Lab Test 1 [25 marks]

Coding time: 1.5 hours

General Instructions:

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

2. You are not allowed to communicate with anyone or accessing any website (except for downloading/uploading files from/to eLearn) during the test. **Anyone caught communicating with another person or accessing any website during the test WILL be considered as cheating and subjected to disciplinary actions.**

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

4. 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.

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

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

7. Python script files 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.

8. 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.2021
```

Instructions on how to submit your solutions:

1. Before the test begins, you will be instructed to download the resource files of the test from eLearn.

2. You will be given a password to unzip the downloaded .zip file. After unzipping the file, **you should rename the folder "your_email_id" to your actual email id.** For example, if your SMU email is lum.ting.wong.2021@scis.smu.edu.sg, you should rename the folder to lum.ting.wong.2021. You need to save your solutions to this folder for submission later. You may be penalized for not following our instructions.

3. When the test ends, you will be instructed to submit your solutions as a single zip file to eLearn.

Question 1: Fundamentals and Conditions [7 marks]

Part (a) (Difficulty Level: *)

Inside `q1a.py`, implement a function called `get_area_difference`. The function takes in four *positive integers* as its parameters: `width1`, `length1`, `width2`, `length2`. `width1` and `length1` store the width and length of a rectangle, respectively, and `width2` and `length2` store the width and length of another rectangle. The function *returns* the difference between the areas of the two rectangles. Note that the difference should not be negative. If the two rectangles are of the same size, the function returns `0`.

Examples:

- `get_area_difference(2, 4, 3, 3)` returns `1`, because the second rectangle (with a width of 3 and a length of 3) is larger than the first rectange (with a width of 2 and a length of 4), and $(3 \times 3) - (2 \times 4) = 1$.
- `get_area_difference(4, 3, 2, 5)` returns `2`, because the first rectangle (with a width of 4 and a length of 3, and thus an area of 12) is larger than the second rectange (with a width of 2 and a length of 5, and thus an area of 10), and $(4 \times 3) - (2 \times 5) = 2$.
- `get_area_difference(4, 6, 3, 8)` returns `0`, because $(4 \times 6) - (3 \times 8) = 0$.

Use **`q1a_test.py`** to test your code. You do not need to modify `q1a_test.py`. We will only grade your `q1a.py`.

Part (b) **(Difficulty Level: **)**

Inside **`q1b.py`**, implement a function called `is_official_language()`. The function takes in two parameters:

- `country` (of type `str`): This string is *always* one of the following four countries: `'Belgium'`, `'Canada'`, `'Philippines'`, `'Singapore'`.
- `language` (of type `str`): This string is *always* one of the following: `'Chinese'`, `'Dutch'`, `'English'`, `'Filipino'`, `'French'`, `'German'`, `'Malay'`, `'Tamil'`.

The function returns `True` if the specified language is one of the official languages of the specified country and `False` otherwise. You do not need to handle parameter values that are not one of the countries/languages above.

The table below shows which languages are the official languages of each country:

	Chinese	Dutch	English	Filipino	French	German	Malay	Tamil
Belgium		X			X	X		
Canada			X		X			
Philippines			X	X				
Singapore	X		X				X	X

Examples:

- `is_official_language("Belgium", "French")` returns `True`
- `is_official_language("Belgium", "Malay")` returns `False`
- `is_official_language("Singapore", "German")` returns `False`
- `is_official_language("Canada", "English")` returns `True`

