3.3 Operators

Introduction

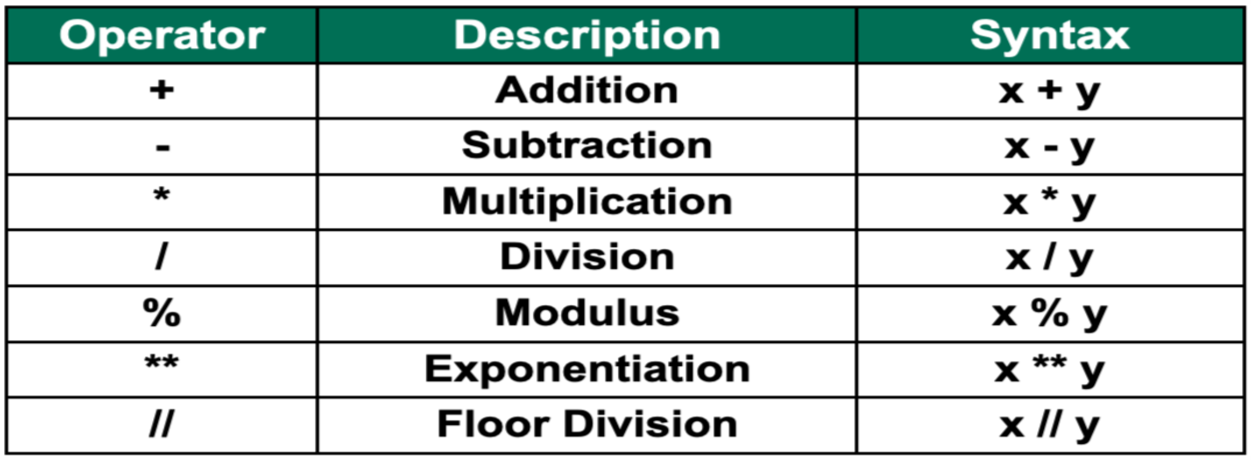
Operators are used for performing operations on variables and values. The simplest operations we know are addition and subtraction.

```
>>> x + 5
```

The values that an operator acts on are called operands. In the above example, + is the operator. X, a variable, is the first operand, while 5, a value, is the second operand

Arithmetics Operators

Arithmetic operators are used for performing common mathematical operations in Python. The below table shows Python's arithmetic operators with their symbols and meaning.



Operator	Description	Syntax
+	Addition	x + y
-	Subtraction	x - y
*	Multiplication	x * y
/	Division	x / y
%	Modulus	x % y
**	Exponentiation	x ** y
//	Floor Division	x // y

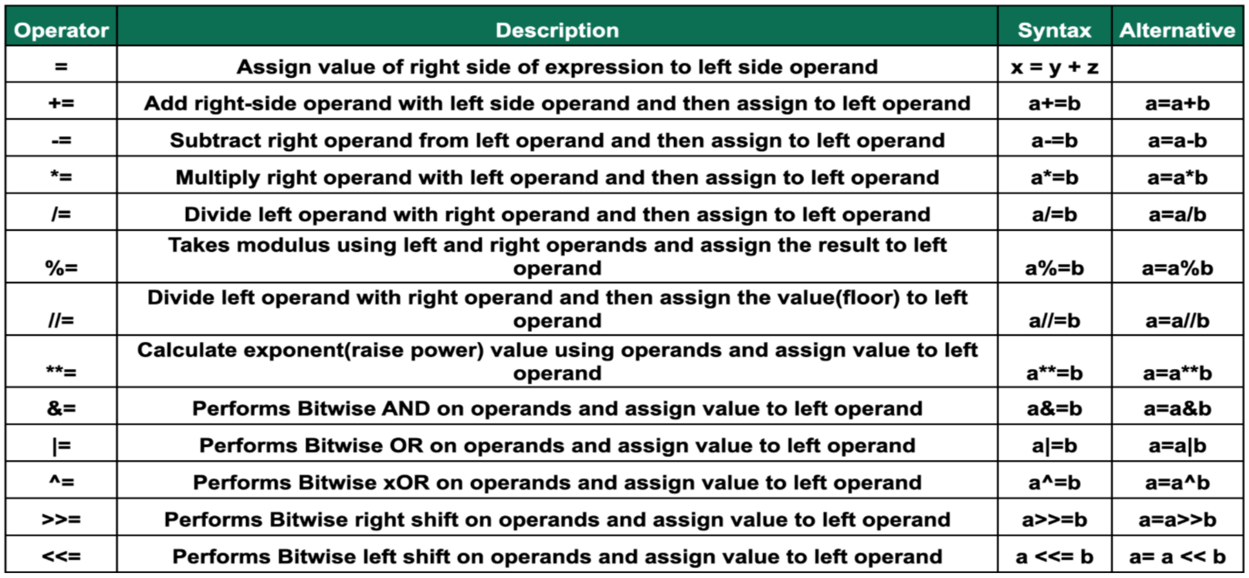
We have the basic arithmetic operators: Addition, Subtraction, and Multiplication. Then we have two types of Division: Standard Division and Floor Division. Standard Division always results in a float. Whereas in the Floor Division, the fractional portion of the result is truncated, and only the integer part is kept.

