Python is a general-purpose interpreted, interactive, object-oriented, and high-level programming language. It was created by Guido van Rossum during 1985- 1990. Like Perl, Python source code is also available under the GNU General Public License (GPL).

Python is a high-level, interpreted, interactive and object-oriented scripting language. Python is designed to be highly readable. It uses English keywords frequently where as other languages use punctuation, and it has fewer syntactical constructions than other languages.

- **Python is Interpreted:** Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is similar to PERL and PHP.

- **Python is Interactive:** You can actually sit at a Python prompt and interact with the interpreter directly to write your programs.

- **Python is Object-Oriented:** Python supports Object-Oriented style or technique of programming that encapsulates code within objects.

- **Python is a Beginner's Language:** Python is a great language for the beginner-level programmers and supports the development of a wide range of applications from simple text processing to WWW browsers to games.

Reserved Words

- The following list shows the Python keywords. These are reserved words and you cannot use them as constant or variable or any other identifier names. All the Python keywords contain lowercase letters only.

and	exec	not
assert	finally	or
break	for	pass
class	from	print

continue	global	raise
def	if	return
del	import	try
elif	in	while
else	is	with
except	lambda	yield

Standard Data Types

The data stored in memory can be of many types. For example, a person's age is stored as a numeric value and his or her address is stored as alphanumeric characters. Python has various standard data types that are used to define the operations possible on them and the storage method for each of them.

Python has five standard data types –

- Numbers
- String
- List
- Tuple
- Dictionary

Python Numbers

Number data types store numeric values. Number objects are created when you assign a value to them. For example –

```
var1 = 1
var2 = 10
```

You can also delete the reference to a number object by using the del statement. The syntax of the del statement is –

```
del var1[,var2[,var3[....,varN]]]
```

You can delete a single object or multiple objects by using the del statement. For example –

```
del var
del var_a, var_b
```

Python supports four different numerical types –

- int (signed integers)
- long (long integers, they can also be represented in octal and hexadecimal)
- float (floating point real values)
- complex (complex numbers)

Examples

Here are some examples of numbers –

int	long	float	complex
10	51924361L	0.0	3.14j
100	-0x19323L	15.20	45.j
-786	0122L	-21.9	9.322e-36j
080	0xDEFABCECBDAECBFBAEl	32.3+e18	.876j

-0490	535633629843L	-90.	-.6545+0J
-0x260	-052318172735L	-32.54e100	3e+26J
0x69	-4721885298529L	70.2-E12	4.53e-7j

- Python allows you to use a lowercase l with long, but it is recommended that you use only an uppercase L to avoid confusion with the number 1. Python displays long integers with an uppercase L.
- A complex number consists of an ordered pair of real floating-point numbers denoted by x + yj, where x and y are the real numbers and j is the imaginary unit.

Python Strings

Strings in Python are identified as a contiguous set of characters represented in the quotation marks. Python allows for either pairs of single or double quotes.

Python Lists

Lists are the most versatile of Python's compound data types. A list contains items separated by commas and enclosed within square brackets ([]). To some extent, lists are similar to arrays in C. One difference between them is that all the items belonging to a list can be of different data type.

The values stored in a list can be accessed using the slice operator ([ ] and [:]) with indexes starting at 0 in the beginning of the list and working their way to end -1. The plus (+) sign is the list concatenation operator, and the asterisk (*) is the repetition operator. For example –

Tuples:

- Tuple is another form of collection where different type of data can be stored.

- It is similar to list where data is separated by commas. Only the difference is that list uses square bracket and tuple uses parenthesis.
- Tuples are enclosed in parenthesis and cannot be changed.

Eg:

```
>>> tuple=('rahul',100,60.4,'deepak')
>>> tuple1=('sanjay',10)
>>> tuple
('rahul', 100, 60.4, 'deepak')
>>> tuple[2:]
(60.4, 'deepak')
>>> tuple1[0]
'sanjay'
>>> tuple+tuple1
('rahul', 100, 60.4, 'deepak', 'sanjay', 10)
>>>
```

Dictionary:

- **Dictionary is a collection which works on a key-value pair.**
- **It works like an associated array where no two keys can be same.**
- **Dictionaries are enclosed by curly braces ({}) and values can be retrieved by square bracket([]).**

Eg:

```
>>> dictionary={'name':'charlie','id':100,'dept':'it'}
>>> dictionary
{'dept': 'it', 'name': 'charlie', 'id': 100}
>>> dictionary.keys()
['dept', 'name', 'id']
>>> dictionary.values()
['it', 'charlie', 100]
```

