Algorithms - a simple introduction in Python: Part Two

A computational process is indeed much like a sorcerer's idea of a spirit. It cannot be seen or touched. It is not composed of matter at all. However, it is very real. It can perform intellectual work. It can answer questions. It can affect the world ...

Abelson and Sussman: *The Structure and Interpretation of Computer Programs.*[1]

Seeing as we looked at squaring last time, it seems like a good idea to explore square roots. This is a little more complex.

First, let's define a simple function "my_abs" which will give us the absolute value of a number.

```
def my_abs(x):
    """Returns the absolute value of x."""
    if x < 0:
        return -x
    else:
        return x
```

Strictly speaking, we don't need to do this, as "abs()" is built-in to Python. But I think it is interesting to see how it works.

Next we need to work out how to find a square root. The algorithm for this is usually attributed to the ancient mathematician Heron of Alexandria.

To find the square root of x, first make a guess.
Then keep taking the average of the guess and x divided by the guess, until you have an accurate enough result.

```
# sqrt.py - finding square roots

ERROR = 0.0000000001

def square(x):
    """Returns the square of a number."""
    return x*x

def average(x,y):
    """Returns the average of two numbers."""
    return (x + y) / 2

def my_abs(x):
    """Returns the absolute value of x."""
    if x < 0:
        return -x
    else:
        return x

def sqrt(x, error):
    """Returns an approximation to the square root.
        Using Heron of Alexandria's algorithm."""
    guess = 1
    while my_abs(square(guess) - x) > error:
        guess = average(guess, x / guess)
    return guess

#main
print(sqrt(float(input("Square Root of: ")), ERROR))
```

I've used a global definition of "ERROR". I used capitals to show this is a *constant*. As using global variables is frowned upon by programmers, I pass this to the sqrt() function along with the number.

Type this program up and run it. Make sure you understand all the functions and how the code works. You can alter the margin of error to see what happens.

[1] This wonderful book can be found at: http://mitpress.mit.edu.sicp.

What we are actually doing here is trying to find a “fixed point” for the function:

$$f\left(n\right)=\frac{n+\frac{x}{n}}{2}$$

Where x is the number we are finding the square root of.

The fixed-point is a value which will not change when it is passed to the function. For instance, if we guessed that 5 was the square root of 25:

$$\frac{5+\frac{25}{5}}{2}=5$$

Fixed points are very useful things, and we’ll use them again when looking at other algorithms.

Recursion
Here’s an alternative way to write the square root function.

```
def sqrt(x, guess, error):
    """Recursive version."""
    if my_abs(square(guess) - x) < error:
        return guess
    else:
        return sqrt(x, average(guess, x / guess), error)

#main
print(sqrt(float(input("Square Root of: ")), 1, ERROR))
```

The major change here is that the function is defined in terms of itself.

If the guess is not accurate enough, sqrt calls itself, passing the average of guess plus x divided by the guess as the new guess.

This means we also have to add a guess (it’s simplest always to guess 1 the first time!) to the initial call to the function.

Recursive definitions like this are fun, however there are some cases in which they are much less efficient than “iterative” definitions (the way we did it the first time).

Task

Isaac Newton found the following function to use when looking for cube roots. Use it to adapt the square root program so that it finds cube roots instead. If x is the number for which we want the cube root:

$$f(n)=\frac{\frac{n}{x^2}+2x}{3}$$