Programming for Science
Workshop 3

This workshop aims to give you some more experience of using lists and `for` loops, together with conditionals.

1. We start by writing versions of the `birthday` program from the last workshop, but this time using indexing. 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 by indexing each element in turn, printing the elements, finding the sum of the elements, finding the product of the elements and printing the result at the end. Remember that you can generate the list of integer indices for a list with `range(len(list))`.

2. Creating a list with the first line of the poem:

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

copy your program from the last workshop and adapt it to use indexing to count the number of letters and words in `rhyme`.

3. Copy your program from the last workshop to find the average length of words in a file and adapt it to use indexing. Also, rather than subtracting 1 from the length of each word to account for the newline, use `strip` to remove white space at the ends:

```
>>> s = " The quick spotted Python   \n"
>>> s
' The quick spotted Python   \n'
>>> print s
 The quick spotted Python

>>> u = s.strip()
>>> print u
The quick spotted Python
>>> u
'The quick spotted Python'
```

4. Given two lists of numbers:

```
a = [ 0.04904772,  1.38633599,  0.94084723, -0.96782833, -0.66058869]
b = [ 1.07227778, -0.75143678,  0.54550933,  0.20866252, -1.4619395 ]
```

write a program `dot.py` to find the S sum of the product of their elements, that it

$$S = \sum_i a_i b_i$$

This is just the *dot* or *inner product* between the lists `a` and `b` regarded as vectors.

Make sure that your program works on lists of any length, not just the lists with 5 elements here.

If you are doing this workshop after functions have been discussed, write a function `dot(u, v)` which returns the sum of the products of the elements of `u` and `v`. Make your function print an error message if it is given arguments which have different lengths.

5. Write a function `zipper(a, b)` that zips two lists `a` and `b` of equal length together to form a single list, each element of which is a list containing the corresponding elements of `a` and `b`. For example

```
zipper([3, 4, 5, 6], ['a', 'b', 'c', 'd'])
```

should return the list of lists:

```
[[3, 'a'], [4, 'b'], [5, 'c'], [6, 'd']]
```

Provide code to demonstrate that your function works correctly and does something useful if it is called with arguments of different lengths. Your code should be in a file named `zipper.py`.

Note that this function is similar to the standard Python function `zip`, but you should write your own function.

6. The object of this exercise is to write a program `palindrome.py` that finds all the palindromes in a file of words. Recall that a palindrome is a word reads the same in either direction: for example, *anna*, *level* or *madam*. The file of words is the dictionary file `words.txt` on the module resources website `http://empslocal.ex.ac.uk/studyres/ECM1408`. This is the file used in the workshops and it contains 235886 words, one word per line.

To check whether a string is a palindrome you will need to test whether the string is the same as itself reversed. An easy way to do this is to use a *slice expression* which we will learn more about later. Briefly if `L` is a list or string, then `L[start:stop:stride]` means the elements of from index `start` up to (but not including) index `stop` in steps of `stride`. If `stride` is omitted, it is taken as 1; if `start` or `stop` are omitted they are taken as the ends of the list or string. Thus:

```
>>> L = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i']
>>> L[3:5]                      # Elements 3 and 4
['d', 'e']
>>> L[4:]                       # Elements 4 to the end
['e', 'f', 'g', 'h', 'i']
>>> L[:4]                       # Elements up to 4
['a', 'b', 'c', 'd']
>>> L[::2]                      # Elements 0 to the end in steps of 2
['a', 'c', 'e', 'g', 'i']
>>> L[1:8:3]                    # Elements of 1 to 8 in steps of 3
['b', 'e', 'h']
```

The useful slice expression for this exercise is:

```
>>> L[::-1]
['i', 'h', 'g', 'f', 'e', 'd', 'c', 'b', 'a']
>>> s = "Hello there"           # For a string instead of a list
>>> s[::-1]
'ereht olleH'
>>> u = 'satanoscillatemymetallicsonatas'
print u, 'backwards is', backwards
satanoscillatemymetallicsonatas backwards is
satanoscillatemymetallicsonatas
>>>
```

For more on "satanoscillatemymetallicsonatas" see the rock band *Soundgarden.*

Write a program to make a list of all the palindromes in `words.txt` that are at least three characters long.

7. Write a version of the program to construct a list of the first 20 Fibonacci numbers, but use indices this time. Writing the program is perhaps a little easier with indices because we can express the nth Fibonacci number F_n as the sum of the two previous numbers:

$$F_n = F_{n-1} + F_{n-2} \quad \text{with } F_0 = 0 \quad \text{and } F_1 = 1$$

8. The file `trader.txt` on the resources website contains a trader's daily returns in pounds, one per line. Positive numbers represent a profit on the day and negative numbers a loss. The trader gets a bonus when his accumulated profit exceeds £200,000 (after which the counting starts again). The bonus is that day's trading.

Use the same form that we have used previously to read the lines from `trader.txt` into a list:

```
returns = open('trader.txt', 'r').readlines()
```

Note that `returns` is a list of strings. You can convert a string to a float with `number = float(string)`.

Write a program to find the ratio of the number of days on which he makes a profit to the number of days that he makes a loss. Your program should also print the days on which the trader is awarded a bonus. It costs £100k per year to employ him (apart from the bonuses). Write a program to calculate his employer's average annual profit for 20000 days listed in `trader.txt`. Assume that there are 365 days in the year and he gets 28 days a year holiday so he trades on $5 \times (365 - 28)/7$ days per year.