CS1010X — Programming Methodology
National University of Singapore

Practical Examination

4 June 2016

Time allowed: 2 hours

Instructions (please read carefully):

1. This is **an open-book exam**.
2. This practical exam consists of **three** questions. The time allowed for solving this quiz is **2 hours**.
3. The maximum score of this quiz is **30 marks** and you will need to answer all three questions to achieve the maximum score. Note that the number of marks awarded for each question **IS NOT** correlated with the difficulty of the question.
4. You are advised to attempt all questions. Even if you cannot solve a question correctly, you are likely to get some partial credit for a credible attempt.
5. Your answers should be submitted on Coursemology.org. You will be allowed to run some public tests cases to verify that your submission is correct. Note that you can run the test cases on Coursemology.org **up to 5 times** because they are only for checking that your code is submitted correctly. You are expected to test your own code for correctness using IDLE and not on Coursemology.org. Do however ensure that you submit your answers correctly by running the test cases at least once.
6. In case there are problems with Coursemology.org, you are also provided with the template `practical-template.py` to work with. If Coursemology.org fails, you will be required to rename your file `practical-exam-<mat no>.py` where `<mat no>` is your matriculation number and upload the file to the Practical Exam folder in IVLE Workbin. You should only upload one file. If you upload multiple files, we will choose one at random for grading.
7. Please note that it shall be your responsibility to ensure that your solution is submitted correctly to Coursemology (or uploaded to IVLE, if there are problems) at the end of the examination. Failure to do so will render you liable to receiving a grade of **ZERO** for the Practical Exam, or the parts that are not uploaded correctly. You have been warned.
8. Please note that while sample executions are given, it is not sufficient to write programs that simply satisfy the given examples. Your programs will be tested on other inputs and they should exhibit the required behaviours as specified by the problems to get full credit. There is no need for you to submit test cases.

GOOD LUCK!

Question 1 : Warm Up [10 marks]

A. [**Number Substring**] Write a function `contains` that takes in two positive integers *a* and *b* and returns `True` if *a* contains the digits for *b* in the same sequence. Or, `False` otherwise. You may assume that neither *a* nor *b* contains the digit 0. [5 marks]

Sample execution:

```
>>> contains(123,123)
True

>>> contains(1234,123)
True

>>> contains(4123,123)
True

>>> contains(123555,123)
True

>>> contains(123555,23)
True

>>> contains(1243555,123)
False
```

B. [**Repeated Digits**] Write a function `countlongeststreak` that takes in a positive number *a* and returns the maximum length of a sequence of the same digit within the number *a*. You may assume that *a* does not contain the digit 0. [5 marks]

Sample execution:

```
>>> count_longest_streak(123456789)
1

>>> count_longest_streak(111123456789)
4

>>> count_longest_streak(123444456789)
4

>>> count_longest_streak(11112211111)
5
```

Question 2 : Stock Market Analysis [10 marks]

Python is widely regarded as the academic tool of choice when it comes to data analysis. In this problem, you will apply Python to the analysis of stock prices. The data we will be using comes from the daily stocks ticker for the S&P500. Each data file contains the time series information for one stock with the following information:

- Date : Day of the stock ticker
- Time : Time in integer
- Open : Opening price of the stock
- High : Highest price reached
- Low : Lowest price reached
- Close : Closing price of the stock
- Volume : Number of shares traded

Note that opening price of the next day and the closing price of the previous day might not match exactly due to after hours trading. Take note that the data types of the Date and Time column are integers whereas the others are floats. In Figure 1, we show a diagram visualizing the opening prices for Stock TAP.

A. [**Minimum Price**] Write a function `min_stock` that takes in a data file for a stock, a start date and an end date, and returns the minimum price for the stock for the stated duration (inclusive). You can assume that the data file is in chronological order, i.e. the entries for the earlier dates will occur before the later dates in the file. If the starting date is not within the dataset, we will choose the date closest to the start date. Likewise for the end date. This is known as clamping the date ranges. If there are no data within the given date range at all, the function should return `None`. [3 marks]

Sample execution:

```
>>> min_stock("table_tap.csv", 19980218, 19980309)
11.6277

>>> min_stock("table_tap.csv", 19980102, 20130809)
11.0139

>>> min_stock("table_apa.csv", 19980102, 20130809)
6.91758
```

