**MATH20621 Coursework 2021 (**[**Stefan Güttel**](http://guettel.com)**)**

**stocktrader: A Python module for virtual stock trading**

**Introduction**

The aim of this coursework is to code a Python 3 module stocktrader that allows the user to load historical financial data and to simulate the buying and selling of shares on the stock market. *Shares* represent a fraction of ownership in a company and the total of all shares form the *stock* of that company. Shares can be bought and sold on the stock market at a price that varies daily. A reasonable investor aims to buy cheap shares and sells them when the price is higher. The actions of buying and selling shares are referred to as *transactions*. Transactions usually incur *transaction fees*, but we will not consider these in this project.

**Frequently asked questions**

**What can/cannot be used for this coursework?**

The Python knowledge contained in the lecture notes is essentially sufficient to complete this project. While you are allowed to consult the web for getting ideas about how to solve a specific problem, the most straightforward solution may already be in the lecture notes. You are also allowed to use online sources, but you must clearly indicate copy-and-pasted code like so:

# the following 3 lines follow a similar code on

# http://webaddress.org/somecode.htm as retrieved on 24/11/2021

**Let’s be clear about what is \*not\* allowed: You are NOT allowed to send, give, or receive Python code to/from classmates and others. This project is the equivalent of a standard exam and it counts 70% towards your final mark. Consequently, the standard examination rules apply to this project.**

Once your Python module is submitted via the Blackboard system, it will undergo plagiarism tests:

1. The turn-it-in system automatically checks for similarities in the codes among all students.
2. Another Python program compares the syntax of all submitted codes.

**Even if you are the originator of the work (and not the one who copied), the University Guidelines Plagiarism and Academic Malpractice (link below) require that you will be equally responsible for this case of academic malpractice and may lose all marks on the coursework (or even be assigned 0 marks for the overall course).**

Link to the University guidelines: <https://documents.manchester.ac.uk/DocuInfo.aspx?DocID=2870>

**How is this coursework assessed?**[**¶**](https://personalpages.manchester.ac.uk/staff/stefan.guettel/py/coursework2021.html#How-is-this-coursework-assessed?)

There are several factors that enter the assessment:

* First, it will be checked whether you have followed the tasks and format specified below, using the prescribed function names, variable names, and data structures.
* Second, your module will be tested manually by performing a number of predefined transactions, loading/saving a couple of portfolios, and testing trading strategies.
* All functions of your module will be tested automatically by another computer program. The outputs are then compared to verified outputs of another implementation. (This also means that if you do not strictly follow the format specified below then some of these automatic tests will fail and you may lose marks.)
* Each function/line of code should be bug free. Functionality will be the main factor in the assessment.
* Your code should be robust to exceptional user inputs (using Exceptions). It should not be possible to crash the code even when a user provides an unexpected input.
* It is required that your module is properly documented, and that every function has a meaningful docstring. Each function must explain its own input arguments and their types, the returned values and their types, and any exceptions that may be raised. There should be no room for misinterpretation. Check that print(functionname.\_\_doc\_\_) returns a proper docstring. Essentially, each function should be useable from the docstring alone, without reading the code.
* Further, marks will be given on code effciency. Have you solved a problem using 100 lines of code, although we have learned a very straightforward way which would only require 2 lines? You may lose marks in this case.

The rough split (subject to scaling) between the total marks is 65% for testing and manual inspection, 15% for documentation, and 20% for code efficiency.

**When and how to submit the coursework?**

The coursework can be completed and submitted as a single Python module named stocktrader.py. The submission is via Blackboard and **the strict deadline is Friday, December 17th, at 1pm.** You can resubmit your coursework as often as you like, but only the last submission counts. **Submissions after the deadline will not be accepted (unless you have a DASS extension).**

**Task 0: Prepare the module and understand the data structures**

[**Download the coursework.zip file**](https://personalpages.manchester.ac.uk/staff/stefan.guettel/py/coursework.zip) and unzip the folder to a convenient location on your computer (e.g., your Desktop). The folder already contains a template for your stocktrader.py module. Your whole coursework project can be completed using this module. You just need to replace the TODO comments with the actual code. Make sure that all code is contained in functions so your module "does not do anything" when it is imported into another Python program.

