CS1026: Assignment 4: Country Classes

Due: Wednesday, December 5th 2018 at 9:00pm
Weight: 13%

Learning Outcomes:
By completing this assignment, you will gain skills relating to

- Strings and text files
- Writing your own classes
- Testing code
- Using complex data structures (i.e., sets and dictionaries)

In this assignment you will create a complete program that uses classes to store, search, sort, remove, and filter country data. The two major tasks are outlined below.

TASK 1

Implement a class **Country** that holds the information about a single country; name the file **country.py**.

Instance Variables

1) name: The name of the country (string)
2) population: The population in the country (integer)
3) area: The area of the country (float)
4) continent: The name of the continent to which the country belongs (string)

Methods

1) Constructor, __init__(self, name, pop, area, continent)
2) Getter Methods: getName, getPopulation, getArea, getContinent,
3) Setter Methods: setPopulation, setArea, setContinent
4) getPopDensity: This calculates and returns the population density for the country. Population density is the population divided by the area rounded to 2 decimal places.
5) **def** __repr__(self): generates a string representation for class objects.

   Example of output from __repr__
   Name (pop: population value, size: area value) in Continent
   e.g Nigeria (pop: 11723456, size: 324935) in Africa

Test all your classes and methods before moving to the next section. Feel free to create other helper methods if necessary.

TASK 2

Implement a class called **CountryCatalogue;** name the file **catalogue.py** .
This class will use two files to build the data structures to hold the information about countries and continents. The file `data.txt` contains information about a number of countries and the file `continent.txt` contains information about countries and continents.

This class should have the following two instance variables:

1. **countryCat:** This is a collection of countries, i.e., each member is an object of the class **Country**. The collection could be a set, dictionary or list. **It is important that the collection <u>store objects of type Country</u>.**
2. **cDictionary:** This is a dictionary **<u>where the key is the country name</u>** and the **<u>value is the name of the continent</u>**.

   You may add other instance variables as you see fit.

Methods

1. **Constructor:** The constructor method will take as parameters two files:

   - the ***first file*** parameter is the name of the file that contains the information about each country.
   - the ***second file*** parameter is the name of the file that contains the country and continent information.

The constructor will use the data in the files to construct the two data structures **countryCat** and **cDictionary**. It does this as follows:

- **Fill the Dictionary:** Using the second file, fill **cDictionary**; the key is the country name, while the name of the continent is the value.
- **Fill the Catalogue:** Using the first file, read each line of the file and from that create a country and add it to **countryCat**. (Note: you will also need to use the data in cDictionary to determine the continent for each country)

  Sample data files has been included: `continent.txt` and `data.txt`
  <u>Notes</u>*: *a) Both files have headers and b) these files are meant for you to test your program; they may NOT be the same used to evaluate your solution.

2. **findCountry:** This method allows a user to look up information on countries. It takes as a parameter a country name and checks the catalog. If the operation is successful (i.e., if the country exists in the catalog), the method should return the *<u>country object</u>*, if it is not successful the method should return the null object (i.e., **None**).

3. **setPopulationOfCountry**: This method will take as parameters a country name and new population and then set the population of the country (if it is in the catalogue) to the value. The method should return **True** if the operation is successful, or **False** if it is not, e.g. the country is not in the catalogue.

4. **setAreaOfCountry**: This method will take as parameters a country name and a new area and then set the area of the country (if it is in the catalogue) to the value. The method should return **True** if the operation is successful, or **False** if it is not, e.g. the country is not in the catalogue.

5. **addCountry:** This method provides a way to add a new country to the dictionary of countries and country catalogue. The method takes as parameters the country name, population, area and continent of the country. The method should return **True** if the operation is successful (country successfully added), or **False** if it is not, e.g. the country they entered already exists. In addition to adding the newly created country to the country catalogue, make sure you add the country and continent to the cDictionary.

6. **deleteCountry:** This method will take as parameter a country name and then if this country exists then it should be deleted from the catalogue and from cDictionary. If the country does not exist, no further action is needed. (This method doesn't return anything).

7. **printCountryCatalogue:** This method displays the whole catalogue to the screen, using the default string representation for the Country objects.

