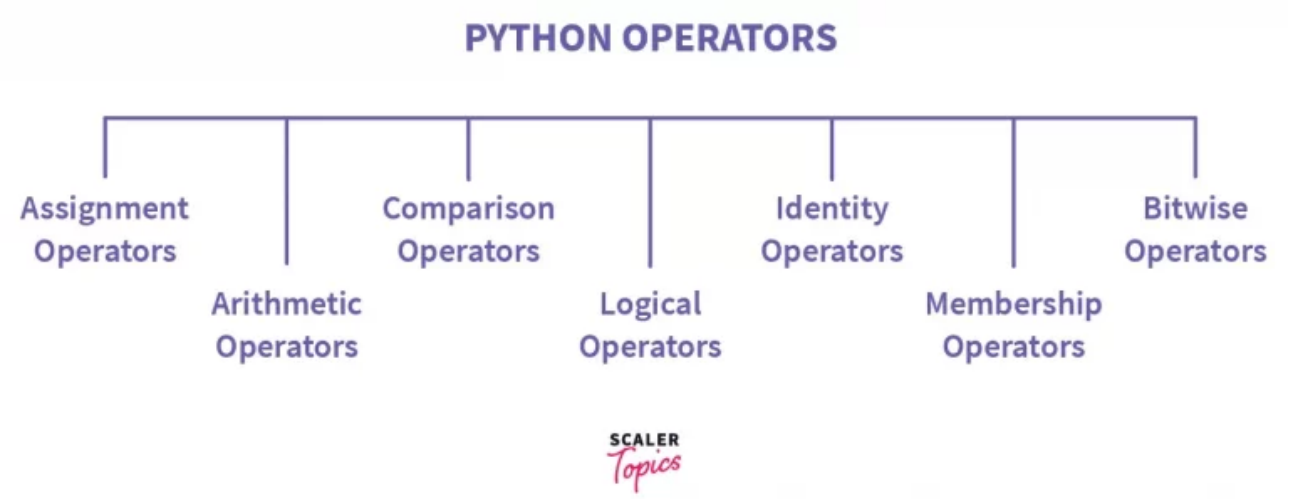
Python Operators

Python divides the operators in the following groups:

- Arithmetic operators
- Assignment operators
- Comparison operators
- Logical operators
- Identity operators
- Membership operators
- Bitwise operators ..



Arithmetic operators are used to perform mathematical operations like addition, subtraction, multiplication and division.

There are 7 arithmetic operators in Python :

1. Addition
2. Subtraction
3. Multiplication
4. Division
5. Modulus
6. Exponentiation
7. Floor division

Operator	Operation	Example	Description
+	Addition	a+b	Adds a and b
-	Subtraction	a-b	Subtracts b from a
*	Multiplication	a*b	Multiplies a and b
/	Division	a/b	Divides a and b
%	Modulus	a%b	Gives remainder after dividing a from b
**	Exponential	3**2=9	Ignores the decimal value if present
//	Floor Division	15/2=7	Gives the result of 3 to the power of 2

1. Addition Operator : In Python, + is the addition operator. It is used to add 2 values. Example :

```
val1 = 2
val2 = 3

# using the addition operator
res = val1 + val2
print(res)
```

```
5
```

```
print("Addition of two num")
val1=int(input("Enter the number1:"))
val2=int(input("Enter the number2:"))
print("Result : ",(val1+val2))
```

```
Addition of two num
Enter the number1:10
Enter the number2:20
Result :  30
```

```
val3=input("Enter the number1:")
```

```
Enter the number1:10
```

```
type(val3)
```

```
str
```

```
val3
```

```
'10'
```

```
print(val3)
```

```
10
```

```
val1
```

```
10
```

```
val1=int(input("Enter number1\n"))
val2=int(input("Enter number2\n"))
print("Result",(val1+val2))
```

```
Enter number1
20
Enter number2
20
Result 40
```

2.Subtraction Operator : In Python, – is the subtraction operator. It is used to subtract the second value from the first value.Example :

```
val1 = 2.5;val2 = 1.234

# using the subtraction operator
res = val1 - val2
print(res)
```

```
1.266
```

