Practice Exercises for Day 2

These exercises are meant for you to practice your Python coding and the concepts you have been shown today. Note that the level of difficulty differs quite a bit, and some of the exercises may be challenging for beginners. Please do not hestitate to collaborate on them, or ask us for advice or help if you get stuck. There are probably too many exercises to cover in the time available, and they have no natural order (except that 4 partly refers to 2), so feel free to do the exercises in the order you prefer.

Exercise 1: Solve a system of ODEs

The following system of ODEs is the classical Lotke-Volterra equations (or predator-prey equations), describing the interaction between two species x and y:

$$x'(t) = rx - axy$$

$$y'(t) = -my + bxy$$

Use the solvers `scipy.integrate.odeint` or `scipy.integrate.solve_ivp` to solve this system with parameters $r = m = 1, a = 0.3, b = 0.2$ and initial conditions $x(0) = 1, y(0) = 1$, for the time interval 0 to 20. Plot the two solutions in the same window.

Exercise 2: Read and plot climate data

Historical climate data is available from various meteorological services. We have downloaded data from the Oxford station of the UK metoffice. The data is in a text file with the following format:

```
Oxford
Location: 4509E 2072N, 63 metres amsl
Estimated data is marked with a * after the value.
Missing data (more than 2 days missing in month) is marked by  ---.
Sunshine data taken from an automatic ...
   yyyy  mm   tmax    tmin      af    rain     sun
              degC    degC    days      mm   hours
   1853   1    8.4     2.7       4    62.8     ---
   1853   2    3.2    -1.8      19    29.3     ---
   1853   3    7.7    -0.6      20    25.9     ---
   1853   4   12.6     4.5       0    60.1     ---
   1853   5   16.8     6.1       0    59.5     ---

...
```

Read the file, compute the mean of the monthly max temperatures for each year, and store the result in a dictionary. Plot the resulting mean temperature a as a function of the year.

Algorithm:

1. Skip the first 7 (for us uninteresting) lines
2. Read the column data and use data in column 1 as dictionary keys and column 3 to compute the corresponding values. You can also read these two data columns into two lists, and then traverse the lists to do the data processing.
3. Plot the data

In [2]:
```
"""
Data file from:
http://www.metoffice.gov.uk/pub/data/weather/uk/climate/stationdata
"""

filename = '../data/oxford.txt'
infile = open(filename,'r')

#skip a few lines:
for i in range(7):
    infile.readline()


"""
Replace the line below with a for-loop to read the rest of the file
line by line,
and store the variables of interest in lists
"""
print(infile.readline())
```

```
   1853   1    8.4     2.7       4    62.8     ---
```

Exercise 3: Computing Name Scores

This exercise is taken from Project Euler, where it is Problem 22 (https://projecteuler.net/problem=22).

a)

The file `names.txt` contains over five-thousand first names separated by a comma. Find a way to read this file and get these names into a list. Before moving on, verify that the names seem to have been read in correctly.

b)

Sort your list of names, so that it is in an alphabetical order. Verify that the list seems to be sorted.

Hint: There is a built-in function called `sorted` that returns a sorted copy of a given sequence. Python lists also have the very handy `.sort()` method you can use, which instead sorts the list *in-place*.

c)

We now want to associate an alphabetical value with each name. For a given name, each letter in that name is given a score equal to its position in the alphabet. The value for the whole name is the sum of each letter's value.

Take for instance the name `COLIN`, which would be

$$3 + 15 + 12 + 9 + 14 = 53.$$

Define a function `alphabetical_value(name)` that takes a name in as a string, and returns the value of that name. Verify that it works by checking that `alphabetical_value('COLIN')` returns 53.

Hint: To get a letters position in the alphabet, you can first define the alphabet as a string, and then use `.find()` to find the position of a letter in the alphabet. Be aware that Python starts counting at 0 however, so `alphabet.find('A')` would give us 0, instead of 1.

d)

We now want to compute a score for each name in the list of names, by first computing it's alphabetical value, as above, and then multiplying this value with the names position in the list. We found that `COLIN` had a value of 53, so if this name was for example 938th in the list, this name would obtain a score of

$$938 \times 53 = 49714.$$

Use a `for`-loop to loop over the entire list, computing the score of each name in the list and add them all together. What is the total score of all the names in the complete file?

Exercise 4: Explore the pandas dataframe format

In this exercise we want to glimpse into the popular pandas dataframe format, which has become the standard tool for data processing applications in Python. A nice feature of pandas is that it is well integrated with notebooks, and data is automatically rendered into nicely formatted and readable tables. We will use the oxford climate data file as an example. Pandas is part of the SciPy package, and is included in the standard anaconda Python installation.

a) Read the oxford weather data using the pandas `read_csv` function

The `read_csv` is the standard for reading files in csv or similar formats, including Excel files and whitespace deliminated files. Use the function to read the oxford weather data into a pandas dataframe object.

Hint: Since our file is whitespace deliminated, pass the argument `delim_whitespace = True` to the `read_csv` function. The function also assumes that all rows have an equal number of data items, and therefore gets confused by the first lines. Use the argument `skiprows` to avoid this problem.

b) Fix the column labels

The `read_csv` function will automatically use the first row as the column headers. This is obviously not what we want. We can fix this by passing the argument `header=None` to the function when reading the file, and then manually setting the column headers for the resulting dataframe using the object's `columns` attribute.

c) Check the datatypes for each column, convert to numerical values if needed

You can check the datatypes for each column with the dataframe object's `dtype` attribute. Numerical values are usually automatically stored as int or float, while other values have a generic `object` datatype. We want the year to be an integer and the max temperatures to be a floating point number (`float64`). If needed, data can be converted using the `astype` function, for instance: `df['high'] = df['high'].astype('float64')`.

d) Plot the max temperature as a function of year

Plot the column showing the maximum temperatures as a function of year.

Hint: You can either extract the two columns and plot using the usual `plot` function from `matplotlib.pyplot`, or use the built-in plot function of the datafield object, which takes the column labels as arguments. Since we plot the data as-is, without computing the mean value for each year, the plot does not make as much sense as the plot you got in Exercise 2.

e) Save your dataframe to a hdf file format.

Save the dataframe to the hdf file format using the `to_hdf` function of the dataframe object. The function takes a filename and a key as arguments (both strings). Read the file back into a new dataframe using the `read_hdf` function, and verify that the result looks correct.

In [5]:
```
import pandas as pd
filename = 'data/oxford.txt'

#frame = pd.read_csv(...)
```