**MSCF Python Programming Basics**

**Homework 3**

***Due At 11:59 pm US Eastern Daylight Time,***

***Sunday, July 10, 2022***

1. (10 points) **Reading from expenses.txt**
   1. **Using open() and close()**

In Homework 2, we asked you to process a **list** of **str** values describing business expenses to produce a report in this format:

1 Amount Category Date Description

2 5.25 supply 20170222 box of staples

...

30 284.23 util 20170323 Peoples Gas

31 8.98 supply 20170325 Flair pens

The **expenses** **list** in Homework 2 was hard-coded, which is not realistic.

Make a copy of your Homework 2 code file into a new file named **hw3\_1\_a.py**. Update the **File** and **Author(s)** comments at the top of **hw3\_1\_a.py**:

**# File: hw3\_1\_a.py**

**# Author(s):** *... HW3 team member names here ...*

In the code, modify **expenses** to be an *empty* **list**, then read lines from the **expenses.txt** file (excluding the terminating newline characters) and append them to the **expenses** list. Modify the remainder of the code to produce just the final requested report, as illustrated above. Save and test.

***Make sure to save*** your **hw3\_1\_a.py** code file to submit as part of this homework.

* 1. **Using with open()**

A problem with using **open()** and **close()** with a file is that it’s very easy to forget to call **close()** (even if you have been writing code for years!).

As an alternative to this code:

**fin = open('infile.txt', 'rt', encoding = 'utf-8')**

**for line in fin:**

*... do something with each input line ...*

**fin.close()** # need explicit close() call

**print('Done with infile.txt')**

you can use a **with open()** block of code so that **close()** occurs automatically, like this:

**with open('infile.txt', 'rt',**

**encoding = 'utf-8') as fin:**

**for line in fin:** # indented block

*... do something with each input line ...*

# **close()** is automatic at end of block!

**print('Done with infile.txt')**

Make a copy of **hw3\_1\_a.py** named **hw3\_1\_b.py**. Update the **File** comment:

**# File: hw3\_1\_b.py**

**# Author(s):** *... HW3 team member names here ...*

Instead of **open()** and **close()** of the **expenses.txt** file, use a **with open()** block. Save and test, to confirm that this works correctly.

1. (20 points) **Collection Construction and Comprehension**

Using the IDE of your choice, edit the **hw3\_2.py** code file provided with this homework assignment. Update the **Author(s)** comment with the names of your homework team members.

* 1. Below the comments, the code defines a variable **s1** to refer to the **str** value **"Choo Choo Ch'Boogie"**. Uncomment the **print(s1)** function call for part 2.a, then save and test to confirm that the value of **s1** is displayed.
  2. Using **list** *construction* (that is, the **list()** function with an iterable argument), define a variable **m1** to refer to a **list** of the one-character substrings of **s1**, where the one-character substrings in **m1** occur in the same order as the characters in **s1**. Uncomment the **print(m1)** function call for part 2.b, then save and test.
  3. Using **set** *construction*, define a variable **set1** to refer to a **set** of the unique one-character substrings of **m1**. Uncomment **print(set1)**; save and test.
  4. Using an appropriate sequence of *constructions* and other *methods*, define a variable **t1** to refer to a **tuple** of the unique one-character substrings from **set1**, in ascending (non-descending) order. Uncomment **print(t1)**; save and test.
  5. The code for part 2.e defines a variable **s2** to refer to the **str** value **"the quick brown fox jumps over the lazy dog"**. Uncomment **print(s2)**; save and test.
  6. Using an appropriate *method*, define a variable **m2** to refer to a **list** of the words from **s2**. That is, **m2** should refer to **['the', 'quick', 'brown', …, 'lazy', 'dog']**. Uncomment **print(m2)**; save and test.
  7. Using **dict** *construction* and either **zip()** or **enumerate()**, define a variable **d1** that refers to a **dict** in which the *keys* are **int** indexes starting from 1, and the *values* are the words from **m2**. Uncomment **print(d1)**; save and test.
  8. Write a **for** loop using *multiple assignment* that displays the *keys* and corresponding *values* from **d1**, like this:

1: the

2: quick

3: brown

4: fox

5: jumps

6: over

7: the

8: lazy

9: dog

Save and test.