3. Multiplication Operator : In Python, * is the multiplication operator. It is used to find the product of 2 values. Example :

```
val1=2
val2=3.6

# using the multiplication operator
res = val1 * val2
print(res)
```

```
7.2
```

```
print("Enter the first number")
n1=int(input())
print("Enter the second number")
n2=int(input())
s=n1+n2
d=n1-n2
print("sum of two number is ",s)
print("difference between two number is ",d)
```

```
Enter the first number
10
Enter the second number
20
sum of two number is  30
difference between two number is  -10
```

```
a = -10
b = -14
c =-12
print(max(a, b, c))
```

```
-10
```

Simple Interest = (P x T x R)/100 Where, P is the principal amount , T is the time and R is the rate

```

```

```
P = int(input("Enter Amount"))
R = 2.5
T = 1
# simple interest
SI = (P * R * T) / 100
print("simple interest is", SI)
```

```
Enter Amount1000
simple interest is 25.0
```

Celsius To Fahrenheit and Fahrenheit to Celsius conversion formulas

Celsius = (Fahrenheit – 32) * 5/9 , Fahrenheit = (Celsius * 9/5) + 32

```
fahrenheit=75
celsius=(fahrenheit-32)*5/9
celsius
```

```
23.88888888888889
```

4. Division Operator : In Python, / is the division operator. It is used to find the quotient when first operand is divided by the second. Example :

```
val1 = 322222222222222
val2 = 25

# using the division operator
res = val1 // val2
res1=val1/val2
print(res)
print(res1)
```

```
128888888888888
128888888888888.88
```

```
# A Python program to demonstrate the use of
# "//" for integers
print (5//2)
print (-5//2)
```

```
2
-3
```

```
print (5.0//2)
print (-5.0//2)
```

```
2.0
-3.0
```

```
print (5//2)
print (-5//2)
print (5.0//2)
print (-5.0//2)
```

```
2
-3
2.0
-3.0
```

5. Modulus Operator : In Python, % is the modulus operator. It is used to find the remainder when first operand is divided by the second. Example :

```
val1 = 3
val2 = 3

# using the modulus operator
res = val1 % val2
print(res)
```

```
0
```

6. Exponentiation Operator : In Python, ** is the exponentiation operator. It is used to raise the first operand to power of second. Example :

```
val1 = 3.2
val2 = 3

# using the exponentiation operator
res = val1 ** val2
print(res)
```

```
32.76800000000001
```

7. Floor division : In Python, // is used to conduct the floor division. It is used to find the floorof the quotient when first operand is divided by the second. Example :

```
val1 = 3
val2 = 2

# using the floor division
res = val1 // val2
print(res)
```

```
1
```

```
# Examples of Arithmetic Operator
a = 9
b = 4

# Addition of numbers
add = a + b

# Subtraction of numbers
sub = a - b

# Multiplication of number
mul = a * b
```

```
# Division(float) of number
div1 = a / b

# Division(floor) of number
div2 = a // b

# Modulo of both number
mod = a % b

# Power
p = a ** b

# print results
print(add)
print(sub)
print(mul)
print(div1)
print(div2)
print(mod)
print(p)
```

```
13
5
36
2.25
2
1
6561
```

Assignment Operators in Python

- Operators are used to perform operations on values and variables. These are the special symbols that carry out arithmetic, logical, bitwise computations. The value the operator operates on is known as **Operand**.
- Here, we will cover Assignment Operators in Python. So, Assignment Operators are used to assigning values to variables.

Operator	Description	Example
=	Assigns right side operands to left variable.	>>>'number'=10
+=	Added and assign back the result to left operand.	>>>'number'+=20 # 'number'='number'+20
-=	Subtracted and assign back the result to left operand.	>>>'number'-=5
*=	Multiplied and assign back the result to left operand.	>>>'number'*=5
/=	Divide and assign back the result to left operand.	>>>'number'/=2
%=	Taken modulus (Remainder) using two operands and assign the result to left operand.	>>>'number'%=3
**=	Performed exponential (power) calculation on operators and assign value to the left operand.	>>>'number'**=2
**=	Performed exponential (power) calculation on operators and assign value to the left operand.	>>>'number'**=2
**=	Performed floor division on operators and assign value to the left operand.	>>>'number'//=3

SCALER Topics

1) Assign: This operator is used to assign the value of the right side of the expression to the left side operand.

