IS430 – Instructions for the Final Project

General Instructions

In the Final Project, I will be expecting you to follow all of the good programming practices that we have adopted in the course. Here is a quick summary of good practices that we have covered:

- 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.
- Close 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 a 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.
- Regular expression patterns should be expressed as Python *raw strings*
- Your finished code must be refactored to meet all good program design practices covered in this course.
- Custom Python classes should be created using Python Dataclasses and follow all practices demonstrated in our course.
- Python programs that will be called from a Jupyter notebook, should not run when imported. They should only run when they have been explicitly called.

- Python programs that will be called from a Jupyter notebook should not prompt the user for input. Instead, parameters should be passed explicitly to the Python program to set all configuration values.
- Python programs that will be called from a Jupyter notebook should be unit tested using PyCharm. Unit testing scenarios should not be repeated in the Jupyter notebook. The Jupyter notebook code should implement the production workflow.
- Jupyter notebooks should not include extensive Python code. The Python code included in the notebook should be just enough to set configuration parameters, import Python functions that have been written and tested in PyCharm, and call those functions.

Important Note: Do NOT use the Python *numpy* or *pandas* packages
It is possible that you are already aware of the Python *numpy* and *pandas* packages. In this course, we do NOT cover either of these packages. Instead, we are exploring how to create program solutions without them. So, please do NOT include *numpy* or *pandas* in your solutions to these exercises. When grading this assignment, we will be making point deductions for solutions that use either of these packages.

Overview
The Final Project is a special coding assignment. It is longer than the typical weekly coding assignments and the exercises are more interrelated. Nevertheless, you should recognize this assignment as being similar to a weekly coding assignment.

There is a tutorial video provided for this assignment:

- A typical tutorial for Exercise 1 in which I show the code for the exercise with the expectation that you will follow along on your computer.

Although there are no tutorial videos provided for the other exercises, the instructions for these exercises identify exercises from the weekly coding assignments that are similar. You should feel free to consult the solutions and video tutorials for those exercises as a reference.

The code for one of the exercises is provided in the starter file. Your responsibility will be to add the program to your project, run it successfully, and integrate it into the Jupyter Notebook (see Exercise 7).

In this project you will select, summarize, and report on expense data from the 3 operating divisions of an international company. These divisions include U.S., U.K., and France.

The workflow has three major steps:

- Select and Summarize
- Join Country Summaries
- Create Company-Wide Expense Summary Reports

In the *Select and Summarize* step, there is a separate program that must be run for each country's operations (see Exercises 1, 2, and 3). The raw expense data from each country's

operations are contained in a single file with a descriptive name. In the first step of the workflow, you will run individual *select and summarize* programs for each of these 3 files. In this step, expense records are selected based upon the year and expense category. The target year for this project is *2020*. The target expense category for this project is *Fuel*.

In the *Join Country Summaries* step, you will join the data from the 3 output files generated in the previous step. One summary data file is created that joins the expense data for U.K., U.S., and France operations (see Exercise 4).

In the *Create Company-Wide Expense Summary Reports* step, you will generate company-wide expense summary reports that combine expense data from U.S., U.K., and France operations. You have the option of creating two versions of these reports. The first is the regular version produced in Exercise 6. The second is the challenge version produced in Exercise 8. The programs for Exercises 6 and 8 require a custom Python dataclass that is created in Exercise 5.

Finally, in Exercise 7, you will build a Jupiter Notebook that provides documentation for the Final Project and a means to run all of it parts. This means that all of the programs that you create for the Final Project must follow the conventions that we have used for Python programs to be called from a Jupyter Notebook.

PLEASE NOTE: The exercises in this Final Project were inspired by an exercise in one of our weekly coding assignments. Despite this similarity, there are many differences. So, please read the instructions carefully and don't be distracted by assuming that these exercises are more similar to the exercise that inspired them than they actually are.

Starter Files

I have provided starter files for this project in the following ZIP file:

- `starter_files_for_final_project.zip`

Exercise 1 (Regular)
Create a program named *us_select_and_summarize.py*. This is the same program that I demonstrate in the tutorial video. You should follow along to create this program in your project.