Assignment Operators

Assignment operators are used for assigning values to the variables or any other object in Python.

For instance, x = 5 is a simple assignment operator that assigns the value 5 on the right to the variable x on the left. There are also multiple compound operators, as depicted in this table. x += 2, adds 2 to the variable x, and assigns the result to x. This is the equivalent of x = x + 2. And now the value of x will be 7.

You can go through the below table to learn about all other compound operators in Python.

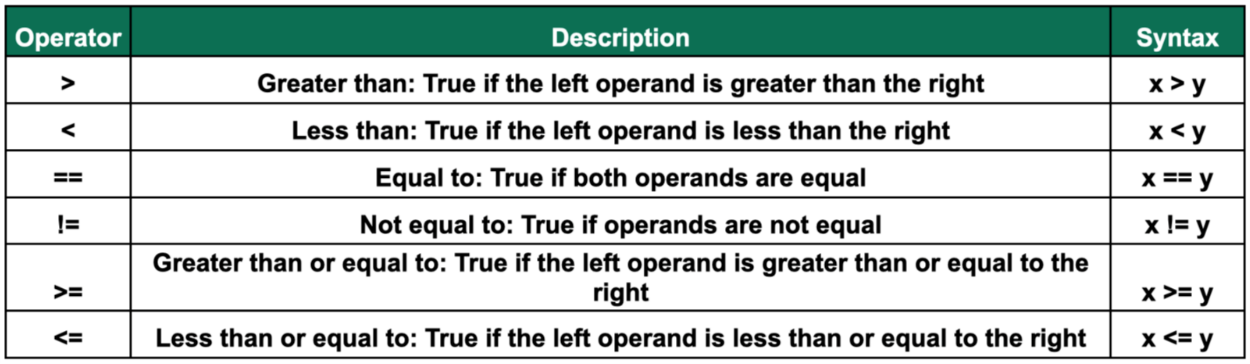


Operator	Description	Syntax	Alternative
=	Assign value of right side of expression to left side operand	x = y + z	
+=	Add right-side operand with left side operand and then assign to left operand	a+=b	a=a+b
-=	Subtract right operand from left operand and then assign to left operand	a-=b	a=a-b
*=	Multiply right operand with left operand and then assign to left operand	a*=b	a=a*b
/=	Divide left operand with right operand and then assign to left operand	a/=b	a=a/b
%=	Takes modulus using left and right operands and assign the result to left operand	a%=b	a=a%b
//=	Divide left operand with right operand and then assign the value(floor) to left operand	a//=b	a=a//b
**=	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

Comparison Operators

Comparison operators are used for comparing two values. Since it returns either true or false, comparison operators are typically used in Boolean contexts. They are especially used in conditional and loop statements in order to direct program flow.