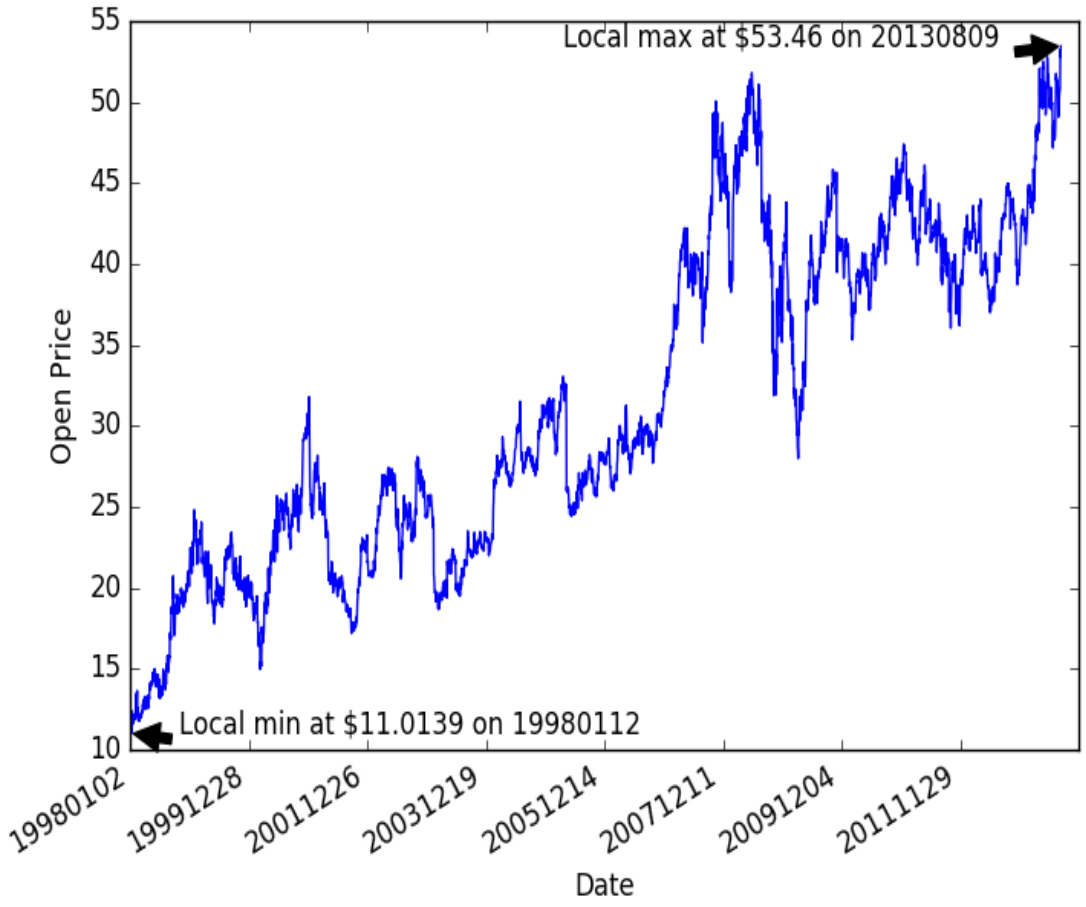


Figure 1: Opening Stock Price of Stock TAP from 1999 to 2013

B. [**Understanding Volatility**] It is important to understand how much stock prices fluctuate. Write a function `average_daily_variation` that takes in a data file for a stock, a start date and an end date (inclusive), and returns the average range of price fluctuations (i.e. average of the difference between the highest and lowest price for each day) over the stated duration. If there are no data within the given date range at all, the function should return `None`. [3 marks]

Sample execution:

```
>>> average_daily_variation("table_tap.csv", 19980218, 19980309)
0.47104285714285715

>>> average_daily_variation("table_tap.csv", 19980102, 20130809)
0.668375878757002

>>> average_daily_variation("table_apa.csv", 19980102, 20130809)
1.6702794752929195
```

C. [**Optimal Trade (One Shot)**] Now for what really matters. Write a function `tradestock` that takes in a data file for a stock, a start date and an end date (inclusive), and returns the best single buy-sell trade that can be made in this period. The profit equals to the absolute difference between the highest price and the lowest price. In other words, suppose you can buy one share and sell that share during this period, what is the maximum profit? Note that you cannot sell a share before you buy it and you are allowed only one trade. For simplicity, you may assume that the lowest price will come before the highest price between the opening and closing price in a single day of trading data. If there are no data within the given date range at all, the function should return `None`. [4 marks]

Hint: Instead of just working with the 2 data files provided, you might want to create your own data file so that you can check that your algorithm is correct. Try out some simple cases.

Sample execution:

```
>>> trade_stock("table_tap.csv", 19980218, 19980309)
1.3630999999999993

>>> trade_stock("table_tap.csv", 19980102, 20130809)
42.7361

>>> trade_stock("table_apa.csv", 19980102, 20130809)
136.89742

>>> trade_stock("table_apa.csv", 20180101, 20181230) # Answer is None
```

Question 3 : Saving the National Carrier [10 marks]

Singapore Airlines isn't doing so well in recent years and so you decided to join the airlines upon graduation. Your first job is in the flight planning department and your boss has tasked you to create a new flight `Map` object.