PLEASE NOTE: In this exercise, you will be expected to unit test your program using PyCharm. However, you should be aware that in Exercise 7, you will be expected to call this same program from a Jupyter notebook. With that in mind, you should take care to design this program such that it is easily called from a Jupyter notebook. If you have not done so already, I suggest that you play my tutorial video: *Python Program Design for Jupyter Notebooks (https://www.youtube.com/watch?v=RHwLMKZGVWY)*.

Two alternative input files have been provided in the starter files for this program:

- `empty_file.txt`
- `us_expense_records.txt`

This program will produce one output file:

- `us_summarized_target_expenses.txt`

Please note that processing the U.S. expense data does not require currency conversion because the amounts are already stated in U.S. Dollars (USD).

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

```
0 lines were read from empty_file.txt.
0 qualifying expense records were accumulated.


U.S. OPERATIONS
SUMMARY OF FUEL EXPENSES FOR 2020

Month    Amount (USD)
    1            0.00
    2            0.00
    3            0.00
    4            0.00
    5            0.00
    6            0.00
    7            0.00
    8            0.00
    9            0.00
   10            0.00
   11            0.00
```

```
   12            0.00


Summary file was created as us_summarized_target_expenses.txt.
```

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

```
20,001 lines were read from us_expense_records.txt.
749 qualifying expense records were accumulated.


U.S. OPERATIONS
SUMMARY OF FUEL EXPENSES FOR 2020

Month    Amount (USD)
    1       72,742.79
    2       45,758.93
    3       64,399.68
    4       58,308.71
    5       61,662.60
    6       70,589.85
    7       59,945.88
    8       69,182.55
    9       73,945.30
   10       69,162.65
   11       62,375.22
   12       83,871.87


Summary file was created as us_summarized_target_expenses.txt.
```

When running a test with the populated input file, you should expect the following contents in the output file:

```
Country,Expense Category,Month,Year,Amount (USD)
US,Fuel,1,2020,72742.79
US,Fuel,2,2020,45758.93
US,Fuel,3,2020,64399.68
US,Fuel,4,2020,58308.71
US,Fuel,5,2020,61662.60
US,Fuel,6,2020,70589.85
US,Fuel,7,2020,59945.88
US,Fuel,8,2020,69182.55
US,Fuel,9,2020,73945.30
US,Fuel,10,2020,69162.65
US,Fuel,11,2020,62375.22
```

```
US,Fuel,12,2020,83871.87
```

Exercise 2 (Regular)
Create a program named *uk_select_and_summarize.py*. Model this program after the program created in Exercise 1 (*us_select_and_summarize.py).* Your program should be different in the following respects:

1. It will process expense records from U.K. operations rather than U.S. operations.

2. It will require the recognition of a European format date (DD/MM/YYYY).

3. It will require the conversion of amounts from U.K. pounds to U.S. dollars using the following conversion factor:

    a. POUNDS_TO_DOLLARS_CONVERSION_FACTOR = 1.3114

PLEASE NOTE: In this exercise, you will be expected to unit test your program using PyCharm. However, you should be aware that in Exercise 7, you will be expected to call this same program from a Jupyter notebook. With that in mind, you should take care to design this program such that it is easily called from a Jupyter notebook. If you have not done so already, I suggest that you play my tutorial video: *Python Program Design for Jupyter Notebooks (https://www.youtube.com/watch?v=RHwLMKZGVWY).*

Two alternative input files have been provided in the starter files for this program:

- `empty_file.txt`
- `uk_expense_records.txt`

This program will produce one output file:

- `uk_summarized_target_expenses.txt`

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

```
0 lines were read from empty_file.txt.
0 qualifying expense records were accumulated.


U.K. OPERATIONS
SUMMARY OF FUEL EXPENSES FOR 2020

Month    Amount (USD)
    1            0.00
    2            0.00
    3            0.00
    4            0.00
    5            0.00
    6            0.00
```

```
    7            0.00
    8            0.00
    9            0.00
   10            0.00
   11            0.00
   12            0.00


Summary file was created as uk_summarized_target_expenses.txt.
```

