CISC106 – Lab 05

This lab is due 07/14/2013 at 11:55 p.m. (submission via Sakai)

- **Please do all of the following problems in ONE file named lab5.py. This is an INDIVIDUAL assignment, please do all work accordingly.**
- **Use comments to separate your program for each problem. For questions where you should write your answers, envelop them as comments.**
- **For now on we will be using the Design recipe to write functions and the assertEqual function to run our test cases. You need to provide at least 3 test cases for each new function you write, when applicable. Please keep in mind that those elements are worth at least half of the question, so you may not want to forget them!**
- **Make sure the file cisc106.py (attached) is in the same directory of your lab5.py while running it. Also make sure you include it in your lab5.py program. You can do this by adding the following line at the very beginning of your file.**

  from cisc106 import *

- **The problems are worth 90 points + 10 points for attending the lab session.**
- **You do not need to make tests for functions that use random generation (unless they are already provided, on which case you should copy them into your file).**

Problem 1: (8 points)

Each new term in the Fibonacci sequence is generated by adding the previous two terms. By starting with 1 and 2, the first 10 terms will be:

1, 2, 3, 5, 8, 13, 21, 34, 55, 89...

By considering the terms in the Fibonacci sequence whose values do not exceed four million, write a function uses a repetition structure and returns the sum of the even-valued terms.

Problem 2: (8 points)

a) Write a function called **tuple_avg** that, using **repetition**, takes a list of non-empty tuples which elements are integers, and returns a list of the averages of the elements on each tuple.
b) Write a function called **tuple_avg_rec** that, using **recursion**, takes a list of non-empty tuples which elements are integers, and returns a list of the averages of the elements on each tuple.

Problem 3: (8 points)

a) Write a function called **del_dup_neighbor** that, using **repetition**, receives a list of numbers and returns a list where all adjacent elements that are equal to each other have been reduced to a single element, so [6, 7, 8, 8, 9, 2, 1, 2, 2, 3] returns [6, 7, 8, 9, 2, 1, 2, 3]. You may create a new list or modify the passed in list.
b) Write a function called **del_dup_neighbor_rec** that, using **recursion**, receives a list of numbers and returns a list where all adjacent elements that are equal to each other have been reduced

to a single element, so [6, 7, 8, 8, 9, 2, 1, 2, 2, 3] returns [6, 7, 8, 9, 2, 1, 2, 3]. You may create a new list or modify the passed in list.

Problem 4: (8 points)

Write a function called **merging_lists** that receives two lists, which are sorted in increasing order, and returns a merged list of all the elements also sorted in increasing order. You may modify the passed in lists. Your solution should work in "linear time", which means that you are only allowed to make a single pass of both lists.

Problem 5: (8 points)

Write a function called **counting_strings**, which receives a list of strings, return the count of the number of strings where the string length is at least 2 and the first and last characters of the string are the same.

Problem 6: (8 points)

Write a function called **swap_two** that takes two strings *x* and *y* and returns a single string with x and y separated by a space, except swap the first 2 characters of each string.

e.g.:

'fix', 'pod' -> 'pox fid'

'dog', 'dinner' -> 'dig donner'

Problem 7: (8 points)

Write a function called **verb_ending**, which receives a string and, if the length of the string is at least 3, add 'ing' to its end. Unless it already ends in 'ing', in which case add 'ly' instead. If the string length is less than 3, leave it unchanged. Return the resulting string.

Problem 8: (10 points)

A palindromic word is one that reads the same backwards as forwards. Hence the words hello and peel are not palindromes, but the words peep, deed and dad are palindromes.

a) Create a method called **reverse** which takes a string argument. Your method should return the reverse of the argument as a string. For example, if the argument is **Foobar** then your method should return **rabooF**.

b) Create a second method called **isPalindrome** which takes a string argument. This method should return True if the argument is a palindrome and False otherwise.

Problem 9: (8 points)

Write a function called **be_happy**, which receives a string. The function needs to find the first appearance of the substring 'not' and 'sad'. If the 'sad' follows the 'not', replace the whole 'not'...'sad' substring with 'happy'. The function should return the resulting string.

So 'Actually, he was not that sad.'

yields: 'Actually, he was happy. '

Problem 10: (8 points)

Consider dividing a string into two halves. If the length is even, the front and back halves are the same length. If the length is odd, we'll say that the extra char goes in the front half.

e.g. 'room', the front half is 'ro', the back half 'om'.

Write a function called **mixing_up**, which receives 2 strings, str1 and str2, and returns a string of the form

str1-front + str2-front + str1-back + str2-back

Your function should work for the following test cases:

assertEqual(mixing_up(“house”, “truck”), “houtruseck”)

assertEqual(mixing_up(“computer”, “sciences”), “compscieuternces”)

Problem 11: (8 points)

For this question of the lab you need to submit the function from your project that ranks the most probably 5 letters that can appear after the letter the user typed. Do not forget to provide the design recipe of your function and also to provide comments to explain your decisions.