Last Name: ______________ First: ______________ Netid: ________ Section ________

CS 1110 Final, December 8th, 2016

This 150-minute exam has 8 questions worth a total of 100 points. Scan the whole test before starting. Budget your time wisely. Use the back of the pages if you need more space. You may tear the pages apart; we have a stapler at the front of the room.

It is a violation of the Academic Integrity Code to look at any exam other than your own, look at any reference material, or otherwise give or receive unauthorized help.

You will be expected to write Python code on this exam. We recommend that you draw vertical lines to make your indentation clear, as follows:

```
def foo():
|   if something:
|   |   do something
|   |   do more things
|   do something last
```

Unless you are explicitly directed otherwise, you may use anything you have learned in this course.

Question	Points	Score
1	2	
2	10	
3	16	
4	14	
5	14	
6	14	
7	14	
8	16	
Total:	100	

The Important First Question:

1. [2 points] Write your last name, first name, netid, and *lab section* at the top of each page.

Throughout this exam, there are several questions on sequences (strings and lists). All sequences support slicing. In addition, you may find the following sequence expressions below useful (though not all of them are necessary).

Expression	**Description**
`len(s)`	**Returns**: number of elements in sequence `s`; it can be 0.
`x in s`	**Returns**: `True` if `x` is an element of sequence `s`; `False` otherwise.
`s.index(x)`	**Returns**: index of the FIRST occurrence of `x` in `s`. Raises a `ValueError` if `x` is not found.
`x.append(a)`	(**Lists Only**) Adds `a` to the end of list `x`, increasing length by 1.
`x.remove(a)`	(**Lists Only**) Removes first occurrence of `a` in `x`, decreasing length by 1.
`x.extend(y)`	(**Lists Only**) Appends each element in `t` to the end of list `x`, in order.
`x.insert(i,y)`	(**Lists Only**) Inserts `y` at position `i` in list `x`. Elements after position `i` are shifted to the right.

2. [10 points total] **Short Answer**

(a) [3 points] What is a parameter? What is an argument? How are they related?

(b) [3 points] What is a watch? What is a trace? What purpose do they serve?

(c) [4 points] In the code below, you are given variables `x` and `n`. You can assume that they have already been initialized so that the initial assertion is true. Add code so that the later assertions are true (you may **not** reassign the variable `n`).

```
# Assertion:  x is the sum of all odd numbers in 1..n
n = n + 1



# Assertion:  x is the sum of all odd numbers in 1..n
n = n - 1



# Assertion:  x is the sum of all odd numbers in 1..n
```

Last Name: ______________ First: ______________ Netid: ________ Section ________

3. [16 points] **Classes and Subclasses**

The very first step in Assignment 7 was to make a welcome message with the class `GLabel`. As you may recall, it has the following following primary attributes.

Attribute	Invariant	Description
`x`	`float`	x-coordinate of the label center.
`y`	`float`	y-coordinate of the label center.
`text`	`str`	The text to display.
`linecolor`	`colormodel.RGB`	The color for the text.
`fillcolor`	`colormodel.RGB`	The background color.

There are other attributes, such as `width` and `height`, but they are not important for this question. In particular, `width` and `height` are computed automatically from the `text`.

`GLabel` is a very useful class and is often the subclass of more interesting classes. In this problem, you are to make a `GButton`, which provides support for a clickable button. `GLabel` already has a method `contains` to help us determine if it was touched, so the `GButton` class simply adds a new attribute indicating whether or not it is pressed.

To make the button a little more interesting, we want it to change color when we press it. The simplest way to do this is to swap the `linecolor` and `fillcolor` of a pressed button. This is shown to the right.

Putting this all together, the specification of this class is as follows.

```
class GButton(GLabel):
    """Instance is a clickable button.

    (MUTABLE) ATTRIBUTES:
        _pressed [bool]:  whether the button is pressed
        _visible [bool]:  whether the button is visible

    In addition, there is one more invariant.  When _pressed is true, the
    inherited attributes linecolor and fillcolor are swapped.  They go back
    to normal when _pressed is false"""
```

Implement this class on the next page. We have provided you with the specifications for the methods `__init__` and `draw`. You should fill in the missing details to meet these specifications. In addition, you must add the getters and setters (where appropriate) for the new attributes. Remember that setters must have preconditions to enforce the attribute invariants.

Hint: The attributes inherited from `GLabel` work like they do in Assignment 7, and have invisible setters and getters. Therefore, you never have to enforce the invariants for these attributes. You only need to worry about your two new attributes: `_pressed` and `_visible`.

