CS1110 Lab 6: Practice for A5 (Mar 22-23, 2016)

First Name: _____ **Last Name**: _____ **NetID**: _____

The lab assignments are very important. Remember this: *The lab problems feed into the assignments and the assignments define what the exams are all about.*

Start *before* your lab meets.

We recommend spending an hour or two on the lab *before* coming to your section, so you can use your in-person time to ask questions most efficiently.

Also, this strategy of starting beforehand increases your chances of checking in at your lab section, which will probably take less time than waiting in line at consulting hours!

Getting credit

Complete all required blank boxes and lines on this handout. When you are finished, show your written answers to one of the CS 1110 lab staff in your section on March 22-23 or in any consulting hours up to and including April 4 (earlier days have shorter lines). The staff member will ask you a few questions to make sure you understand the material, and then swipe your Cornell ID card or directly make a notation in CMS to record your success. This physical piece of paper is yours to keep.

Getting set up

From the Lab webpage, download and unzip `Lab_6.zip` into a folder named (for example) `Lab_6`. In the command shell, navigate the file system so that this folder is THE CURRENT WORKING DIRECTORY. Review the slides from the March 17 lecture.

1 Lists: Basic Ideas

These exercises feature lists of numbers. (Lists of strings are coming soon.)

1.1 Typical List Errors

Remember, it is square brackets with comma separators for setting up short lists, e.g., `[10,20,30]`. For each of these examples, how does Python respond?

```
>>> x = [10 20 30]
```

```
>>> x = [10, 20, 30]
>>> x[3] = 40
```

```
>>> x = [10 ; 20]
```

1.2 Lists are a Type

What does Python say after this:

```
>>> x = [1,2,3]
>>> type(x)
```

What does Python say after this:

```
>>> x = [10]
>>> y = 10
>>> type(x)
>>> type(y)
```

1.3 The Void Methods `append` and `extend`

Do these problems without the computer. Then check your answers by using interactive Python.
What does Python say after this:

```
>>> x = [10,20,30]
>>> x.append(99)
>>> x
```

What does Python say after this:

```
>>> x = [10,20,30]
>>> y = x.append(99)
>>> type(y)
```

ASK: Make sure the students understand why the type is NoneType

What does Python say after this:

```
>>> x = [10,20,30]
>>> y = [40,50]
>>> x.extend(y)
>>> x
```

What does Python say after this:

```
>>> x = [10,20,30]
>>> y = [40]
>>> x.extend(y)
>>> x
```

What does Python say after this:

```
>>> x = [10,20,30]
>>> y = 40
>>> x.extend(y)
```

should get "TypeError: 'int' object is not iterable". TELL THE STUDENT: lists are "iterable" objects, and extend is expecting a list as argument. Since it got an int instead, an error was generated.

1.4 The Void Method `insert`

Do these problems without the computer. Then check your answers by using interactive Python.

What does Python say after this:

```
>>> x = [1,2,3,4]
>>> i = 3
>>> y = 99
>>> x.insert(i,y)
>>> x
```

What does Python say after this:

```
>>> x = [1,2,3,4]
>>> i = 3
>>> y = [10,20]
>>> x.insert(i,y)
>>> x
```

(A list can have a list as an entry.) THAT IS A HINT :)

1.5 The Void Method `sort`

Do these problems without the computer. Then check your answers by using interactive Python.

What does Python say after this:

```
>>> x = [4,1,3,2]
>>> x.sort()
>>> x
```

What does Python say after this:

```
>>> x = [4,1,3,2]
>>> x = x.sort()
>>> print x
```

If the students see nothing at all, make sure they typed the PRINT, because that's what's needed to reveal the None

What does Python say after this:

```
>>> x = [4,1,3,2]
>>> x = x.sort(reverse=True)
>>> x
```

It would be a good idea to also ask the students, what do they think happens to x if there hadn't been an assignment to it. (Answer: list would be sorted in, well, reverse order.)

1.6 The Fruitful Methods `count` and `pop`

Do these problems without the computer. Then check your answers by using interactive Python.

What does Python say after this:

```
>>> x = [4,1,3,2,4]
>>> m = x.count(4)
>>> m
```

What does Python say after this:

```
>>> x = [10,20,30,40]
>>> m = x.pop(1)
>>> print m,x
```

What does Python say after this:

```
>>> x = [10,20,30]
>>> a = x.pop(0)
>>> b = x.pop(0)
>>> c = x.pop(0)
>>> print a,b,c,x
```

What does Python say after this:

```
>>> x = [10,20,30]
>>> a = x.pop(len(x)-1)
>>> b = x.pop(len(x)-1)
>>> c = x.pop(len(x)-1)
>>> print a,b,c,x
```

Note that x becomes the empty list, so do make sure the students know that [] is a list, just one with no elements

1.7 Slicing and Subscripts

Do these problems without the computer. Then check your answers by using interactive Python.

