Python Bootcamp & Masterclass

logical operators

gknxt


Operators
Arithmetic +
Comparision >
Assignment =
Logical or
Bitwise &
Membership in
Identity is

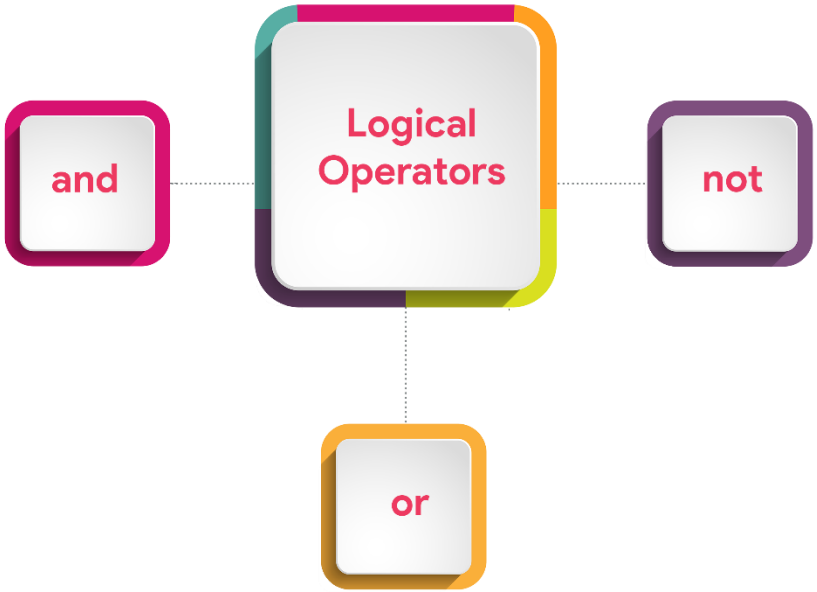
gknxt

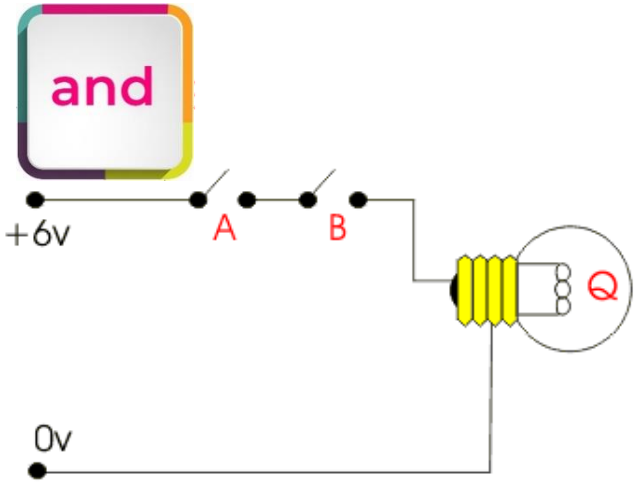
Python has three logical operators: **and**, **or** and **not**.

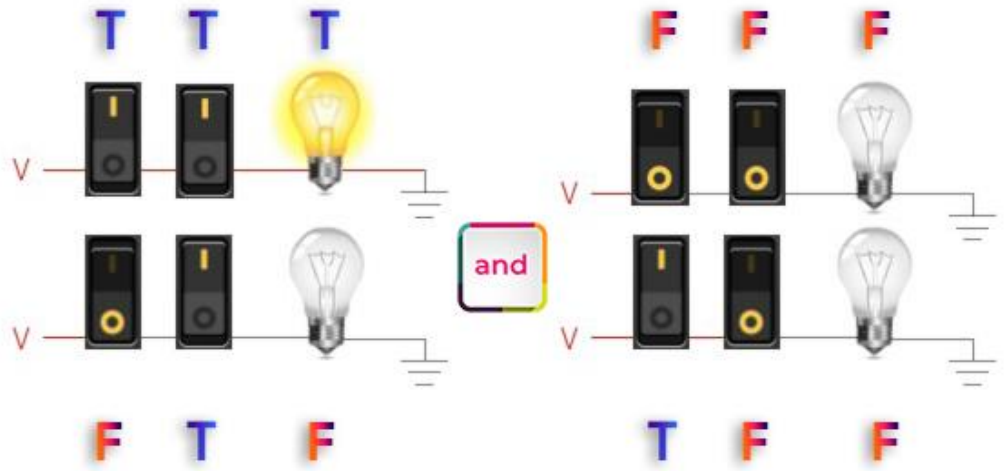
1. **and** works with two operands and evaluates to False unless both operands are True.

