FLOATS

Equality Testing

In the previous video we saw that some decimal numbers (with a finite representation) cannot be represented with a finite binary representation

This can lead to some "weirdness" and bugs in our code (but not a Python bug!!)

```
x = 0.1 + 0.1 + 0.1      format(x, '.25f') → 0.3000000000000000444089210
y = 0.3                  format(y, '.25f') → 0.2999999999999999888977698

x == y   → False
```

Using rounding will not necessarily solve the problem either!

It is no more possible to exactly represent `round(0.1, 1)` than `0.1` itself

```
round(0.1, 1) + round(0.1, 1) + round(0.1, 1) == round(0.3, 1)   → False
```

But it can be used to round the entirety of both sides of the equality comparison

```
round(0.1 + 0.1 + 0.1, 5) == round(0.3, 5)  → True
```

To test for "equality" of two different floats, you could do the following methods:

round both sides of the equality expression to the number of significant digits

```
round(a, 5) == round(b, 5)
```

or, more generally, use an appropriate range (ε) within which two numbers are deemed equal

for some , if and only if | |

```
def is_equal(x, y, eps)
    return math.fabs(x-y) < eps
```

This can be tweaked by specifying that the difference between the two numbers be a percentage of their size → the smaller the number, the smaller the tolerance

i.e. are two numbers within x% of each other?

But there are non-trivial issues with using these seemingly simple tests

→ numbers very close to zero vs away from zero

Using absolute tolerances...

```
x = 0.1 + 0.1 + 0.1
y = 0.3
print(format(x, '.20f'))   → 0.30000000000000004441
print(format(y, '.20f'))   → 0.29999999999999998890
```

17th digit after decimal pt

Δ = 0.00000000000000005551
0.000000000000001

```
a = 10000.1 + 10000.1 + 10000.1
b = 30000.3
print(format(a, '.20f'))   → 30000.30000000000291038305
print(format(b, '.20f'))   → 30000.29999999999927240424
```

12th digit after decimal pt

Δ = 0.00000000000363797881
0.000000000000001

Using an absolute tolerance: `abs_tol` = 10^{-15} = 0.000000000000001

then

```
math.fabs(x - y) < abs_tol    → True

math.fabs(a - b) < abs_tol    → False
```

Maybe we should use relative tolerances…

```
x = 0.1 + 0.1 + 0.1
y = 0.3
```
→ `tol = 0.000003000000000`

```
a = 10000.1 + 10000.1 + 10000.1
b = 30000.3
```
→ `tol = 0.300003000000000`

Using a relative tolerance: `rel_tol = 0.001% = 0.00001 = 1e-5`

i.e. maximum allowed difference between the two numbers,
relative to the larger magnitude of the two numbers

```
tol = rel_tol * max(|x|, |y|)

math.fabs(x - y) < tol        → True

math.fabs(a - b) < tol        → True
```

Success! but is it really?

```
x = 0.0000000001      (1e-10)
y = 0
```

Using a relative tolerance: `rel_tol = 0.1% = 0.0001 = 1e-3`

```
tol = rel_tol * max(|x|, |y|) → tol = rel_tol * |x|  → 1e-3 * 1e-10 = 1e-13

math.fabs(x - y) < abs_tol     → False
```

Using a relative tolerance technique does not work well for numbers close to zero!

So using absolute and relative tolerances, in isolation, makes it difficult to get a one-size-fits-all solution

We can combine both methods

calculating the absolute and relative tolerances

and using the larger of the two tolerances

```
tol = max(rel_tol * max(|x|, |y|), abs_tol)
```

→ PEP 485

The math module has that solution for us! → PEP 485

```
math.isclose(a, b, *, rel_tol=1e-09, abs_tol=0.0)
```

⚠ If you do not specify abs_tol, then it defaults to 0 and you will face the problem we encountered in the last slide when comparing numbers close to zero.

```
x = 1000.0000001                          a = 0.0000001
y = 1000.0000002                          b = 0.0000002

math.isclose(x, y)   → True               math.isclose(a, b)   → False
```

but

```
math.isclose(x, y, abs_tol=1e-5) → True   math.isclose(a, b, abs_tol=1e-5)  → True
```

Also works well in situations like this:

```
x = 1000.01
y = 1000.02

math.isclose(x, y, rel_tol=1e-5, abs_tol=1e-5)      → True
```

```
a = 0.01
b = 0.02

math.isclose(x, y, rel_tol=1e-5, abs_tol=1e-5)      → False
```

If you are going to be using this method, you should play around with it for a while until you get a good feel for how it works

Code