INTRODUCTION TO PYTHON

Python Introduction, Technical Strength of Python, Python interpreter and interactive mode, Introduction to colab, pycharm, and jupyter idle(s), Values and types: int, float, boolean, string, and list; Built-in data types, variables, Literals, Constants, statements, Operators: Assignment, Arithmetic, Relational, Logical, Bitwise operators and their precedence, Expressions, tuple assignment, Accepting input from Console, printing statements, Simple Python programs.

INTRODUCTION TO PYTHON

Python is a general-purpose interpreted, interactive, object-oriented, and high-level programming language.

- It was created by Guido van Rossum during 1985- 1990.
- Python got its name from “Monty Python’s flying circus”. Python was released in the year 2000.
- Python is interpreted: Python is processed at runtime by the interpreter. You do not need to compile your program before executing it.
- 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 an Object-Oriented style or technique of programming that encapsulates code within objects.
- Python is a Beginner's Language: Python is a great language for beginner-level programmers and supports the development of a wide range of applications.

Python Features:

- Easy-to-learn: Python is clearly defined and easily readable. The structure of the program is very simple. It uses a few keywords.
- Easy-to-maintain: Python's source code is fairly easy-to-maintain.
- Portable: Python can run on a wide variety of hardware platforms and has the same interface on all platforms.
- Interpreted: Python is processed at runtime by the interpreter. So, there is no need to compile a program before executing it. You can simply run the program.
- Extensible: Programmers can embed python within their C, C++, Java script, ActiveX, etc.
- Free and Open Source: Anyone can freely distribute it, read the source code, and edit it.
- High-Level Language: When writing programs, programmers concentrate on solutions to the current problem, no need to worry about the low-level details.
- Scalable: Python provides a better structure and support for large programs than shell scripting.

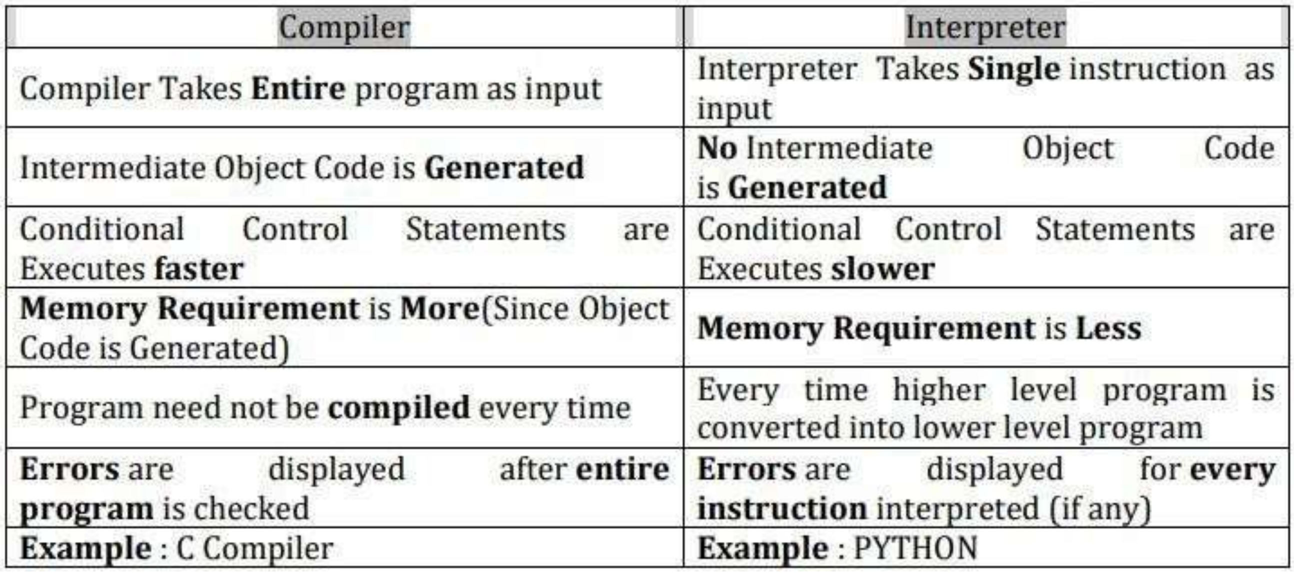
Applications

- Bit Torrent file sharing
- Google search engine, Youtube
- Intel, Cisco, HP, IBM
- i–Robot
- NASA
- Facebook, Dropbox

Python interpreter

Interpreter: To execute a program in a high-level language by translating it one line at a time.

Compiler: To translate a program written in a high-level language into a low-level language all at once, in preparation for later execution.



Compiler	Interpreter
Compiler Takes **Entire** program as input	Interpreter Takes **Single** instruction as input
Intermediate Object Code is **Generated**	**No** Intermediate Object Code is **Generated**
Conditional Control Statements are Executes **faster**	Conditional Control Statements are Executes **slower**
Memory Requirement is **More**(Since Object Code is Generated)	**Memory Requirement** is **Less**
Program need not be **compiled** every time	Every time higher level program is converted into lower level program
Errors are displayed after **entire program** is checked	**Errors** are displayed for **every instruction** interpreted (if any)
Example : C Compiler	**Example** : PYTHON

MODES OF PYTHON INTERPRETER:

Python Interpreter is a program that reads and executes Python code. It uses 2 modes of Execution.

Interactive mode & Script mode

1. Interactive mode:

Interactive Mode, as the name suggests, allows us to interact with OS.

When we type a Python statement, the interpreter displays the result(s) immediately.

Advantages:

Python, in interactive mode, is good enough to learn, experiment or explore.

Working in interactive mode is convenient for beginners and for testing small pieces of code.

Drawback:

We cannot save the statements and have to retype all the statements once again to re-run them.

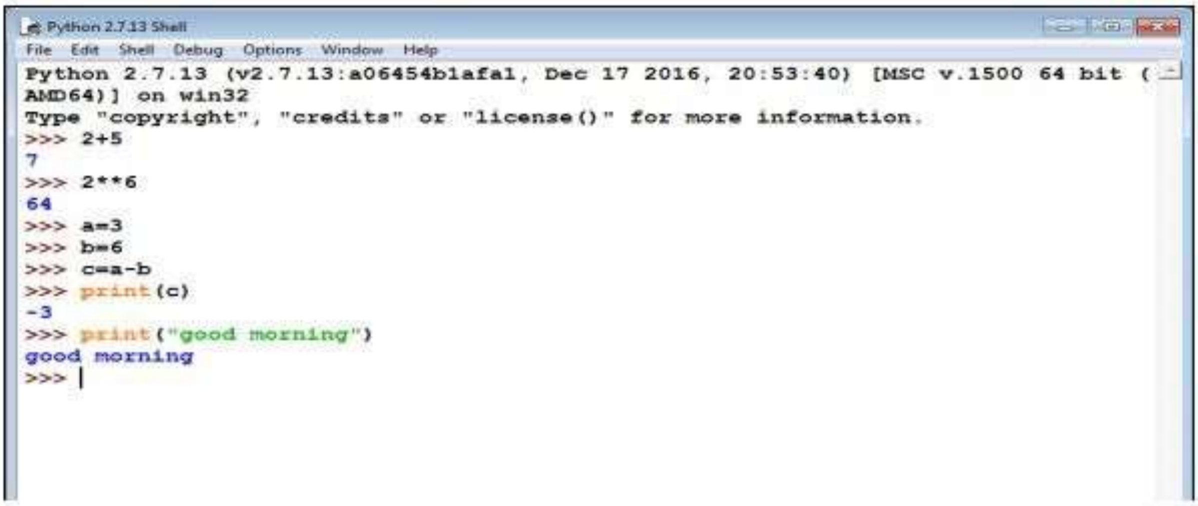
In interactive mode, you type Python programs and the interpreter displays the result:

```
>>>     1 + 1
2
```

