**Operators and Control statements**

* Operator is a symbol that performs operation on one or more operands.
* The member on which operator operates is called the operand.
* In the expression a = 5+9,

a, 5,9 are operands

=, + are operators

Python supports the following operators:

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

**Arithmetic operators:** Arithmetic operators are used to perform mathematical operations like addition, subtraction, multiplication etc.

|  |  |  |
| --- | --- | --- |
| Operator | Meaning | Example |
| + | Add 2 operands | X+Y |
| - | Subtract right from left | X-Y |
| \* | Multiply 2 operands | X\*Y |
| / | Divide left with right | X/Y |
| % | Divide and returns Remainder | X%Y |
| // | Floor division - division that results into whole number adjusted to the left in the number line | X//Y |
| \*\* | Exponent - left operand raised to the power of right | X\*\*Y |

**>>> 5+3**

**8**

**>>> 5+3-2 # +, - having same priority , left to right evaluation**

**6**

**>>> 5\*3/2 # \*, / , % having same priority**

**7.5**

**>>> 5+3\*2 # \* having higher priority than +**

**11**

**>>> (5+3)\*2 # () having higher priority than all arithmetic operator**

**16**

**>>> 5/2 # gives 2.5**

**2.5**

**>>> 5%2 # it performs operation only on integers and returns remainder**

**1**

**Floor division(//) : After performing division operation, it display the floor value of result**

**For example**

1. **2.5 floor value is 2**
2. **1.8 floor value is 1**
3. **3.0 floor value is 3**

>>> 5/2

2.5

>>> 5//2

2

Exponent:

|  |  |
| --- | --- |
| \*\* | Exponent - left operand raised to the power of right |

>>> 2\*\*3

8

>>> 4\*\*3

64

>>> 2.5\*\*2

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

|  |  |  |
| --- | --- | --- |
| **Operator** | **Meaning** | **Example** |
| > | Greater than | x > y |
| < | Less than | x < y |
| == | Equal to | x == y |
| != | Not equal to | x != y |
| >= | Greater than or equal to | x >= y |
| <= | Less than or equal to | x <= y |

>>> 5>3

True

>>> 3!=3

False

**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 |

**And examples:**

>>> True and True

True

>>> 5>3 and 3>2

True

>>> True and False

False

>>> False and True

False

>>> False and False

False

>>> 5>3 and 5!=5

False

**Or examples:**

>>> False or False

False

>>> False or True

True

>>> True or False

True

>>> True or True

True

>>> 3>5 or 5>2

Note : Among operators (and,or), and execute first.

>>> True and True

True

>>> True and False or True

True

>>> True or True and False

True

>>> True or False and False

True

Not operator :

>>> not 5

False

>>> not 0

True

>>> not 'python'

False

>>> not ''

True

>>> not 5

False

>>> not 5 and 4

False

>>> not 5 or 4

4

>>> 3 and 4

4

>>> 5 or 3 and 4

5

>>> 5 or 4

5

>>> not 5 or 3 and 4

4

>>> 0 or 4

4

**Not examples:**

>>> not True

False

>>> not False

True

>>> not 5>3

False

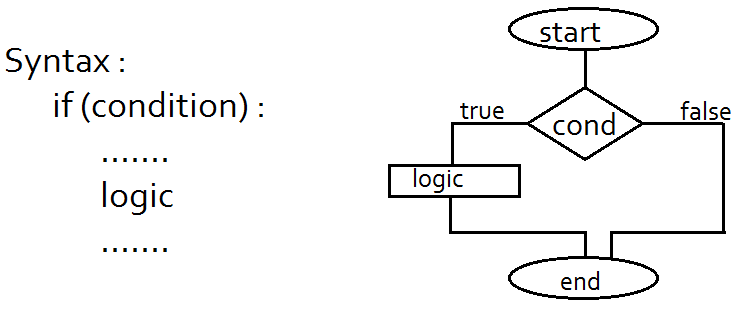
>>> not 3!=3

True

Control statements:

* Used to control the execution flow of logic.
* Used to execute block of instructions randomly and repeatedly.

