CIS 5357 Fall 2020

Programming Assignment # 6

(20 points)

**Due Date: Before 11:59 pm on Sunday, October 25, 2020**

**Caution:**

**The instructor has found many submissions with duplicate or similar code. The instructor expects individual effort on each and every submission. Assignments and exams assigned in this course are neither group projects nor any kind of group activity or collaboration are sanctioned. Such activities will be treated per the Academic Dishonesty policy as stated in the course syllabus. Each submission will be closely examined for plagiarism**. **A slightest hint of duplicate or similar code will be examined for academic integrity violation. Such submissions will automatically receive a grade of zero and reported to the Graduate College for further disciplinary action. To avoid such scrutiny, please do your own work. Please consult your instructor if you any questions or need clarification about assignments and examinations.**

1. **Requirements for Assignment 6:**
2. Name your Jupyter Notebook ‘YourName-Assignment6.ipynb’
3. Include your name and submission date as level 2 headings in the first cell of the notebook.
4. Insert a markdown cell with level 3 heading - Grading Comments – Totality of all points noted below resulted in a reduction of xx Points.
5. Insert a markdown cell with level 3 heading for the Program Name and its objective
6. The program design must be broken down into functions, with each function performing a specified task. The main driver program will also be coded as main() function. (see Hierarchy and IPO charts later)
7. Each function must include an appropriately written docstring.
8. Include functions defined for each program in their individual cells (in the same order as the IPOs described later).
9. Main() function should be the last one defined.
10. To execute the Main() function, include a call statement in its own cell below the Main() function cell.
11. Upload your source code file (.ipynb) to the Assignments section of Canvas ***BEFORE*** 11:59 pm on Sunday, October 25, 2020. **No other files need to be uploaded.**
12. **NO LATE ASSIGNMENTS WILL BE ACCEPTED. ASSIGNMENTS SENT VIA EMAIL AS ATTACHMENTS WILL ALSO NOT BE ACCEPTED. YOU ARE ALLOWED A MAXIMUM OF THREE ATTEMPTS TO SUBMIT YOUR FILE BEFORE THE DUE DEADLINE. YOU MISS THE DEADLINE, YOU LOSE IT. So, please start early to have a chance at getting any problem resolved before the submission deadline.**

**Introduction:**

Programming Assignment 6 will require you to design a python program that will use functions to modularize the program as well as consume JSON data retrieved via an Application Programming Interface or API or Endpoint (web service). These data will eventually be used to visualize them in graphic form in a later assignment.

**Specifications:**

You will access a Web API (web service) to retrieve historical stock market data for a specific company. You will use the following API URL to access historical stock data for Microsoft, Inc.

<https://www.alphavantage.co/query?function=TIME_SERIES_MONTHLY&symbol=MSFT&apikey=demo>

The data returned by this URL is in JSON format and shows the stock’s monthly prices (open, high, low, and close) and trading volumes for the end of the month. Your application will extract the date, high and low stock price, and trading volume at the end of each month and display these in console output.

**Requirements:**

1. The program design must be modularized, delegating each specific major task to individual function. A Hierarchy chart and an IPO chart for each function are provided later in these specifications. Your program design MUST adhere to these charts, including naming of functions.
2. In addition to using the provided function names, each function must include an appropriate docString as well as brief comments about the primary task of the function.
3. The program will NOT use ANY global variables. All variables must be local to either the function or main program. Functions and main program can exchange data only through the use of parameters in function calls and Return statements.
4. The program will be accessing JSON data stream from a Web API.
5. **Hierarchy Chart:**

**\*\* implies that the main function must call get\_stock\_data() function first before the main() function can call any other function(s). Note that *get\_stock\_data()* function does not call any functions. Only the Main() function can call extract\_time\_series\_data() and print\_stock\_info() .**

