FLOATS

Internal Representation

The float class is Python's default implementation for representing real numbers

The Python (CPython) float is implemented using the C double type which (usually!) implements the IEEE 754 double-precision binary float, also called binary64

The float uses a fixed number of bytes → 8 bytes *(but Python objects have some overhead too)*

→ 64 bits → 24 bytes (CPython 3.6 64-bit)

These 64 bits are used up as follows:

sign → 1 bit

exponent → 11 bits → range [-1022, 1023] `1.5E-5` → 1.5×10^{-5}

exponent is –5

significant digits → 52 bits → 15-17 significant (base-10) digits

significant digits → for simplicity, all digits except leading and trailing zeros

`1.2345   1234.5   12345000000   0.00012345   12345e-50   1.2345e10`

Representation: decimal

Numbers can be represented as base-10 integers and fractions:

$0.75 \rightarrow \frac{7}{10} + \frac{5}{100} \rightarrow 7 \times 10^{-1} + 5 \times 10^{-2}$ 2 significant digits

$0.256 \rightarrow \frac{2}{10} + \frac{5}{100} + \frac{6}{1000} \rightarrow 2 \times 10^{-1} + 5 \times 10^{-2} + 6 \times 10^{-3}$ 3 significant digits

$123.456 \rightarrow 1 \times 100 + 2 \times 10 + 3 \times 1 + \frac{4}{10} + \frac{5}{100} + \frac{6}{1000}$ 6 significant digits

$\rightarrow 1 \text{ x } 10^2 + 2 \times 10^1 + 3 \times 10^0 + 4 \times 10^{-1} + 5 \times 10^{-2} + 6 \times 10^{-3}$

In general: $d = \sum_{i=-m}^{n} d_i \times 10^i$

$sign$ = 0 for positive
$sign$ = 1 for negative

$$d = (-1)^{sign} \sum_{i=-m}^{n} d_i \times 10^i$$

Some numbers cannot be represented using a finite number of terms

$$d = (-1)^{sign} \sum_{i=-m}^{n} d_i \times 10^i$$

Obviously numbers such as

$$\pi = 3.14159\ldots$$

$$\sqrt{2} = 1.4142\ldots$$

but even some rational numbers

$$\frac{1}{3} = 0.33\dot{3}$$

$$= \frac{3}{10} + \frac{3}{100} + \frac{3}{1000} + \ldots$$

Representation: binary

Numbers in a computer are represented using bits, not decimal digits

→ instead of powers of 10, we need to use powers of 2

$$(0.11)_2 = \left(\frac{1}{2} + \frac{1}{4}\right)_{10} = (0.5 + 0.25)_{10} = (0.75)_{10}$$

$$= (1 \times 2^{-1} + 1 \times 2^{-2})_{10}$$

Similarly,

$$(0.1101)_2 = \left(\frac{1}{2} + \frac{1}{4} + \frac{0}{8} + \frac{1}{16}\right)_{10} = (0.5 + 0.25 + 0.0625)_{10} = (0.8125)_{10}$$

$$= (1 \times 2^{-1} + 1 \times 2^{-2} + 0 \times 2^{-3} + 1 \times 2^{-4})_{10}$$

This representation is very similar to the one we use with decimal numbers

but instead of using powers of 10, we use powers of 2

a binary representation

$$d = (-1)^{sign} \sum_{i=-m}^{n} d_i \times 2^i$$

The same problem that occurs when trying to represent $\frac{1}{3}$ using a decimal expansion also happens when trying to represent certain numbers using a binary expansion

$$0.1 = \frac{1}{10}$$

Using binary fractions, this number does not have a finite representation

base 10

$$(0.1)_{10} = (0.0\,0011\,0011\,0011\ \ldots)_2$$

$$= \frac{0}{2} + \frac{0}{4} + \frac{0}{8} + \frac{1}{16} + \frac{1}{32} + \frac{0}{64} + \frac{0}{128} + \frac{1}{256} + \frac{1}{512} + \frac{0}{1024} + \frac{0}{2048} + \frac{1}{4096} + \frac{1}{8192} + \ \ldots$$

$$= \frac{1}{16} + \frac{1}{32} + \frac{1}{256} + \frac{1}{512} + \frac{1}{4096} + \frac{1}{8192} + \ \ldots$$

$$= 0.0625 + \frac{1}{32} + \frac{1}{256} + \frac{1}{512} + \frac{1}{4096} + \frac{1}{8192} + \ \ldots$$

$$= 0.09375 + \frac{1}{256} + \frac{1}{512} + \frac{1}{4096} + \frac{1}{8192} + \ \ldots$$

$$= 0.09765625 + \frac{1}{512} + \frac{1}{4096} + \frac{1}{8192} + \ \ldots$$

$$= 0.099609375 + \frac{1}{4096} + \frac{1}{8192} + \ \ldots$$

$$= 0.0999755859375 + \ \ldots$$

So, some numbers that do have a finite decimal representation,
do not have a finite binary representation,
and some do

$(0.75\,)_{10} = \;(0.11)_2$ finite

$(0.8125\,)_{10} = (0.1101)_2$ finite

→ exact float representation

$(0.1)_{10} = (0\;0011\;0011\;0011\;\;\ldots)_2$ infinite → approximate float representation

Code