Operators :

(returns either true or false).

(i) ==
(ii) !=
(iii) <=
(iv) >=
(v) <
(vi) >

3. Assignment Operators :

(i) = (Basic Assignment Operator)
(ii) +=
(iii) -=
(iv) *=
(v) /=
(vi) %=
(vii) //=
(viii) **=

4) Logical Operators

(i) and (used to check two or more
(ii) or conditions at a time).
(iii) not

a, b, c = 3, 2, 5

Ex: Syntax: * a>b and a>c
false

* a>b or a>c
True.

* not a>c
false.

5) Bitwise Operators

To perform Binary level Operations. (This Operator are performed only on numbers)

(i) & (Bitwise and)
(ii) | (" or)
(iii) ^ (" xor)
(iv) ~ (" negation)
(v) << (" left shift)
(vi) >> (" right shift)

Ex:-
a = 3 011
b = 2 010

* print (a & b)
2

* print (a | b)
3

* print (a >> b)
0

* print (a << b)
12

6) Membership Operators (used to check whether particular item is present in the given list)

(used in Sequences (or) data structures)

(i) in (returns true if element is present)

(ii) not in (returns true if element is not present in given list or any sequence)

* python doesn't suppoets array concepts
* to Overcome this problem python uses sequences.

Sequences - list, tuple, ...
↓
Collection of heterogenous Operators

Ex: l1 = [1, 2, 3, 4, 5, 6]

print (l1)

>> 1, 2, 3, 4, 5, 6

>> print (4 in l1)

True

>> print (7 in l1)

false

>> print (4 not in l1)

True.

Control statements

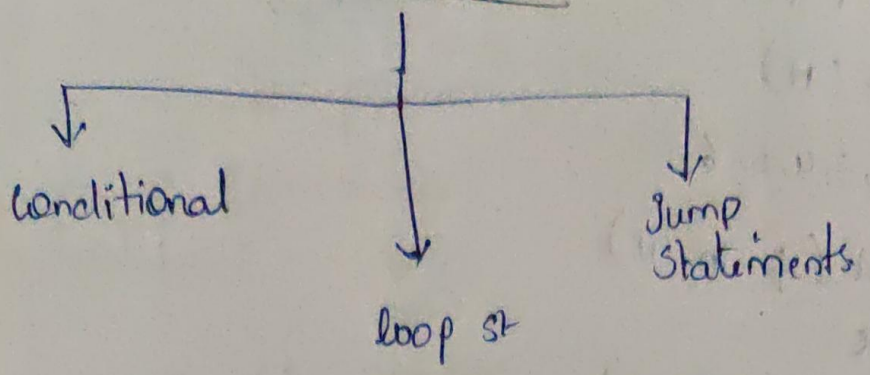
used to control the flow of Execution of program.

* Indentation :- It starts with ':' and It can have One tab space (or) 4 void spaces.

```
if (a>b):
        print ("a is max")
        print ("Done")
print ("Othercode")
```

→ This two statements come under the if block. because both have same indentation.

Control Statements

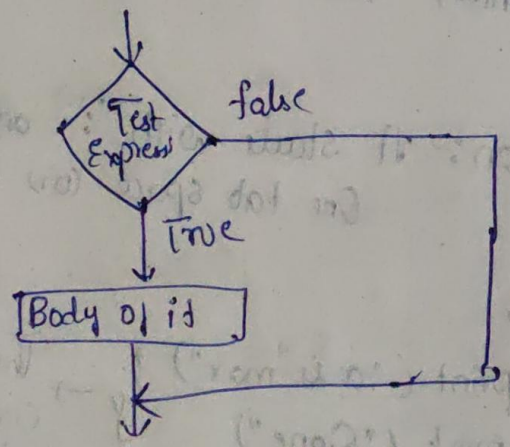


* Conditional statements :

Python Supports following Conditional statements

(i) if
(ii) if-else
(iii) if-elif-else

(i) if



Syntax : if Condition :
(One tab space) Statement 1
Statement 2

* Example

```
a = 23
b = 100
if b>a:
    print ("b is max")
    print ("done")
if a>b:
    print("a is max")
print("done")
```

o/p : b is max
done

input( ):

↳ This function is used to get Input from the User.

