CS 1110, LAB 8: RECURSION EXERCISES

http://www.cs.cornell.edu/courses/cs1110/2017fa/labs/lab8/

First Name: ____________________ **Last Name**: ____________________ **NetID**: ______________

This lab is designed to help you with recursive functions. All of the functions in this lab will either be recursive functions on sequences (e.g. strings or lists), or recursive functions on integers, just as we saw in class. They are also the right length for questions on Prelim 2.

This is a fairly important lab, but it is longer than normal. There are four functions that you must implement, and then a long list of optional ones. We will have a shorter lab next week to make up for it. It will help a lot with Prelim 2 if you not only finish the required parts of the lab, but also finish the optional functions as well.

1. Lab Files

For today's lab you will need the following additional files:

- `lab08.py` (a module with functions to implement)
- `test08.py` (a completed unit test script to aid you)

Once again you should create a *new* directory on your hard drive and download all of the files into that directory. Alternatively, you can get all of the files bundled in a single ZIP file called lab08.zip from the Labs section of the course web page.

Getting Credit for the Lab. By now you should have a good idea of which version of the lab (PDF or online) you are more comfortable with. This lab involves quite a bit of coding, so you may have trouble finishing it during lab time. For that reason, you might want to complete it using the online system. Otherwise, you get credit by showing the completed file `lab08.py` to your instructor.

As with all previous labs, if you do not finish during the lab, you have until the **beginning of lab next week to finish it**. Over the next week, you may either (1) complete the lab online, (2) show your lab to a consultant during consulting hours, or (3) show your lab to an instructor at the *beginning* of the next lab session.

2. Recursive Functions

Most of the recursive functions in this lab are designed to be solved with the divide-and-conquer process. When you write your functions, you should break the solution down as follows:

Handle small data directly. The base case involves the "smallest" parameter values, for which the result can be given easily, without the need for recursive calls. For a function that works on a sequence (e.g. either a string or a list), the base case is usually a sequence of length 0 (or both length 0 and 1). If the data is a natural number, then small values are `0` and `1`.

Course authors: D. Gries, L. Lee, S. Marschner, W. White

Break the data into two parts. For recursion to work, you must make the data "smaller." The recursive calls make the data smaller and smaller, until it gets small enough to be handled directly. For sequences, you make the data smaller via slicing. For natural numbers, you make the value `n` smaller by breaking it into two numbers `a` and `b` such that either `a+b` or `a*b` is `n`.

When you break the data into the two parts, you will need to call the function recursively on each half. As we saw in class, sometimes one of the halves is so small that we can compute the answer directly and do not need to use recursion on that half.

Combine the answer. Once you have the recursive answer for each half, you need to combine the answers together to get the answer on the entire data. Many times this is as simple as adding the answers together. However, as we saw in class, some functions require more complicated means of combination. Read the specification carefully to figure out what to do.

3. Lab Activities

In this lab, you are to implement the first four functions from the module `lab08.py`, specified below. All implementations must be recursive, using divide-and-conquer.

```
def numberof(thelist,v):
    """Returns: number of times v occurs in thelist.

    Precondition: thelist is a list of ints, v is an int"""

def replace(thelist,a,b):
    """Returns: a COPY of thelist but with all occurrences of a replaced by b.

    Example: replace([1,2,3,1], 1, 4) = [4,2,3,4].

    Precondition: thelist is a list of ints, a and b are ints"""

def remove_dups(thelist):
    """Returns: a COPY of thelist with adjacent duplicates removed.

    Example: for thelist = [1,2,2,3,3,3,4,5,1,1,1], the answer is [1,2,3,4,1]

    Precondition: thelist is a list of ints"""

def oddsevens(thelist):
    """Returns: copy of the list with odds at front, evens in the back.

    Odd numbers are in the same order as thelist. Evens are reversed.
    Example: oddsevens([3,4,5,6]) returns [3,5,6,4]
             oddsevens([2,3,4,5,6]) returns [3,5,6,4,2]

    Precondition: thelist is a list of ints (may be empty)"""
```

Even though we only ask you to work on the first four functions in module `lab08.py` in this lab, you will get greater fluency in recursion if you do them all. Of the course of the next month, take some time to work on the remaining functions and test them. You should particularly try some of the integer recursive functions that appear later in `lab08.py`.