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

```
6,667 lines were read from uk_expense_records.txt.
217 qualifying expense records were accumulated.


U.K. OPERATIONS
SUMMARY OF FUEL EXPENSES FOR 2020

Month    Amount (USD)
    1       16,083.88
    2       24,874.43
    3       19,376.45
    4       19,591.66
    5       20,596.45
    6       17,399.42
    7       22,621.51
    8       23,754.80
    9       11,856.29
   10       12,775.92
   11       22,577.32
   12       17,280.50


Summary file was created as uk_summarized_target_expenses.txt.
```

When running a test with the populated input file, you should expect the following contents in the output file:

```
Country,Expense Category,Month,Year,Amount (USD)
UK,Fuel,1,2020,16083.88
UK,Fuel,2,2020,24874.43
UK,Fuel,3,2020,19376.45
UK,Fuel,4,2020,19591.66
UK,Fuel,5,2020,20596.45
UK,Fuel,6,2020,17399.42
UK,Fuel,7,2020,22621.51
UK,Fuel,8,2020,23754.80
UK,Fuel,9,2020,11856.29
UK,Fuel,10,2020,12775.92
UK,Fuel,11,2020,22577.32
UK,Fuel,12,2020,17280.50
```

Exercise 3 (Regular)
Create a program named *fr_select_and_summarize.py*. Model this program after the program created in Exercise 2 (*uk_select_and_summarize.py).* Your program should be different in the following respects:

1. It will process expense records from France operations rather than U.K. operations.

2. Note that it will also require the recognition of a European format date (DD/MM/YYYY).

3. It will require the conversion of amounts from Euros to U.S. dollars using the following conversion factor:

    a. EUROS_TO_DOLLARS_CONVERSION_FACTOR = 1.1043

PLEASE NOTE: In this exercise, you will be expected to unit test your program using PyCharm. However, you should be aware that in Exercise 7, you will be expected to call this same program from a Jupyter notebook. With that in mind, you should take care to design this program such that it is easily called from a Jupyter notebook. If you have not done so already, I suggest that you play my tutorial video: *Python Program Design for Jupyter Notebooks (https://www.youtube.com/watch?v=RHwLMKZGVWY).*

Two alternative input files have been provided in the starter files for this program:

- `empty_file.txt`
- `fr_expense_records.txt`

This program will produce one output file:

- `fr_summarized_target_expenses.txt`

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

```
0 lines were read from empty_file.txt.
0 qualifying expense records were accumulated.


FRENCH OPERATIONS
SUMMARY OF FUEL EXPENSES FOR 2020

Month    Amount (USD)
    1            0.00
    2            0.00
    3            0.00
    4            0.00
```

```
    5            0.00
    6            0.00
    7            0.00
    8            0.00
    9            0.00
   10            0.00
   11            0.00
   12            0.00


Summary file was created as fr_summarized_target_expenses.txt.
```

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

```
5,001 lines were read from fr_expense_records.txt.
191 qualifying expense records were accumulated.


FRENCH OPERATIONS
SUMMARY OF FUEL EXPENSES FOR 2020

Month    Amount (USD)
    1       14,365.95
    2       14,354.80
    3       16,059.11
    4       17,538.96
    5       12,199.59
    6       14,119.40
    7       24,520.53
    8       13,556.39
    9       16,615.88
   10       16,768.71
   11       17,964.19
   12       18,998.29


Summary file was created as fr_summarized_target_expenses.txt.
```

