Previous: Output Data To Your Screen With Python's print() Function

Next: Python For Loop and While Loop ›

Home » Introduction to Python » Python Boolean and Conditional Programming: if.. else

Python Boolean and Conditional Programming: if.. else

June 8, 2022

Besides numbers and strings, Python has several other types of data. One of them is the Boolean data type. Booleans are extremely simple: they are either true or false. Booleans, in combination with Boolean operators, make it possible to create conditional programs: programs that decide to do different things, based on certain conditions.

The Boolean data type was named after George Boole, the man that defined an algebraic system of logic in the mid 19th century.

Table of Contents [hide]

What is a Boolean?

Let's start with a definition:

Boolean
A boolean is the simplest data type; it's either `True` or `False`.

In computer science, booleans are used a lot. This has to do with how computers work internally. Many operations inside a computer come down to a simple "true or false." It's important to note, that in Python a Boolean value starts with an upper-case letter: `True` or `False`. This is in contrast to most other programming languages, where lower-case is the norm.

In Python, we use booleans in combination with conditional statements to control the flow of a program:

```
>>> door_is_locked = True
>>> if door_is_locked:
...     print("Mum, open the door!")
...
Mum, open the door!
>>>_
```

Here's an interactive version of the same code that you can experiment with:

The Python if statement

First, we define a variable called `door_is_locked` and set it to `True`. Next, you'll find an if-statement. This is a so-called conditional statement. It is followed by an expression that can evaluate to either `True` or `False`. If the expression evaluates to `True`, the block of code that follows is executed. If it evaluates to `False`, it is skipped. Go ahead and change `door_is_locked` to `False` to see what happens.

An if can be followed by an optional else block. This block is executed only when the expression evaluates to `False`. This way, you can run code for both options. Let's try this:

```
>>> door_is_locked = False
>>> if door_is_locked:
...     print("Mum, open the door!")
... else:
...     print("Let's go inside")
...
Let's go inside
>>>_
```

Thanks to our else-block, we can now print an alternative text if `door_is_locked` is `False`. As an exercise, try to modify the interactive code example above to get the same result.

Python operators

The ability to use conditions is what makes computers tick; they make your software smart and allow it to change its behavior based on external input. We've used `True` directly so far, but more expressions evaluate to either `True` or `False`. These expressions often include a so-called operator.

There are multiple types of operators, and for now, we'll only look at these:

1. Comparison operators: they compare two values to each other
2. Logical operators

Comparison operators

Let's look at comparison operators first. You can play around with them in the REPL:

```
>>> 2 > 1
True
>>> 2 < 1
False
>>> 2 < 3 < 4 < 5 < 6
True
>>> 2 < 3 > 2
True
>>> 3 <= 3
True
>>> 3 >= 2
True
>>> 2 == 2
True
>>> 4 != 5
True
>>> 'a' == 'a'
True
>>> 'a' > 'b'
False
```

This is what all the comparison operators are called:

Operator	Meaning
>	greater than
<	smaller than
>=	greater than or equal to

Operator	**Meaning**
<=	smaller than or equal to
==	is equal
!=	is not equal

Python's boolean operators

As can be seen in the examples, these operators work on strings too. Strings are compared in the order of the alphabet, with these added rules:

- Uppercase letters are 'smaller' than lowercase letters, e.g.: 'M' < 'm'
- Digits are smaller than letters: '1' < 'a'

You're probably wondering what the logic is behind these rules. Internally, each character has a number in a table. The position in this table determines the order. It's as simple as that. See Unicode on Wikipedia to learn more about it if you're interested.

Logical operators

Next up: logical operators. These operators only work on booleans and are used to implement logic. The following table lists and describes them:

Operator	What is does	Examples
and	`True` if both statements are true	True and False == False False and False == False True and True == True
or	`True` if one of the statements is true	True or False == True True or True == True False or False == False
not	Negates the statement that follows	not True == False not False == True

Python logical operators

Here are some examples in the REPL to help you play around with these:

```
>>> not True
False
>>> not False
True
>>> True and True
True
>>> True and False
False
```

Comparing different types in Python

When you try to compare different types, you'll often get an error. Let's say you want to compare an integer with a string:

```
>>> 1 < 'a'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: '<' not supported between instances of 'int' and 'str'
>>>
```

This is how Python tells you it can't compare integers to strings. But there are types that can mix and match. I would recommend against it because it makes your code hard to understand, but for sake of demonstration, let's compare a boolean and an int:

```
>>> True == 1
True
>>> False == 0
True
>>> True + True
2
>>> False + False
0
>>> False + True
1
>>> True + 3
4
```

```
>>>
```

As can be seen, `True` has a value of `1`, and `False` has a value of `0`. This has to do with the internal representation of booleans in Python: they are a special kind of number in Python.

Get certified with our courses

Learn Python properly through small, easy-to-digest lessons, progress tracking, quizzes to test your knowledge, and practice sessions. Each course will earn you a downloadable course certificate.

‹ Previous: Output Data To Your Screen With Python’s print() Function

Next: Python For Loop and While Loop ›