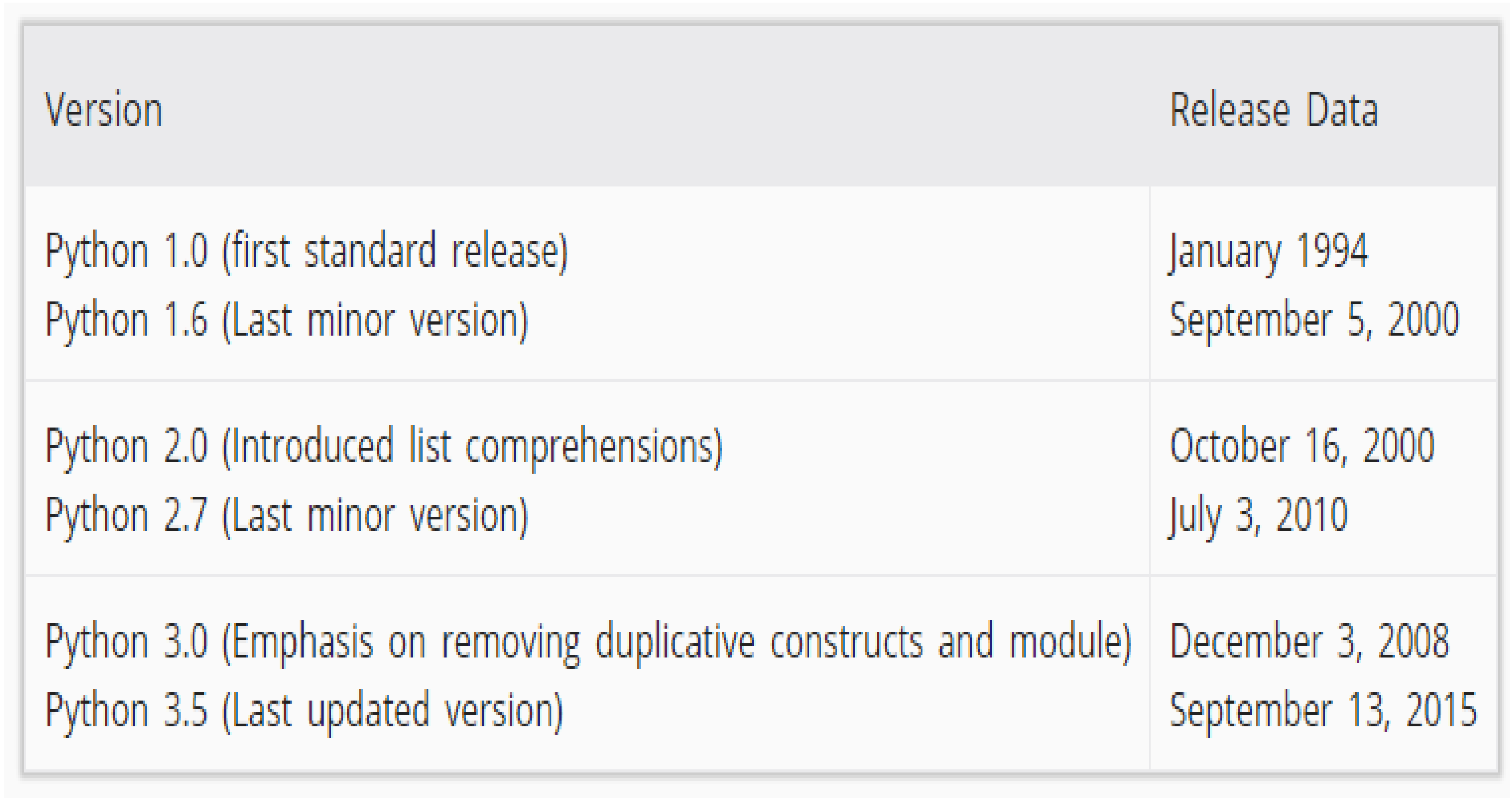
python

Python is a fairly old language created by **Guido Van Rossum**. The design began in the late 1980s and was first released in February 1991

It wasn't named after a dangerous snake.

Rossum was fan of a comedy series from late seventies. The name "Python" was adopted from the same series "**Monty Python's Flying Circus**".

Release Dates of Different Versions



Version	Release Data
Python 1.0 (first standard release) Python 1.6 (Last minor version)	January 1994 September 5, 2000
Python 2.0 (Introduced list comprehensions) Python 2.7 (Last minor version)	October 16, 2000 July 3, 2010
Python 3.0 (Emphasis on removing duplicative constructs and module) Python 3.5 (Last updated version)	December 3, 2008 September 13, 2015

What is Python

- Python is widely use general purpose high level programming language.
- Python is an interpreted, object-oriented programming , that has gained popularity because of its clear syntax and readability.
- The source code is freely available and open for modification and reuse.

Python Overview

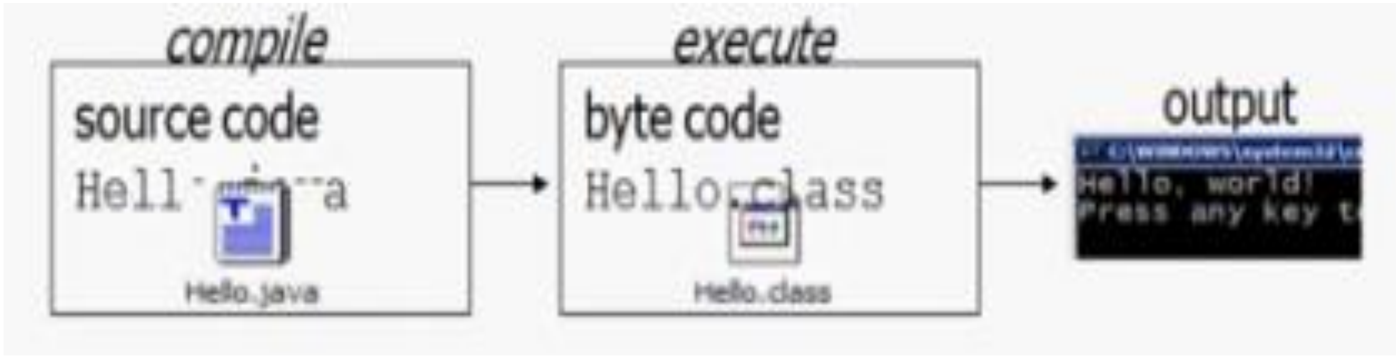
- Scripting Language
- Object Oriented
- Portable
- A simple language which is easier to learn
- Extensible and Embeddable
- Free and open-source
- A high-level, interpreted language

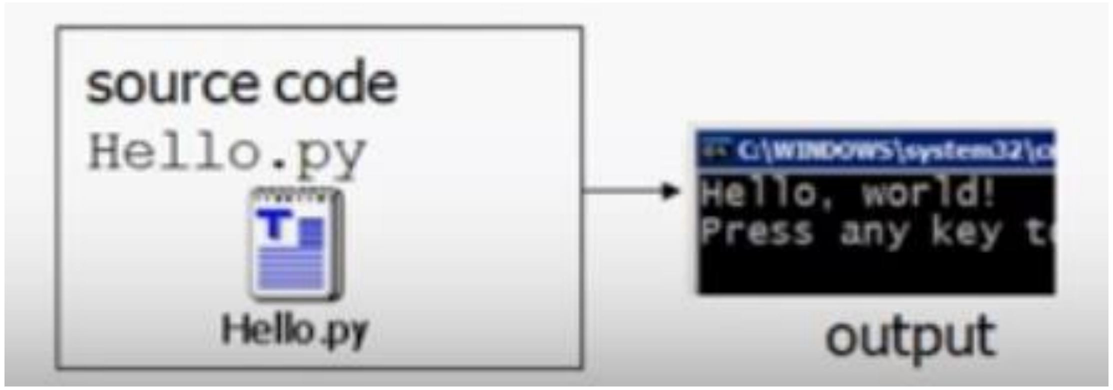
A simple language which is easier to learn
Python has a very simple and elegant syntax. It's much easier to read and write Python programs compared to other languages like: C++, Java, C#.

Extensible and Embeddable
You can easily combine pieces of C/C++ or other languages with Python code.This will give your application high performance as well as scripting capabilities which other languages may not provide out of the box.

A high-level, interpreted language
Unlike C/C++, you don't have to worry about daunting tasks like memory management, garbage collection and so on.
Likewise, when you run Python code, it automatically converts your code to the language your computer understands. You don't need to worry about any lower-level operations.

Compiling And Interpreting



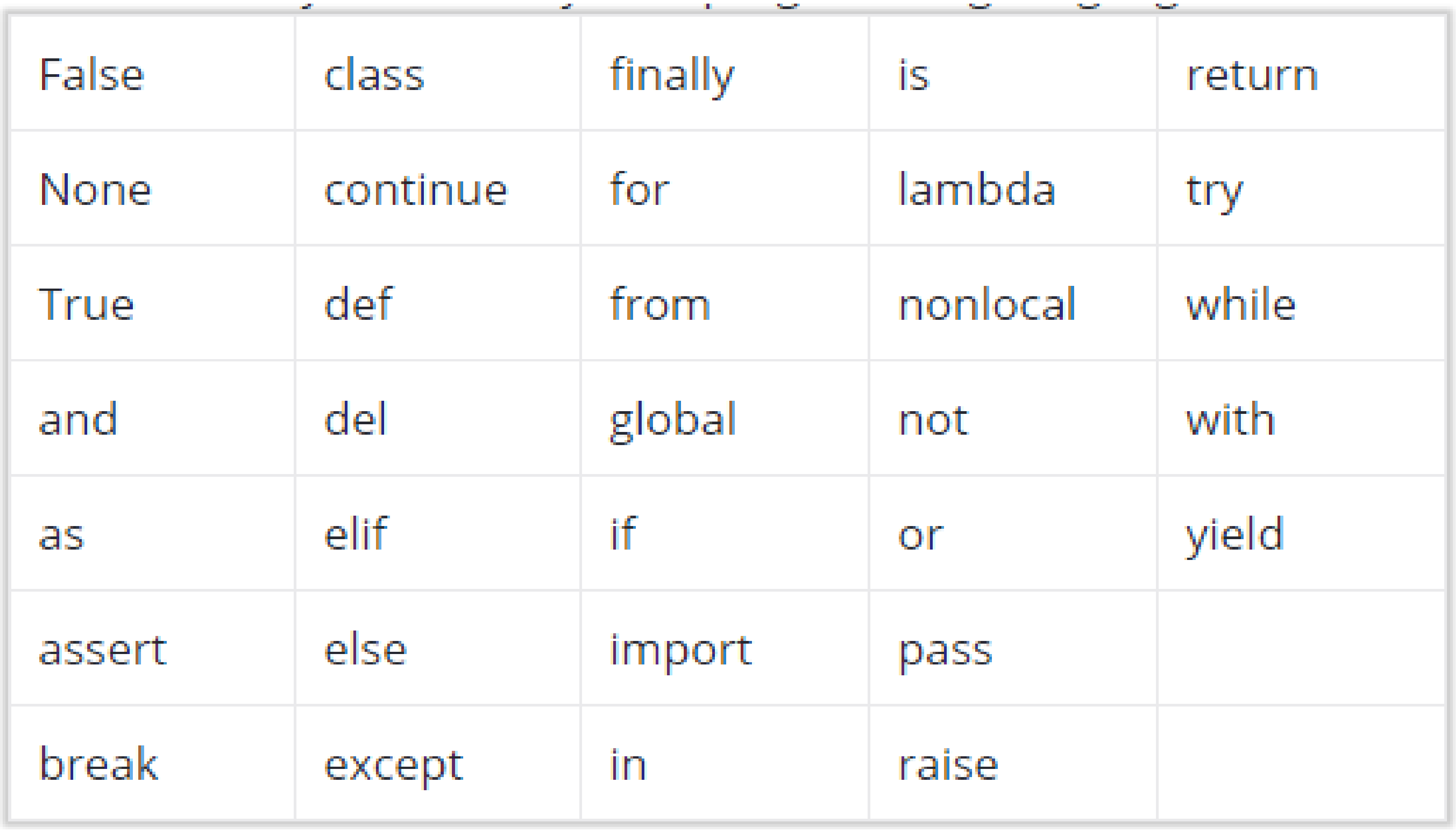


Application

- System Utilities
- GUI
- Internet Scripting
- Embedded Scripting
- Database Programming
- Data Mining
- Artificial Intelligence
- Image Processing