Ex: a = input(" Enter any value");

(or)

a = input( );

Output:

Program

```
a = int(input(" Enter 'a' value")
b = float(input(" Enter b value")
if b>a:
    print("b is max")
```

```
Enter a value 2
Enter b value 3
b is max.
```

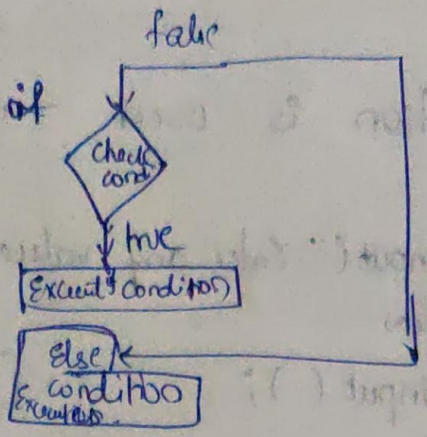
* Input(): reads the input as string.
  - We need to convert the given input to any integer format by using functions as int (or) float (or) double.

Ex: a = int(input(" Enter a value"))

(or)

a = float(input( ))

> if - else :



Syntax:

```
if condition:
    statements 1
else:
    statements 2
```

Example:

```
age = int(input("Enter your age:"))
if age>=18:
    print("You are In")
else:
    print("You are Out")
```

Output 1

```
Enter your age : 48
You are In
```

Output 2

```
Enter your age : 14
You are Out
```

→ if - elif - else : can check more conditions.

Syntax:

```
if Condition1:
    Statements1
elif Condition2:
    statement2
elif Condition3:
    statemet3
    :
else:
    statements.
```

Ex: max of three Numbers.

```
a = int (input ("Enter a value : "))
b = int (input (" Enter b value :"))
c = int (input (" Enter c value : "))
if (a>b) && (a>c):
    print (" a is max:",a)
elif (b>c) && (b>a):
    print (" b is max:",b)
else:
    print ("c is max:", c)
```

python ifelidemo.p

Olp:-

```
Enter a value : 4
Enter b value : 6
Enter c value : 2
b is max : 6
```

> Loop Statements

(To Execute the Statements Multiple times).

* python Supports two types of Loop Statements)

1) while Loop
2) for Loop

1) while Loop

All the statement in while loop are Executed untile the condition is true.

* If we don't know the number of iteration then we can go with the while loop.

Syntax: while Expression:
    Statements

Example: To print 1 to 5 Numbers.

```
i = 1
while (i<=5):
    print(i)
    i = i+1
```

Olp:

```
python filename.py
1
2
3
4
5
```

To print all the nos in the same line.

```
print(i, end=" ")
```

Olp: 1 2 3 4 5

Note

* python allows us to use while block with Else block.

(While with else)

Syntax:
```
While Expression:
    Statements(s)
else:
    statements.
```

Ex:
```
i=1
While (i<=5)
    print(i, end=" ")
    i=i+1
else:
    print ("Loop is Over")
```

O/P: 1 2 3 4 5 Loop is Over.

for with else

> for Loop: used to (Iterate...)

* Mainly used for working with the Sequences

Syntax:
```
for var in sequence:
    statements.
```

Example:
```
i=1
n=int(input("Enter n value :"))
for i in range(i,n+1):
    print(i)
else:
    print("Loop is Over")
```

range: Inbuit function in python to produce the range of function

* range function by default display the range of values from starting to (n-1) values

O/P: python fn.py
```
Enter n value : 5
1
2
3
4
5
Loop is Over.
```

Jump statements :

Used to alter the flow of Execution

* python supports following jump statements.