The chevron, >>>, is the prompt the interpreter uses to indicate that it is ready for you to enter code. If you type 1 + 1, the interpreter replies 2.

```
>>>     print ('Hello, World!')
Hello, World!
```

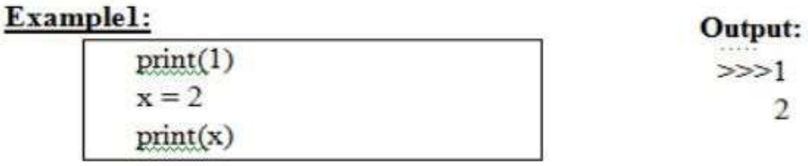
This is an example of a print statement. It displays a result on the screen. In this case, the result is the words.



```
Python 2.7.13 (v2.7.13:a06454b1afa1, Dec 17 2016, 20:53:40) [MSC v.1500 64 bit (
AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> 2+5
7
>>> 2**6
64
>>> a=3
>>> b=6
>>> c=a-b
>>> print(c)
-3
>>> print("good morning")
good morning
>>> |
```

2. Script mode:

- In script mode, we type a python program in a file and then use an interpreter to execute the content of the file.
- Scripts can be saved to disk for future use. Python scripts have the extension .py, meaning that the filename ends with .py
- Save the code with filename.py and run the interpreter in script mode to execute the script.

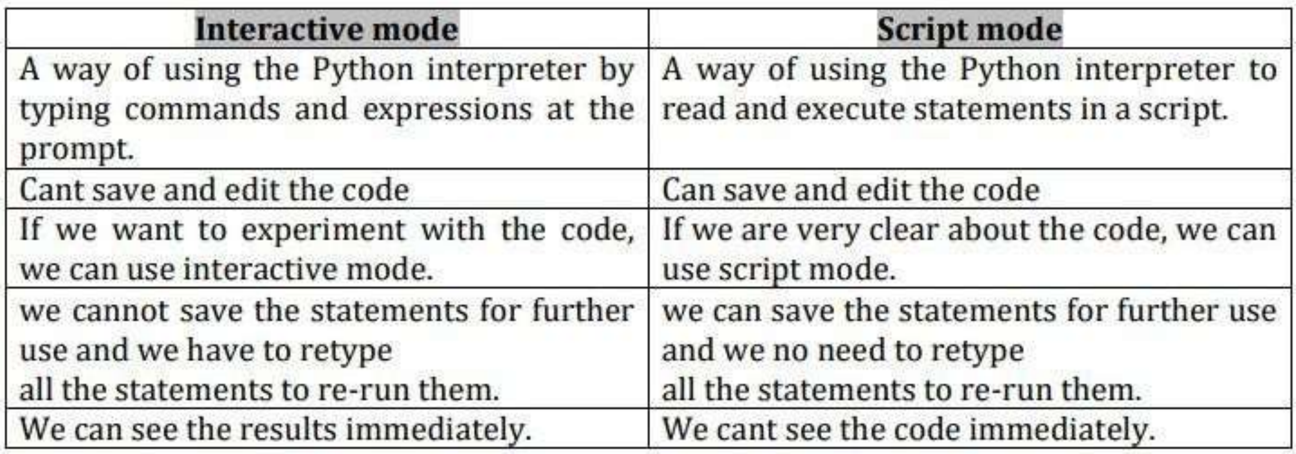


Example1:

```
print(1)
x = 2
print(x)
```

Output:

```
>>>1
   2
```



Interactive mode	Script mode
A way of using the Python interpreter by typing commands and expressions at the prompt.	A way of using the Python interpreter to read and execute statements in a script.
Cant save and edit the code	Can save and edit the code
If we want to experiment with the code, we can use interactive mode.	If we are very clear about the code, we can use script mode.
we cannot save the statements for further use and we have to retype all the statements to re-run them.	we can save the statements for further use and we no need to retype all the statements to re-run them.
We can see the results immediately.	We cant see the code immediately.

Integrated Development Learning Environment (IDLE):

- Is a graphical user interface that is completely written in Python.
- It is bundled with the default implementation of the python language and also comes with an optional part of the Python packaging.

Features of IDLE:

- Multi-window text editor with syntax highlighting.
- Auto-completion with smart indentation.
- Python shell to display output with syntax highlighting.

VALUES AND DATA TYPES

Values

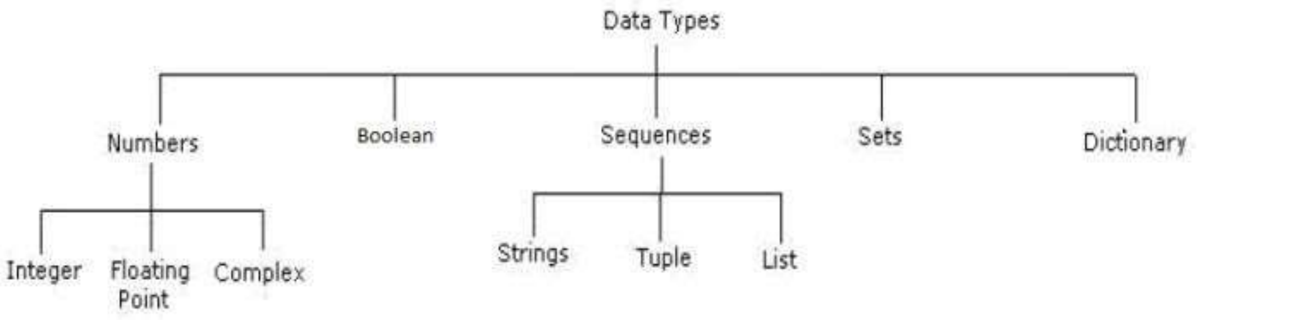
Value can be any letter, number, or string.

Eg, Values are 2, 42.0, and 'Hello, World!'. (These values belong to different datatypes.)

Data type:

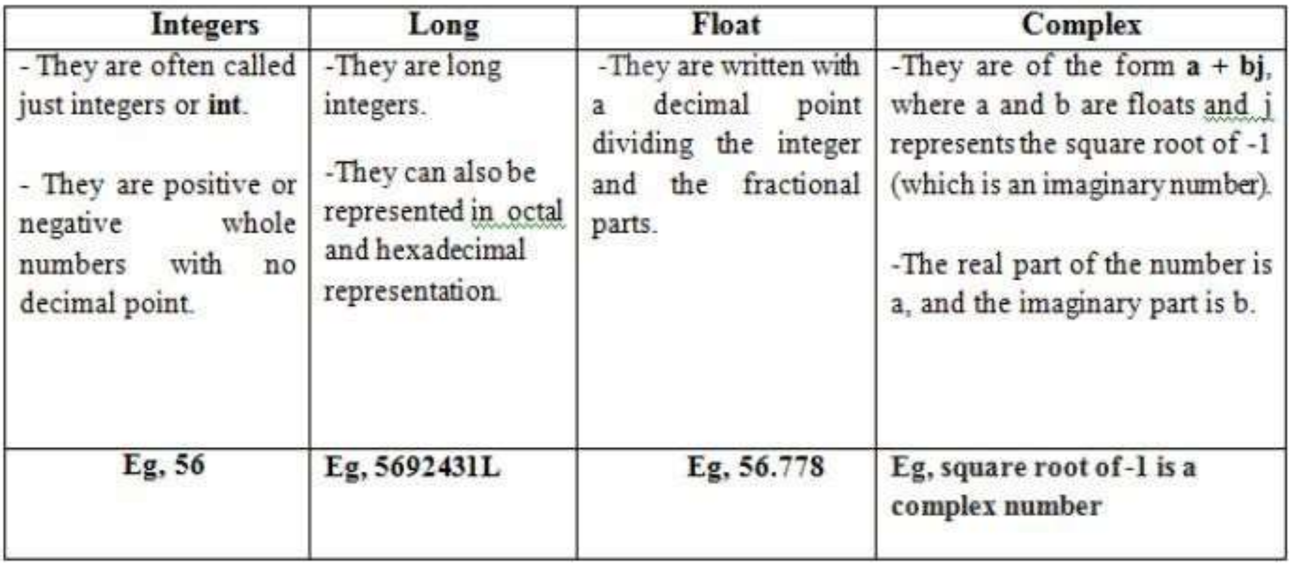
- Every value in Python has a data type.
- It is a set of values and the allowable operations on those values.

