BOOLEANS

Boolean Operators in Python

Boolean Operators and Truth Values

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

Normally, Boolean operators are defined to operate on and return Boolean values

`True or False` → True

```
a = 2
b = 3
```

`a > 0 and b < 5` → True

But every object in Python has a truth value (truthiness)

so, for any object X and Y, we could also write `bool(X) and bool(Y)` `bool(X) or bool(Y)`

In fact, we don't need to use `bool()` `X and Y` `X or Y`

So, what is returned when evaluating these expressions?

A Boolean? No!

Definition of or in Python

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

`X or Y` If X is truthy, returns X, otherwise returns Y

Does this work as expected when X and Y are Boolean values?

X	Y	Rule	Result
0	0	X is False, so return Y	0
0	1	X is False, so return Y	1
1	0	X is True, so return X	1
1	1	X is True, so return X	1

✓

If X is truthy, returns X, otherwise evaluates Y and returns it

Definition of and in Python

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

X and Y If X is falsy, returns X, otherwise returns Y

Does this work as expected when X and Y are Boolean values?

X	Y	Rule	Result
0	0	X is False, so return X	0
0	1	X is False, so return X	0
1	0	X is True, so return Y	0
1	1	X is True, so return Y	1

✓

If X is falsy, returns X, otherwise evaluates Y and returns it

Consequence: `or`

`X or Y` If X is truthy, returns X, otherwise evaluates and returns Y

X	Y	X or Y
`None`	`'N/A'`	`'N/A'`
`''`	`'N/A'`	`'N/A'`
`'hello'`	`'N/A'`	`'hello'`

`a = s or 'N/A'`

`if s is None`	`a → N/A`
`if s is ''`	`a → N/A`
`if s is a string with characters`	`a → s`

i.e. a will either be s or `'N/A'` if s is None or an empty string

Example

We can expand this further:

```
a = s1 or s2 or s3 or 'N/A'
```

In this case, `a` will be equal to the first truthy value (left to right evaluation)

and is guaranteed to have a value, since `'N/A'` is truthy

Example

We have an integer variable `a` that cannot be zero – if it is zero, we want to set it to `1`.

```
a = a or 1
```

Consequence: and

`X and Y`   If X is falsy, returns X, otherwise evaluates and returns Y

X	Y	X and Y
10	20/X	2
0	20/X	0

Seems like we are able to avoid a division by zero error using the and operator

```
x = a and total/a

a = 10    →    x = 10 and total/10    → total/10

a = 0     →    x = 0 and total/0      → 0
```

Example

Computing an average

`sum, n` Sometimes `n` is non-zero, sometimes it is

In either case: `avg = n and sum/n`

Example

You want to return the first character of a string `s`, or an empty string if the string is None or empty

Option 1

```
if s:
    return s[0]
else:
    return ''
```

Option 2

`return s and s[0]` → doesn't handle `None` case

```
return (s and s[0]) or ''
```

The Boolean not

not is a built-in function that returns a Boolean value

not x → True if x is falsy

→ False if x is truthy

[] → falsy	not [] → True
[1, 2] → truthy	not [1, 2] → False
None → falsy	not None → True

Code