CS1026: Assignment 4

Country Classes

Due: Thursday, December 9th 2020, 9:00pm
Weight: 12%

>Learning Outcomes

By completing this assignment, you will gain skills relating to

- Strings and text files
- Writing and using your own classes
- Using complex data structures (i.e., lists, sets and dictionaries)
- Testing code and developing test cases; adhering to specifications

>Background

Data files are very common in many disciplines. They form the input for a variety of different kinds of processing, analysis, summaries, etc. An associated problem is to maintain these data files – to add information, update or correct information. Manually updating data files can be tedious and can lead to errors. A common approach is to use a program – an "update" program that takes a data file and a file of "updates" and then creates a new version of the data file with the updates applied.

In this assignment you will create a complete program that will update an existing data file containing information about countries. The tasks are outlined below.

>Tasks

✓ TASK 1

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

1. Instance Variables

- *name*: The name of the country (string)
- *population*: The population in the country (string)
- *area*: The area of the country (string)
- *continent*: The name of the continent to which the country belongs (string)
  Note*: the instance variable* population *and* area *can be left as strings for this assignment as there are not specific computations involving either. The update program just needs to update these values and save them to a file.*

2. Methods

- Constructor, `__init__(self, name, pop, area, continent)`
- Getter Methods: `getName`, `getPopulation`, `getArea`, `getContinent`
- Setter Methods: `setPopulation`, `setArea`, `setContinent`
- **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 a file to build the data structures to hold the information about countries. The file `data.txt` contains information about a number of countries. An example of the data file is:

```
Country|Continent|Population|Area
Brazil|South America|193,364,000|8,511,965
Canada|North America|34,207,000|9,976,140
China|Asia|1,339,190,000|9,596,960
Colombia||48,654,392|1,090,909
Egypt|Africa|93,383,574|
```

Notice that:

- There is a header line: "`Country|Continent|Population|Area`". This is really for information purpose for the data file; it is a reminder of the format of the data in the file.
- The file consists of "records" for each country – one per line with the information as described in the header line.
- The different "fields" of each "record" are separated by vertical bars, "|".
- A field (except for country name) may be missing (see Colombia and Egypt) but the record will still have the three vertical bars.

The class `CountryCatalogue` should have the following instance variable:

- `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 store objects of type Country.**

You may add other instance variables as you see fit.

>Methods

1. Constructor, `__init__` **(self, countryFile)**
   The constructor method will take a single parameter (besides "self") that is the name of the file that contains the information about each country. The constructor will use the data in the file to construct the data structure **countryCat.** Sample data files has been provided: `data.txt` and `data2.txt`. [***Note***: ***Both files have headers and these files are meant for you to test your program; they may NOT be the same used to evaluate your solution.]***

2. Setter Methods: `setPopulationOfCountry, setAreaOfCountry, setContinentOfCountry`

3. `findCountry(self,country)`
   Given a country object (that is, an object of the **Country** class), this methods checks to see if that country object is in `countryCat` . If it is, it just returns the country object, but if it is not, it returns the null object **NONE**.

4. `addCountry(self,countryName,pop,area,cont)`
   Given the name, population, area and continent of a country, this method adds a new country to `countryCat`; it should only be added if the country does not exist in `countryCat` . 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.

5. `printCountryCatalogue(self)`
   This method prints a list of the countries and their information to the screen; it should make use of the **repr** method of the **Country** class.

6. `saveCountryCatalogue(self,fname)`

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. If the operation is successful, the method should return the number of items written, otherwise it should return a value of -1. ***The file should appear in the exact same format as the original file, namely:***

```
Country|Continent|Population|Area
Brazil|South America|193,364,000|8,511,965
Canada|North America|34,207,000|9,976,140
China|Asia|1,339,190,000|9,596,960
Egypt||93,384,001|1,000,000
Indonesia||260,581,345|
```

Notice that:

- There should be no spaces between the fields and vertical bars.
- If a field has no value, then nothing should be printed, e.g. see Egypt and Indonesia with no continent value and Indonesia with no area given.

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.

✓ TASK 3

Implement a Python module, called `processUpdates.py`, that has a function `processUpdates(cntryFileName,updateFileName)`. This function takes two parameters: the first parameter will be the name of a file containing country data (e.g. `data.txt`). The second file will contain the name of a file containing *updates*. Each record in the update file specifies updates for a single country. The format of a record in the update file is as follows:

```
Country;update1;update2;update3
```

Where an "update" is of the form "L=value" and L can be **P**, for population, **A** for area and **C** for continent. Updates are separated by semicolons and a line can have 0, 1, 2 or 3 updates; ***any updates after the first 3 are simply ignored and you need not worry about their values.***

Valid values in an update are as follows:

- For **Population** and **Area**, a valid value is a string of digits with commas separating groups of 3. Examples of valid values are "123,456", "1,333,456", "23,333,444". Examples of invalid values are "123456", "123 345 555", "123,344.44".
- For **Continent**, a valid value is one of the continents: "Africa", "Antarctica", "Arctic", "Asia", "Europe", "North America", "South America". The string of letters can be upper or lower case and there may be more than one space between the "North" and "America" and the "South"

and the “America”. Once a valid continent is found, the first letter of the name or parts should be capitalized and there should only be 1 space between the “North” and “America” and the “South” and the “America”. For example, “asia” should be stored as “Asia”, “aFRICA” should be stored as “Africa”, “north America” should be stored as “North America”.

