**Week 5 : In-Class Exercises (Lists)**

**Q1: Code Tracing [ \*\* ]**

What’s the output of the following code?

def modify\_list(my\_list):

    for index in range(len(my\_list)):

        x = my\_list[index]

        if len(x) > 5:

            my\_list[index] = x[0:5]

str\_list = ["IS111", "Python", "Programming", "List"]

modify\_list(str\_list)

print(str\_list)

What’s the output of the following code?

def modify\_list(my\_list):

    for element in my\_list:

        if len(element) > 5:

            element = element[0:5]

str\_list = ["IS111", "Python", "Programming", "List"]

modify\_list(str\_list)

print(str\_list)

**Q2: Email Extractor [ \*\* ]**

**Part I**

Implement a function called extract\_email\_id() that extracts email ID from an email address. The function takes in a string as a parameter. This string is supposed to be an email address. The function **returns** the front part of the email address before the ‘@’ symbol.

For example,

* extract\_email\_id("jerry.lee@sis.smu.edu.sg") returns the string "jerry.lee".
* extract\_email\_id("alan\_wong@gmail.com") returns the string "alan\_wong".
* extract\_email\_id("alan\_wong.com") returns "".
* extract\_email\_id("") returns "".

If the given string does not contain the ‘@’ symbol, the function returns an empty string. You can assume that the string either contains a single ‘@’ symbol or no ‘@’ symbol.

Use the test code provided below to test your code.

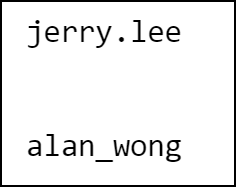
print(extract\_email\_id("jerry.lee@sis.smu.edu.sg"))

print(extract\_email\_id("alan\_wong.com"))

print(extract\_email\_id(""))

print(extract\_email\_id("alan\_wong@gmail.com"))

It should provide the following output:



**Part II**

Implement another function called extract\_multiple\_email\_ids(). This function takes in a string that contains multiple email addresses, separated by semi-colons. (You can assume that semi-colons cannot be used as part of an email address.) The function **prints out** the email IDs one by one in separate rows. The function doesn’t return anything.

For example, calling extract\_multiple\_email\_ids("jerry.lee@sis.smu.edu.sg;alan\_wong@gmail.com;george\_tan@yahoo.com") gives the following output:

jerry.lee  
 alan\_wong  
 george\_tan

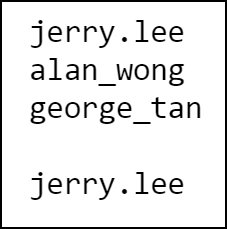
Use the test code below to test your code:

extract\_multiple\_email\_ids("jerry.lee@sis.smu.edu.sg;alan\_wong@gmail.com;george\_tan@yahoo.com")

extract\_multiple\_email\_ids("")

extract\_multiple\_email\_ids("jerry.lee@sis.smu.edu.sg")

It should provide the following output:



Hint: You can use the split() method on a string.

**Q3: Check Username [ \*\*\* ]**

A website allows people to sign up as members, but when choosing their usernames, there are the following restrictions:

* The username cannot be empty nor contain any space.
* Each character in the username must be a lowercase letter, a digit, or one of the following special symbols: \_.!#$%?
* The length of the username cannot exceed 8.

Write a function that checks whether a string is a valid username. The function takes in a string as its parameter and **returns** True or False.

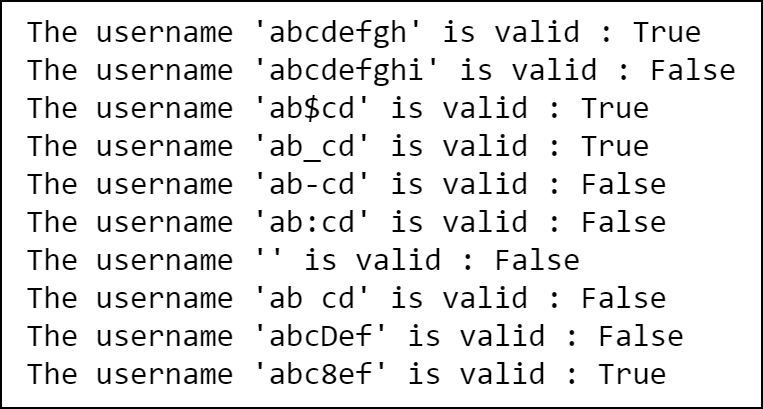
Use the test code provided below to test your function.

username\_list = ['ab{cdefgh','abcdefghi','ab$cd','ab\_cd','ab-cd','ab:cd','','ab cd','abcDef','abc8ef']

for username in username\_list:

    print("The username '" + username +"' is valid : " + str(is\_valid\_username(username)))

It should provide the following output:



**Q4: List of Strings**

1. [ \*\* ] Implement a function called get\_avg\_len() that takes in a list of strings. The function returns the **average length** of all the strings in the given list.

For example, get\_avg\_len(["C", "Java", "Python", "PHP"]) returns 3.5.