If – block:



>>> x=2

>>> y=3

>>> if(x>y):

print('x is greater than y')

>>> if(x<y):

print('x is less than y')

x is less than y

>>> if True:

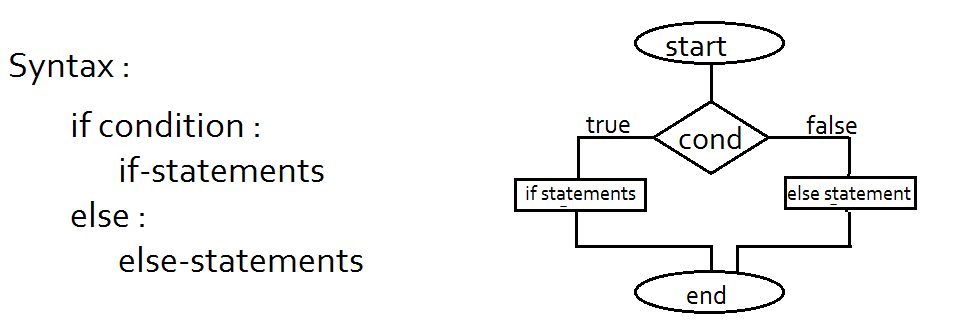
print('It is true')

It is true

>>> if False:

print('It will not execute')

if-else block:



>>> if True:

print('if block')

else:

print('else block')

>>> if False:

print('if block')

else:

print('else block')

Even or Odd number program:

def isEven(num):

if num%2==0 :

print(num , ' is Even number')

else:

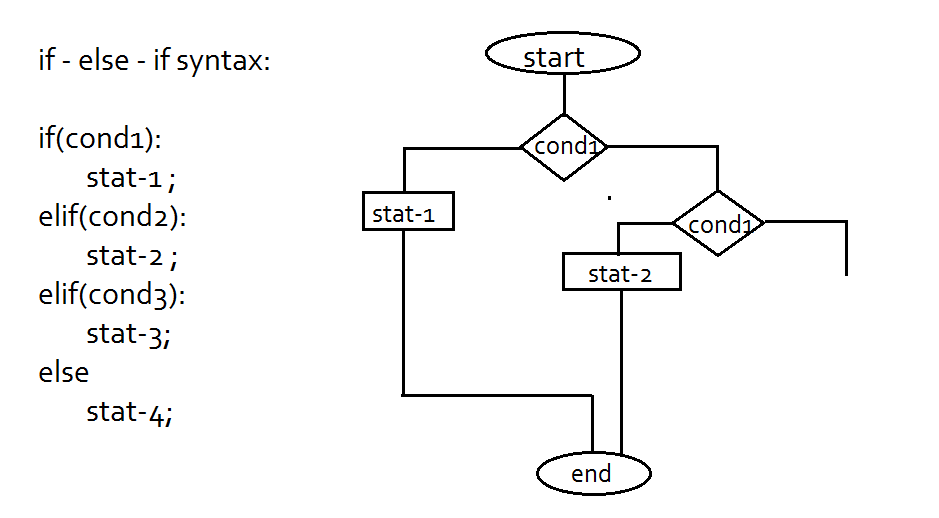
print(num , ' is Odd number')

return

isEven(4)

isEven(3)

if – else – if block :



>>> if True:

print('stat1')

elif True:

print('stat2')

elif True:

print('stat3')

output: stat1

>>> if False:

print('stat1')

elif True:

print('stat2')

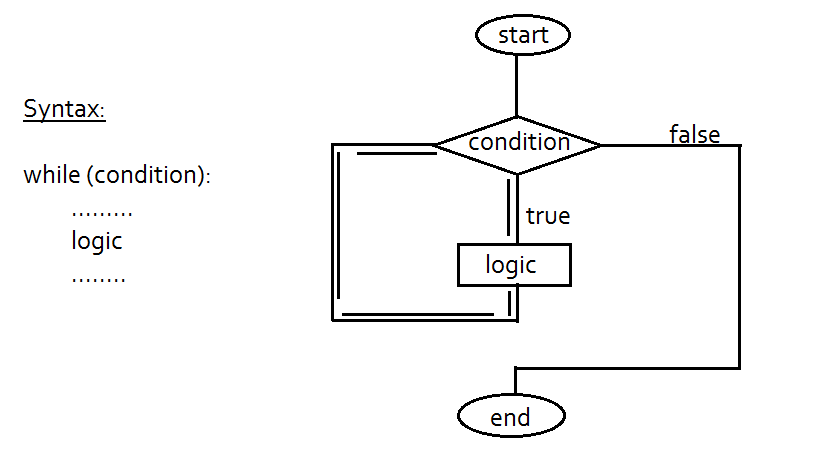
elif True:

print('stat3')

output : stat2

While loop:

Execute a block of instructions repeatedly as long as the given condition is true.



Program to print 1 to 10 numbers:

From shell:

>>> x=1

>>> while x<=10:

print('x value : ',x)

x=x+1

output:

x value : 1

x value : 2

x value : 3

x value : 4

x value : 5

x value : 6

x value : 7

x value : 8

x value : 9

x value : 10

Using functions:

def printEven(n):

x=1

while(x<=n):

print('x value : ',x)

x=x+1

return

limit = input("Enter limit : ") # input() returns value in String format.

n = int(limit)

printEven(n)

Program to print even numbers:

def printEven(n):

x=1

while(x<=n):

if(x%2==0):

print('x value : ',x)

x=x+1

return

limit = input("Enter limit : ") # input() returns value in String format.

n = int(limit)

printEven(n)

for loop:

* It is used to execute a set of instructions repeatedly as long as the given condition is true.
* For-loop using range() function.

Range(stop): it takes upper bound to repeat a loop.

It iterates the loop from 0 to upper-bound-1.

For example Range(10) : prints 0 to 9

>>> for x in range(10):

print('x value : ',x)

x value : 0

x value : 1

x value : 2

x value : 3

x value : 4

x value : 5

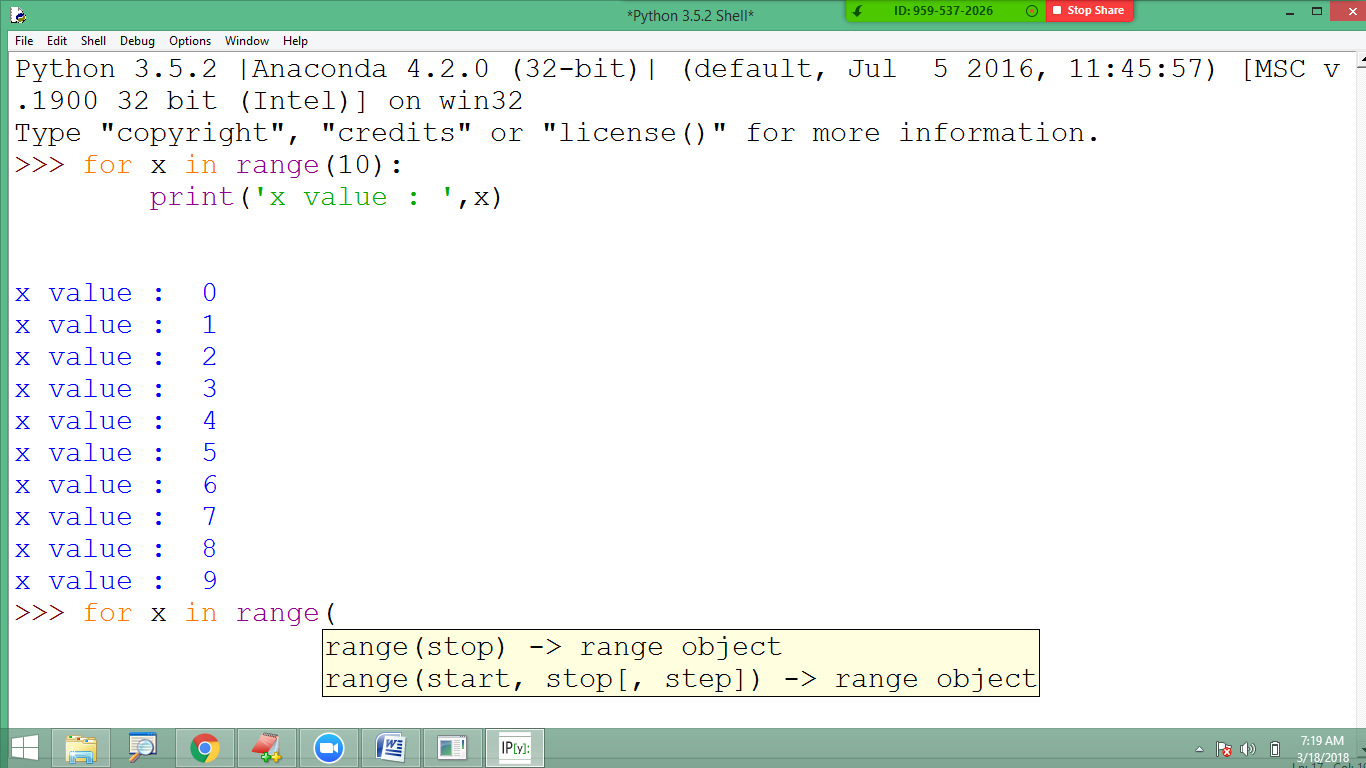
x value : 6

x value : 7

x value : 8

x value : 9

range(start,stop): collects lower bound and upper bound values to repeat the loop



>>> for x in range(5,11):

print('x value : ',x)

x value : 5

x value : 6

x value : 7

x value : 8

x value : 9

x value : 10

Printing alternate values:

>>> for x in range(1,11,2):

print('x value : ',x)

x value : 1

x value : 3

x value : 5

x value : 7

x value : 9

Reverse the loop:

>>> for x in range(10,0):