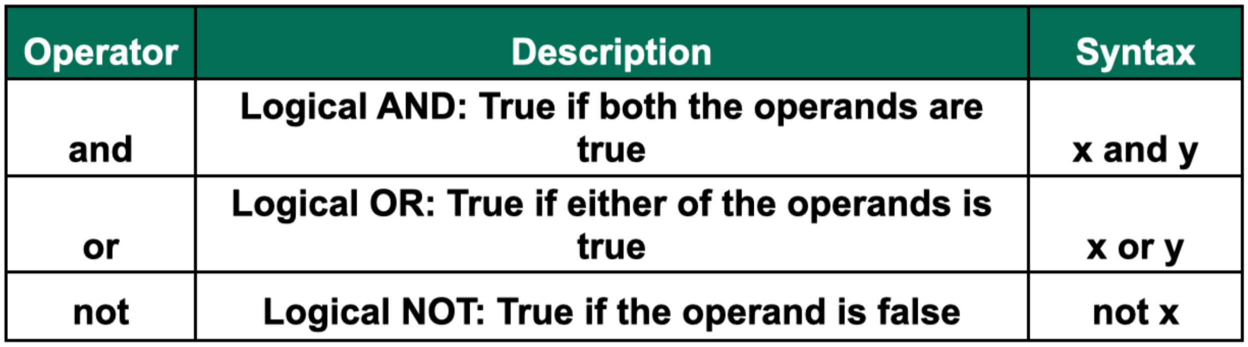
We have greater than, less than, equal to, not equal to, greater than or equal to, and less than or equal to as comparison operators. As we can see here in the example, the result is a Boolean value. 5 is greater than 2 returns true, and 4 is not equal to 4 returns false.



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

Logical Operators

Logical operators "and", "or", and "not" operators are used to compare two conditional statements. The logical operators modify and join together expressions evaluated in Boolean context to create more complex conditions. When using the "and" operator, the expression will only return true if both operands are true. Whereas using the "or" operator will result in true if at least one of the operands is true. Finally, the "not" operator will only give true if the Boolean expression in the operand is false.



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

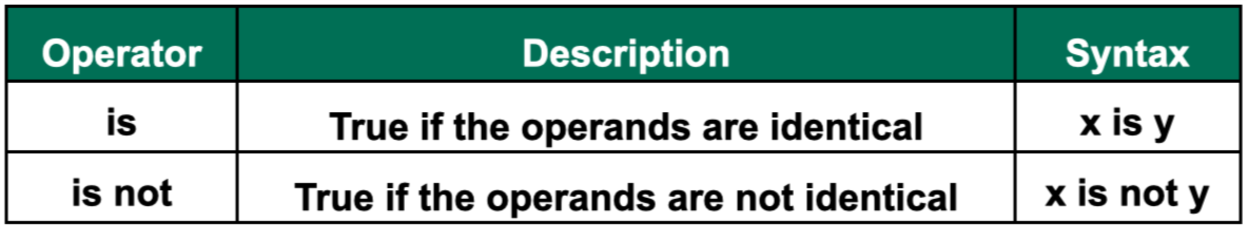
Identity Operators

Identity operators "is" and "is not" are used to compare two objects. Not if they are equal but if they are actually the same object located on the same part of the memory. Hence, two equal objects do not necessarily imply that they are identical, with the same identity.

Here is an example of two objects that are equal but not identical.

x and y both refer to lists containing the same elements. They are equal but not identical, as they do not reference the same object in memory. So, when you try x is y, it returns false. You can verify that x and y do not refer to the same object by using the id() function.

When you make an assignment like x = z, Python merely creates a second reference to the same object. Therefore, when you try x is z, it returns true. Both x and z have the same object identity.



Operator	Description	Syntax
is	True if the operands are identical	x is y
is not	True if the operands are not identical	x is not y

Membership Operators

Membership operators "in" and "not in" are used to test whether a value or variable is found in a sequence. A sequence can be anything from a string, a list, a tuple, a set, or a dictionary. Similar to identity operators, membership operators also return Boolean values.

For example, here we have a list with "a" and "b". The statement" a" in list returns true. The statement "c" not in list also returns true as "c" is not an element of the list.

