Zelle 4e Chapter 9 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 Python Docstring that describes the intent of the program.
- Place your highest-level code in a function named *main*.
- 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.
- Choose names for your variables that are properly descriptive.
- Define CONSTANT_VALUES and use them in place of *magic numbers*.
- Always use f-strings for string interpolation and number formatting.
- When processing items from Python lists and tuples, unpack the values into variables with meaningful variable names to avoid using indexed expressions in your code.
- Remember that your program should behave reasonably when it is not given any input.
- Model your solution after the code that I demonstrate in the tutorial videos.
- Make sure that your test input/output matches the sample provided.
- Remember to test your program thoroughly before submitting your work.
- Make sure that your test input/output matches the sample provided.
- All functions that are not *main()* should have descriptive, action-oriented names.
- All functions should be of reasonable size.
- All functions should have high *cohesion*, and low *coupling*.
- 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).
- Use of the *break* statement is allowed.
- Use of the *continue* statement is forbidden.

Exercise 1 (Regular)
Create a program named *distribute_race_ribbons_with_dictionary.py*. It should modeled after the program that I demonstrated in the tutorial *(lookup_region_name_with_dictionary.py*). Your program should be different in the following respects:

1. Your program will prompt the user for the place number in which the runner finished, and it will respond with the name of the ribbon to be awarded.

The following table indicates which ribbon the participant should receive based up the place number in which they finished.

Place	**Ribbon**
1	Blue
2	Red
3	Orange
4	Gold
5	Green
6	Purple
>6	White

If the user enters a place number that is less than 1, then the program should display an error message in place of a ribbon name.

If the user enters a non-integer, then the program should display an error message.

In any case, the program should continue re-prompting the user for input until the user enters the empty string (<Enter>).

Your unit testing for this program should be manual unit testing.

When running a test where the user provides no input, you should expect the following input/output on your console:

```
Please enter place finished (1, 2, 3...):

Thanks for playing.
```

When running a test where the user provides bad integer input, you should expect the following input/output on your console:

```
Please enter place finished (1, 2, 3...): hi mom
An integer value greater than zero was expected. You entered hi
mom
Please enter place finished (1, 2, 3...): 7
Ribbon Awarded: White

Please enter place finished (1, 2, 3...):

Thanks for playing.
```

When running a test where a user provides an invalid place number, you should expect the following input/output on your console:

```
Please enter place finished (1, 2, 3...): 0
An integer value greater than zero was expected. You entered 0
Please enter place finished (1, 2, 3...): 4
Ribbon Awarded: Gold

Please enter place finished (1, 2, 3...):

Thanks for playing.
```

When running a test with more typical input, you should expect the following input/output on your console:

```
Please enter place finished (1, 2, 3...): 1
Ribbon Awarded: Blue

Please enter place finished (1, 2, 3...): 2
Ribbon Awarded: Red

Please enter place finished (1, 2, 3...): 3
Ribbon Awarded: Orange

Please enter place finished (1, 2, 3...): 4
Ribbon Awarded: Gold

Please enter place finished (1, 2, 3...): 5
Ribbon Awarded: Green

Please enter place finished (1, 2, 3...): 6
Ribbon Awarded: Purple

Please enter place finished (1, 2, 3...): 7
```

```
Ribbon Awarded: White

Please enter place finished (1, 2, 3...): 99
Ribbon Awarded: White

Please enter place finished (1, 2, 3...):

Thanks for playing.
```

Exercise 2 (Regular)
From the starter files provided for this assignment, copy the program named *search_for_forbidden_passwords.py*. The starter file contains code that will fabricate the set of passwords that you need in your program.

Model your code after the program that I showed in the tutorial named *search_for_special_zipcodes.py*.

Your program should be different in the following respects:

1. It should be processing passwords rather than zipcodes.

2. The set of forbidden passwords will be created by the function provided in the starter code named *fabricate_a_set_of_forbidden_passwords().*

Your unit testing for this program should be manual unit testing.

When running a test where the user provides no input, you should expect the following input/output on your console:

```
Search for forbidden passwords in a set...

Please enter password (<Enter> to stop):

Thanks for playing.
```

When running a test with more typical input, you should expect the following input/output on your console:

```
Search for forbidden passwords in a set...

Please enter password (<Enter> to stop): superman
superman is in the forbidden password set.

Please enter password (<Enter> to stop): mustang
mustang is in the forbidden password set.

Please enter password (<Enter> to stop): &goodPassword98^
&goodPassword98^ is NOT in the forbidden password set.

Please enter password (<Enter> to stop): querty
querty is NOT in the forbidden password set.

Please enter password (<Enter> to stop):

Thanks for playing.
```

Exercise 3 (Regular)
From the starter files provided for this assignment, copy the program named *create_population_density_reports.py*. . The starter file contains code that will fabricate the set of Country data objects that you need in your program.

Model your code after the program that I showed in the tutorial named *create_state_area_reports.py*.

Your program should be different in the following respects:

1. It should create reports about population density.

2. It should use the provided Country class as the data holder class.

3. The list of Country data objects will be created by the function provided in the starter code named *fabricate_countries_list().*

Your unit testing for this program should be manual unit testing.

When running a test, you should expect the following output on your console:

```
                         BY COUNTRY NAME

Country                  Population           Area        Density
                                            (SQMI)        (/SQMI)
India                 1,344,098,517      1,269,211          1,059
Japan                   126,320,000        145,925            866
Nigeria                 195,875,237        356,669            549
Pakistan                203,841,217        310,403            657
South Korea              51,635,256         38,691          1,335
United Kingdom           66,040,229         93,788            704


        BY DESCENDING POPULATION DENSITY PER SQUARE MILE

Country                  Population           Area        Density
                                            (SQMI)        (/SQMI)
South Korea              51,635,256         38,691          1,335
India                 1,344,098,517      1,269,211          1,059
Japan                   126,320,000        145,925            866
United Kingdom           66,040,229         93,788            704
Pakistan                203,841,217        310,403            657
Nigeria                 195,875,237        356,669            549
```

Exercise 4 (Challenge)
Create a program named *create_population_density_reports_with_lambda.py*. This program should be modeled after your solution to Exercise 3 (*create_population_density_reports.py*). This program should be different in the following respects:

1. Instead of using conventional Python functions to specify the sort keys, this program should use Python lambdas.
   See https://realpython.com/python-lambda/.

Your unit testing for this program should be manual unit testing.

The testing for this program should be the same as the program in Exercise 3. Please refer to those instructions for expected output.

Tools
Use PyCharm to create and test all 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 PyCharm project:

```
surname_givenname_exercises_zelle_4e_chapter_09
```

If this were my own project, I would name my PyCharm project as follows:

```
trainor_kevin_exercises_zelle_4e_chapter_09
```

Use a zip utility to create one zip file that contain the PyCharm project directory. The zip file should be named according to the following scheme:

```
surname_givenname_exercises_zelle_4e_chapter_09.zip
```

If this were my own project, I would name the zip file as follows:

```
trainor_kevin_exercises_zelle_4e_chapter_09.zip
```

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

Last Revised
2025-10-03