BOOLEANS

Object Truth Values

Objects have Truth Values

All objects in Python have an associated truth value

We already saw this with integers (although to be fair, `bool` is a subclass of `int`)

But this works the same for any object

In general, the rules are straightforward

Every object has a `True` truth value, except:

- `None`
- `False`
- `0` in any numeric type (e.g. `0`, `0.0`, `0+0j`, …)
- empty sequences (e.g. `list`, `tuple`, `string`, …)
- empty mapping types (e.g. `dictionary`, `set`, …)
- custom classes that implement a `__bool__` or `__len__` method that returns `False` or `0`

which have a `False` truth value

Under the hood

Classes define their truth values by defining a special instance method:

`__bool__(self)` (or `__len__`)

Then, when we call `bool(x)` Python will actually executes `x.__bool__()`

or `__len__` if `__bool__` is not defined

if neither is defined, then `True`

Example: Integers

```
def __bool__(self):
    return self != 0
```

When we call `bool(100)` Python actually executes `100.__bool__()`

and therefore returns the result of `100 != 0` which is `True`

When we call `bool(0)` Python actually executes `0.__bool__()`

and therefore returns the result of `0 != 0` which is `False`

We will cover this and many other special functions in a later section

Examples

```
bool([1, 2, 3]) → True

bool([]) → False

bool(None) → False

bool('abc') → True
bool('') → False

bool(0) → False
bool(0 + 0j) → False
bool(Decimal('0.0') → False

bool(-1) → True
bool(1 + 2j) → True
bool(Decimal('0.1') → True
```

```
if my_list:
    # code block
```

code block will execute if and only if my_list is both not None and not empty

this is equivalent to:

```
if my_list is not None and len(my_list) > 0:
    # code block
```

Code