Python Variables

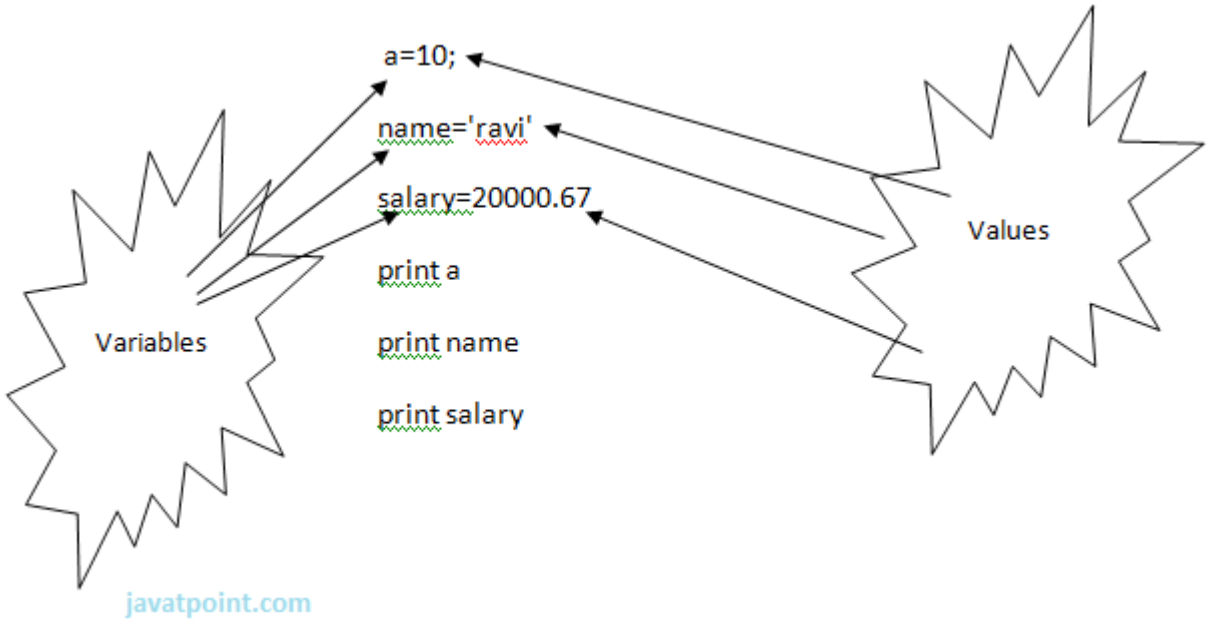
Variable is a name of the memory location where data is stored. Once a variable is stored that means a space is allocated in memory.

Assigning values to Variable:

We need not to declare explicitly variable in Python. When we assign any value to the variable that variable is declared automatically.

The assignment is done using the equal (=) operator.

Eg:



Output:

```
>>>
10
ravi
20000.67
>>>
```

Multiple Assignment:

Multiple assignment can be done in Python at a time.

There are two ways to assign values in Python:

1. Assigning single value to multiple variables:

Eg:

```
x=y=z=50
print x
print y
print z
```

Output:

```
>>>
50
50
50
>>>
```

2.Assigning multiple values to multiple variables:

Eg:

```
a,b,c=5,10,15
print a
print b
print c
```

Output:

```
>>>
5
10
15
>>>
```

The values will be assigned in the order in which variables appears.

Eg

```
x = y = z = 1
```

Now check the individual value in Python Shell.

view plaincopy to clipboardprint?

```
>>> x = y = z = 1
>>> print(x)
1
```

```
>>> print(y)
1
>>> print(z)
1
>>>
```

Here is an assignment statement where the variables assign many values at the same time.

Example :

```
x, y, z = 1, 2, "abcd"
```

In the above example x, y and z simultaneously get the new values 1, 2 and "abcd".


```
>>> x,y,z = 1,2,"abcd"
>>> print(x)
1
>>> print(y)
2
>>> print(z)
abcd
```

You can reuse variable names by simply assigning a new value to them :


```
>>> x = 100
>>> print(x)
100
>>> x = "Python"
>>> print(x)
Python
```

```
>>>
```

Swap variables

Python swap values in a single line and this applies to all objects in python.

Syntax

```
var1, var2 = var2, var1
```

Example :

view plaincopy to clipboardprint?

```
>>> x = 10
>>> y = 20
>>> print(x)
10
>>> print(y)
20
>>> x, y = y, x
>>> print(x)
20
>>> print(y)
10
>>>
```

Local and global variables in Python

In Python, variables that are only referenced inside a function are implicitly global. If a variable is assigned a value anywhere within the function's body, it's assumed to be a local unless explicitly declared as global.

Example :

view plaincopy to clipboardprint?

```
var1 = "Python"
def func1():
    var1 = "PHP"
    print("In side func1() var1 = ",var1)

def func2():
    print("In side func2() var1 = ",var1)
func1()
func2()
```

Output :

```
In side func1() var1 =  PHP
In side func2() var1 =  Python
```

You can use a global variable in other functions by declaring it as global keyword :

Example :

view plaincopy to clipboardprint?

```
def func1():
    global var1
    var1 = "PHP"
    print("In side func1() var1 = ",var1)

def func2():
    print("In side func2() var1 = ",var1)
func1()
func2()
```

Output :

```
In side func1() var1 =  PHP

In side func2() var1 =  PHP
```

Basic Fundamentals:

This section contains the basic fundamentals of Python like :

i)Tokens and their types.

ii) Comments

a)Tokens:

- Tokens can be defined as a punctuator mark, reserved words and each individual word in a statement.
- Token is the smallest unit inside the given program.

There are following tokens in Python:

- Keywords.
- Identifiers.
- Literals.
- Operators.

Python Keywords

Keywords are special reserved words which convey a special meaning to the compiler/interpreter. Each keyword have a special meaning and a specific operation. List of Keywords used in Python are:

True	False	None	and	as
asset	def	class	continue	break
else	finally	elif	del	except
global	for	if	from	import
raise	try	or	return	pass

nonlocal	in	not	is	lambda

Identifiers

Identifiers are the names given to the fundamental building blocks in a program.