Python Keywords

There are 33 keywords in Python 3.3



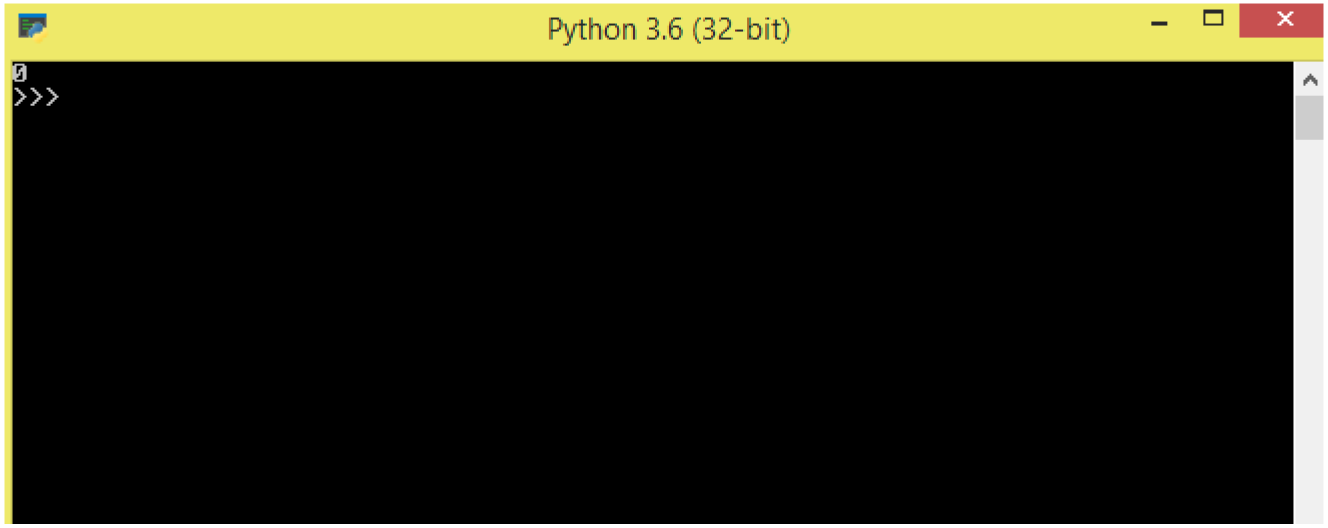
False	class	finally	is	return
None	continue	for	lambda	try
True	def	from	nonlocal	while
and	del	global	not	with
as	elif	if	or	yield
assert	else	import	pass	
break	except	in	raise	

How to check keyword In the python

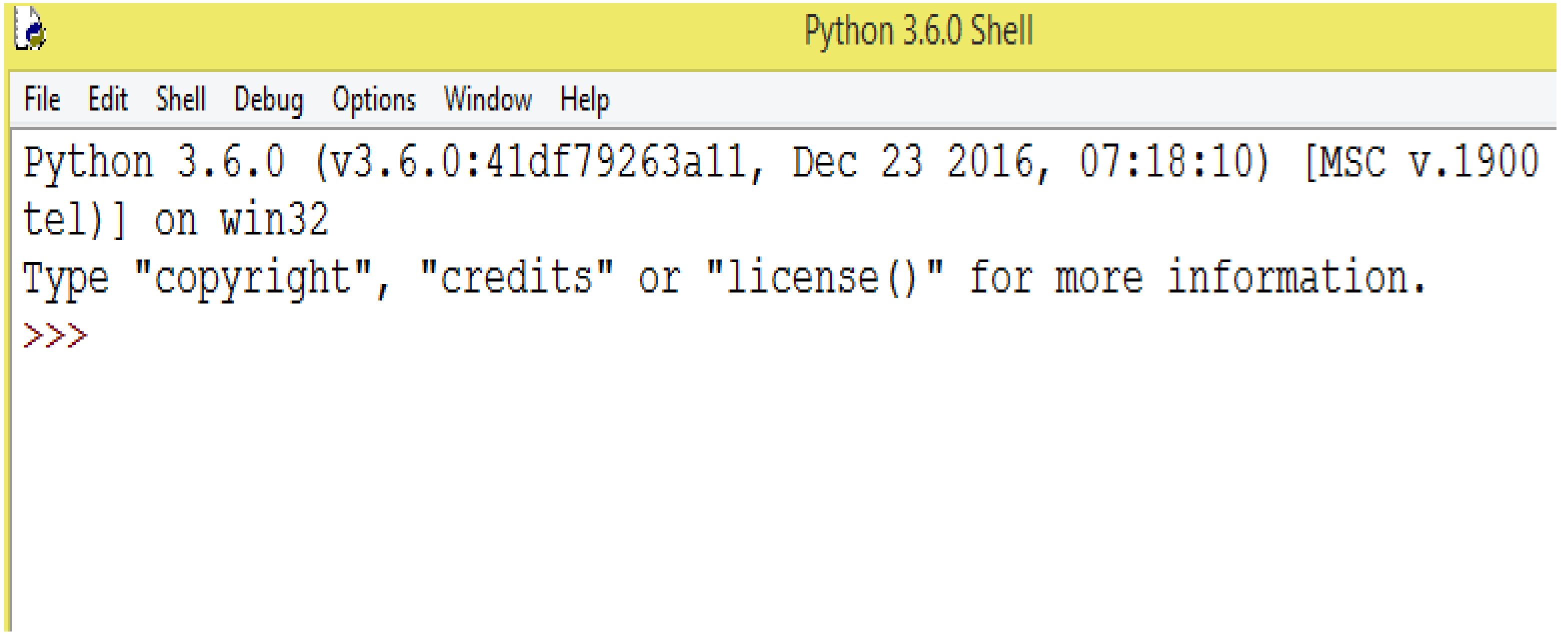
```
>>> import keyword
>>> print(keyword.kwlist)
```

Python Identifiers

- Identifiers can be a combination of letters in lowercase (a to z) or uppercase (A to Z) or digits (0 to 9) or an underscore (_).
- An identifier cannot start with a digit. 1variable is invalid, but variable1 is perfectly fine.
- Keywords cannot be used as identifiers.
- We cannot use special symbols like !, @, #, $, % etc. in our identifier.



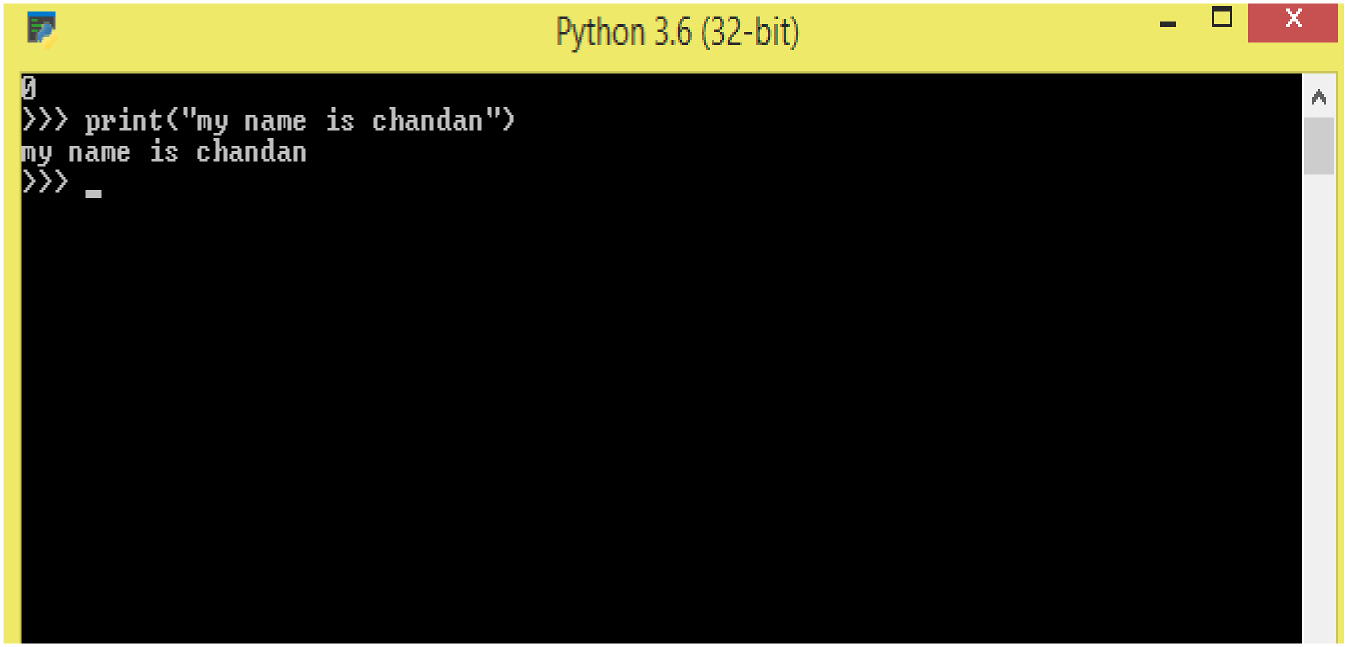
```
>>>
```



```
Python 3.6.0 (v3.6.0:41df79263a11, Dec 23 2016, 07:18:10) [MSC v.1900
tel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
```

