

Python Operators

Operators are used to perform operations on variables and values.

Python divides the operators in the following groups:

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

Python Assignment Operators

Assignment operators are used to assign values to variables:

Operator	Example	Same As
=	<code>x = 5</code>	<code>x = 5</code>
+=	<code>x += 3</code>	<code>x = x + 3</code>
-=	<code>x -= 3</code>	<code>x = x - 3</code>
*=	<code>x *= 3</code>	<code>x = x * 3</code>
/=	<code>x /= 3</code>	<code>x = x / 3</code>
%=	<code>x %= 3</code>	<code>x = x % 3</code>
//=	<code>x //= 3</code>	<code>x = x // 3</code>
**=	<code>x **= 3</code>	<code>x = x ** 3</code>
&=	<code>x &= 3</code>	<code>x = x & 3</code>
=	<code>x = 3</code>	<code>x = x 3</code>
^=	<code>x ^= 3</code>	<code>x = x ^ 3</code>
>>=	<code>x >>= 3</code>	<code>x = x >> 3</code>
<<=	<code>x <<= 3</code>	<code>x = x << 3</code>
:=	<code>print(x := 3)</code>	<code>x = 3</code> <code>print(x)</code>

Operator	Name	Example
+	Addition	<code>x + y</code>
-	Subtraction	<code>x - y</code>
*	Multiplication	<code>x * y</code>
/	Division	<code>x / y</code>
%	Modulus	<code>x % y</code>
**	Exponentiation	<code>x ** y</code>
//	Floor division	<code>x // y</code>

Python Comparison Operators

Comparison operators are used to compare two values:

Operator	Name	Example
==	Equal	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

Python Logical Operators

Logical operators are used to combine conditional statements:

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

Python Identity Operators

Identity operators are used to compare the objects, not if they are equal, but if they are actually the same object, with the same memory location:

Operator	Description	Example
is	Returns True if both variables are the same object	x is y
is not	Returns True if both variables are not the same object	x is not y

Python Membership Operators

Membership operators are used to test if a sequence is presented in an object:

Operator	Description	Example
in	Returns True if a sequence with the specified value is present in the object	x in y
not in	Returns True if a sequence with the specified value is not present in the object	x not in y

Python Bitwise Operators

Bitwise operators are used to compare (binary) numbers:

Operator	Name	Description	Example
&	AND	Sets each bit to 1 if both bits are 1	x & y
	OR	Sets each bit to 1 if one of two bits is 1	x y
^	XOR	Sets each bit to 1 if only one of two bits is 1	x ^ y
~	NOT	Inverts all the bits	~x
<<	Zero fill left shift	Shift left by pushing zeros in from the right and let the leftmost bits fall off	x << 2
>>	Signed right shift	Shift right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off	x >> 2

Python Loops

Python has two primitive loop commands:

- `while` loops
- `for` loops

The while Loop

With the `while` loop we can execute a set of statements as long as a condition is true.

```
i = 1
while i < 6:
    print(i)
    i += 1
```

The break Statement

With the `break` statement we can stop the loop even if the while condition is true:

```
i = 1
while i < 6:
    print(i)
    if i == 3:
        break
    i += 1
```


The continue Statement

With the `continue` statement we can stop the current iteration, and continue with the next:

```
i = 0
while i < 6:
    i += 1
    if i == 3:
        continue
    print(i)
```

The else Statement

With the `else` statement we can run a block of code once when the condition no longer is true:

```
i = 1
while i < 6:
    print(i)
    i += 1
else:
    print("i is no longer less than 6")
```

[Try it Yourself »](#)

Infinite Loop

A loop becomes infinite loop if a condition never becomes false. The results in this Loop that never ends. That loop is called Infinite Loop.

```
num = 1
while num < 2:
    print( num)
```

Nested While Loop

A while loop contained another while loop is called Nested While Loop.

```
i = 1
while i <= 4:
    j = 0
    while j <= 3 :
        print (i*j, end=" ")
        j += 1
    print ()
    i += 1
```

Python For Loops

With the for loop we can execute a set of statements, once for each item in a list, tuple, set etc.

```
fruits = ["apple", "banana", "cherry"]  
for x in fruits:  
    print(x)
```

Looping Through a String

```
for x in "banana":  
    print(x)
```

The break Statement

With the **break** statement we can stop the loop before it has looped through all the items:

```
fruits = ["apple", "banana", "cherry"]  
for x in fruits:  
    print(x)  
    if x == "banana":  
        break
```

Exit the loop when x is "banana", but this time the break comes before the print:

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
    if x == "banana":
        break
    print(x)
```

The continue Statement

With the `continue` statement we can stop the current iteration of the loop, and continue with the next:

Do not print banana:

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
    if x == "banana":
        continue
    print(x)
```

The range() Function

To loop through a set of code a specified number of times, we can use the `range()` function,

The `range()` function returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default), and ends at a specified number.

Using the `range()` function:

```
for x in range(6):  
    print(x)
```

The `range()` function defaults to 0 as a starting value, however it is possible to specify the starting value by adding a parameter: `range(2, 6)`, which means values from 2 to 6 (but not including 6):

Using the start parameter:

```
for x in range(2, 6):  
    print(x)
```

The `range()` function defaults to increment the sequence by 1, however it is possible to specify the increment value by adding a third parameter: `range(2, 30, 3)`:

Increment the sequence with 3 (default is 1):

```
for x in range(2, 30, 3):  
    print(x)
```

Else in For Loop

The `else` keyword in a `for` loop specifies a block of code to be executed when the loop is finished:

Print all numbers from 0 to 5, and print a message when the loop has ended:

```
for x in range(6):  
    print(x)  
else:  
    print("Finally finished!")
```

Note: The `else` block will NOT be executed if the loop is stopped by a `break` statement.

Break the loop when `x` is 3, and see what happens with the `else` block:

```
for x in range(6):
    if x == 3: break
    print(x)
else:
    print("Finally finished!")
```

Nested Loops

A nested loop is a loop inside a loop.

The "inner loop" will be executed one time for each iteration of the "outer loop":

Print each adjective for every fruit:

```
adj = ["red", "big", "tasty"]
fruits = ["apple", "banana", "cherry"]
```

```
for x in adj:
    for y in fruits:
        print(x, y)
```