These can be variables ,class ,object ,functions , lists , dictionaries etc.

There are certain rules defined for naming i.e., Identifiers.

I. An identifier is a long sequence of characters and numbers.

II.No special character except underscore (_) can be used as an identifier.

III.Keyword should not be used as an identifier name.

IV.Python is case sensitive. So using case is significant.

V.First character of an identifier can be character, underscore (_) but not digit.

Python Literals

Literals can be defined as a data that is given in a variable or constant.

Python support the following literals:

I. String literals:

String literals can be formed by enclosing a text in the quotes. We can use both single as well as double quotes for a String.

Eg:

"Aman" , '12345'

Types of Strings:

There are two types of Strings supported in Python:

a).Single line String- Strings that are terminated within a single line are known as Single line Strings.

Eg:

```
>>> text1='hello'
```

b).Multi line String- A piece of text that is spread along multiple lines is known as Multiple line String.

There are two ways to create Multiline Strings:

1). Adding black slash at the end of each line.

Eg:

```
>>> text1='hello\
user'
>>> text1
'hellouser'
>>>
```

2).Using triple quotation marks:-

Eg:

```
>>> str2='''welcome
to
SSSIT'''
>>> print str2
welcome
to
SSSIT
>>>
```

II.Numeric literals:

Numeric Literals are immutable. Numeric literals can belong to following four different numerical types.

Int(signed integers)	Long(long integers)	float(floating point)	Complex(complex)
Numbers(can be both positive and negative) with no fractional part.eg: 100	Integers of unlimited size followed by lowercase or uppercase L eg: 87032845L	Real numbers with both integer and fractional part eg: -26.2	In the form of a+bj where the real part and b for imaginary part of complex eg: 3.14j

III. Boolean literals:

A Boolean literal can have any of the two values: True or False.

IV. Special literals.

Python contains one special literal i.e., None.

None is used to specify to that field that is not created. It is also used for end of lists in Python.

Eg:

```
>>> val1=10
>>> val2=None
>>> val1
10
>>> val2
>>> print val2
None
>>>
```

V.Literal Collections.

Collections such as tuples, lists and Dictionary are used in Python.

List:

- List contain items of different data types. Lists are mutable i.e., modifiable.
- The values stored in List are separated by commas(,) and enclosed within a square brackets([]). We can store different type of data in a List.
- Value stored in a List can be retrieved using the slice operator([] and [:]).
- The plus sign (+) is the list concatenation and asterisk(*) is the repetition operator.

Eg:

```
>>> list=['aman',678,20.4,'saurav']
>>> list1=[456,'rahul']
>>> list
['aman', 678, 20.4, 'saurav']
>>> list[1:3]
[678, 20.4]
>>> list+list1
```

```
['aman', 678, 20.4, 'saurav', 456, 'rahul']
>>> list1*2
[456, 'rahul', 456, 'rahul']
>>>
```

Python Operators

Operators are particular symbols which operate on some values and produce an output.

The values are known as Operands.

Eg:

```
4 + 5 = 9
```

Here 4 and 5 are Operands and (+) , (=) signs are the operators. They produce the output 9.

Python supports the following operators:

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

Arithmetic Operators:

Operators	Description
//	Perform Floor division(gives integer value after division)
+	To perform addition

-	To perform subtraction
*	To perform multiplication
/	To perform division
%	To return remainder after division(Modulus)
**	Perform exponent(raise to power)

eg:

```
>>> 10+20
30
>>> 20-10
10
>>> 10*2
20
>>> 10/2
5
>>> 10%3
1
>>> 2**3
8
>>> 10//3
3
>>>
```

Relational Operators:

Operators	Description
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
==	Equal to

!=	Not equal to
<>	Not equal to(similar to !=)

eg:

```
>>> 10<20
True
>>> 10>20
False
>>> 10<=10
True
>>> 20>=15
True
>>> 5==6
False
>>> 5!=6
True
>>> 10<>2
True
>>>
```

Assignment Operators:

Operators	Description
=	Assignment
/=	Divide and Assign
+=	Add and assign
-=	Subtract and Assign
*=	Multiply and assign
%=	Modulus and assign
**=	Exponent and assign

//=	Floor division and assign

eg:

```
>>> c=10
>>> c
10
>>> c+=5
>>> c
15
>>> c-=5
>>> c
10
>>> c*=2
>>> c
20
>>> c/=2
>>> c
10
>>> c%=3
>>> c
1
>>> c=5
>>> c**=2
>>> c
25
>>> c//=2
>>> c
12
>>>
```

Logical Operators:

Operators	Description
and	Logical AND(When both conditions are true output will be true)
or	Logical OR (If any one condition is true output will be true)

not	Logical NOT(Compliment the condition i.e., reverse)

eg:

```
a=5>4 and 3>2
print a
b=5>4 or 3<2
print b
c=not(5>4)
print c
```

Output:

```
>>>
True
True
False
>>>
```

Membership Operators:

Operators	Description
in	Returns true if a variable is in sequence of another variable, else false.
not in	Returns true if a variable is not in sequence of another variable, else false.

eg:

```
a=10
b=20
list=[10,20,30,40,50];
if (a in list):
   print "a is in given list"
else:
   print "a is not in given list"
if(b not in list):
   print "b is not given in list"
else:
   print "b is given in list"
```

Output:

```
>>>
a is in given list
b is given in list
>>>
```

Identity Operators:

Operators	Description
is	Returns true if identity of two operands are same, else false
is not	Returns true if identity of two operands are not same, else false.