print('x value : ',x)

Output : Blank🡪 by default it increase the value.

Hence we can reverse the loop by giving step-value as -1

>>> for x in range(10,0,-1):

print('x value : ',x)

x value : 10

x value : 9

x value : 8

x value : 7

x value : 6

x value : 5

x value : 4

x value : 3

x value : 2

x value : 1

Note: for-loop is mainly used to iterate (process) the elements of data structure models such as sets, tuples, dictionaries ….

For example….

>>> set = [10,20,30,40,50]

>>> for ele in set:

print('element : ', ele)

element : 10

element : 20

element : 30

element : 40

element : 50

break:

* It is used to break the loop.
* It can be used inside while loop or for loop

>>> for x in range(1,10):

if(x==5):

break

print('x value : ', x)

x value : 1

x value : 2

x value : 3

x value : 4

Continue:

* It is used to terminate a particular iteration in the loop execution.
* It continues with the next iteration in the loop after terminating current iteration.

Following program prints 1 to 9 except 5:

for x in range(1,10):

if(x==5):

continue

print('x value : ', x)

x value : 1

x value : 2

x value : 3

x value : 4

x value : 6

x value : 7

x value : 8

x value : 9

>>> for x in range(1,10):

if(x==4 or x==7):

continue

print('x value : ', x)

x value : 1

x value : 2

x value : 3

x value : 5

x value : 6

x value : 8

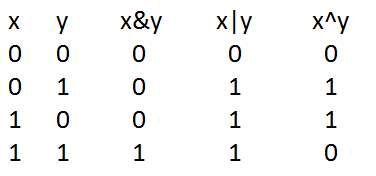
x value : 9

**Bitwise operators:**

* Bitwise operators act on operands as if they were string of binary digits. It operates bit by bit, hence the name.
* For example, 2 is 10 in binary and 7 is 111.
* In the table below: 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) |

Bitwise truth table:



>>> x=15

>>> y=8

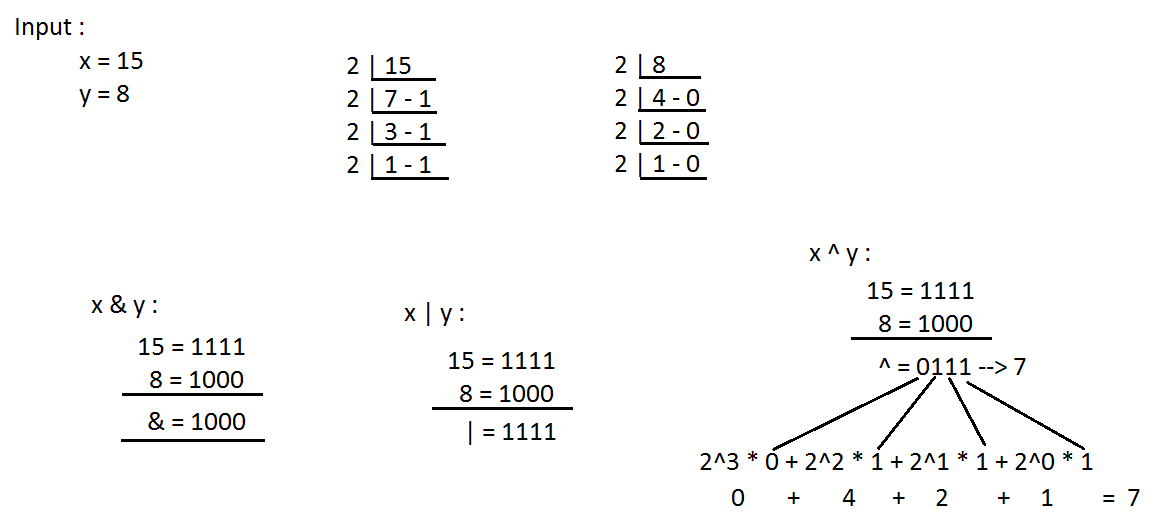
>>> x&y

8

>>> x|y

15

>>> x^y



Shift operators: These are used to move the bits in the memory either to right side or to left side.

Right shift (>>)

Left shift (<<)

>>> x=8

>>> x>>2

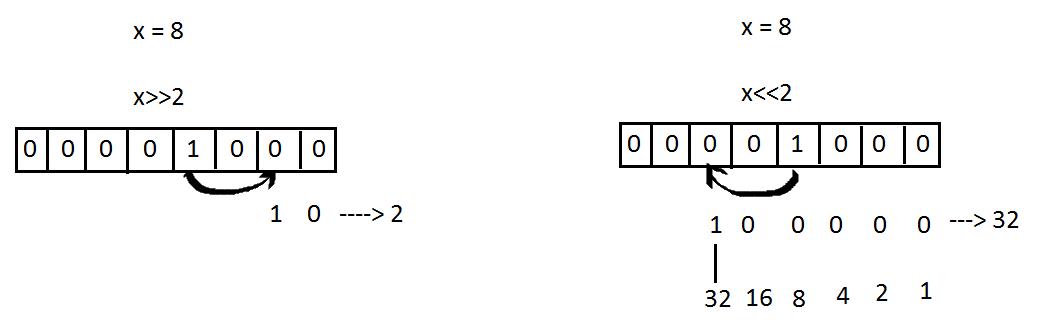
2

>>> x<<2

32

Right shift: n/2^s 🡪 8/2^2 🡪 8/4 🡪 2

Left shift : n\*2^s 🡪 8\*2^2 🡪 8\*4 🡪 32



**Assignment operators:**

* Assignment operators are used in Python to assign values to variables.
* a = 5 is a simple assignment operator that assigns the value 5 on the right to the variable a on the left.
* There are various compound operators in Python like a += 5 that adds to the variable and later assigns the same. It is equivalent to a = a + 5.

|  |  |  |
| --- | --- | --- |
|  | | |
| Operator | Example | Equivatent to |
| = | x = 5 | x = 5 |
| += | x += 5 | x = x + 5 |
| -= | x -= 5 | x = x - 5 |
| \*= | x \*= 5 | x = x \* 5 |
| /= | x /= 5 | x = x / 5 |
| %= | x %= 5 | x = x % 5 |
| //= | x //= 5 | x = x // 5 |
| \*\*= | x \*\*= 5 | x = x \*\* 5 |
| &= | x &= 5 | x = x & 5 |
| |= | x |= 5 | x = x | 5 |
| ^= | x ^= 5 | x = x ^ 5 |
| >>= | x >>= 5 | x = x >> 5 |
| <<= | x <<= 5 | x = x << 5 |
|  |  |  |