2. **or** works with two operands and evaluates to True unless both of its inputs are False.

3. **not** works with only one operand and returns the opposite of the operand: False for True and True for False.







Python evaluates the operand on the right of **and** operator only when it needs to. It starts evaluation from the left operand. If the operand on the left is False, there's no need to evaluate the operand on the right (regardless the result of right operand evaluation, the whole expression is False when left operand evaluates to False)

This is called short-circuit evaluation, or lazy evaluation.

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

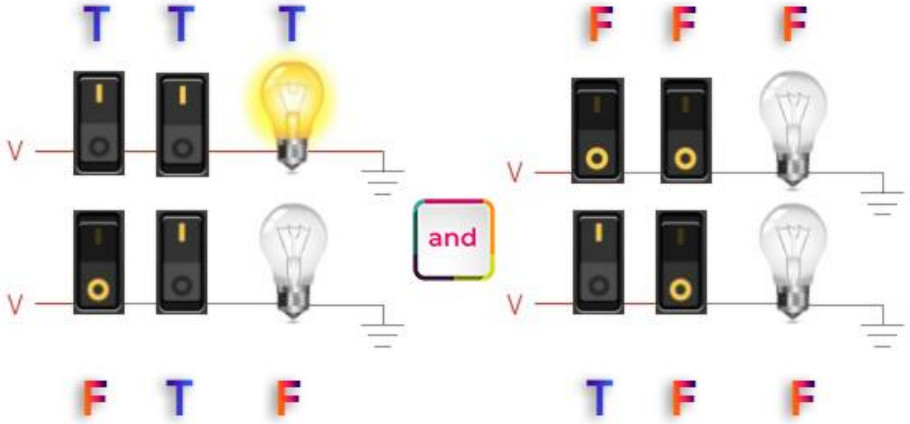
```
True

False

False

False
```

```
5 > 4 and 5 == 3 + 2
5 != 7 and not None
```

```
True

True
```

```
5 > 4 and 5 == 3 + 2 and 5 != 7
```

```
True
```



1. When **and** operator is used to combine two objects in a single expression, Python internally uses **`bool()`** to determine the truth value of the operands and returns the operand on the left if it evaluates to False. Otherwise, it returns the operand on the right.

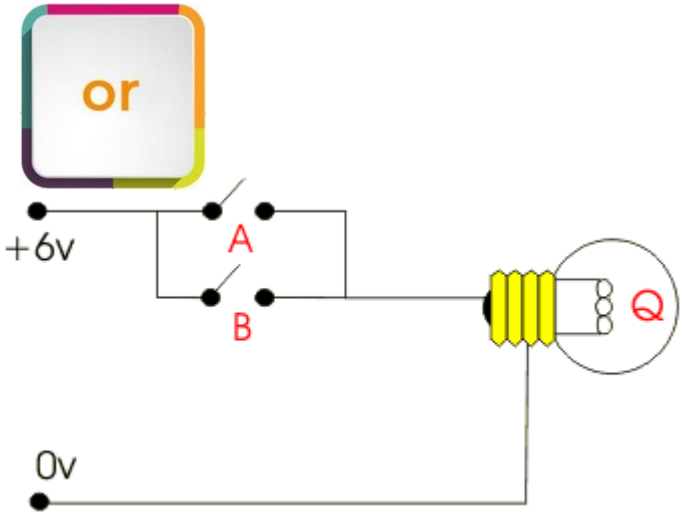
2. `None, False, 0, 0.0, 0j, Decimal(0), Fraction(0, 1), '', (), [], {}, set(), range(0)` etc. evaluates to False

