Lab Test 2 [25 marks]

Coding time: 1 hour and 30 mins

Instructions on how to download the resource files:

1. Download from http://blue.smu.edu.sg/is111/2019/LT2.zip

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, misspellings, etc. in the output.

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

7. Follow IS111 code conventions (e.g. naming functions and variables) or risk a 3 mark penalty on your final score.

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    : BAI She Jing
# Email ID: shejing.bai.2019
```

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.2019@sis.smu.edu.sg`, you should name the archive as **`shejing.bai.2019.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: *) [5 marks]

Implement the `get_pairs_with_total` function. This function takes 2 parameters:

a. `num_pairs` (type: `list`) : Each element is a tuple, and is of the format `(num1, num2)`.
b. `total` (type: `int`)

The function returns a list of tuples whereby `num1 + num2` is equals to the `total`.

Example 1:
If the function is invoked like this:

```
print(get_pairs_with_total([(3,4), (5,4), (3,3), (2,0), (5,2), (1,6)], 7))
```

the statement generates the following output:

```
[(3, 4), (5, 2), (1, 6)]
```

Example 2:
If the function is invoked like this:

```
print(get_pairs_with_total([(1,4), (5,4), (3,3), (2,0), (5,-9)], 7))
```

the statement generates the following output:

```
[]
```

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

Implement the `get_valid_ip_addresses` function. This function takes 1 parameter:

- `addr_list` (type: `list`). It contains a list of IP addresses of type `str`.

A valid IP address must be in the form of xxx.xxx.xxx.xxx, where xxx is a number from 0 (inclusive) - 255 (inclusive). For example,

1. `'12.12.12.12'` is a valid IP address since 12 is a number between 0 and 255.
2. `'127.1.1.256'` is **NOT** a valid IP since 256 is not a number between 0 and 255.
3. `'255.255.255'` is **NOT** a valid IP since there are only 3 groups of xxx

Example 1:

If the function is invoked like this:

```
print(get_valid_ip_addresses(['10.10.10.10', '0.0.0.0', '99.7.7.299']))
```

the statement generates the following output:

```
['10.10.10.10', '0.0.0.0']
```

Note:

1. `'99.7.7.299'` is invalid because 299 is not a number between 0 and 255.

Example 2:

If the function is invoked like this:

```
print(get_valid_ip_addresses(['999.10.10.10', '-1.0.0.0', 'a.10.a.a']))
```

the statement generates the following output:

```
[]
```

Note:

1. `'999.10.10.10'` is invalid because 999 is not a number between 0 and 255.
2. `'a.10.a.a'` is invalid because `'a'` is not a numeric character.

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

Implement the `rotate` function. The function takes in a parameter:

- `matrix` (type: `list`): Represents a n x m matrix, where n and m are numbers greater than 0. In other words, the parameter contains 1 or more list elements. Each list element will contain 1 or more elements. For example, the matrix

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

  is represented as a list `[[1, 2, 3], [4, 5, 6], [7, 8, 9]]`.

This function will rotate a matrix by 90 degrees in anti-clockwise direction. The matrix after rotation will be `[[3, 6, 9], [2, 5, 8], [1, 4, 7]]`.

$$\begin{bmatrix} 3 & 6 & 9 \\ 2 & 5 & 8 \\ 1 & 4 & 7 \end{bmatrix}$$

Example 1:

If the function is invoked like this:

```
print(rotate([[1, 2, 3], [4, 5, 6], [7, 8, 9]]))
```

the statement generates the following output:

```
[[3, 6, 9], [2, 5, 8], [1, 4, 7]]
```

Example 2:

If the function is invoked like this:

```
print(rotate([[1, 2], [4, 5], [7, 8]]))
```

the statement generates the following output:

```
[[2, 5, 8], [1, 4, 7]]
```

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

Implement the function `get_free_timings`. The function takes in the following parameters:

1. `file_name`: the name of the data file.
2. `target_building`: the name of the building. For example, `'SIS'`.
3. `target_facility`: the name of the booking facility. For example, `'SR2.1'`
4. `target_date`: a string representing a particular day, month and year in 'dd/mm/yyyy' format. E.g. `'01/01/2017'` represents 1 Jan 2017.

`file_name` is the data file that contains the list of bookings. Each row has 3 or 4 columns. Each column is separated from another column by one or more white spaces or tabs. If a facility is not found in the file, it is considered to be available for the whole duration, 08:00 (inclusive) to 22:00 (exclusive).

The four columns are (in left-to-right order):

1. **building**: the name of the building. For example, `'SIS'`

2. **facility**: the name of the booking facility. For example, `'SR2.1'`

3. **date**: a string representing a particular day, month and year in `'dd/mm/yyyy'` format. E.g. `'01/01/2017'` represents 1 Jan 2017.

4. **timings**: the bookings' timings. For example, `'11:30-11:45,12:30-13:30'` means that the facility is booked from 11:30 (inclusive) to 11:45 (exclusive) and 12:30(inclusive) to 13:30 (exclusive). The timings are sorted in chronological order. In other words, you will not see timings like`'12:30-13:30,14:20-15:00,11:30-11:45'`(where an earlier timing appears after a later timing).

   Note:

   a. In the 24-hour time notation, the day begins at midnight, 00:00, and the last minute of the day begins at 23:59.
   b. Every facility can only be booked between 08:00 (inclusive) to 22:00 (exclusive).

A sample of the `booking_file` looks as follows:

```
SIS  SR2.1      12/12/2019  11:30-11:45,12:30-13:30,14:20-15:00
SIS SR2.2   12/12/2019  11:30-11:45
SIS SR3.4  12/12/2019
```

1. SIS SR 2.1 is booked on 12/12/2019 for the following timings:
    a. 11:30 - 11:45
    b. 12:30 - 13:30
    c. 14:20 - 15:00

    The room's free slots are:

    a. 08:00 - 11:30 (Note: 08:00 is the start operating hour)
    b. 11:45 - 12:30
    c. 13:30 - 14:20
    d. 15:00 - 22:00 (Note: 22:00 is the end operating hour)

2. SIS SR 2.2 is booked on 12/12/2019 for the following timing:
    a. 15:30 - 18:45

    The room's free slots are:

    a. 08:00 - 15:30 (**Note**: 08:00 is the start operating hour)
    a. 18:45 - 22:00 (**Note**: 22:00 is the end operating hour)

3. SIS SR 3.4 has 0 bookings since **timings** column is empty.

    The room's free slots is: 08:00 - 22:00.

    Note: 08:00 is the start operating hour and 22:00 is the end operating hour)

E.g. 1: If the function is invoked like this:
`print(get_free_timings('bookings.txt', 'SIS', 'SR2.2', '12/12/2019'))`
the statement generates the following output:

```
[('08:00', '11:30'), ('11:45', '22:00')]
```

Note: `'bookings.txt'` is the sample of the booking_file showed earlier.

E.g. 2: If the function is invoked like this:
`print(get_free_timings('bookings.txt', 'SIS', 'SR2.1', '12/12/2019'))`
the statement generates the following output:

```
[('08:00', '11:30'), ('11:45', '12:30'), ('13:30', '14:20'), ('15:00',
'22:00')]
```

E.g. 2: If the function is invoked like this:
`print(get_free_timings('bookings.txt', 'SIS', 'SR3.4', '12/12/2019'))`
the statement generates the following output:

```
[('08:00', '22:00')]
```

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

You are sending a love message to your boyfriend/girlfriend. Just in case your nosy instructors intercept them, you decide to encrypt them. Implement the function `encode`. This function takes in 2 parameters:

1. `message` (type: `str`): the string that is to be encrypted
2. `num_rows` (type: `int`): It is a positive number greater than 0. It represents the number of rows to use when encrypting the message. See **step 3**.

Given the following message and the value of `num_rows` is `7`:

```
"The stars must be jealous. You shine way better than they do!"
```

The **steps** to encode is as follows:

1. Remove all non-alphanumeric characters. For example, whitespaces(`' '`), full stops (`'.'`) and exclamations (`'!'`).

   ```
   "ThestarsmustbejealousYoushinewaybetterthantheydo"
   ```

2. Capitalize all letters.

   ```
   "THESTARSMUSTBEJEALOUSYOUSHINEWAYBETTERTHANTHEYDO"
   ```

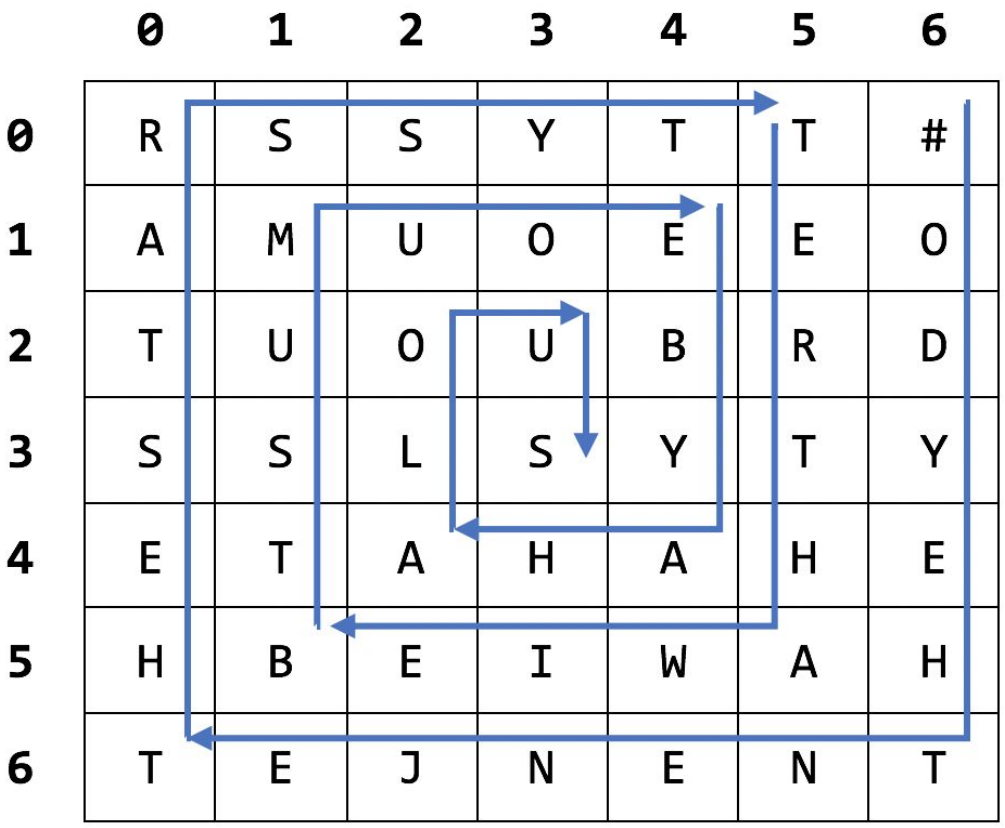
3. Write out the letters in a grid.
   a. from column 0, row n (i.e. num_rows - 1) to row 0, then
   b. from column 1, row 0 to row n,
   c. from column 2, row n to row 0 and so on to fill all the letters in the grid

   Below is the message written in a grid where `num_rows` is 7.

	0	1	2	3	4	5	6
0	R ↑ →	S ↓	S ↑ →	Y ↓	T ↑ →	T ↓	# ↑
1	A ↑	M ↓	U ↑	O ↓	E ↑	E ↓	O ↑
2	T ↑	U ↓	O ↑	U ↓	B ↑	R ↓	D ↑
3	S ↑	S ↓	L ↑	S ↓	Y ↑	T ↓	Y ↑
4	E ↑	T ↓	A ↑	H ↓	A ↑	H ↓	E ↑
5	H ↑	B ↓	E ↑	I ↓	W ↑	A ↓	H ↑
6	T ↑	E ↓ →	J ↑	N ↓ →	E ↑	N ↓ →	T ↑

Note: If the string does not fill all the cells of the grid, the empty spaces are filled with a `'#'`.

4. Generate your encryption string by starting at the top right corner and going down to the bottom right corner, following a spiral route around the grid from outside to the inside in a clockwise direction.



The encrypted message is

'#ODYEHTNENJETHESTARSSYTTERTHAWIEBTSUMUOEBYAHALOUS'

Note: The characters at the "corners" are bolded to aid reading.

Question 6 (Difficulty Level: ***) [3 marks]

Implement the `expand` function. The function takes in 1 parameter:

1. `compressed`(type: `str`): The compressed string that is to be expanded.

The `compressed` parameter is a string consisting of uppercase and lowercase letters or a string of format `n[<value>]` where `n` is an `int` indicating the number of times `value`(the characters within the square brackets, i.e. `[]`) has to be expanded. value could have `n[value]` format string within it. For example,

1. `'2[a]'` will be expanded to `'aa'`
2. `'3[abc]'` will be expanded to `'abcabcabc'`
3. `'2[3[abc]Z]'` will be expanded to `'abcabcabcZabcabcabcZ'`

   (`value` can contain 0 or more occurrences of the `'n[value]'`)
4. `'2[3[ab2[c]]Z]'` will be expanded to `'abccabccabccZabccabccabccZ'`

Note:

1. `n` is a positive number greater than `0`.
2. `value` contains only uppercase(`'A' - 'Z'`) and lowercase letters (`'a' - 'z'`) or occurrences of `'n[value]'`
3. `compressed` contains only uppercase or lowercase letters, or occurrences of `'n[value]'`

END

EMPTY PAGE

EMPTY PAGE