A02 Specification

Version History

1. 2017-04-27: Initial Draft
2. 2017-04-29: Fleshed out tasks

Outcomes

This assignment will test your understanding of file input/output, loops, conditionals and functions.

Submission

The Assignment is due on the **17th of May** at **10pm**. Submission will be via Moodle. If for any reason there are issues with the submission to Moodle it should be emailed to nick.reynolds@mq.edu.au before the submission time. Feel free to email me to discuss solutions.

Task

You will be writing a stock trading application that will read in some stock data and buy/sell it based on predefined rules. The functionality is split into three sections below, it may look daunting but there's just lots of examples and explainations.

1. **File Input**
   a. `read_stock_file(filename)`
      This function takes in a filename, such as 'stocks.csv', you need to use the filename variable passed in to read in the file and return the file contents. Your contents should be returned as a list of dictionaries.
      e.g. `[{ 'Company': 'AAA', 'Price': '$19.29' ...etc...}]`
      Hint: Use the csv.DictReader

   b. `parse_stocks(stocks)`
      The purpose of this function is to change the stock data to be the right data types. The 'stocks' variable is going to be the same list that you read in in the method above.

      If you notice above the 'Price' of the stock is '$19.29' as a string, this should be converted to 19.29 as a float/decimal number. Similarly, the 'Units Available' will need to be changed from a string to an integer. If you're feeling a bit more advanced you may want to change the date to be a proper python date format, but that's not required.

      Hint: You'll want to loop through the stocks, changing the types of Price and Units Available. Recall that you access dictionary properties like this `stock['Price']`

2. **The Buffett Class** (Named after one of the most successful investors in the world More)
   Buffett will keep track of purchases, and how much money they have.
   This class has two properties (if you missed the lesson on classes, make sure you revise it!), it has Money and Purchases. Money is a float/decimal, while purchases is going to be a dictionary of purchases. The buffett class is initialized with an amount of money.

   a. `buy(self, company, quantity, price)`
      This function handles Buffett buying a stock. Company will be a company name e.g. 'Company A', quantity will be a number, price will be a decimal number.

      This function involves a few things:
      - Needs to check if they have enough money to afford the stock. e.g. Check if their money is >= the quantity multiplied by the price. If they can't afford it we do nothing.
      - Decrease their money by the right amount if the above condition is met.
      - Add the company and quantity to the purchases (self.purchases) dictionary.

      Example: If Buffett has no purchases and $200.
      They then `buy('Company A', 40, 2.50)`, it's going to cost $100 (2.50 * 40). So I would check that they can afford it (they can). Then subtract the cost. So Buffett now has $100 and add it to my dict of purchases, which would now look like this:
      `{ 'Company A': 40 }`
      They then `buy('Company B', 20, 1)`, it's going to cost $20 (1 * 20). So I would check if they can afford it (they can). Then subtract the cost. So Buffett now has $80 and add it to my dict of purchases, which would now look like this:
      `{ 'Company A': 40, 'Company B': 20 }`
      If they then `buy('Company A', 5, 2)`, they can afford it, we subtract money (now at $70) and update the dict of purchases, which now looks like this:
      `{ 'Company A': 45, 'Company B': 20 }`

      Hint: Remember that self.money is how we access the classes properties.

   b. `how_many_can_i_sell(self, company, potential_sell_quantity)`
      This function handles the possibility of Buffett selling some stock. Company is again a company name, potential_sell_quantity is a number of stocks that someone on the market is buying (i.e. the max number we can sell).
      The purpose of this function is to calculate how many stocks for that company we can sell.
      Examples:
      - If Buffett has no purchases: `how_many_can_i_sell('AAA', 10)`, would return 0 as we do not have any stocks to sell.
      - Purchases = `{ 'AAA': 40 }`. `how_many_can_i_sell('AAA', 10)` would return 10, as we have 40 stocks to sell of AAA, but someone is only buying 10.
      - Purchases = `{ 'AAA': 40 }`. `how_many_can_i_sell('AAA', 50)` would return 40, as we have 40 stocks to sell of AAA, so we can't sell 50, best we can do is 40.
      - Purchases = `{ 'AAA': 40 }`. `how_many_can_i_sell('BBB', 50)` would return 0, as we do not have any BBB stocks.

c. `sell(self, company, quantity, price)`
   This will handle selling of a stock, the parameters are the same as the buy function. This function needs to do a few things:
   - Check that we have the stock to sell (just use `how_many_can_i_sell`)
   - Remove that quantity from our purchases dictionary
   - Add to our money (quantity * price)

   Examples:
   - Money = 200, Purchase `= { 'AAA': 40 }`
     `sell('BBB', 5, 10)` would do nothing, as we have no 'BBB' to sell
   - Money = 200, Purchase `= { 'AAA': 40 }`
     `sell('AAA', 5, 10)` would increase our money by $50 (5*10) and change our purchases dict to remove 5 of 'AAA', `{ 'AAA': 35 }`

3. The Open Market
   a. `open_market(money, stocks)`
      This is where we will process the stocks and decide what to do with our trading.
      The money variable is just how much money Buffett will start with. Code is already in place that will create an instance of Buffett (called warren) and loop through each of the stocks. All you need to do is add the logic for when we want to buy and sell, based on the rules below.

Trading Rules

Buffett has a few rules on how he trades stocks for each company:

- AAA
  - Buy whenever the price is <= $20
  - Sell whenever the price is >= $30
- Company BBB
  - Buy whenever the price is <= $50
  - Sell whenever the price is >= $90
- Company CCC
  - Buy whenever the price is <= the minimum of the previous day
  - Sell whenever the price is >= the maximum of the previous day
- Company DDD
  - Buy whenever the price is <= the average price on the previous day
  - Sell whenever the price is >= the maximum of the previous day

Company CCC and DDD are much harder than the other two. Feel free to add your own test data file. There are no automated tests for CCC and DDD, if you're competent enough to do them, then you're competent enough to test them yourself too!

Starter Pack

Please download the zip here. It includes two files:

- A02.py
  This is the file you will do your work in. Each function has already been defined for you. Note that you can remove the “pass” keyword once you have written your body.
- Test.py
  This is the testing file that will check your work, running it will tell you what failed and why. Do not modify this file!

Hints

- You may not have learned everything you need for this assignment in class. A big part of software development is learning how to find resources online, go back and look through my reference material.
- You'll notice that when we pass a list to a function and add a value to it, the original list will change as well (unlike strings or integers!). This is called mutability, see the above point!

Contact me with any questions or concerns. I'm here to help: nick.reynolds@mq.edu.au