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.**

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 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.

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.2020
```

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.2020@sis.smu.edu.sg, you should rename the folder to lum.ting.wong.2020. 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_tank_volume()`. This function takes in three parameters called `width`, `depth` and `height`, which are all positive integers. They represent the width, depth and height of a tank **in centimetres**. The function returns the volume of this tank **in litres**, ***dropping all decimical places***.

Note that 1 litre is equal to 1000 cubic centimetres. So the formula to calculate the volume in litres is as follows:

$$volume = \frac{width \times depth \times height}{1000}$$

where width, depth and height are in centimetres and volume is in litres.

Examples:

- `get_tank_volume(30, 45, 40)` returns `54`, because $\frac{30 \times 45 \times 40}{1000} = 54$.
- `get_tank_volume(36, 45, 124)` returns `200`, because $\frac{36 \times 45 \times 124}{1000} = 200.88$, and after dropping all the decimal places, we get 200.

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_compatible()`. The function checks if a patient of a certain blood group type can receive plasma from a donor of a certain blood group type. The table below shows the compatible donor's blood groups given a patient's blood group. For example, Group A recipients can only receive Group A or Group AB plasma.

Patient Group	Compatible Plasma Donor
A	A, AB
B	B, AB
AB	AB
O	O, AB, A, B

The function `is_compatible()` takes two parameters called `patient_group` and `donor_group`, representing the blood groups of a patient and a donor, respectively. The function returns `True` if the donor's blood group is compatible and `False` otherwise.

You can assume that the two parameters' values are always one of the four strings: `"A"`, `"B"`, `"AB"` or `"O"`.

Examples:

- `is_compatible("B", "AB")` returns `True`
- `is_compatible("O", "A")` returns `True`
- `is_compatible("AB", "O")` returns `False`

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 `get_sum_of_digits()`. The function takes in a string called `my_str` as its parameter. The function sums up all the digits found in the string and returns the sum as an int. If the string is empty or the string doesn't contain any digit, the function returns `0`.

Examples:

- `get_sum_of_digits('1234')` returns `10` because 1 + 2 + 3 + 4 = 10
- `get_sum_of_digits('SMU1-2-3-4')` returns `10` because 1 + 2 + 3 + 4 = 10
- `get_sum_of_digits('32-4*5(0)36')` returns `23` because 3 + 2 + 4 + 5 + 0 + 3 + 6 = 23
- `get_sum_of_digits('SMU-SIS')` returns `0` because there is no digit in the string.
- `get_sum_of_digits('')` returns `0` because there is no digit in the string.

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_max_of_min()`. The function takes in a list of tuples called `list_of_num_tuples` as its parameter, where each tuple contains three integer numbers. The function returns the maximum of the minimum values in these tuples.

If the list is empty, the function returns None.

Note: You're **NOT** allowed to use either `min()` or `max()` to solve this problem.

Examples:

- `get_max_of_min([(1, 5, 3), (5, 1, 9), (4, 10, 19)])` returns `4`, because the minimum value of the first tuple is 1, the minimum value of the second tuple is 1, and the minimum value of the third tuple is 4. Then the maximum value among 1, 1 and 4 is 4.
- `get_max_of_min([(-15, 20, -40), (-4, -1, -4)])` returns `-4`, because the minimum value of the first tuple is -40, the minimum value of the second tuple is -4. Then the maximum value between -40 and -4 is -4.
- `get_max_of_min([(3, -9, 0)])` returns `-9`, because the minimum value of the only tuple in the list is -9.
- `get_max_of_min([])` returns `None`.

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: Math Equations [4 marks]

(Difficulty Level: **)

In **q3.py**, implement a function called `check_math()`. The function takes in a parameter called `list_of_equations`, which is a list of strings where each string is a math equation that always consists of an integer, a Python operator, another integer, the equal sign, and a third integer. The string may also contain some spaces.

For example,

- `"3 + 5 = 9 "` is a math equation of this format.
- `" 4 *2= 8 "` is also a math equation of this format.

The function returns `True` if ***all*** math equations are mathematically ***correct***, and `False` if ***at least one of the equations*** is mathematicall ***wrong***. If the list is empty, the function returns `True`.

Assumptions:

- All strings are of the format described above.
- All numbers in these equations are *positive integers or 0.*
- The operators are always one of the following:
  - `+` (addition)
  - `-` (subtraction)
  - `*` (multiplication)
  - `//` (floor division)
  - `%` (modulo).