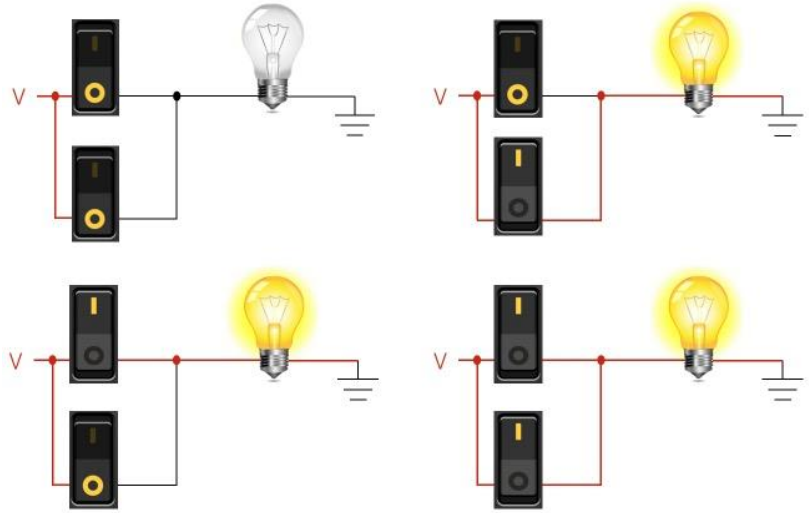
```
[] and 4
0 and 7
{} and 0/0                # divide by zero is ignored (short-circuiting)
'' and 7/0                # divide by zero is ignored (short-circuiting)
() and 'st' == 'ST'
0j and 6
False and "Hello"
'gk' == 'GK' and 0/0      # divide by zero is ignored (short-circuiting)
```

```
[]

0

{}

''

()

0j

False

False
```

```
[1, 2, 3] and []
3 > 2 and 7
'gk' != 'GK' and 'Hello'
'nxt' and 0/7
True and 18 + 3
[None] and 'Python' == 'python'
1 + 0j and 6
' ' and "Hello"
[1, 2, 3] and [4, 5, 6]
```

```
[]

7

'Hello'

0.0

21

False

6

'Hello'

[4, 5, 6]
```





Python evaluates the operand on the right of **or** operator only when it needs to. It starts evaluation from the left operand. If the operand on the left is True, there's no need to evaluate the operand on the right (regardless the result of right operand evaluation, the whole expression is True when left operand evaluates to True)

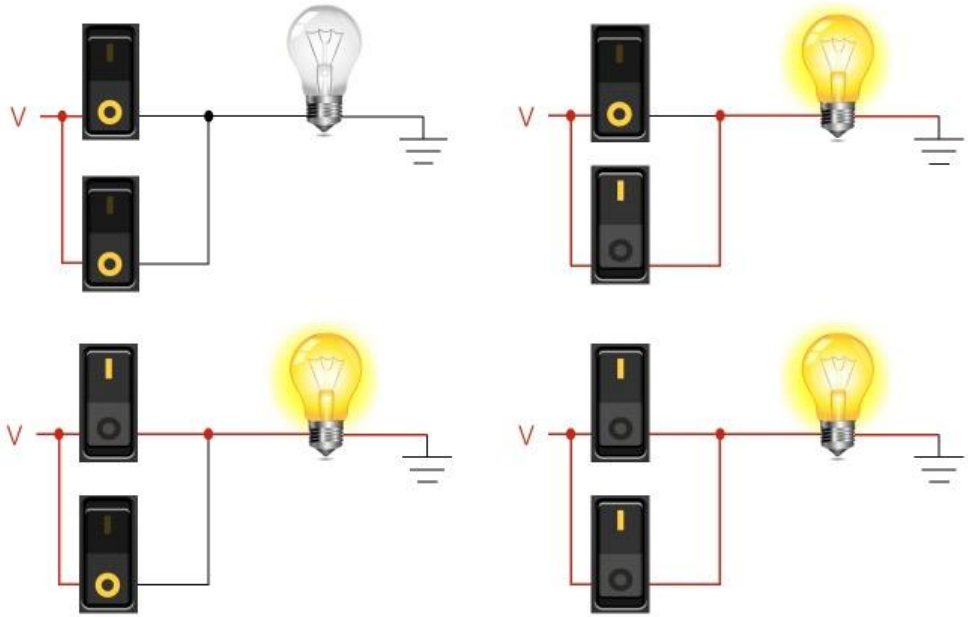
This is called short-circuit evaluation, or lazy evaluation.

