ASSIGNMENT 2

Contents

[Contents 1](#_Toc493068213)

[Purpose 2](#_Toc493068214)

[Requirements: 2](#_Toc493068215)

[Code/Comment Format 4](#_Toc493068216)

[What to Deliver 5](#_Toc493068217)

[Notes 5](#_Toc493068218)

[Grading 5](#_Toc493068219)

# Purpose

In this assignment, you will be demonstrating your understanding and use of Python constructs, as covered in module 2.

This assignment is design to help you practice your skills at working with files. You will read data from a CSV file and, following input from the user, save view of the data to a CSV file.

# Requirements:

NOTE: You are required to use the given dataset found in the attached file named:

“2010\_Census\_Populations\_by\_Zip\_Code.csv”.

1. Initialize in the global scope a tuple listing the names of the columns of the CSV.
2. Let’s read the data into memory.
   1. Using what you have learned in the past 2 modules, create a ***load\_census\_data*** function.
      1. The function should have 1 parameter: the path to the CSV file
      2. The function must use ***with*** to open the file
      3. Using the ***csv*** module (https://docs.python.org/3.6/library/csv.html#csv.DictReader), you must read the contents of the file. The DictReader allows you to get each line of the CSV as a dictionary. The keys of the dictionaries correspond to the names of the columns and the values are the values for that row.
      4. Use exception handling appropriately and all the tool you have learnt so far:
         1. to validate that all the column names are found in each row’s keys
         2. to validate that all corresponding values are ints. Replace the validated values for a key into to the dictionary.
         3. To discard any row that does not have valid data (do not add it to the list)
      5. Place each validated row (from iii) into a list.
      6. The function must return a ***tuple*** by converting the list into a tuple.

*Note: the returned data structure is a tuple containing dictionaries.*

1. Invoke ***load\_census\_data*** to get the data stored in the CSV file by passing the path to the file as an argument. It is OK to hard code the path to the CSV file.
   1. Keep a reference to the resulting tuple by creating a variable
2. Now that we have data let’s ask the user what data they want to retrieve from the data set:
   1. Using the tuple of all the column names, enumerate (https://docs.python.org/3/library/functions.html#enumerate) all the column names and display to the user.

Like so:

1) Zip Code

2) Total Population

3) Median Age

4) Total Males

5) Total Females

6) Total Households

7) Average Household Size

* 1. ask the user which column to filter data by.
  2. Read the input from the user (the integer).
  3. Validate user input:
     1. User input corresponds to one of the sought numbers [1, 7].
     2. If the user’s input is invalid, ask for re-entry.
     3. Allow the user to retry three times and then exit the program if the user keeps providing an invalid value.

1. Let’s ask the user to provide a floor value to filter our data.
   1. Read the input from the user.
   2. Validate the user input:
      1. User input is a positive integer.
      2. In the same way as 4.diii, allow three retries and then exit the program.
2. Now that we have the column name and a floor value to filter by, we need to filter the data by writing ***filter\_data\_by\_column\_and\_floor*** (place the function in the same file as ***load\_census\_data*** and declare the new function right below it to keep your code organized).
   1. the function must take a tuple (our data), a column name, and a floor value as parameters
   2. iterate through each row of the data and build a new list of all the “rows” having the selected column’s value greater than or equal to the floor value.
   3. Return a tuple built from the list created in b.
3. Invoke ***filter\_data\_by\_column\_and\_floor*** with the appropriate arguments: the tuple of rows (our data), the column name selected by the user at step 4. (we want the string value here e.g. “Zip Code”.), the floor value selected by the user at step 5.
4. Now that we have data filtered let’s ask the user to know what column they want to sort by.
   1. Read the input from the user (the integer).
   2. Remember to validate the user input:
      1. the input corresponds to one of the sought numbers[1, 7].
      2. Apply the same validation you did in 4.d.
5. Using the ***sorted*** function (<https://docs.python.org/3.6/library/functions.html#sorted>), sort the filtered data by the column name selected by the user. You may need to create a function as ***key*** to sort. (use the default sorting order).
6. Now, let’s save our data to a CSV file so the user can use it later. We are going to place the file into a folder named ***exports***.
   1. Create a folder named exports if it does not already exist ( <https://docs.python.org/3/library/os.html#os.makedirs> ). It is OK to create it wherever you want on the file system.
   2. Prompt the user for a file name (make sure that is ends with “.csv” and is not empty).
   3. Use the filename in conjunction with the “os.join” python function to create the path ( <https://docs.python.org/3/library/os.path.html#os.path.join> ) where the file will be saved under the folder ***exports***
   4. Make sure that the file does not already exist (ask for a new filename if it does)
      1. Allow the user to retry three times and then exit the program if the user keeps providing an invalid filename.
   5. Write the filtered, sorted data to the file.
      1. Make sure to use ***with*** to open the file
      2. Use the ***csv*** module to write the file (https://docs.python.org/3.6/library/csv.html#csv.DictWriter).

# Code/Comment Format

Good code includes well named variables that are consistent from the beginning to the end of the program. Naming of objects should be self-explanatory. For instance, iterator\_for\_noun\_list is much better than i.

Every program consists of a sequence of paragraphs, each of which has objectives, and which builds on the previous paragraphs. We are mostly interested in objectives that are valid at the end of the program so we can verify the program's design. The following is a preferred form for such paragraph headings. The # sign is adequate when the comment is a single line.

#This is an in-line comment – used to document the code for you, or anyone else, that intends

#To extend the code

In-line comments are helpful when one has to go back to the code 6 months later to make changes.

For doc strings, python allows the use of triple quotes. The triple quotes can be either single or double quotes. A doc sting is generally used as user documentation. It does not need to include details of the implementation of the program, but instead it provides documentation as how to use the API for the program (input, output etc.)

For example:

“””

This is an example of a doc string

It allows multiple lines within the string.

“””

‘’’

This is an example of a doc string

It allows multiple lines within the string.

‘’’

This becomes significant when using functions, classes etc. as the triple quotes help to self-document the parameters and return values of the function.

# What to Deliver

You are required to supply *Last Name\_First Name\_Assignment2.py.*

# Notes

* Assignments can be submitted once. If extenuating circumstances exist, contact your facilitator.
* Start by identifying and ordering the objectives.
* There are no testing requirements for this assignment. However, it would be prudent to make sure your program does not crash and all input validation is performed correctly.
* Note the statement in the syllabus on timeliness of submissions (the gist being that all assignments must observe the deadlines). A ten point deduction per day late up to 20 points. After 2 days late, a 0 will be given. Please see the syllabus. This is waived ONLY with the prior approval of your facilitator.

# Grading

Requirement Points

1. 2
2. 18 (i. thru vi. 3 points each)
3. 2
4. 18 (a through c 5 points each, di-iii 1 point each)
5. 6 ( a – 2 points, bi-ii, 2 points each)
6. 9 (a-c 3 points each)
7. 4
8. 6 ( a – 2 points, bi-ii, 2 points each)
9. 15
10. 20 – (a-d 4 points each, ei-ii 2 points each)