Example:

```
a=20
b=20
if( a is b):
   print  ?a,b have same identity?
else:
   print ?a, b are different?
b=10
if( a is not b):
   print  ?a,b have different identity?
else:
   print ?a,b have same identity?
```

Output:

```
>>>
a,b have same identity
a,b have different identity
>>>
```

Python Comments

Python supports two types of comments:

1) Single lined comment:

In case user wants to specify a single line comment, then comment must start with ?#?

Eg:

```
# This is single line comment.
```

2) Multi lined Comment:

Multi lined comment can be given inside triple quotes.

eg:

```
''' This
   Is
   Multipline comment'''
```

eg:

```
#single line comment
print "Hello Python"
'''This is
multiline comment'''
```

Python If Statements

The if statement in python is same as c language which is used test a condition. If condition is true, statement of if block is executed otherwise it is skipped.

Syntax of python if statement:

```
if(condition):
  statements
```

Example of if statement in python

```
a=10
if a==10:
     print  "Hello User"
```

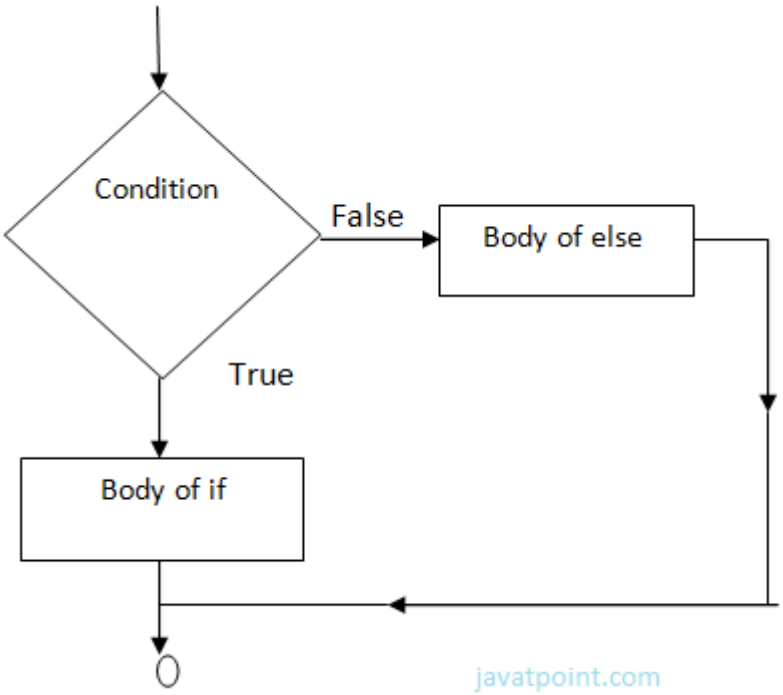
Output:

```
Hello User
```

Python If Else Statements

Syntax:

```
if(condition):  False
          statements
   else:   True
          statements
```



Example-

```
year=2000
if year%4==0:
    print  "Year is Leap"
else:
    print "Year is not Leap"
```

Output:

```
Year is Leap
```

Nested If Else Statement:

When we need to check for multiple conditions to be true then we use elif Statement.

This statement is like executing a if statement inside a else statement.

Syntax:

```
If statement:
    Body
elif statement:
    Body
else:
```

6. Body

Example:

```
a=10
if a>=20:
   print "Condition is True"
else:
   if a>=15:
      print "Checking second value"
   else:
      print "All Conditions are false"
```

Output:

```
All Conditions are false.
```

For Loop

for Loop is used to iterate a variable over a sequence(i.e., list or string) in the order that they appear.

Syntax:

```
for <variable> in <sequence>:
```

Output:

```
1

7

9
```

Explanation:

- Firstly, the first value will be assigned in the variable.
- Secondly all the statements in the body of the loop are executed with the same value.
- Thirdly, once step second is completed then variable is assigned the next value in the sequence and step second is repeated.
- Finally, it continues till all the values in the sequence are assigned in the variable and processed.

Program to display table of Number:

```
num=2
for a in range (1,6):
   print  num * a
```

Output:

```
2

4

6

8

10
```

Program to find sum of Natural numbers from 1 to 10.

```
sum=0
for n in range(1,11):
   sum+=n
print sum
```

Output:

```
55
```

Nested Loops

Loops defined within another Loop is called Nested Loop.

When an outer loop contains an inner loop in its body it is called Nested Looping.

Syntax:

```
for  <expression>:
     for <expression>:
        Body
```

eg:

```
for i in range(1,6):
   for j in range (1,i+1):
      print i,
   print
```

Output:

```
>>>
```

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

Explanation:

For each value of Outer loop the whole inner loop is executed.

For each value of inner loop the Body is executed each time.