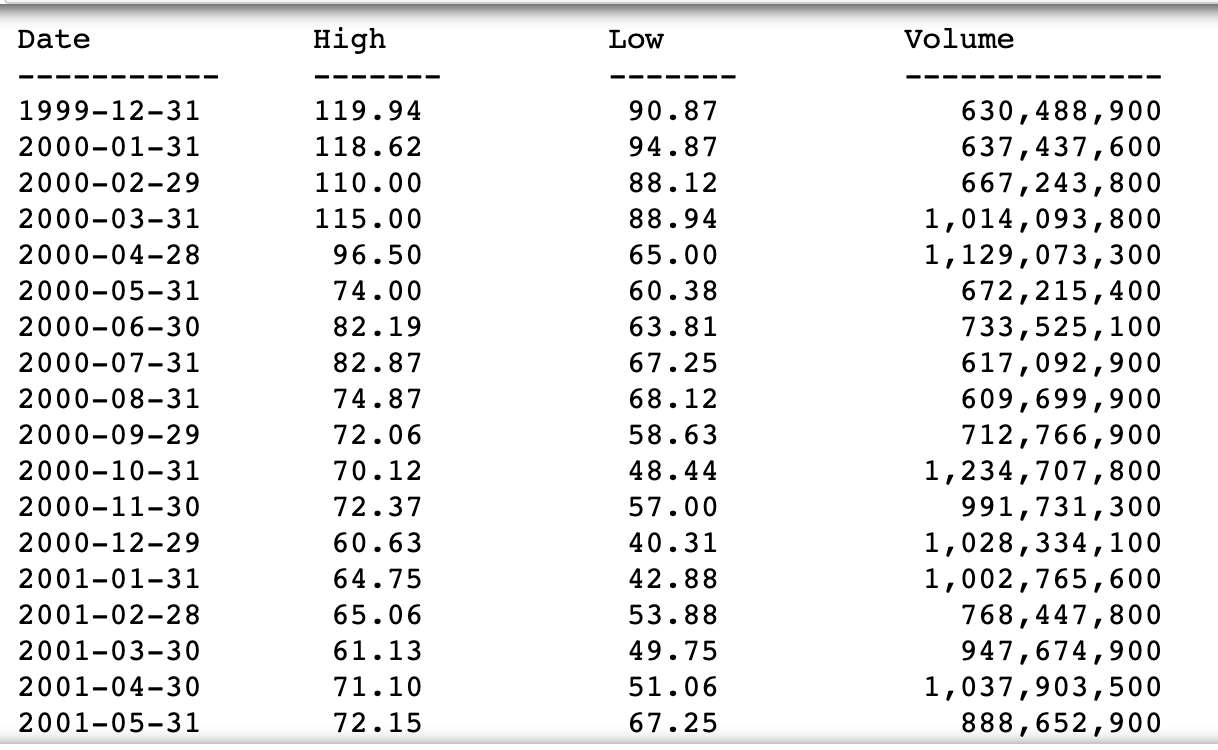
1. **Input-Process-Output (IPO) charts:**

|  |  |  |
| --- | --- | --- |
| get\_stock\_data() | | |
| Input | Process | Output |
| None | * Access the Web API URL to receive JSON formatted data stream * Transform JSON-formatted data into Python data structure (dictionary) * Extract the data for the monthly time series key into appropriate python data structure (dictionary) * Return the Python dictionary containing the time series data. | Dictionary containing time-series data |

|  |  |  |
| --- | --- | --- |
| extract\_time\_series\_data() | | |
| Input | Process | Output |
| Dictionary containing time-series data | * Receive the dictionary containing time series data. * Using the appropriate method of iterating through the dictionary items, extract **date, high price, low price, and volume** of each date, create a CSV string from these values and add CSV string to a list structure. This is so the items can be arranged in chronological order by date. (see section 9.1, pages 446-452 for an appropriate method) * Sort the list from the step above and sort it using the sorted method of the list. This will put the list in chronological order by date, with the earliest date first. * Iterate though the sorted list to extract the **date, high and low stock prices, and trading volume** from the sorted list into their own individual lists to get them ready for printing and later visualization. Make sure that date, associated stock prices, and trading volume are synchronized correctly by date. * High and low prices must be rounded to 2 decimal places, using the *round()* function, and not *format*() function. * Return the date, high and low stock prices, and trading volume data lists to the Main() function. There will be one data structure for each item – date, high prices, low prices, and trading volumes. | Properly synchronized data structures for date, high and low stock prices. |

|  |  |  |
| --- | --- | --- |
| print\_stock\_info() | | |
| Input | Process | Output |
| Date, High and Low stock prices, and training volume lists | * Print to the console, a list of highs, lows, and trading volumes by date, with data for earliest date listed first – in chronological order. These data will be used to generate some graphs in a later assignment. * Print the date, high and low stock prices, and trading volume in that order. The stock prices must be displayed as $ currency value to two decimal places. The trading volume must use 1000s comma separator. * All high and low price data must be aligned on the decimal point. That is, decimal point should line up. Within the column, these values must be right aligned. * The trading volume is aligned to the right within its column. * All data must be properly aligned under their respective column headers. – date must be left-aligned while all numeric data are right aligned within their respective columns. | None |

|  |  |  |
| --- | --- | --- |
| main() | | |
| Input | Process | Output |
| None | * **Only If needed**, initialize all local constants and variables to be used in the main function * Call *get\_stock\_data()* function to get the historical data for Microsoft stock and receive the python data structure from the *get\_stock\_data()* function * Call *extract\_time\_series\_data*(), passing in the python data structure received in step above. Receive individual data structures for date, high and low stock prices, and trading volumes from the *extract\_time\_series\_data()* function. * Call *print\_stock\_info()* function, passing in the individual data structures for date, high and low stock prices, and trading volumes.. * Print the end of program message to inform the user that the program was ended normally. | None |

Partial Console Output to show formatting, alignment, and values