First Python Program

```
print("my name is chandan")
```

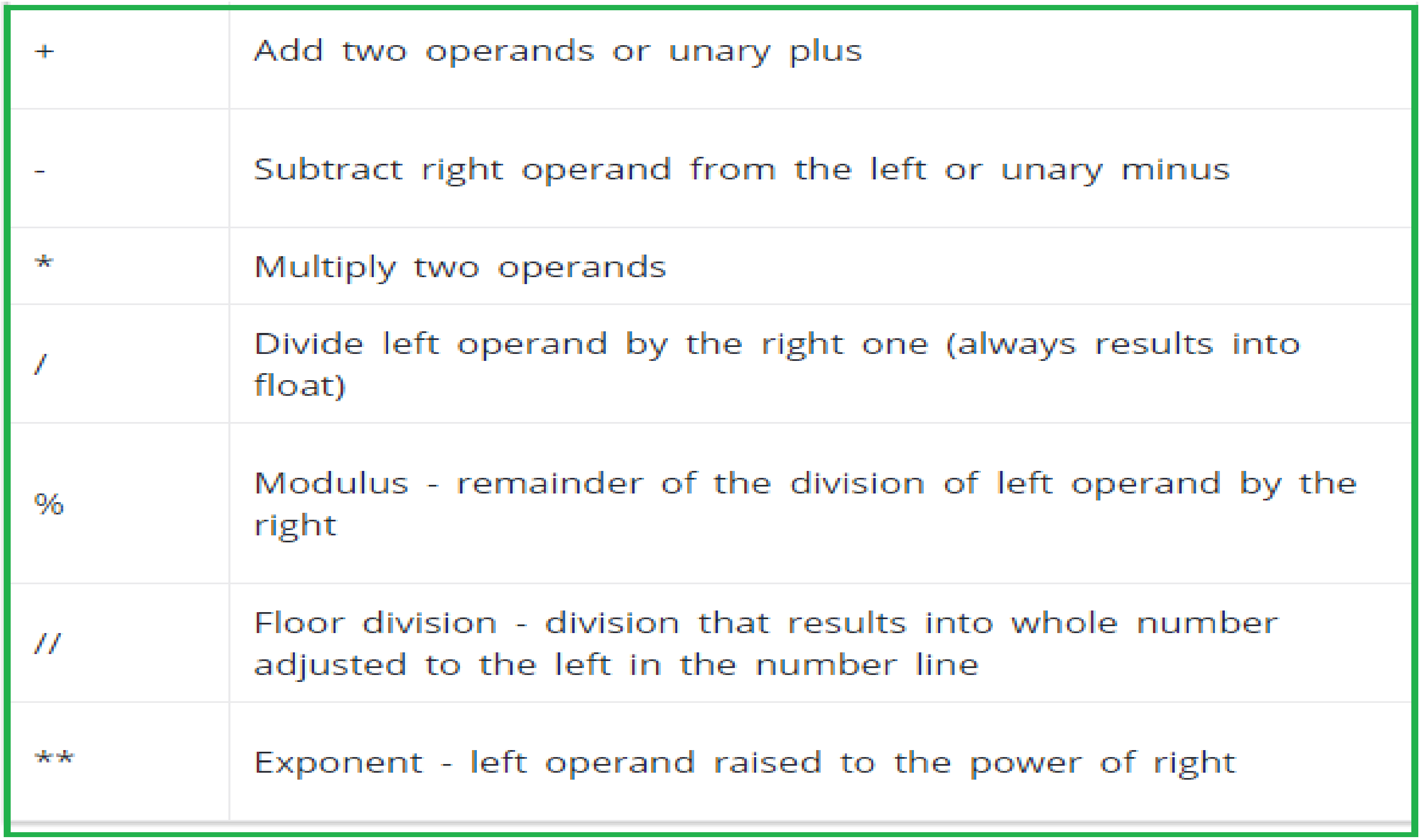


```
0
>>> print("my name is chandan")
my name is chandan
>>> _
```

Type of operators in Python

- ✓Arithmetic operators
- ✓Comparison (Relational) operators
- ✓Logical (Boolean) operators
- ✓Bitwise operators
- ✓Assignment operators
- ✓Special operators

Arithmetic operators



+	Add two operands or unary plus
-	Subtract right operand from the left or unary minus
*	Multiply two operands
/	Divide left operand by the right one (always results into float)
%	Modulus - remainder of the division of left operand by the right
//	Floor division - division that results into whole number adjusted to the left in the number line
**	Exponent - left operand raised to the power of right

```
>>> x=5
>>> y=6
>>> print("x+y",x+y)
```

```
>>> 3*4+5-6
```

PEMDAS

```
>>> 5%3
>>> 22/2
>>> 22/2.5
>>> 22//2.5
```

Python Variables

A variable is a location in memory used to store some data (value).

They are given unique names to differentiate between Different memory locations. The rules for writing a variable name is same as the rules for writing identifiers in Python.

We don't need to declare a variable before using it.

```
>>> a=5
>>> b=5.5
>>> c="chandan"
>>> print(a,b,c)
```

multiple statements in one line, separated by semicolon

```
a=5;b=3.5;c="chandan"
```

Multiple assignments

```
>>> a,b,c=5,5.5,"chandan“
>>>x=y=z=89
```

Clear

```
def cls():
    print('\n'*50)
```

```
>>> import os
>>> os.system('cls')
```

Multi-line statement

```
>>> a = 1 + 2 + 3 + \
    4 + 5 + 6 + \
    7 + 8 + 9
```

```
>>> a = (1 + 2 + 3 +
    4 + 5 + 6 +
    7 + 8 + 9)
```

```
>>> colors = ['red',
          'blue',
          'green']
```

Python Comments

In Python, we use the hash (#) symbol to start writing a comment.

```
#this is comments

"""This is also a perfect
 example of multi-line
 comments"""
```

Data types in Python

- Numbers
- Strings
- List
- Tuple
- Dictionary

Python Numbers

- Integer
- Long
- Float
- Complex

Integers, floating point numbers and complex numbers falls under Python numbers category. They are defined as **int, float** and **complex** class in Python

```
>>> a=1;b=4.5;c=5+6j
>>> type(a)
>>> type(b)
>>> type(c)
>>> a=1+2j
>>> print(a, "is complex number?",
isinstance(1+2j,complex))
```

print() function

We use the print() function to output data to the standard output device (screen).

print(*objects, sep=' ', end='\n', file=sys.stdout)

Here, **objects** is the value(s) to be printed.

The **sep** separator is used between the values. It defaults into a space character.

After all values are printed, **end** is printed. It defaults into a new line.

The **file** is the object where the values are printed and its default value is sys.stdout(screen)

```
>>> print(1,2,3,4)
1 2 3 4
>>> print(1,2,3,4,sep='*')
1*2*3*4
>>> print(1,2,3,4,sep='#',end='$')
1#2#3#4$
```

Output formatting

- **str.format()** This method is visible to any string object

```
>>> x=9;y=0
>>> print('the value of x is {} and the value of y is {}'.format(x,y))
```

```
 >>> x=9;y=0
>>> print('the value of x is {} and the value of y is{}'.format(y,x))
```

```
print('The value of x is {0} and y is {1}'.format('10','11'))
print('my name is {0} and my age is {1}'.format('chandan','28'))
```

```
print('my name is {name} and my age is
{age}'.format(name='chandan',age='28'))
```

```
>>> x = 12.3456789
>>> print("The value of x is %3.2f" %x)
>>>print('the value of x is %3.1f' %x)
>>> print("the value of x is %3.4f" %x)
```

Python Input

input([prompt])

where prompt is the string we wish to display on the screen. It is optional

```
>>> x=input('enter any string')
enter any string 12
>>> x
'12'
>>> int(x)
12
```

```
int('2+3')
eval('2+3')
```

Built-in Modules and Functions

```
>>> dir(__builtins__)
```

```
>>> abs(3+4j)
>>> len("chandan")
>>> max(3,4,2)
>>> min(3,4,2)
>>>bin(10)
>>>hex(15)
```

Python Import

```
>>> 2**3
>>> import math
>>> pow(2,3)
```

```
>>> import math
>>> print(math.pi)
3.141592653589793
```

```
>>> import sys
>>> sys.path
```

```
>>> import os
>>> os.system('cls')
```

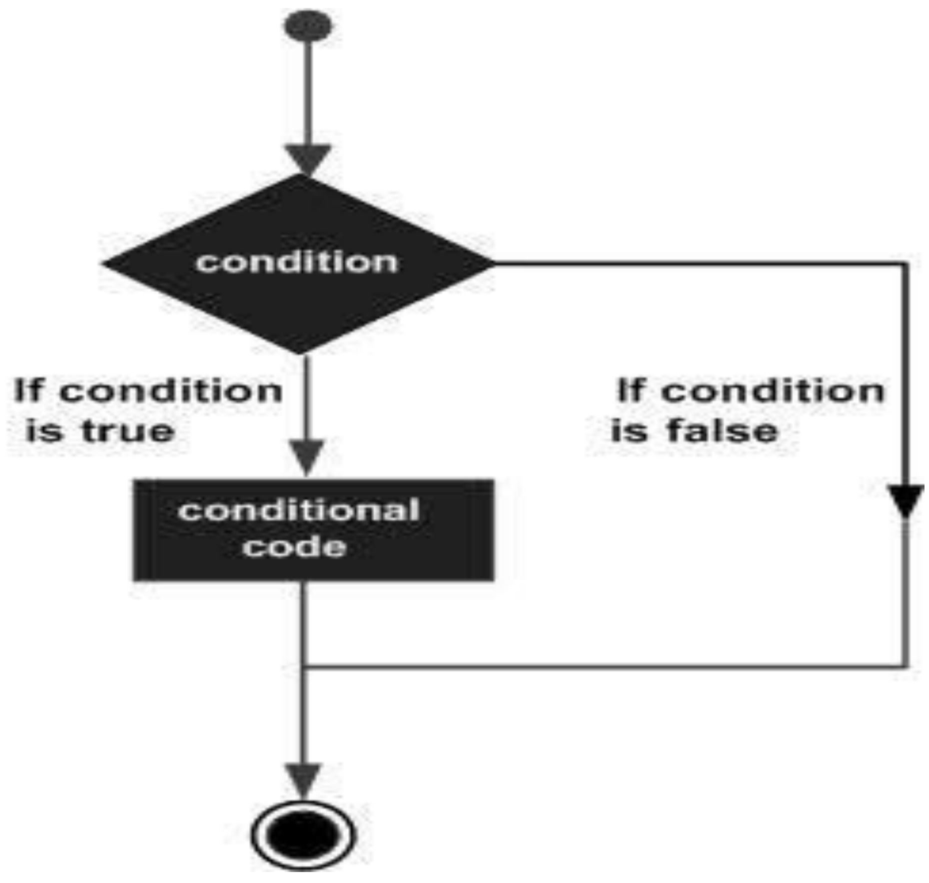
```
>>> import math
>>> math.sqrt(9)
>>> mysqrt=math.sqrt
>>> mysqrt(9)
```

Simple Example

- ✓ Add Two Numbers
- ✓ Add Two Numbers Provided by The Use
- ✓ Find The Maximum Number from Three Number
- ✓ Program to Solve Quadratic Equation
- ✓ Program to Swap Two Variables
- ✓ Python program to convert decimal number into binary, octal and hexadecimal number system

Python Decision Making

Decision making is required when we want to execute a code only if a certain condition is satisfied.

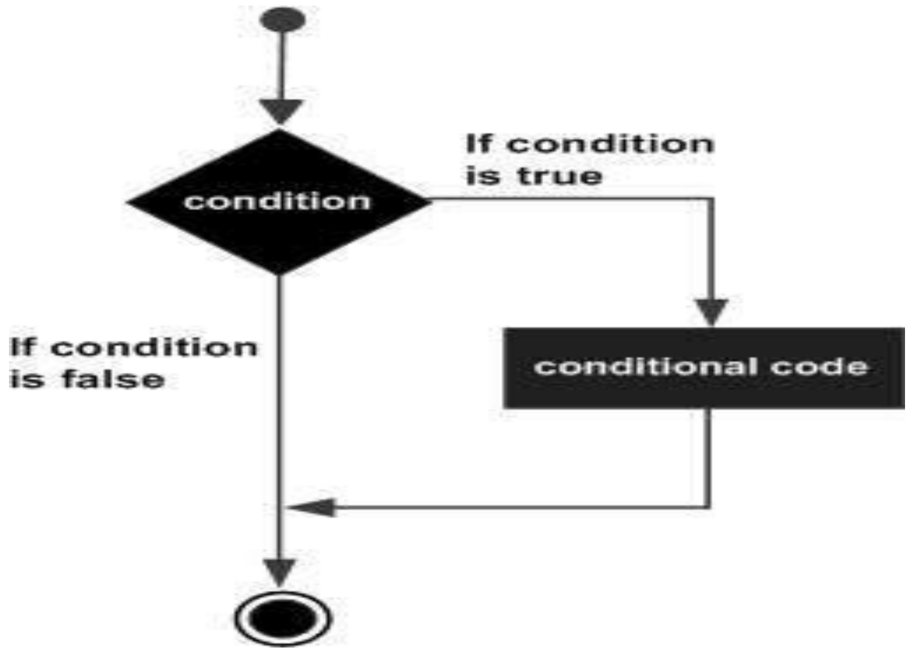


Python programming language assumes any **non-zero** and **non-null** values as TRUE, and if it is either **zero** or **null**, then it is assumed as FALSE value.

