# **Operators**

**Python Operators** in general are used to perform operations on values and variables. These are standard symbols used for the purpose of logical and arithmetic operations. In this article, we will look into different types of Python operators.

## **Arithmetic Operators**

[Arithmetic operators](https://www.geeksforgeeks.org/python-arithmetic-operators/) are used to performing mathematical operations like addition, subtraction, multiplication, and division.

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Syntax** |
| + | Addition: adds two operands | x + y |
| – | Subtraction: subtracts two operands | x – y |
| \* | Multiplication: multiplies two operands | x \* y |
| / | Division (float): divides the first operand by the second | x / y |
| // | Division (floor): divides the first operand by the second | x // y |
| % | Modulus: returns the remainder when the first operand is divided by the second | x % y |
| \*\* | Power: Returns first raised to power second | x \*\* y |

### **Example: Arithmetic operators in Python**

|  |
| --- |
| # 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) |

**Output**13

5

36

2.25

2

1

6561

**Note:** Refer to [Differences between / and //](https://www.geeksforgeeks.org/benefits-of-double-division-operator-over-single-division-operator-in-python/) for some interesting facts about these two operators.

## **Comparison Operators**

[Comparison](https://www.geeksforgeeks.org/relational-operators-in-python/) of [Relational operators](https://www.geeksforgeeks.org/relational-operators-in-python/) compares the values. It either returns **True** or **False** according to the condition.

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Syntax** |
| > | Greater than: True if the left operand is greater than the right | x > y |
| < | Less than: True if the 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 the left operand is greater than or equal to the right | x >= y |
| <= | Less than or equal to True if the left operand is less than or equal to the right | x <= y |

### **Example: Comparison Operators in Python**

|  |
| --- |
| # Examples of Relational Operators  a =13  b =33    # a > b is False  print(a > b)    # a < b is True  print(a < b)    # a == b is False  print(a ==b)    # a != b is True  print(a !=b)    # a >= b is False  print(a >=b)    # a <= b is True  print(a <=b) |

**Output**False

True

False

True

False

True

## **Logical Operators**

[Logical operators](https://www.geeksforgeeks.org/python-logical-operators-with-examples-improvement-needed/) perform **Logical AND**, **Logical OR**, and **Logical NOT** operations. It is used to combine conditional statements.

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Syntax** |
| and | Logical AND: True if both the operands are true | x and y |
| or | Logical OR: True if either of the operands is true | x or y |
| not | Logical NOT: True if the operand is false | not x |

### **Example: Logical Operators in Python**

|  |
| --- |
| # Examples of Logical Operator  a =True  b =False    # Print a and b is False  print(a andb)    # Print a or b is True  print(a orb)    # Print not a is False  print(nota) |

**Output**False

True

False

## **Bitwise Operators**

[Bitwise operators](https://www.geeksforgeeks.org/python-bitwise-operators/) act on bits and perform the bit-by-bit operations. These are used to operate on binary numbers.

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Syntax** |
| & | Bitwise AND | x & y |
| | | Bitwise OR | x | y |
| ~ | Bitwise NOT | ~x |
| ^ | Bitwise XOR | x ^ y |
| >> | Bitwise right shift | x>> |
| << | Bitwise left shift | x<< |

### **Example: Bitwise Operators in Python**

|  |
| --- |
| # 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) |

**Output**0

14

-11

14

2

40

## **Assignment Operators**

[Assignment operators](https://www.geeksforgeeks.org/assignment-operators-in-python/) are used to assigning values to the variables.