Use **`q1b_test.py`** to test your code. You do not need to modify `q1b_test.py`. We will only grade your `q1b.py`.

Question 2: Loops [7 marks]

Part (a) (Difficulty Level: *)

In `q2a.py`, implement a function called `get_right_most_even_digit()`. The function takes in a *positive integer* `num` as its parameter. The function ***returns*** the right-most digit of `num` that is even or `None` if `num` does not have any even digit. The returned value should be either an `int` or `None`.

Examples:

- `get_ right_most_even_digit(7684357)` returns `4` because starting from the right-hand side of the number, the digits 7, 5 and 3 are all odd, and 4 is the first even digit.
- `get_right_most_even_digit(50570)` returns `0`.
- `get_right_most_even_digit(853751)` returns `8`.
- `get_right_most_even_digit(37953)` returns `None` because there is no even digit in the number.

Use **`q2a_test.py`** to test your code. You do not need to modify `q2a_test.py`. We will only grade your `q2a.py`.

Part (b) **(Difficulty Level: **)**

In `q2b.py`, implement a function called `get_all_third_digits()`. The function takes in a single parameter called `str_list`, which is a list of strings. The function *returns* a list that contains the *third digit* (from the left-hand side) found in each string inside `str_list`. If a string contains fewer than 3 digits, that string is skipped.

Examples:

- `get_all_third_digits(['IS113 G1', 'xyz1', '5-6-7-8', 'S$10,500'])` returns `[3, 7, 5]`, because
  - o `3` is the third digit of `'IS113 G1'`,
  - o there is no third digit in `'xyz1'`,
  - o `7` is the third digit of `'5-6-7-8'`,
  - o `5` is the third digit of `'S$10,500'`.
- `get_all_third_digits(['987654', ''])` returns `[7]`.
- `get_all_third_digits(['abc', '10'])` returns `[]`.
- `get_all_third_digits([])` returns `[]`.

Use **`q2b_test.py`** to test your code. You do not need to modify `q2b_test.py`. We will only grade your `q2b.py`.

Question 3: Diversity Orientation [7 marks]

A group of students are participating in a diversity orientation program. The students are to be divided into teams of 5, where each team should have some level of racial, gender and religious diversity. Specifically, each team must satisfy the following requirements:

- Each team should have either (a) 3 boys and 2 girls, or (b) 3 girls and 2 boys.
- Each team should have representatives from *at least 3 different racial groups*. Note that racial groups are not limited to the four major racial groups in Singapore, i.e., there may be other racial groups such as "Thai", "Vietnamese", etc.
- Each team should have representatives of *at least 2 different religions* such as "Christianity" and "Islam". Note that in this question we do not consider "Freethinker" to be a religion. E.g., if a team has 3 members who are "Freethinker" and 2 members whose religion is "Islam", that team only has representatives of one religion. Also note that religions are not limited to the ones seen below, i.e., there may be other religions such as "Sikhism", "Shinto", etc.

Part (a) (Difficulty Level: **)

Inside `q3a.py`, implement a function called `check_diversity()`.

- The function takes in a single parameter called `team`, which is a list of 5 tuples that represents a team of 5.
- Each tuple represents a student and has the following elements:
  - name of the student (of type `str`),
  - gender of the student (of type `str`), which is either `'M'` or `'F'`,
  - racial group of the student (of type `str`)
  - religion of the student (of type `str`) or the string `"Freethinker"` when the student does not have a religion.

  For example, the tuple `('Xavier', 'M', 'Chinese', 'Freethinker')` represents a male student named Xavier who is Chinese and who doesn't have a religion. The tuple `('Noah', 'M', 'Jew', 'Judaism')` represents a male student named Noah who is a Jew and whose religion is Judaism.
- The function *returns* `True` if the team satisfies the requirements stated above and `False` otherwise.

Examples:

-
  ```
  team = [('Xavier', 'M', 'Chinese', 'Freethinker'),
          ('Noah', 'M', 'Jew', 'Judaism'),
          ('Hui Ling', 'F', 'Chinese', 'Buddhism'),
          ('Siti', 'F', 'Malay', 'Islam'),
          ('Aryan', 'M', 'Indian', 'Hinduism')]
  ```
  `check_diversity(team)` returns `True` because in this team
  - there are 3 boys and 2 girls,
  - there are 4 different racial groups (Chinese, Jew, Malay and Indian), above the minimum requirement of 3,
  - there are 4 different religions represented (Judaism, Buddhism, Islam and Hinduism), above the minimum requirement of 2.

-
  ```
  team = [('Xavier', 'M', 'Chinese', 'Freethinker'),
          ('Faith', 'F', 'Chinese', 'Christianity'),
          ('Li Wen', 'F', 'Chinese', 'Freethinker'),
          ('Siti', 'F', 'Malay', 'Islam'),
          ('Nicholas', 'M', 'Chinese', 'Freethinker')]
  ```
  `check_diversity(team)` returns `False` because although the team satisfies the gender and racial requirements, the team only has members from 2 racial groups (Chinese and Malay).

Use **`q3a_test.py`** to test your code. You do not need to modify `q3a_test.py`. We will only grade your `q3a.py`.

Part (b) **(Difficulty level: ***)**

Inside `q3b.py`, implement a function called `swap_members()`. The function takes in two parameters called `team1` and `team2`. Each parameter is a list of 5 tuples, representing a team of 5. Each tuple represents a student in the same way as Part (a) of Q3.

It is known that *neither* `team1` *nor* `team2` satisfies the requirements above. I.e., you can assume that `check_diversity(team1)` returns `False` and `check_diversity(team2)` returns `False`. The function `swap_members()` checks if it's possible to swap one member in `team1` with one member in `team2` such that after the swapping, *both* teams will satisfy the requirements above.

- The function returns `None` if such a swapping *cannot* be found.
- Otherwise, the function returns a tuple containing the names of the two members to be swapped, where the *first* name is a member of `team1` and the *second* name is a member of `team2`. If there are multiple ways to do the swapping, the function just needs to return one of such swaps.

Note: The function should NOT alter the team membership of the two teams, i.e., the function simply returns the two members that could be swapped but the two lists `team1` and `team2` should not be changed.

Examples:

-
  ```
  team1 = [('Xavier', 'M', 'Chinese', 'Freethinker'),
           ('Faith', 'F', 'Chinese', 'Christianity'),
           ('Li Wen', 'F', 'Chinese', 'Freethinker'),
           ('Siti', 'F', 'Malay', 'Islam'),
           ('Nicholas', 'M', 'Chinese', 'Freethinker')]
  team2 = [('Ismail', 'M', 'Malay', 'Islam'),
           ('Boon Kiat', 'M', 'Chinese', 'Freethinker'),
           ('Julian', 'M', 'Chinese', 'Freethinker'),
           ('Sajid', 'M', 'Indian', 'Islam'),
           ('Aisha', 'F', 'Malay', 'Islam')]
  ```

  `swap_members(team1, team2)` returns `None` because it is not possible to swap any two members of the two teams to make both teams satisfy the requirements.