An example of an update file is:

```
Brazil;P=193,364,111;A=8,511,966
Indonesia;P=260,581,345
Japan;P=127,380,555;A=377,800
Sweden;P=9,995,345;A=450,295;C = europe
Italy; P = 59,801,999
Canada;C=North America
Egypt;A=1,000,000 ; P=93,384,001
France;P = 64,668,129;P=64,668,150;A=541,666;C=Europe
```

Notice that:

- The updates can be in different orders, e.g. the updates for Egypt are area first, then population.
- If a country is specified and that country is ***NOT*** already in the catalogue ***then that country and its attributes (at least the ones specified; not all have to be specified) should be added to the catalogue (again, any attributes beyond the first 3 are to be ignored)***.
- There can be spaces before and/or after each semicolons and equal signs; these spaces can be ignored.
- There can also be spaces in other places, such as in country names or within numbers. These represent errors and could cause problems in processing. You should treat these as errors (do not try to fix them) and should be handled as exceptions (see below).
- In the updates to **France**, for example, there are 4 updates; the last update is simply ignored and your program should only consider the first 3. As well, the first three updates to **France** contain two updates for population. These are applied in order. This would mean that the update **P = 64,668,129** is applied and then the update **P=64,668,150** is applied, essentially overwriting the first update.

The function `processUpdates` gets two files as parameters. The first parameter is the name of the file with the country data and the second parameter is the name of file with the update data.

The country file should be processed first. The function `processUpdates` should ensure that it exists. If it does exist, it should then proceed to process the data file using the class `CountryCatalogue`. If the country file does not exist, then `processUpdates` should give the user the option to quit or not (prompts for “Y” (yes) or “N” (no)). If the user does not wish to quit (responds with “N”), then the function should prompt the user for the name of a new file. If the user wishes to quit (responds with a “Y” or anything other than a “N”, then the method should exit and

return **False**. The method should continue to prompt for file names until a valid file is found or the user quits.

If the user chooses to quit with no country file processed, then the function `processUpdates` should **NOT** try to process the updates file and should write to the file "`output.txt`" the message **"Update Unsuccessful\n"** – just the single line. It should then exit (i.e.,) and return `False`.

If the function `processUpdates` has successfully processed the country file, then it should proceed to process the file of updates. It should prompt the user in a similar manner to how the country file was input, i.e., using a loop to continuously prompt the user until a file was found that existed or until the user quits.

If the user quits without an update file being selected, then the function `processUpdates` should write to the file "`output.txt`" the message **"Update Unsuccessful\n"** – just the single line. It should then exit and return `False`.

If an update file is found, then the function `processUpdates` should process the updates, i.e., update the information about the countries and then output the new updated list of countries in alphabetical order to the file "`output.txt`". The output should use the method `saveCountryCatalogue(self,fname)` from the class `CountryCatalogue` (see Task 2). It should then exit and return `True`.

In processing the updates from the update file, the updates may have invalid values (see above). **If an update has an invalid value, then the entire line should be skipped and none of the updates processed**. It is difficult to anticipate all possible problems with the updates, and some may cause your program to fail – that is, some invalid inputs **MAY** cause exceptions when processing the line – your program should be designed to catch any of these as exceptions and then print a message and skip all processing of that line.

The Python program `main.py` file is provided as the interface to your program. It has been designed to prompt for the names of a country file and a file of updates and then call the function `processUpdates`; ***make sure that you use these names of functions and methods as described above***. You may use the `main.py` to test your program in other ways. Two additional data sets of countries have been provided, `data2.txt` and `data3.txt`, as well as two different files of updates, `upd2.txt` and `upd3.txt`. You may use these to test your program. ***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. A program similar to, but different than, *main.py* will be used to test your program - it will include other tests.**

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:
   - Meaningful variable names
   - Conventions for naming variables and constants
   - Readability: indentation, white space, consistency

Submit the files in which your classes and functions are defined:

- `country.py` containing the class `Country`;
- `catalogue.py` containing the class `CountryCatalogue`;
- `processUpdates.py` contain the function `processUpdates`.

Make sure you attach your python file to your assignment; DO NOT put the code inline in the textbox.

>What You Will Be Marked On:

1. **Your program will be executed by an automated testing program**. This testing program assumes that:
   - The modules are named `main.py`, `procesUpdates.py`, `catalogue.py` and `country.py`.
   - That you are using Python 3.8.
   - That you have submitted it via OWL by uploading it.

Failure to adhere to these constraints will likely cause the testing program to fail. This may require a remarking of your program which will include a 20% penalty.

2. **Functional specifications:**
   - Are the modules, classes and functions named correctly for testing?
   - Is there a program `processUpdates.py` which has a function `processUpdates`?
   - Does the program behave according to specifications? Does it work on with the test program, `main.py`?
   - Is there an effective use of functions and methods?
   - Is the output according to specifications?

3. **Non-functional specifications**: as described above.

4. **Assignment submission**: via the OWL, though the assignment submission in OWL.