Python has four standard data types:



Numbers:

- Number data type stores **Numerical Values.**
- This data type is immutable [i.e. values/items cannot be changed].
- Python supports integers, floating point numbers and complex numbers. They are defined as,



Integers	Long	Float	Complex
- They are often called just integers or **int**. - They are positive or negative whole numbers with no decimal point.	-They are long integers. -They can also be represented in octal and hexadecimal representation.	-They are written with a decimal point dividing the integer and the fractional parts.	-They are of the form **a + bj**, where a and b are floats and j represents the square root of -1 (which is an imaginary number). -The real part of the number is a, and the imaginary part is b.
Eg, 56	**Eg, 5692431L**	**Eg, 56.778**	**Eg, square root of -1 is a complex number**

Sequence:

- A sequence is an **ordered collection of items**, indexed by positive integers.
- It is a combination **of mutable** (value can be changed) **and immutable** (values cannot be changed) data types.
- There are three types of sequence data type available in Python, they are

1. **Strings**
2. **Lists**
3. **Tuples**

1. Strings

A String in Python consists of a series or sequence of characters - letters, numbers, and special characters.

Strings are marked by quotes:

- single quotes (' ') Eg, 'This a string in single quotes'
- double quotes (" ") Eg, "'This a string in double quotes"'
- triple quotes(""" """) Eg, This is a paragraph. It is made up of multiple lines and sentences."""

➢ Individual character in a string is accessed using a subscript (index).

➢ Characters can be accessed using indexing and slicing operations

Strings are immutable i.e. the contents of the string cannot be changed after it is created.

2. Lists

A list is an ordered sequence of items. Values in the list are called elements/items.

It can be written as a list of comma-separated items (values) between **square brackets[ ]**.

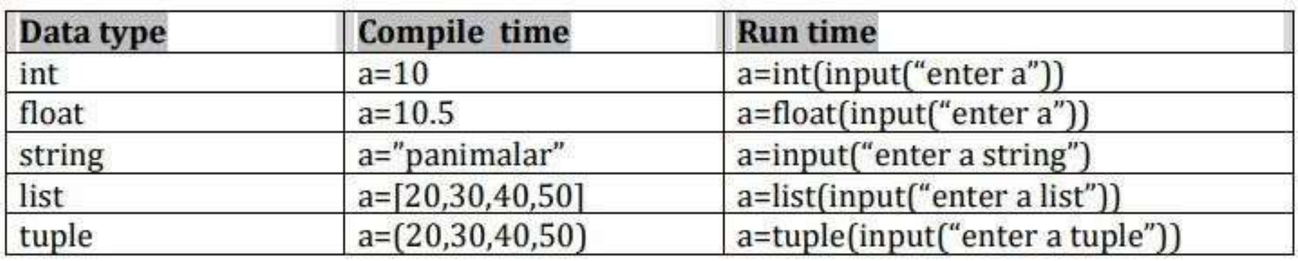
Items in the lists can be of different data types.

3. Tuple:

- A tuple is the same as a list, except that the set of elements is **enclosed in parentheses** instead of square brackets.
- **A tuple is an immutable list.** i.e. once a tuple has been created, you can't add elements to a tuple or remove elements from the tuple.
  The benefit of Tuple:
  - Tuples are faster than lists.
  - If the user wants to protect the data from accidental changes, a tuple can be used.
  - Tuples can be used as keys in dictionaries, while lists can't.

4. Dictionaries:

- Lists are ordered sets of objects, whereas **dictionaries are unordered sets**.
- Dictionary is created by using **curly brackets**. i,e. {}
- Dictionaries **are accessed via keys** and not via their position.
- A dictionary is an associative array (also known as hashes). Any key of the dictionary is associated (or mapped) to a value.
- The values of a dictionary can be any Python data type. So dictionaries are **unordered key-value-pairs(**The association of a key and a value is called a key-value pair)
- Dictionaries don't support the sequence operation of the sequence data types like strings, tuples and lists.



Data type	Compile time	Run time
int	a=10	a=int(input("enter a"))
float	a=10.5	a=float(input("enter a"))
string	a="panimalar"	a=input("enter a string")
list	a=[20,30,40,50]	a=list(input("enter a list"))
tuple	a=(20,30,40,50)	a=tuple(input("enter a tuple"))

VARIABLES:

A variable allows us to store a value by assigning it to a name, which can be used later.

- Named memory locations to store values.
- Programmers generally choose names for their variables that are meaningful.
- It can be of any length. No space is allowed.
- We don't need to declare a variable before using it. In Python, we simply assign a value to a variable and it will exist.

Assigning value to a variable:

The value should be given on the right side of the assignment operator(=) and the variable on the left side.

>>>**counter =45**

print(counter)

Assigning a single value to several variables simultaneously:

>>> **a=b=c=100**

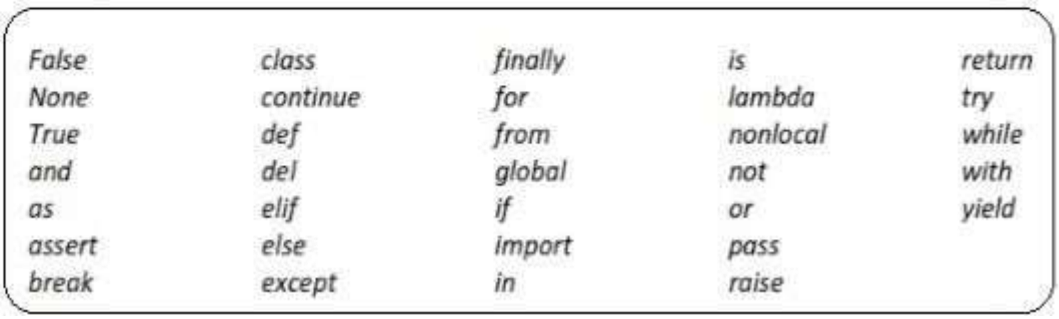
Assigning multiple values to multiple variables:

>>> **a,b,c=2,4,"ram"**

KEYWORDS

Keywords are the reserved words in Python.

- We cannot use a keyword as a variable name, function name, or any other identifier.
- They are used to define the syntax and structure of the Python language.
- Keywords are case-sensitive.