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Syntax** |
| = | Assign value of right side of expression to left side operand | x = y + z |
| += | Add AND: Add right-side operand with left side operand and then assign to left operand | a+=b     a=a+b |
| -= | Subtract AND: Subtract right operand from left operand and then assign to left operand | a-=b     a=a-b |
| \*= | Multiply AND: Multiply right operand with left operand and then assign to left operand | a\*=b     a=a\*b |
| /= | Divide AND: Divide left operand with right operand and then assign to left operand | a/=b     a=a/b |
| %= | Modulus AND: Takes modulus using left and right operands and assign the result to left operand | a%=b     a=a%b |
| //= | Divide(floor) AND: Divide left operand with right operand and then assign the value(floor) to left operand | a//=b     a=a//b |
| \*\*= | Exponent AND: Calculate exponent(raise power) value using operands and assign value to left operand | a\*\*=b     a=a\*\*b |
| &= | Performs Bitwise AND on operands and assign value to left operand | a&=b     a=a&b |
| |= | Performs Bitwise OR on operands and assign value to left operand | a|=b     a=a|b |
| ^= | Performs Bitwise xOR on operands and assign value to left operand | a^=b     a=a^b |
| >>= | Performs Bitwise right shift on operands and assign value to left operand | a>>=b     a=a>>b |
| <<= | Performs Bitwise left shift on operands and assign value to left operand | a <<= b     a= a << b |

### **Example: Assignment Operators in Python**

|  |
| --- |
| # 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) |

**Output**10

20

10

100

102400

## **Identity Operators**

**is** and **is not** are the [identity operators](https://www.geeksforgeeks.org/python-membership-identity-operators-not-not/) both are used to check if two values are located on the same part of the memory. Two variables that are equal do not imply that they are identical.

**is** True if the operands are identical

**is not** True if the operands are not identical

### **Example: Identity Operator**

|  |
| --- |
| a =10  b =20  c =a    print(a isnotb)  print(a isc) |

**Output**True

True

## **Membership Operators**

**in** and **not in** are the membership operators; used to test whether a value or variable is in a sequence.

**in** True if value is found in the sequence

**not in** True if value is not found in the sequence

### **Example: Membership Operator**

|  |
| --- |
| # Python program to illustrate  # not 'in' operator  x =24  y =20  list=[10, 20, 30, 40, 50]    if(x notinlist):      print("x is NOT present in given list")  else:      print("x is present in given list")    if(y inlist):      print("y is present in given list")  else:      print("y is NOT present in given list") |

**Output**x is NOT present in given list

y is present in given list

## **Precedence and Associativity of Operators**

[**Precedence and Associativity of Operators:**](https://www.geeksforgeeks.org/python-operators/) Operator precedence and associativity determine the priorities of the operator.

### **Operator Precedence**

This is used in an expression with more than one operator with different precedence to determine which operation to perform first.

### **Example: Operator Precedence**

|  |
| --- |
| # Examples of Operator Precedence    # Precedence of '+' & '\*'  expr =10+20\*30  print(expr)    # Precedence of 'or' & 'and'  name ="Alex"  age =0    ifname =="Alex"orname =="John"andage >=2:      print("Hello! Welcome.")  else:      print("Good Bye!!") |

**Output**610

Hello! Welcome.

### **Operator Associativity**

If an expression contains two or more operators with the same precedence then Operator Associativity is used to determine. It can either be Left to Right or from Right to Left.

### **Example: Operator Associativity**

|  |
| --- |
| # Examples of Operator Associativity    # Left-right associativity  # 100 / 10 \* 10 is calculated as  # (100 / 10) \* 10 and not  # as 100 / (10 \* 10)  print(100/10\*10)    # Left-right associativity  # 5 - 2 + 3 is calculated as  # (5 - 2) + 3 and not  # as 5 - (2 + 3)  print(5-2+3)    # left-right associativity  print(5-(2+3))    # right-left associativity  # 2 \*\* 3 \*\* 2 is calculated as  # 2 \*\* (3 \*\* 2) and not  # as (2 \*\* 3) \*\* 2  print(2\*\*3\*\*2) |

**Output**100.0

6

0

512

##### [**Quiz on Python Operators**](https://www.geeksforgeeks.org/operators-gq/)

**Ternary Operator**

Ternary operators are also known as conditional expressions are operators that evaluate something based on a condition being true or false. It was added to Python in version [2.5](https://mail.python.org/pipermail/python-dev/2005-September/056846.html).   
It simply allows testing a condition in a **single line** replacing the multiline if-else making the code compact.

