CSC108H/A08H Lists and While Loops Lab

At the end of the lab, please show your work to your TA and return this handout. As usual, we will post the handout on the course website at the end of the week.

Switch driver and navigator frequently.

1 Objectives

- Practice manipulating lists using indexing, slicing and list methods
- Practice using `while`

2 Lists

In this section, you will write short functions or statements that involve lists. In some of the exercises, you will be expected to use list methods so that you can become familiar with the tools available to you. Remember to use the Python functions `dir` and `help` to get information about methods.

1. Type this assignment statement into the Python shell:

   ```
   names = ['Bob', 'Ho', 'Zahara', 'Amitabha', 'Dov', 'Maria']
   ```

   For the following steps, use `names` and slice notation:

   (a) Write an expression that produces this new list: `['Zahara', 'Amitabha', 'Dov']`

   (b) Write an expression that produces this new list: `['Bob']`

   (c) Write an expression that produces this new list: `['Amitabha', Dov', 'Maria']`

2. Given a list `L` and a value `v`, write an expression that removes the first occurrence of `v` from `L`.

3. Write an expression that adds the string `"How are you?"` to the **front** of the list `["I am well."]`.

4. Write code that turns `[2, 4, 99, 0, -3.5, 86.9, -101]` into `[99, 86.9, 4, 2, 0, -3.5, -101]`. You should use just two method calls.

5. Write a function `every_third` that takes a list as a parameter and returns a new list that contains every third element of the original list, starting at index 0. For example, the call `every_third(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]` should return `[1, 4, 7, 10]`. Don't use slice notation.

6. Write a function `every_ith` that takes a list `L` and an integer `i` as parameters and returns a list consisting of every `i`*th* element of `L`, starting at index 0. Don't use slice notation.

Show your TA your work and continue with the `while` exercises in the next section.

3 `while` loops

Use `while` loops to complete the following functions. (Do *not* use `for` loops.) Verify your work. As always, we give the type of the parameter in parentheses. Do not actually use `list` as the name of a parameter or variable.

`display_list(list)`	Print the elements of the given list.
`display_list_even(list)`	Print the elements of the given list that occur at even indices.
`display_list_reverse(list)`	Print the elements of the given list from the end of the list to the front.
`sum_elements(list)`	Sum the elements of the given list of ints, starting from the front of list, until the total is over 100 or the end of the list is reached, and return the sum at that point (as an int).
`duplicates(list)`	Return True if the given list contains at least two adjacent elements with the same value, and return False otherwise.

4 Nested Lists

List elements may be lists themselves. When this happens it is called a *nested list* or a *list of lists*:

```
pets = [["Shoji", "cat", 18], ["Hanako", "dog", 15], ["Sir Toby", "cat", 10],
    ["Sachiko", "cat", 7], ["Sasha", "dog", 3], ["Lopez", "dog", 13]]
```

We can access each element of list `pets` using its index:

```
>>> pets[3]
["Sachiko", "cat", 7]
```

We can also access elements of the inner lists. For example, since `pets[3]` refers to a list, we can use `pets[3]` to access the element at position 2:

```
>>> pets[3][2]
7
```

This is saying that element 2 of element 3 of the list `pets` (the age of the cat named "Sachiko") is 7.

Write the following loops and functions and call each function to verify your work. You may use `for` loops.

1. Write a loop that prints each list from list `pets` on a separate line.
2. Write a loop that prints the second element of each inner list in list `pets` on a separate line.
3. Write a loop that examines list `pets` and computes the number of dogs in the list.
4. Write a loop that examines list `pets` and computes the sum of the ages of the animals in the list. Ages are the third element of the inner lists.
5. Write a function `nested_lengths` that takes a list `L` as a parameter and returns a list of the lengths of the sublists. More formally: for each element `e` in `L`, the returned list contains a corresponding element, `c`, that represents the number of elements in `e`.

Show your TA your work so that you can get credit for the lab.