**Module template:**

*"""*

*stocktrader -- A Python module for virtual stock trading*

*TODO: Add a description of the module...*

*Also fill out the personal fields below.*

*Full name: Peter Pan*

*StudentId: 123456*

*Email: peter.pan.123@student.manchester.ac.uk*

*"""*

**class** **TransactionError**(**Exception**):

**pass**

**class** **DateError**(**Exception**):

**pass**

stocks = {}

portfolio = {}

transactions = []

**def** normaliseDate(s):

*# TODO*

*# TODO: All other functions from the tasks go here*

**def** main():

*# Test your functions here*

*# the following allows your module to be run as a program*

**if** \_\_name\_\_ == '\_\_main\_\_' **or** \_\_name\_\_ == 'builtins':

main()

**CSV data:**

* The coursework folder also contains the files portfolio0.csv and portfolio.csv in the same location as the stocktrader.py file.
* The coursework folder also contains several CSV files with historic stock prices of different companies.

The module stocktrader uses three essential data structures as explained below.

**The stocks dictionary**

The dictionary stocks stores historic financial data that your module can work with. The data for stocks is located in the coursework, with each file of the form SYMBOL.csv corresponding to a particular company. Every entry in the stocks dictionary is a key-value pair. Each key is a string corresponding to a symbol and the value is again a dictionary.

The dictionaries in stocks contain key-value pairs where the key (a string) corresponds to a date in the form YYYY-MM-DD and the value is a list of floating point numbers [ Open, High, Low, Close ] corresponding to the prices of a stock at that particular date.

Here is an excerpt of a valid stocks dictionary containing data for the symbol EZJ (easyJet plc) and SKY (Sky plc):

stocks = {

'EZJ' : {

'2012-01-03' : [435.273010, 435.273010, 425.050995, 434.835999],

'2012-01-04' : [434.618011, 434.618011, 423.273010, 428.072998],

'2012-01-05' : [430.472992, 430.472992, 417.273010, 418.364014],

#...

},

'SKY' : {

'2012-01-03' : [751.000000, 755.500000, 731.500000, 742.000000],

'2012-01-04' : [740.000000, 741.125000, 718.000000, 730.000000],

'2012-01-05' : [733.500000, 735.500000, 719.500000, 721.000000],

#...

},

}

The interpretation of this data at an example is as follows: on the 4rd of January 2012 the price of a Sky share ranged between £718.00 (the "low") and £741.125 (the "high").

**The portfolio dictionary**

portfolio is a dictionary that represents our capital at a given date. Our capital is the combination of cash and the shares that you hold. The keys in portfolio are strings date, cash, and arbitrarily many symbols. The respective values are the date of the last transaction performed on the portfolio in the form YYYY-MM-DD, the cash amount as a floating point number, and the integer number of shares held for each symbol.

Here's an example of a valid portfolio dictionary:

portfolio = {

'date' : '2013-11-27',

'cash' : 12400.45,

'EZJ' : 10

}

The interpretation of this is as follows: on the 27th of November 2013 we have £12,400.45 in cash and we own 10 shares of easyJet. We could now look up in the stocks dictionary that the *low price* of easyJet on that day is £1426.00. Hence, if we sold all 10 easyJet shares on this day, we'd have £12,400.45 + 10 x £1426.00 = £26,660.45 of cash and no more EZJ shares. In this case the portfolio dictionary would only have two keys, date and cash.