**Syntax :**

[on\_true] if [expression] else [on\_false]

* **Simple Method to use ternary operator:**

|  |
| --- |
| # Program to demonstrate conditional operator  a, b =10, 20    # Copy value of a in min if a < b else copy b  min=a ifa < b elseb    print(min) |

**Output:**

10

* **Direct Method by using tuples, Dictionary**, **and lambda**

|  |
| --- |
| # Python program to demonstrate ternary operator  a, b =10, 20    # Use tuple for selecting an item  # (if\_test\_false,if\_test\_true)[test]  # if [a<b] is true it return 1, so element with 1 index will print  # else if [a<b] is false it return 0, so element with 0 index will print  print( (b, a) [a < b] )    # Use Dictionary for selecting an item  # if [a < b] is true then value of True key will print  # else if [a<b] is false then value of False key will print  print({True: a, False: b} [a < b])    # lambda is more efficient than above two methods  # because in lambda  we are assure that  # only one expression will be evaluated unlike in  # tuple and Dictionary  print((lambda: b, lambda: a)[a < b]()) |

**Output:**

10

10

10

* **Ternary operator can be written as nested if-else:**

|  |
| --- |
| # Python program to demonstrate nested ternary operator  a, b =10, 20    print("Both a and b are equal"ifa ==b else"a is greater than b"          ifa > b else"b is greater than a") |

The above approach can be written as: 

|  |
| --- |
| # Python program to demonstrate nested ternary operator  a, b =10, 20    ifa !=b:      ifa > b:          print("a is greater than b")      else:          print("b is greater than a")  else:      print("Both a and b are equal") |

**Output:**

b is greater than a

* **To use print function in ternary operator be like:-**

Example: Find the Larger number among 2 using ternary operator in python3

|  |
| --- |
| a=5  b=7    # [statement\_on\_True] if [condition] else [statement\_on\_false]    print(a,"is greater") if(a>b) elseprint(b,"is Greater") |

**Output:**

7 is Greater

### **Important Points:**

* First the given condition is evaluated (a < b), then either a or b is returned based on the Boolean value returned by the condition
* Order of the arguments in the operator is different from other languages like C/C++ (See [C/C++ ternary operators](https://www.geeksforgeeks.org/cc-ternary-operator-some-interesting-observations/)).
* Conditional expressions have the lowest priority amongst all Python operations.

**Method used prior to 2.5 when** the **ternary operator was not present**   
In an expression like the one given below, the interpreter checks for the expression if this is true then on\_true is evaluated, else the on\_false is evaluated.

**Syntax :**

'''When condition becomes true, expression [on\_false]

is not executed and value of "True and [on\_true]"

is returned. Else value of "False or [on\_false]"

is returned.

Note that "True and x" is equal to x.

And "False or x" is equal to x. '''

[expression] and [on\_true] or [on\_false]

**Example :**

|  |
| --- |
| # Program to demonstrate conditional operator  a, b =10, 20    # If a is less than b, then a is assigned  # else b is assigned (Note : it doesn't  # work if a is 0.  min=a < b anda orb    print(min) |

**Output:**

10

**Note :** The only drawback of this method is that **on\_true must not be zero or False**. If this happens on\_false will be evaluated always. The reason for that is if the expression is true, the interpreter will check for the on\_true, if that will be zero or false, that will force the interpreter to check for on\_false to give the final result of the whole expression.

**Division Operators**

Consider the below statements in Python.

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

**Output:**

2

-3

The first output is fine, but the second one may be surprised if we are coming Java/C++ world. In Python, the “//” operator works as a floor division for integer and float arguments. However, the operator / returns a float value if one of the arguments is a float (this is similar to C++)

**Note:**

The “//” operator is used to return the closest integer value which is less than or equal to a specified expression or value. So from the above code, 5//2 returns 2. You know that 5/2 is 2.5, the closest integer which is less than or equal is 2[5//2].( it is inverse to the normal maths, in normal maths the value is 3).

|  |
| --- |
| # A Python program to demonstrate use of  # "/" for floating point numbers  print(5.0/2)  print(-5.0/2) |

**Output:**

2.5

-2.5

**The real floor division operator is “//”. It returns floor value for both integer and floating point arguments.**

|  |
| --- |
| # A Python program to demonstrate use of  # "//" for both integers and floating points  print(5//2)  print(-5//2)  print(5.0//2)  print(-5.0//2) |

**Output:**

2

-3

2.0

-3.0

See [this](https://ide.geeksforgeeks.org/Rm9JMm) for example.