CS 1110, LAB 6: RECURSION

Name: _____ **Net-ID**: _____

There is an online version of these instructions at

http://www.cs.cornell.edu/courses/cs1110/2012fa/labs/lab6

You may wish to use that version of the instructions.

This lab gives you experience with writing 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. This is a fairly important lab. If you do not finish today, please be sure to finish it soon after you get back from Fall Break.

Requirements For This Lab. We have created a few files for this lab. You should create a new directory on your hard drive and download the following modules into that directory:

- `recursive.py` (http://www.cs.cornell.edu/courses/cs1110/2012fa/labs/lab6/recursive.py)
- `testlab.py` (http://www.cs.cornell.edu/courses/cs1110/2012fa/labs/lab6/testlab.py)
- `cunittest.py` (http://www.cs.cornell.edu/courses/cs1110/2012fa/labs/lab6/cunittest.py)

The last two files are unit tests that we have provided for you. You do not need to write your own unit tests in this lab. All you need to worry about are the functions in `recursive.py`. You will note that all of the functions in this module are stubs, and are not fully implemented. Your entire lab will consist of modifying this one file.

To successfully complete this lab, you should **implememnt the first four functions in the file `recursive.py`**. When you have done this, show your file to a TA. As always, you should try to finish the lab during your section. However, this is a longer lab than the past two (now that you are done with all that work), and so it may not be possible. If you do not finish during section, you have **two weeks to finish, as there is no lab section the week of fall break**. Show your work to your instructor at the beginning of your next lab. As always, remember that labs are graded on effort, not correctness.

---

1. Recursive Methods

Remember that creating and understanding a recursive function involves four important steps:

A precise specification of the function. Without this, you cannot write the function at all.

Handling the base case properly. 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). However, it could be something else, depending on the problem.

For a function that works on the natural numbers `0, 1, ...`, the base case is usually `0` (or both `0` and `1`).

In the module `recursive.py` there is one function that has very different base cases, but we explicitly spell this one out for you.

Handling the recursive case properly. Solve the original problem in terms of the same problem but on a "smaller" value. For example, if the function involves a sequence (e.g. a string or a list) `s`, the solution should be describable in terms of the same problem on some smaller slice of `s`.

In writing/understanding a recursive call, understand it in terms of the *specification* of the function called. Do not try to trace the execution in your head.

Making progress toward termination. In keeping with the last point, the arguments of a recursive call must be in some measure smaller than the parameters of the method; this is what ensures termination. Each recursive call has "smaller arguments", so that after some point, the base case will be reached. For example, if the argument is a sequence (e.g. a string or a list), each call should be on a smaller slice of the sequence.

---

2. Lab Activities

In this lab, you are to implement the first four functions from the module `recursive.py`. These are the ones specified below. All implementations must be recursive (practicing recursion is the point of this lab).

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

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

def number_not(thelist):
    """Returns:  number of elements in thelist that are NOT v.

    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"""
```

Even though we only ask you to work on the first four functions in module `recursive.py` in this lab, you will get greater fluency in recursion if you do them all. So, during the week, every one in a while write one of the remaining functions and test it. You should particularly try some of the integer recursive functions that appear later in `recursive.py`.