False

None

True

and

as

assert

break

class

continue

def

del

elif

else

except

finally

for

from

global

if

import

in

is

lambda

nonlocal

not

or

pass

raise

return

try

while

with

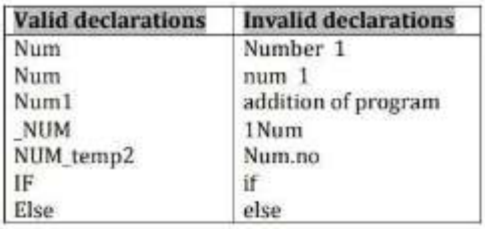
yield

IDENTIFIERS

Identifier is the name given to entities like classes, functions, variables, etc. in Python.

- Identifiers can be a combination of letters in lowercase (a to z) or uppercase (A to Z) or digits (0 to 9) or underscore (_). all are valid examples.
- An identifier cannot start with a digit.
- Keywords cannot be used as identifiers.
- Cannot use special symbols like! @, #, $, %, etc. in our identifier.
- Identifiers can be of any length.

Example: Names like myClass, var_1, and this_is_a_long_variable



Valid declarations	Invalid declarations
Num	Number 1
Num	num 1
Num1	addition of program
_NUM	1Num
NUM_temp2	Num.no
IF	if
Else	else

STATEMENTS AND EXPRESSIONS

Statements: Instructions that a Python interpreter can execute are called statements.

A statement is a unit of code like creating a variable or displaying a value.

```
>>> n = 17
>>> print(n)
```

Here, The first line is an assignment statement that gives a value to n.

The second line is a print statement that displays the value of n.

Expressions: An expression is a **combination of values, variables, and operators.**

A value all by itself is considered an expression, and also a variable.

So the following are all legal expressions:

```
>>>  42
42
>>>  a=2
>>> a+3+2
7
>>>  z=("hi"+"friend")
>>> print(z)
Hifriend
```

INPUT AND OUTPUT

INPUT: Input is data entered by the user (end-user) in the program.

In python, the **input () function** is available for input.

Syntax for input() is: variable = input (“data”)

Example:

```
>>>  x=input("enter the name:") enter the name: george
>>>  y=int(input("enter the number"))
enter the number 3  #python accepts a string as the default data type. conversion is required for type.
```

OUTPUT: Output can be displayed to the user using a Print statement.

Syntax: print (expression/constant/variable)

Example: print ("Hello")

Hello

COMMENTS

A **hash sign (#)** is the beginning of a comment.

Anything written after # in a line is ignored by interpreter.

Eg: percentage = (minute * 100) / 60 **# calculating percentage of an hour**

Python **<u>does not have multiple-line commenting feature.</u>** You have to comment each line individually as follows :

Example:

```
#       This is a comment.

#       This is a comment, too.

#       I said that already.
```

LINES AND INDENTATION

- Most of the programming languages like C, C++, Java use braces { } to define a block of code. But, python uses indentation.
- <u>Blocks of code</u> are denoted by line indentation.
- It is a <u>space given to the block of codes</u> for class and function definitions or flow control.

<u>Example:</u>

```
a=3

b=1

if a>b:

        print("a is greater")

else:

        print("b is greater")
```

TUPLE ASSIGNMENT

An assignment to all of the elements in a tuple using a single assignment statement.

- Python has a very powerful **tuple assignment** feature that allows a tuple of variables on the left of an assignment to be assigned values from a tuple on the right of the assignment.
- The left side is a tuple of variables; the right side is a tuple of values.
- Each value is assigned to its respective variable.
- All the expressions on the right side are evaluated before any of the assignments. This feature makes tuple assignments quite versatile.
- Naturally, the number of variables on the left and the number of values on the right have to be the same.

```
>>>   (a, b, c, d) = (1, 2, 3)

ValueError: need more than 3 values to unpack
```

Example:

It is useful to swap the values of two variables. With **conventional assignment statements**, we have to use a temporary variable. For example, to swap a and b:

Swap two numbers

a=2;b=3
print(a,b)
temp = a
a = b
b = temp
print(a,b)
	Output:
	(2, 3)
	(3, 2)

Tuple assignment solves this problem neatly: (a, b) = (b, a)

OPERATORS

Operators are the constructs that can manipulate the value of operands.

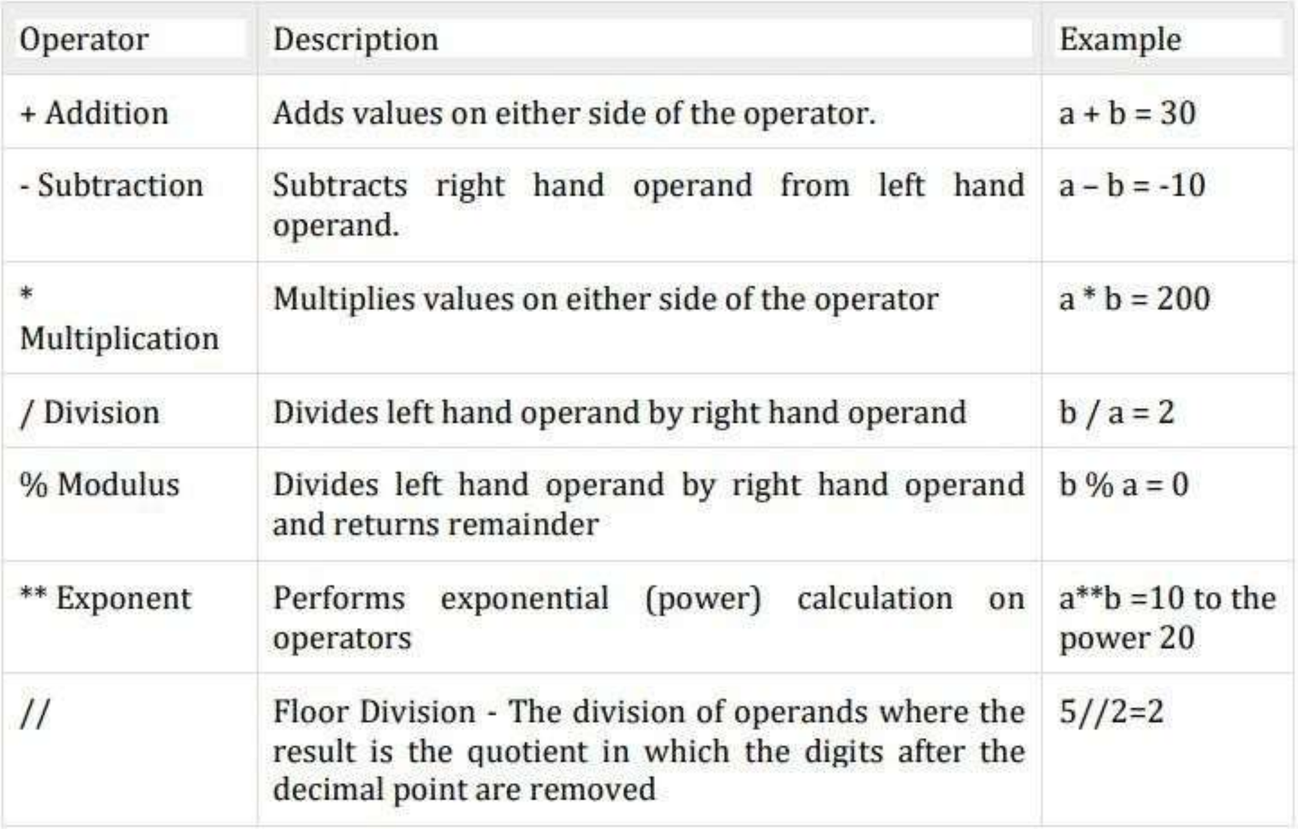
Consider the expression *4 + 5 = 9*. Here, 4 and 5 are called operands and + is called operator