8. **getCountriesByContinent:** This method will return a list of countries on a specific continent. This method takes as a parameter the name of a continent. If the parameter is the name of a valid continent, then it will return a list of the countries on that continent, otherwise it returns an empty list. (Note: the return type is list and the elements of the list are Country objects)

9. **getCountriesByPopulation**: This method will return a ***list of pairs (tuples)*** where a pair is a country name and its population. The list should be in descending order by population from the largest to the smallest. This method takes as a parameter the name of a continent as input or takes nothing at all (default is the empty string). If the parameter is the name of a valid continent, then it will return a list of only the countries and their population on that continent – by population from largest to smallest. If the continent name is the empty string (default), then it will return a list of all the countries and their population in the catalogue. For an input value that is not the empty string and not a valid continent, it returns an empty list.

10. **getCountriesByArea**: This method will return a ***list of pairs (tuples)*** where each pair is a country name and their area (just the value, no units). The list should be in descending order by area from the largest to the smallest. This method takes as a parameter the name of a continent as input or takes nothing at all (default is the empty string). If the parameter is the name of a valid continent, then it will return a list of only the countries and their areas on that continent – sorted by area from largest to smallest. If the continent name is the empty string (default), then it will return a list of all the countries and their areas in the catalogue (once again sorted by area). For an input value that is not the empty string and not a valid continent, it returns an empty list.

11. **findMostPopulousContinent**: This method will find and return the name of the continent with the most number of people living in it ***and*** the number of people living in the continent. This should return a tuple (continent name, population of continent). You cannot assume that the continent names are fixed, therefore you must first determine what are the continents that exist in your dataset.

12. **filterCountriesByPopDensity:** This method will take as parameters as a lower bound and upper bound for a population density range and then return a list of countries that have a population density that falls within the range (inclusive of the endpoints). The population density

is the population divided by the area. It should return a list of pairs (tuples), where each pair is a country name and a population density. The list should be descending order from the highest density to the lowers. If there are no countries with population density within the range, it should return an empty list.

13. **saveCountryCatalogue:** This method will enable all the country information in the catalogue to be saved to a file. The method takes as a parameter the name of the file. All the countries should be sorted alphabetically by name (A – Z) prior to saving. The file should appear in the following format:

Format:
```
Name|Continent|Population|AreaSize|PopulationDensity
```

For example: Canada|North America|34207999|9976200.00|3.43

Note: no spaces between attributes, 2 decimal places for the area and the population density. If the operation is successful, the method should return the number of items written, otherwise it should return a value of -1.

You may add other methods as you see fit.

Test all your classes and methods before moving to the next section. Feel free to create other helper methods if necessary.

The **main.py** file is provided as the interface to your program. It has been designed to call the methods of the classes as named above; ***make sure that you use these names***. You may use the functions in **main.py** to test your program in other ways. ***NOTE:*** **while main.py will test some aspects of your program it does not do a complete and thorough testing – this is left up to you. The TA will use a program similar to, but different than, *main.py* to grade your assignment - it will include other tests.**

For output to both the console and file you do not need to format numbers to include commas. (e.g 1000 does NOT need to be written as 1,000)

You may assume all the data is correct and there are no errors relating to the data file *(so don't worry about Exceptions or validating input except where otherwise noted for this assignment).*

The following may be helpful in sorting data structures
http://pythoncentral.io/how-to-sort-a-list-tuple-or-object-with-sorted-in-python/
http://pythoncentral.io/how-to-sort-python-dictionaries-by-key-or-value/

Non-functional Specifications:

1. Include brief comments in your code identifying yourself, describing the program, and describing key portions of the code.
2. Assignments are to be done individually and must be your own work. Software may be used to detect cheating.
3. Use Python coding conventions and good programming techniques, for example:
   i. Meaningful variable names
   ii. Conventions for naming variables and constants
   iii. Readability: indentation, white space, consistency

Submit the files in which your classes are defined. The name of the file for the **Country** class should be **country.py** and the name of the file for the **CountryCatalogue** class should be **catalogue.py**.
Make sure you attach your python file to your assignment; DO NOT put the code inline in the textbox.

Make sure that you develop your code with Python 3.7 as the interpreter. TAs will not endeavor to fix code that uses earlier versions of Python.

What You Will Be Marked On:

1. Functional specifications:
   - Does the program behave according to specifications?
   - Does it run when tested with a program similar to the main program provided?
   - Are your classes created properly?
   - Are you using appropriate data structures?
   - Is the output according to specifications?
2. Non-functional specifications: as described above
3. Assignment submission: via OWL assignment submission