CS 1133, LAB 5: LISTS AND CONTROL STRUCTURES

http://www.cs.cornell.edu/courses/cs1133/2017fa/labs/lab5/

First Name: ____________ **Last Name**: ____________ **NetID**: ________

Just when you had become an expert at string slicing, you discovered another sliceable data type: lists. However, lists are different from strings in that they are *mutable*. Not only can we slice a list, but we can also change its contents. The purpose of the lab is to introduce you to these new features, and demonstrate just how powerful the list type can be.

In addition to lists, this lab gives you some experience working with control structures: both if-statements and for-loops. While you did not use these in the first assignment, they are an important part of the second assignment. Once you complete this lab, you will be ready to work on the second assignment.

Lab Materials. There is a file to download for this lab. You can find it from the course web page:

http://www.cs.cornell.edu/courses/cs1133/2017fa/labs

The file is named `lab05.py`. This file contains specifications for the functions in the second half of this lab. This is the only file that you need; you do not need to create a unit test file.

When you are done, show both this handout, and `lab05.py` to your instructor. You instructor will then record the lab as completed. You do not need to submit the paper with your answers, and you do not need to submit the computer file.

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**. You should always do your best to finish during lab hours. Remember that labs are graded on effort, not correctness.

1. List Expressions and Commands

This part of the lab will take place in the Python interactive prompt, much like the first labs. You do not need to create a module. First, execute the following assignment statement:

```
lablist = ['H','e','l','l','o',' ','W','o','r','l','d','!']
```

Like a string, this is a list of individual characters. Unlike a string, however, the contents of this list can be changed.

On the next page enter the statements **in the order they are presented**. Some of the commands below are actually expressions, and so Python will immediately display the value. In the case of actual statements, we include a print statement showing the new contents of the list. Each case, describe what you see and *explain the result.*

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

Commands	Result/Explanation

`lablist.remove('o)`
`print lablist`	

`lablist.remove('x)`	

`pos = lablist.index('o)`
`print pos`	

`pos = lablist.index('B)`	

`lablist[0] = 'J`
`print lablist`	

`lablist.insert(5,'o)`
`print lablist`	

`s = lablist[:]`
`print s`	

`s[0] = 'C`
`print s`
`print lablist`	

`a = '-'.join(s)`
`print a`	

`a = ''.join(s)`
`print a`	

`t = list(a)`
`print t`	

2. Pig Latin

Pig Latin is childish encoding of English that adheres to the following rules:

1. The vowels are 'a', 'e', 'i', 'o', 'u', as well as any 'y' that is *not* the first letter of a word. All other letters are consonants.

For example, 'yearly' has three vowels ('e', 'a', and the last 'y') and three consonants (the first 'y', 'r', and 'l').

2. If the English word begins with a vowel, append 'hay' to the end of the word to get the Pig Latin equivalent. For example, 'ask' becomes 'askhay', 'use' becomes 'usehay'.

3. If the English word starts with 'q', assume it is followed by 'u'; move 'qu' to the end of the word, and append 'ay'. Hence 'quiet' becomes 'ietquay', 'quay' becomes 'ayquay'.

4. If the English word begins with a consonant, move all the consonants up to the first vowel (if any) to the end and add 'ay'. For example, 'tomato' becomes 'omatotay', 'school' becomes 'oolschay', 'you' becomes 'ouyay', and 'ssssh' becomes 'sssshay'.

Your goal is to write a function `pigify` which has the following specification.

```
def pigify(w):
    """Returns: copy of w converted to Pig Latin

    See the lab instructions for the complete rules on Pig Latin.

    Parameter w: the word to change to Pig Latin
    Precondition: w is a nonempty string with only lowercase letters"""
```

To help you with this function, we have provided the function `first_vowel()` already implemented in `lab05.py`. This function has the following specification.

```
def first_vowel(w):
    """Returns: position of the first vowel; -1 if no vowels.

    Parameter w: the word to check
    Precondition: w is a nonempty string with only lowercase letters"""
```

You do not need to modify or implement the function `first_vowel`; just use it.

When you are done with your implementation of `pigify`, you will want to test your answer. We do not want you to create a test script. Just write down four test cases in the table below, and confirm that they work in the Python interactive shell.

Input	**Expected Output**

3. List Functions

Below are two function specifications. You are to implement them implement them in the file `lab05.py`. You will probably want to test them out before you submit them. However, we are not asking for any test cases this time. We just want the implementation.

Lists do *not* have a `find()` method like strings do. They only have `index()`. To check if an element is in a list, use the `in` operator (e.g. `x in thelist`). Lists also have the following methods.

Method	Result When Called
l.index(c)	**Returns**: the first position of c in list l; error if not there
l.count(c)	**Returns**: the number of times that c appears in the list l.
l.append(c)	Add the value c to the end of the list. This method alters the list; it does not make a new list.
l.sort()	Rearrange the elements of the list l in ascending order. This method alters the list; it does not make a new list.

None of the functions below should ever alter `thelist`. If you need to call a method that might alter the contents of thelist, you should make a copy of it first.

```
def lesser_than(thelist,value)
    """
    Returns: number of elements in thelist strictly less than value
    without altering thelist.

    Example: lesser_than([5, 9, 1, 7], 6) evaluates to 2

    Precondition: thelist is a list of ints; value is an int
    """


def unique(thelist):
    """
    Returns: The number of unique elements in the list.

    Example: unique([5, 9, 5, 7]) evaluates to 3
    Example: unique([5, 5, 1, 'a, 5, 'a]) evaluates to 3
    Precondition: thelist is a list.
    """
```