Project 2

Capital Gain Calculator using Queues

Objective

This project is meant to help you develop skills at working with strings and queues. It also gives you exposure to classes and objects in Python. You will be provided the code for a complete class ArrayQueue, so you will not need to create one. You will simply use the methods from this class just like you have been using methods from Python's built-in classes (such as strings, lists, and dictionaries). As in the first project, you'll accept command-line arguments, read a text file, and record your program's execution time. In this project you will be provided with a project description, and list of program requirements, a sample input file, and some starting code (p2.py).

Project Description

When a share of common stock of some company is sold, the capital gain (or, sometimes, loss) is the difference between the share's selling price and the price originally paid to buy it. This rule is easy to understand for a single share, but if we sell multiple shares of stock bought over a long period of time, then we must identify the shares actually being sold. A standard accounting principle for identifying which shares of a stock were sold in such a case is to **use a FIFO protocol** — the shares sold are the ones that have been held the longest (indeed, this is the default method built into several personal finance software packages).

For example, suppose we:

- buy 100 shares, at $20 each, on day 1,
- buy 20 shares, at $24 each, on day 2,
- buy 200 shares, at $36 each, on day 3, and then
- sell 150 shares, on day 4, at $30 each:

Then applying the FIFO protocol means that of the 150 shares sold, 100 were bought on day 1, 20 were bought on day 2, and 30 were bought on day 3. The capital gain in this case would therefore be: (100*(30-20)) + (20*(30-24)) + (30*(30-36)) =
(100*10) + (20*6) + (30*(-6)) = $940.

Write a program that takes as input a text file containing a sequence of transactions of the form "buy x share(s) at $y each" or "sell x share(s) at $y each," assuming that the values x and y are integers. Each line in the file represents a transaction (either "buy" or "sell"). Given this input sequence, the output must include the capital gain (or loss) per "sell" transaction, and total capital gain (or loss) for the entire sequence.

Hint: Keep information about the purchase shares and prices in a **queue**, and then match those against sales. Care must be taken if only part of a "buy" transaction is sold.

Command Line Arguments

You must take the name of one text file as a command line argument. The following code checks that one argument was passed via the command line.

```
import sys

if len(sys.argv) != 2:
    raise ValueError('Please provide one file name.')
inputFileName = sys.argv[1]
print("\nThe file that will be used for input is", inputFileName)
```

Read from a File

The input file will have many lines. Each line will be stored as a string in your program. Each string will be placed in a list, so that each item in the list will correspond to a line in the file.

To open the file for reading, you must use the command:

```
f = open(inputFileName,"r")
```

At this point you can read the lines from the file using:

```
myList = f.readlines()
```

This will read the whole file and store each line as an item in the list of strings *myList*.

Don't forget to close the file after you finish reading it:

```
f.close()
```

ArrayQueue Class

A class definition has been provided in the starter code. You do not need to modify this class. You'll simply use its methods to add to, remove from, or update the queue. For example, if your queue is named `q`, you may use `anyVarName = q.first()` to get the data in the element in the front of the queue and assign it to `anyVarName`, without removing the element from the queue. If you use `anyVarName = q.dequeue()` you will get the element in the front of the queue and also remove it from the queue.

Process the List of Transactions

In the main body of your program, loop through the list of transactions and, for each item in the list, parse the string to find the numbers that represent the *number of shares* and the *price*. The format of the strings is:

- buy **x** share(s) at $**y** each
- sell **x** share(s) at $**y** each

Each "buy" transaction must be added to the queue. Since there are 2 numbers to add per transaction, consider using a tuple for each element of the queue. For example, a line *buy 100 share(s) at $20 each* can be saved as the tuple (100, 20), which will be added to the queue.

Each "sell" transaction will remove and/or update elements from the front of the queue. For transactions of type "sell", also print the capital gain/loss for that transaction.

After processing all transactions in the input file, the program prints the total capital gain (sum of the gains/losses per transaction).

Then it prints all the shares remaining in the queue (it's okay to remove them from the queue as you print). Finally, it prints the running time.

Running Time

You need to record the total running time of your program. This time should include the time to read the text file, process each string, add and remove from the queue, and print the capital gains. It is suggested that you start a timer as one of the first things you do in the program and stop it immediately before printing out the statistics. The following is a little sample code that shows how you can time events.

```
import time

start = time.time()
# ...
end = time.time()
print(end - start)
```

Sample Run

Here is a sample run of the program. A few notes:

1. I have the text file (transactions.txt) and my python solution (p2.py) in the same folder (C:\Code). You may place them in your preferred folder.
2. Using the command prompt on Windows (terminal on MacOS), I run the program by typing the program name followed by the input text file. You may use an IDE instead if you prefer, such as VS Code, the Python's IDLE, or PyCharm.

```
C:\Code>python p2.py transactions.txt

****************************************
The file that will be used for input is: transactions.txt
****************************************

buy: 20 at $2
buy: 50 at $1
buy: 10 at $3
sell: 20 at $4
This transaction's capital gain is: 40
```

The queue will now have:
(50, 1)
(10, 3)

```
sell: 30 at $3
This transaction's capital gain is: 60
```

The queue will now have:
(20, 1)
(10, 3)

```
sell: 30 at $4
This transaction's capital gain is: 70
```

The queue will now be empty.

```
buy: 40 at $2
sell: 20 at $1
This transaction's capital gain is: -20
```

The queue will now have:
(20, 2)

```
buy: 100 at $5
buy: 5 at $10
sell: 10 at $10
This transaction's capital gain is: 80
```

The queue will now have:
(10, 2)
(100, 5)
(5, 10)

```
sell: 100 at $10
This transaction's capital gain is: 530

buy: 5 at $1
buy: 15 at $5
buy: 2 at $10
buy: 55 at $5
buy: 2 at $10
sell: 70 at $5
This transaction's capital gain is: -15

**************************
The total capital gain is: 745

******************************
Shares remaining in the queue:
22 shares bought at $5 per share.
2 shares bought at $10 per share.

**********************
Total Time of Program: 0.00654410 milliseconds
```

Handing in Your Project

Be sure to comment your code well so that it can be easily browsed to see what is going on. Code that is excessively difficult to read may be penalized.

Turn in a single file called "p2.py" to the assignment on Canvas.

Grading

The project will be graded out of 100 points and be graded on the following criteria:

- Code Organization
- Correct Output
- Performance (running time)

We will grade your code by running it against a series of test files and checking the output of the program against a known, correct implementation. More in depth tests may be run than the initial one provided. This means you should write some of your own test files to check for any errors.

We will also examine parts of your code to make sure that you are following the instructions listed in the requirements.