**Lab 5 – Lists (Part I)**

**Q1: Initials [ \*\* ]**

Write a program that prompt for the names of people attending a meeting. After that, print out the initials of these people.

You can assume that each participant’s name consists of a sequence of words separated by a single space. You can assume that each person’s name contains at least one word. The initial of a person contains the first letters of all the words in that person’s name.

A sample run of the program looks like the following:

How many people will attend the meeting? **5**  
Participant 1: **John Smith**  
Participant 2: **Jerry Lee Xiong Yi**  
Participant 3: **Eric Wong Kee Wei**  
Participant 4: **Felicia Koh**  
Participant 5: **Julia Chan**

The initials of the participants are as follows:  
JS  
JLXY  
EWKW  
FK  
JC

**Q2: List of Numbers [ \*\* ]**

Define the following functions that handle a list of numbers:

1. Define a function called get\_leap\_years(). This function takes in a list of numbers that indicate years. It returns a list that contains only those years that are leap years. For definition of leap years, see the following link:

<https://en.wikipedia.org/wiki/Leap_year#Algorithm>

For example, get\_leap\_years([2018, 2000, 1800, 1900, 2011, 2020]) returns the list [2000, 2020].

1. Define a function called all\_older\_than(). The function takes two parameters: (1) A list of integers called age\_list, where each element indicates the age of a person. (2) An integer called n, which is a threshold. The function returns True if ALL the age values in age\_list are larger than n, and False otherwise.

For example, all\_older\_than([24, 36, 45, 21], 20) returns True, and all\_older\_than([24, 36, 45, 21], 23) returns False.

If age\_list is empty, the function returns True.

1. Define a function called get\_sum\_of\_multiples(). The function takes in two parameters: (1) A list of integers called int\_list. (2) An integer n. The function returns an integer, which is the sum of all the integers in int\_list that are multiples of n, i.e., that are divisible by n. You can assume that n is always a positive integer.

For example, get\_sum\_of\_multiples([2, 4, 5, 9, 13, 15], 3) returns 24 (sum of 9 and 15), and get\_sum\_of\_multiples([2, 4, 5, 9, 13, 15], 5) returns 20 (sum of 5 and 15).

1. Define a function called get\_prime\_numbers(). The function takes in two parameters: (1) A list of integers called num\_list. (2) A string sep that serves as a separator. The function returns a **string** that contains the prime numbers inside num\_list, separated by sep.

For example, get\_prime\_numbers([2, 4, 7, 9, 11, 16, 19, 21], '-') returns the string "2-7-11-19".

See the following link for the definition of prime numbers:

<https://en.wikipedia.org/wiki/Prime_number>

**Note:** You should write a function to help you check whether a number is a prime number.

1. Define a function called calculate\_sums(). The function takes in a list of numbers, call num\_list. It returns a new list of numbers that has the same length of num\_list. The n’th element of the returned list is the sum of the first n numbers in num\_list.

For example, calculate\_sums([2, 3, 6, 1, 5]) returns the list [2, 5, 11, 12, 17]. (Here 5 is the sum of 2 and 3; 11 is the sum of 2, 3 and 6; 12 is the sum of 2, 3, 6 and 1; and 17 is the sum of 2, 3, 6, 1 and 5.)

If the list num\_list is empty then return an empty list.

**Q3: Shopping Cart [ \*\* ]**

You will be implementing a few functions dealing with an item\_list. Each element of item\_list is a tuple with three values: the name of an item, its unit price, and the quantity of the item in the shopping cart.

For example, item\_list may look like the following:

[("milk", 5.45, 2), ("eggs", 2.45, 1), ("shampoo", 8.90, 2)]

1. Define a function called calculate\_total\_price() that takes a parameter called item\_list, as described above. The function returns the total price (unit price multiplied by quantity) of all the items in the shopping cart.
2. Define a function called get\_items() that takes a parameter called item\_list, as described above. The function returns a list of strings, which are the names of the items in item\_list. For example, get\_items([("milk", 5.45, 2), ("eggs", 2.45, 1), ("shampoo", 8.90, 2)]) returns ["milk", "eggs", "shampoo"].
3. Define a function called get\_items\_more\_expensive\_than(). The function takes in two parameters: (1) item\_list. (2) A float value called min\_price. The function returns a list of tuples that represents those items in item\_list whose unit price is above min\_price. Each tuple in the returned list contains the name of an item and its unit price. For example, get\_items\_more\_expensive\_than([("milk", 5.45, 2), ("eggs", 2.45, 1), ("shampoo", 8.90, 2)], 3.0) returns [("milk", 5.45), ("shampoo", 8.9)]