SMU SINGAPORE MANAGEMENT UNIVERSITY	**IS111 – Lab 7** File Handling

Instructions

You should attempt the questions using Visual Studio Code. Download and extract the files from the **Lab7_starting_code.zip**. You should obtain several *.py and *.txt files to be used for some of the lab questions.

Q1: Reading Files

Part (a) [*]

In the given file `q1a.py`, define a function called `print_alternate_lines` that takes in the name of a file as its parameter. The function prints out alternate lines from the file, starting from the first line. That is, the function prints out the first line, the third line, the fifth line, etc. of the file until the file ends.

For example, if the file looks like the following:

```
1
22
333
4444
4444
333
22
1
```

The function should print out the following output:

```
1
333
4444
22
```

Use the test cases provided to test your code.

Part (b) [**]

In the given file `q1b.py`, define a function called `print_alternate_columns` that takes in the name of a file as its parameter. Each line of the file contains several columns, separated by the symbol `'&'`. The function prints out alternate columns of the file, starting from the first column. That is, the function prints out the first column, the third column, the fifth column, etc., still separated by `'&'`.

You can assume that each line contains at least one column. However, different lines may have different numbers of columns.

For example, if the file looks like the following:

```
apple & orange & pear
banana & cherry & durian & mango & starfruit
watermelon & peach
```

The function should print out the following output:

```
    apple & pear
    banana & durian & starfruit
    watermelon
```

Use the test cases provided to test your code.

Q2: Universities

You are given a file that contains exactly 4 columns in each row, separated by tabs. The 4 columns represent the following information:

```
name_of_a_university    acronym    year_founded    number_of_undergraduate
_students
```

For example, the file may look like the following:

```
    National University of Singapore    NUS    1905    27972
    Nanyang Technological University    NTU    1981    24300
    Singapore Management University    SMU    2000    7827
    University of California, Berkeley    Berkeley    1868    29310
    Harvard University    Harvard    1636    6700
    University of California, Los Angeles    UCLA    1919    30873
```

Part (a) [*]

In the given file `q2a.py`, define a function called `get_universities_founded_before()`. This function takes in the name of a file as described above and an integer called year as its parameters. The function returns a list of strings that represent the names of the universities in the given file which were founded **before** the specified `year`.

For example, if the file is as shown above and `year` is `1905`, then the function should return `['University of California, Berkeley', 'Harvard University']`.

Use the test cases provided to test your code.

Part (b) [*]

In the given file `q2b.py`, define a function called `get_average_num_students()`. This function takes in the name of a file as described above and returns the average number of undergraduate students of all the universities in the given file. If the file is empty, the function returns 0.0.

Use the test cases provided to test your code.

Part (c) [*]

In the given file q2c.py, define a function called search_with_keyword(). This function takes in the name of a file as described above and a string that represents a keyword. The function returns a list of strings that represent the **acronyms** of the universities whose **full names** contain the given keyword.

For example, given the file as shown above, if the keyword is 'California', then the function should return ['Berkeley', 'UCLA'].

Your search needs to be **case-insensitive**. So searching for 'california' or 'California' should give you the same results.

Use the test cases provided to test your code.

Q3: Temperatures [**]

You are given a file called 'temperatures.txt' that stores some patients' temperature readings. Each line of the file is about a single patient. The patient's name is displayed first followed by a sequence of temperature values. The name and the different temperature values are separated by tabs. Note that the numbers of temperature readings are different for different patients.

Create q3.py and write a piece of code that reads in this file and prints out the minimum and maximum temperature readings of each patient.

You can assume that for each patient, there is at least one temperature reading in the file.

Your program should display the following output:

```
Kate: 36.3, 37.6
Larry: 36.1, 38.4
Jerry: 37.6, 38.9
```

Q4: Transactions

You are given a file called "transactions.txt" that contains the transaction history of a person's credit card. Each line of the file contains the following three columns: (1) A date in dd-mm-yyyy format. (2) A text description of the transaction. (3) The amount of the transaction. The three columns are separated by tabs.

Note that these transactions are not sorted in any particular order.

Part (a) [**]

Create q4a.py and write a piece of code that reads this file and writes to an output file called "transactions-output-1.txt". The output file should contain those transactions where the transacted amount is $30.00 or above.

Your generated file should contain the following lines:

```
09-12-2016    NTUC FairPrice   $59.43
08-01-2017    Shell Station    $47.56
```

```
09-03-2017    SingTel $31.50
08-01-2017    SingTel $32.10
21-12-2016    SIA     $546.90
19-03-2017    NTUC FairPrice   $108.32
```

Part (b) [***]

Create `q4b.py` and write a piece of code that reads this file and writes to an output file called `"transactions-output-2.txt"`. The output file should group the transactions by months, with the total amount per month shown. The order of the months doesn't matter.

