Exercises L1

Exercise 1 (variables)

a) Store the digits of your age in two variables, and their sum in a new variable
b) Define a list containing the numbers 3, 6, 9, …, 81
c) Define a dictionary that associates the numbers 0 to 9 with their English spelling
   E.g., 0 -> 'zero'

Exercise 2 (for loops)

Write a script that computes and prints Pi according to Wallis' formula:

$$\pi = 2\prod_{i=1}^{\infty}\frac{4i^2}{4i^2-1}$$

Exercise 3 (functions)

Write a function that takes a string containing a full name ("Firstname Lastname") and returns the initials ("F.L."). (Look at the string method `my_string.split()`).

Exercise 4 (numpy)

a) Define a 5x5 numpy array with the numbers 0 to 24 (use numpy.arange and numpy.reshape)
b) Using numpy, draw a set of 100x5 random numbers from a normal distributions (numpy.random.randn), and compute the mean and variance for the 5 columns (my_array.mean, my_array.std).
c) Compute the cosine function between 0 and 4*Pi, and plot it.

Exercise 5 (random walks)

The goal of the exercise is to write a function `random_walk(nsteps)` that simulates a random walk for `nsteps` time steps. The behavior of the function is

1. start from x=0
2. at every time step, x is randomly increased or decreased by 1
   (have a look at the functions in the module numpy.random)
3. the function returns an array of length nsteps that contains the value of x at every point in time

a) Start writing tests for the function in a file `test_random_walk.py`:
   - Test that the length of the returned array is correct
   - Test that the difference between elements is always 1
b) Write the function in a file `random_walk.py`, and debug until the tests pass
c) Store 1000 random walks of length 50 in an array
d) Plot the 95% interval for the position of x as a function of time
   (i.e., plot +/- twice the standard deviation of the random walks)

Exercise 6 (deceivingly simple function)

Download the file `maxima.py` from
http://people.brandeis.edu/~berkes/data/exercises/ .

The file contains a function, `find_maxima`, that finds local maxima in a list.

a) Execute the function with these input arguments and others of your own invention until you are satisfied that it does the right thing for typical cases:
```
x = [0, 1, 2, 1, 2, 1, 0]
x = [i**2 for i in range(-3, 4)]
x = [numpy.sin(2*alpha)
      for alpha in numpy.linspace(0, 2*3.14, 100)]
```

b) Now try with the following inputs:
```
x = [4, 2, 1, 3, 1, 2]
x = [4, 2, 1, 3, 1, 5]
x = [4, 2, 1, 3, 1]
```

For each bug you find, solve it using the agile programming cycle:

i. Find the bug
ii. Write a new test case that reproduces the bug. Try to make the test case as simple as possible; here, this means using the simplest input data that still triggers the bug
iii. Correct the code
iv. Make sure that all the tests pass

c) So you think that the code is now clean and robust… Look at the output of the function for the input list
```
x = [1, 2, 2, 1]
```
Does the output correspond to your intuition? Think about a reasonable behavior in this situation, and meditate about how such a simple function can hide so many complications

d) *(optional)* Implement the “reasonable behavior” you conceived in e) and document it in the docstring, adding a new doctest.
Make sure that your function handles these inputs correctly (include them in the tests):
```
x = [1, 2, 2, 3, 1]
x = [1, 3, 2, 2, 1]
x = [3, 2, 2, 3]
```