Severance Chapter 14 Coding Assignment

General Instructions

My expectations for your work on coding assignment exercises will grow as we progress through the course. In addition to applying any new programming techniques that have been covered in the current chapter, I will be expecting you to follow all of the good programming practices that we have adopted in the preceding weeks. Here is a quick summary of good practices that we have covered so far:

- Include a single-line comment with name of program file.
- Include a single-line comment that describes the intent of the program.
- Place your highest-level code in a function named `main`.
- Your code should be factored such that there is a function in your program for each part of the problem.
- Each function should contain code relating to the same thing – it should have *high cohesion*.
- Functions should know as little as possible about the workings of other functions – they should have *low coupling*.
- Include a final line of code in the program that executes the `main` function.
- Follow all PEP-8 Python coding style guidelines enforced by the PyCharm Editor. For example, place two blank lines between the code making up a function and the code surrounding that function.
- Output printed by the program (both prompts and results) should be polite and descriptive.
- Choose names for your variables that are properly descriptive.
- Choose names for your functions that are properly descriptive.
- Close all files before the conclusion of the program.
- Model your solution after the code that I demonstrate in the tutorial videos.
- Remember to test your program thoroughly before submitting your work.
- Your code must pass all relevant test cases. Make sure that it passes tests at the boundaries created by *if, else*, and *elif* conditions in your program (boundary value tests).

Assignment Overview

The goal of this assignment is to demonstrate skills needed to design, code, and test custom Python classes. To do this, we revisit the solution to one of the exercises that was part of the Zelle Chapter 11 coding assignment:

country_population_lists.py (see starter files)

In that exercise, we used a tuple-based data structure to hold the values for each country/population data fact. In this assignment, we will be working with the same data and producing similar output. The difference will be that we will use instances of a custom Python class to hold values for each country/population data fact instead of using a tuple.

In this assignment, we will create two successive versions of a custom Python class named *Country* that holds Country/Population data facts.

First, we will create a basic version of the *Country* class that does not contain any getter or setter capabilities. Then, we will demonstrate the use of this basic version of *Country* using a revised version of the *country_population_lists* program that we created in earlier in the semester.

Next, we will create an evolved version of the *Country* class that includes both getter or setter capabilities. Then, we will demonstrate the use of this evolved version of *Country* using a further revised version of the *country_population_lists* program that we created in earlier in the semester.

There are 4 exercises that make up this coding assignment:

1. Create the basic version of the *Country* class.
2. Create *country_population_lists_using_basic_class* program to demonstrate the use of the basic version of the *Country* class (see starter files).
3. Create the evolved version of the *Country* class.
4. Create *country_population_lists_using_evolved_class* program to demonstrate the use of the evolved version of the *Country* class (see Exercise 2).

This assignment has a starter files zip file that contains the following:

- country_pop_data.txt
- broken_country_pop_data.txt
- country_population_lists.py

Exercise 1
Create a Python module file named *my_basic_countries.py*. This module should contain the following:

1. A class named *Country* that contains the basic implementation of the class that holds Country/Population data facts.

2. A *main()* function that contains unit test cases for the *Country* class.

This basic version of the *Country* class should NOT contain explicit getter or setter features. When doing this part of the assignment, follow the approach that I demonstrated in the Part 1 tutorial video.

When this program is run directly (rather than having been imported), the console session should contain the unit testing output and should look like this:

```
Unit testing output follows...

Test 1:  Test Constructor
Passed

Test 2:  Test setting name
Passed

Test 3:  Test setting population
Passed

Test 4:  Test __str__ Method
Passed
```

Exercise 2

Create a program named *country_population_lists_using_basic_class* by copying the *country_population_lists.py* program included in the starter files and modifying the code to fit the specifications for this exercise. When coding and testing this program, follow the approach that I take in the Part 2 tutorial video. The changes that you make to the starter file should include:

- Change the program name.
- Import the proper version of the *Country* class.
- Use instances of the *Country* class to hold Country/Population data facts instead of using tuples.
- Change the names of variables and parameters to eliminate references to the word *tuple*.
- Change the names of functions to eliminate references to the word *tuple*.
- Add code that creates a third listing in the output in descending order of population with the title DESCENDING POPULATION ORDER.

When testing, use the *country_pop_data.txt* file included in the starter files as your input data file.

When testing, please check that the unit test output from the *my_basic_countries.py* module is NOT printed in the test output for this program.

