Homework 7 Instructions

UCSD Extension CSE-41273, Spring 2019.

The zip download file has, besides this file, several data files that you can use for testing/validating your programs as described below.

You will be writing 5 separate Python module/files for this homework assignment. **Please put your name in a comment at the top of *each* program file.** The 5 modules are: `file_stats.py`, `re_order.py`, `tabs_to_commas.py`, `re_sort.py` and `country_convert.py`. Zip these 5 `.py` files together into a file as described in the *About Homework* document. **Do not include data/results files; only the `.py` files.**

Please read and follow all the directions carefully! Some students have said that this is the hardest homework assignment. But if you think about what you are doing, and what should be happening, it should become clear how to proceed. Please email me with any questions you have! **Everything you need to know to do the homework is in the lessons, handouts, and required readings.**

Pay careful attention to the requirements for each program. Most have requirements as to how it works on the command line *in addition to how it works in the REPL.* Some require an input filename alone, some an input *and* output file. I should be able to run your code both from the command line and from the REPL (when required) *as shown in the examples for the programs*. You don't need to use `argparse` for this assignment, but you can if you want. Don't forget to run `flake8` on all of the program files before submitting.

If you have a Windows computer: Be sure to use `newline=''` in the file open statements. **Please** assume that the input to your files/functions is always valid.

For those of you who are spending too much time Googling for answers, **do not use Pandas (or any other third party package).** It is not necessary for any of the homework programs, and **you will not receive full credit if you use Pandas.** You have been given all the information you need to do the homework without needing other Python packages.

For the programs that have command line interface *and* a function to be called from the REPL: When the program is run from the command line, it should call the function. In other words, do not duplicate code for use by the REPL and the command line.

Program 1: file_stats. (15 pts)

Write a program `file_stats.py` that, when run on the command line, accepts a text file name as an argument and outputs the number of characters, words, lines, and the length (in characters) of the longest line in the file.

Please note that this should work on **any** *text file, csv file or any other file that is text-based, including a .py file. For the number of words, you want to separate words based on* **default** *"whitespace"; string methods are your friends here.*

It should work like this from the command line:

```
$ python file_stats.py beautiful_soup_by_Lewis_Carroll.txt
Characters: 553
Words: 81
Lines: 21
Longest line: 38
$ python file_stats.py books.csv
```



```
Characters: 557
Words: 45
Lines: 6
Longest line: 120
```

The program should have a function `stats` that takes the filename as an input and returns the statistics (in the same order) as a tuple, so it can also be used in the REPL like this:

```
>>> from file_stats import stats
>>> c, w, l, ll = stats('beautiful_soup_by_Lewis_Carroll.txt')
>>> print(f"characters: {c}, words: {w}, lines: {l}, longest: {ll}")
characters: 553, words: 81, lines: 21, longest: 38
```

A file `beautiful_soup_by_Lewis_Carroll.txt` is provided in the files.

Program 2: tabs_to_commas. (15 pts)

Write a program `tabs_to_commas.py` that when run on the command line, takes the name of a CSV file (or TSV file) that uses *tabs as delimiters* and converts it to one using *commas as delimiters* with the filename modified with `_commas` appended. So if the tab-delimited file you read is called `my_info.tsv`, the new file should be called `my_info_commas.csv`. Note: The file names are strings, so string methods are your friends.

This program should have a function `tab_to_comma` that has two arguments: `in_file`, the name of the csv/tsv file to read, and `out_file`, the name of the csv file to create, so it can be used from the REPL or another Python module.

Hint: It's *specifically* for csv-type files, so use the `csv` module.

Two files are included to test this that use the tab delimiter: `colors.csv` and `people.tsv`. Both files use the tab character as a delimiter.

I have included a file, `people_result.csv`, which contains the correct result from running the following command. I have changed the file name so you can compare it.

```
$ python tabs_to_commas.py people.tsv
```

Results in the new file `people_commas.csv`. The file `people_commas.csv` should be the same as `people_results.csv`.

```
>>> from tabs_to_commas import tab_to_comma
>>> tab_to_comma('people.tsv', 'people.csv')
```

Results in a file `people.csv`. Similarly, you can use the keyword arguments:

```
>>> from tabs_to_commas import tab_to_comma
>>> tab_to_comma(out_file='colors_commas.csv', in_file='colors.csv')
```

Will result in the creation of a new file `colors_commas.csv`. The new `colors_commas.csv` file should look like this in an editor:

```
purple,0.15
indigo,0.25
crimson red,0.30
sky blue,0.05
spring green,0.25
```

Program 3: re_order. (15 pts)

Write a program `re_order.py` that when run on the command line, takes 2 file names; an input csv file and output file name. It opens the input CSV file, reads it and writes out the data with the first and second columns switched.

