Lab Test 1 [25 marks]

Coding time: 1 hour and 30 mins

Instructions on how to download the resource files:

1. Download from eLearn a resource file called **your_email_id.zip** under Content → Lab Test 1.

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, 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: QIAN Tiao She
# Email ID: tiaoshe.qian.2018
```

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 tiaoshe.qian.2018@sis.smu.edu.sg, you should name the archive as tiaoshe.qian.2018.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 a function called **`get_color`** in **q1.py**.

- This function takes a parameter (type: **`str`**) called **`code`**.
- The function ***returns a string*** that is either **`'Red'`**, **`'Green'`**, **`'Blue'`** or **`'Invalid'`**, based on the value of **`code`**. The mapping from **`code`** to the returned value is shown below:

`code`	Return Value
`'R'` or **`'r'`**	**`'Red'`**
`'G'` or **`'g'`**	**`'Green'`**
`'B'` or **`'b'`**	**`'Blue'`**
any other value (e.g., **`'Red Riding Hood'`**, **`'greet'`**)	**`'Invalid'`**

E.g. 1: If the function is invoked like this:

```
print(get_color ('r'))
```

the statement generates the following output:

```
Red
```

E.g. 2: If the function is invoked like this:

```
print(get_color('G'))
```

the statement generates the following output:

```
Green
```

E.g. 3: If the function is invoked like this:

```
print(get_color('Blue Sky'))
```

the statement generates the following output:

```
Invalid
```

E.g. 4: If the function is invoked like this:

```
print(get_color(''))
```

the statement generates the following output:

```
Invalid
```

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

Implement a function called **`represent_numbers`** in **q2.py**.

- This function takes two parameters: **`start`** and **`end`**. You can assume that both parameters are integers.
- The function ***returns a string*** that represents all the integers between **`start`** (inclusive) and **`end`** (inclusive) in the following way:
    - Each integer **`n`** is represented by **`m`** ***dashes*** (**`'-'`**), where **`m`** is the absolute value of **`n`** (i.e., |**`n`**|). In other words, if **`n`** is positive or 0, **`m`** is the same as **`n`**. If **`n`** is negative, **`m`** is **`-n`**.

      For example,
        - **3** is represented as `'---'`.
        - **-5** is represented as `'-----'`.
        - **0** is represented as `''` (empty string).

    - There is a ***hash key*** (**`'#'`**) between the representations of two integers.
- If **`end`** is smaller than **`start`**, the function returns an empty string.

E.g. 1: If the function is invoked like this:

```
print(represent_numbers(1, 5))
```

the statement generates the following output:

```
-#--#---#----#-----
```

Note: The integers 1, 2, 3, 4 and 5 are represented as −, −−, −−−, −−−− and −−−−−.

E.g. 2: If the function is invoked like this:

```
print(represent_numbers(3, 5))
```

the statement generates the following output:

```
---#----#-----
```

E.g. 3: If the function is invoked like this:

```
print(represent_numbers(-3, 1))
```

the statement generates the following output:

```
---#--#-##-
```

Note: -3, -2 and -1 are represented as −−−, −− and −, respectively.

E.g. 4: If the function is invoked like this:

```
print(represent_numbers(1, 1))
```

the statement generates the following output:

```
-
```

E.g. 5: If the function is invoked like this:

```
print('[' + represent_numbers(3, 1) + ']')
```

the statement generates the following output:

```
[]
```

Note: '[' and ']' are part of the print statement above.

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

Implement a function called **`mask_out`** in **q3.py**.

- The **`mask_out`** function takes in three parameter:
  - **`sentence`** (type: `str`)
  - **`banned`** (type: `str`)
  - **`substitutes`** (type: `str`)

- This function ***returns a string*** that is a copy of **`sentence`**, except that if a character in **`sentence`** is found in **`banned`**, this character will be replaced with a corresponding character in **`substitutes`** whose index in **`substitutes`** is the same as the index of the banned character in the parameter **`banned`**.

  For example, if **`banned`** is **`'abc'`** and **`substitutes`** is **`'xyz'`**, then every **`'a'`** in **`sentence`** is replaced with **`'x'`**, every **`'b'`** is replaced with **`'y'`**, and every **`'c'`** is replaced with **`'z'`**.

- You can assume the following:

  - **`banned`** contains unique characters, i.e., there are no duplicated characters in **`banned`**.
  - **`substitutes`** contains at least one character.
  - **`banned`** is either of the same length as **`substitutes`** or longer than **`substitutes`**. When **`banned`** is longer than **`substitutes`**, the additional characters in **`banned`** will be replaced with the first character in **`substitutes`**. For example, if **`banned`** is **`'abcde'`** and substitutes is **`'xy'`**, then **`'c'`**, **`'d'`** and **`'e'`** will be replaced by **`'x'`**.

E.g. 1: If the function is invoked like this:

```
print(mask_out('pineapple', 'e', '#'))
```

the statement generates the following output:

```
pin#appl#
```

E.g. 2: If the function is invoked like this:

```
print(mask_out('pineapple', 'aeiou', '#$%^&'))
```

the statement generates the following output:

```
p%n$#ppl$
```

E.g. 3: If the function is invoked like this:

```
print(mask_out('pineapple', 'aeiou', '#$'))
```

the statement generates the following output:

```
p#n$#ppl$
```

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

Implement a function called **`print_dancing_string`** in **q4.py**.

- The **`print_dancing_string`** function takes two parameters:
  - **`sentence`** (type: `str`)
  - **`start`** (type: `str`)
- This function ***prints out*** all the characters in **`sentence`** in three rows in a zigzag pattern, enclosed by vertical bars on both sides, as shown by the examples below. Depending on the value of **`start`**, the zigzag pattern starts at different positions.
  - If **`start`** is `'T'`, then the zigzag pattern starts from the top row.
  - If **`start`** is `'M'`, then the zigzag pattern starts from the middle row and moves upward first.
  - If **`start`** is `'B'`, then the zigzag pattern starts from the bottom row.
  - You can assume that the value of **`start`** is always `'T'`, `'M'` or `'B'`.

E.g. 1: If the function is invoked like this:

```
print_dancing_string('a', 'T')
```

the statement generates the following output:

```
|a|
| |
| |
```

E.g. 2: If the function is invoked like this:

```
print_dancing_string('abcdefghi', 'T')
```

the statement generates the following output:

```
|a   e   i|
| b d f h |
|  c   g  |
```

E.g. 3: If the function is invoked like this:

```
print_dancing_string('abcdefghi', 'M')
```

the statement generates the following output:

```
| b   f   |
|a c e g i|
|   d   h |
```

E.g. 4: If the function is invoked like this:

```
print_dancing_string('abcdefghi', 'B')
```

the statement generates the following output:

```
|  c   g  |
| b d f h |
|a   e   i|
```

E.g. 5: If the function is invoked like this:

```
print_dancing_string('', 'T')
```

the statement generates the following output:

```
||
||
||
```

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

Implement a function called **`expand`** in **q5.py**.

- The **`expand`** function takes in a parameter called **`text`** (type: `str`).
- The string **`text`** always consists of uppercase letters, lowercase letters, spaces, and segments of the form "`&start-end`", where `start` and `end` are non-negative integers, and `end` is always greater than or equal to `start`. We call these segments ***index references***.

  For example, **`text`** could be `'abCD &1-3xyz&2-15'`, which has two ***index references***, namely, `'&1-3'` and `'&2-15'`.
- The function expands **`text`** by replacing those ***index references*** with characters from other parts of **`text`**, using the index numbers in the ***index references***.
- To perform the expansion, you need to do the following:
  - Assign index numbers starting from 0 to all characters in **`text`** excluding the ***index references***. For example, if **`text`** is `'ABC &5-7 XYZ'`, then the assigned index numbers should be as follows:

Assigned index number:	0	1	2	3					4	5	6	7
Character in text:	A	B	C		&	5	-	7		X	Y	Z

  - To perform the expansion, look for all ***index references*** in **`text`**. For each ***index reference***, replace it with the characters between the corresponding `start` index and `end` index.

    In the example above, `'&5-7'` is replaced with `'XYZ'`. Therefore, `'ABC &5-7 XYZ'` is expanded to `'ABC XYZ XYZ'`.
  - If any value between `start` and `end` (inclusive of `start` and `end`) is an invalid index, a `'?' is used to represent the character at that invalid index`. For example, if **`text`** is `'ABC &6-9 XYZ'`, then the expanded string will be `'ABC YZ?? XYZ'` (because index numbers 8 and 9 are invalid).

E.g. 1: If the function is invoked like this:

```
print(expand('ABC &5-7 XYZ'))
```

the statement generates the following output:

```
ABC XYZ XYZ
```

E.g. 2: If the function is invoked like this:

```
print(expand('&0-3&4-5ABC XYZ'))
```

the statement generates the following output:

```
ABC XYABC XYZ
```

E.g. 3: If the function is invoked like this:

```
print(expand('&2-5ABC'))
```

the statement generates the following output:

```
C???ABC
```

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

Implement a function called **`print_snake`** in **q6.py**.

- This function takes two parameters:
  - **`sequence`** (type: `str`): This is an input string. You can assume that it is not empty and it doesn't contain any space.
  - **`w`** (type: `int`): This is the width of the output. You can assume that **`w`** is at least 2.

- The function prints out the characters in **`sequence`** in **`w`** columns, in the shape of a snake, as illustrated by the examples below.

  Note that
  - the "tail" of the snake is always at the ***bottom right corner*** of the output, and
  - each vertical segment of the snake has a height of 3.

E.g. 1: If the function is invoked like this:

```
print_snake('abcdefghijkl', 4)
```

the statement generates the following output:

E.g. 2: If the function is invoked like this:

```
print_snake('abcdefghijkl', 3)
```

the statement generates the following output:

```
a
bcd
  e
hgf
i
jkl
```

E.g. 3: If the function is invoked like this:

```
print_snake('abcdefghijkl', 5)
```

the statement generates the following output:

```
    a
fedcb
g
hijkl
```

E.g. 4: If the function is invoked like this:

```
print_snake('abcdefghijk', 4)
```

the statement generates the following output:

```
   a
   b
fedc
g
hijk
```