Types of Operators:

Python language supports the following types of operators

- Arithmetic Operators
- Comparison (Relational) Operators
- Assignment Operators
- Logical Operators
- Bitwise Operators
- Membership Operators
- Identity Operators

Arithmetic operators: They are used to perform mathematical operations like addition, subtraction, multiplication etc. Assume, a=10 and b=5



Operator	Description	Example
+ Addition	Adds values on either side of the operator.	a + b = 30
- Subtraction	Subtracts right hand operand from left hand operand.	a – b = -10
* Multiplication	Multiplies values on either side of the operator	a * b = 200
/ Division	Divides left hand operand by right hand operand	b / a = 2
% Modulus	Divides left hand operand by right hand operand and returns remainder	b % a = 0
** Exponent	Performs exponential (power) calculation on operators	a**b =10 to the power 20
//	Floor Division - The division of operands where the result is the quotient in which the digits after the decimal point are removed	5//2=2

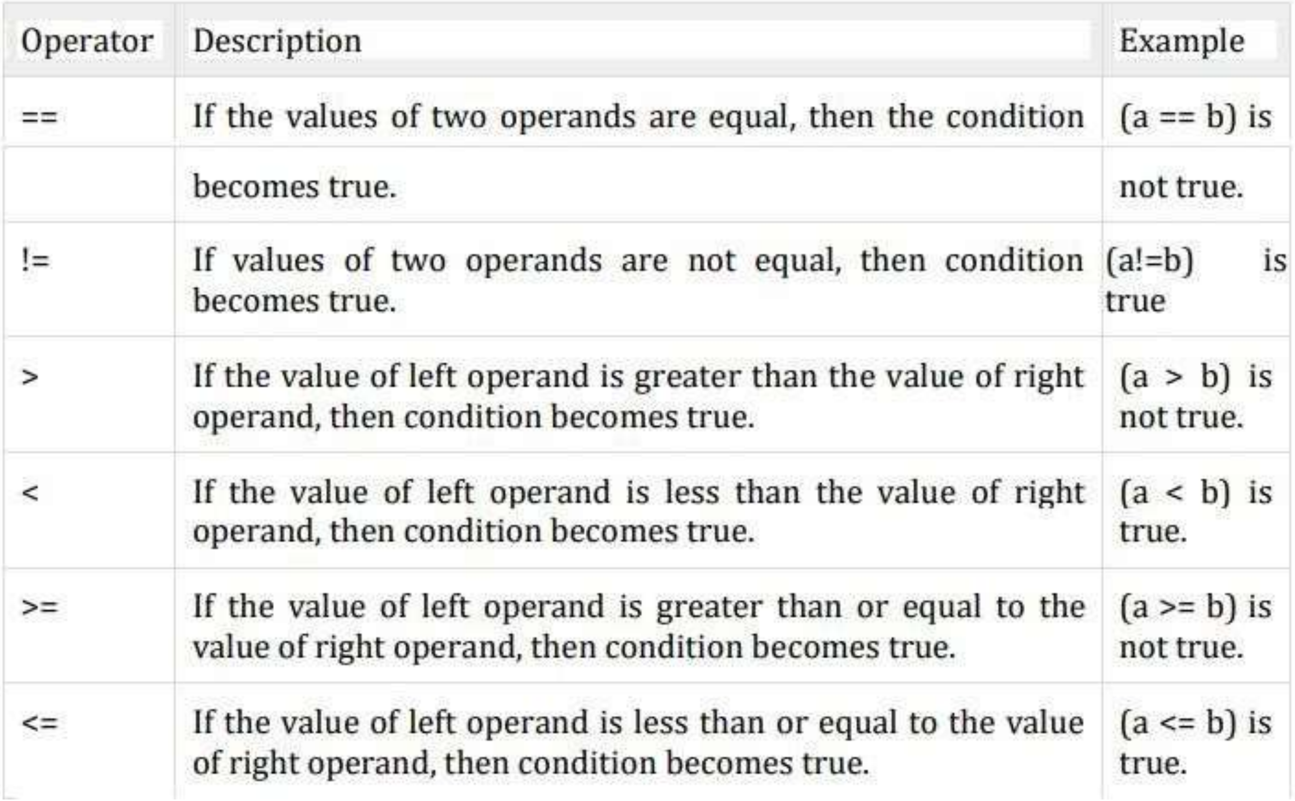
Examples	**Output:**

a=10
b=5
print("a+b=",a+b)
print("a-b=",a-b)
print("a*b=",a*b)
print("a/b=",a/b)
print("a%b=",a%b)
print("a//b=",a//b)
print("a**b=",a**b)
	a+b= 15
	a-b= 5
	a*b= 50
	a/b= 2.0
	a%b= 0
	a//b= 2
	a**b= 100000

Comparison (Relational) Operators:

Comparison operators are used to comparing values.

