CSE 160 Section 4 Solutions - Lists

Exercise 1:

After the following lines of code are printed, what values are printed?

```
list_1 = [1, 2, 3, 4, 5]
list_2 = list_1
list_3 = list_1[0:5]
# ^ Note: list_1[:] is considered more general,
# better overall and equivalent to list_1[0:5]

list_2[0] = 98
list_1[4] = 99
print("List 1:", list_1)
print("List 2:", list_2)
print("List 3:", list_3)
```

```
List 1: [98, 2, 3, 4, 99]
List 2: [98, 2, 3, 4, 99]
List 3: [1, 2, 3, 4, 5]
```

Exercise 2:

Given the following code:

```
list_1 = [1, 2, 3, 4]

list_2 = [10, 12]

list_3 = [21, 22, 23, 24]

list_4 = [13, 14, 15]

list_5 = [16, 17, 18, 19, 20, 25, 26]

list_6 = [9, 8, 7, 6, 5]

list_7 = [11]
```

Use only list accesses `[ ]`, `extend()` ,`insert()` , and `reverse()` – and possibly, a `for` loop. DO NOT add, subtract, or perform other mathematical operations on the numbers in the lists.

-- to modify `list_1` so that it contains numbers 1 through 26 in increasing order:

```
print("List 1:", list_1)
```

The command above should print a list that contains numbers 1 through 26 in increasing order.

ONE POSSIBLE SOLUTION:

```
list_6.reverse()
list_1.extend(list_6)
list_2.insert(1, list_7[0])
list_1.extend(list_2)
list_1.extend(list_4)
list_1.extend(list_5)

for item in list_3:
    list_1.insert(-2, item)
```

Exercise 3:

Using the same initial lists as Question 3, redo the problem, but this time you are allowed to use the `sort()` function.

```
list_1.extend(list_2)
list_1.extend(list_3)
list_1.extend(list_4)
list_1.extend(list_5)
list_1.extend(list_6)
list_1.extend(list_7)
list_1.sort()
print(list_1)
```

Exercise 4:

Create a function "`dot_product`" which takes in two lists of integers and returns the dot product of the two lists. You can assume the lists are of equal length, and contain only integer values. To calculate the dot product of one index, find the product of each value in one list, with the value at the same index in the other list. The dot product of the array is the sum of these products. For example, the product of the lists `[1,2]` and `[3,4]` is: `1 * 3 + 2 * 4 = 11`.

```
def dot(list1, list2):
    dotProduct = 0
    for num in range(len(lst1)):
        curNum2 = lst2[num]
        curNum1 = lst1[num]
        dotProduct = dotProduct + (curNum2 * curNum1)
     return dotProduct
```

Challenge Problem:

For each of the following questions, implement a function that returns the answer to the question. For each question, follow this procedure:

1. Identify a good name for the function.
2. Identify the return value.
3. Identify any necessary parameters.
4. Write the function definition.
5. Write the function's docstring.
6. Describe, on paper, in words, or in your head, the algorithm you'll use to solve the problem.
7. Implement the function.

Note: Practice effective programming by making sure you understand your approach in step 6 before implementing the code in step 7. It is also a good idea to think of test cases that validate your function input/outputs.

You and your friends have set up "best friends" lists. Each of your lists contains the name of the list owner followed by the names of his/her top friends. Your name in your friends' lists is "me".

```
my_friends = ["me", "Emily", "John", "Ed", "Louise", "Tom"]
emily_friends = ["Emily", "me", "Rob", "Sue", "Alice", "Eric"]
john_friends = ["John", "Ed", "Rob", "Sue", "Eric", "Meg", "Emily"]
ed_friends = ["Ed", "me", "Tom", "John", "Emily","Sue", "Liam"]
```

```
louise_friends = ["Louise", "me", "Alice", "Sue", "Emily", "Meg"]
tom_friends = ["Tom", "John", "Alice", "Ed", "Louise", "Emily", "me"]
```

Friends (a list of lists) is constructed as follows. Your list is at position `friends[0]` .

```
friends = [my_friends, emily_friends, john_friends, ed_friends,
louise_friends, tom_friends]
```

1. Write a function to return the index of the first occurrence of name in a friend list, given the name and a friend list. If the name is not in the list, the function will return `-1` .

```
def find_index(my_list, my_value):
    for index in range(len(my_list)):
        if my_list[index] == my_value:
            return index
    return -1
```

2. Write a function that given a name and `friends`(the list of friend lists), will return the index of that person's list in the friends list. If that name is not found, your function will return `-1`.

```
def friend_list_index(friends_list, name):
    for index in range(len(friends_list)):
        if friends_list[index][0] == name:
            return index
    return -1
```

3. Write a function that, when given `friends` will calculate how many of your best friends views you as one of their best friends (if you are their “mutual friend”) and return the value.

```
def really_my_friends(friends_list):
    num_friends = 0
    my_list = friends_list[0]
    others = friends_list[1:]
    for friends in others:
        if "me" in friends:
            num_friends = num_friends + 1
    return num_friends
```

4. Modify your function above to calculate the "mutual friend" value for any individual friend list. Hint: Using a function you made in 1-3 may simplify your solution.

```
def really_friends(friends_list, name):
    num_friends = 0
    index = friend_list_index(friends_list,name):
    name_list = friends_list[index]
    others = friends_list[0:len(friends_list] others.remove(name_list)
    for friends in others:
        if name in friends:
            num_friends = num_friends + 1
    return num_friends
```

5. Write a function to determine, given `friends`, if any of the lists consist entirely of people who "agreed" on their mutual "best friend status. That is, if you view them as a best friend, they also view you as a best friend.

```
def all_mutual_friends(friends_list):
    for person_list in friends_list:
        if really_friends(friends_list, person_list[0]) ==
                                                (len(person_list) - 1):
            return True
    return False
```