Syntax:

```
a = 3
b = 5

c = a + b
```

```
# Output
print(c)
```

```
8
```

2) Add and Assign: This operator is used to add the right side operand with the left side operand and then assigning the result to the left operand.

Syntax:

```
a = 3
b = 5

# a = a + b
a += b

# Output
print(a)
```

```
8
```

3) Subtract and Assign: This operator is used to subtract the right operand from the left operand and then assigning the result to the left operand.

Syntax:

```
a = 3
b = 5

# a = a - b
a -= b

# Output
print(a)
```

```
-2
```

4) Multiply and Assign: This operator is used to multiply the right operand with the left operand and then assigning the result to the left operand.

Syntax:

```
a = 3
b = 5

# a = a * b
a *= b

# Output
print(a)
```

```
15
```

5) Divide and Assign: This operator is used to divide the left operand with the right operand and then assigning the result to the left operand.

```
a = 3
b = 5

# a = a / b
a /= b

# Output
print(a)
```

```
0.6
```

6) Modulus and Assign: This operator is used to take the modulus using the left and the right operands and then assigning the result to the left operand.

```
a = 13
b = 5
```

```
# a = a % b
a %= b

# Output
print(a)
```

```
3
```

7) Divide (floor) and Assign: This operator is used to divide the left operand with the right operand and then assigning the result(floor) to the left operand.

```
a = 13
b = 5

# a = a // b
a //= b

# Output
print(a)
```

```
2
```

8) Exponent and Assign: This operator is used to calculate the exponent(raise power) value using operands and then assigning the result to the left operand.

```
a = 3
b = 5

# a = a ** b
a **= b

# Output
print(a)
```

9) Bitwise AND and Assign: This operator is used to perform Bitwise AND on both operands and then assigning the result to the left operand.

```
a = 3
b = 5

# a = a & b
a &= b

# Output
print(a)
```

```
1
```

```
# Examples of Assignment Operators
a = 10

# Assign value
b = a
print(b)

# Add and assign value
b += a
print(b)

# Subtract and assign value
b -= a
print(b)

# multiply and assign
b *= a
```

```
print(b)

# bitwise lishift operator
b <<= a
print(b)
```

```
10
20
10
100
102400
```

```
# Python code to swap two numbers
# without using another variable



x = 5
y = 7

print ("Before swapping: ")
print("Value of x : ", x, " and y : ", y)

# code to swap 'x' and 'y'
x, y = y, x

print ("After swapping: ")
print("Value of x : ", x, " and y : ", y)
```

```
Before swapping:
Value of x :  5  and y :  7
After swapping:
Value of x :  7  and y :  5
```

Relational operators are used for comparing the values. It either returns True or False according to the condition. These operators are also known as **Comparison Operators**.

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

SCALER Topics

1) Greater than: This operator returns True if the left operand is greater than the right operand.

Syntax: x > y

```
a = 9
b = 5

# Output
print(a > b)
```

```
True
```

2) Less than: This operator returns True if the left operand is less than the right operand.

```
a = 9
b = 5
```

```
# Output
print(a < b)
```

```
False
```

3) Equal to: This operator returns True if both the operands are equal i.e. if both the left and the right operand are equal to each other.

```
a = 9
b = 5

# Output
print(a == b)
```

```
False
```

4) Not equal to: This operator returns True if both the operands are not equal.

```
a = 9
b = 5

# Output
print(a != b)
```

```
True
```

5) Greater than or equal to: This operator returns True if the left operand is greater than or equal to the right operand.

```
a = 9
b = 5

# Output
print(a >= b)
```

```
True
```

6) Less than or equal to: This operator returns True if the left operand is less than or equal to the right operand.

```
a = 9
b = 5

# Output
print(a <= b)
```

