Zelle 4e Chapter 10 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.
- Always use the *with* statement as a context manager for files to assure that all files before the conclusion of the program.
- Remember that your program should behave reasonably when it is not given any input. This might be the result of the user pressing enter at a console prompt. Or, it might be the result of the user providing an input file that is empty.
- Model your solution after the code that I demonstrate in the tutorial videos.
- Make sure that your test input/output matches the sample provided.
- Create a sub-directory named *data* within your PyCharm project to hold data files.
- Remember to submit all data files with your PyCharm project – including the files that were provided as starter files to this assignment.
- 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 but not encouraged.
- Use of the *continue* statement is forbidden.

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

1. Instead of counting and reporting vowels, your program should count and report on numeric digits (0 – 9).

2. Your program should use the input file *sentences_with_numbers.txt* that is provided with the starter files.

3. Remember to confirm that your program also works properly with the empty file test file (*empty_file.txt*).

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

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

```
Please enter the name of the file to be analyzed:
data/empty_file.txt

0 lines were analyzed from the file: data/empty_file.txt
```

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

```
Please enter the name of the file to be analyzed:
data/sentences_with_numbers.txt

Analyzing: There are 12 inches in 1 foot.
"There" contains 0 numeric characters.
"are" contains 0 numeric characters.
"12" contains 2 numeric characters.
"inches" contains 0 numeric characters.
"in" contains 0 numeric characters.
"1" contains 1 numeric characters.
"foot." contains 0 numeric characters.
7 words were analyzed.

Analyzing: The United States is comprised of 50 states.
"The" contains 0 numeric characters.
"United" contains 0 numeric characters.
"States" contains 0 numeric characters.
```

```
"is" contains 0 numeric characters.
"comprised" contains 0 numeric characters.
"of" contains 0 numeric characters.
"50" contains 2 numeric characters.
"states." contains 0 numeric characters.
8 words were analyzed.

Analyzing:
0 words were analyzed.

Analyzing: 1, 2, 3, 4. I am the giant that you are looking for.
"1," contains 1 numeric characters.
"2," contains 1 numeric characters.
"3," contains 1 numeric characters.
"4." contains 1 numeric characters.
"I" contains 0 numeric characters.
"am" contains 0 numeric characters.
"the" contains 0 numeric characters.
"giant" contains 0 numeric characters.
"that" contains 0 numeric characters.
"you" contains 0 numeric characters.
"are" contains 0 numeric characters.
"looking" contains 0 numeric characters.
"for." contains 0 numeric characters.
13 words were analyzed.

Analyzing: 50 men, and 75 horses, took 10 wagons over 1 big
mountain.
"50" contains 2 numeric characters.
"men," contains 0 numeric characters.
"and" contains 0 numeric characters.
"75" contains 2 numeric characters.
"horses," contains 0 numeric characters.
"took" contains 0 numeric characters.
"10" contains 2 numeric characters.
"wagons" contains 0 numeric characters.
"over" contains 0 numeric characters.
"1" contains 1 numeric characters.
"big" contains 0 numeric characters.
"mountain." contains 0 numeric characters.
12 words were analyzed.

5 lines were analyzed from the file:
data/sentences_with_numbers.txt
```

Exercise 2 (Regular)
Create a program named *file_based_float_adding_machine.py*. It should modeled after the program that I demonstrated in the tutorial *(file_based_int_adding_machine.py)*. Your program should be different in the following respects:

1. Instead of processing only int values, your program should be able to read and accumulate either int or float values. It should accomplish this by storing values as floats.

2. Your program should format the printed accumulated total with 3 decimal digits.

4. Your program should use the input file *float_entries.txt* that is provided with the starter files.

5. Remember to confirm that your program also works properly with the empty file test file (*empty_file.txt*).

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

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

```
File-based float adding machine

Please enter the name of the input file: data/empty_file.txt

ENTRIES:

0 lines were read from file: data/empty_file.txt

0 entries were processed.

The accumulated total of the entries was 0.000
```

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

```
File-based float adding machine

Please enter the name of the input file: data/float_entries.txt

ENTRIES:
1.25
2.0
3.0
4.5
9.0
0.0
2.22
3.35
4.2
15.0

7 lines were read from file: data/float_entries.txt

10 entries were processed.

The accumulated total of the entries was 44.520
```

Exercise 3 (Regular)
Create a program named *generate_usernames.py*. It should modeled after the program that I demonstrated in the tutorial *(generate_noble_names.py*). Your program should be different in the following respects:

1. Instead of generating a file with potentially humorous names, your program will generate a file with username assignments.

2. Usernames should be constructed from the first letter of the first name concatenated with the first seven letters of the last name.

3. Each line in the output file should have the following format:

    a. First name
    b. One space
    c. Last name
    d. A comma (no space)
    e. Username

6. Your program should use the input file *person_names.txt* that is provided with the starter files.

7. Remember to confirm that your program also works properly with the empty file test file (*empty_file.txt*).

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

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

```
Please enter the name of the input file containing names:
data/empty_file.txt
Please enter the name of the output file that will contain
username assignments: data/username_assignments.txt

0 lines were processed from input file: data/empty_file.txt

Created username assignments file as:
data/username_assignments.txt
```

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

```
Please enter the name of the input file containing names:
data/person_names.txt
Please enter the name of the output file that will contain
username assignments: data/username_assignments.txt

125 lines were processed from input file: data/person_names.txt

Created username assignments file as:
data/username_assignments.txt
```

When running a test using *person_names.txt* as the input file, you should expect the following to be the first 5 lines of the output file:

```
Elizabeth Wilson,ewilson
Victor Lewis,vlewis
Christopher Paterson,cpaterso
Nicola Powell,npowell
Ava White,awhite
```

Exercise 4 (Challenge)
Create a program named *analyze_slot_machine_tries_ignoring_duplicates.py*. It should modeled after the program that I demonstrated in the tutorial *(analyze_slot_machine_tries.py*). Your program should be different in the following respects:

1. When counting colors on each line of the input file, your program should ignore duplicates. So, if the color values on a line were:

   ```
   Orange Purple Green Purple Purple
   ```

   Your program should count these data as though they were:

   ```
   Orange Purple Green
   ```

2. Your program should use the input file *slot_values.txt* that is provided with the starter files.

3. Remember to confirm that your program also works properly with the empty file test file (*empty_file.txt*).

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

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

```
Please enter the input filename: data/empty_file.txt

COLOR       COUNT
```

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

```
Please enter the input filename: data/slot_values.txt

COLOR       COUNT
Blue        2,990
Green       2,983
Orange      3,024
Purple      3,015
Red         2,959
Yellow      3,011
```

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_10
```

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

```
trainor_kevin_exercises_zelle_4e_chapter_10
```

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_10.zip
```

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

```
trainor_kevin_exercises_zelle_4e_chapter_10.zip
```

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

Last Revised
2025-10-09