When running a test with the populated input file, you should expect the following contents in the output file:

```
Country,Expense Category,Month,Year,Amount (USD)
FR,Fuel,1,2020,14365.95
FR,Fuel,2,2020,14354.80
FR,Fuel,3,2020,16059.11
FR,Fuel,4,2020,17538.96
FR,Fuel,5,2020,12199.59
FR,Fuel,6,2020,14119.40
FR,Fuel,7,2020,24520.53
FR,Fuel,8,2020,13556.39
FR,Fuel,9,2020,16615.88
FR,Fuel,10,2020,16768.71
FR,Fuel,11,2020,17964.19
FR,Fuel,12,2020,18998.29
```

Exercise 4 (Regular)
The program for this exercise has been provided in the starter files:

- `join_selected_summarized_expenses.py`

This program reads the summarized files created in Exercises 1, 2, and 3. It joins these files to create a summarized file that includes amounts for operations in U.S., U.K., and France.

This program has 3 input files that were created in the preceding exercises:

- `us_summarized_target_expenses.txt`
- `uk_summarized_target_expenses.txt`
- `fr_summarized_target_expenses.txt`

This program creates a short report on the console and one output file:

- `joined_summarized_target_expenses.txt`

Despite the fact that this program has been provided to you, you are expected to unit test it. This includes running the two test scenarios described below.

When running a test with one of the 3 input files set to *empty_file.txt*, you should expect the following input/output on your console:

```
RuntimeError: One or more input files are empty.
```

When running a test with all 3 populated input files, you should expect the following output on your console:

```
12 joined expense summary records were written to
joined_summarized_target_expenses.txt.
```

When running a test with 3 populated input files, you should expect the following contents in the output file:

```
Expense Category,Month,Year,US Amount (USD), UK Amount (USD), FR Amount
(USD)
Fuel,1,2020,72742.79, 16083.88, 14365.95
Fuel,2,2020,45758.93, 24874.43, 14354.80
Fuel,3,2020,64399.68, 19376.45, 16059.11
Fuel,4,2020,58308.71, 19591.66, 17538.96
Fuel,5,2020,61662.60, 20596.45, 12199.59
Fuel,6,2020,70589.85, 17399.42, 14119.40
Fuel,7,2020,59945.88, 22621.51, 24520.53
Fuel,8,2020,69182.55, 23754.80, 13556.39
Fuel,9,2020,73945.30, 11856.29, 16615.88
Fuel,10,2020,69162.65, 12775.92, 16768.71
Fuel,11,2020,62375.22, 22577.32, 17964.19
Fuel,12,2020,83871.87, 17280.50, 18998.29
```

Exercise 5 (Regular)
Create a program named *my_expense_summary.py*. It should be modeled after the solution to Exercise 1 of the Severance Chapter 14 Coding Assignment. Your program should be different in the following respects:

1. Your program will implement the *ExpenseSummary* class that holds data facts regarding expense amounts for U.S., U.K., and France operations.

2. The *ExpenseSummary* class should implement the following instance variables:

   - `category: str`
   - `month: int`
   - `year: int`
   - `us_amount: float`
   - `uk_amount: float`
   - `fr_amount: float`

3. You will also need to implement the following method:

   a. *calculate_total_amount*() returns the sum of us_amount, uk_amount, and fr_amount as a float.

4. Unit testing code should be placed in the *main()* function and should follow the approach used in Exercise 1 of the Severance Chapter 14 Coding Assignment. .

When running the unit tests, you should expect the following output on your console:

```
Unit testing output follows...

Test Case #1: Test constructor
Passed
Passed
Passed
Passed
Passed
Passed

Test Case #2: Test calculate_total_amount
Passed
```

Exercise 6 (Regular)
Create a program named *create_companywide_expense_summary_report.py*. It should be modeled after the solution to Exercise 2 in the Severance Chapter 14 Coding Assignment. Your program should be different in the following respects:

1. Your program will create a report of *ExpenseSummary* data facts in two different sort orders:

    a. By Month
    b. By Descending Total Amount

2. Your program should accumulate totals for each row of data facts and print a total line at the bottom of each report.

    Hint: Once the list of *ExpenseSummary* data facts have been created, these facts never change. So, regardless of how you might sort the *ExpenseSummary* data facts in the list, the annual total amounts never change. The preferred way to calculate annual totals for the report total lines is to calculate them once at the beginning of the program and pass these totals as parameters to the function that creates the report.

