Boolean Operators

by Sophia

WHAT'S COVERED

In this lesson, you will learn how to use multiple boolean operators to handle various conditions in Python. Specifically, this lesson covers:

1. Single Argument Boolean Operators

As we've seen, the Python boolean (`bool`) type has two possible values: True or False. Note that they can also be represented by numbers in Python, with False equal to 0 and 1 equal to True.

In addition to the `bool` type, Python provides three boolean operators (also called logical operators): `and`, `or`, and `not`. **Boolean operators** allow us to build and combine more complex boolean expressions from basic expressions. Boolean operators will take boolean inputs and return boolean results.

Since the boolean values can only have one of two values (True or False), we can specify the operators assigned in a truth table. A **truth table** is a small table that lists every possible input value and gives results for the operators.

EXAMPLE
Here is the most simplistic truth table (A is a `bool` type and can be either True or False):

A
True
False

Remember that boolean operators allow us to build more complex boolean expressions from basic expressions. That is where singular vs. multiple arguments come into play. One of the three boolean operators only works with a singular argument.

1a. not Operator

A boolean operator that works with only one argument is `not`. The **not operator** is a reserved keyword and boolean operator that works with one argument and returns the opposite truth value of the argument. This means that if False is passed to the operator, True is returned. If True is passed to the operator, False is returned.

EXAMPLE
Here is what the `not` operator in a truth table looks like.

A	not A
True	False
False	True

Here is a simple example to compare two values when they are equal, which should return True.

EXAMPLE

```
num=1
if num==1:
  print('True')
else:
  print('False')
```

Here is the output.

```
True
```

As expected, `num` does equal 1, so it is True.
Here we compare two values when they are not equal, which should return False.

EXAMPLE

```
num=1
if num == 0:
  print('True')
else:
  print('False')
```

Here is the output.

```
False
```

As expected, `num` does equal 0, so it is False.

Now let's try the same example, but this time adding in the `not` boolean operator.

EXAMPLE

```
num=1
if not (num == 1):
  print('True')
else:
  print('False')
```

Here is the output.

```
False
```

This time `num` does equal 1, but the `not` boolean operator causes it to output False.

EXAMPLE

```
num=1
if not (num == 0):
  print('True')
else:
  print('False')
```

Here is the output.

```
True
```

This time `num` does not equal 0, but the `not` boolean operator causes it to output True.

TERMS TO KNOW

Boolean Operators
Boolean operators (also known as logical operators) allow us to build and combine more complex boolean expressions from more basic expressions.

Truth Table
A truth table is a small table that lists every possible input value and gives results for the operators.

not
The `not` operator is a reserved keyword and boolean operator that works with one argument and returns the opposite truth value of the argument.

2. Multiple Argument Boolean Operators

Now for the other two boolean operator types `and` and `or`. These two operators take multiple arguments.

2a. and Operator

The **and operator** is a reserved keyword and boolean operator that takes two arguments and returns a boolean value of False unless both inputs are True. Let's see what this looks like in a truth table.

A	B	A and B
False	False	False
False	True	False
True	False	False
True	True	True

Notice in this table that if A is False, then no matter what the value is in B, the output is still False. And, if B is False, then no matter what the value of A is, the output will still be False.

KEY CONCEPT

Short-Circuit Evaluation
Python boolean operators such as `and` and `or` use an evaluation method called short-circuit evaluation. Python reads code from left to right, just like we read. Python will start evaluating the left operand and when it detects that there is nothing to be gained by evaluating the rest of a logical expression, it stops its evaluation and does not do the computations in the rest of the logical expression. This is called **short-circuit evaluation**, when the evaluation of a logical expression stops because the overall value is already known.

For example, as seen in the truth table above, if Python evaluates a False in the first operand, it can stop evaluating since the overall value has to be False.

So, looking back at the table above, the operator short-circuits if the first input is False. That means that if the first input is False, the second input isn't even evaluated.

We can double-check this through code to ensure that the conditions match the truth table above.

