**Programming for Science**
**Workshop 2**

This workshop aims to give you some experience of using lists and `for` loops.

Do as many of the questions as you can in the workshop, but make sure you've done all of the questions by the next workshop. Where you've written programs, please have them available at the next workshop to discuss with the people taking the workshop.

Occasionally you may write a loop that goes on forever or at least longer than you wanted (usually a mistake!). You can stop an infinite loop (in the interpreter or when running a program) by pressing control-C, that is: hold down the control key and press the C-key at the same time. This will interrupt your program and you'll see a 'KeyboardInterrupt' exception reported by the interpreter, but then everything should be back to normal.

**1.** Initially, we'll extend the example talked about in lectures. Write a Python program to:

**(a)** create a list (called `birthday`) which has as its elements the three integers corresponding to the day, month and year of your birth;

**(b)** iterate over the list, printing the elements, finding the sum of the elements, finding the product of the elements and printing the result at the end.

**2.** Create a list with the first line of the poem:

```
rhyme = ['Mary', 'had', 'a', 'little', 'lamb', 'whose', 'fleece', 'was',
'white', 'as', 'snow']
```

Now write a program to count the number of letters and words in `rhyme`. Note the `len(word)` will tell you the number of characters in the variable `word`; thus

```
word = 'This string has 29 characters'
print len(word)
```

will print 29.

**3.** We haven't learned how to read from a file yet, but the following code will make a list, here called `lines`, whose elements are the lines of the file `myfile`:

```
lines = open('myfile', 'r').readlines()
```

Create your own file (e.g., `mary.txt`) with the words of the rhyme on *separate* lines of the file and use the above line to read it into a list in the interpreter:

```
>>> lines = open('mary.txt', 'r').readlines()
```

(Note that the file must be in the same directory as the directory in which you invoke the Python interpreter.)

Inspect the contents of the list by naming it in the interpreter:

```
>>> lines
```

and note that each element of the list (corresponding to a line) ends with a newline character, `\n`. We will see how to remove these newlines later in the module.

Write a program to read your file and to count the number of (non-newline) characters in it. Extend the program to print the average length of the words in the file.

The file `words.txt` on the resources page (`http://empslocal.ex.ac.uk/studyres/ECM1408/`) is a dictionary of common (and not so common) English words. Download the file and, by changing the name of the file in the program to count the words in the rhyme, find the number of words in `words.txt` and the average length of the words in the dictionary.

**4.** In a similar way to the Simpsons example discussed in lectures, build (and then print) a list whose elements are the following list with the phrase " and eggs" added to each item of the list:

```
foods = ['ham', 'toast', 'spam', 'bacon', 'spinach']
```

**5.** In lectures we saw that elements can be appended to lists using `append`. Lists can be concatenated with the `+` operator and multiplied by a (positive) integer, rather like strings. Creating lists of your own, check how these operators work on lists.

**6.** `append` unsurprisingly adds elements to the end of a list. Items can be inserted into a list (before the $n$th) element using `insert`. For example:

```
>>> L = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h']
>>> L.insert(3, 'X')
>>> L
['a', 'b', 'c', 'X', 'd', 'e', 'f', 'g', 'h']
>>> L.insert(0, 'Y')
>>> L
['Y', 'a', 'b', 'c', 'X', 'd', 'e', 'f', 'g', 'h']
>>>
```

Note that the first element in the list is numbered 0. Use `insert` to write a program that reverses a list, that is it should construct a new list with the elements of the original list in reverse order. Test your program.

**7.** The function `range` generate a list of integers; for example

```
>>> range(10)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> range(1, 11)
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> range(-2, 6)
[-2, -1, 0, 1, 2, 3, 4, 5]
```

Write a program to sum the integers from 32 to 84 (inclusive).

`range` can also be useful if you just want to do something a number of times. So, for example, you can print "I must improve!" ten times with:

```
>>> for n in range(10):
...     print 'I must improve!'
...
```

(Try it; the Python, not necessarily the improving!) Notice that the loop variable `n` is not used for anything here.

**8.** Write a loop to generate a list whose elements are the cumulative sums of the first 20 (non-negative) integers. That is, the list you generate should be: `[ 0, 1, 3, 6, 10, ...]`.

**9.** Suppose we want to generate a list of length 8, each element of which is the previous element times 3 plus 5, and the first element is 11. One way is like this:

```
>>> prev = 11
>>> L = [prev]
>>> for n in range(7):
...     x = 3*prev + 5
...     L.append(x)
...     prev = x
...
>>> L
[11, 38, 119, 362, 1091, 3278, 9839, 29522]
```

Note that `n` is not used in the loop; the list generated by `range` is only used to traverse the loop 7 times to generate a list of 8 items.

Use code like this to generate a list of 20 elements, each element of which is the square root of 1 plus the previous element of the list, and whose first element is 1. Print the list. Each element in this list is a closer approximation to the Golden Ratio $(1 + \sqrt{5})/2$, which, among other things, describes the most aesthetically pleasing ratio of the sides of a rectangle; see `http://en.wikipedia.org/wiki/Golden_ratio` for example. Having generated your list of approximations to the Golden ratio, use another `for` loop to construct a new list each element of which is the difference between the true golden ratio and your approximations.

**10.** The aim of this exercise is to generate the first few elements of the Fibonacci series. The Fibonacci series is the series

$$0,\ 1,\ 1,\ 2,\ 3,\ 5,\ 8,\ 13,\ 21,\ 34,\ 55,\ 89, \ldots$$

where each number in the series is given by the sum of the previous two, with the first two being defined as 0 and 1. Write a program to generate a list whose elements are the Fibonacci series. *Hint:* Use the same sort of idea as the previous question, appending newly-calculated elements to the Fibonacci list, but be careful how you initialise the Fibonacci list.

**11.** Without running it, explain what this code prints.

```
L = [1]
for x in L:
    print L
    L.append(x+1)
```

Having decided, run it and check. Make sure you understand the result.

**12.** Lists can contain anything, including other lists. So, a natural way of representing a limerick might be

```
limerick = [
    ['There','was', 'a', 'young', 'lady', 'named', 'Wright'],
    ['Whose', 'speed', 'was', 'much', 'faster', 'than', 'light'],
    ['She', 'left', 'home', 'one', 'day'],
    ['In', 'a', 'relative', 'way'],
    ['And', 'returned', 'on', 'the', 'previous', 'night']
]
```

Here `limerick` is a list with 5 items, each of which is a list with the words for that line of the limerick. To save typing, you can download `limerick.py` from the resources page. Write a program to print the limerick line by line and to count the number of number of words and letters in the limerick. (Remember that `print 'Hello',` prints "Hello" without a newline.)