Example 23. Python As our next upgrade, let us collect the digits in a list instead of printing them to the screen. Here is how we can create a list in Python and add an element to it:

```
>>> L = [1, 2, 3]

>>> L.append(4)

>>> print(L)

   [1, 2, 3, 4]
```

Here is our code adjusted for using a list (and now it is more pleasant to ask for more digits):

```
>>> x = 0.1  # or any value < 1
    nr_digits = 10  # we want this many digits of x
    digits = []  # this list will store the digits of x
    from math import trunc
    for i in range(nr_digits):
        x = 2*x
        digit = trunc(x)
        digits.append(digit)
        x = x-digit
    print(digits)

   [0, 0, 0, 1, 1, 0, 0, 1, 1, 0]
```

Example 24. Python For our final upgrade, we collect the code into a function that we call `fracpart_digits`. This is crucial for making it possible to use the code on different numbers.

```
>>> def fracpart_digits(x, nr_digits):
        digits = []
        from math import trunc
        for i in range(nr_digits):
            x = 2*x
            digit = trunc(x)
            digits.append(digit)
            x = x-digit
        return digits
```

We are now able to compute the digits of numbers by simply calling our function:

```
>>> fracpart_digits(0.1, 10)

   [0, 0, 0, 1, 1, 0, 0, 1, 1, 0]

>>> fracpart_digits(0.2, 10)

   [0, 0, 1, 1, 0, 0, 1, 1, 0, 0]

>>> from math import pi

>>> fracpart_digits(pi/4, 10)

   [1, 1, 0, 0, 1, 0, 0, 1, 0, 0]
```

Comment. Recall that, if you are not in a Python console, you need to add `print(..)` to see any output.

As an advanced use of lists, here is how we could compute 5 digits of $1/n$ for $n \in \{2, 3, 4, 5\}$:

```
>>> [fracpart_digits(1./n, 5) for n in range(2,6)]

   [[1, 0, 0, 0, 0], [0, 1, 0, 1, 0], [0, 1, 0, 0, 0], [0, 0, 1, 1, 0]]
```

Comment. Note how the digits of $1/2 = (0.1)_2$ and $1/4 = (0.01)_2$ are particularly easy to verify.

Example 25. One of the most famous/notorious mathematical results is **Fermat's last theorem**. It states that, for $n>2$, the equation $x^n+y^n=z^n$ has no positive integer solutions!

Pierre de Fermat (1637) claimed in a margin of Diophantus' book *Arithmetica* that he had a proof ("I have discovered a truly marvellous proof of this, which this margin is too narrow to contain.").

It was finally proved by Andrew Wiles in 1995 (using a connection to modular forms and elliptic curves).

This problem is often reported as the one with the largest number of unsuccessful proofs.

On the other hand, in a Simpson's episode, Homer discovered that

$$1782^{12}+1841^{12} \text{ “=” } 1922^{12}.$$

If you check this on an old calculator it might confirm the equation. However, the equation is not correct, though it is "nearly": $1782^{12}+1841^{12}-1922^{12}\approx -7.002\cdot 10^{29}$.

Why would that count as "nearly"? Well, the smallest of the three numbers, $1782^{12}\approx 1.025\cdot 10^{39}$, is bigger by a factor of more than 10^9. So the difference is extremely small in comparison.

More precisely, if $1782^{12}+1841^{12}$ is the true value, then approximating it with 1922^{12} produces

- an absolute error of $|1782^{12}+1841^{12}-1922^{12}|\approx 7.00\cdot 10^{29}$ (rather large), and
- a relative error of $\left|\frac{1782^{12}+1841^{12}-1922^{12}}{1782^{12}+1841^{12}}\right|\approx 2.76\cdot 10^{-10}$ (very small).

Comment. We can immediately see that Homer is not quite correct by looking at whether each term is even or odd. Do you see it?

http://www.bbc.com/news/magazine-24724635

Example 26. Strangely, $e^\pi-\pi=19.999099979\ldots$. Determine both the absolute and the relative error when approximating this number by 20.

https://xkcd.com/217/

Solution. The absolute error is $|20-(e^\pi-\pi)|\approx 9.0\cdot 10^{-4}$.

The relative error is $\left|\frac{20-(e^\pi-\pi)}{e^\pi-\pi}\right|\approx 4.5\cdot 10^{-5}$.

Solving equations

Now that we have discussed how computers deal with numbers, it is natural to think about how to compute numbers of interest. Often, these arise as solutions of equations.

For instance. As simple but instructive instances, how do we compute numbers like $\sqrt{2}$, $\log(3)$ or π?

