CS 112 – Fall 2020 – Programming Assignment 10

File I/O

Due Date: Tuesday, November 10th, 11:59pm

Tester Release: Friday, November 6th

See the **"assignment basics"** file for more detailed information about getting assistance, running the test file, grading, commenting, and many other extremely important things. Each assignment is governed by the rules in that document. Make sure that you are skimming Piazza regularly for any clarifications that get made to the assignment.

Background

The purpose of this assignment is to practice manipulating files in Python. You will be calculating a few different things based on real hydroelectric dam data from the United States.

Restrictions

Any function that violates any of the following will receive **zero** points even if it passes all tests.

- You are **not** allowed to `import` anything. No, you **may not** use the csv module.
- You are **not** allowed to use anything that hasn't been covered in class, including but not limited to: *list comprehension, lambda functions, generators, builtins like zip(), etc.*
- You can take advantage of the file's structure, but you should not hard code to any specific file.
- If it has been covered in class, you can use it. The following are things you may have covered, but are explicitly forbidden for this assignment:
  - list.count()
  - string.find(), string.replace()
  - sum()

NOTICE*:*

The visualizer does not work with File I/O. If you want to work in the visualizer, you have to copy the files in as strings, and leave the File I/O (open/process/close) out of your code.

There are better ways to test. *Seek help sooner rather than later.*

Testing

From this assignment on, testing will be done a bit different than before. The grader we use to grade your work will be different from the tester we provide. This means that there will be cases that the tester doesn't check and, therefore, your code might have logic errors that the tester doesn't detect. You must do additional checks, on your own, to make sure that you haven't missed anything and your code is correct. You do ***not*** need to modify the tester. The goal is to help you put more focus on writing logically correct programs instead of trying to pass certain tests only. Thus, the grader (a.k.a. autograder) will be made available on Gradescope a few days after the release of this assignment, and will include '**hidden**' test cases.

Grading Rubric

Submitted correctly:	2	# see assignment basics file for file requirements!
Code is well commented:	8	# see assignment basics file for how to comment!
Autograder:	90	# see assignment basics file for how to test!
TOTAL:	100	

Note: If your code does not run and crashes due to errors, it will receive **zero** points. Turning in running code is essential. **Do not submit code with uncommented print statements**

Assignment

You are going to work with data related to hydropower in the United States. You are given files that contain a variety of information about dam projects. Familiarize yourself with the file format **before** you start on these functions.

Hydropower files are in the "csv" format, short for "comma separated values." These are just plain text files to represent a spreadsheet of information, where each piece of data is separated by a comma. The first row of each file (called the "header" row) is a description of what each column of information represents. Here is a listing (in order by column as seen in a hydropower file) of what the data represents

- **CrestElevation:** the height of the dam (relative to sea level)
- **CrestLength:** the length of the dam
- **StructuralHeight:** the height of the dam
- **Name:** the name of the dam
- **Watercourse:** the waterway where the dam is located
- **County:** the county of the state where the dam is located
- **Latitude:** first part of the exact GPS coordinates for the dam
- **Longitude:** second part of the exact GPS coordinates for the dam
- **State:** the state where the dam is located
- **Organization:** who is responsible for building the dam
- **Year:** what year the dam was built

The signature of each function is provided on the following pages.

Do ***not*** make any changes to them otherwise the tester will not work properly.

Be sure to download all of the provided example files and place them in the same folder as your Python file on your computer.

Otherwise, the examples **will not work**.

def dams_by_state(inputfilename) [25pts]

Description: Implement a function that transforms a file into a dictionary of data to make working with these files a little easier.

Parameters: `inputfilename` is a string, it is the name of the hydroelectric file you will use for this function (you have to open the file!)

Assumptions:

- the file given by `inputfilename` will exist and contain at least the header row
- you only have to include states that are represented in the file
- the data about each dam should be stored as the appropriate type

Return value: A multi-level dictionary. The first level maps state abbreviations to dams in that state. Each dam in a state is another dictionary which maps dam names to information about that dam. The information about each dam is a dictionary that maps the column headings to specific data about this dam.

