DeVry University

College of Engineering and Information Sciences

1. Python Stock Tracking Project

Module 7

# Background

1. The project will provide students with experience creating applications in Python. Students will use object-oriented techniques to develop a stock tracking application. By processing the historical stock data, profit/loss reports can be generated. Data storage will allow users to save and retrieve stock data. The system will use the Python libraries to create charts.

**In this part of the project, you will implement a graphical user interface for your stock tracking program.**

# Objectives – Module 7

1. Use tkinter to implement a graphical user interface.

# Steps

1. **Always test as you go!**

Python can be used to create software with many types of user interfaces including text-based, graphical user interfaces (GUIs), web, and there is more work being done to bring Python to mobile devices.

We have worked out the logic and functionality of our stock tracking program using a text-based menu-driven interface. This is a good way to start, however, users are used to more modern and easy to use graphical user interfaces.

There are several libraries and frameworks that can be used to develop GUIs for Python programs. In this part of the project, we'll use the **tkinter** library which is the interface for the Tk toolkit (hence the name tkinter which stands for Tk Interface). An advantage of using tkinter is that it's cross-platform. Your Python program will run on Windows, Linux, or Mac.

***Remember, we are REPLACING the stub code, so you should delete the stub and use your own code.***

***Delete the following stub code in all functions that you will be implementing.***

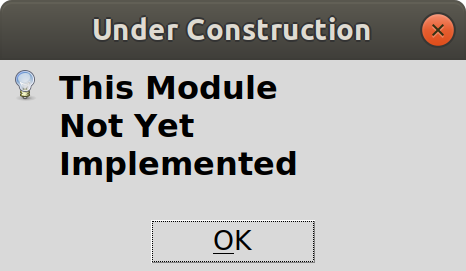
messagebox.showinfo("Under Construction","This Module Not Yet Implemented")

## 1. Create a GUI Interface

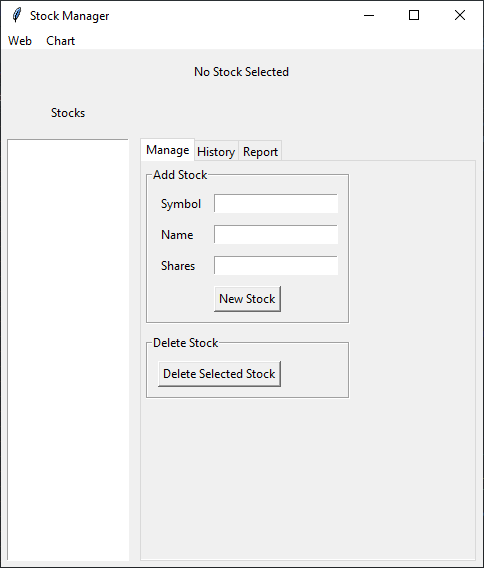
For this part of the project we will be working with the **stock\_GUI.py file**. To save time, this file has the user interface elements already included. It will be your job as a programmer to make it work.

