Lab Test 2 [25 marks]

Coding time: 1 hour and 30 mins

Instructions on how to download the resource files:

1. Download from eLearn a resource file called **your_email_id.zip** under Content → Lab Test 2.

General Instructions:

1. You will take the lab test on your personal laptop.

2. You are not allowed to communicate with anyone or access any network during the test. After downloading the resource files, disable the following connections on your laptop before the test begins: Wi-Fi, Bluetooth, and any other communication devices (e.g. 3G/4G modems).

4. You may refer to any file on your laptop during the test.

5. Make sure your code can generate exactly the same output as we show in the sample runs. You may be penalized for missing spaces, missing punctuation marks, misspelling, etc. in the output.

6. Do not hardcode. We will use different test cases to test and grade your solutions.

7. Follow standard Python coding conventions (e.g. naming functions and variables).

8. Python script file that cannot be executed will NOT be marked and hence you will be awarded 0 marks. You may wish to comment out the parts in your code which cause execution errors.

9. Include your name as author in the comments of all your submitted source files. For example, include the following block of comments at the beginning of each source file you need to submit.

```
# Name: QIAN Tiao She
# Email ID: tiaoshe.qian.2018
```

Instructions on how to submit your solutions:

1. When the test ends, zip up all the files required for submission in a zip archive. The name of the zip archive should be your email ID. For example, if your email is tiaoshe.qian.2018@sis.smu.edu.sg, you should name the archive as **tiaoshe.qian.2018.zip**. You may be penalized for not following our instructions.

2. Once everybody has zipped up his/her files, your invigilator will instruct you to enable your laptop's Wi-Fi and submit your solutions as a single zip file to eLearn Assignments.

Question 1 (Difficulty Level: *) **[6 marks]**

Implement a function called `count_names_with_space` in **q1.py**.

- This function takes in a single parameter called `name_list`.
- `name_list` is a list of strings that represent people's names.
- The function counts how many names in `name_list` contain at least a space and ***returns*** the count.

E.g. 1:

`count_names_with_space(['George W. Bush', 'George Washington', 'Bill', 'Gates', 'Bill Gates'])` returns `3`.

E.g. 2:

`count_names_with_space(['Mark', 'George', 'William'])` returns `0`.

E.g. 3:

`count_names_with_space([])` returns `0`.

Test your code using **q1_test.py**.

Question 2 (Difficulty Level: *) **[6 marks]**

Implement a function called `get_prices_in_range` in **q2.py**.

- This function takes three parameters:
  - `price_list` (type: `list`) : This is a list of float numbers representing prices of products.
  - `low` (type: `float`) : This is the lowerbound of a price range.
  - `high` (type: `float`) : This is the upperbound of a price range.

  You can assume that `high >= low`. You can also assume that all elements in `price_list` as well as `low` and `high` are non-negative values.

- The function ***returns a list*** that contains all the prices between `low` (inclusive) and `high` (inclusive) from `price_list`.

E.g. 1:
`get_prices_in_range([249.99, 24.49, 10.0, 100.0], 0.0, 30.0)` returns `[24.49, 10.0]`.

E.g. 2:
`get_prices_in_range([15.0, 10.9, 5.0], 5.0, 20.0)` returns `[15.0, 10.9, 5.0]`.

E.g. 3:
`get_prices_in_range([108.9, 354.8], 0.0, 100.0)` returns `[]`.

E.g. 4:
`get_prices_in_range([], 0.0, 100.0)` returns `[]`.

Test your code using **q2_test.py**.

Question 3 (Difficulty Level: **) **[5 marks]**

Implement a function called `get_total_transactions_in_month` in **q3.py**.

- The function takes in two parameters:
  - `trans_file` (type: `str`) : This is the name of a file containing transaction data. The format of the file will be explained below.
  - `month` (type: `str`) : This is a string representing a particular month of a particular year in 'mm/yyyy' or 'm/yyyy' format. E.g., `'10/2017'` represents October 2017, `'4/2015'` represents April 2015, and `'04/2015'` also represents April 2015. (Note that the year is always represented with 4 digits. The month is represented with either 1 digit or 2 digits.)

  The `trans_file` contains transaction data. Each transaction is in one row with three columns ***separated by tabs***. The first column is the date of the transaction in 'mm/dd/yyyy' format (where month and day can be either 1 digit or 2 digits). The second column is the transaction amount. The last column is a description of the transaction.

  An example of a `trans_file` looks like the following:

```
10/3/2015  $34.50     phone bill
4/15/2015  $20.00     EZ-link card
12/03/2016 $80.50     birthday gift
9/30/2016  $15.20     meal
04/01/2015 $300.00    holiday air ticket
```

- This function returns the total transacted amount (as a `float`) during the month and year as indicated by the parameter `month`.

E.g. 1:

Suppose the file `'my_transactions.txt'` looks like the file shown above. Then `get_total_transactions_in_month('my_transactions.txt', '04/2015')` returns `320.0` (because there are two transactions during the month of '04/2015' in the file, adding up to $320.00).

E.g. 2:

Given the same file,
`get_total_transactions_in_month('my_transactions.txt', '9/2016')` returns `15.2`.

