TMTplus Introduction to Scientific Programming

Mahdi Farnaghi, Mahdi Khodadadzadeh & Robert Ohuru

March 2021

Note from the teaching staff

The materials that we use for this course are somewhat new and have been heavily reworked. We will likely have made mistakes in this work or have glossed over issues that deserved better or different treatment. Where you find issues worth noting please do report these to us as it will allow to repair and improve.

By the way, welcome to the Exercise book for Introduction to Scientific Programming! We believe this book is self-explanatory, so go ahead and practice.

Chapter 7

Files

In many forms of scientific work, substantial amounts of data need to be considered and handled. Professional routines demand that that data is acquired, prepared and scrutinized before actual analysis takes place. All such processes lead to versions of the data, which, again in the light of professional robustness of the routines, lead to the data being kept in different versions. Such allows reruns of partial routines once deeper causes of errors in the data, for instance, have been identified. Data is commonly kept in files, and one way of doing version management is ensuring that the file versions are kept separate by using sensible filename conventions. Some Phase 4 of a data workflow would take Stage 3 data to Stage 4 data. If we ensure that the workflow can only read Stage 3 data, we are protecting accidental corruption of our data files. Version numbers of the files serve a similar purpose but are handled orthogonally: some workflow phase can only take a version N data file to another data file of the same version (but later stage!).

The exercises

The main goal of this practical session is to use the Python programming language to learn how to work with files. In these early exercises, the focus is strongly on the basic file operations such as open and close, read and write. The student is invited to give thought to the note above on file management practices.

7.1 Reading text files

Ex 7.1

The file points.txt contains id, longitude and latitude for 100 points. Write a function get_points() to read the file data and store the points as a list of lists, with each inner list containing three items: id, longitude and latitude. (hint: you could use string.split()). Ensure that the function can be used for different files.

The list should look like: [['0', '-121.904167', '41.974556'], ...]

Also write a few lines of code to test the function. Finally, pose yourself the question under which conditions your function will not work as it should. We are not asking you here to address those conditions in the code ... just think up when it will not work.

7.2 Reading GPS files

Ex 7.2

The file gps.csv contains NMEA sentences recorded with a GPS device. All of the standard sentences begin with a 5-letter command: the first two letters define the device that uses that sentence type (for GPS receivers the prefix is GP), the last three letters define the type of the command.

Write a function read_csv_file() that reads the GPS.csv file and returns a list of lists, with each inner list containing the sentence items.

Then write a function list_gps_commands() that uses the output of read_csv_file() to build a dictionary where the key is the GPS command and the value is the number of times the GPS command is observed.

Use the test below to test your functions:

```
gps_list = read_csv_file('GPS.csv')
gps_dict = list_gps_commands(gps_list)
print (gps_dict)
```

The response should be something of the form:
{'$GPGGA': 282, '$GPGSA': 282, '$GPGSV': 846, '$GPRMC': 283}

If you want to know more about NMEA sentences you can look at the NMEA website.

7.3 Counting words

Ex 7.3 Create a function `word_count` that reads a large file called `snark.txt`, splits it into words, and returns a word:count table in the form of a Python dictionary. Before rushing to the keyboard think about the following concerns:

- Read the section "Reading and Writing Files" in the Python Tutorial. Would you read the file in its entirety and then process the resulting string? Or would you read the file line by line to process it?
- Convert all words to lowercase characters.
- How do you split a string on whitespace (spaces, tabs and newlines) into words?
- The words may still contain leading or trailing punctuation marks. How do you strip those off? Hint: Python's standard module called `string` contains two variables called `whitespace` and `punctuation` that contain these characters. You may use those variables for stripping the characters.
- There may still be punctuation characters within words. You are allowed to ignore those, because it is impossible to tell whether they are part of the word, like in"object-oriented," or part of the punctuation in a sentence, like in "tomorrow-whether you like it or not."
- Which method of a dictionary can be used to check whether a particular key already exists in the dictionary? See, for instance, the Python Standard Library.

Use this test to try out your function

```
wordtable = word_count('snark.txt')
print(wordtable['and'])
```

Expect an answer like 228.

Ex 7.4 Write a function that uses two parameters: a dictionary (as the one produced in the previous exercise), and a file name. The function should create a file with the given name, and store each record of the dictionary in one line of the file.

Ex 7.5 Write a function that determines, based on the word table, which word occurs most often in the poem. Then write a function that determines which 5-letter word occurs most often in the poem. This involves iteration over the dictionary, testing the number of characters in a string, and determining the maximum count. Do the results surprise you?

Can you generalize the function, such that you can specify the length of the word using a parameter?

Ex 7.6 Write a function to determine how many words in the file start with a letter 'b'.

Can you generalize the function, such that you can specify the starting letter of the word using a parameter?