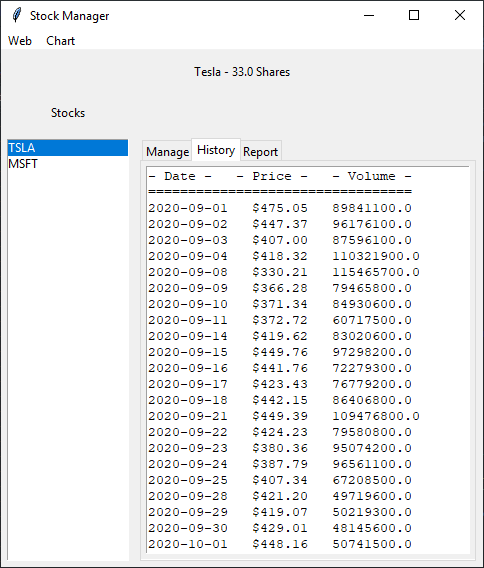
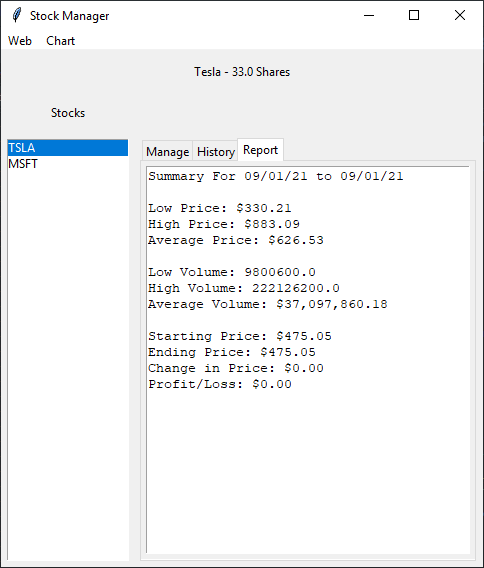
**Run stock\_GUI.py and explore the interface. Click on the menus, tabs, and buttons to get a feel for how the program will operate.**

Note that as you explore the interface, you will notice that the functions for the buttons and menus have not been implemented. Instead you will see an “under construction” message.



Note, the screen shot included here shows the completed program with sample data. Since the starter version does not have the functionality included, you will not see any data in your version until you implement the code.

* At the top of the window is the title bar with the name of the program. You will replace the tile with your own name for your stock program (ex. “Mary’s Stock Manager”, or “Stan’s Stocks”...be creative!).
* Below that is the menu. The file menu lets us save the data or load the data from a CSV file. The Web menu lets us retrieve stock price history by importing .csv files from Yahoo! Finance. The Chart menu displays a chart of the selected stock.
* Next is a label that will show the name of the selected company and number of shares owned.



* The main area has a list of stocks on the left.
* On the right side is a tabbed area with tabs to Manage stocks, show stock History, and display the Report.
* The Manage tab allows us to add stocks or delete stocks.
* The History tab shows the daily stock price and volume history.
* The Report tab shows the summary report including the profit or loss for the period.

### 1.2 Initial Code Review

After exploring the interface take a look at the code. Open the **stock\_GUI.py** file in your IDE.

The main interface is implemented as a class called StockApp which is instantiated in the main() function.

The code has two main sections. The top part sets up the user interface and has been written for you. Below that is a section that implements the functionality for loading data, saving data, adding stocks, buying and selling shares, and removing stocks. There are also sections for web scraping and loading the .csv file which will be implemented later.

**User Interface**

As you look at the code for the user interface, it’s important to understand the concepts of parent or master widget. Widgets are the user interface elements. Some widgets (such as windows, tabs, etc.) can be containers for other widgets. For example, you can place a tab inside a window and inside the tab you might have a button.

In our application root is the parent window, and all the other controls are placed in this. We create our window using the following code:

self.**root** = Tk()

In some cases, widgets are placed within other widgets. For example we create a tab (called a notebook in tkinter) which is placed in the root window:

self.**notebook** = ttk.Notebook(self.**root**,padding="5 5 5 5")

To add a tab to the notebook we call the add method. These tabs can become containers for other objects. For example to create our Manage tab we use the following:

self.**notebook**.add(self.**mainFrame**,text='Manage')

On the tab, we can place other widgets. For example on the Manage tab we use a group to put a box around the controls related to adding a new stock:

self.**addStockGroup** = LabelFrame(self.**mainFrame**,text="Add Stock",padx=5,pady=5)

Within the group, we have labels, text boxes, and buttons. Here’s how we add a button:

self.**addStockButton** = Button(self.**addStockGroup**,text = "New Stock",command=self.add\_stock)

Looking at the code, you can see how the different user interface elements are created, and how widgets can be placed inside other widgets.

Another thing to be aware of when creating graphical user interfaces is the positioning of widgets within their parent container. Tkinter supports several different methods for positioning items. The method used here is the grid approach. When using grids, you specify the row and column **within the parent widget**.