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

```
True

True

True

False
```

```
x = 4
x == 3 or 0 or print(None)
```

```
None
```



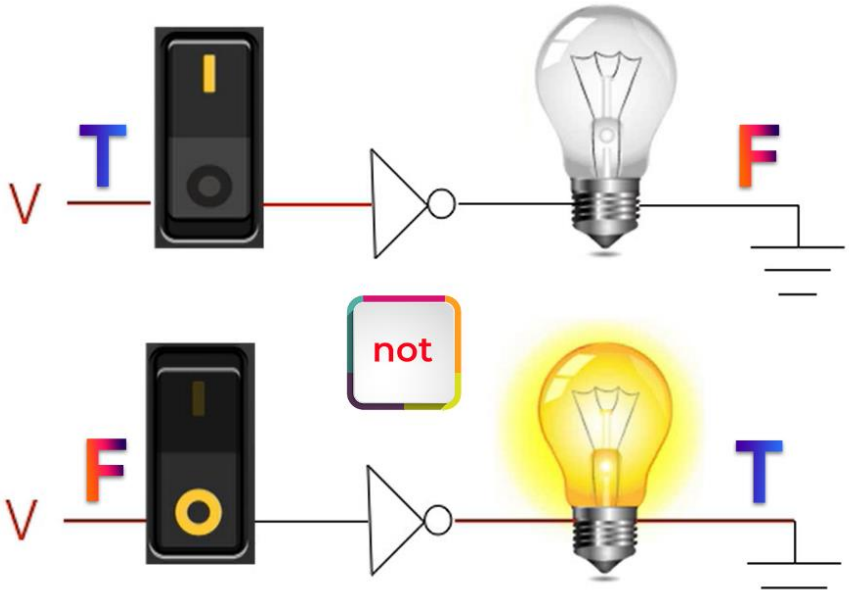
When **or** operator is used to combine two objects in a single expression, Python internally uses **`bool()`** to determine the truth value of the operands and returns the operand on the left if it evaluates to True. Otherwise, it returns the operand on the right.

```python
[1] or 4
3 > 2 or 7
'gk' != 'GK' or 0/0     # divide by zero is ignored (short-circuiting)
'nxt' or 7/0            # divide by zero is ignored (short-circuiting)
True or 18 + 3
[None] or 'Py' == 'py'
1 + 0j or 6
' ' or "Hello"
2 < 8 or 2
2 < 7.2 or []
```

```
[1]

True

True

'nxt'

True

[None]

(1+0j)

' '

True

True
```

```python
[] or 4
0 or 7
{} or 0/2
'' or 7/7
None or 21
() or 'st' == 'ST'
0j or 6
False or "Hello"
'gk' == 'GK' or 2 + 4
7 > 10 or []
5 > 10 or 5
```

```
4

7

0.0

1.0

21

False

6

'Hello'

6

[]

5
```



not operator inverts the truth value of Boolean expressions and objects. It's a unary operator (it takes only one operand) Its operand can be a Boolean expression or any object including user-defined objects. The sole task of **not** is to reverse the truth value of its operand.

```
not 0
not 0.0
not complex(0, 0)
not ''
not []
not {}
not ()
not set()
not (())
not None
not False
```

```
True

True

True

True

True

True

True

True

True

True

True
```

```
not '0'
not 0.00001
not complex(0, 1)
not ' '
not [1]
not {0}
not 42
not True
not range(1)
```

```
False

False

False

False

False

False

False

False

False

False
```

Operator Precedence

In the expression evaluation, **not** has higher priority than **and** which has higher priority than **or** and evaluation moves from left to right for equal priority operators and brackets **( )** have higher priority then all the operators

```
1 or 0 and 0
```

```
1
```

```
5 or 4 and 2 > 1
(5 or 4) and 2 > 1
```

```
5

True
```

```
1 or 0 and 0 and not []
1 or 0 and 0.0 and not [1, 2, 3]
```

```
1

1
```

```
True or True and not False
```

```
True
```

```
False and True or True or not True
```

```
True
```

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

```
True
```

Online Resources

For best python resources, please visit:

gknxt.com/python/

gknxt

Python
Bootcamp
& Masterclass

Thank You

for your Rating & Review

gknxt