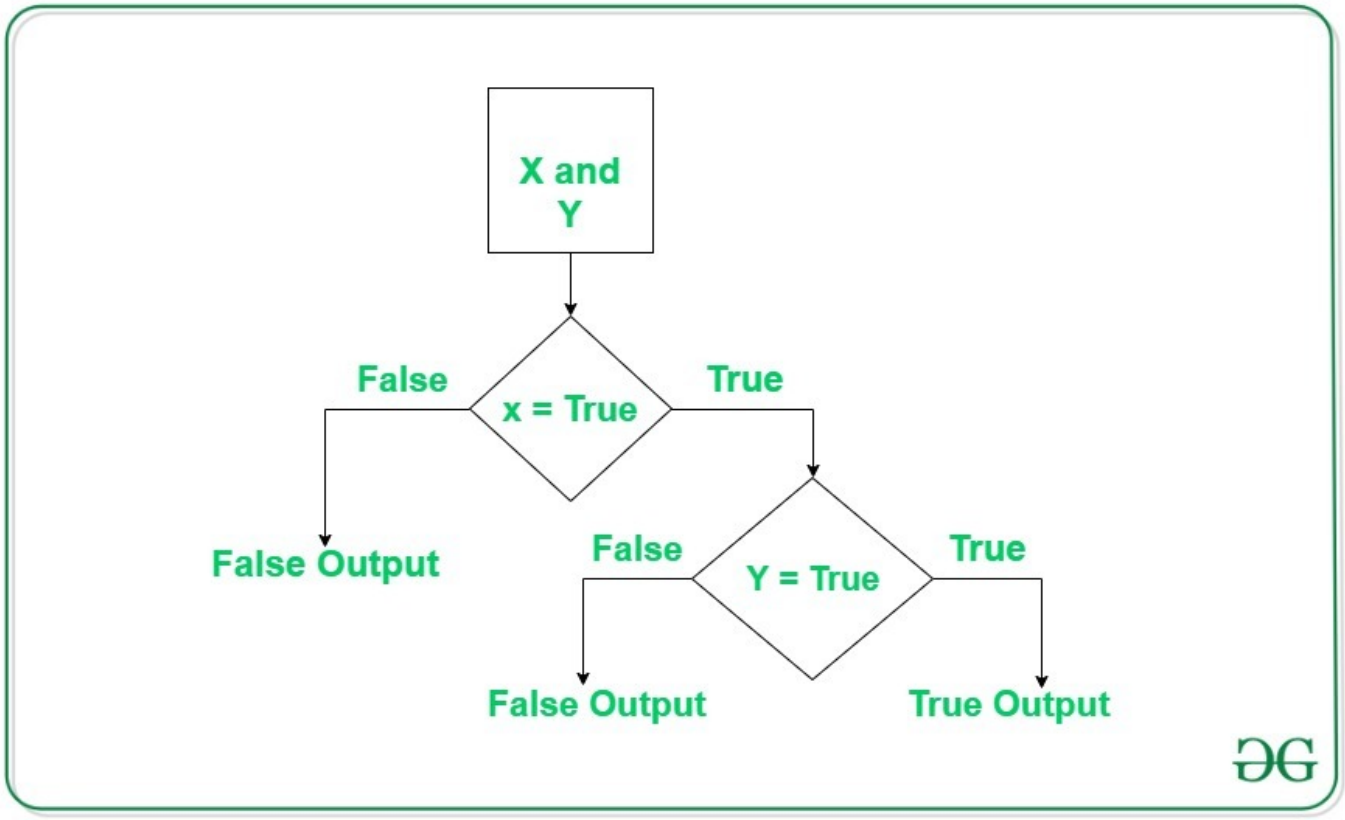
```
False
```

Logical operators

In Python, Logical operators are used on conditional statements (either True or False). They perform **Logical AND**, **Logical OR** and **Logical NOT** operations.

