Zelle 3e Chapter 3 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.
- Model your solution after the code that I demonstrate in the tutorial videos.
- Remember to test your program thoroughly before submitting your work.
- Make sure that your test input/output matches the sample provided.

Exercise 1 (Regular)
Create a program named *integer_division_2.py*. It should meet the following requirements:

1. Prompt the user for two integer values.

2. Calculate the quotient using the // operator.

3. Calculate the remainder using the % operator.

4. Print the results in a properly formatted message.

For a refresher on the terms used in division, please consult http://www.math-only-math.com/terms-used-in-division.html .

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

```
Please enter an integer for the dividend: 100
Please enter an integer for the divisor: 99

The quotient is 1 and the remainder is 1.
```

Exercise 2 (Regular)
Create a program named *remainder_accumulator.py.* It should meet the following requirements:

1. Prompt the user for the quantity of integer pairs that they wish to enter.

2. For each pair of integers entered, calculate the remainder resulting from dividing the first integer by the second integer.

3. As you are processing the integers, keep an accumulated sum of the remainders.

4. Finally, print the accumulated sum of the remainders in a properly formatted message.

For a refresher on the terms used in division, please consult http://www.math-only-math.com/terms-used-in-division.html .

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

```
Please enter the quantity of integer pairs that you wish to enter: 5

Now collecting integer pair 1 of 5...
Please enter an integer for the dividend: 9
Please enter an integer for the divisor: 7
The remainder is 2.

Now collecting integer pair 2 of 5...
Please enter an integer for the dividend: 11
Please enter an integer for the divisor: 9
The remainder is 2.

Now collecting integer pair 3 of 5...
Please enter an integer for the dividend: 12
Please enter an integer for the divisor: 10
The remainder is 2.

Now collecting integer pair 4 of 5...
Please enter an integer for the dividend: 8
Please enter an integer for the divisor: 6
The remainder is 2.

Now collecting integer pair 5 of 5...
Please enter an integer for the dividend: 13
Please enter an integer for the divisor: 11
The remainder is 2.

The accumulated sum of the remainders is 10.
```

Exercise 3 (Regular)
Create a program named *mixed_division_2_accumulator.py.* It should meet the following requirements:

1. Prompt the user for the quantity of number pairs that they wish to enter.

2. For each number pair: Prompt the user to enter a float for the dividend. Then, prompt the user to enter an int value for the divisor.

3. Calculate the quotient using the / operator.

4. Print the quotient in a properly formatted message with 3 decimal places. To control the decimal places, use the number formatting feature of f-strings.

5. As you are processing the number pairs, keep an accumulated sum of the quotients.

6. Finally, print the accumulated sum of the quotients in a properly formatted message with 3 decimal places. To control the decimal places, use the *round()* function.

For a refresher on the terms used in division, please consult http://www.math-only-math.com/terms-used-in-division.html .

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

```
Please enter the quantity of number pairs that you wish to enter: 3

Now collecting number pair 1 of 3...
Please enter a float value for the dividend: 3.5
Please enter an integer value for the divisor: 4
This quotient is 0.875

Now collecting number pair 2 of 3...
Please enter a float value for the dividend: 6.66
Please enter an integer value for the divisor: 7
This quotient is 0.951

Now collecting number pair 3 of 3...
Please enter a float value for the dividend: 8.88
Please enter an integer value for the divisor: 9
This quotient is 0.987

The accumulated sum of the quotients is 2.813
```

Exercise 4 (Challenge)
Create a program named *decimal_accumulator.py.* It should meet the following requirements:

1. Prompt the user for the quantity of decimal values that they wish to enter.

2. For each decimal value provided by the user, store the value using the Python Decimal type. For guidance on using the Python Decimal type, see documentation on the Python decimal module in the Python documentation at: https://docs.python.org/3/library/decimal.html .

3. As you are processing the decimal values, keep an accumulated sum of those values as a Python Decimal type.

4. Finally, print the accumulated sum of the decimal values in a properly formatted message.

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

```
Please enter the quantity of decimal values that you wish to enter: 5

Please enter decimal value 1 of 5: 1
Please enter decimal value 2 of 5: 2.2
Please enter decimal value 3 of 5: 3.33
Please enter decimal value 4 of 5: 4.444
Please enter decimal value 5 of 5: 5.5555

The accumulated sum of the decimal values is 16.5295
```

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_3e_chapter_03
```

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

```
trainor_kevin_exercises_zelle_3e_chapter_03
```

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_3e_chapter_03.zip
```

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

```
trainor_kevin_exercises_zelle_3e_chapter_03.zip
```

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

Last Revised
2022-05-22