What does Python say after this:

```
>>> x = [3,2,0,1]
>>> y = x[x[x[1]]]
>>> y
```

x[1] is 2; x[2] is 0; x[0] = 3. So x[x[x[1]]] is x[x[2]] is x[0] is 3.

What does Python say after this:

```
>>> x = [3,2,0,1]
>>> y = x[1:]
>>> y.append(x[0])
>>> y
```

2 Dice Rolls

The given module `DiceRolls.py` contains this function (also discussed in the March 17 lecture):

```
def randiList(L,R,n):
    """ Returns a length-n list of
    random integers from interval [L,R]

    PreC: L,R,n ints with L<=R and n>=1
    """
    x = []
    for k in range(n):
        r = randi(L,R)
        x.append(r)
    return x
```

The same module has an application script with parts that you have to complete. Fill in the boxes and check your answers by running `DiceRolls.py`. To force issues, you are not allowed to use `count` or `sum`.

Better name for TwoThrows would have been ThreeThrows, but oh well.

```
if __name__ == '__main__':
    """ Roll 3 die many times and record the outcomes
    using lists. Then pose some questions of the stored data.
    """
    N = 100000
    D1 = randiList(1,6,N)
    D2 = randiList(1,6,N)
    D3 = randiList(1,6,N)
    # Set up a list  D where D[k] is the sum of the three die values
    # on roll k.
    D = []
    for k in range(N):
        ThreeThrows = D1[k] + D2[k] + D3[k]
        D.append(ThreeThrows)

    # Assign to a variable ave0 the average value in D.
```

```
print 'Average when 3 die are rolled = %6.4f' % ave0

# Assign to a variable Prob1 an estimate of the probability
# that the sum of the three die values is 7.
```

Solution:

```
s = 0
for k in range(N):  # OK also to do "range(len(D))"
    s = s+D[k]
ave0 = s/float(N)
```

can also do:

```
s = 0
for v in D:
    s = s+v
ave0 = s/float(N)
```

```
print  'Prob1 = %6.4f' % Prob1

# Assign to a variable Prob2 an estimate of the probability
# that one die value is at least as big as the sum of the other
# two dice values
```

Solution: Maybe there's something clever that can be done with maxes and mins, as well.

```
m = 0
for k in range(N):
    B1 = (D1[k] >= D2[k]+D3[k])
    B2 = (D2[k] >= D1[k]+D3[k])
    B3 = (D3[k] >= D1[k]+D2[k])
    if B1 or B2 or B3:
        m+=1
Prob2 = m/float(N)
```

```
print  'Prob2 = %6.4f' % Prob2
```

3 Functions and Lists

Do these problems without the computer. Then check your answer by running `ShowAdd1.py` and `ShowAdd2.py`.

3.1 Add1

If

```
def Add1(x,y):
    """
    PreC: x and y are lists of numbers with len(x)==len(y)
    """
    z = []
    for k in range(len(x)):
        s = x[k]+y[k]
        z.append(s)
    return z
```

then what is the output if we run

```
# Example 1
a = [1,2,3];  b = [10,20,30]
c = Add1(a,b)
print a, b, c
# Example 2
a = [1,2,3]; b = [10,20,30]
b = Add1(a,b)
print a, b
```

3.2 Add2

If

```
def Add2(x,y):
    """PreC: x and y are lists of numbers with len(x)==len(y)"""
    for k in range(len(x)):
       x[k] = x[k]+y[k]
    return x
```

then what is the output if we run

```
   # Example 1
    a = [1,2,3]; b = [10,20,30]; c = Add2(a,b)
    print a, b, c
    # Example 2
    a = [1,2,3]; b = [10,20,30]
    a = Add2(a,b)
    print a, b
```

Solution: POINT OUT: THEY SHOULD NOTICE! The argument list was changed! THis is a new concept for them, most likely.

4 Reverse

This script creates a list `y` that is the same as `x` except that the order of the entries is reversed:

```
x = [10,20,30,40]
y = []
y.append(x[3])
y.append(x[2])
y.append(x[1])
y.append(x[0])
```

Give an implementation of the following function so that it performs as specified:

```
def Reverse(x):
    """ Returns a list that is the same as x except that the order of its
    entries is reversed.

    PreC: x is a list of numbers
    """
```

Solution: If they use a "reverse" function or method, that's legal, BUT: (a) make sure they don't alter the input x; (b) make sure they return; and (c) do ask if they know how they would do this with a loop.