(1) break

(2) continue

(3) pass.

(1) break : used to exit from the current loop.

Syntax: break.

Example

```
i = 1
while (i<=5):
    print(i)
    if (i==3):
        break
    i=i+1
else:
    print("Loop is
        Successful")
```

O/p

```
1
2
3
```

```
1
2
3
```

* (Else block executes if a while block executes all the iteration successfully)

(2) Continue : It skips the current Iteration.

Syntax: continue.

Example

```
i= 1
n= int (input ("Enter n value: "))
for i in rang (i,n+1):
    if (i%2==0):
        continue
    print (i)
else:
    print ("Loop is Over")
```

O/P

```
Enter n value : 10
1
3
5
7
9
Loop is Over
```

(3) pass : It doesn't displays anything.
(as a null statement)

Syntax: pass.

Example

```
a = 12
if (a>=10):
    print("Hello")
else:
    pass
```

O/P:

Hello.

```
a = 3
```

O/P:

∴ it doesn't displays any thing.

String Sequence in Python

08/aug

* String Declaration:

String is a collection or Sequence of characters which are Enclosed with Single, double or triple quotes.

Ex:
```
str1 = 'Hello'    # Single quotes
str2 = "Hello"    # double quotes
str3 = '''Hello'''  # triple.
```

Note: python doesn't Support char Datatype
* Even Single char is treated as string in python.

Ex: str4 = 'p' # string with length of 1.

* String Indexing:

String Indexing starts from 0. and upto length-1

Ex:
```
Str = "Hello"
      01234
print(str[0]) → (Olp H)
print(str[3]) → (Olp L)
```

* python allows to access Sequence of characters

Ex: print(str[1:3]) → EL
↳ (range slice Operator)

* python supports Negative Indexing:
→ Negative Indexing starts from -1

Ex:
```
H  E  L  L  O   (Indexing starts
-5 -4 -3 -2 -1   from last char)
```

to access last char of string
print(str[-1]) → O

print(str[-2]) ⟶ L.

* To access Sequence of characters.

print(str[-3:-1]) ⟶ LL

(-3,-2) → it prints.

Ex programs

* str1 = 'Hello'
  str2 = "Hello"
  str3 = '''Hello'''
  print(str1, str2, str3)

  O/P: Hello Hello Hello

* String Indexing

  str = "Hello"
  (indexes: 0, 1, 2, 3, 4)

  print(str[0]) → H
  print(str[2]) → L
  print(str[-1]) → O
  print(str[-2]) → L
  print(str[-3:-1]) → LL

Range slice Operator :- This Operator is used to access the Sequence of characters in the strings.

Syntax: StringName[min possible value : max possible value]

* String Operators : *

These operators are used to

They are:

(1) Concatination Operator (+):

To Concatinate two or more strings.

Ex: str1 = "Hello"
str2 = "World"
print (str1 + str2)
Op HelloWorld.

* Repitition Operator (*):

Used to concatinate some string Multiple times.

Ex: str1 = "Hello"
print (str1 * 3) → Number specifies how many times the number should repeat
Op HelloHelloHello

* Slice Operator ([ ]):

Used to access the sub part of the given string.

Ex: str = "Hello" (0,1,2,3,4)
print (str[4])
Op o

Note: Used to access only one char from the give string

* Range slice Operator ([:]):

Used to access the range of character from the given string.

Ex: str = "Hello"
0,1,2,3,4

print(str[2:4])

Op: ll

* Membership Operator

> in → It returns true if the particular given substring is present in the given string.

Ex: rollno = "177Z1A0512"

print ("7z" in rollno)

Op: true

(if 7z is present in the given strings it returns true else, it return false)

> not in → It returns true if the particular given substring is not present in the given string.

* raw string Operator ([r]):

Used To print the string as it is On screen.