* 1. Write a **list** *comprehension* that defines a variable **m3** to refer to a **list** of the one-character substrings of **s2**. Uncomment **print(m3)**; save and test.
  2. Write a **list** *comprehension* that defines a variable **m4** to refer to a **list** of the one-character substrings of **s2** *except* for spaces (that is, the **list** should not contain any **' '** values). Uncomment **print(m4)**; save and test.
  3. Write a **set** *comprehension* that defines a variable **set2** to refer to a **set** of the unique one-character substrings of **s2** *except* for spaces. Uncomment **print(set2)**; save and test.
  4. Using a **list** *comprehension* followed by another *method*, define a variable **m5** that refers to a **list** of the unique one-character substrings of **set2**, in ascending (non-descending) order. Uncomment **print(m5)**; save and test.
  5. Write a **dict** *comprehension* that defines a variable **d2** in which the *keys* are lowercase letters (one-character strings) in ascending order, and the corresponding *values* are 0. Write a **for** loop using *multiple assignment* that displays the *keys* and corresponding *values* from **d2**, like this:

a: 0

b: 0

c: 0

...

y: 0

z: 0

Save and test.

* 1. Recall that an open file is *iterable*. Using **with open()**, open the **expenses.txt** file for reading, then write a **list** *comprehension* that defines a variable **m6** to refer to a **list** of the lines (strings) from the **expenses.txt** file, with the terminating newline character eliminated from each line. Uncomment **print(m6)**; save and test.
  2. Write nested **for** loops that step through each line in **m6** and each one-character substring in each line and, *if* the one-character substring is a *key* in **d2**, increments the corresponding *value* for that *key*. Then, write a **for** loop using multiple assignment that displays the *keys* and corresponding *values* from **d2**, like you did in part 2.m, above. (This will display each lowercase letter and the count of occurrences of that lowercase letter in the **expenses.txt** file.) Save and test.

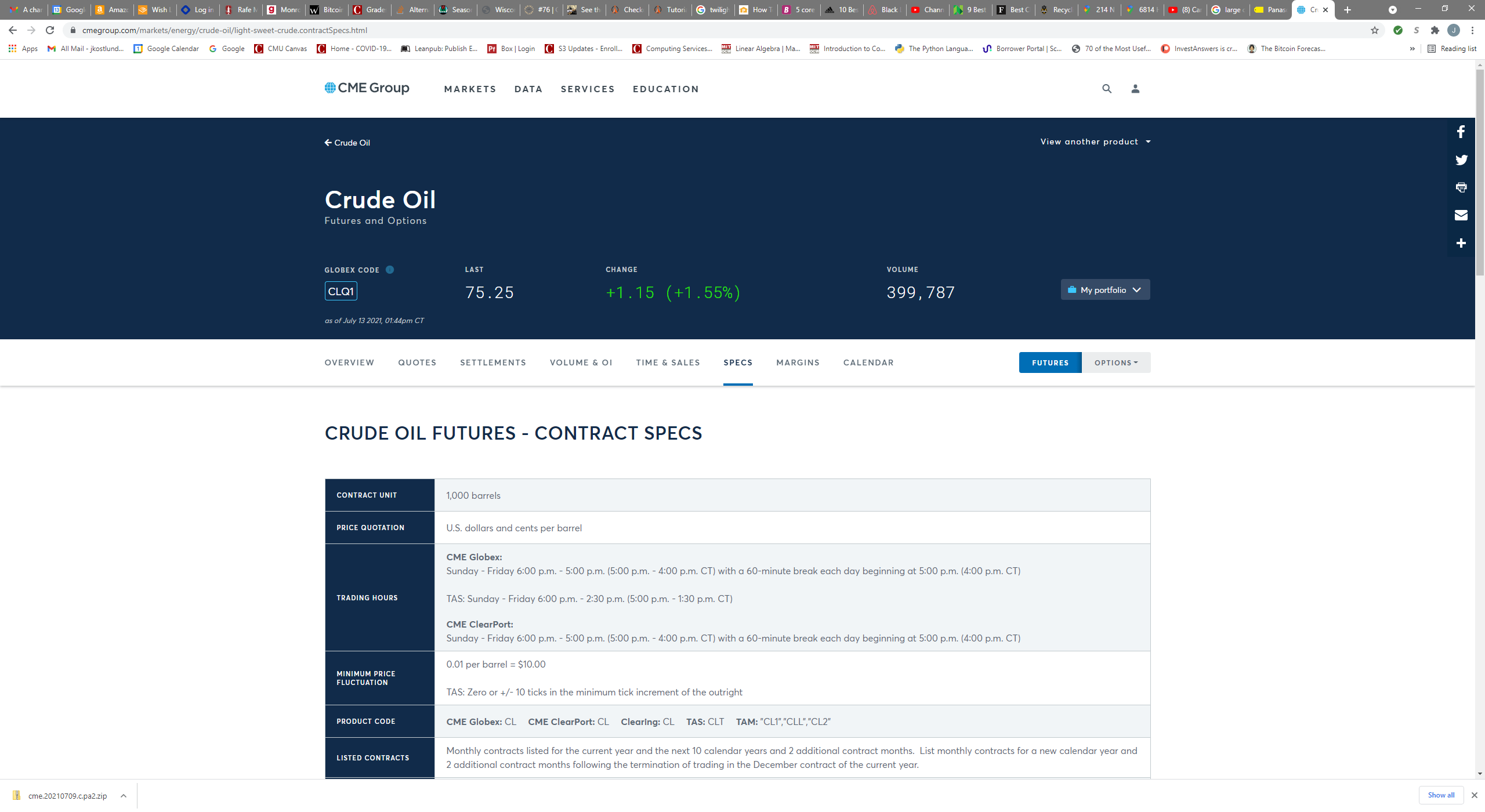
1. (70 points) **Commodity Futures and Option Contracts**