For example we’ve placed a header at the top of the form that shows the company name and number of shares. The label is called headingLabel. The parent container is root so the column and row references are based on the main window. We’ve placed it in column 0 (the first column) and row 0 (the first row). We want it to span across the entire window so we have a columnspan of 3.

self.headingLabel = Label(self.root,text="No Stock Selected")

self.headingLabel.grid(column=0,row=0,columnspan=3,padx = 5, pady = 10)

The New Stock button is inside the addStockGroup widget, so the grid is in reference to the group, not the entire window. The grid row and column always refer to the parent control. The button is in the 4th row of the addStockGroup widget (rows and columns start at 0).

self.addStockButton = Button(self.addStockGroup,text = "New Stock",command=self.add\_stock)

self.addStockButton.grid(column=0,row=3,columnspan = 2, padx = 5, pady = 5)

After setting up the form the way we want, we call the mainloop() method to open the form and handle events such as mouse clicks, etc.

self.root.mainloop()

Using this basic knowledge and looking at the existing code, you can customize the interface. As long as it works, feel free to use your creativity.

**Methods to Implement**

You will be working on the following methods.

* display\_stock\_data()
* add\_stock()
* delete\_stock()
* display\_chart()

Because much of the functionality has already been created for the console version of the program, we can take advantage of existing code so we don’t have to write as much new code.

Because much of the functionality has already been created for the console version of the program, we can take advantage of existing code so we don’t have to write as much new code.

### 1.3 Write the Code to Implement Features

**display\_stock\_data(self)**

The display\_stock\_data() method in the stock\_GUI.py file basically provides the same functionality as the display\_report() function in the stock\_console.py file.

We will use the symbol that is selected using the code below:

symbol = self.stockList.get(self.stockList.curselection())

The difference is that instead of using print() to print to the console, we will use the insert() method to add rows to the dailyDataList and stockReport text boxes. Also, since we have the company and shares shown at the top of the window, we don’t need to display the header information as we did on the console-based report.

Note, code may be shown with line breaks due to space limitations but each statement should be entered on its own line. Add the following **python code** to **display\_stock\_data(self)**

if (self.stockList.curselection()):

symbol = self.stockList.get(self.stockList.curselection())

for stock in self.stock\_list:

if stock.symbol == symbol:

self.headingLabel['text'] = stock.name + " - " + str(stock.shares) + " Shares"

self.dailyDataList.delete("1.0",END)

self.stockReport.delete("1.0",END)

self.dailyDataList.insert(END,"- Date - - Price - - Volume -\n")

self.dailyDataList.insert(END,"=================================\n")

for daily\_data in stock.DataList:

row = daily\_data.date + " " + '${:0,.2f}'.format(daily\_data.close) + " " + str(daily\_data.volume) + "\n"

self.dailyDataList.insert(END,row)

#display report

count = 0

price\_total = 0.00

volume\_total = 0

lowPrice = 999999.99

highPrice = 0.00

lowVolume = 999999999999

highVolume = 0

for daily\_data in stock.DataList:

count = count + 1

price\_total = price\_total + daily\_data.close

volume\_total = volume\_total + daily\_data.volume

if daily\_data.close < lowPrice:

lowPrice = daily\_data.close

if daily\_data.close > highPrice:

highPrice = daily\_data.close

if daily\_data.volume < lowVolume:

lowVolume = daily\_data.volume

if daily\_data.volume > highVolume:

highVolume = daily\_data.volume

priceChange = lowPrice-highPrice

if count > 0:

self.stockReport.insert(END,"Summary Data--\n\n")

self.stockReport.insert(END,"Low Price: " + "${:,.2f}".format(lowPrice) + "\n")

self.stockReport.insert(END,"High Price: " + "${:,.2f}".format(highPrice) + "\n")

self.stockReport.insert(END,"Average Price: " + "${:,.2f}".format(price\_total/count) + "\n\n")

self.stockReport.insert(END,"Low Volume: " + str(lowVolume) + "\n")

self.stockReport.insert(END,"High Volume: " + str(highVolume) + "\n")

self.stockReport.insert(END,"Average Volume: " + "${:,.2f}".format(volume\_total/count) + "\n\n")

self.stockReport.insert(END,"Change in Price: " + "${:,.2f}".format(priceChange) + "\n")

self.stockReport.insert(END,"Profit/Loss: " + "${:,.2f}".format(priceChange \* stock.shares) + "\n")

else:

self.stockReport.insert(END,"\*\*\* No daily history.")