Program to print Pyramid:

```
for i in range (1,6):
    for j in range (5,i-1,-1):
        print "*",
    print
```

Output:

```
>>>
* * * * *
* * * *
* * *
* *
*
```

While Loop

while Loop is used to execute number of statements or body till the condition passed in while is true. Once the condition is false, the control will come out of the loop.

Syntax:

```
while <expression>:
        Body
```

Here, body will execute multiple times till the expression passed is true. The Body may be a single statement or multiple statement.

Eg:

```
a=10
while a>0:
    print "Value of a is",a
    a=a-2
```

```
print "Loop is Completed"
```

Output:

```
>>>
Value of a is 10
Value of a is 8
Value of a is 6
Value of a is 4
Value of a is 2
Loop is Completed
>>>
```

Explanation:

- Firstly, the value in the variable is initialized.
- Secondly, the condition/expression in the while is evaluated. Consequently if condition is true, the control enters in the body and executes all the statements . If the condition/expression passed results in false then the control exists the body and straight away control goes to next instruction after body of while.
- Thirdly, in case condition was true having completed all the statements, the variable is incremented or decremented. Having changed the value of variable step second is followed. This process continues till the expression/condition becomes false.
- Finally Rest of code after body is executed.

Program to add digits of a number:

```
n=153
sum=0
while n>0:
    r=n%10
    sum+=r
    n=n/10
print sum
```

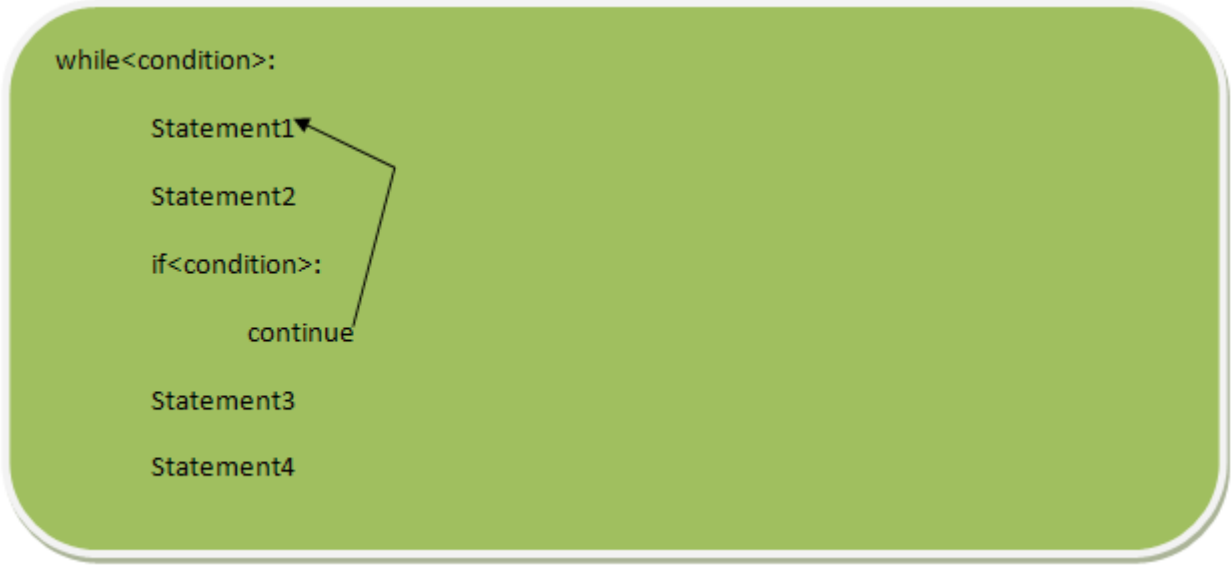
Output:

```
>>>
9
>>>
```

Python Break

break statement is a jump statement that is used to pass the control to the end of the loop.

When break statement is applied the control points to the line following the body of the loop , hence applying break statement makes the loop to terminate and controls goes to next line pointing after loop body.



eg:

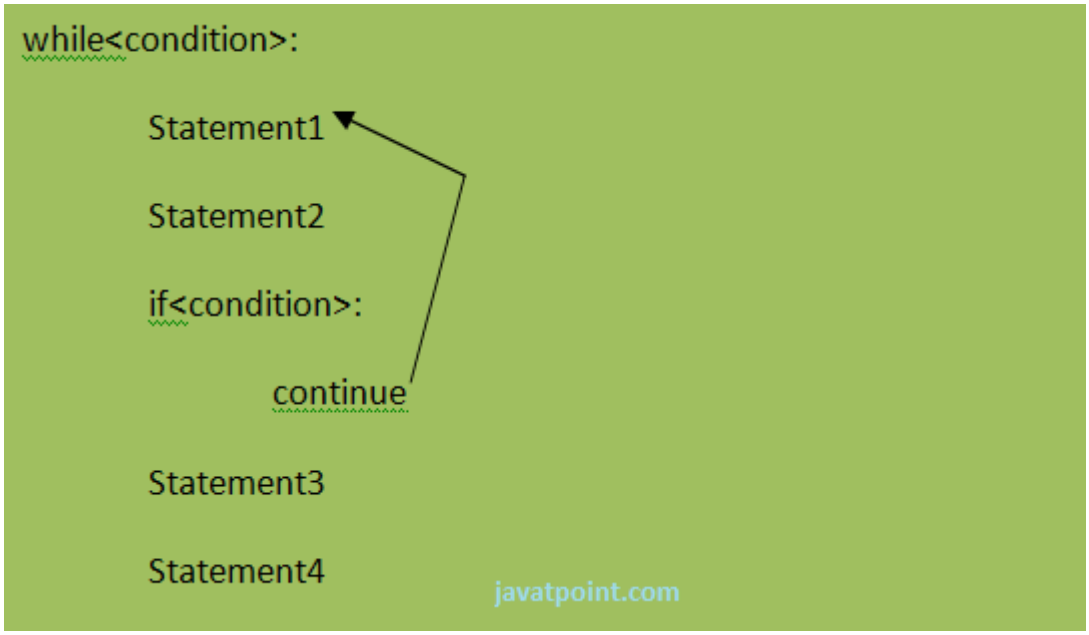
```
1.          for i in [1,2,3,4,5]:
2.              if i==4:
3.                  print "Element found"
4.                  break
5.              print i,
```

