Assignment 7: Working with Excel and tests

The purpose of this assignment is to generate some statistics about Copenhagen's citizens. On the way you will repeat/learn:

- How to read Excel files.
- Create functions in Python.
- Use `for` loops to iterate over elements of a list.
- Lookup elements in dictionaries.
- Use `set` s in Python together with some of their functionality.

We will use two datasets. Both are found in this folder with the homework assignment.
One is `befkbhalderstatkode_small.xlsx` , which contains statistics about Copenhagen's population corresponding to the year 2015. This dataset was created from the data available at http://data.kk.dk/dataset/befolkningen-efter-ar-bydel-alder-og-statsborgerskab (http://data.kk.dk/dataset/befolkningen-efter-ar-bydel-alder-og-statsborgerskab), where you can also find a more in depth description of the given data.

The second dataset, `country_codes.xlsx` , comes originally from Danmarks Statistik and maps country codes to countries of origin, see http://www.dst.dk/da/Statistik/dokumentation/Times/forebyggelsesregistret/statkode.aspx (http://www.dst.dk/da/Statistik/dokumentation/Times/forebyggelsesregistret/statkode.aspx).

Exercise A: Reading Excel Files

- Inspect the code in the file `read_excel_files.py` .
- It contains two functions `country_codes_in_kbh_data` and `country_codes_in_stats_data` , which reads one Excel file each (`befkbhalderstatkode_small.xlsx` and `country_codes.xlsx` respectively.) The two functions are almost the same. Compare them line by line. Can you spot the difference?
- Open both `.xlsx` files with one of the programs *Excel*, *OpenOffice*, *GoogleSpreadheets*, etc. to inspect their contents.
- Set a breakpoint on line `5` in `read_excel_files.py` and use the debugger to step through the execution of this program.
- Try to understand what is going on by observing the values of variables.

Hand-in:

- A brief (4-5 lines) description of what the program `read_excel_files.py` is doing.

It is quite unpractical and redundant to use both the two functions `country_codes_in_kbh_data` and `country_codes_in_stats_data` when they do almost the same thing. It would be much better to have one function that would read both files!

```
- Write down that function in around 4-5 lines of pseudocode
```

Hand-in your description and pseudo-code in a file `A.txt` .

Exercise B: Creating a New Function

Implement the function `country_codes_data` in `country_codes_stats.py`, to unify the two functions `country_codes_in_kbh_data` and `country_codes_in_stats_data` into one. The function should capture variability via two arguments.

That is, the function definition will look something like the following, see file `country_codes_stats.py`:

```
def country_codes_data(filename, column_number):
    """Read country code data from a given Excel file,
    in which entire columns contain country codes and
    the first row is a header row without values.

    :param filename: str
        Path to the Excel (.xslx) file containing a row
        with country codes
    :param column_number: int
        The index number of the column containing the
        country codes.
    :return: list
        A list of country code numbers.
    """

    # TODO: Implement me!
    pass
```

Hand-in `country_codes_stats.py` completed with your implementation.

Exercise C: Filtering for Unique Elements and Sorting

Observe that the list of `country_codes`, which is returned from `befkbhalderstatkode_small.xlsx` contains many duplicates.

1. Modify the function `country_codes_data(filename, column_number)` from the earlier `country_codes_stats.py` so that it returns a list of **unique** country codes. That is, it does not contain duplicates!
    - *Hint:* remember a basic Python data structure, which can hold collections of elements, where each element is unique.
2. Modify the function `country_codes_data(filename, column_number)` even further, so that it returns a sorted list of unique country codes.
    - *Hint:* use the function `sorted` to return a sorted list.
    - *Hint:* In case you are in doubt, use Mu to read the documentation of the `sorted` function by typing `sorted(` and read the text that appears or `print(help(sorted))`.

Hand-in your solution as `C.py`.

Exercise D: A Test Case for Checking your Solution

To check that your implementation (or parts of it) works correctly, run the test case `test_country_codes_stats.py` against your `country_codes_stats.py` module. You can do so by running it with `pytest` from the command-line.

```
$ pytest test_country_codes_stats.py
```

- You should first of all make sure that all the tests pass. If they do not there is something wrong with your code!
- Second, try to come up with a test case and add it to the bottom of the file. Do not forget to run it to see if it passes!

Hand-in your extended implementation of `test_country_codes_stats.py`.

Exercise E (OPTIONAL): Dictionary Lookup

Implement the function `translate_code_to_text` in your `country_codes_stats.py` module. The goal is to allow you to translate a numerical country code into an explanatory string. For example, the following code should print `'Belgien'`.

```
descriptive_string = translate_code_to_text(5126)
print(descriptive_string)
```

- *Hint:* Obviously, your function gets a single argument for the country code.
- *Hint:* Make use of the module `stats_code.py`, which contains a variable with a dictionary `COUNTRY_CODES`. That is, after importing the module, you can lookup the description of a country code via something like the following:

```
belgien_code = stat_codes.COUNTRY_CODES['5126']
print(belgien_code)
```

which should print `'Belgien'`

Hand-in a file called `E.py` in which the function `translate_code_to_text` is implemented.