- There can be some spaces between the numbers and the operators, or at the beginning/end of the equations.

Note:

- You are **NOT** allowed to use the Python built-in function `eval()` to solve this problem.

Tips:

- You can use the `split()` function.
- Note that `int()` can handle a string that has spaces in front or at the back. E.g., `int("  45 ")` returns `45`.

Examples:

- `check_math([" 3 +5 =9", "4 * 3 = 12 "])` returns `False` because the first equation is wrong.
- `check_math(["13 % 5 = 3", "4 // 3 = 1", "90 - 50 = 40"])` returns `True` because all equations are correct.
- `check_math([])` returns `True`.

Use **q3_test.py** to test your code. You do not need to modify `q3_test.py`. We will only grade your `q3.py`.

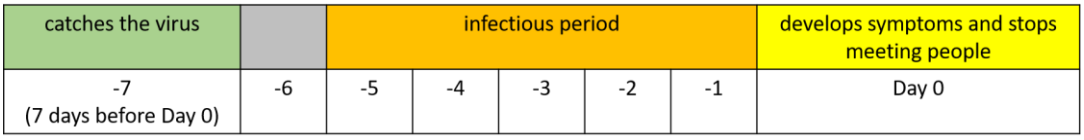
Question 4: Contact Tracing [7 marks]

(Difficulty Level: *)**

Part (a)

In Year 2030, a new virus called Ω appears. This virus strictly follows the patterns below to spread among humans:

- Suppose person P starts to develop symptoms on Day 0.
  - P must have caught the virus exactly 7 days before Day 0. For example, if P starts to develop symptoms on September 20, 2030, then he must have caught the virus exactly on September 13, 2030.
  - P starts to infect other people he meets exactly 2 days after he catches the virus. In the example above, P starts to infect people on September 15, 2030.
  - For 5 days, P is infectious but asymptomatic.
  - Once P starts to development symptoms, he stops meeting people and therefore will not infect more people.



In `q4a.py`, implement a function to help contact tracing. The function is called `trace_contacts()`. It does the following:

- The function takes in 2 parameters:
  - `patient` (type: `str`): the name of a person who just starts to develop symptoms *on the current date*.
  - `history` (type: list): a list of tuples that stores the meeting history of people in the community. Specifically, each tuple contains three elements:
    - `p1` (type: `str`): a person's name
    - `p2` (type: `str`): another person's name (which is always different from `p1`)
    - `day` (type: `int`): a *negative* integer indicating the day of the meeting relative to the current date (i.e., the date when `patient` develops symptoms)

    For example, the tuple `("Jason", "Gideon", -3)` means Jason met Gideon three days ago. (I.e., if the current date is September 23, 2030, which is the date when `patient` develops symptoms, then Jason and Gideon met on September 20, 2030.)
- The function returns a list of strings, which are the names of those people who may have caught the virus either directly or indirectly from `patient`. This returned list should not contain any duplicate elements.

Note:

- The same pair of people could meet each other on different dates, e.g., `('A', 'B', -3)` and `('A', 'B', -2)` may both appear in `history`.
- All dates in `history` (i.e., the negative integers) are with respect to the original `patient`'s Day 0.
- The original `patient` could appear anywhere in the meeting history (not necessarily in the first tuple).
- The tuples in `history` are not necessarily chronologically ordered.