The `Map` object supports 5 methods:

1. `addnode` takes a string and creates a new node in the flight map. If there is already a node in the map with the specified name, this method does nothing.

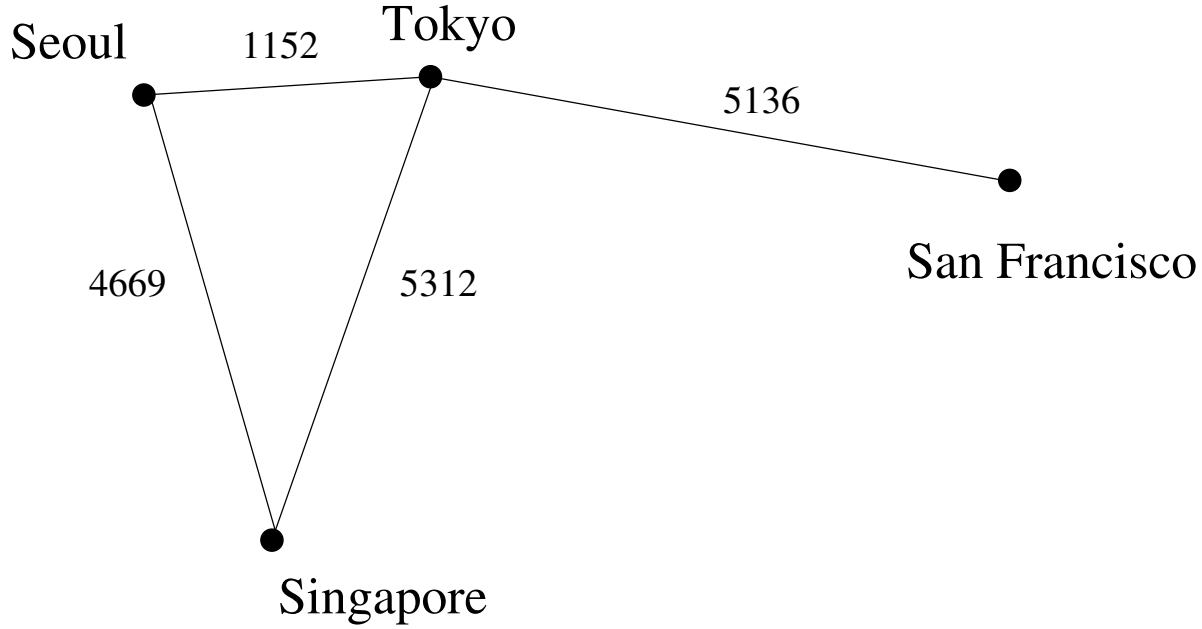
2. `addlink` takes 2 nodes (strings) and a number denoting distance and creates a link between the 2 nodes with the specified distance. If either of the 2 nodes are not valid nodes, `False` is returned; otherwise, if the operation is successful, this method will return `True`.

3. `get_distance` takes 2 nodes (strings) and returns the distance between the 2 nodes. If either of the 2 nodes are not valid nodes, `False` is returned instead.

4. `get_paths` takes 2 nodes (strings) and returns a list of the possible paths between the 2 nodes. A path is a list of the names of the nodes between the 2 specified nodes, including the 2 nodes. Note that a path will not contain any repeated nodes. If either of the 2 nodes are not valid nodes or if the 2 nodes are not connected, an empty list is returned instead. You can return the paths in any order.

5. `shortest_path` takes 2 nodes (strings) and returns the length of the shortest path between them. If either of the 2 nodes are not valid nodes or if the 2 nodes are not connected, `False` is returned instead.

Please implement the `Map` class. You are welcome to define additional helper classes and methods if you think they would be helpful. Consider the very simple flight map shown below.



Sample execution:

```
>>> m = Map()
>>> m.add_node("Singapore")
>>> m.add_node("Seoul")
>>> m.add_node("San Francisco")
>>> m.add_node("Tokyo")
>>> m.add_link("Tokyo","Seoul",1152)
>>> m.add_link("Singapore","Seoul",4669)
>>> m.add_link("Singapore","Tokyo",5312)
>>> m.add_link("Tokyo","San Francisco",5136)

>>> m.get_paths("Singapore","Seoul")
[['Singapore', 'Seoul'], ['Singapore', 'Tokyo', 'Seoul']]

>>> m.get_paths("San Francisco","Seoul")
[['San Francisco', 'Tokyo', 'Seoul'],
['San Francisco', 'Tokyo', 'Singapore', 'Seoul']]

>>> m.get_paths("Seoul","San Francisco")
[['Seoul', 'Tokyo', 'San Francisco'],
['Seoul', 'Singapore', 'Tokyo', 'San Francisco']]

>>> m.shortest_path("Singapore","Seoul")
4669

>>> m.shortest_path("San Francisco","Seoul")
6288

>>> m.shortest_path("Seoul","San Francisco")
6288
```