**The transactions list**

transactions is a list of dictionaries, with each dictionary corresponding to a buy/sell transaction on our portfolio. Here is an example of a valid transactions list:

transactions = [

{ 'date' : '2013-08-11', 'symbol' : 'SKY', 'volume' : -5 },

{ 'date' : '2013-08-21', 'symbol' : 'EZJ', 'volume' : 10 }

]

The interpretation of this is as follows: on 11th of August 2013 we **sold** 5 shares of Sky (because volume is negative), and on the 21st of August 2013 we **bought** 10 shares of easyJet (because volume is positive). The value of volume is always an integer, and the date values are chronological: while there can be two or more neighboring list entries in transactions having the same date, the following ones can never have an earlier date. This makes sense as the time order of transactions is important.

**Task 1: function normaliseDate(s)**

Write a function normaliseDate(s) which takes as input a string s and *returns* a date string of the form YYYY-MM-DD. The function should accept the following input formats: YYYY-MM-DD, YYYY/MM/DD and DD.MM.YYYY, where DD and MM are integers with one or two digits (the day and/or month can be given with or without a leading 0), and YYYY is a four-digit integer. The function converts all of these formats to YYYY-MM-DD.

If the conversion of the format fails (i.e., it is not exactly in any of the formats specified above), the function raises a DateError exception.

Note that this function is only about *conversion of formats*, and there is no need to check whether the date YYYY-MM-DD actually exists.

**Example:** Both normaliseDate('08.5.2012') and normaliseDate('2012/05/8') should return the string 2012-05-08, while normaliseDate('8.5.212') should raise a DateError exception.

**Task 2: function loadStock(symbol)**

Write a function loadStock(symbol) which takes as input a string symbol and loads the historic stock data from the corresponding CSV file into the dictionary stocks. The function does not need to return anything as the dictionary stocks is in the outer namespace and therefore accessible to the function. Note that symbol only contains the symbol of a stock, not the full file name. So, for example, if symbol = # then the file to be loaded has the name fname = symbol + '.csv'.

The stock data CSV files are of the following format:

* the first line is the header and can be ignored
* every following line is of the comma-separated form Date,Open,High,Low,Close,AdjClose,Volume, where Date is in any of the formats accepted by the function normaliseDate(), and all other entries are floating point numbers corresponding to prices and trading volumes. Note that only the first values are relevant for filling the stocks dictionary and AdjClose,Volume can be ignored.

If the file given by symbol cannot be opened (as it is not found), a FileNotFoundError exception should be raised.

If a line in the CSV file is of an invalid format, a ValueError exception should be raised.

**Example:** loadStock('EZJ') should load the easyJet data from the file EZJ.csv into the dictionary stocks, whereas loadStock('XYZ') should raise a FileNotFoundError exception.

**Fallback:** If, for some reason, you struggle to implement this function, you can copy-and-paste the above example stocks dictionary (containing only three days of EZJ and SKY shares) into your stocktrader.py file and use this for testing the other functions.

**Task 3: function loadPortfolio(fname='portfolio.csv')**

Write a function loadPortfolio(fname) which takes as input a string fname corresponding to the name of a CSV file (assumed to be in the same directory as stocktrader.py). The function loads the data from the file and assigns them to the portfolio dictionary, with all entries of the form described above (including the date!).

Make sure that the portfolio dictionary is emptied before new data is loaded into it, and that the list transactions is emptied as well.

The function does not need to return anything as the dictionary portfolio is in the outer namespace and therefore accessible to the function. If no filename is provided, the name portfolio.csv should be assumed.

As the loadPortfolio(fname) function goes through the list of shares in the CSV file, it should use the function loadStock(symbol)from Task 2 to load the historic stock data for each symbol it encounters.

A valid portfolio CSV file is of the following form:

* the first line contains the date of the portfolio in any of the forms accepted by the function normaliseDate()
* the second line contains the cash in the portfolio, a nonnegative floating point number
* the following lines (if present) are of the form symbol,volume. Here, symbol is the symbol of a stock and volume is an integer corresponding to the number of shares.

Here is an example of a portfolio.csv file:

2012/1/16

20000

SKY,5

EZJ,8

If the file specified by fname cannot be opened (as it is not found), a FileNotFoundError exception should be raised.