Statement	Description
if statements	An **if statement** consists of a boolean expression followed by one or more statements.
if...else statements	An **if statement** can be followed by an optional **else statement**, which executes when the boolean expression is FALSE.
nested if statements	You can use one **if** or **else if** statement inside another **if** or **else if** statement(s).

if Statement

An if statements generally consist of a Boolean expression (this results either in True or False.). An if statement can consist of one or more than one statements.



```
if expression:
   statement(s)
```

```
num = 3
if num > 0:
    print(num, "is a positive number.")
    print("This is always printed.")
```

```
x=1
if x==1:
    print("codotion is true")
print("this is if after if block")
```

```
x=1
if x==2:
    print("codotion is true")
print("this is if after if block")
```

Python if...else Statement

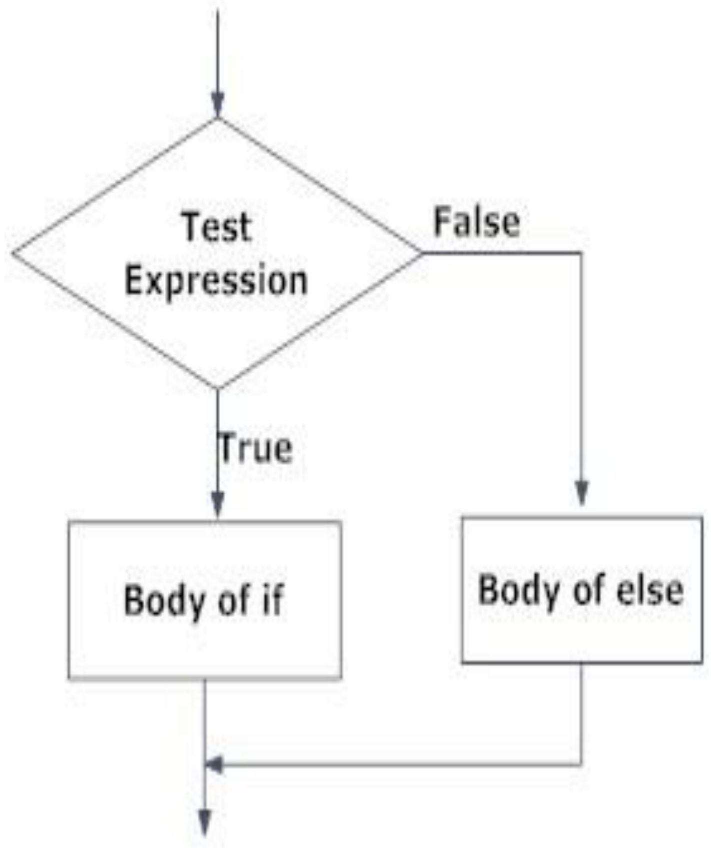


Fig: Operation of if...else statement

```
if test expression:
    Body of if
else:
    Body of else
```

```
x=1
if x==1:
    print('codotion is true value of x is {}'.format(x))
else:
    print("this is else block")
```

```
x=2
if x==1:
    print('codotion is true value of x is {}'.format(x))
else:
    print("this is else block")
```

Python if...elif...else

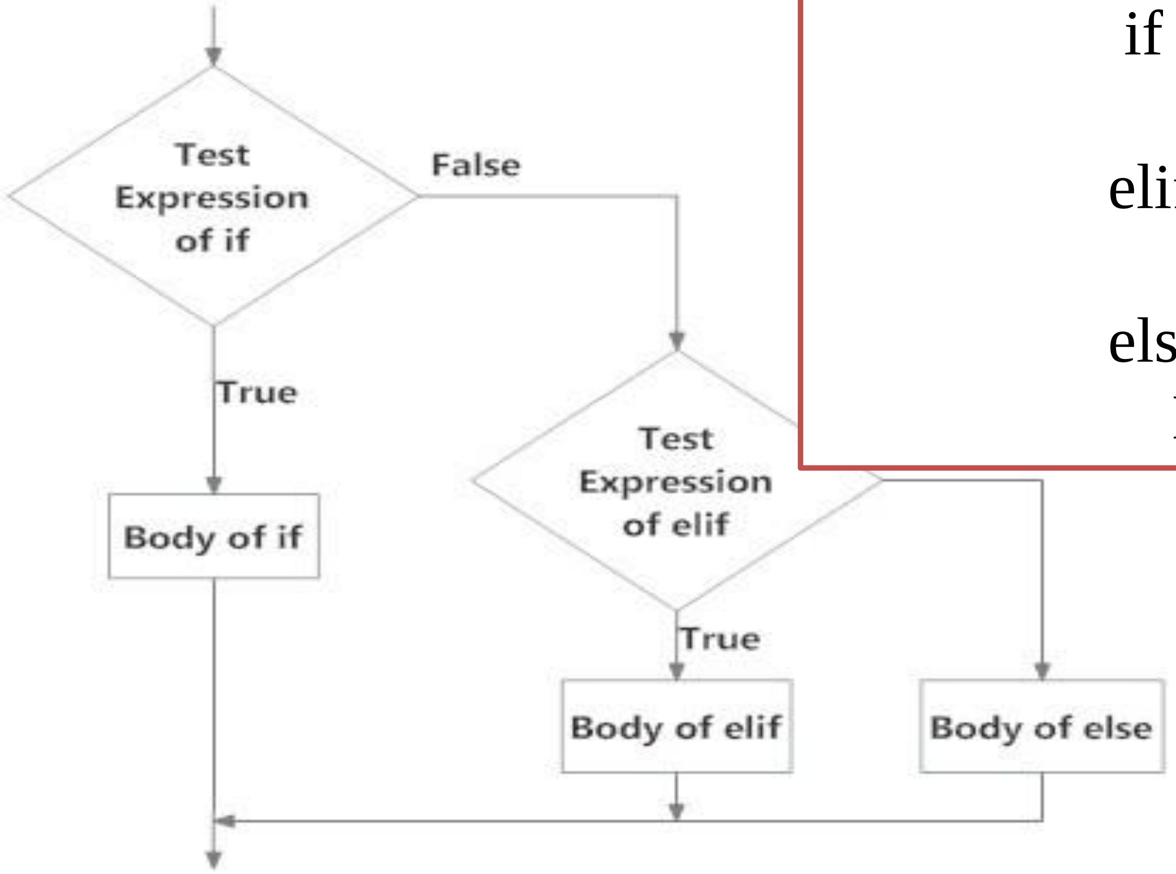


Fig: Operation of if...elif...else statement

```
if test expression:
    Body of if
elif test expression:
    Body of elif
else:
    Body of else
```

function

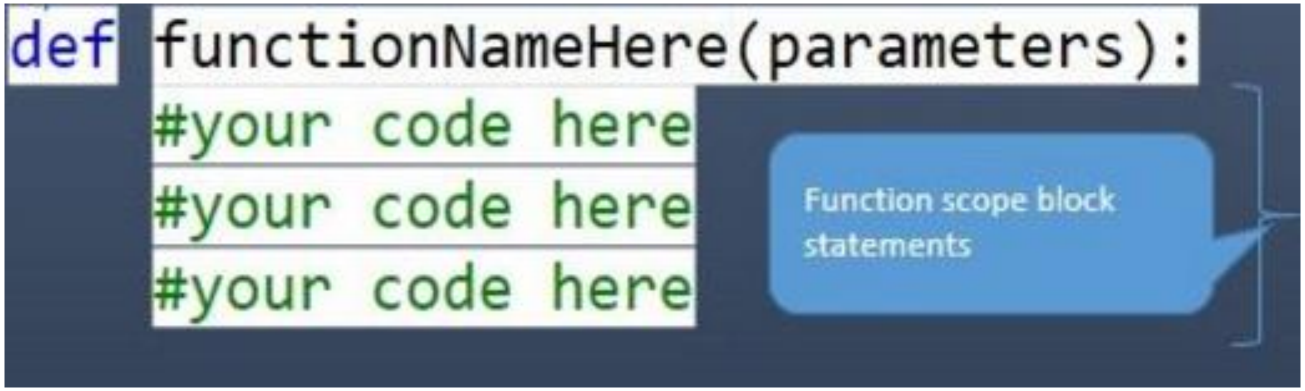
- In Python, function is a group of related statements that perform a specific task.
- Functions help break our program into smaller and modular chunks. As our program grows larger and larger, functions make it more organized and manageable.
- Furthermore, it avoids repetition and makes code reusable.

Defining Function in Python

Defining a function in Python is very easy. You just need to place the def before any name and provide it some block scope.

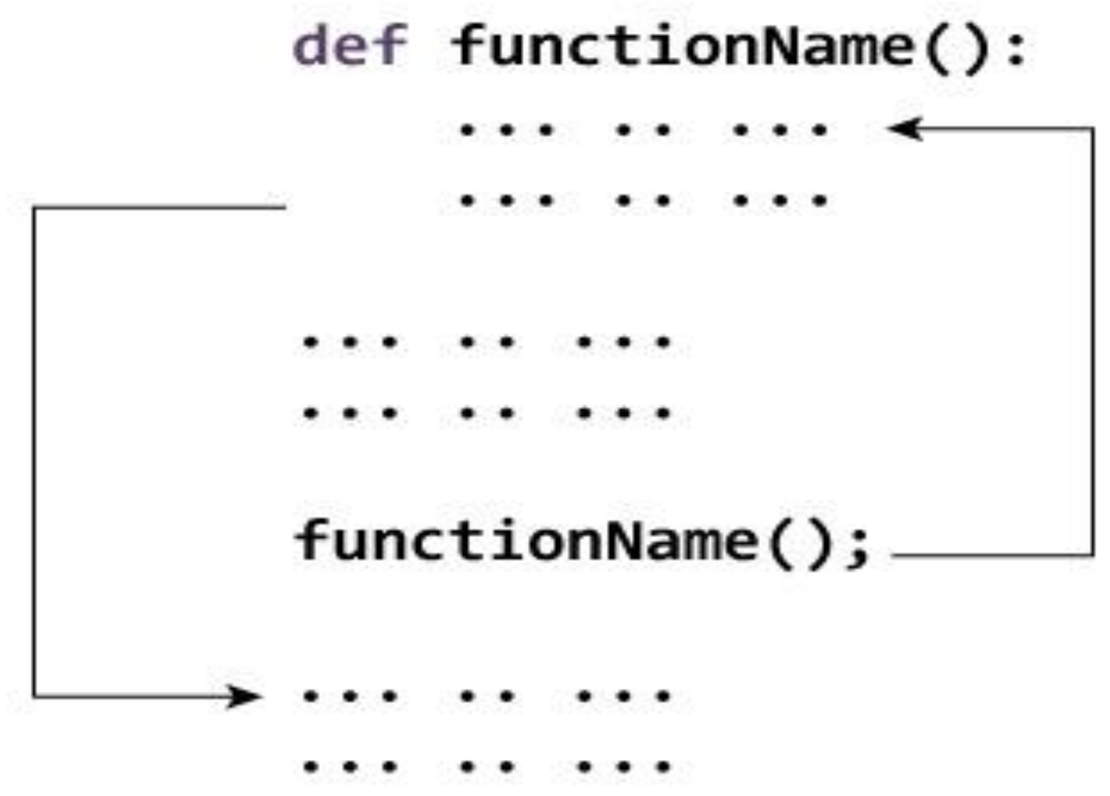
Syntax:

```
def function_name(parameters):
"""docstring"""
 Statement(s)
```



```
def functionNameHere(parameters):
    #your code here
    #your code here
    #your code here
```