The output file looks like the following:

```
03-2016: total transaction amount is $21.3
    Popular    $21.30

12-2016: total transaction amount is $606.3299999999999
    NTUC FairPrice    $59.43
    SIA    $546.90

01-2017: total transaction amount is $79.66
    Shell Station    $47.56
    SingTel    $32.10

03-2017: total transaction amount is $139.82
    SingTel    $31.50
    NTUC FairPrice    $108.32

02-2017: total transaction amount is $49.7
    Popular    $25.40
    Shaw Theatres    $24.30
```

Q5: Matrix

In mathematics, a square matrix refers to a square array of numbers. For example, the following is a 3×3 square matrix:

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

Part (a) [**]

In the given file `q5a.py`, define a function called `get_matrix` that takes in a string, which is the name of a file that stores such a square matrix. For example, to store an n×n square matrix, the file has n rows, where each row contains nn numbers separated by spaces.

The function returns a **list of lists of numbers**, representing the square matrix stored in the file.

Use the test cases provided to test your code.

Part (b) [***]

In the given file `q5b.py`, define a function called `get_matrix_transpose` that takes in a string, which is the name of a file that stores such a square matrix. The function returns a list of lists representing the **transpose** of the square matrix stored in the file.

For example, given the matrix above, its transpose is the following:

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

Use the test cases provided to test your code.

Q6: [***]

In this question, you need to read in a Python script and extract all the docstrings together with their function headers.

Assume the Python script you're processing contains several function defintions. For each function defintion, assume it is strictly written in the following way:

- The function header (i.e., `def name_of_function(list_of_parameters):`) takes one line.
- Immediately below the function header, a docstring is presented to explain the function.
    - The docstring may take one line or multiple lines.
    - It always starts with `"""` (i.e., three double quotes), but there are 4 spaces in front of the `"""` for indentation purpose.
    - It always ends with `"""`, but there may be some additional spaces behind the `"""`.

For example, a file may look like the following:

```
def my_test_func(param1, param2):
    """
    This is a test function that takes in two parameters and returns a
number.
    The two parameters are compared. 0 or 1 is returned.
    """
    if (param1 < param2):
        return 0
    else:
        return 1
def another_function():
    """This is a function without any parameter."""
    pass
def third_funct():
    """This is a function
    that has two lines of docstring."""
    print("hello")
```

You can assume that inside the Python script you're processing, the keyword `def` occurs only for function definitions. Also, the pattern `"""` does not occur anywhere else in the file except for docstrings.

Create `q6.py` and write a function called `extract_functions()` that takes in two parameters:

- `python_filename`: The name of a Python script, with .py as its extension.
- `output_filename`: The name of an output file.

The function should read the Python script, extracts all function headers and docstrings, and write them into the output file. In the output file, the functions are numbered. The doctrings are indented with 4 spaces. The keyword `def`, the colon, and `"""` are removed. A blank line is inserted after each function.

For example, given the file above, the generated output file should contain the following:

```
1. my_test_func(param1, param2)
    This is a test function that takes in two parameters and returns a
    number.
    The two parameters are compared. 0 or 1 is returned.

2. another_function()
    This is a function without any parameter.

3. third_funct()
    This is a function
    that has two lines of docstring.
```

You are given a file called "python_script.py" that can be used to test your program.

Q7 [****]

[This question is adapted from the book “The practice of Computing using Python” - Programming Project on Page 340.]

There is a belief that humans are able to read even when the order of the letters are misplaced. Try reading this:

```
I cnduo't bvleiee taht I culod uesdtannrd waht I was rdnaieg
```

Interesting read here: https://www.mrc-cbu.cam.ac.uk/people/matt.davis/cmabridge/ .

Create `q7.py` and write a program that reads text from a file called `"talk.txt"` and prints the content on the output screen scrambling the words in the content as described below:

- Scramble every word in the text. For each word, **scramble the letters in the middle only, leaving untouched the first and the last letters**.
- You may assume that every two consecutive words are separated by a single space. Maintain single space between every two consecutive words in the scrambled version.
- Challenge yourself to leave the punctuations [, ' ! .] in the content untouched in the original place.

As an example, if the text reads “Programming is really fun”, then one version after scrambling could be “Pgmnarirmog is rlelay fun”. Note that since we scramble only the letters in the middle of each word, the two words “is” and “fun” in the example are not scrambled.

It helps if you define functions to help you with this question.

Hint: To scramble the letters in the middle of a 6-lettered word, for example, you could take all the letters in the indices ranging from 1 to 4, and put them in random positions ranging from 1 to 4. A 6-letter word after scrambling should contain exactly the same six letters. Note that after you randomly scramble the letters in the middle, if the resulting word happens to be the same as the original word, you should discard this scrambled word and try again.
You probably want to use the `random` module to help you with scrambling letters.

Some partial output from a sample of your code with the given `"talk.txt"` file may look like the following:

```
    We do not need miagc to chgnae the wlrod, we crray all the pwoer we ne
    ed inisde oleeursvs aedlray: we have the pweor to imngiae betetr.
```

Note that your code will generate different output because of the randomness in the code.