22C:16 Exam 2

April 1st, 9:30-10:20

This is an open notes exam. You have 50 minutes to complete it.

1. [**40 points**] For *each* of the following expressions, assume that
   `L = ["hello", "good day", "hi", "see you", "ciao", "au revoir"]`.

   (a) `L.extend(L[1].split())`
   What is the value of `L` after the execution of this expression?

   (b) `map(len, L[2::2])`
   What value does this expression evaluate to?

   (c) `L[3].split()[1][2]`
   What value does this expression evaluate to?

   (d) `map(range, map(len, L)[2:5])`
   What value does this expression evaluate to?

   (e) `del L[::3]`
   What is the value of `L` after the execution of this expression?

(f) `L in (L + [L])`
What value does this expression evaluate to?

(g) `L[2] = L[2] + " " + L[1]`
What is the value of `L` after the execution of this assignment?

(h) `L.insert(L.index("ciao"), "Tschuess")`
What is the value of `L` after the execution of this expression?

(i) `(L*3)[3:9]`
What value does this expression evaluate to?

(j) `L.insert(0, L.pop())`
What is the value of `L` after the execution of this expression?

2. **[35 points]** Consider the following program

```
def foo(x):
    global z
    z = 20
    return x + y - z

x = 12
y = 15
z = 11
y = foo(8)
print x + y - z
```

(a) What is the output of this program when it is executed?

(b) Now delete the `global z` statement from the function `foo`. What is the output of this modified program when it is executed?

(c) Now start with the original program and replace the function header by a new header:

`def foo(x=10, y=20):`

What is the output of this modified program when it is executed?

(d) Now start with the program as modified in part (c) and replace the line containing the function call (i.e., the second last line) by `y = foo(y, foo())` What is the output of this modified program when it is executed?

3. **[35 points]** There is a popular word puzzle in which you are given two words and you are asked to transform the first word into the second via a sequence of words, each obtained by substituting exactly one letter in the previous word. Of course, you have to make sure that all the intermediate words are valid words in the English language. For example, given the words "salt" and "bond," the following would be a solution to the puzzle: "salt" "sale" "bale" "bane" "band" "bond."

   Your task is to write a program that reads a sequence of words that is supposedly a solution to this puzzle and determines if the sequence is a *valid* solution or not. For example, if the input is

   ```
   salt sale bale bane band bond
   ```

   your program should respond with a message such as `Valid word sequence!`, whereas, if the input was

   ```
   text test bent bend bond
   ```

   your program should respond with a message `Not valid. Try again!` Note that in this example, going from `test` to `bent` required substituting two letters. Another reason a word sequence may be invalid is that it may contain one or more words that are not valid English words.

   (a) To complete this program, let us first write a function `notNeighbors` that takes two words `word1` and `word2` as parameters and returns `True` if `word2` cannot be obtained from `word1` by substituting exactly one letter by another and returns `False` otherwise. My plan is to count the number of corresponding letters in the two words that are not identical. If this count is not equal to 1 then the two words are not "neighbors" and the function should return `True`. The code given below is missing some lines and your task is to supply the missing code and complete the function.

```
def notNeighbors(word1, word2):
    if len(word1) != len(word2):
        return True

    count = 0
    # Three lines of missing code for scanning the two words
    # and comparing corresponding letters goes here.











    if count != 1:
        return True
    else:
        return False
```

(b) Here is a partial attempt at writing the main program. The program contains code (in the first two lines) for reading in the "dictionary" from the file "dictionary.txt" and for reading in the user input. You can assume that `loadDictionary` and `getInput` are appropriately defined functions and so you do not have to bother with defining these. After the first two lines of code are executed, the variable `dictionary` contains a list of valid English words and the variable `L` contains, as a list, the sequence of words input by the user. The program is missing some important code for scanning the sequence of words in `L` and determining if these words are valid English words and if they form a valid sequence. Your task is to supply this missing code.

```
# reads words from the file into a list called dictionary
dictionary = loadDictionary("dictionary.txt")

# reads input provided by the user
L = getInput()

# bool variable that tracks if the word sequence is valid
valid = True

# Missing lines of code should go here.














if valid:
    print "Valid word sequence!"
```

4. [**40 points**] You are given a list `L` or arbitrary length, but guaranteed to contain only numbers from the set $\{1, 2, \ldots, 10\}$, in some order. Your task is to write an efficient function to *sort* the list `L` taking advantage of the fact that `L` in known to contain only these numbers.

   Here I describe a simple algorithm called *counting sort* to do this:

   Step 1 Count the number of times 1 occurs in `L`, 2 occurs in `L`, etc. Store these counts in a size-10 list, called `countList`.

   Step 2 Produce a new list that first contains as many 1's as specified by `countList`, followed by as many 2's as specified, by `countList`, and so on. The function should return this newly constructed list.

   Notes: (i) Use the following function header: `def countingSort(L):`
   (ii) The function is fairly short: I wrote Step 1 in 3 lines of code, Step 2 in 3 lines of code, with one extra line to return the new list from the function.