E.g. 3:

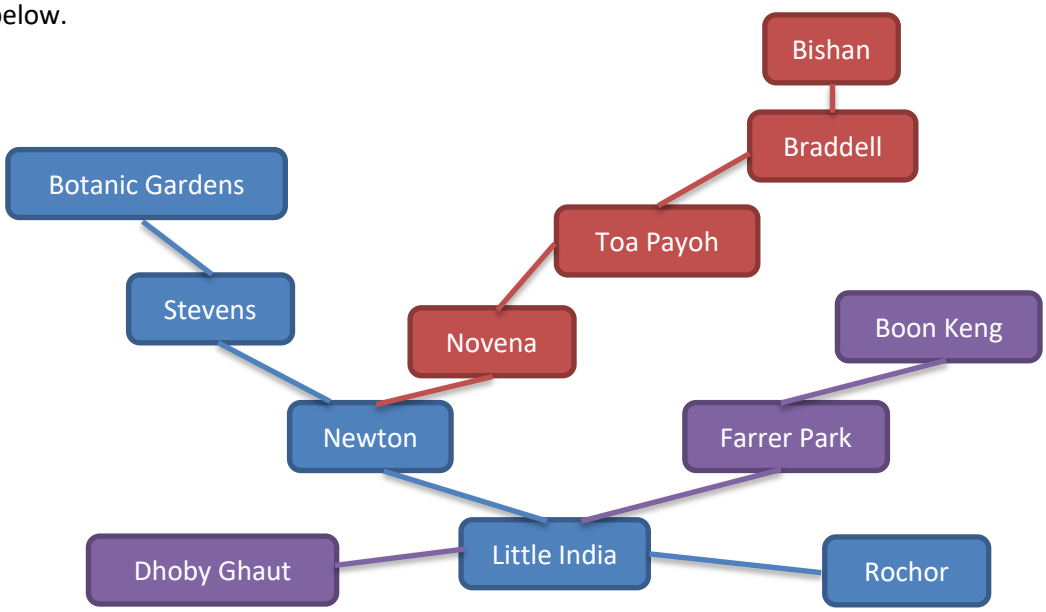
Given the same file,
`get_total_transactions_in_month('my_transactions.txt', '01/2017')` returns `0.0`.

Test your code using **q3_test.py**.

Question 4 (Difficulty Level: *)** **[4 marks]**

Implement a function called `find_stations_within_distance()` inside **q4.py**. The function takes in the following parameters:

- `mrt_map` (type: `list`): This is a list in which each element is a list of strings representing names of MRT stations in the same line. For example, given the following MRT map, its corresponding list representation is given below.




```
[ ['Botanic Gardens', 'Stevens', 'Newton', 'Little India', 'Rochor'],
  ['Newton', 'Novena', 'Toa Payoh', 'Braddell', 'Bishan'],
  ['Dhoby Ghaut', 'Little India', 'Farrer Park', 'Boon Keng'] ]
```

- `orig` (type: `str`): This is the name of an MRT station, representing the origin of a trip.
- `dist` (type: `int`): This is a positive integer representing a distance (in terms of number of stops).

The function ***returns a list*** that contains all the stations that can be reached from the station `orig` within `dist` stops, i.e., these stations are ***no more than*** `dist` stops away from `orig`. The order of the stations in the returned list doesn't matter. Test your code using **q4_test.py**.

- E.g. 1: If `mrt_map` is equal to the list shown above (i.e., it corresponds to the map above), then `find_stations_within_distance(mrt_map, 'Botanic Gardens', 2)` should return either `['Stevens', 'Newton']` or `['Newton', 'Stevens']`.

- E.g. 2: If `mrt_map` is the same as above, `find_stations_within_distance(mrt_map, 'Little India', 1)` should return `['Farrer Park', 'Newton', 'Rochor', 'Dhoby Ghaut']` or a list of these 4 stations in a different order.

- E.g. 3: If `mrt_map` is still the same as above, `find_stations_within_distance(mrt_map, 'Dhoby Ghaut', 3)` should return `['Little India', 'Farrer Park', 'Boon Keng', 'Rochor', 'Newton', 'Stevens', 'Novena']` or a list of these 7 stations in a different order.
  Note: The destination stations `'Rochor', 'Newton', 'Stevens', 'Novena'` are on a different line than `'Dhoby Ghaut'`.

Question 5 (Difficulty Level: *)** **[4 marks]**

Implement a function called `convert_to_list()` inside **q5.py**. The function takes a single parameter called `num_list_str`, which is a string that represents a Python list. Each element of this list is either a positive integer, or a list of positive integers, or a list whose elements are either positive integers or lists of positive integers. In other words, this is a nested list, but the level is no more than 3.

For example, `num_list_str` could be `'[4,5,[6,7],[8,[9,10]],11]'`. But `num_list_str` won't be `'[1,[2,[3,[4]]]]'` because this one has 4 levels of nested brackets.

You can assume that `num_list_str` contains only the following characters: square brackets ('[' and ']'), digits, and commas. You can also assume that `num_list_str` always starts with '[' and ends with ']', and it always represents a well-formatted list as described above (i.e., no missing brackets, no extra comma, etc.).

The function returns a list that corresponds to `num_list_str`, i.e., it converts the string representation of the list to the list itself.

For example, `convert_to_list('[4,5,[6,7],[8,[9,10]],11]')` returns the actual list `[4, 5, [6, 7], [8, [9, 10]], 11]`. See the following expected output to understand the behavior:

- ```
  my_list = convert_to_list('[4,5,[6,7],[8,[9,10]],11]')
  print(len(my_list))
  ```

  Expected Output: `5`

- ```
  my_list = convert_to_list('[4,5,[6,7],[8,[9,10]],11]')
  print(my_list[2])
  ```

  Expected Output: `[6, 7]`

- ```
  my_list = convert_to_list('[4,5,[6,7],[8,[9,10]],11]')
  print(my_list[3][0])
  ```

  Expected Output: `8`

Note: You're not allowed to import any non-standard Python libraries, or to use Python's built-in `eval()` and `exec()` function. You're not allowed to use files.

Test your code using **q5_test.py**.
  ```