Output:

```
1.          >>>
2.          1 2 3 Element found
3.          >>>
```

Continue Statement

continue Statement is a jump statement that is used to skip the present iteration and forces next iteration of loop to take place. It can be used in while as well as for loop statements.



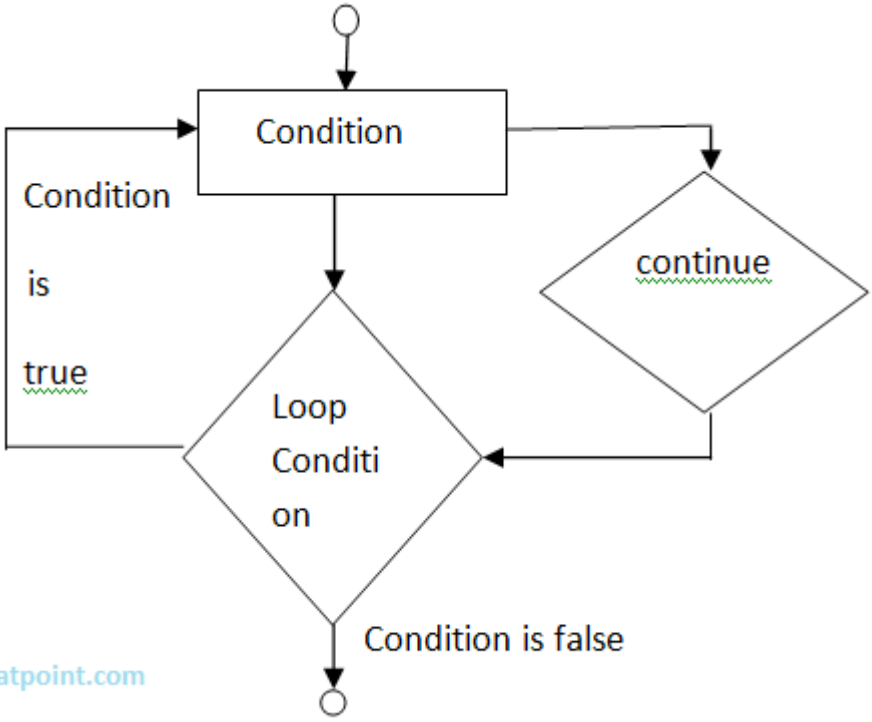
eg:

```
1.  a=0
2.  while a<=5:
3.      a=a+1
4.      if a%2==0:
5.          continue
6.      print a
7.  print "End of Loop"
```

Output:

```
1.  >>>
2.  1
3.  3
4.  5
5.  End of Loop
6.  >>>
```

Flow chart of continue:-



Python Pass

When you do not want any code to execute, pass Statement is used. It is same as the name refers to. It just makes the control to pass by without executing any code. If we want to bypass any code pass statement can be used.

Syntax:

```
pass
```

eg:

```
for i in [1,2,3,4,5]:
    if i==3:
        pass
        print "Pass when value is",i
    print i,
```

Output:

```
>>>
1 2 Pass when value is 3
3 4 5
>>>
```

PYTHON STRINGS

Strings are the simplest and easy to use in Python.

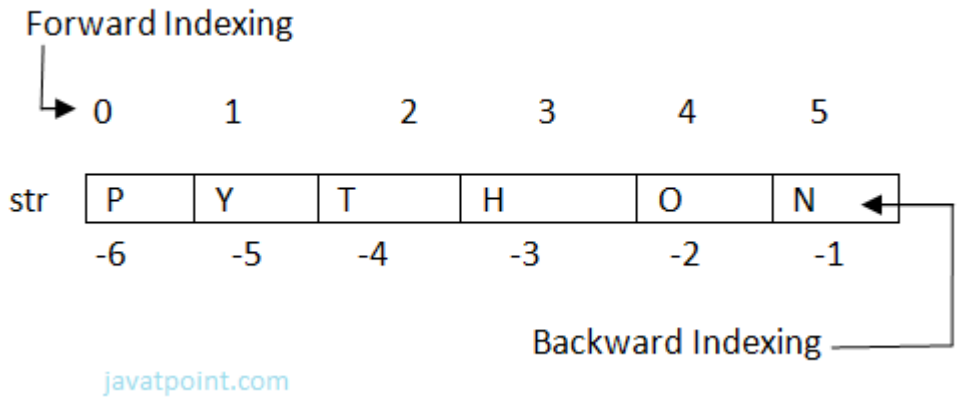
String pythons are immutable.

