CSE 231 **Fall 2022**

Computer Project #9

This assignment focuses on the design, implementation and testing of a Python program that uses dictionaries and sets.

It is worth 55 points (5.5% of course grade) and must be completed no later than **11:59 PM on Monday, November 21, 2022.** After the due date, 1 point will be deducted for every 1 hour late. No more submissions will be accepted after Tuesday November 22, 2022.

If you submit by Sunday, November 20, 2022, you will receive a 2-point bonus. Note that you need to click on the submit button to be considered as submitted.

Assignment Overview

In this assignment, you will practice with reading files and using the csv library to process the file. You will practice with dictionaries, sets, and lists on handling the data and displaying it.

Assignment Background

A lot of people have heard about stocks. One of the most famous stock markets is the New York Stock Exchange (NYSE). This is where the companies raise capital that they use for shaping their future. You can use this information to invest money according to how the prices in stock for different companies change over time. In this project, you will be examining data from about 2010-2016 and making a program that displays some information to the user. The details of the functions are described below.

Assignment Specifications

You will develop a Python program that always reads two files, one related to prices of stocks using the companies' codes, and the other one the security file where all of the companies are with their companies' codes. You will create a program that can read both files and do some simple tasks with them according to what the user inputs. You will have one main dictionary. You will use that and one some other data structures to deliver the user's request.

`open_file () → prices_file_pointer, securities_file_pointer:`

a. This function is going to ask the user for both files to open. You will keep looping for the first one until a file is open. Afterwards, you have to do the same for the second file.
b. Parameters: None
c. Returns : Two file pointers. (file pointer of prices, and the file pointer of securities) in that order
d. The function displays nothing.

…

read_file(securities_fp) → set, dictionary:

Ticker symbc	Security	SEC filings	GICS Sector	GICS Sub Industry	Address of Headquarters	Date first added	CIK
MMM	3M Company	reports	Industrials	Industrial Conglom	St. Paul, Minnesota		66740
ABT	Abbott Labor	reports	Health Care	Health Care Equipn	North Chicago, Illinois	3/31/64	1800
ABBV	AbbVie	reports	Health Care	Pharmaceuticals	North Chicago, Illinois	12/31/12	2E+06
ACN	Accenture pl	reports	Information	IT Consulting & Oth	Dublin, Ireland	7/6/11	1E+06
ATVI	Activision Bli	reports	Information	Home Entertainme	Santa Monica, California	8/31/15	7E+05
AYI	Acuity Brand	reports	Industrials	Electrical Compone	Atlanta, Georgia	5/3/16	1E+06
ADBE	Adobe Syste	reports	Information	Application Softwa	San Jose, California	5/5/97	8E+05
AAP	Advance Aut	reports	Consumer Di	Automotive Retail	Roanoke, Virginia	7/9/15	1E+06
AES	AES Corp	reports	Utilities	Independent Powe	Arlington, Virginia		9E+05
AET	Aetna Inc	reports	Health Care	Managed Health Ca	Hartford, Connecticut	6/30/76	1E+06

a. This function takes the security's file pointer that has the names of the companies and their codes. It has a header line that you need to skip. It creates a set of all the company's names. And it also creates a master dictionary where the key is the company code (column 1), and the value is a list with 6 things. The list has the company name (column 2), the sector of the company (column 4), the subsector of the company (column 5), the address (column 6), the date added (column 7), and an empty list which later is going to be used to add more information from the other file. You can refer to the image above. All values in the list are strings.

```
D = {code: [name, sector, subsector, address, date_add, []]}
```

b. Parameters: `A file pointer`
c. Returns: `set, dictionary`
d. The function displays nothing.

add_prices (master_dictionary, prices_file_pointer) → None:

date	symbol	open	close	low	high	volume
1/5/16 0:00	WLTW	123.43	125.839996	122.309998	126.25	216360(
1/6/16 0:00	WLTW	125.239998	119.980003	119.940002	125.540001	238640(
1/7/16 0:00	WLTW	116.379997	114.949997	114.93	119.739998	248950(
1/8/16 0:00	WLTW	115.480003	116.620003	113.5	117.440002	200630(
##########	WLTW	117.010002	114.970001	114.089996	117.330002	140860(
1/5/10	STZ	16.08	15.92	15.86	16.129999	243160(
1/5/10	SWK	52.59	53.48	52.169998	53.549999	247410(
1/5/10	SWKS	14.89	15.05	14.7	15.07	510870(
1/5/10	SWN	50.610001	51.650002	50.43	51.93	476080(

a. This function does not return anything, but it changes the master dictionary while reading the prices file. Remember that at index 5 of each dictionary entry is an empty list. You need to first skip the header line. As you read the prices file you add to the empty list for the particular company in each prices file line. As you read each line of the prices file

create a list. The order of that list should be the date (column1), open info (column3), close info (column 4), low info (column 5), and high info (column 6). It should have 5 things in total each list. Use the company symbol (column 2) to index into the master dictionary so you can append the price list to that list (originally empty) at index 5 of the master dictionary entry. All price values (open, close, low, high) should be floats. Every line will have values for every column. Make sure that the company symbol exists in the dictionary before adding the information.

```
D = {code: [name, sector, subsector, address, date, [[date,
open info, close info, low info, high info],…]]}
```

b. Parameters: dictionary, file pointer
c. Returns : None
d. Displays nothing

`get_max_price_of_company (master_dictionary, company_symbol)` → `max_price, date:`

a. This function takes the master dictionary and a company symbol, and it gets the max high price and the date of the max price. It returns the tuple
`(max_price, date)`
or a tuple `(None, None)` if no max exists (when might that happen? when the list of prices is empty)
In the case of multiple entries of the same maximum price, return the date that is considered as maximum when compared as string (which happens to be the one returned when you use the `max()` function as shown in the example below).
Hint: create a list of tuples (what might you have in this tuple?), and then use `max()` on that list. List comprehension is handy.
Return `(None, None)` if the company does not exist.