This program should have a function `re_order` that has two arguments: `in_file`, the name of the csv file to read, and `out_file`, the name of the csv file to create, so it can be used from the REPL or another Python module.

Please note that this program/function must work for **any** *input csv file that has at least 2 columns, not just the one provided for your testing purposes. You need not worry about checking for invalid files. For example, it should also work on the colors_commas.csv file created with the tabs_to_commas.py program.*

You may assume that the input file will use the default delimiter (comma) and will always have at least 2 columns.

Hint: It's *specifically* for csv files, so use the `csv` module. Also, it might help to review how slicing works.

A file `books.csv` is provided to use as a test file.

```
$ python re_order.py books.csv books_author.csv
```

Results in the creation of a new file `books_author.csv` that contains the same data as in books.csv, with the first two columns (title and author) switched.

```
>>> from re_order import re_order
>>> re_order(in_file='books.csv', out_file='books_author.csv')
```

Will also result in the creation of a new file `books_author.csv`.

The first 4 lines of `books_author.csv` should be:

```
Author,Title,Publisher,Year,ISBN-10,ISBN-13
Al Sweigart,Automate the Boring Stuff with Python,No Starch Press,2015,1593275994,978-1593275990
Mark Pilgrim,Dive into Python 3,Apress,2009,1430224150,978-1430224150
"David Beazley, Brian K Jones","Python Cookbook, Third edition",O'Reilly
Media,2013,1449340377,978-1449340377
```

Program 4: re_sort. (15 pts)

Create a program `re_sort.py` that when run on the command line, takes a filename of a CSV file as input, then reads the CSV file, sorts the file by the values of the **second** column, and writes the data to a new CSV file with `_sort` appended to the file name. So if the input filename is `books.csv`, the output file should be `books_sort.csv`. Assume the CSV file has 1 single header row.

Think about what might be a useful coding thing to use for a CSV file when we know there is a header row in the file.

This program should have a function `re_sort` that has two arguments: `in_file`, the name of the csv file to read, and `out_file`, the name of the csv file to create, so it can be used from the REPL or another Python module.

Hint: It's *specifically* for csv files, so use the `csv` module. Look into using DictReader/DictWriter.

Please note that this program/function must work for **any** *input csv file that has a header row and at least 2 columns, not just the one we use for an example. You need not worry about checking for invalid files.*

```
$ python re_sort.py books.csv
```

Will result in a file named `books_sort.csv`.

```
>>> from re_sort import re_sort
>>> re_sort(in_file='books.csv', out_file='books_sorted.csv')
```

Will result in a file named `books_sorted.csv`.

The *last* 2 lines of `books_sort.csv` should be:

```
"Fluent Python: Clear, Concise, and Effective Programming",Luciano Ramalho,O'Reilly
Media,2015,1491946008,978-1491946008
Dive into Python 3,Mark Pilgrim,Apress,2009,1430224150,978-1430224150
```

Program 5: country_convert. (40 pts)

Write a program `country_convert.py` that opens the included `countryInfo.csv` file and extracts the country name, capital city and population from each row, then writes a new file named `country_simple_info.csv` with country, capital and population in each row, with the rows sorted by population size, largest first.

Note this program does not take input/output file names, as it is specific to the file given. It only needs to run on the command line like this:

```
$ python country_convert.py
```

Which will result in a file named `country_simple_info.csv` whose first 4 lines are:

```
country,capital,population
China,Beijing,1330044000
India,New Delhi,1173108018
United States,Washington,310232863
[...]
```

Hints: Don't try to do everything all together. There are several steps involved here, so break the problem down into steps that will have intermediate data before getting the final data. Built-in functions are your friends. Note column headings. You don't need intermediate files.

I suggest using `DictReader` to read the file. Read the whole file in at one time. Each element in the resulting list will be a *dictionary* of row data, containing the column name & cell value as the key, value pairs.

Then make a new list such that each element (namely, row) consists of a dictionary that only contains the values (columns) from the initial list that we want.

Now, you can make a new list of row data, in order, by using the built-in function `sorted()`. Look up the inputs to `sorted()`, since the values of the dictionary are only strings, not numbers. You may want to define a helper function.

Then write out the sorted list data with `DictWriter`.

If you have a Windows computer: Be sure to use `newline=''` in the file open statements. Most importantly, don't print all the country or capital names in debug print statements (maybe just do the first few lines if you really think it's necessary), because, amazingly enough, the Windows Command window does NOT support display of UTF-8 characters,

even in Windows 10, so your print statements will crash the program. This totally blows my mind.