LEARN PYTHON PROGRAMMING

Logical comparisons in Python: and & or

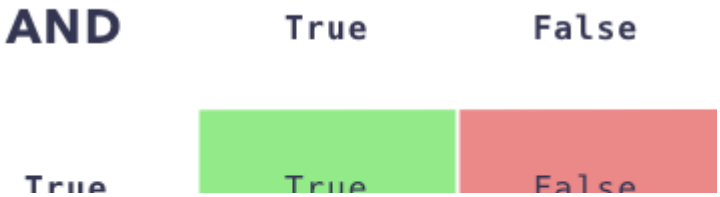
Jose Salvatierra
4 min read · Mar 12

Python's `and` and `or` keywords can be confusing for beginner programmers, but also for programmers coming from other programming languages. That's because in Python, these operators can behave differently than in other languages!

Traditionally, `and` and `or` are used to compare true and false values. We do this in English too:

- This is true when this and that are both true: "This man is a bachelor if he is both a man and unmarried".
- This is true when either this or that (or both) are true.

For simplicity, here's a diagram of how `and` works. If both conditions are true, the result is true. If either condition is false, the result is false.



False	False	False

`or` is somewhat opposite: the result is false only if both conditions are false. Otherwise, the result is true.

OR	True	False
True	True	True
False	True	False

In Python, we would write `and` or `or` between the `True` and `False` keywords. Like this:

```
>>> True and True
True
>>> True and False
False
>>> True or False
True
>>> False or False
False
```

Boolean logic in Python

Python takes this logical evaluation one step further:

returns the second value.

- `or` returns the first value if it evaluates to true, otherwise it returns the second value.

And

Let's start with `and` and look back at the code we had above:

```
>>> False and True
False
>>> True and True
True
>>> True and False
False
```

In the first example, `False and True`, the first value is `False`. According to our truth table when any value is `False`, the result will always be `False`. Python knows this, so as soon as it encounters `False`, it returns that. It doesn't even check the second value!

For the second example, `True and True`, Python will look at both values. Although they are both the same, it is actually giving us the second `True`.

We can see this in the final example, `True and False`, where Python checks the first value and it is `True`, so it gives us the second value, `False`. Again, it doesn't need to check this value.

Or

```
>>> True or False
True
```

```
>>> False or False
False
```

In the first example, `True or False`, Python starts by looking at the first value. It is `True`, so it returns that immediately. Looking back at the table above, if either of the values is `True` the result will always be `True`, so Python saves time by returning immediately after the first value.

In the second example, `False or True`, the first value is `False`. Python then returns the second value.

In the third example, the same thing happens. The first value is `False`, so Python returns the second value—also `False`.

It may seem weird that we're talking about returning the first or second value, but if we compare all this against the true/false tables earlier, you'll see they map exactly. Plus, Python can use this in other scenarios which don't involve `True` and `False`!

Using this without True and False

You can also use these operators on values other than Boolean values.

And

Remember, when using `and` Python returns the first value if it evaluates to `False`. So what about this?

```
>>> None and '35'
None
```

Should that give you an error?

In many programming languages it would, but in Python it gives you `None`. That's because in Python, `None` evaluates to `False`. We'll cover why in just a moment!

Here's a couple more examples:

```
>>> None and '35'
None
>>> x = []
>>> x and 12
[]
>>> 10 and 20
20
```

In these examples, `None` and `[]` evaluate to `False`, so the `and` operator returned them in both cases since they are the first value. `10` evaluates to `True`, so the `and` operator returned the second value.

Or

With `or`, the first value is returned if it evaluates to `True`. Look at this:

```
>>> None or '35'
'35'
```

Since `None` evaluates to `False`, the `'35'` is returned.

Here's a couple more examples:

```
>>> None or '35'
'35'
>>> x = []
>>> x or 12
12
>>> 10 or 20
10
```

Here, `None` and `[]` evaluate to `False`, so the `or` operator returned the second value in both cases. `10` evaluates to `True`, so the `or` operator returned the first value.

What are "Truthy" and "Falsy" values?

In Python, when a value evaluates to `True` we call it "Truthy". Similarly, when it evaluates to `False` we call it "Falsy".

Most values in Python are "Truthy". What that means is when you pass them to the `bool()` function, it returns `True`:

```
>>> bool(None)
False
>>> bool('35')
True
>>> bool(10)
True
>>> bool([])
False
>>> bool([1, 2, 3])
True
```

The values that are "Falsy" are:

- Empty strings, lists, tuples, dictionaries, and other empty

- `None`
- `0` and other zero numbers (such as `0.0`).
- All "Falsy" values can be seen in this link.

Recap

- The `or` keyword returns the value in front of it (the first value) if it is "Truthy". Otherwise it returns the second value.
- The `and` keyword returns the first value if it is "Falsy", otherwise it returns the second value.

Jose Salvatierra
I'm a software engineer turned instructor! I founded Teclado to help me do this for everyone. I hope you enjoy the content!

Read More

Learn Python Programming

Introduction to Object-Oriented Programming in Python →

Dictionary Merge and Update Operators in Python 3.9 →

Working with Python virtual environments: the complete guide →

LEARN PYTHON PROGRAMMING

Python's modulo operator and floor division

Learn about the some less well-known operators in Python: modulo and floor division—as well as how they interact with each other and how they're related!

Phil Best
4 min read · Mar 26

Share this