>>> balance = 3000

>>> balance = balance + 2000

>>> print(balance)

5000

>>> balance = 3000

>>> balance += 2000

>>> print(balance)

5000

**Identity operators:**

* is and is not are the identity operators in Python.
* They are used to check if two values (or variables) are located on the same part of the memory.
* Two variables that are equal does not imply that they are identical.

|  |  |  |
| --- | --- | --- |
|  | | |
| 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 Code:**

>>> x = 5

>>> y = 5

>>> x is y

True

>>> x = [10,20]

>>> y = [30,40]

>>> x is y

False

>>> x is not y

True

**Membership operators:**

* in and not in are the membership operators in Python. They are used to test whether a value or variable is found in a sequence ([string](https://www.programiz.com/python-programming/string), [list](https://www.programiz.com/python-programming/list), [tuple](https://www.programiz.com/python-programming/tuple), [set](https://www.programiz.com/python-programming/set) and [dictionary](https://www.programiz.com/python-programming/dictionary)).
* In a dictionary we can only test for presence of key, not the value.

|  |  |  |
| --- | --- | --- |
| **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 Code:**

>>> s = 'Python'

>>> 'y' in s

True

>>> 'py' in s

False

>>> 'Py' in s

True

**Operator precedence:**

* The operator precedence determines which operators need to be evaluated first.
* To avoid ambiguity in values, precedence operators are necessary.
* Just like in normal multiplication method, multiplication has a higher precedence than addition.
* For example in 3+ 4\*5, the answer is 23, to change the order of precedence we use a square bracket (3+4)\*5, now the answer is 35.
* Precedence operator used in Python are (unary + - ~, \*\*, \* / %, + - , &) etc.

|  |  |
| --- | --- |
| **Operator precedence rule in Python** | |
| **Operators** | **Meaning** |
| () | Parentheses |
| \*\* | Exponent |
| +x, -x, ~x | Unary plus, Unary minus, Bitwise NOT |
| \*, /, //, % | Multiplication, Division, Floor division, Modulus |
| +, - | Addition, Subtraction |
| <<, >> | Bitwise shift operators |
| & | Bitwise AND |
| ^ | Bitwise XOR |
| | | Bitwise OR |
| ==, !=, >, >=, <, <=, is, is not, in, not in | Comparisions, Identity, Membership operators |
| not | Logical NOT |
| and | Logical AND |
| or | Logical OR |

**Associativity of Python Operators:**

* We can see in the above table that more than one operator exists in the same group. These operators have the same precedence.
* When two operators have the same precedence, associativity helps to determine which the order of operations.
* Associativity is the order in which an expression is evaluated that has multiple operator of the same precedence. Almost all the operators have left-to-right associativity.
* For example, multiplication and floor division have the same precedence. Hence, if both of them are present in an expression, left one is evaluates first.

**Example code:**

# Left-right associativity

# Output: 3

print(5 \* 2 // 3)

# Shows left-right associativity

# Output: 0

print(5 \* (2 // 3))

**The following table lists all operators from highest precedence to lowest:**

|  |  |
| --- | --- |
| **Operator** | **Description** |
| \*\* | Exponentiation (raise to the power) |
| ~ + - | Complement, unary plus and minus |
| \* / % // | Multiply, divide, modulo and floor division |
| + - | Addition and subtraction |
| >> << | Right and left bitwise shift |
| & | Bitwise 'AND'td> |
| ^ | | 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 |

**Example Code:**

#!/usr/bin/python

a = 20

b = 10

c = 15

d = 5

e = 0

e = (a + b) \* c / d #( 30 \* 15 ) / 5

print "Value of (a + b) \* c / d is ", e

e = ((a + b) \* c) / d # (30 \* 15 ) / 5

print "Value of ((a + b) \* c) / d is ", e

e = (a + b) \* (c / d); # (30) \* (15/5)

print "Value of (a + b) \* (c / d) is ", e

e = a + (b \* c) / d; # 20 + (150/5)

print "Value of a + (b \* c) / d is ", e