Operator	Example	Description
and	x < 5 and x < 10	Returns True if both statements are true
or	x < 5 or x < 4	Returns True if one of the statements is true
not	not(x < 5 and x < 10)	Reverse the result, returns False if the result is not true

Logical AND operator



Logical operator returns True if both the operands are True else it returns False.

```
a = 12
b = 26
c = 4

if a > b and a > c:
    print("Number a is larger")

if b > a and b > c:
    print("Number b is larger")

if c > a and c > b:
    print("Number c is larger")
```

```
Number b is larger
```

```
a = 10

if (a == 10 and "Hello"):
    print("a has value zero(0)")
else:
    print("a is not equal to zero")
```

```
a has value zero(0)
```

```
# Python program to demonstrate
# logical and operator

a = 10
b = 10
c = -10

if a > 0 and b > 0:
    print("The numbers are greater than 0")

if a > 0 and b > 0 and c > 0:
    print("The numbers are greater than 0")
else:
    print("Atleast one number is not greater
```

```
The numbers are greater than 0
Atleast one number is not greater than 0
```

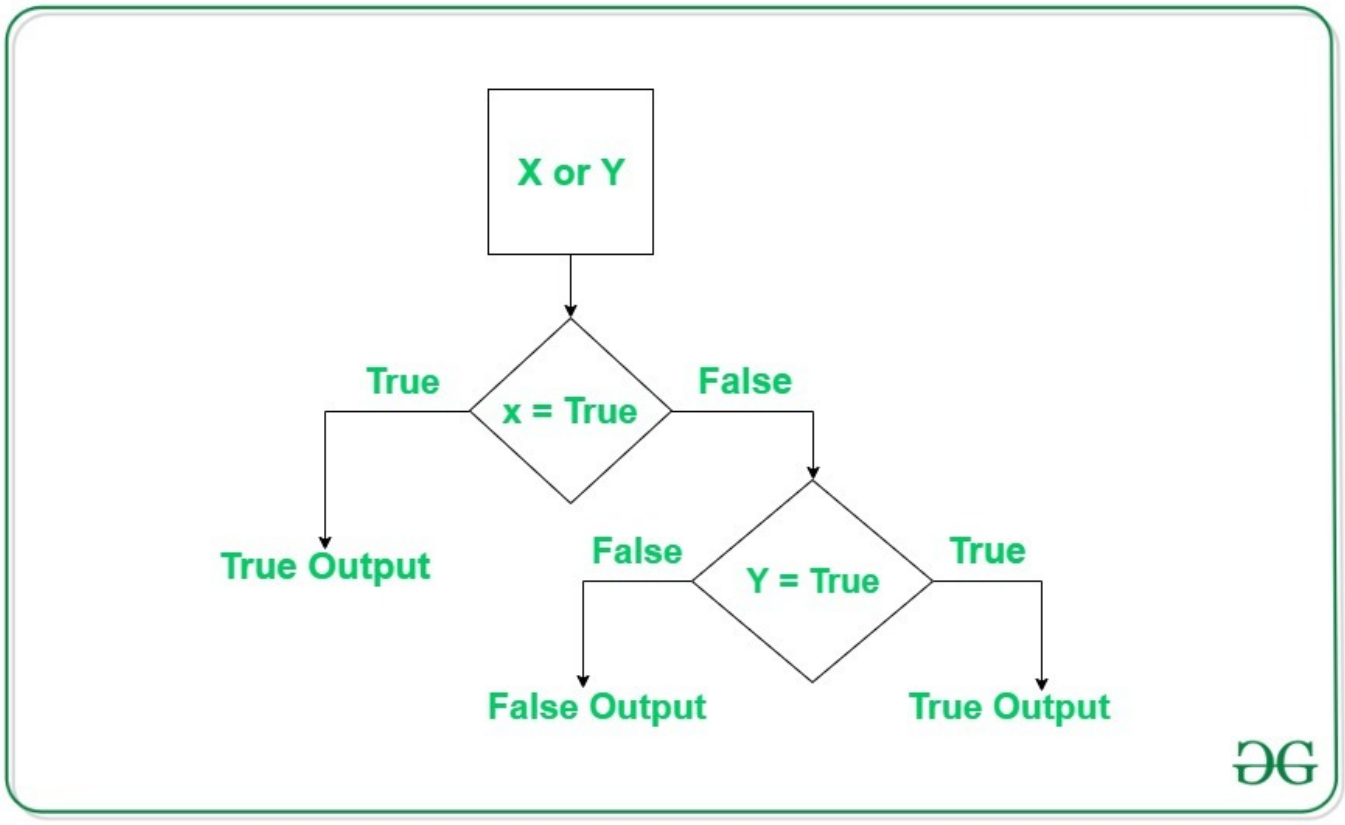
```
a = 10
b = 12
c = 0

if a and b and c:
    print("All the numbers have boolean value
else:
```

```
    print("Atleast one number has boolean val
```

```
Atleast one number has boolean value as False
```