How Function works in Python

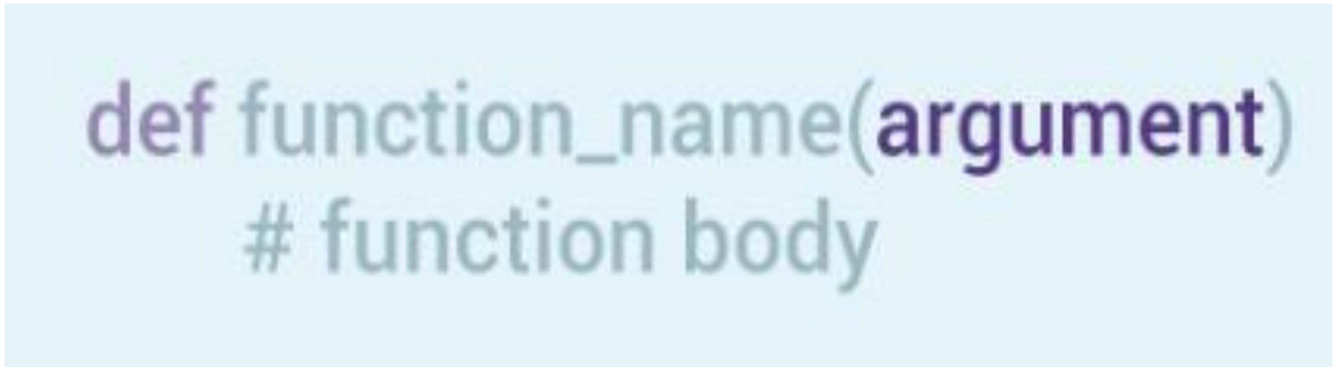


```
def functionName():
    ... .. ...
    ... .. ...

... .. ...
... .. ...

functionName();

... .. ...
... .. ...
```

```
def welcome(name):
        """This function welcome to
        the person passed in as
        parameter"""
        print("Hello, " + name + ". Good morning!")
```

```
def absolute_value(num):
        """This function returns the absolute
        value of the entered number"""
        if num >= 0:
                return num
        else:
                return -num
```

Python Function Arguments



```
def function_name(argument)
        # function body
```

```
def welcome(name,msg):
   """This function greets to
   the person with the provided message"""
   print("Hello",name + ', ' + msg)

welcome("chandan","Good morning!")
```

```
def welcome(name,msg):
   """This function greets to
   the person with the provided message"""
   print("Hello",name + ', ' + msg)

welcome("chandan")
```

```
def welcome(name,msg="good morning"):
   """This function greets to
   the person with the provided message"""
   print("Hello",name + ', ' + msg)

welcome("chandan")
```

```
welcome("chandan","good after noon")
```

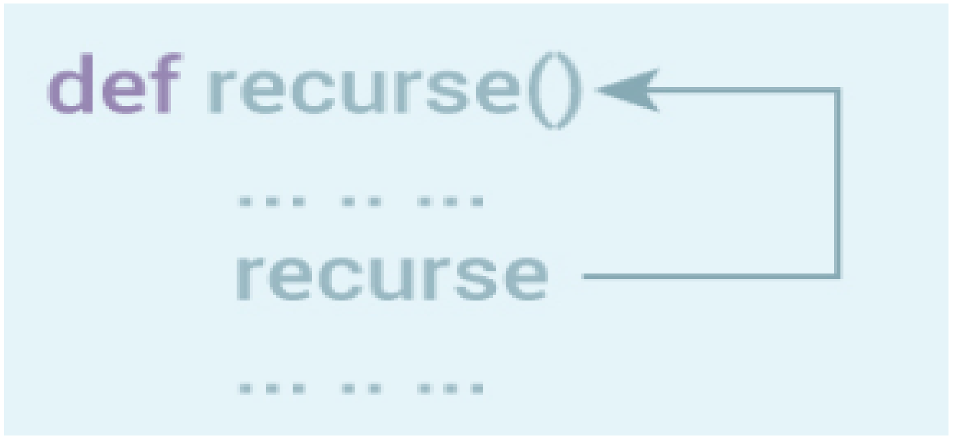
Python Arbitrary Arguments

Sometimes, we do not know in advance the number of arguments that will be passed into a function.Python allows us to handle this kind of situation through function calls with arbitrary number of arguments.
In the function definition we use an asterisk **(*)** before the parameter name to denote this kind of argument.

```
def welcom(*names):
for name in names:
     print("Hello",name)

welcom("chandan","amit","Swati","anil")
```

Python Recursion

Recursion is the process of defining something in terms of itself
function to call itself



```
def recurse()
    ... .. ...
    recurse
    ... .. ...
```

find the factorial of a number
Find Sum of Natural Numbers Using Recursion

```
def my_fact(x):
    """This is a recursive function
    to find the factorial of an integer"""
    if x == 1:
        return 1
    else:
        return (x * my_fact(x-1))

n=int(input("enter any number"))
print("The factorial of", n, "is", my_fact(n))
```

```
def recur_sum(n):
   """Function to return the sum
   of natural numbers using recursion"""
   if n <= 1:
       return n
   else:
       return n + recur_sum(n-1)



num = int(input("Enter a number: "))

if num < 0:
   print("Enter a positive number")
else:
   print("The sum is",recur_sum(num))
```

Python Anonymous Function

In Python, anonymous function is a function that is defined without a name.
While normal functions are defined using the def keyword, in Python anonymous functions are defined using the lambda keyword.

Syntax of Lambda Function
lambda arguments: expression

```
double = lambda x: x * 2
print(double(5))
```

```
def double(x):
   return x * 2
```

Importing User Defined Module

```
def add(a, b):
   """This program adds two
   numbers and return the result"""
   result = a + b
   return result
```

```
>>> mymodule.add(2,4)
```

Scope and Lifetime of variables

Scope of a variable is the portion of a program where the variable is recognized. Parameters and variables defined inside a function is not visible from outside. Hence, they have a local scope.

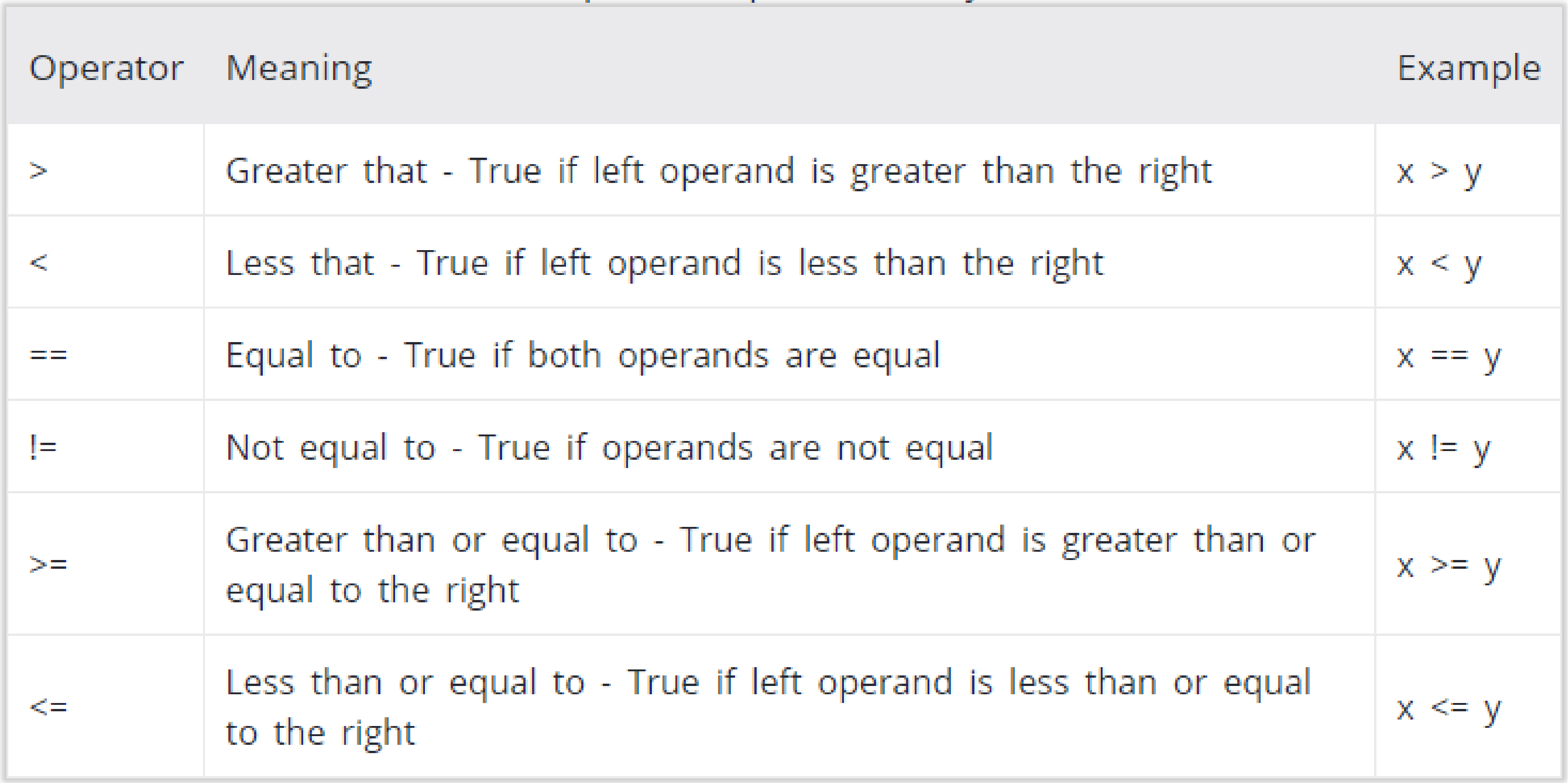
Lifetime of a variable is the period throughout which the variable exits in the memory. The lifetime of variables inside a function is as long as the function executes.
They are destroyed once we return from the function. Hence, a function does not remember the value of a variable from its previous calls.

```
def my_func():
        x = 10
        print("Value inside function:",x)
x = 20
my_func()
print("Value outside function:",x)
```

the variable x inside the function is different (local to the function) from the one outside. Although they have same names, they are two different variables with different scope

Comparison operators

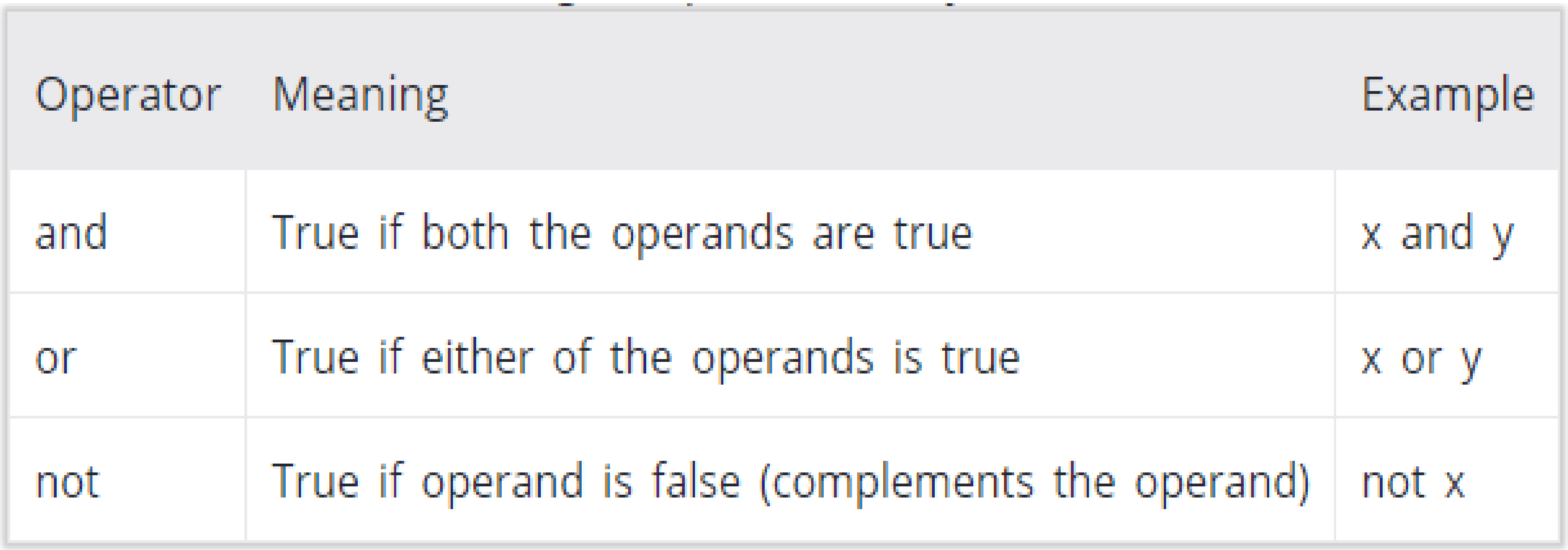
Comparison operators are used to compare values. It either returns True or False according to the condition.