If a line in the file is of an invalid format, a ValueError exception should be raised. The coursework folder contains a faulty portfolio\_faulty.csv file which you can use for testing.

**Example:** loadPortfolio() should empty the dictionary portfolio and the list transactions, and then load the data from portfolio.csv into the dictionary portfolio, as well as the corresponding stock data into the dictionary stocks.

**Fallback:** If, for some reason, you struggle to implement this function, you can copy-and-paste the above example portfolio dictionary (containing 10 shares of EZJ) into your stocktrader.py file and use this for testing the other functions.

**Task 4: function valuatePortfolio(date, verbose)**

Write a function valuatePortfolio(date, verbose) with two *named parameters* date and verbose. The function valuates the portfolio at a given date and *returns* a floating point number corresponding to its total value. The parameter date is any string accepted by the normaliseDate() function and when it is not provided, the date of the portfolio is used. The parameter verbose is a Boolean value which is False by default. When the function is called with verbose=True it should still return the total value of the portfolio but also print to the console a table of all capital with the current *low* prices of all shares, as well as the total value.

**Example:** With the portfolio.csv example given in Task 3, a call to valuatePortfolio('2012-2-6') should return the floating point number 27465.372072.... When valuatePortfolio('2012-2-6', True) is called, it should also print a table like this:

Your portfolio on 2012-02-06:

[\* share values based on the lowest price on 2012-02-06]

Capital type | Volume | Val/Unit\* | Value in £\*

-----------------------+--------+-----------+-------------

Cash | 1 | 20000.00 | 20000.00

Shares of SKY | 5 | 686.50 | 3432.50

Shares of EZJ | 8 | 504.11 | 4032.87

-----------------------+--------+-----------+-------------

TOTAL VALUE 27465.37

**Note 1:** For the valuation we use the *low prices* of Sky and easyJet on date, in this case the 6th of February 2012. This is to be on the safe side: if we were selling the shares on that day, we would *at least* get those prices.

**Note 2:** A call to valuatePortfolio(date) should raise DateError exceptions in two cases:

* When date is earlier than the date of the portfolio, there might have been transactions afterwards and we no longer know what was the value back then. For example, valuatePortfolio('2012-1-3') should fail if the portfolio is already dated 2012-02-06.
* When date is not a trading day (e.g., a bank holiday or weekend) the CSV files will not contain any price for it and hence we cannot look up the values of shares. For example, valuatePortfolio('2012-2-12') should fail for that reason.

**Task 5: function addTransaction(trans, verbose)**

Write a function addTransaction(trans, verbose) which takes as input a dictionary trans corresponding to a buy/sell transaction on our portfolio and an optional Boolean variable verbose (which is False by default). The dictionary trans has three items as follows:

* the key date whose value is any string accepted by the function normaliseDate()
* the key symbol whose value is a string corresponding to the symbol of a stock
* the key volume whose value is an integer corresponding to the number of shares to buy or sell.

**Example:** Here are two valid transaction dictionaries, the first one for selling 5 shares of Sky on 12th of August 2013, and the second for buying 10 shares of easyJet on the 21st of August 2013.

{ 'date' : '2013-08-12', 'symbol' : 'SKY', 'volume' : -5 },

{ 'date' : '21.08.2013', 'symbol' : 'EZJ', 'volume' : 10 }

A call to the addTransaction(trans) function should

* update the portfolio value for cash
* insert, update, or delete the number of shares
* update the date of portfolio to the date of the transaction
* append trans to the list transactions.

To be on the safe side, we always assume to **sell at the daily low price** and **buy at the daily high price.**

The addTransaction(trans) function does not need to return any values as both portfolio and transactions are available in the outer namespace and therefore accessible to the function.

If the optional Boolean parameter verbose=True, the function should print to the console an informative statement about the performed transaction.