Logical or operator Logical or operator returns True *if either *of the operands is True.




```
a = 10
b = -5

if a < 0 or b < 0:
  print("Their product will be negative")
else:
  print("Their product will be positive")
```

```
Their product will be negative
```

```
a = 10

if (a == 0 or "GeeksForGeeks"):
  print("Is Awesome")
```

```
else:
  ("Try Again!")
```

```
Is Awesome
```

```
a = 10
b = -10
c = 0

if a > 0 or b > 0:
    print("Either of the number is greater th
else:
    print("No number is greater than 0")

if b > 0 or c > 0:
    print("Either of the number is greater th
else:
    print("No number is greater than 0")
```

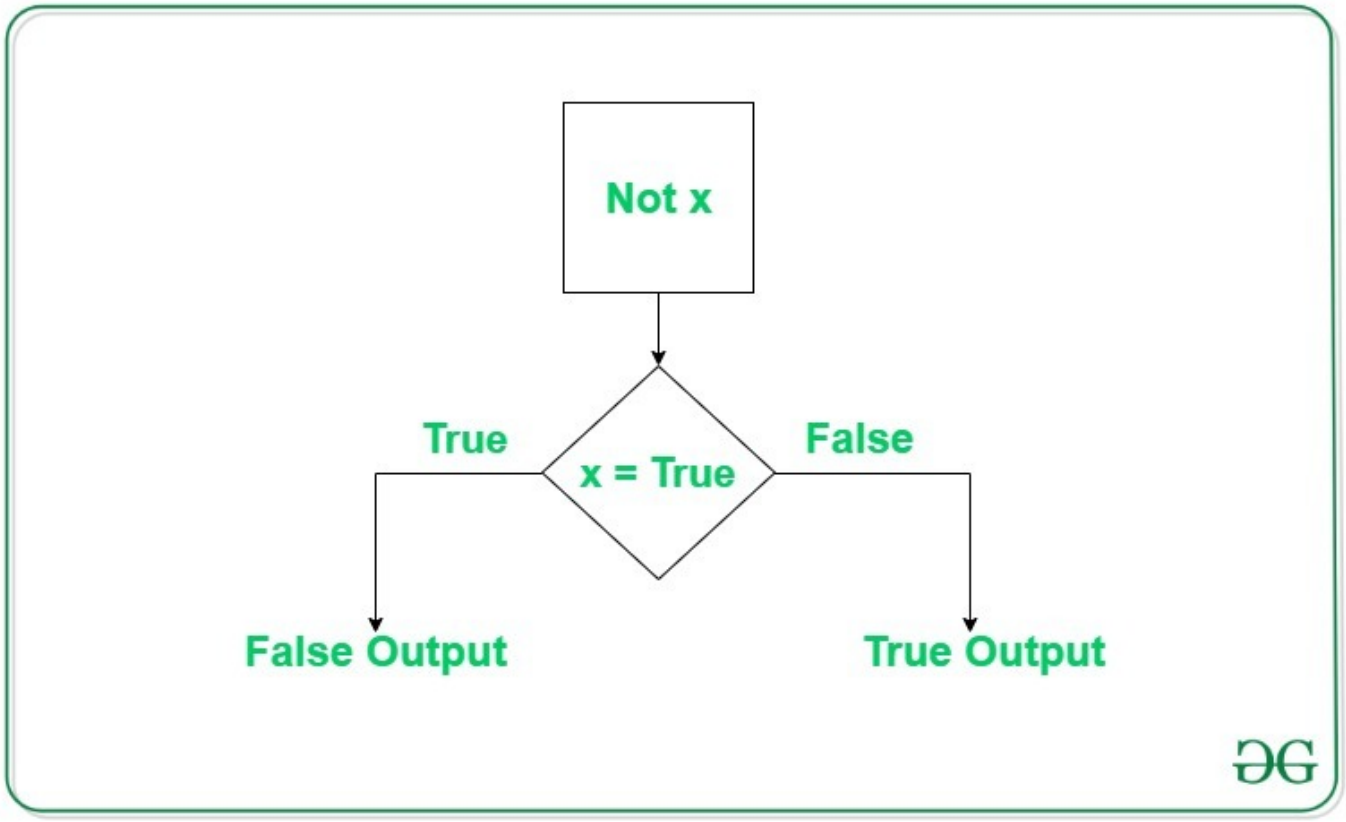
```
Either of the number is greater than 0
No number is greater than 0
```

```
a = 10
b = 12
c = 0

if a or b or c:
    print("Atleast one number has boolean val
else:
    print("All the numbers have boolean value
```

```
Atleast one number has boolean value as True
```