Operator	Meaning	Example
>	Greater that - True if left operand is greater than the right	x > y
<	Less that - True if left operand is less than the right	x < y
==	Equal to - True if both operands are equal	x == y
!=	Not equal to - True if operands are not equal	x != y
>=	Greater than or equal to - True if left operand is greater than or equal to the right	x >= y
<=	Less than or equal to - True if left operand is less than or equal to the right	x <= y

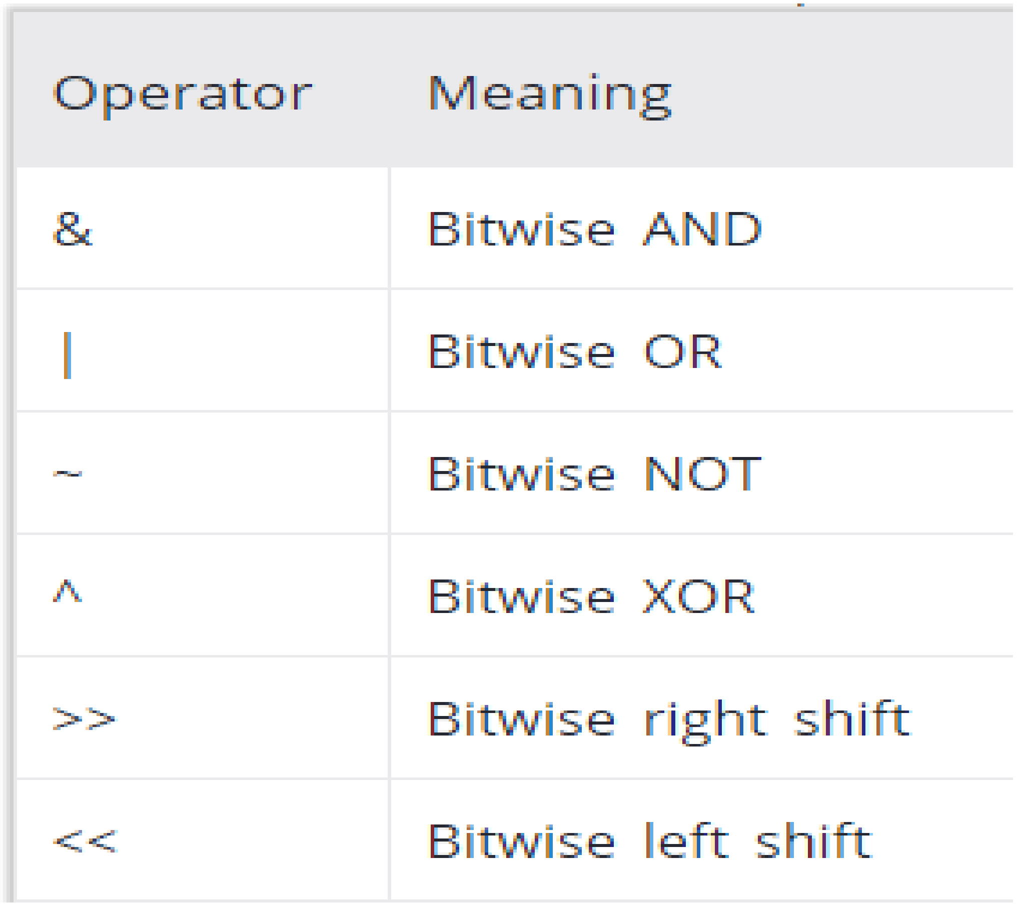
Logical operators

Logical operators are the and, or, not operators.



Operator	Meaning	Example
and	True if both the operands are true	x and y
or	True if either of the operands is true	x or y
not	True if operand is false (complements the operand)	not x

Bitwise operators



Operator	Meaning
&	Bitwise AND
|	Bitwise OR
~	Bitwise NOT
^	Bitwise XOR
>>	Bitwise right shift
<<	Bitwise left shift

```
>>> x=10;y=4
>>> c=x&y
```

```
>>> x=10;y=4
>>> d=x|y
```

```
>>>x=10
>>>~x
>>> x
```

```
>>> x=10
>>> x>>2
```

```
>>> x=10;y=4
>>> f=x^y
```

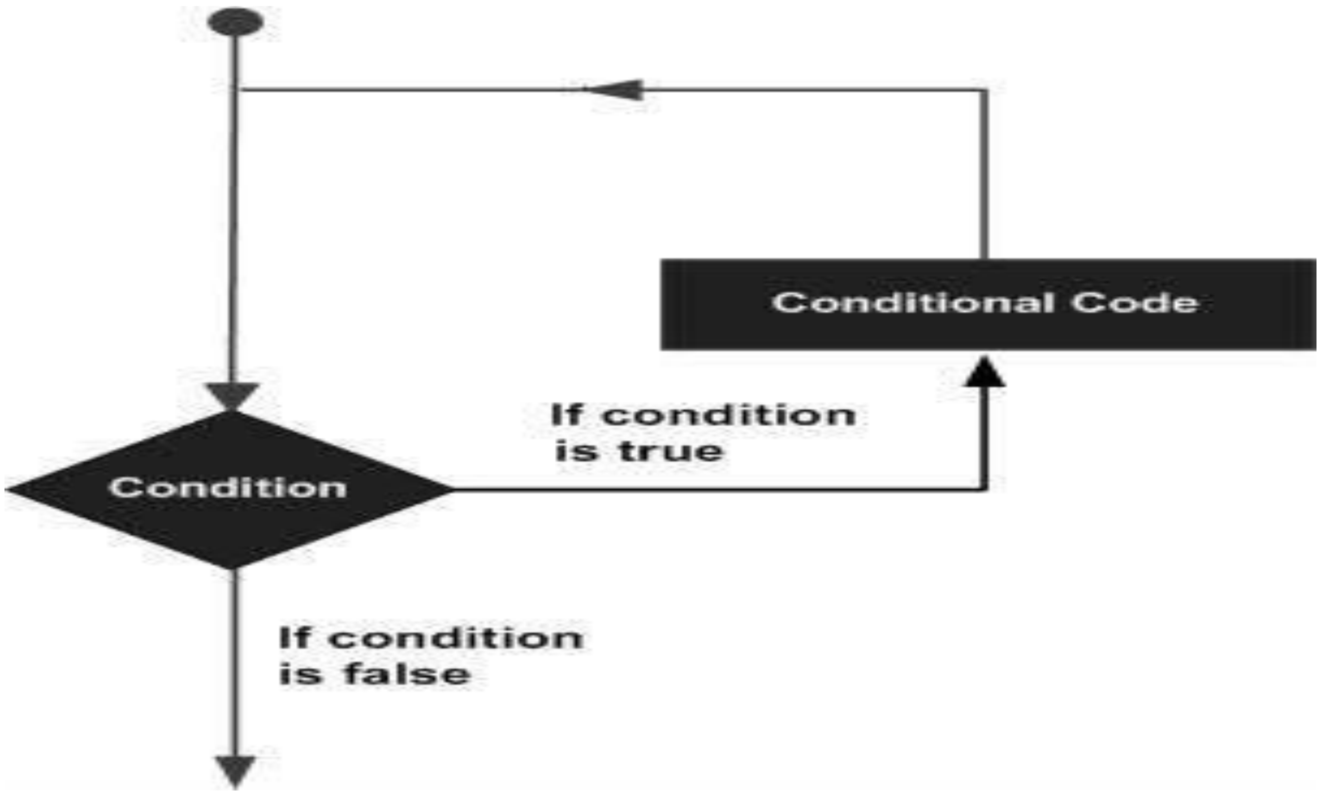
```
>>> x=10
>>> x<<2
```

Program

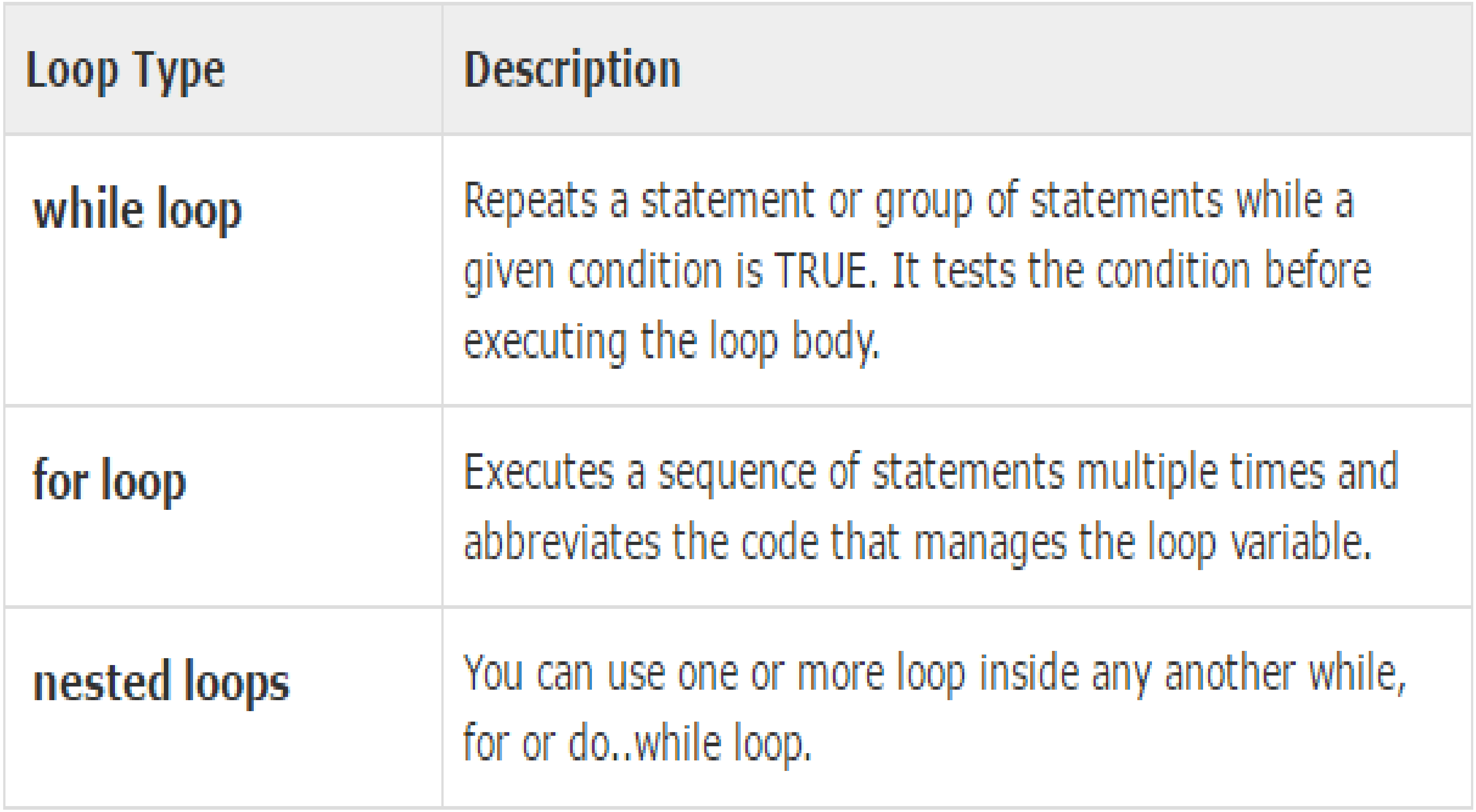
- to check if a number is positive, negative or zero
- Check if a Number is Odd or Even
- to Find the Largest Among Three Numbers
- Program to Check Leap Year
- Make a Simple Calculator

Python Loops

A loop statement allows us to execute a statement or group of statements multiple times.



