Lab Test 2 [30 marks]

Coding time: 2 hours

General Instructions:

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

2. You are not allowed to communicate with anyone or access any network during the test. 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: Lum Ting Wong
# Email ID: lum.ting.wong.2017
```

Instructions on how to submit your solutions:

1. Before the test begins, a thumb-drive will be issued to you. It will contain a folder called `<your_email_id>` with all the resources required for this test. **You should rename the folder to your email id.** For example, if your SMU email is **lum.ting.wong.2017**@sis.smu.edu.sg, you should rename the folder to lum.ting.wong.2017. You need to save your solutions to the same thumb-drive. You may be penalized for not following our instructions.

2. When the test ends, your thumb-drive will be collected for assessment. Make a copy of your solution code on your desktop before submitting your thumb-drive. Only your solutions on the thumb-drive will be considered for assessment.

3. After all the thumb-drives have been collected, your invigilator will instruct you to enable your laptop's Wi-Fi and submit your solutions as a single zip file to eLearn Assignments. This copy serves as a backup and will not be marked.

Question 1 (Difficulty Level: *) **[5 marks]**

Inside `q1.py`, implement a function called `count_long_strings()`. The function takes a list of strings (called `str_list`) as its parameter. The function returns the number of strings in the list that have **at least 5** characters.

- Example #1:
  `str_list`: `["apple", "orange", "pear"]`
  returned value: `2` (because "apple" and "orange" have at least 5 characters but "pear" doesn't)
- Example #2:
  `str_list`: `["", "S M U", "SMU"]`
  returned value: `1` (because only the string `"S M U"` has at least 5 characters)
- Example #3:
  `str_list`: `[]`
  returned value: `0` (because this list doesn't have any string that has at least 5 characters)

Question 2 (Difficulty Level: *) **[5 marks]**

In `q2.py`, implement a function called `get_sum_of_odd_numbers()`. The function takes in a list of integers (called `num_list`) as its parameter. The function returns the sum of all the odd numbers in `num_list`.

If there is no odd numbers in `num_list`, or if `num_list` is empty, the function returns `0`.

- Example #1:
  `num_list`: `[9, 4, 6, 7]`
  returned value: `16` (because there are two odd numbers, 9 and 7, and their sum is 16)
- Example #2:
  `num_list`: `[4, 13, 8, 12]`
  returned value: `13` (because 13 is the only odd number in the list)
- Example #3:
  `num_list`: `[6, 18, 44]`
  returned value: `0` (because there is no odd number in the list)
- Example #4:
  `num_list`: `[]`
  returned value: `0`

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

In `q3.py`, implement a function called `get_postal_codes()`. The function takes in as its parameter a string called `file_name`, which is the name of a file. The file contains names, home addresses and postal codes of some people. Each line of the file contains the information of a single person: (1) name, (2) home address, (3) postal code. The three pieces of information are separated by commas.

You can assume that people's names and home addresses do not contain any comma. You can also assume that the people's names are unique in the file.

For a postal code, it always appears in the following format:

```
Singapore XXXXXX
```

where XXXXXX is the postal code.

The function should **return a list of tuples**, where each tuple contains a person's name and his/her 6-digit postal code. **The order of the names in the returned list should be the same as they appear in the file.**

For example, suppose file_name is 'addresses-1.txt', and 'addresses-1.txt' looks like the following:

```
Jason Wong,444 Ang Mo Kio Ave 10,Singapore 560444
Ted Ng,704 Tampines Street 71,Singapore 520704
Lindy Tan,310 Jurong East Street 32,Singapore 600310
```

Then the function returns the following list:

```
[("Jason Wong", "560444"), ("Ted Ng", "520704"), ("Lindy Tan", "600310")]
```

Note: We may use different files to test your code.

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

Define a function called `evaluate()`. The function takes in a string called `expression` as its parameter. The string is a mathematical expression that contains a sequence of numbers (integers or decimal numbers) separated by plus signs, minus signs, multiplication signs and division signs (i.e., `+`, `-`, `*` and `/`). For example, an input string may look like `"3.5+4*3.0/4-12/5+1.2"`. The function returns the result of this mathematical expression.

You can make the following assumptions:

- The string `expression` is always well formatted, i.e., it is always a valid expression. Also, `expression` is not an empty string.
- There is no space or other character in the input string. I.e., the only characters found in `expression` are the following: digits, dot(`'.'`), plus(`'+'`), minus(`'-'`), asterisk(`'*'`) and slash(`'/'`).
- You can treat all numbers as of float type (although some numbers are integers).
- You do not need to handle unary `+` and `-`. For example, you will not have `'-8+5'` (which means '(-8) + 5') or `'9-+2'` (which means `'9 - (+2)'`) as an expression.

Important note:

- Multiplication and division have **higher precedence** than addition and subtraction.
- You cannot assume the number of decimal places. For example, there may be numbers like 3.45, 2.8954 in the expression. But there will not be any negative number in the expression.
- If your code can handle only addition and subtraction but not multiplication and division, you will still receive some partial marks.
- You're not allowed to use Python's built-in `eval()` function or the `exec()` function.
- You're not allowed to use files.

Some examples are below:

- Example #1:
  `expresson:"1+2-3+4"`
  returned value : `4.0`
- Example #2:
  `expresson:"3.5+4*3.0/4-12/5+1.2"`
  returned value : `5.3`
- Example #3:
  `expresson:"1.0/2"`
  returned value : `0.5`
- Example #4:
  `expresson:"2"`
  returned value : `2.0`

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

Let us use tuples and lists to represent a family tree.

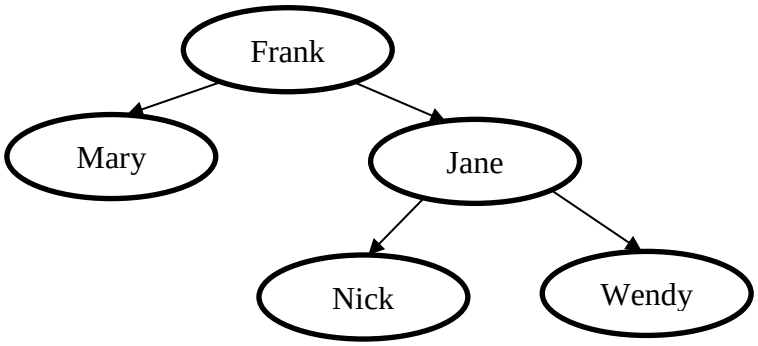
Each person is represented as a tuple. The first element of the tuple is this person's name. The second element of the tuple is a list that contains this person's children. Each child is represented as a tuple itself. If a person has no child, the list of his/her children is empty.

For example, if Mary has no child, she is represented as `('Mary', [])`.

If Jane has two children named Nick and Wendy, and neither Nick nor Wendy has any child, then Jane is represented as `('Jane', [('Nick', []), ('Wendy', [])])`.

If Mary and Jane are Frank's children, and Frank doesn't have any other child, then Frank is represented as `('Frank', [('Mary', []), ('Jane', [('Nick', []), ('Wendy', [])])])`.

The following picture shows their relationship:



Define a function called `get_family_members()` that takes in the representation of a person called `head`, who represents the head of a family tree. The `head` may have several children, grandchildren, great-grandchildren, etc. The function returns all the members of this person's family (including himself/herself) in a list. The members should be ordered by generation. In other words, **a person of the *n*-th generation should always appear after a person of the (*n*-1)-th generation**. Also, people of the same generation should be ordered based on their sequence in the original list. For example, if Nick and Wendy are of the same generation, and Nick appears before Wendy in the original list, then Nick should appear before Wendy in the returned list.

- Example #1:
  `head : ('Mary', [])`
  returned value: `['Mary']`
- Example #2:
  `head : ('Jane', [('Nick', []), ('Wendy', [])])`
  returned value: `['Jane', 'Nick', 'Wendy']`
- Example #3:
  `head : ('Frank', [('Mary', []), ('Jane', [('Nick', [])])])`
  returned value: `['Frank', 'Mary', 'Jane', 'Nick']`
- Example #4:
  `head : ('Alan', [('Bob', [('Chris', [])]), ('Eric', [])])`
  returned value: `['Alan', 'Bob', 'Eric', 'Chris']`

- Example #5:

```
head : ('Alan', [('Bob', [('Chris', []), ('Debbie', [('Cindy',
[])])]), ('Eric', [('Dan', []), ('Fanny', [('George', [])])]),
('Hannah', [])])
```

returned value: `['Alan', 'Bob', 'Eric', 'Hannah', 'Chris', 'Debbie', 'Dan', 'Fanny', 'Cindy', 'George']`

Question 6 (Difficulty Level: **)**

Note: For this question (both Part (a) and Part (b)), you are NOT allowed to use the following methods in Python:

- `itertools.permutations()`

Part (a) **[3 marks]**

Define a function called `scramble()` inside **`q6a.py`**.

The function takes in two parameters:

- `letters` : a string that contains only lowercase letters and is not an empty string
- `file_name` : the name of a text file that contains a list of commonly used English words, one word per line. (You can open the file 'english_words.txt' to check how the file looks like.) Do NOT assume the words are alphabetically ordered.

The function checks if by scrambling the letters in `letters` we can form an English word. If so, all the English words that can be formed by scrambling the letters are to be returned inside a list. If such a word cannot be formed, an empty list is returned.

Note: If the returned list contains multiple words, the order of these words doesn't matter.

- Example #1:
  `letters:'rteid'`
  `file_name:'english_words.txt'`
  returned value : `['tired', 'tried']`
  (Note: It is also OK if the returned value is `['tried', 'tired']`.)
- Example #2:
  `letters:'odg'`
  `file_name:'english_words.txt'`
  returned value : `['dog', 'god']`
  (Note: It is also OK if the returned value is `['god', 'dog']`.)
- Example #3:
  `letters:'ibg'`
  `file_name:'english_words.txt'`
  returned value : `['big']`
- Example #4:
  `letters:'aaaa'`
  `file_name:'english_words.txt'`
  returned value : `[]`

Part (b) **[3 marks]**

Define a function called `get_all_permutations()` inside **`q6b.py`**.

The function takes in a string called `letters` that consists of only lower case letters. Note that `letters` can be of any length and `letters` is not an empty string. The function returns a list that contains all possible permutations of the letters in the string. Note that the returned list doesn't have any duplicate elements.

- Example #1:
  `letters:'abc'`
  returned value : `['abc', 'acb', 'bac', 'bca', 'cab', 'cba']`
  (Note: It is OK if the elements in the returned list are in a different order.)
- Example #2:
  `letters:'ab'`
  returned value : `['ab', 'ba']`
  (Note: It is OK if the elements in the returned list are in a different order.)
- Example #3:
  `letters:'aab'`
  returned value : `['aab', 'aba', 'baa']`
- Example #4:
  `letters:'a'`
  returned value : `['a']`

Note: If you can solve part (b), then you can use it to help you solve part (a). However, you can also solve part (a) without solving part (b).