It either returns True or False according to the condition. Assume, a=10 and b=5

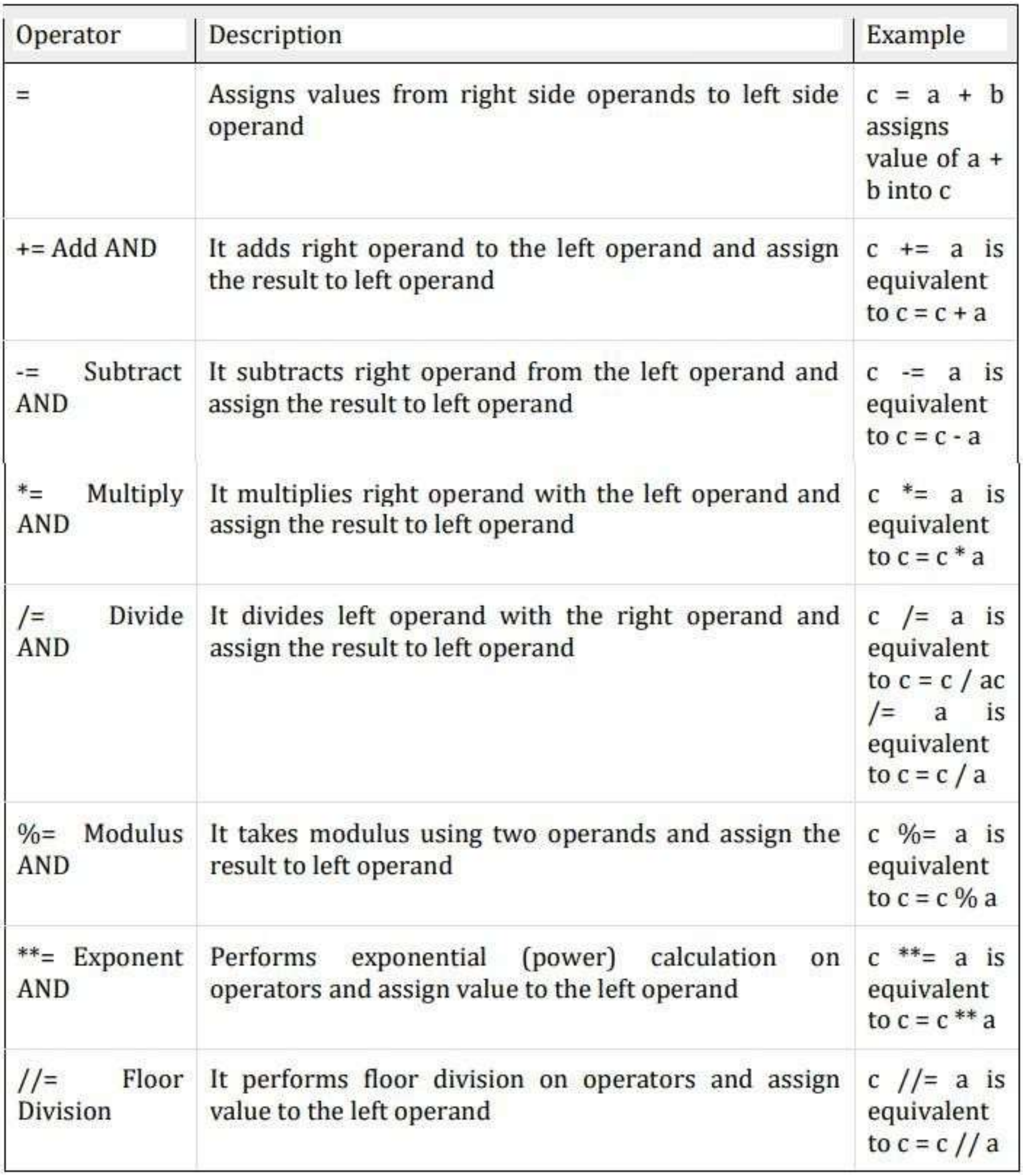


Operator	Description	Example
==	If the values of two operands are equal, then the condition	(a == b) is
	becomes true.	not true.
!=	If values of two operands are not equal, then condition becomes true.	(a!=b) is true
>	If the value of left operand is greater than the value of right operand, then condition becomes true.	(a > b) is not true.
<	If the value of left operand is less than the value of right operand, then condition becomes true.	(a < b) is true.
>=	If the value of left operand is greater than or equal to the value of right operand, then condition becomes true.	(a >= b) is not true.
<=	If the value of left operand is less than or equal to the value of right operand, then condition becomes true.	(a <= b) is true.

Example	**Output:**
a=10	a>b=> True
b=5	a>b=> False
print("a>b=>",a>b)	a==b=> False
print("a>b=>",a<b)	a!=b=> True
print("a==b=>",a==b)	a>=b=> False
print("a!=b=>",a!=b)	a>=b=> True
print("a>=b=>",a<=b)	
print("a>=b=>",a>=b)	

Assignment Operators:

Assignment operators are used in Python to assign values to variables.

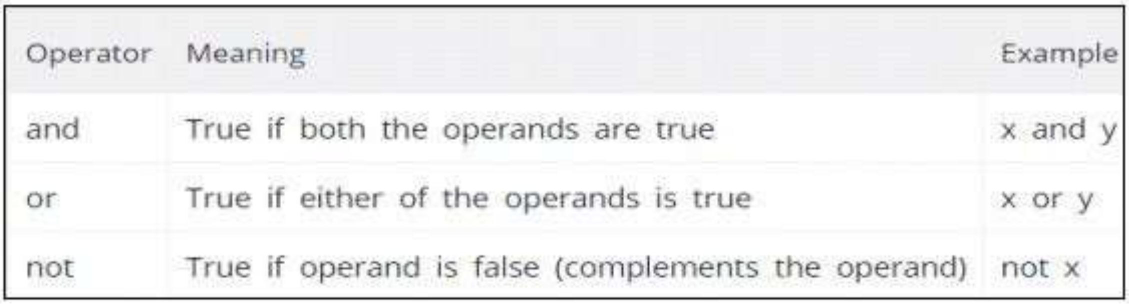


Operator	Description	Example
=	Assigns values from right side operands to left side operand	c = a + b assigns value of a + b into c
+= Add AND	It adds right operand to the left operand and assign the result to left operand	c += a is equivalent to c = c + a
-= Subtract AND	It subtracts right operand from the left operand and assign the result to left operand	c -= a is equivalent to c = c - a
*= Multiply AND	It multiplies right operand with the left operand and assign the result to left operand	c *= a is equivalent to c = c * a
/= Divide AND	It divides left operand with the right operand and assign the result to left operand	c /= a is equivalent to c = c / ac /= a is equivalent to c = c / a
%= Modulus AND	It takes modulus using two operands and assign the result to left operand	c %= a is equivalent to c = c % a
**= Exponent AND	Performs exponential (power) calculation on operators and assign value to the left operand	c **= a is equivalent to c = c ** a
//= Floor Division	It performs floor division on operators and assign value to the left operand	c //= a is equivalent to c = c // a

Example	Output
a = 21 b = 10 c = 0 c = a + b print("Line 1 - Value of c is ", c) c += a print("Line 2 - Value of c is ", c) c *= a print("Line 3 - Value of c is ", c) c /= a print("Line 4 - Value of c is ", c) c= 2 c %= a print("Line 5 - Value of c is ", c) c **= a print("Line 6 - Value of c is ", c) c //= a print("Line 7 - Value of c is ", c)	Line 1 - Value of c is 31 Line 2 - Value of c is 52 Line 3 - Value of c is 1092 Line 4 - Value of c is 52.0 Line 5 - Value of c is 2 Line 6 - Value of c is 2097152 Line 7 - Value of c is 99864

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

Example	Output

a = True
b = False
print('a and b is',a and b)
print('a or b is',a or b)
print('not a is',not a)
	x and y is False
	x or y is True
	not x is False

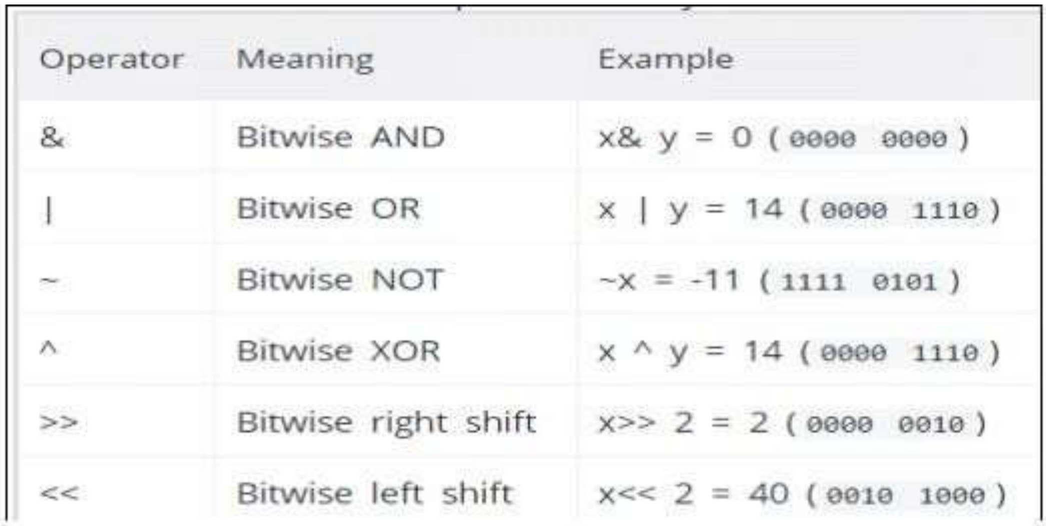
Bitwise Operators:

A bitwise operation operates on one or more bit patterns at the level of individual Bits

Example:

Let x = 10 (0000 1010 in binary) and

y = 4 (0000 0100 in binary)



