CSE 160 Section 4 Solutions

1. What values would be printed when you run the following lines of code?

```
list_1 = [1, 2, 3, 4, 5]
list_2 = list_1
list_3 = list_1[:]  # 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]

2. Given the following, modify `list_1` so it contains numbers 1 through 26 in increasing order:

```
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]
```

a) using only the following operations:

- `list accesses []`
- `extend()`
- `insert()`
- `reverse()`
- `for loop`

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)
```

b) using the same operations as in part (a), 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)
```

3. Create a function `dot_product(list1, list2)` which takes in two lists of integers and returns their dot product. Assume the lists are of equal length and contain only integer values. For example, the dot product of the lists `[1,2]` and `[3,4]` is: `1 * 3 + 2 * 4 = 11`.

```
def dot_product(list1, list2):
    dot_product = 0
    for num in range(len(list1)):
        cur_num_1 = list1[num]
        cur_num_2 = list2[num]
        dot_product = dot_product + (cur_num_1 * cur_num_2)
     return dot_product
```

4. You and your friends each have a list containing the name of the list owner followed by the names of their best friends. Your name is "me".

```
my_besties = ["me", "Emily", "John", "Ed", "Louise", "Tom"]
emily_besties = ["Emily", "me", "Rob", "Sue", "Alice", "Eric"]
john_besties = ["John", "Ed", "Rob", "Sue", "Eric", "Meg", "Emily"]
louise_besties = ["Louise", "me", "Alice", "Sue", "Emily", "Meg"]
```

`friends` (a list of lists) is constructed as follows:

```
friends = [my_besties, emily_besties, john_besties, louise_besties]
```

Your list is at position `friends[0]`.

Write a function for each of the following:

a. Given a name and a friend list, return the index of the first occurrence of the name in the list. If the name is not in the list, the function should return `-1`.

```
def find_friend(name, friend_list):
 ONE POSSIBLE SOLUTION:
   for index in range(len(friend_list)):
       if friend_list[index] == name:
           return index
   return -1

 ANOTHER POSSIBLE SOLUTION:
   if name in friend_list:
       return friend_list.index(name)
   return -1
```

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

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

c. Given `friends` (the list of lists described earlier), calculate and return how many of your best friends view you as one of their best friends.

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

d. Now, calculate the "mutual friend" value for any individual friend. Given a person's `name`, return how many of that person's best friends also view them as one of their best friends.

For example, `mutual_best_friends("John", friends)` would return `0`, because none of John's best friends include his name in their best friends list. However, `mutual_best_friends("me", friends)` would return 2 because Emily and Louise are in my best friend list and also include me in theirs.

You might find it helpful to use the helper function you wrote in part (b).

```
def mutual_best_friends(name, friends):
    num_friends = 0
    index = find_friend_list(name, friends) # Helper function from (b)
    name_list = friends[index]  # Person's list
    for friend in name_list:
        other_friend_index = find_friend_list(friend, friends)
        if other_friend_index != -1: # Make sure that the list exists
            other_friend_list = friends[other_friend_index]
            if name in other_friend_list and other_friend_index != other_friend_index:
                num_friends = num_friends + 1
    return num_friends
```