**Example:** The call

addTransaction({ 'date':'2013-08-12', 'symbol':'SKY', 'volume':-5 }, True)

should print something like

> 2013-08-12: Sold 5 shares of SKY for a total of £4182.50

Available cash: £24182.50

**Exceptions:** The function addTransaction(trans) may fail for several reasons, in which case both portfolio and transactionsshould remain unchanged and the appropriate exception should be raised:

* if the date of the transaction is earlier than the date of the portfolio, a DateError exception should be raised (i.e., one cannot insert any transactions prior to the last one)
* if the symbol value of the transaction is not listed in the stocks dictionary, a ValueError exception should be raised
* if the volume is such that we either do not have enough cash to perform a buying transaction or we do not have enough (or none at all) shares to perform a selling transaction, a TransactionError exception should be raised.

**Task 5.5: Take a coding break and test**

When you arrive here, it's time to take a break and test your code extensively. You should now be able to use your module to load portfolio and stock data files into your computers memory, print the value of your portfolio, and perform buying and selling transactions. For example, if you create a test\_stocktrader.py file (or use the one in the coursework.zip folder) the following code should now work:

**import** **stocktrader** **as** **s**

s.loadPortfolio()

val1 = s.valuatePortfolio(verbose=True)

trans = { 'date':'2013-08-12', 'symbol':'SKY', 'volume':-5 }

s.addTransaction(trans, verbose=True)

val2 = s.valuatePortfolio(verbose=True)

**print**("Hurray, we have increased our portfolio value by £{:.2f}!".format(val2-val1))

The console output should be something like this:

Your portfolio on 2012-01-16:

[\* share values based on the lowest price on 2012-01-16]

Capital type | Volume | Val/Unit\* | Value in £\*

-----------------------+--------+-----------+-------------

Cash | 1 | 20000.00 | 20000.00

Shares of SKY | 5 | 677.50 | 3387.50

Shares of EZJ | 8 | 429.93 | 3439.42

-----------------------+--------+-----------+-------------

TOTAL VALUE 26826.92

> 2013-08-12: Sold 5 shares of SKY for a total of £4182.50

Available cash: £24182.50

Your portfolio on 2013-08-12:

[\* share values based on the lowest price on 2013-08-12]

Capital type | Volume | Val/Unit\* | Value in £\*

-----------------------+--------+-----------+-------------

Cash | 1 | 24182.50 | 24182.50

Shares of EZJ | 8 | 1327.35 | 10618.80

-----------------------+--------+-----------+-------------

TOTAL VALUE 34801.30

Hurray, we have increased our portfolio value by £7974.38!

Before moving on to the final tasks, make sure that all the functions of Tasks 1-5 work as expected, that all calculations are correct, and that the appropriate exceptions are raised whenever a problem occurs. The following tasks will rely on these core functions.

**Task 6: function savePortfolio(fname)**

Write a function savePortfolio(fname) that saves the current dictionary portfolio to a CSV file with name fname (a string). The file should be saved in the same directory as the stocktrader.py module. (This is the default location for saving files and you should not need to worry about selecting the directory.) If no filename is provided, the name portfolio.csv should be assumed.

The function does not need to return anything.

**Example:** savePortfolio('portfolio1.csv') should save the values of the portfolio dictionary to the file portfolio1.csv.

**Task 7: function sellAll(date, verbose)**

Write a function sellAll(date, verbose) that sells all shares in the portfolio on a particular date. Here, date is an optional string of any format accepted by the function normaliseDate() and verbose is an optional Boolean variable which is False by default. If verbose=True all selling transactions are printed to the console. If date is not provided, the date of the portfolio is assumed for the sell out.

**Note:** You should be able to use a simple loop with the function addTransaction(trans, verbose) for this task.

**Task 8: function loadAllStocks()**

Write a function loadAllStocks() which loads all stocks into the dictionary stocks. The function does not need to return anything as the dictionary stocks is in the outer namespace and therefore accessible to the function. The corresponding stock CSV files are assumed to be in the same folder as stocktrader.py, and they are assumed of the form XYZ.csv where XYZ is a string containing only capital letters.

