Lab Test 2 [25 marks]

Coding time: 1 hour and 30 mins

Instructions on how to download the resource files:

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

General Instructions:

1. You will take 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, misspellings, etc. in the output.
5. Do not hardcode. We will use different test cases to test and grade your solutions.
6. 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.
7. 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    : BAI She Jing
# Email ID: shejing.bai.2021
```

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 `shejing.bai.2021@sis.smu.edu.sg`, you should name the archive as **`shejing.bai.2021.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 submit your solutions as a single zip file to eLearn under Assignments.

Question 1 (Difficulty Level *) [7 marks]

Implement a function called `get_tuples_of_size`. This function takes in 2 parameters:

- `tup_list` (type: `list`): a list of tuples. Each tuple contains one or more integers.
- `num` (type: `int`): a positive integer value.

The function returns a new list containing selected tuples in `tup_list`. To be selected, each tuple must have at least `num` or more elements. If there is no such tuple, the function returns an empty list. Your code **must not modify** `tup_list`.

Examples:

- `get_tuples_of_size([(1, -1, 18), (4, 2, 6), (19, 0)], 3)` returns `[(1, -1, 18), (4, 2, 6)]`.
- `get_tuples_of_size([(1, 2, 0), (4, 20)], 1)` returns `[(1, 2, 0), (4, 20)]`.
- `get_tuples_of_size([(1, 2), (9, 6), (-9, 2)], 4)` returns `[]`
- `get_tuples_of_size([], 1)` returns `[]`

Question 2 [7 marks]

Part A (Difficulty Level *)

Implement a function called `extract_names_1`. This function takes in the following parameter:

- `str_name` (type: `str`): a non-empty string of names, in which each name is separated by at least one or more consecutive hashtag (`#`) characters. You can assume that: 1) the names in `str_name` will not contain any hashtag characters; 2) each name will contain at least one character which is not a space character (`' '`); and 3) there are no leading or trailing spaces for each name.

This function returns a list of names extracted from `str_name`. Note that each name in the returned list should not be empty.

Examples:

- `extract_names_1('Alex Tan##xy1 z####abc$#S')` returns `['Alex Tan', 'xy1 z', 'abc$', 'S']`.
- `extract_names_1('ab c#def')` returns `['ab c', 'def']`.

Hint:
You might find the `split` method for strings useful.

```
fruits = "apple,orange,pear"
# the below code prints out: ['apple', 'orange', 'pear']
print(fruits.split(","))
```

Part B (Difficulty Level **)

Implement a function called `extract_names_2`. This function takes in the following parameter:

- `name_list` (type: `list`): a list of non-empty strings. Each string in the list is a name as extracted by Part A.

The function needs to remove all invalid characters, which are neither alphabet characters nor spaces (`' '`), from each given name in the list. It returns a new list of strings containing the modified names having only valid characters (alphabets and/or spaces). If there are no alphabet characters in a name, the name should not be included in the returned list. Your code must not modify `name_list`.

Note: you do **not** need to use the function implemented in Part A for this part.

Examples:

- `extract_names_2(['Alex T5an', '^Harry Potter$', 'Squid$$ Game', 'abc','5 -6- 7-8'])` returns `['Alex Tan', 'Harry Potter', 'Squid Game', 'abc']`.
- `extract_names_2(['Alina Star**kov', 'Jessie Mei  Li'])` returns `['Alina Starkov', 'Jessie Mei  Li']`.
- `extract_names_2(['@@ 12'])` returns `[]`.
- `extract_names_2([])` returns `[]`.

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

Implement a function `get_vaccination_status` to obtain the current vaccination status of a group of persons against an infectious disease named `X`. The function takes the following parameters:

- `filename` (type: `str`): this is the name of the file storing person information.
- `today` (type: `str`): a string in `'dd/mm/yyyy'` format indicating the date of today, where month and day can be either 1 digit or 2 digits, e.g., '02' or '2'.

The `filename` refers to a text file that contains all the vaccination entries. Each entry for a single person is in one row with four columns separated by the pipe character (`'|'`). The file contains at least one entry.

1. The first column stores a person identification string which is unique in the file.
2. The second column stores the age information as an integer.
3. The 3rd and 4th column are the dates of the first and second vaccine doses, respectively, in `'dd/mm/yyyy'` format (the same format as that of `today`). If a column in a row is shown as `'NA'`, it means that the data is not available for that column. We assume that all data in the file are of the proper format.

An example input file named `record1.txt` is shown below:

```
S1|22|11/2/2021|13/3/2021
F1|2|NA|NA
G2|35|1/9/2021|20/10/2021
S2|12|02/6/2021|11/7/2021
S3|60|1/12/2020|NA
S5|62|05/08/2021|10/10/2021
```

The function returns a Python dictionary storing the vaccination statuses of all persons listed in the file. In the dictionary, each key is a person identification string, while the corresponding value is a tuple in the form of `(age, vaccinated)`, where `age` is an `int`, and `vaccinated` is a `bool` value indicating whether the person is vaccinated or not (`True` or `False` respectively).

To determine the vaccination status:

1. A person below the age of 12 is considered **not** vaccinated as no vaccine is available for children yet.

2. For people of 12 years old and above, a person is considered **not** vaccinated if:
    a. No doses of vaccine have been given, or only the first dose has been given.
    b. If the 2nd dose has been given less than 4 weeks (28 days) or more than 8 weeks (56 days) after the first dose.

    Example: for person `A`, if the file shows that the 1st dose was given on 15/8/2021, and the 2nd dose was given on 11/10/2021 or later, the schedule is **not** valid as the interval between doses in this case is 57 days or more. Hence, person `A` is considered **not** vaccinated.

    Another example: for person `B`, if his/her 1st dose was on 15/8/2021, and the 2nd dose was given on 12/9/2021, the schedule is considered **valid** as the interval is exactly 28 days.

c. If the 1st and 2nd dose dates are valid but the 2nd dose has been given less than 14 days before `today`.

   Example: for person `C`, assuming that the interval between his/her 1st dose and 2nd dose is valid, if the 2nd dose was given on 15/10/2021 or earlier, and `today` is 29/10/2021, person `C` is considered vaccinated, as the interval between the 2nd dose and `today` is 14 days or more.

Note: the below function to compute the number of days between two given dates is provided in the starting code of `q3.py`. You are **required** to use this function in your solution to Q3.

```
import datetime
def get_days_between(start_tup, end_tup):
    '''
    Parameters:
      start_tup: the start date which is a tuple in the form (year, month, day).
                 The year, month and day are integers.
      end_tup:   the end date. The format is the same as start_tup.

    Returns the number of days in between the 2 days (end_tuple - start_tuple).
    For example, if start_tup is (2020, 12, 1) and end_tup is (2020, 12, 3), this
    function returns the value 2.
    '''
    start_date = datetime.date(*start_tup)
    end_date = datetime.date(*end_tup)
    return (end_date - start_date).days
```

Example:

- `get_vaccination_status('record1.txt', '25/10/2021')` returns `{'S1': (22, True), 'F1': (2, False), 'G2': (35, False), 'S2': (12, True), 'S3': (60, False), 'S5': (62, False)}`.

- `get_vaccination_status('record1.txt', '15/11/2021')` returns `{'S1': (22, True), 'F1': (2, False), 'G2': (35, True), 'S2': (12, True), 'S3': (60, False), 'S5': (62, False)}`.

- `get_vaccination_status('record1.txt', '15/1/2021')` returns `{'S1': (22, False), 'F1': (2, False), 'G2': (35, False), 'S2': (12, False), 'S3': (60, False), 'S5': (62, False)}`.

Question 4 (Difficulty Level: ***) [3 marks]

Implement a function called `read_schedule`. This function takes in the following parameter:

- `filename` (type: `str`) : the name of a file containing usernames of employees and their correspondingly proposed timeslots for work in a particular week in an organization. The format of the file will be explained below.

The function returns a work schedule representing the content of `filename` in the form of a Python dictionary, in which keys are the days in a week, e.g., Mon, Tue, etc., and the corresponding values are lists of tuples; each tuple represents the proposed work timeslot of an employee for that day in the form of `(start time, end time, username)`. The start time and end time of a tuple are integers and must be in 24h format.

Each entry in `filename` is a row with four columns separated by the pipe character (`'|'`). The first column is the day of week, e.g., Mon, Tue, Sat, Sun, etc. The second column and third column are the start and end time of work in the 12h format (with a suffix of AM or PM), respectively. It is assumed that the work day starts at 9AM and end at 5PM. The file contains timeslots within this range only. The timeslots are on the hour, e.g., they can be 9AM-12PM, 1PM-5PM, etc., but not 9.15AM-12PM, 1.30PM-4.45PM, etc. The last column of a row contains one or more usernames of the employees, i.e., these employees propose to work on the same day with the same time. The usernames on the same line are separated by a space character (`' '`). The file contains at least one entry.

As employees enter their own data into the file, there can be some kinds of issues as follows:

1. Usernames on a row could be duplicated. The returned dictionary must not have this kind of duplication.

2. An employee might be listed on two or more different rows which have the same day in the file. In this case:

    a. Only the timeslot with the shortest duration in terms of the number of hours should be recorded in the dictionary for this employee on the given day, or:

    b. If the durations of the shortest timeslots are the same; the one that appears last in the file should be recorded in the dictionary.

Note that the entries are not sorted in any order. Days in the file might not be consecutive, and not all days of week are in the file. Only the days listed in the file should be included in the returned dictionary, which is not ordered.

An example of the input file named `schedule1.txt` is shown below. **Note that the line number for each line shown in the 1st column is not part of the file**. Some (not all) examples of data entry issues in the file are as follows:

1. On line 2, username `donta` is duplicated for `Mon`. This duplication must not be there in the returned dictionary.
2. On lines 1 and 2, username `ahbeng` has different timeslots that are all on `Mon`. In this case, the shortest timeslot (9AM to 10AM) will be recorded in the returned dictionary for `ahbeng` on `Mon`.

3. On lines 5 and 6, username `wendytan` has different timeslots that are all on `Wed`. In this case, the shortest timeslot (1PM to 2PM) will be recorded in the returned dictionary for `wendytan` on `Wed`. If there are 2 or more "shortest" timings, the timeslot that appears last in the file will be recorded.
4. On lines 2 and 10, username `starkov` has different timeslots that are all on `Mon`. In this case, as the durations of both timeslots are the same, the timeslot that appears last in the file (10AM to 5PM) will be recorded in the returned dictionary for `starkov` on `Mon`.

The input file named `schedule1.txt` is as follows:

```
1  Mon|9AM|10AM|ahbeng
2  Mon|9AM|4PM|donta starkov leongben donta ahbeng
3  Tue|12PM|5PM|jingjiang michellekan
4  Wed|9AM|12PM|markloh vandana
5  Wed|9AM|12PM|wendytan ylee wangyong
6  Wed|1PM|2PM|wendytan joelle
7  Tue|9AM|5PM|leongben darkling
8  Fri|2PM|5PM|tonytan jessicali
9  Tue|2PM|4PM|markloh angelwong
10 Mon|10AM|5PM|starkov
```

Example:

- `read_schedule('schedule1.txt')` returns the dictionary

```
{'Mon': [(9, 10, 'ahbeng'), (9, 16, 'donta'), (10, 17, 'starkov'), (9,
16, 'leongben')],

'Tue': [(12, 17, 'jingjiang'), (12, 17, 'michellekan'), (9, 17,
'leongben'), (9, 17, 'darkling'), (14, 16, 'markloh'), (14, 16,
'angelwong')],

'Wed': [(9, 12, 'markloh'), (9, 12, 'vandana'), (13, 14, 'wendytan'),
(9, 12, 'ylee'), (9, 12, 'wangyong'), (13, 14, 'joelle')],

'Fri': [(14, 17, 'tonytan'), (14, 17, 'jessicali')]}.
```

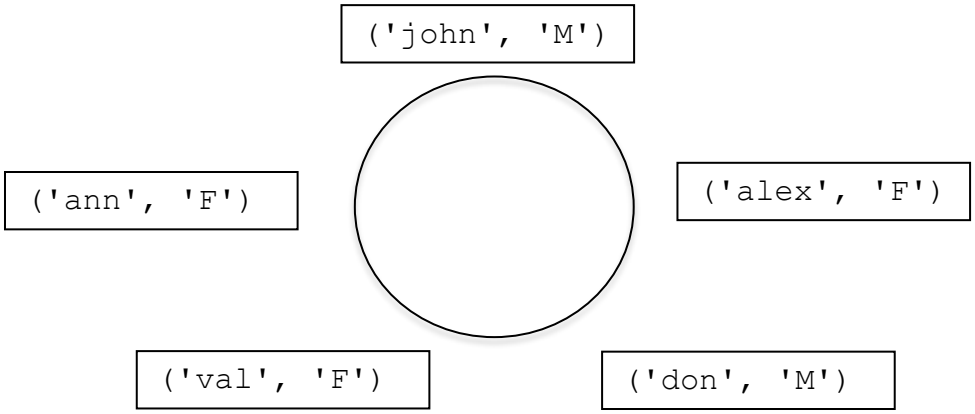
Question 5 [3 marks]

Part A (Difficulty Level *)**

Implement a function called `get_persons`. This function takes in the following parameters:

- `person_list` (type: `list`): a non-empty list of tuples. Each tuple represents a person in the form of `(name, gender)`, where name and gender are both non-empty strings. Gender can only have one of these two values: `'F'` or `'M'`. We assume that names in the list are all unique.

- `n` (type: `int`): a positive number (at least 2 or more). The function uses `n` to determine which persons to get from the list, as explained below.



In `person_list`, two adjacent tuples indicate that the two corresponding persons sit next to each other in a circular table. As an example, the list `[('john', 'M'), ('alex', 'F'), ('don', 'M'), ('val', 'F'), ('ann', 'F')]` represents the seating arrangement in the above figure. Note that the last person in the list, `ann`, sits next to `john`, who is the first person in the list, as the table is circular.

For the above figure, we say that `john` is **enclosed** by three **different** members of the **opposite gender**, namely `ann`, `val` and `alex`, as they surround him on both his left and right side. We also say that `ann` is **not** enclosed by members of the opposite gender, as `val`, who is of the same gender as `ann`, sits directly next to her on one of her sides. In general, for a person `p` to be enclosed in the circular table, there must be some members of the opposite gender of `p` surrounding `p` on **both** his/her sides.

The function `get_persons` returns a new list of persons (tuples) **in any order**, in which each returned person must be **enclosed** by at least `n` **different** members of the opposite gender. If there is no such person, the function returns an empty list. Note that the function must not modify `person_list`.

Examples:

- `get_persons([('john', 'M'), ('alex', 'F'), ('don', 'M'), ('val', 'F'), ('ann', 'F')], 2)` returns `[('john', 'M'), ('alex', 'F'), ('don', 'M')]`. The tuples in the returned list can be in any order.