When this program is run, the console session should look like this:

```
Please enter input file name: country_pop_data.txt

COUNTRY NAME ORDER
Bangladesh       165435000
Brazil           209772000
China           1394860000
India           1338820000
Indonesia        265015300
Japan            126440000
Nigeria          193392517
Pakistan         202477000
Russia           146877088
USA              328077000

POPULATION ORDER
Japan            126440000
Russia           146877088
Bangladesh       165435000
Nigeria          193392517
Pakistan         202477000
Brazil           209772000
Indonesia        265015300
```

```
USA              328077000
India           1338820000
China           1394860000

DESCENDING POPULATION ORDER
China           1394860000
India           1338820000
USA              328077000
Indonesia        265015300
Brazil           209772000
Pakistan         202477000
Nigeria          193392517
Bangladesh       165435000
Russia           146877088
Japan            126440000
```

Exercise 3

Create a Python module file named *my_evolved_countries.py* by copying the module created in Exercise 1 and making changes. This module should contain the following:

1. A class named *Country* that contains the evolved implementation of the class that holds Country/Population facts.

2. A *main()* function that contains unit test cases for the *Country* class.

This evolved version of the *Country* class SHOULD INCLUDE explicit getter and setter features implemented using the *@property* decorator. When doing this part of the assignment, follow the approach that I demonstrated in the Part 3 tutorial video.

When this program is run directly (rather than having been imported), the console session should contain the unit testing output and should look like this:

```
Unit testing output follows...

Test 1:  Test Constructor
Passed

Test 2:  Test setting name
Passed

Test 3:  Test setting population
Passed

Test 4:  Test __str__ Method
Passed

Test 5:  Test Constructor being passed an empty name string
Passed

Test 6:  Test Constructor being passed a population value < 0
Passed

Test 7:  Test Constructor being passed a population value > 2
billion
Passed
```

Exercise 4

Create a program named *country_population_lists_using_evolved_class* by copying the *country_population_using_basic_class* program created in exercise 2. When coding and testing this program, follow the approach that I take in the Part 4 tutorial video. The changes that you make to the original program should include:

- Change the import statement such that the *Country* class is now imported from the *my_evolved_countries.py* module that you created in Exercise 3.
- No further changes should be required!

As the first part of your testing, use the *broken_country_pop_data.txt* file included in the starter files as your input data file. The program should be interrupted by an AttributeError exception and the output should show the appropriate error message instead of normal listing output.

As the second part of your testing, use the *country_pop_data.txt* file included in the starter files as your input data file. The program should run to normal completion and the output should show normal listings.

When testing, please check that the unit test output from the *my_evolved _countries.py* module is NOT printed in the test output for this program.

When this program is run, the console sessions should look like this:

```
Please enter input file name: broken_country_pop_data.txt
Traceback (most recent call last):

(many lines from Python traceback omitted here…)

AttributeError: The population attribute may not be set to a
value < 0 or > 2 billion.



Please enter input file name: country_pop_data.txt

COUNTRY NAME ORDER
Bangladesh       165435000
Brazil           209772000
China           1394860000
India           1338820000
Indonesia        265015300
Japan            126440000
Nigeria          193392517
Pakistan         202477000
Russia           146877088
USA              328077000
```

POPULATION ORDER

Japan	126440000
Russia	146877088
Bangladesh	165435000
Nigeria	193392517
Pakistan	202477000
Brazil	209772000
Indonesia	265015300
USA	328077000
India	1338820000
China	1394860000

DESCENDING POPULATION ORDER

China	1394860000
India	1338820000
USA	328077000
Indonesia	265015300
Brazil	209772000
Pakistan	202477000
Nigeria	193392517
Bangladesh	165435000
Russia	146877088
Japan	126440000

Tools
Use PyCharm to create and test both python programs.

Submission Method
Follow the process that I demonstrated in the tutorial video on submitting your work. This involves:

- Locating the properly named directory associated with your project in the file system.
- Compressing that directory into a single .ZIP file using a utility program.
- Submitting the properly named zip file to the submission activity for this assignment.

File and Directory Naming
Please name your Python program files as instructed in each exercise. Please use the following naming scheme for naming your project:

YourLastName_YourFirstName_exercises_severance_chapter_14

When you have compressed your project directory into a .ZIP file, it should have the following name structure:

YourLastName_YourFirstName_exercises_severance_chapter_14.zip

Due By
Please submit this assignment by the date and time shown in the Weekly Schedule.