BOOLEANS

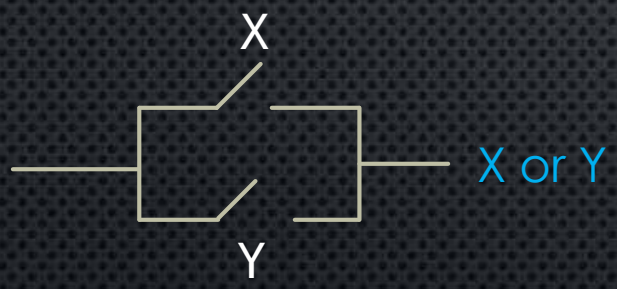
OPERATORS, PRECEDENCE AND SHORT CIRCUIT EVALUATION

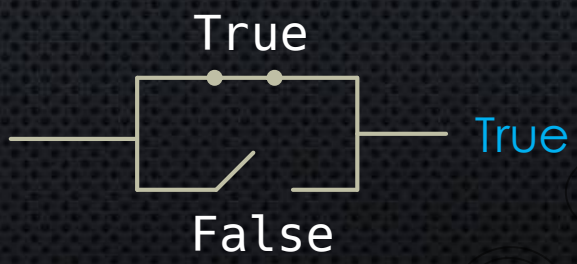
The Boolean Operators: not, and, or

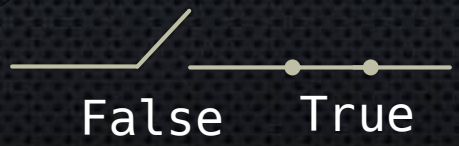
X	Y	not X	X and Y	X or Y
0	0	1	0	0
0	1	1	0	1
1	0	0	0	1
1	1	0	1	1

open False

closed True







Commutativity

```
A or B == B or A
A and B == B and A
```

Distributivity

```
A and (B or C) == (A and B) or (A and C)
A or (B and C) == (A or B) and (A or C)
```

Associativity

```
A or (B or C) == (A or B) or C
A and (B and C) == (A and B) and C
```

```
A or B or C     → (A or B) or C
A and B and C   → (A and B) and C
```

left-to-right evaluation

De Morgan's Theorem

```
not(A or B) == (not A) and (not B)
not(A and B) == (not A) or (not B)
```

Miscellaneous

```
not(x < y)     x >= y      not(x <= y) == x > y
not(x > y) == x <= y       not(x >= y) == x < y
not(not A) == A
```

Operator Precedence

```
( )
< > <= >= == != in is
not
and
or
```

highest

precedence

lowest

```
True or True and False
    → True or False    → True

(True or True) and False
    → True and False   → False
```

When in doubt, or to be absolutely sure, use parentheses!

```
True or (True and False)
```

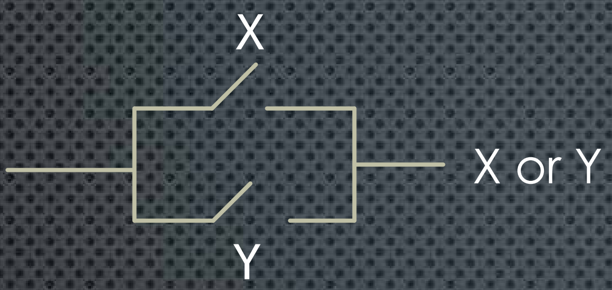
Also, use parentheses to make your code more human readable!

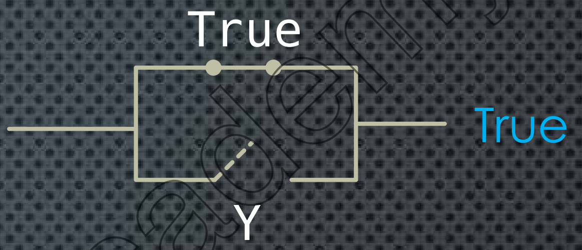
```
a < b or a > c and not x or y
```

```
(a < b) or ((a > c) and (not x)) or y
```

Short-Circuit Evaluation

X	Y	X or Y
0	0	0
0	1	1
1	0	1
1	1	1

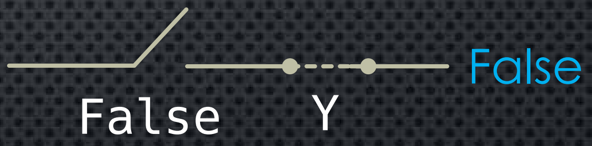




if X is True, then X or Y will be True no matter the value of Y

So, X or Y will return True without evaluating Y if X is True

X	Y	X and Y
0	0	0
0	1	0
1	0	0
1	1	1



if X is False, then X and Y will be False no matter the value of Y

So, X and Y will return False without evaluating Y if X is False

Example

Scenario: There is some data feed that lists a stock symbol, and some financial data.

Your job is to monitor this feed, looking for specific stock symbols defined in some watch list, and react only if the current stock price is above some threshold. Getting the current stock price has an associated cost.

If Boolean expressions did not implement short-circuiting, you would probably write:

```
if symbol in watch_list:
    if price(symbol) > threshold:
        # do something
```

since calling the `price()` method has a cost, you would only want to call it if the symbol was on your watch list

But because of short-circuit evaluation you could write this equivalently as:

```
if symbol in watch_list and price(symbol) > threshold:
    # do something
```

Example

`name` is a string returned from a nullable text field in a database

```
null → None
''
'abc'
```

perform some action if the first character of name is a digit (0-9)

```
if name[0] in string.digits:
    # do something
```

this code will break if `name` is `None` or an empty string

because of short-circuiting and truth values

```
if name and name[0] in string.digits:
    # do something
```

if `name` is falsy (either `None` or an empty string) then `name[0] in string.digits` is not evaluated

Code