Operator	Meaning	Example
&	Bitwise AND	x& y = 0 (0000 0000)
|	Bitwise OR	x | y = 14 (0000 1110)
~	Bitwise NOT	~x = -11 (1111 0101)
^	Bitwise XOR	x ^ y = 14 (0000 1110)
>>	Bitwise right shift	x>> 2 = 2 (0000 0010)
<<	Bitwise left shift	x<< 2 = 40 (0010 1000)

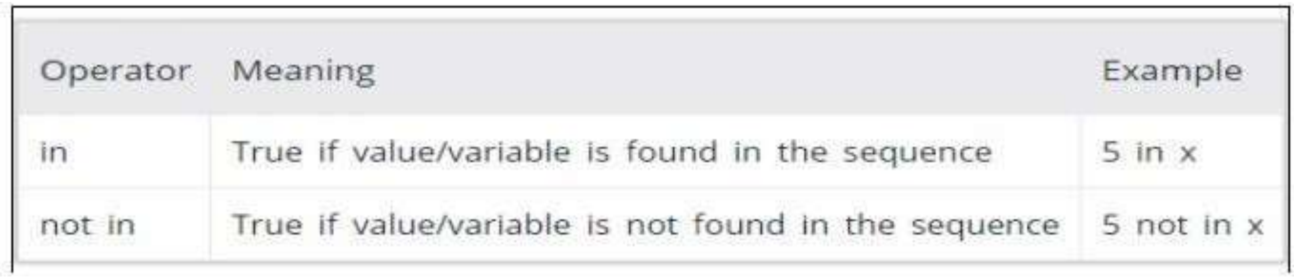
Example	**Output**

a = 60 # 60 = 0011 1100
b = 13 # 13 = 0000 1101
c = 0
c = a & b; # 12 = 0000 1100
print "Line 1 - Value of c is ", c
c = a | b; # 61 = 0011 1101
print "Line 2 - Value of c is ", c
c = a ^ b; # 49 = 0011 0001
print "Line 3 - Value of c is ", c
c = ~a; # -61 = 1100 0011
print "Line 4 - Value of c is ", c
c = a << 2; # 240 = 1111 0000
print "Line 5 - Value of c is ", c
c = a >> 2; # 15 = 0000 1111
print "Line 6 - Value of c is ", c
	Line 1 - Value of c is 12
	Line 2 - Value of c is 61
	Line 3 - Value of c is 49
	Line 4 - Value of c is -61
	Line 5 - Value of c is 240
	Line 6 - Value of c is 15

Membership Operators:

Evaluates to find a value or a variable is in the specified sequence of string, list, tuple, dictionary or not.

Let, x=[5,3,6,4,1]. To check particular item in list or not, in and not in operators are used.



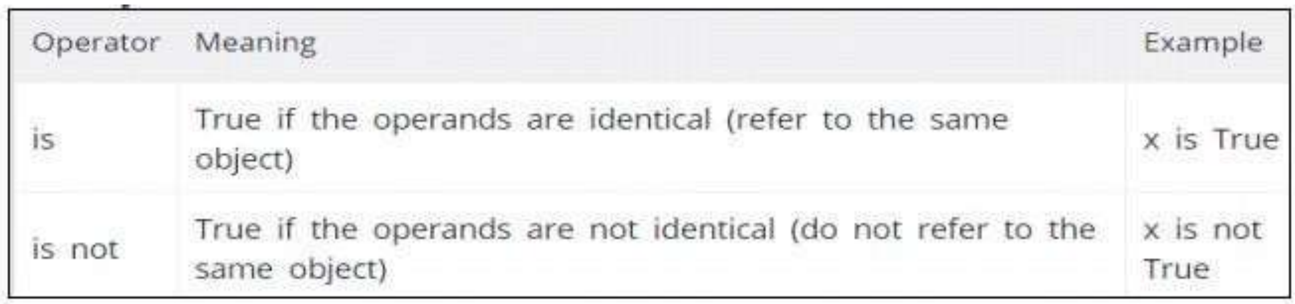
Operator	Meaning	Example
in	True if value/variable is found in the sequence	5 in x
not in	True if value/variable is not found in the sequence	5 not in x

Example:

```
x=[5,3,6,4,1]
>>>  5 in x
True
>>>  5 not in x
False
```

Identity Operators

They are used to check if two values (or variables) are located on the same part of the memory.



Operator	Meaning	Example
is	True if the operands are identical (refer to the same object)	x is True
is not	True if the operands are not identical (do not refer to the same object)	x is not True

Example

```
x = 5
y = 5
x2 = 'Hello'
y2 = 'Hello'
print(x1 is not y1)
print(x2 is y2)
```

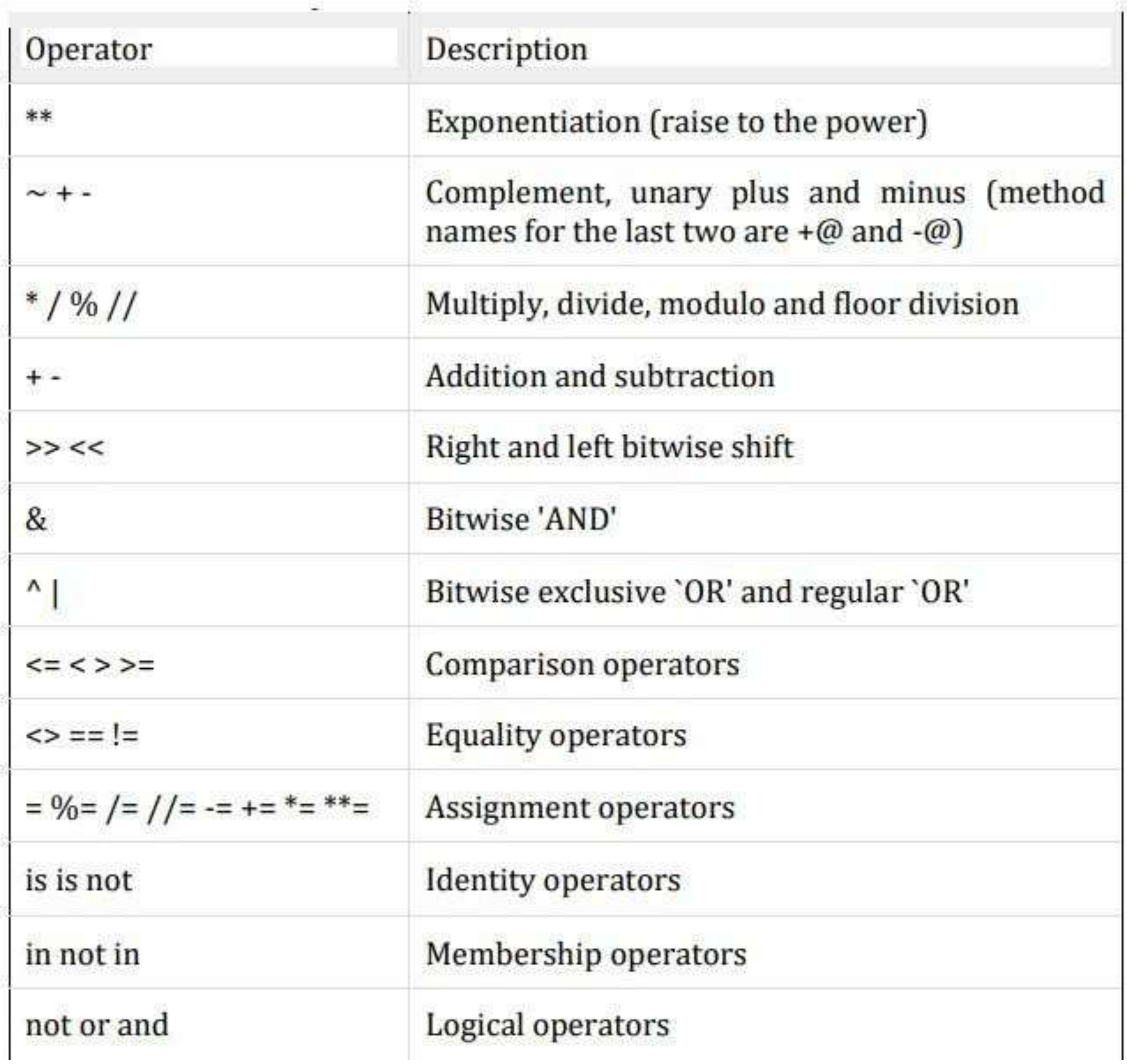
Output

False

True

OPERATOR PRECEDENCE

When an expression contains more than one operator, the order of evaluation depends on the order of operations.



