PHAS1240 Session 2
While loops: iteration and convergence

October 7, 2015

At the end of this session, you should understand:

- How we can use a combination of loops and comparison/conditional operators to *iterate* processes so that we can converge on a solution;
- Some of the problems that can arise when you try and do this in practice;
- Implications for computational physics research problems.

---

1. Introduction

At the end of the first session, we briefly looked at the "if" statement and how to use it.

To make this really powerful, we also need a way of repeating the same piece of code again and again until a condition is met.

Python has two ways of doing this - the "while" loop and the "for" loop. We'll look at the "for" loop in the next session, and concentrate on the "while" loop for now.

---

2. "while" loops

2.1 How do they work?

- They have the same structure as the "if" statement: the first line ends with a colon, and the block of code to loop over is indented.
- It will repeat the indented block *while* the comparison in the first line is true.

Let's look at a simple example to increment the value of a variable from 1 to 10. Copy and paste the following code into the Spyder editor window, save it (using a suitable filename), and run it. You should obtain the same output as here:

```
In [1]: x = 0 # initialise x

        while x <= 10: # start the loop,
            # and continue looping until x becomes greater than 10
            print x # print the current value of x
            x = x + 1 # increment x

        print "We counted to 10!"
```

```
0
1
2
3
4
5
6
7
8
9
10
We counted to 10!
```

Let's analyse this more closely. The condition on the loop is "while x <= 10:", so as long as the value of x is less than or equal to 10, the indented block of code will repeat. As soon as x is greater than 10, Python will skip to the first line of *unindented* code, which will print out "We counted to 10!"

Within the indented block, we have just two lines of code in this case. The first prints the current value of x, but look carefully at the second line. "x = x + 1" takes the current value of x, adds one to it, and replaces the old value of x with the new one. This is, as we'll see, very useful to act as a loop counter.

You'll also sometimes see this written as x +=1, which is identical. Use whichever form you prefer (I personally find the first one clearer and easier to read).

```
In [2]: x = 0 # initialise x

        while x <= 10: # start the loop,
            # and continue looping until x becomes greater than 10
            print x # print the current value of x
            x += 1 # increment x

        print "We counted to 10 again!"
```

```
0
1
2
```

```
3
4
5
6
7
8
9
10
We counted to 10 again!
```

3. Practical applications: iteration and convergence

Being able to count to ten is good, but we (most of us) can do this on our fingers, so let's look at some more advanced applications.

One of the most practical applications of this in Physics is when we want to iterate a procedure until it meets some kind of convergence criterion. To show this we'll look at a simple series: Imagine that we want to calculate a decimal value of 1/3. Of course the simplest way of doing this is to just divide 1 by 3, but that's not very exciting. Instead we'll use a series expansion:

$$\frac{1}{3} = \frac{3}{10} + \frac{3}{100} + \frac{3}{1000} + \dots$$

$$= \sum_{n=1}^{\infty} \frac{3}{10^n}$$

We can implement this in a loop with this basic structure:

```
while <some_condition>:
    value = value + 3*10**(-n)
    n = n + 1
```

What condition should we use on the loop? We could just use a fixed number of iterations, for example "while n < 10". But this does not guarantee convergence to a defined precision, which is generally what we want.

Instead, it's more useful to exit the loop when some precision is achieved, for instance when the summation is converged to 8 decimal places. (Obviously, in order to do this, we first need to be certain that the series does actually converge mathematically. If the summation diverges, no amount of brilliant coding will generate a useful result!)

If this is OK, then we could look at the size of the final term in the series - is this smaller than our precision tolerance? If so, then our series is likely to have converged to within requirements. (See the separate document on Moodle if you want a more mathematically rigorous justification of this.)

The code we worked through in the screencast ended up like this:

```
In [3]: # sample code for series summation example

        tolerance = 1e-10 # the precision we require
        n = 0   # initialize the counter
        newterm = 1 # any number will do here as long as it is > tolerance
        value = 0 # initialize the value of the summation

        while newterm > tolerance:
            n = n + 1
            newterm = 3 * 10**(-n)
            print "Adding ", newterm, " to our sum..."
            value = value + newterm
        print "Our value is calculated as", value
        print "We used ", n, " terms in the series"
```

```
Adding  0.3  to our sum...
Adding  0.03  to our sum...
Adding  0.003  to our sum...
Adding  0.0003  to our sum...
Adding  3e-05  to our sum...
Adding  3e-06  to our sum...
Adding  3e-07  to our sum...
Adding  3e-08  to our sum...
Adding  3e-09  to our sum...
Adding  3e-10  to our sum...
Adding  3e-11  to our sum...
Our value is calculated as 0.33333333333
We used  11  terms in the series
```

This will terminate the series when the final term is less than 1e-10, meaning the value of 1/3 we obtain is likely to be accurate to at least 10 decimal places.

---

4. Task: The Madhava/Gregory/Leibniz approximation of π

The Madhava/Gregory/Leibniz series is a formula for calculating $\pi/4$:

$$\frac{\pi}{4} = \sum_{n=0}^{\infty} \frac{(-1)^n}{2n+1}$$

Write a code to calculate this series to a given convergence tolerance and compare it to numpy's value of $\pi/4$. The code should ask the user to input the value for the tolerance, and output your calculated value and the number of terms in the summation

Test your code with various tolerance values between 10^{-4} and 10^{-7}. Would we be justified in claiming that our result is accurate to n decimal places for a tolerance of 10^n?

Make a rough measurement how long it takes to calculate the sum with a tolerance of 10^{-8} - either use a stopwatch app or just count in your head. Use this, with the results

from above, to guesstimate, using appropriate/sensible units, how long it would take your code to calculate $\pi/4$ converged to 20 decimal places. Write a sentence or two at the end of your code (as # comments), together with this value, explaining whether or not you think this is a good method to calculate π.

Remember to:

- Start with a comment block with your name, one-line description of the code, and the date.
- Include explanatory comments throughout your code.
- Make sure all output has a text string explaining what the numbers represent.

Once your code is ready, submit it via Moodle and add your name to the marking list at the demonstrators' bench to have it graded.