CS 1110, LAB 06: RECURSION

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

There is an online version of these instructions at

http://www.cs.cornell.edu/courses/cs1110/2013sp/labs/lab06/lab06.pdf

You may wish to use that version of the instructions.

This lab gives you some practice 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.

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/cs1110/2013sp/labs/lab06/recursive.py)
- `testlab.py` (http://www.cs.cornell.edu/courses/cs1110/2013sp/labs/lab06/testlab.py)
- `cunittest2.py` (http://www.cs.cornell.edu/courses/cs1110/2013sp/labs/lab06/cunittest2.py)

The last two files are unit tests that we have provided for you. You do not need to write your own 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; your job is to implement them so that they fulfill their specifications. So you only need to modify `recursive.py`, and you can test the functions as you go by running the test suite using `python testlab.py`

To successfully complete this lab, implement **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 lab is a bit on the long side, and so it may not be possible. If you do not finish during section, you have **two weeks to finish, as there is no new lab scheduled for the week of the prelim**. 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. For this lab we have already provided these specifications; be careful to read them carefully and interpret them literally.

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 often 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 often 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. Even though we have seen various functions from the Python library that can compute the required results for you, in this lab the point is practicing the use of recursion and elementary list operations (slicing and concatenation) to solve these problems yourself. So all you implementations must be recursive.

```
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 once 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`.