When we use numbers on our computers their values are (usually) stored in the IEEE 754 floating point format. Usually you don't need to know anything about this format, but occasionally it can cause unexpected behavior. I'll just briefly describe what "floating point" even means, and then tell you about a few properties of floating point numbers that may surprise you.

Floating point numbers use the equivalent of scientific notation, e.g. 1.464844×10^{-2}. You will notice that this number has two parts. The *mantissa* is the number 1464844, and the exponent is -2. On a computer, these are stored in binary, so this number would look more like 15×2^{-10}, with the mantissa being 15 and the exponent being -10. However, to keep things efficient, there are a fixed number of bits (like digits) that are used for the mantissa, and a fixed number of bits for the exponent. This means that many real numbers cannot be represented exactly, including real numbers that *can* be represented exactly in decimal notation.

Common gotchas:

roundoff error Numbers must be rounded to the nearest representable number after most (but not all) arithmetic operations, so frequently you will find equations not quite as you would expect. For instance `1.0 - 0.0001 + 0.0005 + 0.0005 == 1.0` gives a value of `False`. Roundoff error makes comparing for equality with floating point numbers usually a mistake.

arange When using `np.arange`, you can see unpredictable results if your end value is an integer times your step size.

infinity When dividing by zero with `numpy` arrays, you will get infinite values, which a special code that indicates an infinite value.

```
zero = np.zeros(1)
inf = 1/zero
print(inf)
print(1/inf) # this gives us a well-defined result of 0
print(inf-inf) # this gives us an undefined result
```

NaN A `NaN` is 'Not a Number', and is the value you get when you do a computation that has an undefined result (unless Python crashes, which is possibly hard to predict). NaNs are strange beasts that are unequal to themselves, and any arithmetic involving a NaN produces a NaN. The most common origin of NaNs is probably taking the square root of a negative number.