EXAMPLE

```
num1 = 1
num2 = 2
if num1 == 0 and num2 == 0:
    print('A is False, B is False: True')
else:
    print('A is False, B is False: False')

if num1 == 0 and num2 == 2:
    print('A is False, B is True: True')
else:
    print('A is False, B is True: False')

if num1 == 1 and num2 == 0:
    print('A is True, B is False: True')
else:
    print('A is True, B is False: False')

if num1 == 1 and num2 == 2:
    print('A is True, B is True: True')
else:
    print('A is True, B is True: False')
```
Here is that "and" output.

```
A is False, B is False: False
A is False, B is True: False
A is True, B is False: False
A is True, B is True: True
```
As you'll see, the `and` operator outputs True if and only if both inputs result in True.

2b. or Operator

The **or boolean operator** is a reserved keyword and boolean operator that outputs True if either input is True (or both). It only outputs False if both inputs are set to False. Let's take a look at the truth table below for the `or` boolean operator.

A	B	A or B
False	False	False
False	True	True
True	False	True
True	True	True

It's important to note that the "or" word can have two separate meanings when used informally.

1. Exclusive or
    - There is the "exclusive or", where you can only have one of two conditions. For example, we can say: "You can stay home or you can go to the grocery store." This means that you cannot do both at the same time; you have to choose one of the two.
2. Inclusive or
    - The "inclusive or" is sometimes used with and/or as part of a phrase. For example, we can say: "You can have ice cream and/or a cookie for dessert." This means that you could either eat ice cream, a cookie, or even both. Python and other programming languages evaluate the `or` operator using the "inclusive or". This means that if either input is True or if both inputs are True, the result is True.

In this case, the `or` operator also uses short-circuit evaluation. If the first input is True, the result is True and there's no need to evaluate the second argument. Let's see what this looks like in code that evaluates each possibility:

EXAMPLE

```
num1 = 1
num2 = 2
if num1 == 0 or num2 == 0:
    print('A is False, B is False: True')
else:
    print('A is False, B is False: False')

if num1 == 0 or num2 == 2:
    print('A is False, B is True: True')
else:
    print('A is False, B is True: False')

if num1 == 1 or num2 == 0:
    print('A is True, B is False: True')
else:
    print('A is True, B is False: False')

if num1 == 1 or num2 == 2:
    print('A is True, B is True: True')
else:
    print('A is True, B is True: False')
```

Here is that "or" output.

```
A is False, B is False: False
A is False, B is True: True
A is True, B is False: True
A is True, B is True: True
```

TRY IT

You have seen how boolean operators can affect your conditions.

Directions: Using the IDE, try some of the boolean operators out and see if you can change the output based on what you have just learned.

TERMS TO KNOW

and
The `and` operator is a reserved keyword and boolean operator that takes two arguments and returns a boolean value of False unless both inputs are True.

Short-Circuit Evaluation
Short-circuit evaluation is when the evaluation of a logical expression stops because the overall value is already known.

or
The `or` operator is a reserved keyword and boolean operator that outputs True if either input is True (or both).

SUMMARY

In this lesson, you learned how Python provides three boolean operators (also called logical operators): `and`, `or`, and `not`. We discovered that these operators allow us to build and combine more complex boolean expressions from basic expressions. We found that the `not` **boolean operator** works with a **singular argument** to return a True or False value. In a **multiple argument** situation, the `and` and `or` **boolean operators** are used. Finally, we learned that Python conducts short-circuit evaluation on `and` and `or` operations, meaning Python will stop evaluating logical expressions if the overall value is already known.

Best of luck in your learning!

TERMS TO KNOW

Boolean Operators
Boolean operators (also known as logical operators) allow us to build and combine more complex boolean expressions from more basic expressions.

Short-Circuit Evaluation
Short-circuit evaluation is when the evaluation of a logical expression stops because the overall value is already known.

Truth Table
A truth table is a small table that lists every possible input value and gives results for the operators.

and

The and operator is a reserved keyword and boolean operator that takes two arguments and returns a boolean value of False unless both inputs are True.

not

The not operator is a reserved keyword and boolean operator that works with one argument and returns the opposite truth value of the argument.

or

The or operator is a reserved keyword and boolean operator that outputs True if either input is True (or both).