- `get_persons([('john', 'M'), ('alex', 'F'), ('don', 'M'), ('val', 'F'), ('ann', 'F')], 3)` returns `[('john', 'M'), ('don', 'M')]`.

- `get_persons([('a', 'M'), ('b', 'F'), ('c', 'M'), ('d', 'M'), ('e', 'M'), ('f', 'M'), ('g', 'M'), ('h', 'M')] , 7)` returns only `[('b', 'F')]`.

- `get_persons([('a', 'M'), ('b', 'F'), ('c', 'M'), ('d', 'M'), ('e', 'M'), ('f', 'M'), ('g', 'M'), ('h', 'M')] , 8)` returns `[]`.

Part B (Difficulty Level *)**

Implement a function called `get_seating`. This function takes in the following parameters:

- `person_list` (type: `list`): a non-empty list of tuples as described in Part A of this question.
- `n` (type: `int`): a positive number (at least 2 or more) as described in Part A of this question.
- `m` (type: `int`): a positive number (at least 1 or more).

The function returns a seating arrangement for all the persons in `person_list` as a new list of tuples so that at least `m` persons in the returned list will be **enclosed** by at least `n` other persons of the opposite gender. If there are more than one such seating arrangements, the function only has to return one of them. If the original seating arrangement specified in `person_list` already satisfies the enclosing condition, the same `person_list` must be returned. The function returns `[]` if there is no such seating arrangement. Note that the function must not modify `person_list`.

Examples:

- `get_seating([('john', 'M'), ('val', 'F'), ('ann', 'F'), ('alex', 'F'), ('don', 'M')], 3, 2)` returns `[('john', 'M'), ('val', 'F'), ('ann', 'F'), ('don', 'M'), ('alex', 'F')]` because in the returned arrangement, `john` and `don` are the two persons that are enclosed by at least 3 members of the opposite gender. There can be other valid seating arrangements not listed here.
- `get_seating([('john', 'M'), ('val', 'F'), ('ann', 'F'), ('alex', 'F'), ('don', 'M')], 3, 3)` returns `[]`.
- `get_seating([('john', 'M'), ('val', 'F'), ('ann', 'F'), ('don', 'M'), ('alex', 'F')], 3, 2)` returns `[('john', 'M'), ('val', 'F'), ('ann', 'F'), ('don', 'M'), ('alex', 'F')]` because the original arrangement already satisfies the specified enclosing condition.
- `get_seating([('a', 'M'), ('b', 'M'), ('c', 'M'), ('d', 'M'), ('e', 'F'), ('f', 'F'), ('g', 'F'), ('h', 'F')] , 4, 2)` returns `[('a', 'M'), ('b', 'M'), ('c', 'M'), ('e', 'F'), ('d', 'M'), ('f', 'F'), ('g', 'F'), ('h', 'F')]` because in the returned arrangement, `d` and `e` are the two persons that are enclosed by at least 4 members of the opposite gender. There can be other valid seating arrangements not listed here.

You are given the following sample code that shows how you can use the `permutations` function to generate all permutations of all elements in a list. You might find this useful when implementing the function above.

```
import itertools
all_sequences = itertools.permutations([1, 2, 3])
for sequence in all_sequences:
    print(list(sequence))
```