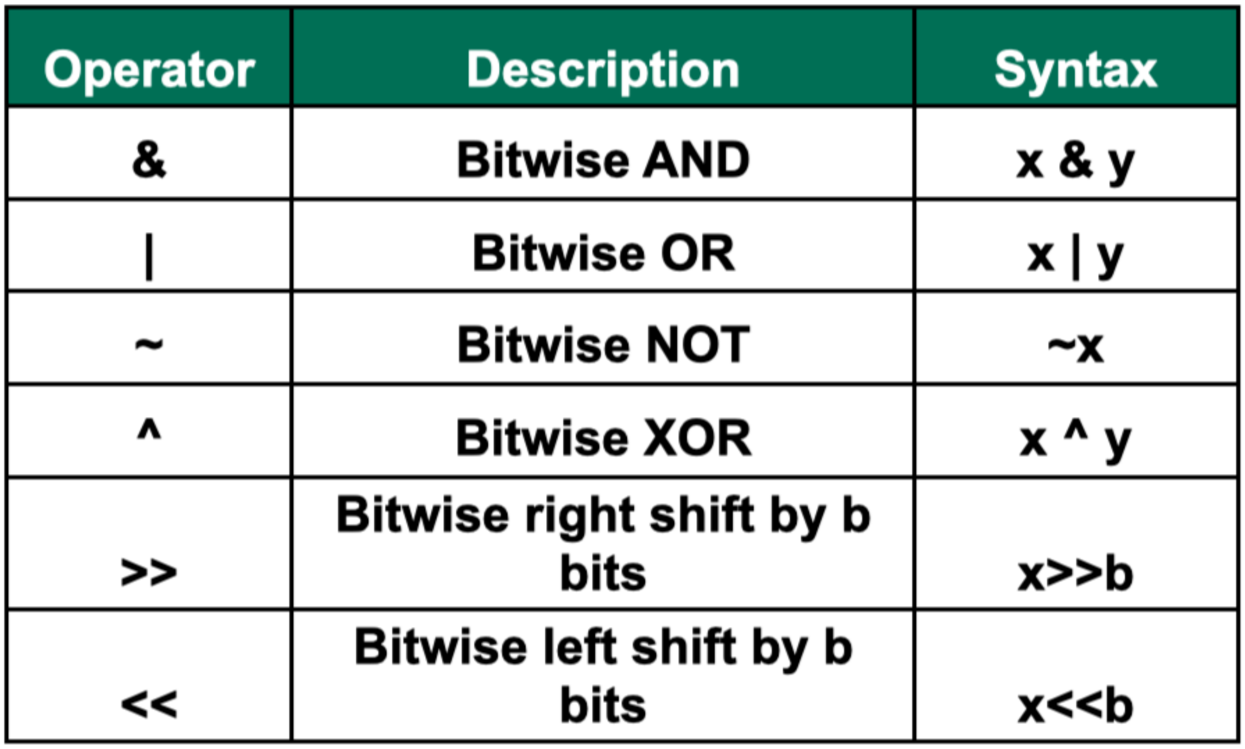
Bitwise Operators

Bitwise operators are similar to logical operators but treat operands as binary values and operate on them bit by bit.

For example, 5 is 101 in binary form, and 7 is 111. When you perform the "and" operation on these operands, it will give you the output 5 (101 in binary form) since you performed a logical operation on each bit of the numbers, one at a time. This is precisely the same when you perform the "or" and "xor" operations but using the appropriate logical operation bit by bit. XOR will return true only if one bit is true, not the both of them.

We also have the right shift and left shift operators. Left shift shifts left a binary value by pushing zeros in from the right and letting the leftmost bits fall off. Right shift shifts right by pushing copies of the leftmost bit in from the left, and letting the rightmost bits fall off.

In Python, an integer is usually represented in 32 bits. When we ask it to shift an x left by 2 bits, it takes the 32-bit binary representation of x and shifts it right by 2 bits, as shown in the example. The resulting binary value is 20. Similarly, we end up with 1 after right shifting x by 2 bits.



Operator	Description	Syntax
&	Bitwise AND	x & y
|	Bitwise OR	x | y
~	Bitwise NOT	~x
^	Bitwise XOR	x ^ y
>>	Bitwise right shift by b bits	x>>b
<<	Bitwise left shift by b bits	x<<b

Further reading: https://docs.python.org/3/library/operator.html