Logical not operator Logical not operator work with the single boolean value. If the boolean value is True it returns False and vice-versa.




```
a = 10

if not a == 10:
  print ("a not equals 10")
else:
  print("a equals 10")
```

```
a equals 10
```

```
a = 10

if not a%5 == 0:
  print("a is not perfectly divisible by 5")
else:
  print("a is perfectly divisible by 5")
```

```
a is perfectly divisible by 5
```

```
a = 10

if not a:
    print("Boolean value of a is True")

if not (a%3 == 0 or a%5 == 0):
    print("10 is not divisible by either 3 or
else:
    print("10 is divisible by either 3 or 5")
```

```
10 is divisible by either 3 or 5
```

```
# Examples of Logical Operator
a = True
b = False

# Print a and b is False
print(a and b)

# Print a or b is True
print(a or b)

# Print not a is False
print(not a)
```

```
False
True
False
```

Bitwise Operators

- Bitwise operators **act on bits** and perform the **bit-by-bit** operations. These are used to operate on **binary numbers**.

Bitwise operators:

1. Bitwise AND operator

2. Bitwise OR operator
3. Bitwise not operator
4. Bitwise XOR operator

Shift Operators:

5. Bitwise right shift
6. Bitwise left shift
7. Bitwise Operator Overloading

Operator	Name	Description
&	AND	Sets each bit to 1 if both bits are 1
|	OR	Sets each bit to 1 if one of two bits is 1
^	XOR	Sets each bit to 1 if only one of two bits is 1
~	NOT	Inverts all the bits
<<	Zero fill left shift	Shift left by pushing zeroes in from the right and let the leftmost bits fall off
>>	Signed right shift	Shift right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off

SCALER Topics

Note: *Python bitwise operators work only on integers.*

Bitwise AND operator: Returns 1 if both the bits are 1 else 0.

```
a = 10 #= 1010 (Binary)
b = 4 # =  0100 (Binary

a & b = 1010
          &
        0100
      = 0000
      = 0 (Decimal)
```

```
a=10
b=4
print(a&b)
```

```
0
```

Bitwise or operator: Returns 1 if either of the bit is 1 else 0.

```
a = 10 = 1010 (Binary)
b = 4 =  0100 (Binary

a | b = 1010
        |
      0100
    = 1110
    = 14 (Decimal)
```

```
a=10
b=4
print(a|b)
```

```
14
```

Bitwise not operator: Returns one's complement of the number.

```
a = 10 = 1010 (Binary)

~a = ~1010
   = -(1010 + 1)
   = -(1011)
   = -11 (Decimal)
```

```
a=10
~a
```

```
-11
```

Bitwise xor operator: Returns 1 if one of the bits is 1 and the other is 0 else returns false.

```
a = 10 = 1010 (Binary)
b = 4 =  0100 (Binary)

a & b = 1010
         ^
        0100
      = 1110
      = 14 (Decimal)
```

```
a=10
b=4
print(a ^ b)
```

```
14
```

```
# Python program to show
# bitwise operators

a = 10
b = 4

# Print bitwise AND operation
print("a & b =", a & b)

# Print bitwise OR operation
print("a | b =", a | b)

# Print bitwise NOT operation
```

```
print("~a =", ~a)

# print bitwise XOR operation
print("a ^ b =", a ^ b)
```

```
a & b = 0
a | b = 14
~a = -11
a ^ b = 14
```

Shift Operators These operators are used to shift the bits of a number left or right thereby multiplying or dividing the number by two respectively. They can be used when we have to multiply or divide a number by two.