We can simply create Python String by enclosing a text in single as well as double quotes. Python treat both single and double quotes statements same.

Accessing Strings:

- In Python, Strings are stored as individual characters in a contiguous memory location.
- The benefit of using String is that it can be accessed from both the directions in forward and backward.
- Both forward as well as backward indexing are provided using Strings in Python.
  - Forward indexing starts with 0,1,2,3,....
  - Backward indexing starts with -1,-2,-3,-4,....

eg:



1. str[0]='P'=str[-6] , str[1]='Y' = str[-5] , str[2] = 'T' = str[-4] , str[3] = 'H' = str[-3]
2. str[4] = 'O' = str[-2] , str[5] = 'N' = str[-1].

Simple program to retrieve String in reverse as well as normal form.

```
1.  name="Rajat"
2.  length=len(name)
3.  i=0
4.  for n in range(-1,(-length-1),-1):
5.      print name[i],"\t",name[n]
```

6. i+=1

Output:

```
>>>
R       t
a       a
j       j
a       a
t       R
>>>
```

Strings Operators

There are basically 3 types of Operators supported by String:

1. Basic Operators.
2. Membership Operators.
3. Relational Operators.

Basic Operators:

There are two types of basic operators in String. They are "+" and "*".

String Concatenation Operator :(+)

The concatenation operator (+) concatenate two Strings and forms a new String.

eg:

```
>>> "ratan" + "jaiswal"
```

Output:

```
'ratanjaiswal'
>>>
```

Expression	Output
'10' + '20'	'1020'
"s" + "007"	's007'
'abcd123' + 'xyz4'	'abcd123xyz4'

NOTE: Both the operands passed for concatenation must be of same type, else it will show an error.

Eg:

```
'abc' + 3
>>>
```

output:

```
Traceback (most recent call last):
  File "", line 1, in
    'abc' + 3
TypeError: cannot concatenate 'str' and 'int' objects
>>>
```

Replication Operator: (*)

Replication operator uses two parameter for operation. One is the integer value and the other one is the String.

The Replication operator is used to repeat a string number of times. The string will be repeated the number of times which is given by the integer value.

Eg:

1. >>> 5*"Vimal"

Output:

```
'VimalVimalVimalVimalVimal'
```

Expression	Output
"soono"*2	'soonosoono'
3*'1'	'111'
'$'*5	'$$$$$'

NOTE: We can use Replication operator in any way i.e., int * string or string * int. Both the parameters passed cannot be of same type.

Membership Operators

Membership Operators are already discussed in the Operators section. Let see with context of String.

There are two types of Membership operators:

1) in:"in" operator return true if a character or the entire substring is present in the specified string, otherwise false.

2) not in:"not in" operator return true if a character or entire substring does not exist in the specified string, otherwise false.

Eg:

```
>>> str1="javatpoint"
>>> str2='sssit'
>>> str3="seomount"
>>> str4='java'
>>> st5="it"
>>> str6="seo"
>>> str4 in str1
True
>>> str5 in str2
>>> st5 in str2
True
>>> str6 in str3
True
>>> str4 not in str1
False
>>> str1 not in str4
True
```

Relational Operators:

All the comparison operators i.e., (<,><=,>=,==,!=,<>) are also applicable to strings. The Strings are compared based on the ASCII value or Unicode(i.e., dictionary Order).

Eg:

```
>>> "RAJAT"=="RAJAT"
True
>>> "afsha">='Afsha'
True
>>> "Z"<>"z"
True
```

Explanation:

The ASCII value of a is 97, b is 98, c is 99 and so on. The ASCII value of A is 65,B is 66,C is 67 and so on. The comparison between strings are done on the basis on ASCII value.

Slice Notation:

String slice can be defined as substring which is the part of string. Therefore further substring can be obtained from a string.

There can be many forms to slice a string. As string can be accessed or indexed from both the direction and hence string can also be sliced from both the direction that is left and right.

Syntax:

```
<string_name>[startIndex:endIndex],
<string_name>[:endIndex],
<string_name>[startIndex:]
```

Example:

```
>>> str="Nikhil"
>>> str[0:6]
'Nikhil'
>>> str[0:3]
'Nik'
>>> str[2:5]
'khi'
>>> str[:6]
'Nikhil'
>>> str[3:]
'hil'
```

Note: startIndex in String slice is inclusive whereas endIndex is exclusive.

String slice can also be used with Concatenation operator to get whole string.

Eg:

```
>>> str="Mahesh"
>>> str[:6]+str[6:]
'Mahesh'
```

//here 6 is the length of the string.