Operator	Description
**	Exponentiation (raise to the power)
~ + -	Complement, unary plus and minus (method names for the last two are +@ and -@)
* / % //	Multiply, divide, modulo and floor division
+ -	Addition and subtraction
>> <<	Right and left bitwise shift
&	Bitwise 'AND'
^ |	Bitwise exclusive `OR' and regular `OR'
<= < > >=	Comparison operators
<> == !=	Equality operators
= %= /= //= -= += *= **=	Assignment operators
is is not	Identity operators
in not in	Membership operators
not or and	Logical operators

For mathematical operators, Python follows mathematical conventions.

The acronym PEMDAS (Parentheses, Exponentiation, Multiplication, Division, Addition, Subtraction) is a useful way to remember the rules:

- Parentheses have the highest precedence and can be used to force an expression to evaluate in the order you want. Since expressions in parentheses are evaluated first, 2 * (3-1)is 4, and (1+1)**(5-2) is 8.
- You can also use parentheses to make an expression easier to read, as in (minute * 100) / 60, even if it doesn't change the result.
- Exponentiation has the next highest precedence, so 1 + 2**3 is 9, not 27, and 2 *3**2 is 18, not 36.
- Multiplication and Division have higher precedence than Addition and Subtraction. So 2*3-1 is 5, not 4, and 6+4/2 is 8, not 5.
- Operators with the same precedence are evaluated from left to right (except exponentiation).

Example:

a=9-12/3+3*2-1	**A=2*3+4%5-3/2+6**	**find m=?**	**a=2*3+4%5-3//2+6**
a=?	A=6+4%5-3/2+6	**m=-43||8&&0||-2**	a=6+4-1+6
a=9-4+3*2-1	A=6+4-3/2+6	m=-43||0||-2	a=10-1+6
a=9-4+6-1	A=6+4-1+6	m=1||-2	a=15
a=5+6-1	A=10-1+6	m=1	
a=11-1	A=9+6		
a=10	A=15		

BASIC PYTHON PROGRAMS

1. Addition of two numbers

a=eval(input(“enter first no”))
b=eval(input(“enter second no”))
c=a+b
print(“the sum is “,c)
	Output
	enter first no
	5
	enter second no
	6
	the sum is 11

2. Area of rectangle

l=eval(input(“enter the length of rectangle”))
b=eval(input(“enter the breath of rectangle”))
a=l*b
print(a)
	Output
	enter the length of rectangle 5
	enter the breath of rectangle 6
	30

3. Area & circumference of a circle

r=eval(input(“enter the radius of circle”)) a=3.14*r*r c=2*3.14*r print(“the area of circle”,a) print(“the circumference of circle”,c)	**Output** enter the radius of circle4 the area of circle 50.24 the circumference of circle 25.12

4. Calculate simple interest

p=eval(input(“enter principle amount”))
n=eval(input(“enter no of years”))
r=eval(input(“enter rate of interest”))
si=p*n*r/100
print(“simple interest is”,si)
	Output
	enter principle amount 5000
	enter no of years 4
	enter rate of interest6
	simple interest is 1200.0

5. Calculate engineering cutoff

p=eval(input(“enter physics marks”))	**Output**
c=eval(input(“enter chemistry marks”))	enter physics marks 100
m=eval(input(“enter maths marks”))	enter chemistry marks 99
cutoff=(p/4+c/4+m/2)	enter maths marks 96
print(“cutoff =”,cutoff)	cutoff = 97.75

6. Check voting eligibility

age=eval(input(“enter ur age”))	**output**
If(age>=18):	Enter ur age
print(“eligible for voting”)	19
else:	Eligible for voting
print(“not eligible for voting”)	

7. Find the greatest of three numbers

a=eval(input(“enter the value of a”))	**output**
b=eval(input(“enter the value of b”))	enter the value of a 9
c=eval(input(“enter the value of c”))	enter the value of a 1
if(a>b):	enter the value of a 8
if(a>c):	the greatest no is 9
print(“the greatest no is”,a)	
else:	
print(“the greatest no is”,c)	
else:	
if(b>c):	
print(“the greatest no is”,b)	
else:	
print(“the greatest no is”,c)	

8. Print n natural numbers

i=1
while(i<=5):
print(i)
i=i+1
	Output
	1
	2
	3
	4
	5

9. Print n odd numbers

i=2
while(i<=10):
print(i)
i=i+2
	Output
	2
	4
	6
	8
	10

10. Print n even numbers

i=1
while(i<=10):
print(i)
i=i+2
	Output
	1
	3
	5
	7
	9

11. Print n squares of numbers

i=1
while(i<=5):
print(i*i)
i=i+1
	Output
	1
	4
	9
	16
	25

12. Print n cubes numbers

i=1
while(i<=3):
print(i*i*i)
i=i+1
	Output
	1
	8
	27

13. Find the sum of n numbers

```
i=1
sum=0
while(i<=10):
sum=sum+i
i=i+1
print(sum)
```

Output

```
55
```

14. Factorial of n numbers/product of n numbers

```
i=1
product=1
while(i<=10):
product=product*i
i=i+1
print(product)
```

Output

```
3628800
```

15. Swap two values of variables

```
a=eval(input("enter a value"))
b=eval(input("enter b value"))
c=a
a=b
b=c
print("a=",a,"b=",b)
```

Output

```
enter a value3
enter b value5
a= 5 b= 3
```

16. Convert the temperature

```
c=eval(input("enter temperature in centigrade"))
f=(1.8*c)+32
print("the temperature in Fahrenheit is",f)
f=eval(input("enter temp in Fahrenheit"))
c=(f-32)/1.8
print("the temperature in centigrade is",c)
```

Output

```
enter temperature in centigrade 37
the temperature in Fahrenheit is 98.6
enter temp in Fahrenheit 100
the temperature in centigrade is 37.77
```

17. Program for a basic calculator

```
a=eval(input("enter a value"))
b=eval(input("enter b value"))
c=a+b
print("the sum is",c)
a=eval(input("enter a value"))
b=eval(input("enter b value"))
c=a-b
print("the diff is",c)
a=eval(input("enter a value"))
b=eval(input("enter b value"))
c=a*b
print("the mul is",c)
a=eval(input("enter a value"))
b=eval(input("enter b value"))
c=a/b
print("the div is",c)
```

Output

```
enter a value 10
enter b value 10
the sum is 20
enter a value 10
enter b value 10
the diff is 0
enter a value 10
enter b value 10
the mul is 100
enter a value 10
enter b value 10
the div is 1
```