Ex: print (r"Hello \n world")

Op:- Hello \n World

* formating Operator (%):

Used to format the string.

Ex: str = "Hello"
print ("The string is : %s" %(str))

The string is : Hello

Welcome
-7 -6 -5 -4 -3 -2 -1

Hello World
0 1 2 3 4 5 6 7 8 9 10

Debug

* String functions and Methods *.

functions & Methods

> len() : Used to find the length of the (fun) string. (no of chars)

Ex:
str = "python"
print(len(str))

Olp 6.

Note:
* It counts the length including void spaces.

> lower() : This function is used to convert the (Method) given string into lower case letters.

Ex: str = "PYTHON"
print(str.lower())

Olp
python

> upper() (Method) :- This Method Converts the given string into uppercase letters.

Ex:

```
str = "python world"
print(str.upper())
```

o/p: PYTHON WORLD

> replace() (fun): It is used to replace old characters with new characters.

Ex:

1)
```
str = "Python world"
str1 = str.replace("python", "hello")
print(str1)
```

o/p: hello world

2)
```
str = "java is Object-Oriented and java"
str1 = str.replace("java", "python")
print(str1)
```

o/p: python is Object-Oriented and python.

Note: To replace only some Ocurences, then we need to give Count.

Ex:
```
str = "java is Object-Orient and java"
str1 = str.replace("java", "python", 1)
print(str1)
```

o/p: python is Object-Oriented and java.

> split() : Used to split the given string into no of items i.e,

* It converts the given string into list of items

Ex:

```
str = "python is programming language"
ls = str.split()
print(ls)
```

Op

['python', 'is', 'programming', 'language'].

Note:- By defaut delimiter is space dlr

* str = 'python & is& . . .
  ls = str.split(sp=&) Op ['python', 'is', ...]
  (consider & as delimiter)

> find() : Used to find substring in the given string and returns index of first Occurence of substring in the given string.

** if substring is not found in the given string it returns -1

Ex: str = "python is programming language"
0 1 2 3 4 5 6 7 8

```
ind1 = str.find('is')
print(ind1)
```

Op 7

to find second p.

* (subl, start, End])

* ~~print~~ ind1 = str.find('p', 5)
  print(ind1)
  (It returns, a/c the 5th pos index)

Op 10

index

* ind1 = str.find('i', 5, 10)

after 5th position and b/w 10th position.

print(ind1)

Op: 7

(It returns the first Occurance)

* ind1 = str.find('java')

print(ind1)

Op:

-1 (sub not found)

> index(): Same as find() but, it returns Error message if the substring was not found in the given string

```
          0 1 2 3 4 5 6 7 8 9 10 11 12 13
str = "python is proglomming"
ind1 = str.index("is")
print(ind1)
ind1 = str.index("p", 3, 15)
print(ind1)
ind1 = str.index("p", 3, 10)
print(ind1)
```

Op:

7

10

ValueError (b'cs there is no p blw 3 and 10)

> isalnum() : Used to check all the characters in the given string is alphanumeric or not

* There are 2 cases

(i) True : all characters are alphabets or numbers

(ii) false : other than alphabets or numbers, if there is any special symbol.

Ex' str1 = "python"
print(str1.isalnum()) — Op: True (Only alphanum char)

str2 = "python383"
print(str2.isalnum()) — Op: True (Only alphanum char)

str3 = "python3.8.3"
print(str2.isalnum()) — Op: false. (special char (.))

> isnumeric() : checks, wether all the characters in the given string are numeric or not

case
(i) True : if all character are numbers
(ii) false : any characters or special symbols.

Ex' str1 = "12345"
str2 = "python383"
print(str1.isnumeric()) — Op: True
print(str2.isnumeric()) — Op: false
str3 = "3.8.3" print(str3.isnumeric()) — Op: false

> islower(): Used ~~to the~~ to check wether the given string contains all the characters in lower case.