3. Your program should give expected results when run with the following input files provided as starter files:

    a. empty_file.txt
    b. joined_summarized_target_expenses.txt

4. The importing of the *my_expense_summary.py* module into your program should NOT cause the unit test code in that program to be executed.

PLEASE NOTE: In this exercise, you will be expected to unit test your program using PyCharm. However, you should be aware that in Exercise 7, you will be expected to call this same program from a Jupyter notebook. With that in mind, you should take care to design this program such that it is easily called from a Jupyter notebook. If you have not done so already, I suggest that you play my tutorial video: *Python Program Design for Jupyter Notebooks (https://www.youtube.com/watch?v=RHwLMKZGVWY)*.

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

```
                    COMBINED OPERATIONS
                SUMMARY OF  EXPENSE AMOUNTS FOR 0
                            BY MONTH

Month      US Amount      UK Amount      FR Amount   Total Amount
               (USD)          (USD)          (USD)          (USD)
Total           0.00           0.00           0.00           0.00


                    COMBINED OPERATIONS
                SUMMARY OF  EXPENSE AMOUNTS FOR 0
                   BY DESCENDING TOTAL AMOUNT

Month      US Amount      UK Amount      FR Amount   Total Amount
               (USD)          (USD)          (USD)          (USD)
Total           0.00           0.00           0.00           0.00
```

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

COMBINED OPERATIONS
SUMMARY OF FUEL EXPENSE AMOUNTS FOR 2020
BY MONTH

Month	US Amount (USD)	UK Amount (USD)	FR Amount (USD)	Total Amount (USD)
1	72,742.79	16,083.88	14,365.95	103,192.62
2	45,758.93	24,874.43	14,354.80	84,988.16
3	64,399.68	19,376.45	16,059.11	99,835.24
4	58,308.71	19,591.66	17,538.96	95,439.33
5	61,662.60	20,596.45	12,199.59	94,458.64
6	70,589.85	17,399.42	14,119.40	102,108.67
7	59,945.88	22,621.51	24,520.53	107,087.92
8	69,182.55	23,754.80	13,556.39	106,493.74
9	73,945.30	11,856.29	16,615.88	102,417.47
10	69,162.65	12,775.92	16,768.71	98,707.28
11	62,375.22	22,577.32	17,964.19	102,916.73
12	83,871.87	17,280.50	18,998.29	120,150.66
Total	791,946.03	228,788.63	197,061.80	1,217,796.46

COMBINED OPERATIONS
SUMMARY OF FUEL EXPENSE AMOUNTS FOR 2020
BY DESCENDING TOTAL AMOUNT

Month	US Amount (USD)	UK Amount (USD)	FR Amount (USD)	Total Amount (USD)
12	83,871.87	17,280.50	18,998.29	120,150.66
7	59,945.88	22,621.51	24,520.53	107,087.92
8	69,182.55	23,754.80	13,556.39	106,493.74
1	72,742.79	16,083.88	14,365.95	103,192.62
11	62,375.22	22,577.32	17,964.19	102,916.73
9	73,945.30	11,856.29	16,615.88	102,417.47
6	70,589.85	17,399.42	14,119.40	102,108.67
3	64,399.68	19,376.45	16,059.11	99,835.24
10	69,162.65	12,775.92	16,768.71	98,707.28
4	58,308.71	19,591.66	17,538.96	95,439.33
5	61,662.60	20,596.45	12,199.59	94,458.64
2	45,758.93	24,874.43	14,354.80	84,988.16
Total	791,946.03	228,788.63	197,061.80	1,217,796.46

Exercise 7 (Regular)
Create a Jupyter Notebook in your PyCharm project named *final_project_notebook.ipynb*. It should be modeled after the solution to the Jupyter Notebook Assignment.

This notebook should run the workflow that includes all of the exercises in this Final Project. If you will be doing the Exercise 8 Challenge, that part of the workflow should also be included.