If the loading of one of the files fails, this file should simply be ignored. The coursework.zip folder contains a faulty file PPB.csv which you can use for testing this.

**Note:** You should be able to use a simple loop with the function loadStock(symbol) for this task. You may want to use the [os module](https://docs.python.org/3.6/library/os.html)for getting a list of all files in a directory.

**Task 9: function tradeStrategy1(verbose)**

Write a function tradeStrategy1(verbose) that goes through all trading days in the dictionary stocks and buys and sells shares automatically. The strategy is as follows:

* The earliest buying decision is either on the date of the portfolio or the tenth available trading day in stock (whichever is later)
* At any time, we buy the highest possible volume of a stock given the available cash.
* When shares have been bought, no other shares will be bought until all shares from the previous buying transaction are sold again.
* All shares from a previous buying transaction are sold at once, so it's a simple "buy as much as possible & sell all" procedure.
* If shares are being sold on trading day j we will only consider buying new shares on the following trading day, j+1.

Assume that j is the index of the current trading day, then we will find the stock to buy as follows:

* For each stock s available in stocks evaluate the quotient

Q\_buy(s,j) = 10\*H(s,j) / (H(s,j) + H(s,j-1) + H(s,j-2) + ... + H(s,j-9))

where H(s,j) is the *high price* of stock s at the j-th trading day. Note that Q\_buy(s,j) is large when the *high price* of stock son trading day j is large compared the average of all previous ten high prices (including the current). This means we might enter a phase of price recovery.

* Find the maximal quotient Q\_buy(s,j) among all stocks s and buy a largest possible volume v of the corresponding stock on trading day j. (It might not be possible to buy any as there might not be enough cash left; in this case do nothing on trading day j and move to the next. If two or more stocks have exactly the same quotient, take the one whose symbol comes first in lexicographical order.)
* Note that, as usual, our buying decision is based on the *high price.*

If we have automatically bought v shares of a stock s on trading day j, then from trading day k = j+1 onwards we will consider selling all of it as follows:

* On trading day k = j+1, j+2, ... calculate the quotient

Q\_sell(k) = L(s,k) / H(s,j),

where L(s,k) corresponds to the *low price* of stock s on trading day k. This quotient is high if the current *low value* of the stock is large compared to the *high value* to which we bought it.

* Sell all v shares of s on day k if Q\_sell(k) < 0.7 (we already lost at least 30%, let's get rid of these shares!) or if Q\_sell(k) > 1.3 (we made a profit of at least 30%, time to cash in!).

**Notes:**

* For solving this task it might be useful to first extract a list of all trading days from the stocks dictionary:

lst = [ '2012-01-03', '2012-01-04', ..., '2018-03-13' ]

You can assume that all loaded stocks in the stocks dictionary can be traded on exactly the same days, and that there is at least one stock in that dictionary.

* All buying and selling transactions should be performed using the addTransaction(trans, verbose) function from Task 5. The verbose parameter of tradeStrategy1(verbose) can just be handed over to addTransaction(trans, verbose).

**Example:** The following code loads a portfolio of £20,000 cash (and no shares) on the 1st of January 2012, runs the tradeStrategy1(verbose=True) until the end of available data, and valuates the portfolio on the 13th of March 2018.

s.loadPortfolio('portfolio0.csv')

s.loadAllStocks()

s.valuatePortfolio(verbose=True)

s.tradeStrategy1(verbose=True)

s.valuatePortfolio('2018-03-13', verbose=True)

The console output is as follows:

Your portfolio on 2012-01-01:

[\* share values based on the lowest price on 2012-01-01]