String Functions and Methods:

There are many predefined or built in functions in String. They are as follows:

capitalize()	It capitalizes the first character of the String.

count(string,begin,end)	Counts number of times substring occurs in a String between be
endswith(suffix ,begin=0,end=n)	Returns a Boolean value if the string terminates with given suf end.
find(substring ,beginIndex, endIndex)	It returns the index value of the string where substring is foun and end index.
index(subsring, beginIndex, endIndex)	Same as find() except it raises an exception if string is not foun
isalnum()	It returns True if characters in the string are alphanumeric i.e., and there is at least 1 character. Otherwise it returns False.
isalpha()	It returns True when all the characters are alphabets and character, otherwise False.
isdigit()	It returns True if all the characters are digit and there is a otherwise False.
islower()	It returns True if the characters of a string are in lower case, oth
isupper()	It returns False if characters of a string are in Upper case, other
isspace()	It returns True if the characters of a string are whitespace, othe
len(string)	len() returns the length of a string.
lower()	Converts all the characters of a string to Lower case.
upper()	Converts all the characters of a string to Upper Case.
startswith(str ,begin=0,end=n)	Returns a Boolean value if the string starts with given str betwe
swapcase()	Inverts case of all characters in a string.
lstrip()	Remove all leading whitespace of a string. It can also be use

	character from leading.
rstrip()	Remove all trailing whitespace of a string. It can also be use character from trailing.

Examples:

1) capitalize()

```
>>> 'abc'.capitalize()
```

Output:

```
'Abc'
```

2) count(string)

```
msg = "welcome to sssit";
substr1 = "o";
print  msg.count(substr1, 4, 16)
substr2 = "t";
print  msg.count(substr2)
```

Output:

```
>>>
2
2
>>>
```

3) endswith(string)

```
string1="Welcome to SSSIT";
substring1="SSSIT";
substring2="to";
substring3="of";
print string1.endswith(substring1);
print string1.endswith(substring2,2,16);
print string1.endswith(substring3,2,19);
print string1.endswith(substring3);
```

Output:

```
>>>
True
False
False
False
>>>
```

4) find(string)

```
str="Welcome to SSSIT";
```

```
2.          substr1="come";
3.          substr2="to";
4.          print str.find(substr1);
5.          print str.find(substr2);
6.          print str.find(substr1,3,10);
7.          print str.find(substr2,19);
```

Output:

```
>>>
3
8
3
-1
>>>
```

5) index(string)

```
1.          str="Welcome to world of SSSIT";
2.          substr1="come";
3.          substr2="of";
4.          print str.index(substr1);
5.          print str.index(substr2);
6.          print str.index(substr1,3,10);
7.          print str.index(substr2,19);
```

Output:

```
>>>
3
17
3
Traceback (most recent call last):
  File "C:/Python27/fin.py", line 7, in
    print str.index(substr2,19);
ValueError: substring not found
>>>
```

6) isalnum()

```
1.          str="Welcome to sssit";
2.             print str.isalnum();
3.          str1="Python47";
4.          print str1.isalnum();
```

Output:

```
>>>
False
True
>>>
```

7) isalpha()

```
1.          string1="HelloPython";    # Even space is not allowed
```

```
print string1.isalpha();
string2="This is Python2.7.4"
print string2.isalpha();
```

Output:

```
>>>
True
False
>>>
```

8) isdigit()

```
string1="HelloPython";
print string1.isdigit();
string2="98564738"
print string2.isdigit();
```

Output:

```
>>>
False
True
>>>
```

9) islower()

```
string1="Hello Python";
print string1.islower();
string2="welcome to "
print string2.islower();
```

Output:

```
>>>
False
True
>>>
```

10) isupper()

```
string1="Hello Python";
print string1.isupper();
string2="WELCOME TO"
print string2.isupper();
```

Output:

```
>>>
False
```

```
True
>>>
```

11) isspace()

```
1.  string1="    ";
2.  print string1.isspace();
3.  string2="WELCOME TO WORLD OF PYT"
4.  print string2.isspace();
```

Output:

```
>>>
True
False
>>>
```

12) len(string)

```
1.  string1="    ";
2.  print len(string1);
3.  string2="WELCOME TO SSSIT"
4.  print len(string2);
```

Output:

```
>>>
4
16
>>>
```

13) lower()

```
1.  string1="Hello Python";
2.  print string1.lower();
3.  string2="WELCOME TO SSSIT"
4.  print string2.lower();
```

Output:

```
>>>
hello python
welcome to sssit
>>>
```

14) upper()

```
1.  string1="Hello Python";
2.  print string1.upper();
3.  string2="welcome to SSSIT"
4.  print string2.upper();
```

Output:

```
>>>
HELLO PYTHON
WELCOME TO SSSIT
>>>
```

15) startswith(string)

```
1.  string1="Hello Python";
2.  print string1.startswith('Hello');
3.  string2="welcome to SSSIT"
4.  print string2.startswith('come',3,7);
```

Output:

```
>>>
True
True
>>>
```

16) swapcase()

```
1.  string1="Hello Python";
2.  print string1.swapcase();
3.  string2="welcome to SSSIT"
4.  print string2.swapcase();
```

Output:

```
>>>
hELLO pYTHON
WELCOME TO sssit
>>>
```

17) lstrip()

```
1.  string1="    Hello Python";
2.  print string1.lstrip();
3.  string2="@@@@@@@@@welcome to SSSIT"
4.  print string2.lstrip('@');
```

Output:

```
>>>
Hello Python
welcome to world to SSSIT
>>>
```

18) rstrip()

```
1.  string1="    Hello Python     ";
2.  print string1.rstrip();
3.  string2="@welcome to SSSIT!!!"
4.  print string2.rstrip('!');
```

Output:

```
>>>
          Hello Python
@welcome to SSSIT
>>>
```