**add\_stock(self)**

The add\_stock() method is called when users click the New Stock button.

self.addStockButton = Button(self.addStockGroup,text = "New Stock",command=self.**add\_stock**)

*Code Description*

First we’ll create a new Stock object. We call the constructor passing the data that the user has entered in the text boxes. For example self.addSymbolEntry.get() will retrieve the symbol that the user entered into the box.

We then add the new\_stock object to our stock list.

We next add the symbol to the end of the stock listbox.

Finally, we delete the text entered by the user for symbol, company name, and shares.

Note, code may be shown with line breaks due to space limitations but each statement should be entered on its own line. **Implement the following pseudocode to add\_stock(self)**

**add\_stock(self)**

create **new\_stock** with the symbol, name, and shares from the GUI. Hint: new\_stock = Stock(self.addSymbolEntry.get(),self.addNameEntry.get(),float(self.addSharesEntry.get()))

append **new\_stock** to **self.stock\_list**

add the symbol to the end of the stock **listbox.** Hint:self.stockList.insert(END,self.addSymbolEntry.get())

delete the text entered for addSymbolEntry hint: self.addSymbolEntry.delete(0,END)

delete the text entered for addNameEntry

delete the text entered for addSharesEntry

**delete\_stock()**

The delete\_stock() method is called when the user clicks the Delete Selected Stock button:

self.deleteStockButton = Button(self.deleteGroup,text="Delete Selected Stock",command=self.**delete\_stock**)

*Code Description*

First we get the symbol for the stock selected by the user in the stockList listbox.

Next we use a loop to search for the stock in the list using i to determine the index of the stock.

When we find the correct stock, we use the pop() method to remove it from the list.

We update the display of stock history and re-display the list of stocks.

Finally we show a message box confirming the deletion. Implement the delete\_stock() method.

**delete\_stock(self)**

Check if a stock is selected. Hint: if (self.stockList.curselection()):

get **symbol** selected by the user in the listbox. Hint: symbol = self.stockList.get(self.stockList.curselection())

set **i** equal to 0

for each **stock** in **self.stock\_list**

if **stock.symbol** = **symbol**

call **self.stock\_list.pop(i)**

increment **i** by 1

call self.display\_stock\_data()

delete the entry from the stock list. Hint: self.stockList.delete(0,END)

for each **stock** in **self.stock\_list**

insert **symbol** in the stock list. Hint: self.stockList.insert(END,stock.symbol)

display a message to the user that the stock was removed. Hint: messagebox.showinfo("Stock Deleted",symbol + " Removed")

**display\_chart(self)**

The display\_chart() method is called when the user select the Display Stock Chart option on the Chart menu. **Add the following python code to the display\_chart() method.**

symbol = self.stockList.get(self.stockList.curselection())

date = []

price = []

volume = []

company = ""

for stock in self.stock\_list:

if stock.symbol == symbol:

company = stock.name

for dailyData in stock.DataList:

date.append(dailyData.date)

price.append(dailyData.close)

volume.append(dailyData.volume)

plt.plot(date,price)

plt.xlabel('Date')

plt.ylabel('Price')