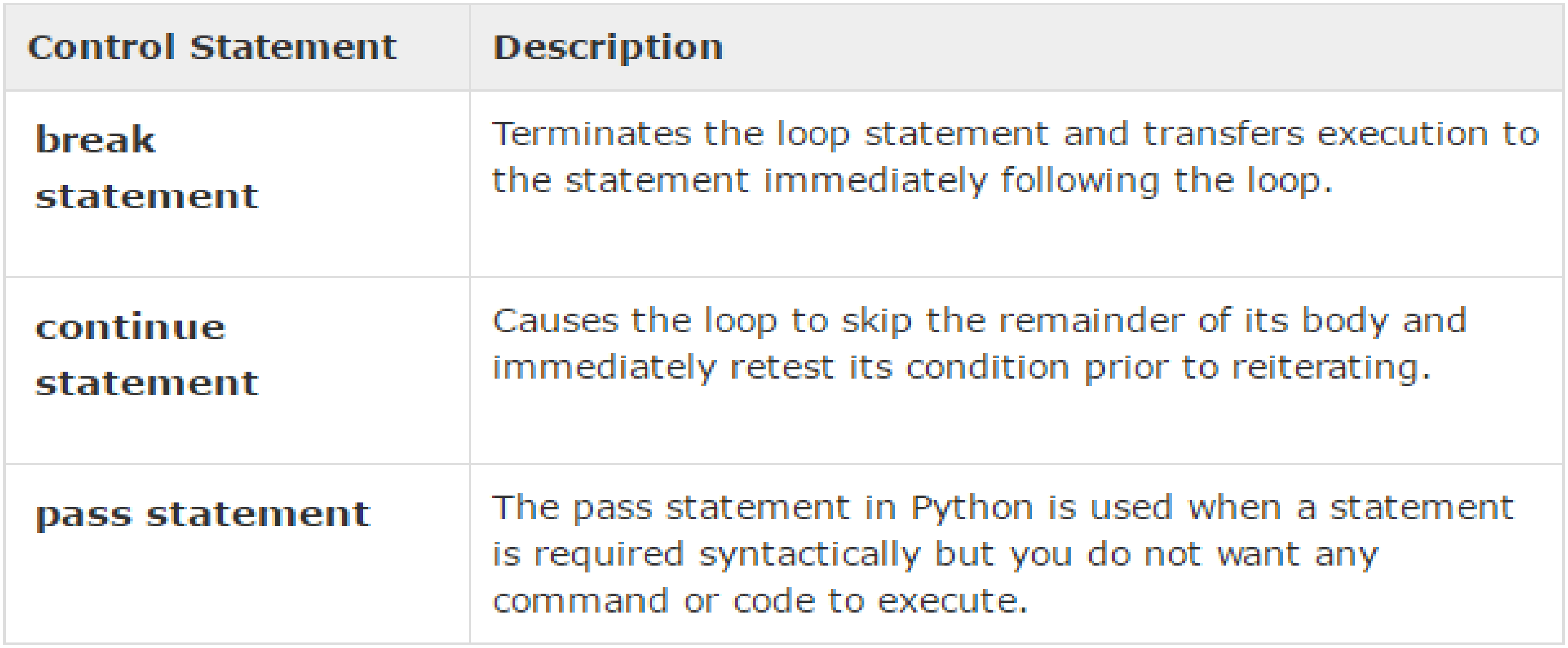
Python Loops



Loop Type	Description
while loop	Repeats a statement or group of statements while a given condition is TRUE. It tests the condition before executing the loop body.
for loop	Executes a sequence of statements multiple times and abbreviates the code that manages the loop variable.
nested loops	You can use one or more loop inside any another while, for or do..while loop.

Loop Control Statements

Loop control statements change execution from its normal sequence. When execution leaves a scope, all automatic objects that were created in that scope are destroyed.



Control Statement	Description
break statement	Terminates the loop statement and transfers execution to the statement immediately following the loop.
continue statement	Causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating.
pass statement	The pass statement in Python is used when a statement is required syntactically but you do not want any command or code to execute.

Python for Loop

```
for val in sequence:
        Body of for
```

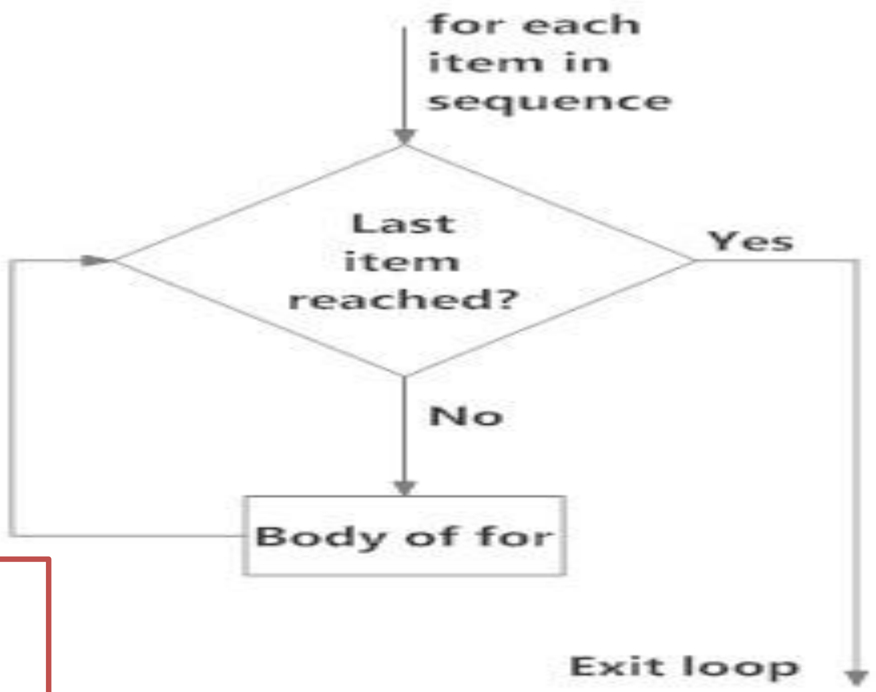


Fig: operation of for loop

```
for count in[0,1,2,3,4,5,6,7,8,9]:
        print("hello")
```

The range function

The range() function is one of Python's built in functions. It is is used to indicate how many times the loop will be repeated.

The structure of the range function is **range(start, upto, step)** in which the arguments of range are used as follows:

```
for count in range(6):
        print("hello")
```

```
for count in range(3,8):
        print(count)
```

```
for count in range(1,50,2):
        print(count)
```

Python while Loop

The while loop in Python is used to iterate over a block of code as long as the test expression (condition) is true.

```
while test_expression:
    Body of while
```