Note that any equation can be put into the form $f(x)=0$ where $f(x)$ is some function. Solving that equation is equivalent to finding roots of that function.

There are many approaches to root finding see, for instance:

https://en.wikipedia.org/wiki/Root-finding_algorithms

Comment. The solve routines implemented in professional libraries often use hybrid versions of the methods we discuss below (as well as others). For instance, Brent's method (used, for instance, in MATLAB, PARI/GP, R or SciPy) is a hybrid of three: the bisection and secant methods as well as inverse quadratic interpolation.

Comment. It depends very much on $f(x)$ which approach to root finding is best. For instance, is $f(x)$ a nice (i.e. differentiable) function? Is it costly to evaluate $f(x)$? This is the reason for why there are many different approaches to finding roots and why it is important to understand their strenghts and weaknesses.

The bisection method

Suppose that we wish to find a root of the continuous function $f(x)$ in the interval $[a,b]$.

If $f(a)<0$ and $f(b)>0$, then the **intermediate value theorem** tells us that there must be an $r\in[a,b]$ such that $f(r)=0$. (Likewise if $f(a)>0$ and $f(b)<0$.)

Comment. Recall that the intermediate value theorem requires $f(x)$ to be continuous so that there are no jumps or singularities.

The **bisection method** now cuts the interval $[a,b]$ into two halves by computing the **midpoint** $c=\frac{a+b}{2}$. Depending on whether $f(c)\leqslant 0$ or $f(c)\geqslant 0$, we conclude that there must be a root in $[a,c]$ or in $[c,a]$. In either case, we have cut the length of the interval of uncertainty in half.

This process is then repeated until we have a sufficiently small interval that is guaranteed to contain a root of $f(x)$.

Example 27. Determine an approximation for $\sqrt[3]{2}$ by applying the bisection method to the function $f(x)=x^3-2$ on the interval $[1,2]$. Perform 4 steps of the bisection method.

Comment. Note that it is obvious that $1<\sqrt[3]{2}<2$ so that the interval $[1,2]$ is a natural choice.

Solution. Note that $f(1)=-1<0$ while $f(2)=6>0$. Hence, $f(x)$ must indeed have a root in the interval $[1,2]$.

- $[a,b]=[1,2]$ contains a root of $f(x)$ (since $f(a)<0$ and $f(b)>0$).

  The midpoint is $c=\frac{a+b}{2}=\frac{1+2}{2}=\frac{3}{2}$. We compute $f(c)=c^3-2=\frac{27}{8}-2=\frac{11}{8}>0$.

  Hence, $[a,c]=\left[1,\frac{3}{2}\right]$ must contain a root of $f(x)$.

- $[a,b]=\left[1,\frac{3}{2}\right]$ contains a root of $f(x)$ (since $f(a)<0$ and $f(b)>0$).

  The midpoint is $c=\frac{a+b}{2}=\frac{1+3/2}{2}=\frac{5}{4}$. We compute $f(c)=-\frac{3}{64}<0$.

  Hence, $[c,b]=\left[\frac{5}{4},\frac{3}{2}\right]$ must contain a root of $f(x)$.

- $[a,b]=\left[\frac{5}{4},\frac{3}{2}\right]$ contains a root of $f(x)$ (since $f(a)<0$ and $f(b)>0$).

  The midpoint is $c=\frac{a+b}{2}=\frac{11}{8}$. We compute $f(c)=\frac{307}{512}>0$.

  Hence, $[a,c]=\left[\frac{5}{4},\frac{11}{8}\right]$ must contain a root of $f(x)$.

- $[a,b]=\left[\frac{5}{4},\frac{11}{8}\right]$ contains a root of $f(x)$ (since $f(a)<0$ and $f(b)>0$).

  The midpoint is $c=\frac{a+b}{2}=\frac{21}{16}$. We compute $f(c)=\frac{1069}{4096}>0$.

  Hence, $[a,c]=\left[\frac{5}{4},\frac{21}{16}\right]$ must contain a root of $f(x)$.

After 4 steps of the bisection method, we know that $\sqrt[3]{2}$ must lie in the interval $\left[\frac{5}{4},\frac{21}{16}\right]=[1.25,1.3125]$.

Comment. For comparison, $\sqrt[3]{2}\approx 1.2599$. Note that our approximations are a bit more impressive if we think in terms of binary digits. Then each step provides one additional digit of accuracy (because the length of the interval of uncertainty is cut by half).

Comment. The above steps are on purpose written in a repetitive manner (reusing the same variable names a, b,c with new values) to make it easier to translate the process into Python code.