-
  ```
  team1 = [('Xavier', 'M', 'Chinese', 'Freethinker'),
           ('Faith', 'F', 'Chinese', 'Christianity'),
           ('Li Wen', 'F', 'Chinese', 'Freethinker'),
           ('Siti', 'F', 'Malay', 'Islam'),
           ('Nicholas', 'M', 'Chinese', 'Freethinker')]
  team2 = [('Ismail', 'M', 'Malay', 'Islam'),
           ('Boon Kiat', 'M', 'Chinese', 'Christianity'),
           ('Benjamin', 'M', 'Eurasian', 'Christianity'),
           ('Sajid', 'M', 'Indian', 'Islam'),
           ('Aisha', 'F', 'Malay', 'Islam')]
  ```

  `swap_members(team1, team2)` returns `('Faith', 'Benjamin')` or `('Li Wen', 'Benjamin')` or `('Li Wen', 'Sajid')`. I.e., any one of these three tuples is correct.

  (Note that you should not return `('Benjamin', Faith')` or `('Benjamin', 'Li Wen')` or `('Sajid', 'Li Wen')` because the the first element in the tuple must be a member of `team1`.)

Use **`q3b_test.py`** to test your code. You do not need to modify `q3b_test.py`. We will only grade your `q3b.py`.

Question 4: Word Puzzle [4 marks]

(Difficulty Level: *)**

In this question, you are to write a program to solve a word puzzle.

Each puzzle consists of an 8 x 8 board with some letters forming various words either horizontally or vertically. The figure below shows such a puzzle, where the numbers shown are the row indices and column indices. An asterisk * represents a missing letter. The player is given all the missing letters (shown as "key" below the figure in the example below). The player is to use each letter in the key *once and only once* to replace the asterisks such that each horizontal or vertical segment of the puzzle forms a word.

	0	1	2	3	4	5	6	7
0	**B**	**R**	**E**	*		**B**	**E**	**D**
1						*		**I**
2			**B**	**I**	**R**	**D**		**R**
3			**I**		**I**			*
4	**R**		*	**I**	**D**	**E**		
5	**I**		**D**					
6	*	**R**	**I**	**D**	**E**			
7			**E**					

Key: **BDEIR**

For example, the solution to the puzzle above is shown as follows:

	0	1	2	3	4	5	6	7
0	**B**	**R**	**E**	**[D]**		**B**	**E**	**D**
1						**[I]**		**I**
2			**B**	**I**	**R**	**D**		**R**
3			**I**		**I**			**[E]**
4	**R**		**[R]**	**I**	**D**	**E**		
5	**I**		**D**					
6	**[B]**	**R**	**I**	**D**	**E**			
7			**E**					

In `q4.py`, you will write a function called `solve_puzzle()` to solve the puzzle in a brute-force manner, i.e., you will try all possible ways of replacing the asterisks with the letters in the key until you find a solution. `solve_puzzle` takes in the following parameters:

- `puzzle`: This parameter is a list of exactly 8 strings, where each string has exactly 8 characters. Each string represents one row of the puzzle. For the puzzle given above, for example, this list looks like the following:

```
puzzle = ['BRE* BED',
          '     * I',
          '  BIRD R',
          '  I I  *',
          'R *IDE  ',
          'I D     ',
          '*RIDE   ',
          '  E     ']
```

- `word_positions`: This parameter is a list of tuples. Each tuple represents the position of a segment of the puzzle that is expected to be a word. Each tuple contains 4 elements: row index, column index, direction (`'H'` for horizontal and `'V'` for vertical), and length. E.g., `(0, 0, 'H', 4)` represents a horizontal segment starting from position (0, 0) of length 4, and in the puzzle above, this corresponds to the incomplete word "BRE*". The entire list of word positions of the puzzle above is the following:

```
word_positions = [(0, 0, 'H', 4), (0, 5, 'H', 3),
                  (2, 2, 'H', 4), (4, 2, 'H', 4),
                  (6, 0, 'H', 5), (0, 5, 'V', 3),
                  (0, 7, 'V', 4), (2, 2, 'V', 6),
                  (2, 4, 'V', 3), (4, 0, 'V', 3)]
```

- `key`: This is a string containing several letters in uppercase. The number of letters in the key is exactly the same as the number of asterisks in the puzzle.
- `vocabulary`: This is a list of known English words given to the function. You can assume that when the puzzle is solved, all words in the puzzle can be found in this vocabulary. Therefore, you can use this vocabulary to check if a solution is valid.

The function is expected to return the solution to the puzzle as a list of tuples where each tuple contains 3 elements: row index, column index, a letter from the key. The tuples are to indicate where the letters in the key are to be placed. E.g., given the puzzle above, the solution should be the following list (or another list with the same tuples as elements in a different order):

```
[(0, 3, 'D'), (1, 5, 'I'), (3, 7, 'E'), (4, 2, 'R'), (6, 0, 'B')]
```

We provide a utility function called `print_puzzle()` that takes in `puzzle` (or any list of 8 strings of length 8) and prints out the puzzle to the screen.

Besides, you are given the following sample code that shows how you can use the `permutations` function to generate all permutations of the characters inside a string. You will find this useful when implementing the function above.

```
import itertools
all_sequences = itertools.permutations('abc')
for sequence in all_sequences:
    print(sequence)
```

Use **`q4_test.py`** to test your code. You do not need to modify `q4_test.py`. We will only grade your `q4.py`.