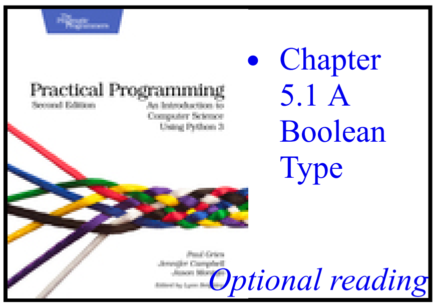
Type `bool`: Booleans in Python



- Chapter 5.1 A Boolean Type

Optional reading

Boolean values

The Python type `bool` has two values: `True` and `False`.

Comparison operators

The comparison operators take two values and produce a Boolean value.

Description	Operator	Example	Result of example
less than	`<`	`3 < 4`	`True`
greater than	`>`	`3 > 4`	`False`
equal to	`==`	`3 == 4`	`False`
greater than or equal to	`>=`	`3 >= 4`	`False`
less than or equal to	`<=`	`3 <= 4`	`True`
not equal to	`!=`	`3 != 4`	`True`

Logical operators

There are also three logical operators that produce Boolean values: `and`, `or`, and `not`.

Description	Operator	Example	Result of example
not	`not`	`not (80 >= 50)`	`False`
and	`and`	`(80 >= 50) and (70 <= 50)`	`False`
or	`or`	`(80 >= 50) or (70 <= 50)`	`True`

The `and` Logic Table

The `and` operator produces `True` if and only if both expressions are `True`.

As such, if the first operand is `False`, the second condition will not even be checked, because it is already known that the expression will produce `False`.

expr1	*expr2*	*expr1* `and` *expr2*
`True`	`True`	`True`
`True`	`False`	`False`
`False`	`True`	`False`
`False`	`False`	`False`

The `or` Logic Table

The `or` operator evaluates to `True` if and only if at least one operand is `True`.

As such, if the first operand is `True`, the second condition will not even be checked, because it is already known that the expression will produce `True`.

expr1	*expr2*	*expr1* or *expr2*
True	True	True
True	False	True
False	True	True
False	False	False

The `not` Logic Table

The `not` operator evaluates to `True` if and only if the operand is `False`.

expr1	not *expr1*
True	False
False	True

Double-negation can be simplified. For example, the expression `not not (4 == 5)` can be simplified to `4 == 5`.

Order of Precedence for Logical Operators

The order of precedence for logical operators is: `not`, `and`, then `or`. We can override precedence using parentheses and parentheses can also be added to make things easier to read and understand.

For example, the `not` operator is applied before the `or` operator in the following code:

```
>>> grade = 80
>>> grade2 = 90
>>> not grade >= 50 or grade2 >= 50
True
```

Parentheses can be added to make this clearer: `(not grade >= 50) or (grade2 >= 50)`

Alternatively, parentheses can be added to change the order of operations: `not ((grade >= 50) or (grade2 >= 50))`

Jennifer Campbell • Paul Gries
University of Toronto