This notebook should include complete documentation for the Final Project workflow. This includes the following sections:

1. Title
2. Requirements
3. Overview
4. Select and Summarize
    a. U.S.
    b. U.K.
    c. France
5. Join Country Summaries
6. Create Company-Wide Expense Summary Report
    a. Regular Version (From Exercise 6)
    b. Challenge Version (From Exercise 8)

When writing the documentation for sections like *Overview*, feel free to use text from these instructions.

Your work on this exercise should conform to all good practices that were required for the Jupyter Notebook Assignment.

PLEASE NOTE: When testing this Jupyter notebook, you need to take care that none of the programs that you are calling from the notebook run when imported. They should only run when they are explicitly called by code within the Jupyter notebook. If you have not done so already, I suggest that you play my tutorial video *Python Program Design for Jupyter Notebooks (https://www.youtube.com/watch?v=RHwLMKZGVWY)*.

Exercise 8 (Challenge)

Create a program named *create_companywide_expense_summary_report_challenge.py*. Begin by copying the program that you created in Exercise 6 (*create_companywide_expense_summary_report.py*) and renaming it. Your program should be different in the following respects:

1. Your program will create a report of *ExpenseSummary* data facts in two different sort orders:

    a. By Month
    b. By Descending Total Amount

2. Your program will create reports in two different formats:

    a. Amount Reports
    b. Percentage Reports

3. Your program should accumulate totals for each row of data facts and print a total line at the bottom of each report. Amount reports will show amount totals. Percentage reports will show percent totals.

4. Your program should give expected results when run with the following input files provided as starter files:

    a. empty_file.txt
    b. joined_summarized_target_expenses.txt

5. The importing of the *my_expense_summary.py* module into your program should NOT cause the unit test code in that program to be executed.

Please remember to revise the Jupyter notebook created in Exercise 7 such that it runs this challenge version of the report program as the last step in the workflow. Do NOT create a separate Jupyter notebook for this exercise.

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

```
RuntimeError: This program does not support processing with an empty input file.
```

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

```
                    COMBINED OPERATIONS
            SUMMARY OF FUEL EXPENSE AMOUNTS FOR 2020
                         BY MONTH

Month      US Amount      UK Amount      FR Amount   Total Amount
               (USD)          (USD)          (USD)          (USD)
    1      72,742.79      16,083.88      14,365.95     103,192.62
    2      45,758.93      24,874.43      14,354.80      84,988.16
    3      64,399.68      19,376.45      16,059.11      99,835.24
    4      58,308.71      19,591.66      17,538.96      95,439.33
    5      61,662.60      20,596.45      12,199.59      94,458.64
    6      70,589.85      17,399.42      14,119.40     102,108.67
    7      59,945.88      22,621.51      24,520.53     107,087.92
    8      69,182.55      23,754.80      13,556.39     106,493.74
    9      73,945.30      11,856.29      16,615.88     102,417.47
   10      69,162.65      12,775.92      16,768.71      98,707.28
   11      62,375.22      22,577.32      17,964.19     102,916.73
   12      83,871.87      17,280.50      18,998.29     120,150.66
Total     791,946.03     228,788.63     197,061.80   1,217,796.46


                    COMBINED OPERATIONS
          SUMMARY OF FUEL EXPENSE PERCENTAGES FOR 2020
                         BY MONTH

Month      US Amount      UK Amount      FR Amount   Total Amount
        (% of Total)   (% of Total)   (% of Total)   (% of Total)
    1          9.19%          7.03%          7.29%          8.47%
    2          5.78%         10.87%          7.28%          6.98%
    3          8.13%          8.47%          8.15%          8.20%
    4          7.36%          8.56%          8.90%          7.84%
    5          7.79%          9.00%          6.19%          7.76%
    6          8.91%          7.61%          7.16%          8.38%
    7          7.57%          9.89%         12.44%          8.79%
    8          8.74%         10.38%          6.88%          8.74%
    9          9.34%          5.18%          8.43%          8.41%
   10          8.73%          5.58%          8.51%          8.11%
```

