CSE 160 Section 3 Problems

Question 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]
list_2[0] = 99
list_1[4] = 99

print "List 1:", list_1
print "List 2:", list_2
print "List 3:", list_3
```

SOLUTION:

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

Question 2

Given the following code:

```
list_1 = ['A', 'B', 'C', 'D']
list_2 = ['J', 'L']
list_3 = ['U', 'V', 'W', 'X']
list_4 = ['M', 'N', 'O']
list_5 = ['P', 'Q', 'R', 'S', 'T', 'Y', 'Z']
list_6 = ['I', 'H', 'G', 'F', 'E']
list_7 = ['K']
```

Use only extend(), insert(), and reverse() – and possibly, a for loop -- to modify list_1 so that it contains all the letters of the alphabet in alphabetical order:

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

The command above should print a list that contains all the letters of the alphabet in alphabetical 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)
```

Question 3

Using the same initial lists as in Question 2, redo the problem, but this time you are allowed to use the sort() command.

ONE POSSIBLE SOLUTION:

```
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
```

Question 4

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.

Effective programmers perform these steps for every program they write, and they solve them very quickly. Ineffective programmers often make the mistake of mixing steps 6 and 7, either skipping step 6 or performing 6 and 7 simultaneously. As mentioned in lecture, the most effective programmers start coding later, and finish earlier. This is because they understood the problem and its solution before ever writing code.

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"]
```

It may be useful to have a "list of lists"
One called friends could be 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 a value in a list. Have your function return -1 if value provided is not found.
2. Write a function that given a name, will return the index of that person's list in the friends list. Have your function return -1 if name provided is not found.
3. How many of your best friends view you as one of their best friends?
4. Modify your function above to calculate the "mutual friend" value for any individual friend list.
5. Given the friends data you have, do any of the lists consist entirely of people who "agreed" on their mutual "best friend status?

POSSIBLE SOLUTIONS:

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.

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

3.

```
    def really_my_friends(friends_list):
        numFriends = 0
        my_list = friends_list[0]
        others = friends_list[1:5]
        for friends in others:
            if "me" in friends:
                numFriends = numFriends + 1
        return numFriends
```

4.

```
    def really_friends(friends_list, name):
        numFriends = 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:
                numFriends = numFriends + 1
        return numFriends
```

5.

```
    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
```