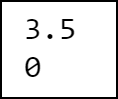
If the list is empty or if the strings are all empty strings, the function returns 0.

Use the test code provided to test your function.

print(get\_avg\_len(["C", "Java", "Python", "PHP"]))

print(get\_avg\_len([]))

It should provide the following output:



1. [ \*\* ] Implement a function called get\_longest\_str() that takes in a list of strings. The function returns the string that is the **longest** among all the strings in the given list. If there are multiple strings having the longest length, the function returns the **first** such string in the list.

For example, get\_longest\_str(["C", "Java", "Python", "PHP"]) returns "Python", and get\_longest\_str(["C", "Java", "HTML", "PHP"]) returns "Java".

If the list is empty, the function returns an empty string.

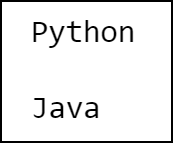
Use the test code provided to test your function.

print(get\_longest\_str(["C", "Java", "Python", "PHP"]))

print(get\_longest\_str([]))

print(get\_longest\_str(["C", "Java", "HTML", "PHP"]))

It should provide the following output:



1. [ \*\*\* ] Implement a function called concatenate\_emails(). This function takes in a list of strings, where some of these strings are email address. The function **returns** a string that contains the email addresses separated by semi-colons.

Here a string is considered an email address if it contains exactly one ‘@’ symbol and it does not contain any space.

For example, concatenate\_emails(["IS111", "a @ b", "jerry.lee@sis.smu.edu.sg", "@@@", "alan\_wong@gmail.com", "Python", "george\_tan@yahoo.com"]) returns "jerry.lee@sis.smu.edu.sg;alan\_wong@gmail.com;george\_tan@yahoo.com".

If the list is empty, the function returns an empty string.

Use the test code provided to test your function.

my\_list1 = ["tommy.goh@sis.smu.edu.sg","alan\_wong@gmail.com"]

print(concatenate\_emails(my\_list1))

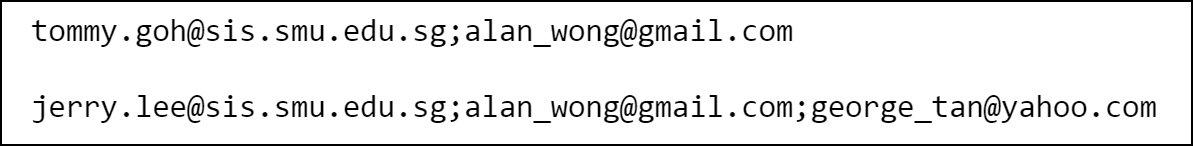
my\_list2 = []

print(concatenate\_emails(my\_list2))

my\_list3 = ["IS111", "a @ b", "jerry.lee@sis.smu.edu.sg", "@@@", "alan\_wong@gmail.com", "Python", "george\_tan@yahoo.com"]

print(concatenate\_emails(my\_list3))

It should provide the following output:



1. [ \*\*\* ] Implement a function called check\_hashtags(). The function takes in a list of strings. The function returns True if all the strings are hashtags, i.e., all the strings start with a ‘#’ symbol and do not contain any space. The function **returns** False if at least one of the strings is not a hashtag.

For example,

* check\_hashtags(["#singapore", "#music", "#travel"]) returns True
* check\_hashtags(["#singapore", "#music album", "#travel"]) returns False
* check\_hashtags(["singapore", "#music", "#travel"]) returns False

If the list is empty, the function returns False.

Use the test code provided to test your function.

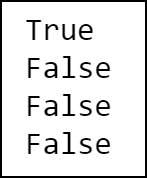
print(check\_hashtags(["#singapore", "#music", "#travel"]))

print(check\_hashtags([]))

print(check\_hashtags(["#singapore", "#music album", "#travel"]))

print(check\_hashtags(["singapore", "#music", "#travel"]))

It should provide the following output:



**Q5: Movies [ \*\*\* ]**

Take a look at the starting code movies.py that’s given to you. Inside the file, you are given a list of tuples, where each tuple represents the title, genre and duration (in minutes) of a movie.

Inside another file called movies\_utility.py, implement the following functions:

* get\_average\_duration(): This function takes in a list of tuples as described above. The function returns the average duration of all the movies in the list. If the list is empty, the function returns 0.0.
* get\_num\_movies\_of\_genre(): This function takes in a list of tuples as described above together with a string that indicates a genre. The function returns the number of movies belonging to that genre in the give list. If the list is empty, the function returns 0.
* get\_title\_of\_longest\_movie(): This function takes in a list of tuples as described above and returns the title of the movie whose duration is the longest among all the movies in the list. If the list is empty, the function returns an empty string.
* get\_movies\_with\_keyword(): This function takes in a list of tuples as described above together with a string that represents a keyword. The function returns a new list of tuples where each tuple still represents a movie. The returned list contains those movies in the original list whose titles have the specified keyword as a substring. If the original list is empty, the function returns an empty list.