For example,

```
L = [(2.0,'d1','dr1'),(3.0, '1/8/16', 'd2'),(1.0, 'd3',
'dd3'),(3.0, '1/13/16','d4')]
max_price = max(L)
```

will return:

```
(3.0, '1/8/16', 'd2')
```

Because `'1/8/16' > '1/13/16'` since `'8' > '13'`

b. Parameters: `dictionary, string`
c. Returns : `(float, string)`
d. Displays nothing

`find_max_company_price (master_dictionary) → max_company, max_price:`

a. This function takes the master dictionary and finds the company with the highest high price. Hint: use the function **`get_max_price_of_company`** to get each company's high, create a tuple (what might you have in this tuple?), put the tuple in a list, and find the `max()` of that list). Note that you need to ignore companies that do not have anything in their price list. This function returns the company with the maximum value and the price of it, in that order.
b. Parameters: `dictionary`
c. Returns : `(string, float)`
d. Displays nothing

`get_avg_price_of_company (master_dictionary, company_symbol) →avg_price:`

a. This function uses the master dictionary and company symbol to find the average high price for the company. `Round` the average to two decimal places (remember round at the last possible part of the function, e.g. right before you return).
If the company does not exist or there is no price data for the company, return 0.0.
Hint: be careful with division by 0.
b. Parameters: `dictionary, string`
c. Returns : `float`
d. Displays nothing

`display_list (list_of_strings):`

a. This function does not return anything, but it takes a list of strings and displays that list in three columns, each column is 35 characters wide. The correct format can be seen in strings.txt. Print items from left to right, and top down. Add a new line character "\n" when done printing all elements of the list. Do not add a new line character when there are less than 3 companies in the last line of the display.
b.
c. Do not add a new line when there are less than 3 in the same line.
d. Parameters: list of strings
e. Returns : None
f. Display: prints the list in three columns going from left to right and top down.

`main():`

This function is the main function where the user is going to interact with your program. You should print the welcoming banner provided as a constant in `proj09.py`.
Then, you should open both files and create your master dictionary by calling the respective functions.
You should always display the options, and then ask for the user input. Re-prompt on invalid input.

Options:

1. You should display the title first. `"Companies in the New York Stock Market from 2010 to 2016"` which should be 105 characters long and centered. Afterwards, you should print the set of companies, sorted alphabetically. After sorting, print it calling the `display_list` function.
2. You should do the same as option 1 but with the companies' code instead of the names. You should print the title first `"\ncompanies' symbols:"`
3. First, ask the user for the company symbol that they are trying to find, re-prompt if the company symbol is not in the master dictionary. Use the symbol to find the max price of that company by calling `get_max_price_of_company`. Then print the message with the price and the date. If there were no prices, then print an appropriate message (see `strings.txt`).
4. Find the company with the maximum stock price by calling `find_max_company_price`. Print the company's name and stock price.
5. Prompt for a company symbol; re-prompt if it is not in the master dictionary. Find the average high price by calling `get_avg_price_of_company`. If there were no prices, then print an appropriate message (see `strings.txt`).
6. Quit the program

As usual, find strings to print in the `strings.txt` file.

Assignment Notes and Hints

1. The coding standard for CSE 231 is posted on the course website:

    http://www.cse.msu.edu/~cse231/General/coding.standard.html

Items 1-9 of the Coding Standard will be enforced for this project.

2. The program will produce reasonable and readable output, with appropriate labels for all values displayed.

3. We provide a `proj09.py` program for you to start with.

4. If you "hard code" answers, you will receive a grade of zero for the whole project. An example of hard coding is to simply print the approximate value of e rather than calculating it and then printing the calculated average.

Assignment Deliverable

The deliverable for this assignment is the following file:

`proj09.py` – the source code for your Python program

Be sure to use the specified file name and to submit it for grading before the project deadline.

Function Test

Check the Function_Test Folder

Input/output Test
Check the Input_Output Folder

Grading Rubric

```
Computer Project #09                                Scoring Summary
General Requirements:
  ( 5 pts) Coding Standard 1-9
      (descriptive comments, function headers, mnemonic identifiers,
      format, etc...)

Implementation:
 ( 4 pts)  open_file function (manually graded)

 ( 7 pts)  read_file function

 ( 7 pts)  add_prices function

 ( 6 pts)  get_max_price_of_company function

( 6 pts)  find_max_company_price function

( 6 pts)  get_avg_price_of_company function

          display_list function (implicitly tested in the Tests below)

 ( 2 pts)  Test 1

( 2 pts)  Test 2
( 2 pts)  Test 3
( 2 pts)  Test 4
( 2 pts)  Test 5
( 2 pts)  Test 6
( 2 pts)  Test 7


Note: hard coding an answer earns zero points for the whole project
-10 points for not using main()
```