Introduction to Data Science and Programming

Assignment 2

Due September 30, 2021, 23:59

Description

In this week's assignment, we are going to read and write data from/to data files. We have provided a few files for this assignment. Inspect the contents of these files manually to get a feel of what the data looks like.

In this assignment, we will read and parse comma-separated-value (CSV) files. CSV files are text files that contain data where each entry is on its own line, and values are separated by commas. You can think of these as tables like in Excel spreadsheets. Be aware that in practice, however, the values are often separated by other delimiters than a comma, such as space, tab (i.e., `.tsv` = tab-separated-value file) or any other special character like |. Here are some examples:

comma separated	a,b,c,d,e
tab separated	a b c d e
custom separated	a‖b‖c‖d‖e

Submission

- Submit the Python file `data.py` containing your solutions. We provided a `data.py` skeleton file where you can fill in your code.
- Your principal function calls must reside inside the `main` function.
- Make sure that we are able to run the Python with the following command: `python data.py`.
- Also, make sure that we can import your data library without automatically running the `main` function.
- Use only the concepts you learned during the lectures. Do not use imports as they are not needed.

Assignment

1. Start by writing a function called `read_file(filename)` that takes a filename as input (a parameter), and that returns a list of strings–one string for each line of the file (no newline characters). You'll need this to test the functions below.
2. Write a function, called `parse_csv_lines(lines)`, that takes a list of strings (the output of `read_file`) and returns a list of lists containing the comma-separated values of each line.
3. Write a function called `parse_delimited_lines(lines, delimiter)`, that returns the same as `parse_csv_lines`, but instead of separating values by commas, it accepts any delimiter.
4. The file `municipalities-2005-2019.csv` contains the annual mean population age of each municipality of Denmark since 2005. Load the file with the above function and write a function called `age_difference(lines)` that takes a list of lists and computes the difference between the first year (2005) and the last year (2019). The function should return a list of numbers that show the change (from 2005 to 2019) in each municipality.
5. Inspect the two single-column CSV files, `female_names.csv` and `male_names.csv`. These files contain a subset of approved Danish male and female names. However, some names are valid for both sexes. Write a function called `find_unisex_names(male_names, female_names)` that takes two lists of strings (the output of `read_file`), and returns a set of unisex names.
   Hint: There is no need for recursion. Use sets.

6. With the generated set of unisex names, we can now create our own dataset. Write a function called `build_name_dataset(female_names, male_names, unisex_names)` that takes two lists of strings (the output of `read_file`), a set of unisex names (the output of `find_unisex_names`) and writes each name and sex to a single file, `all_names.csv`. The file should have two columns, separated by a comma (without a space), with the first column containing the name, and the second column containing the sex: `M` for male, `F` for female, or `U` if both. A line in the file could be: `Adam,M`. Note that there might not be the same amount of names for each sex so remember to account for this. Also, don't worry about `all_names.csv` containing unisex names twice, but feel free to try and solve this. If done correctly, the file `all_names.csv` will contain 948 unique names.

7. Now that we have a combined and updated CSV file (`all_names.csv`) with valid Danish names we would like to distribute the names in separate files based on the first character of the name. Concretely we want the list `["Adam", "M"]` to be written to `A.csv` as `Adam,M` and `["River", "U"]` to be written to `R.csv` as `River,U`, and so forth, one name per line. Write a function called `write_sorted_names(names)` that takes a list of strings as an argument, and writes each name to a file based on the first letter of the name. The function should return `None`, and print `"Done"` (without quotation marks) when the function has completed. If done correctly, this will create 32 csv files (note that some of the files might be named unusual characters like `Ã`. This is intended).