* true : if contains all characters in lowercase.
* false : if not in Uppercase.

Ex:

```
str1 = "python"
print(str1.islower())
str2 = "pYthon"
print(str2.islower())
str3 = "python383"
print(str3.islower())
str4 = "python 383"
print(str4.islower())
```

Op:

True.
false.
true
true.

Note: This Method checks Only alphabets.

> isupper(): Used to check wether the given string contains all uppercase characters or not.

- > true : all uppercase characters.
  false : if not in uppercase

Ex

```
Str1 = "PYTHON"
Str2 = 'PYTHON@123'
print("str1.isuppercase())
print("str2.isuppercase())
```

Op:

True
True.

* To check only subpart of the string

1) str = 'python is PROGRAM'

ls = str.split()

print (ls[2].isupper())

o/p: True

(or)

2) By Using Range Slice Operator [[:]]

Str = 'python is program"

Substr = str[0:6]

print (substr)

print (substr.islower())

o/p

python

false

~~GIST~~ 31/dec

Datatypes in python

(i)

* Numbers
  - int
  - float
  - complex

* string

Standard datatypes

Datatypes in python

31 Aug

* standard datatypes
* Advanced datatypes

(i) Standard datatypes :

* Numbers
  - int
  - float
  - complex.
* string.

(ii) Advanced datatypes (or) collections (or) Sequences:

* List ([ ])
* tuple
* set
* dictionary

* List datatype (or) collection (or) Sequence

It is a collection of items (or) values of same type (or) different types (or) mixed types

Syntax: ls_var = [value1, value2, ...]

(All values in list is speated by comma, and Enclosed with squar brackets)

Examples:

```
lst1 = [ ]   # Empty list
print(lst1) ⟶ [ ]
```

O/P

```
lst2 = [1,2,3,4,5]
print(lst2) ⟶ [1,2,3,4,5]
lst3 = [50, "ABC", 24.5]
print(lst3) ⟶ [50, 'ABC', 24.5]
```

> List Indexing:

Starts from 0 and goes to length-1

Ex: frts = ['banana', 'apple', 'berry']
0 1 2

* List supports negative indexing.

> negative indexing:

Starts from last element with index value -1

frts = ['banana', 'apple', 'berry']
-3 -2 -1

Ex. programs

0 1 2
frts = ['banana', 'apple', 'berry']
-3 -2 -1

```
print(frts[0])    ⟶ banana
print(frts[1])    ⟶ apple
print(frts[0:2])  ⟶ ['banana', 'apple']
print(frts[-1])   ⟶ berry
print(frts[-3:-1]) ⟶ ['banana', 'apple']
```

> List Operators :

(i) Concatenation Operator (+) :-

Used to concatinate two ~~strings~~ or more lists.

(ii) repetition Operator (*) :-

Used to Combine Some List Multiple times.

(iii) slice Operator [] :- Used to access the particular ~~string~~ element from the given list.

(iv) range slice Operator ([:]) :-

Used to access the range of Operators from the given list.

membership Operators :

(v) in ~~membership~~

returns true if the given is present in the given list

(vi) not in

returns true if the given element is not present in the given list

Ex:

```
ls1 = [1,2,3,4,5]
ls2 = ['c', 'java', 'python']
# concatination Operator
print (ls1 + ls2)
```

op

[1, 2, 3, 4, 5, 'c', 'java', 'python']

```
# repitition Operator
print (ls1 * 2)
```

- [1, 2, 3, 4, 5, 1, 2, 3, 4, 5]

```
# slice Operator
print (ls2 [2])
```

- python.

```
# range slice Operator
print (ls1 [2:4])
```

- [3, 4]

```
# in Operator
print ("cpp" in ls2)
```

- False

```
print ("c" in ls2)
```

- True.

```
# not in
print ("cpp" not in ls2)
```

- True

```
print ("c" not in ls2)
```

- false.

```
print (2 not in ls1)
```

- false.