**Week 12 In-Class Exercises**

## Q1: Emails

1. [ \* ] Implement a function called retrieve\_email(). The function takes in **two** **parameters**:
   * email\_dict (type: dict): a dictionary where each key is a string representing a name and each value is another string representing an email address
   * name (type: str): a person’s name

The function returns the email address of the specified person or None if the person cannot be found in the dictionary.

For example, retrieve\_email({'Jack':'jack@gmail.com', 'Peter':'peter@smu.edu.sg'}, 'Jack') returns 'jack@gmail.com', while retrieve\_email({'Jack':'jack@gmail.com', 'Peter':'peter@smu.edu.sg'}, 'John') returns None.

1. [ \*\* ] Implement a function called create\_email\_dict(). This function takes in a list of strings representing people’s names and their email addresses. In this list, a person’s name is always followed by his/her email address, and it is assumed that each person has only a single email. It is also assumed that the names are unique.

For example, the list may look like the following:

["Jack", "jack@gmail.com", "Mary", "mary.loh@smu.edu.sg", "John", "john.smith@microsoft.com"]

You can assume that the input list is always properly constructed in the above format.

The function will re-organize the information stored in the input list so that each person’s name is mapped to his/her email address. The function returns a dictionary that stores all the mappings.

For example, create\_email\_dict(["Jack", "jack@gmail.com", "Mary", "mary.loh@smu.edu.sg", "John", "john.smith@microsoft.com"]) returns the following dictionary:

{'Jack': 'jack@gmail.com', 'Mary': 'mary.loh@smu.edu.sg', 'John': 'john.smith@microsoft.com'}

1. [ \*\* ] Implement a function called add\_new\_email(). The function takes in **two parameters**:
   * email\_dict (type: dict): a dictionary that maps names to emails, the same as in Part (a)
   * pair (type: tuple): a tuple of two elements, representing a person’s name and his/her email address

The function tries to add the tuple as a new (key, value) pair to the dictionary.

If the person already exists in the dictionary, the new value is used to replace the old value, and ***the old value is returned*** by the function.

If the person does not exist in the dictionary originally, the function adds the new (key, value) pair and returns None.

# Test cases used to test your function

print('\nTestcase 1')

print('-' \* 10)

my\_dict = {'Jack':'jack@gmail.com', 'Peter':'peter@smu.edu.sg'}

new\_name = 'Peter'

new\_email = 'peter@sis.smu.edu.sg'

pair = (new\_name, new\_email)

print("Expected returned value: peter@smu.edu.sg")

print('Actual returned value :  ' + str(add\_new\_email(my\_dict, pair)))

print("\nExpected dictionary: {'Jack': 'jack@gmail.com', 'Peter': 'peter@sis.smu.edu.sg'}")

print('Actual dictionary :  ' + str(my\_dict))

print('\nTestcase 2')

print('-' \* 10)

my\_dict = {'Jack':'jack@gmail.com', 'Peter':'peter@smu.edu.sg'}

new\_name = 'Mary'

new\_email = 'mary@sis.smu.edu.sg'

pair = (new\_name, new\_email)

print("Expected returned value: None")

print('Actual returned value :  ' + str(add\_new\_email(my\_dict, pair)))

print("\nExpected dictionary: {'Jack': 'jack@gmail.com', 'Peter': 'peter@smu.edu.sg', 'Mary': 'mary@sis.smu.edu.sg'}")

print('Actual dictionary :  ' + str(my\_dict))

## Q2: Count Strings [ \*\*\* ]

Write a program that prompts the user continuously for strings until the user enters an empty string. The program then prints out the number of strings entered for each starting character that has been observed. Note that for starting characters that are letters, uppercase and lowercase letters are merged. For the final output, it doesn’t matter in what order the numbers for different starting characters are shown. A sample run of the program is shown below:

Enter a string > Singapore

Enter a string > management

Enter a string > school

Enter a string > SIS

Enter a string > $500

Enter a string > 3 apples

Enter a string > Python

Enter a string > programming

Enter a string >

You've entered

3 strings starting with 'S' or 's'

1 string starting with 'M' or 'm'

1 string starting with '$'

1 string starting with '3'

2 strings starting with 'P' or 'p'

Note the following:

* In the output, if the count is 1, you should display '1 **string** starting with...' instead of '1 **strings** starting with...'.
* If the first character is a letter, you should display "…starting with 'X' or 'x'" (where X is a one of the 26 English letters).

## Q3: HTML [ \*\*\* ]

Webpages are in HTML format. In an HTML file, texts surrounded by tag pairs <b> and </b> will be shown in bold font on a Web browser.

For example, given the following text in HTML format:

|  |
| --- |
| This is an <b>example</b> of some <b>text</b> in HTML. <b>Some of the segments</b> will be <b>shown in bold</b>. |

The text segments 'example', 'text', 'Some of the segments' and 'shown in bold' will be bold.

Define a function called extract\_bold\_texts(). The function takes in a single string called html\_text. It returns a list of strings that are segments inside html\_text which are surrounded by <b> and </b>.

E.g.,

* extract\_bold\_texts('<b>ABC</b> abc <b>def 123 </b><b></b>0000') returns ['ABC', 'def 123 ', '']. (Note that the last string is an empty string.)
* extract\_bold\_texts('A piece of text without tags.') returns [].