Python Boolean Logic

Table of contents

The basic operands of logic are `True`, `False`, `or`, `and`, and `not`.

Boolean domain and predicates

Let $\mathbb{B}$ denote the **boolean domain**, $\mathbb{B} = \{\texttt{True}, \texttt{False}\}$. Any function that maps into $\mathbb{B}$ (i.e. one that evaluates to `True` or `False`) is called a **predicate** — e.g. $>: \mathbb{Z} \times \mathbb{Z} \to \mathbb{B}$ ("greater than").

```
>>> type(True), type(False)
(<class 'bool'>, <class 'bool'>)
>>> 7 > 3
True
>>> 7 >= 7 + 1        # addition happens first
False
>>> (7 >= 7) + 1
2
>>> int(True), int(False)
(1, 0)
```

and, or, not

and ($\mathbb{B} \times \mathbb{B} \to \mathbb{B}$) is true only when *both* inputs are true:

and	True	False
True	True	False
False	False	False

or ($\mathbb{B} \times \mathbb{B} \to \mathbb{B}$) is true when *at least one* input is true (false only when both are false):

or	True	False
True	True	True
False	True	False

```
>>> 3 > 7 or 7 > 3
True
>>> 3 > 7 and 7 > 3
False
>>> 0 < 3 and 3 < 8
True
>>> 0 < 3 < 8          # shorthand for the above
True
```

`True or False and False` is ambiguous without a precedence rule, since (`True or False`) `and False` = `False` but `True or (False and False)` = `True`. Python resolves this with **and** having **higher precedence than or** (so the expression above evaluates to `True`, as if bracketed `True or (False and False)`).

not is the negation of a logical statement (`not True == False`, `not False == True`) and is evaluated *first*, before `and`/`or`:

```
>>> a, b = 6, 7
>>> not (a == 6 and b != 5)
False
>>> not a == 6 and b != 5   # not happens before and
False
```

Avoid double negations (e.g. `not (a != 6 or not b != 5)`) — they're valid but hard to read.

Short circuits (lazy evaluation)

`or` stops as soon as it finds a truthy value; `and` stops as soon as it finds a falsy value — the remaining operand is never evaluated:

```
>>> True or 1/0
True
>>> False or 1/0
ZeroDivisionError: division by zero
>>> True or non_existent_variable
True   # Python never bothers looking up non_existent_variable
>>> True and 1/0
ZeroDivisionError: division by zero
>>> False and 1/0
False
```

Truthiness

Any object can be converted to a boolean with `bool`. **Truthy** values evaluate to `True` (non-zero numbers, non-empty strings/tuples/lists); **falsy** values evaluate to `False` (zero, the empty string, the empty tuple/list):

```
>>> bool(1), bool(-10), bool("Hello"), bool([1, 2, 3])
(True, True, True, True)
>>> bool(0), bool(""), bool([])
(False, False, False)
```

What `and`/`or` actually return

`and` and `or` don't always return `True`/`False` — `and` returns its first falsy input (or its last input, if none are falsy), and `or` returns its first truthy input (or its last input, if none are truthy):

```
>>> 0 or 2
2
>>> 0 and 2
0
>>> () or (1,) or (1, 2)
(1,)
>>> (1, 2) or (1,) or ()
```

```
(1, 2)
>>> () and (1,) and (1, 2)
()
>>> (1, 2) and (1,) and ()
()
```