Handin 3 Ⓢ

| **Due** 27 Sep by 18:00 | **Points** 100 | **Submitting** an external tool |

Available after 20 Sep at 18:00

The handins in the course will consist of two parts: 1) a set of short exercises to practice a particular part of the curriculum (e.g. loops), and 2) a project part. In the project part we will consider a specific Data Science case, each week working on the same (or similar) data, and gradually building up a complete analysis of the data.

Note that the tools required to solve the handins will be generally covered in the lectures in the week where the assignment is given. Sometimes, you might have to wait until the Friday lecture before you have all the tools to complete the assignment.

Part 1

In this week's exercise, we'll practice loops and functions a bit more. We'll also try to write our own Python module, and use regular expressions from the re module.

Start by creating two files: `handin3.py` and `handin3_test.py`. The idea is that `handin3.py` will be the module file, containing the main code, and `handin3_test.py` will contain code that uses (tests) our module. You will also need to download the following file to the directory into your project directory: **https://wouterboomsma.github.io/ppds2023/data/british-english** ➯ **(https://wouterboomsma.github.io/ppds2023/data/british-english)** .

1. Write a function called `read_word_file` that takes a single argument called `filename`. The function should open the file, iterate over the lines, and remove the new lines at the end of each line. For each line, it should construct a tuple consisting of two values: the line number (starting at zero), and the line string (without newlines). For example, for a file containing `"hello\n"` on the first line, and `"world\n"` on the second line, you should get the following list: `[(0, "hello"), (1, "world")]`. The function should return this list to the caller of the function.

   Inside the `handin3_test.py` file import `handin3.py`, and then call the `read_word_file` function using the `british-english` file we downloaded above. Save the result in a variable called `word_list`.

2. Copy&Paste your `read_word_file` into a new function called `read_word_file2`. This time, the function takes two arguments: 1) `filename` and 2) `filter_re_str`. As before, the function should open the file corresponding to filename, iterate over the lines, remove the new

lines at the end of each line, and save tuples of (line-number, line-without-newline) to a list. The difference is that this time, we have an extra argument called `filter_re_str`, which is a string value. Your `read_word_file2` function should compile this string into a regular expression, and then use this regular expression to search in each line whether it can find a match of the regular expression in the line. For instance, if I write `read_word_file2(filename='british-english', filter_re_str="^A")`, it return a list of tuples only for the lines that start with an `A`. The `filter_re_str` argument should be optional - it should default to the empty string, in which case it will return a list of tuples for all lines in the file.

Inside the `handin3_test.py` file, call the `read_word_file2` function on the `british-english` file. Specify a `filter_re_str` such that only words of length 6 that start with `"py"` (in lowercase) are returned. The words should contain only lower and upper case letters from the English alphabet (i.e. no apostrophes and other special characters). Save the result in a variable called `filtered_word_list`.

Part 2: Project

Last week, we wrote a function called `read_data`, that read the temperature anomaly data into a list of strings. This week, we will process our data a bit more, such that our numerical data is more readily accessible.

1. Create a file called `handin3_project.py`. Inside this file, create a function called `read_data2`. This function should be similar to the `read_data` function from last week (feel free to copy&paste), but the function should now take two arguments: 1) a string argument called `filename` as before and 2) a tuple of two integers called `year_range`, which specifies the (start,end) years for which we wish to read in data from the file. The function should return a list of strings corresponding to the lines within the range. In order to solve this, you will need to extract the year value from each line (by splitting and converting to int as we did in handin1), but you should keep a copy of your original line string, so you can add it to your list.

   As is common for Python ranges, the start value of the range should be included, while the end value should be excluded. For example, if I write `year_range=(2000, 2010)` I should get the values from 2000 to 2009. Finally, note that if year_range is not specified by the caller of the function, the function should return data for all available years.

   Create a new file called `handin3_project_test.py` which calls the `read_data2` function on the `Land_and_Ocean_summary.txt` file, with and without specifying a range of years.

2. Duplicate (copy&paste) your `read_data` function into a new function called `read_data3`. Just as before, the function should take two arguments: 1) `filename` and 2) `year_range`, where

`year_range` has a default value, making this argument optional. This time, rather than returning a list of strings, the function should return a dictionary. The keys in the dictionary should be the year values (as integers), and each dictionary value should be a list of the numbers (i.e. floating point) in that line of the file. So, for example, if the dictionary was called data, I should be able to able to write: `data[1990][0]` to get the value `0.366`. Hint: the file contains some `NaN` values (i.e. "not a number"), you can still just convert these to a float - they will turn into `nan` float values.

Inside the `handin3_project_test.py`, call the `read_data3` function with the arguments `filename='Land_and_Ocean_summary.txt'` and `year_range=(2015,2018)`. Save the result in a variable called `temp_anomaly_data`.

When you are done, click on the "Load Handin3 in a new window" button below, which will take you to the CodeGrade server. Here, please submit the `handin3.py`, `handin3_test.py`, `handin3_project.py`, and `handin3_project_test.py` files. CodeGrade will then automatically check the code for you, and upgrade your grade for the assignment within Absalon. You can submit as many times as you want. **Note that from this week on, the autotests will also test your code style, and for instance give errors for missing doc-strings.**

This tool needs to be loaded in a new browser window

Load Handin 3 in a new window