Bitwise right shift: Shifts the bits of the number to the right and fills 0 on voids left as a result. Similar effect as of dividing the number with some power of two.

```
a = 10
a >> 1
```

```
5
```

Bitwise left shift: Shifts the bits of the number to the left and fills 0 on voids left as a result. Similar effect as of multiplying the number with some power of two.

```
a = 5 = 0000 0101
b = -10 = 1111 0110

a << 1 = 0000 1010 = 10
a << 2 = 0001 0100 = 20

b << 1 = 0000 1010 = -20
b << 2 = 0001 0100 = -40
```

```
# Python program to show
# shift operators
```

```
a = 10
b = -10

# print bitwise right shift operator
print("a >> 1 =", a >> 1)
print("b >> 1 =", b >> 1)

a = 5
b = -10

# print bitwise left shift operator
print("a << 1 =", a << 1)
print("b << 1 =", b << 1)
```

```
a >> 1 = 5
b >> 1 = -5
a << 1 = 10
b << 1 = -20
```

```
# Examples of Bitwise operators
a = 10
b = 4

# Print bitwise AND operation
print(a & b)

# Print bitwise OR operation
print(a | b)

# Print bitwise NOT operation
print(~a)

# print bitwise XOR operation
```

```
print(a ^ b)

# print bitwise right shift operation
print(a >> 2)

# print bitwise left shift operation
print(a << 2)
```

```
0
14
-11
14
2
40
```

```
number1 = 4  # 0100 in binary
number2 = 3  # 0011 in binary
print("& operation- ", number1 & number2)  #
print("| operation- ", number1 | number2)  #
print("^ operation- ", number1 ^ number2)  #
print("~ operation- ", ~number2)  # negation
print("<< operation- ", number1 << 1)  # shif
print(">> operation- ", number1 >> 1)  # shif
```

```
& operation-  0
| operation-  7
^ operation-  7
~ operation-  -4
<< operation-  8
>> operation-  2
```

Python Membership and Identity Operators

Membership Operators Membership operators are operators used to validate the membership of a value. It tests for membership in a sequence, such as strings, lists, or tuples.

In Python, in and not in are the membership operators. They are used to test whether a value or variable is found in a sequence (string, list, tuple, set and dictionary).

In a dictionary we can only test for presence of key, not the value.

in operator: The 'in' operator is used to check if a value exists in a sequence or not. Evaluates to true if it finds a variable in the specified sequence and false otherwise

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`

```
x = 'Hello world'
y = {1:'a', 2:'b'}

# check if 'H' is present in x string
print('H' in x)  # prints True
```

```
True
```

```
print('h' in x)
```

```
False
```

```
print('w' not in x)
```

```
False
```

```
# check if 'hello' is present in x string
print('hello' not in x)  # prints True
```

```
True
```

```
# check if '1' key is present in y
print(1 in y)  # prints True
```

```
True
```

```
# check if 'a' key is present in y
print('b' in y)  # prints False
```

```
False
```

Here, 'H' is in x but 'hello' is not present in x (remember, Python is case sensitive).

Similarly, 1 is key and 'a' is the value in dictionary y. Hence, 'a' in y returns False.

Identity operators

- In Python, is and is not are used to check if two values 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`

```
x1 = 5
y1 = 5
x2 = 'Hello'
y2 = 'Hello'
x3 = [1,2,3]
y3 = [1,2,3]

print(x1 is not y1)  # prints False

print(x2 is y2)  # prints True

print(x3 is y3)  # prints False
```

```
False
True
```

Here, we see that x1 and y1 are integers of the same values, so they are equal as well as identical. Same is the case with x2 and y2 (strings).

But x3 and y3 are lists. They are equal but not identical. It is because the interpreter locates them separately in memory although they are equal.

```
id(x1)
```

```
140662570121648
```

```
id(y1)
```