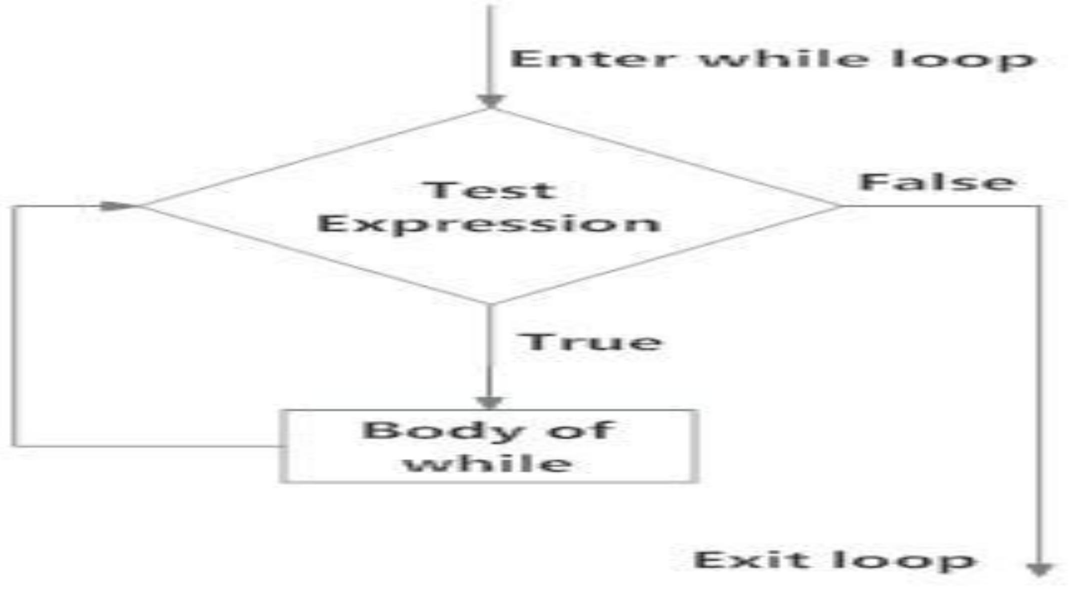


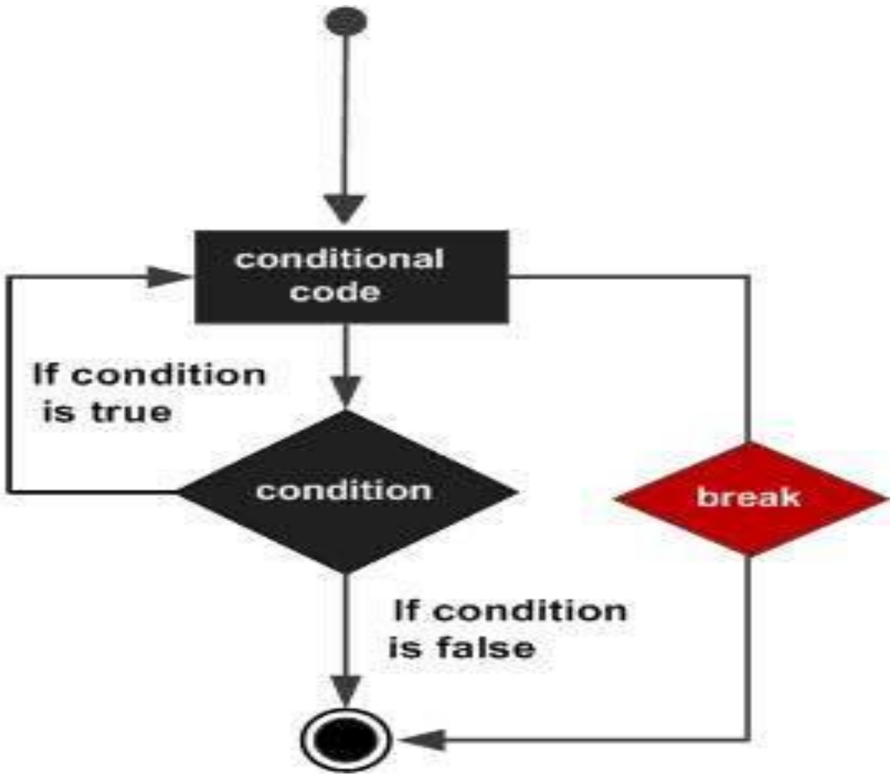
Fig: operation of while loop

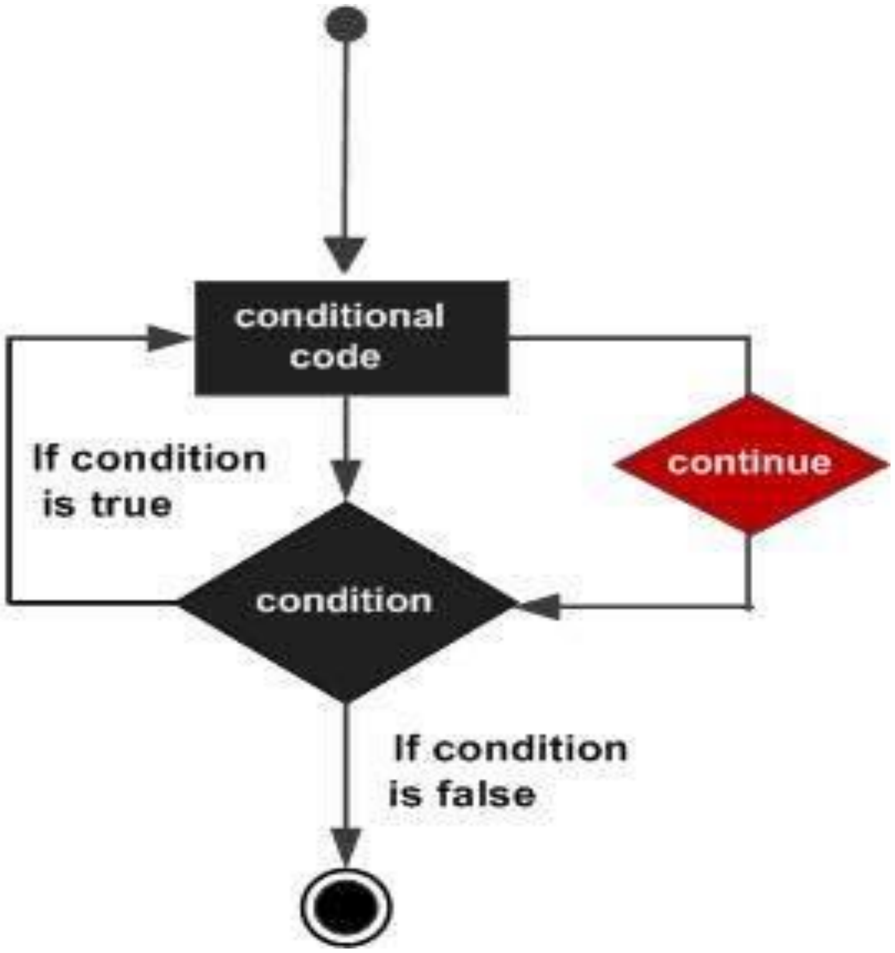
```
counter = 0
while counter < 3:
    print(counter)
    counter = counter + 1
```

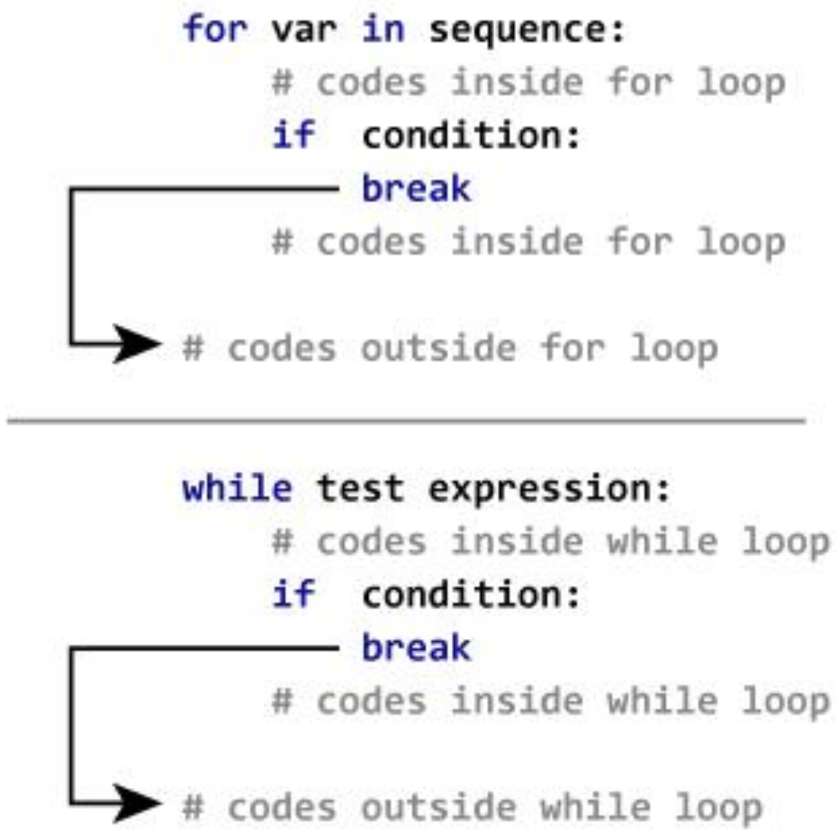
```
counter = 0
while counter < 3:
    print("Inside loop")
    counter = counter + 1
else:
    print("Inside else")
```

Python break and continue

break is used to exit a for loop or a while loop, whereas **continue** is used to skip the current block, and return to the "for" or "while" statement.

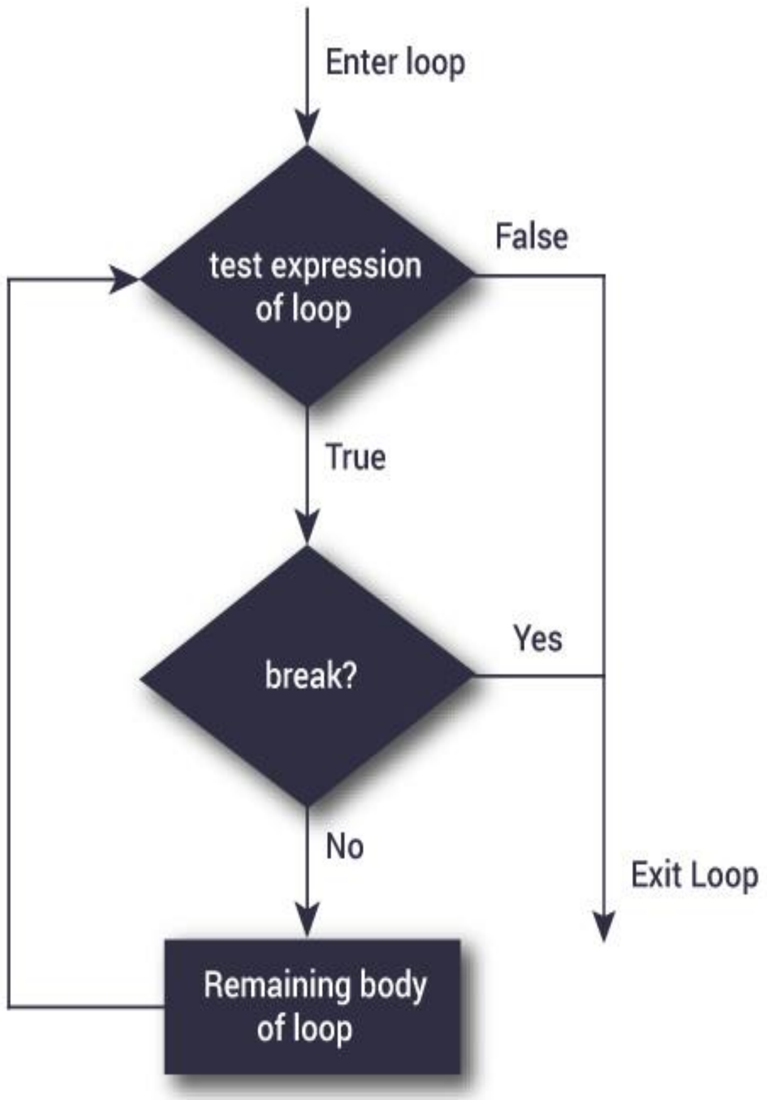



conditional code
If condition is true
continue
condition
If condition is false



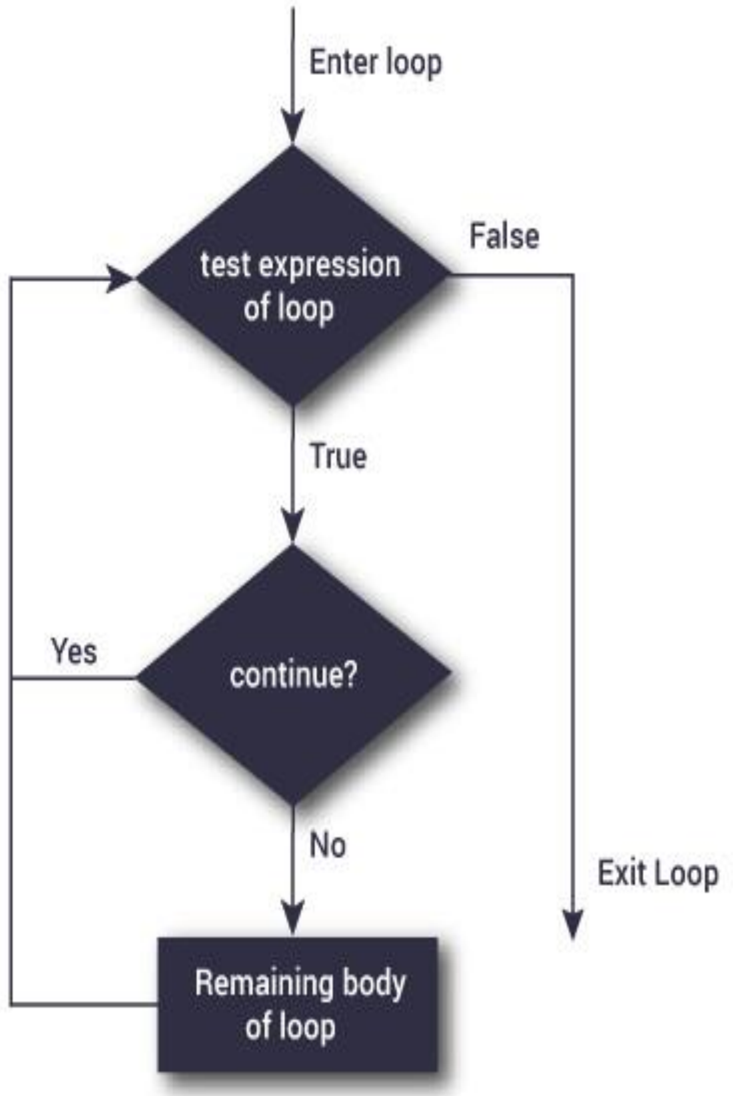
```
for var in sequence:
    # codes inside for loop
    if  condition:
        break
    # codes inside for loop

# codes outside for loop
```

```
while test expression:
    # codes inside while loop
    if  condition:
        break
    # codes inside while loop

# codes outside while loop
```



```
for var in sequence:
    # codes inside for loop
    if  condition:
        continue
    # codes inside for loop

# codes outside for loop
```

```
while test expression:
    # codes inside while loop
    if  condition:
        continue
    # codes  inside while loop

# codes outside while loop
```



```
var = 10
while var > 0:
   print ('Current variable value :', var)
   var = var -1
   if var == 5:
      break
print ("Good bye!")
```

```
var = 10
while var > 0:
   var = var -1
   if var == 5:
      continue
   print ('Current variable value :', var)
print( "Good bye!")
```

```
for val in "string":
    if val == "i":
        break
    print(val)

print("The end")
```

```
for val in "string":
 if val == "i":
continue
 print(val)

print("The end")
```

Python pass Statement

In Python programming, pass is a **null statement**. The difference between a comment and pass statement in Python is that, while the interpreter ignores a comment entirely, pass is not ignored. However, nothing happens when pass is executed. It results into **no operation (NOP).**

```
for letter in 'chandan':
   if letter == 'h':
      pass
      print( 'This is pass block')
   print ('Current Letter :', letter)

print ("Good bye!")
```