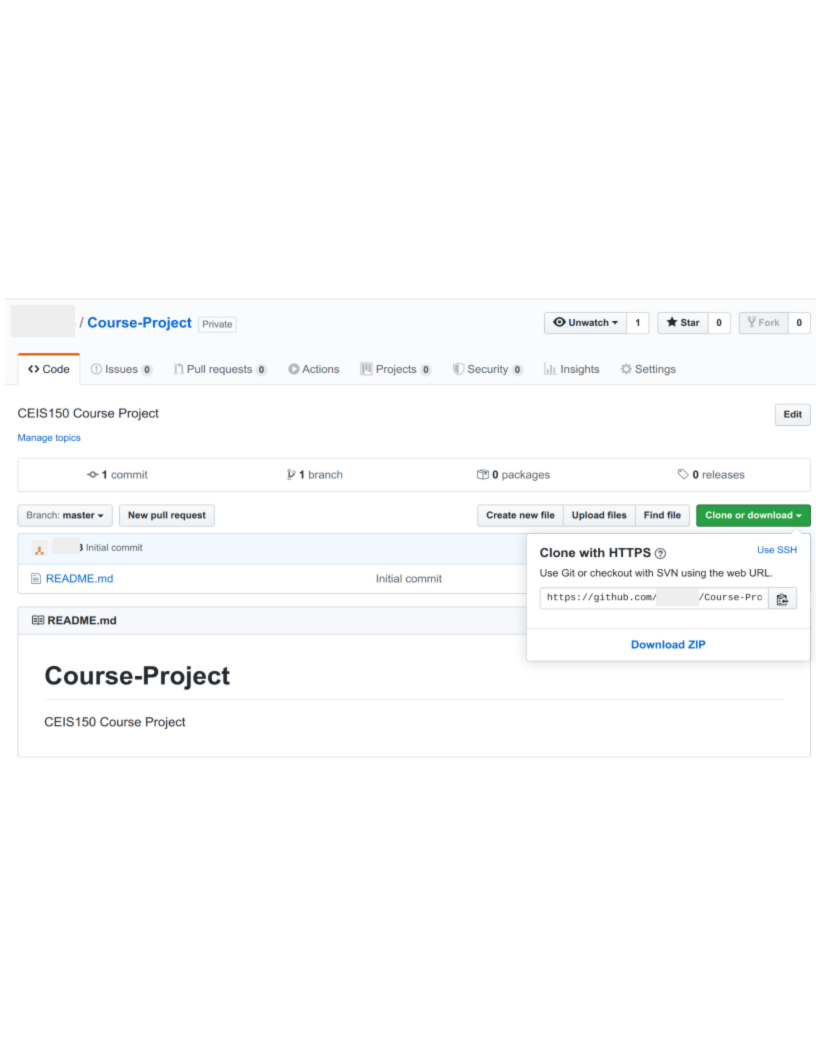
plt.title(company)

plt.show()

# Deliverables

1. **To submit this part of the project:**
2. 1. Run the program (stock\_GUI.py).
3. 2. Add two stocks and take a screenshot showing the stocks on the page (add the same one that you downloaded the historical data for from the previous module).
4. 3. Select the stock.
5. 4. Import CSV data. Take a screenshot of the History tab showing the data imported.
6. 5. Display a chart. Take a screenshot of the chart.
7. 6. Take a screen shot of the Report tab.
8. 7. Submit the file in the Assignments area.

# Optional

1. Push your updated project files to GitHub. This will provide a backup for your project should anything happen to your local files. It will also allow you to go back through your files to see a change history. If you accidentally modify or overwrite code, you can always find an earlier version of the file.
2. ****
3. Find your repository url on GitHub. Then navigate to your project folder and use the following commands. Replace X with the week/module number you are submitting. Replace {your url}with the address of your GitHub repository.

The following commands will: stage the changes, commit the changes, and push the updates to GitHub.

1. **git add --all**
2. **git commit -m "Module X"** ←Change X to the Module you are submitting.
3. **git push {your url}** ←Change {your url} to the url for your repository on GitHub.

**Note: VS Code users can use the Source Control tab on the left to stage, commit, and push updates to GitHub.**

1. **Note: Use the Snipping Tool built into Windows to take screen shots (click Start and search for Snipping Tool)**