Capital type | Volume | Val/Unit\* | Value in £\*

-----------------------+--------+-----------+-------------

Cash | 1 | 20000.00 | 20000.00

-----------------------+--------+-----------+-------------

TOTAL VALUE 20000.00

> 2012-01-16: Bought 29 shares of PRU for a total of £19517.00

Remaining cash: £483.00

> 2012-11-21: Sold 29 shares of PRU for a total of £25520.00

Available cash: £26003.00

> 2012-11-22: Bought 37 shares of EZJ for a total of £25696.50

Remaining cash: £306.50

> 2013-01-25: Sold 37 shares of EZJ for a total of £33633.00

Available cash: £33939.50

> 2013-01-28: Bought 35 shares of EZJ for a total of £33103.00

Remaining cash: £836.50

> 2013-05-21: Sold 35 shares of EZJ for a total of £43120.00

Available cash: £43956.50

> 2013-05-22: Bought 34 shares of EZJ for a total of £43905.22

Remaining cash: £51.28

> 2014-01-22: Sold 34 shares of EZJ for a total of £58208.00

Available cash: £58259.28

> 2014-01-23: Bought 18 shares of BATS for a total of £57456.00

Remaining cash: £803.28

> 2016-04-08: Sold 18 shares of BATS for a total of £74853.00

Available cash: £75656.28

> 2016-04-11: Bought 68 shares of SMIN for a total of £75140.00

Remaining cash: £516.28

> 2016-09-29: Sold 68 shares of SMIN for a total of £98532.00

Available cash: £99048.28

> 2016-09-30: Bought 108 shares of SKY for a total of £98594.49

Remaining cash: £453.79

> 2018-02-27: Sold 108 shares of SKY for a total of £140400.00

Available cash: £140853.79

> 2018-02-28: Bought 104 shares of SKY for a total of £140192.00

Remaining cash: £661.79

Your portfolio on 2018-03-13:

[\* share values based on the lowest price on 2018-03-13]

Capital type | Volume | Val/Unit\* | Value in £\*

-----------------------+--------+-----------+-------------

Cash | 1 | 661.79 | 661.79

Shares of SKY | 104 | 1316.00 | 136864.00

-----------------------+--------+-----------+-------------

TOTAL VALUE 137525.79

Not bad! In a bit more than six years we have multiplied our initial investment of £20,000 by a factor of almost seven.



**Task 10 (optional): function tradeStrategy2(verbose)**

When you modify the start date of your portfolio, or remove some of the stocks from the available data, you will see that tradeStrategy1() is not very robust and we've just been lucky to make so much profit. Also, it is kind of strange that we repeatedly sell and then immediately buy the same stock. This doesn't seem to make much sense. In some cases, this strategy results in big loses.

Can you write your own tradeStrategy2(verbose) function that performs better?

The conditions are as follows:

* tradeStrategy2(verbose) should only perform transactions via the function addTransaction(trans,verbose)
* tradeStrategy2(verbose) itself should not modify the dictionaries portfolio, stocks and neither the list transactions
* tradeStrategy2(verbose) should work on all valid stock and portfolio dictionaries, with different companies than the provided ones and over different time ranges
* on any trading day the buy/sell decision is only based on the price data [ Open, High, Low, Close ] available up to that day (i.e., no information from the future is used for making decisions); no other (external) data should be used
* tradeStrategy2(verbose) can (and probably should) "diversify" to reduce the risk, which means that any time it can decide to have shares of more than one stock in the portfolio, or no shares at all
* tradeStrategy2(verbose) is not restricted to a single buying or selling transaction per day, and even allowed to buy and sell shares of a stock on the same day (which would however incur a loss because buying is at the daily high price, and selling at the low price)
* tradeStrategy2(verbose) should not use any "randomness", i.e., two calls to the function with exactly the same data should result in an identical list of transactions
* tradeStrategy2(verbose) should run reasonably fast, not longer than a couple of seconds on the provided data.

**This task is optional and will not be part of the formal assessment. However, if your module contains a working `tradeStrategy2(verbose)` function, it will enter a \*trading competition.\* In this competition I will test your strategy on different stock data and over shorter and longer time intervals. The winners with the three best-performing strategies will enter a "Hall of Fame" published on the course website in early 2022.**

**End of coursework.**