**Week 9 In-Class Exercises (Extra)**

**(while-loops)**

**Q1: Strings (\*\*)**

Define a function get\_strings that takes in two parameters: (1) A list of strings called str\_list. (2) A positive integer value t. The function returns a list that contains the first n strings in str\_list such that ***the total length*** of all the n strings is **strictly** larger than t. The function should find the smallest n that satisfies the condition above and return those n strings. If the total length of all the strings in str\_list is still not larger than t, then the function returns a list that contains all the strings in str\_list.

For example, suppose str\_list is ['a', 'bc', 'defgh', 'ij', 'k', 'lmn'], and t is 9, then the function should return ['a', 'bc', 'defgh', 'ij'], because these four strings together have a total length of 10, which is larger than 9, but if we do not include 'ij', then total length is 8, which is not enough.

**Hint:** Use an index value and a while-loop to help you go through str\_list.

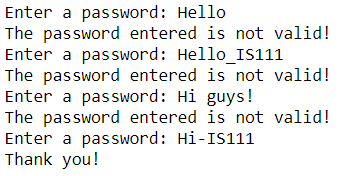
**Q2: Password [\*\*\*]**

Prompt the user for a password that he/she wants to choose. A valid password should satisfy the following condition:

* 1. It has to be at least 6-character long.
  2. It cannot exceed 10 characters.
  3. It cannot contain any space.
  4. It has to contain at least a lowercase letter.
  5. It has to contain at least an uppercase letter.
  6. It has to contain at least a digit (0 to 9).
  7. It has to contain at least one character that is not a letter or a digit.

Keep prompting the user until the password entered is valid.

Your code should provide the following output:



**Q3: Merge Lists [\*\*\*]**

Define a function called merge\_by\_age() that merges two lists of people. The function takes in two lists, where each list contains tuples representing people with their name and age information. **Each list is sorted in increasing order of the people’s age.** For example, list1 may look like [("John", 12), ("Kate", 15), ("Henry", 35)], and list2 may look like [("Mike", 18), ("Scott", 20), ("Joseph", 48), ("Larry", 54)]. The function merges the two lists into a single list in which the people are still ordered by their age. For example, merge\_by\_age(list1, list2) should return [("John", 12), ("Kate", 15), ("Mike", 18), ("Scott", 20), ("Henry", 35), ("Joseph", 48), ("Larry", 54)].

**Q4: Group Numbers [\*\*\*]**

Define a function called group\_numbers. The function takes in two parameters: (1) A list of integers. (2) A threshold. The function tries to group the numbers in the list such that the sum of the numbers in each group does not exceed the specified threshold. The numbers should be grouped sequentially. The function should try to create as few groups as possible. The groups are returned as a list of lists.

For example, suppose the list of numbers given to the function is [1, 3, 2, 4, 3, 2, 3, 6]. Suppose the threshold is 6. The function will group the first three numbers, 1, 3 and 2, together, because their sum is exactly 6. The next number, 4, cannot be grouped with the subsequent number 3 because their sum would exceed 6.

group\_numbers([1, 3, 2, 4, 3, 2, 3, 6], 6) should return [ [1, 3, 2], [4], [3, 2], [3], [6] ]. We can see that the first three numbers 1, 3 and 2 are grouped together because their sum is 6, which does not exceed the threshold 6. The next number, 4, cannot be grouped with the subsequent number, 3, because 4 + 3 has exceeded the threshold 6.

You can assume that none of the numbers in the list of integers is larger than the threshold. If the number list is empty, the function returns an empty list.