Examples:

```
dams_by_state('dams0.csv') returns the following structure
{ # level 1: abbreviations as keys, dictionary of dams as values
    'CA':
        { # level 2: dam names as keys, dam info as values
            'ContraLomaDam':
                { # level 3: info type as key, value from file
                    'CrestElevation': 217.0,
                    'CrestLength': 1050.0,
                    'StructuralHeight': 107.0,
                    'Watercourse': 'offstreamstorage',
                    'County': 'ContraCosta',
                    'Latitude': 37.9772,
                    'Longitude': -121.8231,
                    'Organization': 'CentralValey',
                    'Year': 1966
                }
        }
}
```

`def count_dams(inputfilename, state="VA")` [20pts]

Description: Count the number of hydroelectric dams that are located in the given state. By default, count the number of dams that are in Virginia.

Parameters: **`inputfilename`** is a string, it is the name of the hydroelectric file you will use for this function (you have to open the file!) **`state`** is the two-letter abbreviation of one of the 50 US States

Assumptions:

- the file given by **`inputfilename`** will exist
- the file named by **`inputfilename`** will contain at least the header row

Return value: an integer; the number of dams in the file located in the given state

Examples:

```
count_dams('dams1.csv')          returns 0
count_dams('dams1.csv', 'CA')    returns 1
count_dams('dams1.csv', 'ID')    returns 0
count_dams('dams1.csv', 'OK')    returns 2

count_dams('dams2.csv', 'OR')    returns 0
count_dams('dams2.csv', 'WY')    returns 1
count_dams('dams2.csv', 'ID')    returns 2
count_dams('dams2.csv', 'MD')    returns 0
```

def dams_in_area(inputfilename,x1,y1,x2,y2) [20pts]

Description: Consider the coordinate **(x1, y1)** as the bottom left-hand corner of a square, and the coordinate **(x2, y2)** as the top right-hand corner of the square. Return the set of dam names who are within this square, where the longitude of the dam's location is on the x-axis and the latitude of the dam location is on the y-axis.

Parameters: **inputfilename** is a string, it is the name of the hydroelectric file you will use for this function (you have to open the file!) **x1,y1,x2,y2** are floating point values as described above

Assumptions:

- the file given by **inputfilename** will exist and contain at least the header row
- the coordinates are not guaranteed to form a square
- **x1,x2** will be between -180 and 180. **y1,y2** will be between -90 and 90

Return value: A set of dam names that exist in the particular square.

Examples:

```
dams_in_area('example_dams.csv', -115, 40, -100, 45)
➔ {'AnchorDam', 'AmericanFallsDam'}

dams_in_area('example_dams.csv', -100, 30, -75, 50)
➔ {'ArbuckleDam', 'FossDam', 'AltusDam'}

dams_in_area('example_dams.csv',-70,0,0,40)
➔ set()    # this is the empty set because there are no dams here

dams_in_area('example_dams.csv',100,100,0,0)
➔ set()    # this is the empty set because this square can't exist
```

def average_by_state(inputfilename) [25pts]

Description: Compute the average crest elevation, the average crest length, and the average structural height of all the dams in the file. Each average is computed **per state**, not over all dams in the file. You will write the average information per state into a new file called "averages.csv" which should be sorted by state.

Parameters: **inputfilename** is a string, it is the name of the hydroelectric file you will use for this function (you have to open the file!)

Assumptions:

- the file given by **inputfilename** will exist and will contain at least the header row
- you only have to include states that are represented in the file
- Round all final results to one decimal place

Return value: None. All of the results will be written to a file called "averages.csv", using the header seen in the examples. Make sure you sort by state! The easiest way to do this is to create a list of states, and sort that. s

Examples:

```
average_by_state('dams1.csv') # 'averages.csv' should look like:

State,AverageCrestElevation,AverageCrestLength,AverageStructuralHeight
CA,217.0,1050.0,107.0
OK,1308.5,10010.0,146.0
```

```
average_by_state('dams2.csv') # 'averages.csv' should look like:

State,AverageCrestElevation,AverageCrestLength,AverageStructuralHeight
CA,217.0,1050.0,107.0
ID,4286.2,3313.5,280.0
MT,3010.0,1050.0,42.0
OK,1393.7,7041.3,134.0
OR,4064.0,324.0,23.0
WA,2301.0,1075.0,72.0
WY,6452.5,660.0,208.0
```