11	7.88%	9.87%	9.12%	8.45%
12	10.59%	7.55%	9.64%	9.87%
Total	100.00%	100.00%	100.00%	100.00%

COMBINED OPERATIONS
SUMMARY OF FUEL EXPENSE AMOUNTS FOR 2020
BY DESCENDING TOTAL AMOUNT

Month	US Amount (USD)	UK Amount (USD)	FR Amount (USD)	Total Amount (USD)
12	83,871.87	17,280.50	18,998.29	120,150.66
7	59,945.88	22,621.51	24,520.53	107,087.92
8	69,182.55	23,754.80	13,556.39	106,493.74
1	72,742.79	16,083.88	14,365.95	103,192.62
11	62,375.22	22,577.32	17,964.19	102,916.73
9	73,945.30	11,856.29	16,615.88	102,417.47
6	70,589.85	17,399.42	14,119.40	102,108.67
3	64,399.68	19,376.45	16,059.11	99,835.24
10	69,162.65	12,775.92	16,768.71	98,707.28
4	58,308.71	19,591.66	17,538.96	95,439.33
5	61,662.60	20,596.45	12,199.59	94,458.64
2	45,758.93	24,874.43	14,354.80	84,988.16
Total	791,946.03	228,788.63	197,061.80	1,217,796.46

COMBINED OPERATIONS
SUMMARY OF FUEL EXPENSE PERCENTAGES FOR 2020
BY DESCENDING TOTAL AMOUNT

Month	US Amount (% of Total)	UK Amount (% of Total)	FR Amount (% of Total)	Total Amount (% of Total)
12	10.59%	7.55%	9.64%	9.87%
7	7.57%	9.89%	12.44%	8.79%
8	8.74%	10.38%	6.88%	8.74%
1	9.19%	7.03%	7.29%	8.47%
11	7.88%	9.87%	9.12%	8.45%
9	9.34%	5.18%	8.43%	8.41%
6	8.91%	7.61%	7.16%	8.38%
3	8.13%	8.47%	8.15%	8.20%
10	8.73%	5.58%	8.51%	8.11%
4	7.36%	8.56%	8.90%	7.84%
5	7.79%	9.00%	6.19%	7.76%
2	5.78%	10.87%	7.28%	6.98%
Total	100.00%	100.00%	100.00%	100.00%

Important – Check your work for these issues before submitting your Final Project

The Python programs that we create during this assignment will be imported and run from the Jupyter notebook. They should be created using a special programming style that is suited to this Jupyter notebook use case. Here are some of the program style issues that you should check before submitting:

1. **Programs should not run when imported.**
   The Python programs that we are creating in this assignment should not run when imported into the Jupyter notebook. They should only run when explicitly called. So, remember to include the following in your code:

   ```
   if __name__ == '__main__':
       main()
   ```

2. **Programs should not prompt the user for input.**
   The user interface approach that is appropriate for Jupyter notebooks does not include prompting users from Python programs using the *input()* function. Instead, any values that are needed to configure the Python program should be passed as parameters. This allows the parameter values to be set explicitly in the Jupyter notebook. Setting parameter values in the notebook provides the explicit documentation that the notebook users need to see. Then, the notebook reader sees all the configuration values that contributed to the workflow.

3. **Unit test should be done using PyCharm. The Jupyter notebook is a production run.**
   Use PyCharm to do unit testing of a Python program that will eventually be called from a Jupyter notebook. If you need to run more than one unit test scenario, then just change the parameter values in the *main()* function and run another test. This would, for instance, allow you to test a program twice – once with a populated input file and once with an empty input file. Remember that there is no need to repeat this unit testing in the Jupyter notebook. The call from the Jupyter notebook represents a production run. So, just run it with whatever data is needed to accomplish the production workflow.

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_final_project
```

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

```
trainor_kevin_final_project
```

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_final_project.zip
```

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

```
trainor_kevin_final_project.zip
```

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

Last Revised
2023-03-29