Commodity futures and option contracts of many kinds are traded on NYMEX, owned by CME Group. Each evening of each trading day, sometime between about 6:00 pm and 8:00 pm U.S. Central Time, a SPAN (Standard Portfolio Analysis of Risk) file is posted containing information about the day’s trading. For a given day, the name of this file is cme.*YYYYMMDD*.c.pa2.zip, where *YYYYMMDDD* is the 8-digit year, month, and day of the file. For many years, these files were publicly accessible, but in the past year or so they have not been accessible.

From Canvas, download the zipped SPAN file for Friday, July 9, 2021, **cme.20210709.c.pa2.zip**. Unzip, then display this SPAN file. You will see that it is an enormous text file (nearly one million lines) with its own idiosyncratic format—unfortunately *not* something simple and convenient like CSV or XML or JSON.

The *settlement prices* contained in the SPAN file are used to *mark to market* each trader’s account, so that gains/losses can be credited/debited each day to reduce the risk of counterparty default. Your job is to extract these settlement prices, as well as contract expiration dates (last trading dates), for two of the most heavily traded energy contracts: West Texas Intermediate (WTI) Crude Oil, and Henry Hub Natural Gas.

To learn more about WTI Crude Oil futures contract details, examine this web site: <http://www.cmegroup.com/trading/energy/crude-oil/light-sweet-crude_contract_specifications.html>



Notice that the CME Globex Product Code is **CL**; you will need this for scanning the SPAN file. Using other tabs at the top of this page, you can see current quotes, recent settlements, volume, etc. If you click the **Options** button just to the right of the **Futures** button at right, you will see information about option contracts based on the underlying futures contracts. There are about two dozen different types of option contracts for this underlying; we are interested in the **American Options**. When you look at the contract specifications, you will discover that its CME Globex Product Code is **LO**.

Return to the Futures page. In the **View an Energy Product** menu at upper right, scroll down and select **NG – Natural Gas (Henry Hub) Physical Futures** to learn about natural gas futures contracts and the corresponding American option contracts.

Write a Python program named **hw3\_3.py** that reads **cme.20210709.c.pa2** as its input file, and produces **CL\_and\_NG\_expirations\_and\_settlements.txt** as its output file. The output should be in exactly this form:

Futures Contract Contract Futures Options Options

Code Month Type Exp Date Code Exp Date

------- -------- -------- -------- ------- --------

CL 2021-09 Fut 2021-08-20

CL 2021-10 Fut 2021-09-21

*… and so forth, through contract month 2023-12 …*

CL 2023-12 Fut 2023-11-20

CL 2021-09 Opt LO 2021-08-17

CL 2021-10 Opt LO 2021-09-16

*… and so forth, through contract month 2023-12 …*

CL 2023-12 Opt LO 2023-11-15

NG 2021-09 Fut 2021-08-27

NG 2021-10 Fut 2021-09-28

*… and so forth, through contract month 2023-12 …*

NG 2023-12 Fut 2023-11-28

NG 2021-09 Opt ON 2021-08-26

NG 2021-10 Opt ON 2021-09-27

*… and so forth, through contract month 2023-09 …*

NG 2023-09 Opt ON 2023-08-28

Futures Contract Contract Strike Settlement

Code Month Type Price Price

------- -------- -------- ------ ----------

CL 2021-09 Fut 73.81

*… and so forth, through contract month 2023-12 …*

CL 2023-12 Fut 58.96

CL 2021-09 Call 0.50 73.36

CL 2021-09 Put 0.50 0.01

CL 2021-09 Call 1.00 72.86

CL 2021-09 Put 1.00 0.01

*… and so forth, through contract month 2023-12 …*

CL 2023-12 Call 210.00 0.03

CL 2023-12 Put 210.00 151.00

NG 2021-09 Fut 3.657

NG 2021-10 Fut 3.660

*… and so forth, through contract month 2023-12 …*

NG 2023-12 Fut 2.926

NG 2021-09 Call 0.250 3.407

NG 2021-09 Put 0.250 0.001

NG 2021-09 Call 0.500 3.157

NG 2021-09 Put 0.500 0.001

*… and so forth, through contract month 2023-09 …*

NG 2023-01 Call 4.000 0.174

NG 2023-01 Put 4.000 0.895

NG 2023-09 Call 1.800 0.937

NG 2023-09 Put 1.800 0.059

***Do not try to create a better output format***: it needs to be very easy for us to compare your output to our solution output, and to other students’ outputs. We will take off points if your output format varies too much from what is shown above. Our output format takes into account the order in which records appear in the SPAN file, so you don’t have to remember or accumulate too much information as you go. ***In particular, your program only needs to read through the input file one time.***

***Do not*** include contract months earlier than 2021-09, or later than 2023-12. ***Notice*** that Natural Gas options are not yet available for every month in 2023.

Since there are many, many strike prices for options on futures contracts, the output file is going to be very long (over 15400 lines), but not nearly as long as the SPAN file itself.

Fortunately, there is documentation online that describes the contents of CME SPAN files. If you Google for “cme span pa2 file format” you will find a page named “Risk Parameter File Layouts for the Positional Formats – SPAN…”. You will want to look at Type “B” Records, Expanded Format, and Type “8” Records, Expanded Format, to learn how to obtain the contract name, type, month, expiration date, strike, and settlement prices that you need. (The field names in the output file are *not* identical to the corresponding field names in the SPAN file documentation. The output file is intended to be read by finance people; the SPAN file documentation is intended to be read by computer geeks. You will need figure out how to be both.)

A few hints:

(a) Notice that the documentation counts character column positions from 1, whereas in your code you will need to count character positions from 0 for string slicing or other purposes

(b) Check the contract specifications to discover the number of decimal places you should display for prices of different commodity futures and options contracts.

(c) You will discover that the documentation is not quite perfect, but you should be able to figure out any problem(s) you encounter. (Big hint: talk to others.)

(d) Approach the program in stages: first, make sure you can write a program that simply copies the SPAN file to the output file; next, modify your program to copy just the type B and type 8 records from the SPAN file to the output file; next, modify your program to copy the type B and type 8 CL records; next, the type B and type 8 CL and NG records; and so forth, making definite steady progress toward your eventual goal with each revision of your code. As your coding skills improve, you can do two or three or four things in each revision step. Eventually, you will find that you can write dozens of lines of code encompassing many different tasks and goals, and your code will work the first time! But maybe not every time.

(e) You can use any Python facilities that you know, but you should not need anything other than what we covered in the Weeks 1 through 3 Lecture Notes/Videos and associated homework.

(f) Remember the Discussion board and **jostlund@andrew.cmu.edu**.

At the top of your code file, put in comments including this information:

**# File: hw3\_3.py**

**# Authors:** *... team member names here ...*

***And Finally***

Create a zip archive named **HW3\_Team\_***N***.zip**, where N is your team number, containing your code files for this assignment: **hw3\_1\_a.py**, **hw3\_1\_b.py**, **hw3\_2.py**, and **hw3\_3.py**. One team member should upload this zip archive to Canvas.