For example,

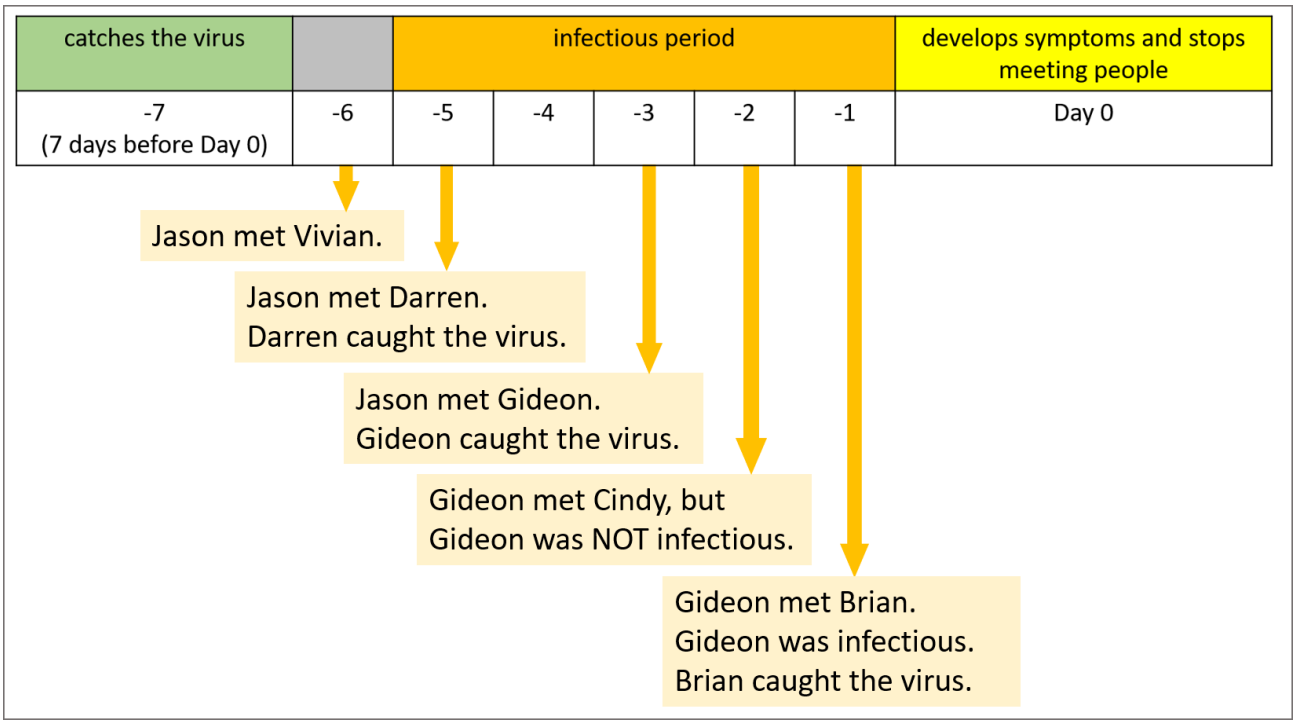
- Suppose the variable `history` stores the following list:

```
history = [("Jason", "Gideon", -3),
           ("Zac", "Yacob", -3),
           ("Gideon", "Brian", -1),
           ("Cindy", "Gideon", -2),
           ("Darren", "Jason", -5),
           ("Jason", "Vivian", -6)]
```

  Then `trace_contacts("Jason", history)` should return

  `["Gideon", "Brian", "Darren"]` (or a list with these three elements in any order)

  This is because Gideon and Darren both met Jason during Jason's infectious period (which is between five days ago and the day before), and Brian met Gideon during Gideon's infectious period (which started the day before), given that Gideon has already caught the virus. This is illustrated in the following diagram:



- Suppose the variable history stores the following list:

```
history = [("B", "A", -5),
           ("C", "A", -1),
           ("B", "C", -2)]
```

  Then `trace_contacts("A", history)` should return `["B", "C"]` or `["C", "B"]`. Note that here C is listed only once in the returned list, although C met both B during B's infectious period and A during A's infectious period.

Use **`q4a_test.py`** to test your code. You do not need to modify `q4a_test.py`. We will only grade your `q4a.py`.

Part (b)

Your implemented `trace_contacts()` function works so well that the infectious disease experts want you to generalize it so it can work with any future virus with a different incubation period.

In `q4b.py`, implement a new function `trace_contacts_2()` that works as follows:

- The function takes in 4 parameters:
  - `patient` (type: `str`): the name of a person who just starts to develop symptoms on the current day. (Same as in Part (a).)
  - `history` (type: `list`): a list of tuples that stores the meeting history of people in the community. (Same as in Part (a).)
  - `m` (type: `int`): a positive integer indicating how many days a person is infectious but asymptomatic. For virus Ω above, `m` is equal to 5.
  - `n` (type: `int`): a positive integer indicating how many days it takes for a person to develop symptoms from the day when he catches the virus. You can assume that `n>m`. For virus Ω above, `n` is equal to 7.

catches the virus	non-infectious period			infectious period				develops symptoms and stops meeting people
$-n$ (n days before Day 0)	$-(n-1)$	...	$-(m+1)$	$-m$ (m days before Day 0)	$-(m-1)$	...	-1	Day 0

- The function returns a list of strings, which are the names of those people who may have caught the virus either directly or indirectly from `patient`. This returned list should not contain any duplicate elements. (Same as in Part (a).)

Use **`q4b_test.py`** to test your code. You do not need to modify `q4b_test.py`. We will only grade your `q4b.py`.