In addition, you have the module `colormodel` at your disposal. If you cannot remember all of the attributes of an `RGB` object, the module provides constants for colors such as `RED`, `YELLOW`, `GREEN`, and `BLACK`. You may use these as you wish.

```
from game2d import *
import colormodel     # Note the difference in the imports

class GButton(GLabel):
    """See the specification on previous page."""
    # PUT THE GETTERS AND SETTERS HERE
















    def __init__(                                      ):   # FILL IN
        """Initializes a button with the given text centered at (x,y).

        There are four parameters: x, y, text, and shown. The first three are
        the same as for GLabel. The last parameter shown indicates if the button
        is visible. It is OPTIONAL and defaults to True.
        All buttons start with a fillcolor of black and a linecolor of green.
        New buttons are not pressed.

        Preconditions: x and y are floats, text is a str, shown is a bool."""
```

```
# Class GButton (CONTINUED).
    def draw(                                          ):           # FILL IN
        """Draws this button to the provided view.

        If the button is not visible, it does not draw anything. Otherwise, it
        draws exactly the same way that its superclass does.

        Precondition: view is a GView object"""
```

4. [14 points total] **Asserts and Error Handling**

   (a) [8 points] As you saw in Assignment 6, it is often handy to have a boolean function to enforce preconditions of other functions. The function specified below is one such function. Implement this function, assuming nothing about the argument `value`.

```
def is_sorted_int_list(value):
    """Returns: True if value is a nonempty, sorted list of only ints.

    It returns False otherwise. It may not modify the contents of value.

    Precondition: value can be anything"""
```

   (b) [6 points] Suppose you are given the following function definitions.

```
def first(n):
    x = 0
    try:
        x = second(n)
    except StandardError:
        x = x+1
    return x
```

```
def second(n):
    y = 2
    try:
        y = third(n)
    except ValueError:
        y = y+5
    return y
```

```
def third(n):
    if n == 0:
        raise ValueError()
    elif n == 1:
        raise TypeError()
    return n+10
```

Assume that `ValueError` and `TypeError` are subclasses of `StandardError`, but neither is a subclass of the other. For each function call below, give the answer returned. If there is no value (e.g. program crashes), tell us that. To get full credit, you must (1) identify what caused the error, if there was one, and (2) identify the line number where it recovered from the error, if it recovers at all.

i. first(0)

ii. first(1)

iii. first(2)

5. [14 points total] **Loop Invariants**

On the next page are two variations of the fixed partition algorithm from the last lab. The algorithm organizes a list into negative and non-negative numbers. The version on the left has been completed for you. The second algorithm on the right is similar to the first except that it has a different precondtion, postcondition and loop invariant. It is also missing the code for initialization, the loop condition, and the body of the loop. Complete this code according to the invariant. **Solutions that violate the invariant will not receive credit.**

(a) [2 points] Draw the horizontal notation representation for the loop invariant on the left.

(b) [2 points] Draw the horizontal notation representation for the loop invariant on the right.

Last Name: ______________________ First: ______________________ Netid: _________ Section _________

(c) [10 points] Add the missing code to the function on the right. Like the function on the left, you may use the helper function `swap(b,n,m)` to swap two positions in the list. Pay close attention to the precondition and the postcondition, as they are also different.

```
def posneg1(b):
    """Return: Boundary i of partition

    Rearranges the list into negatives
    first, then non-negatives.
    The value i indicates the boundary
    of these two groups of values.

    Pre: b is a list of numbers."""
    # pre: b[0..]  ???
    # Make invariant true at start

    i = 0

    j = len(b)

    # inv: b[0..i-1] < 0, b[i..j-1] ???,
    # and b[j..]  >= 0

    while i < j:
        if b[i] < 0:
            i = i+1

        else:
            swap(b,i,j-1)

            j = j-1

    # post: b[0..i-1] < 0 and b[i..] >= 0
    return i
```

```
def posneg2(b,h,k):
    """Return: Boundary i of partition

    Rearranges the list into negatives
    first, then non-negatives.
    The value i indicates the boundary
    of these two groups of values.

    Pre: h,k are indices of list b."""
    # pre: b[h..k] ???
    # Make invariant true at start

    # inv: b[h..j] ???, b[j+1..i] < 0,
    # and b[i+1..k] >= 0

    while _______________:

    # post: b[h..i] < 0 and b[i+1..k] >= 0
    return i
```

6. [14 points] **2-Dimensional Lists**

A matrix is a rectangular table of numbers, as shown below. The *dimension* of a matrix is $n \times m$ where n is the number of rows and m the number of columns. The matrix on the left has dimension 3x4, while the one in the middle has dimension 4x3.

In mathematics, we often want to reduce the dimension of a matrix by removing a single row and a single column. This is shown below, where we remove the third row and second column.

$$\begin{bmatrix} 1 & 5 & 0 & 1 \\ 2 & 3 & -4 & 1 \\ -1 & 0 & 2 & 1 \end{bmatrix}$$

A 3x4 matrix

$$\begin{bmatrix} 1 & 5 & 0 \\ 2 & 3 & -4 \\ 1 & 0 & 2 \\ 1 & 1 & -1 \end{bmatrix}$$

A 4x3 matrix

$$\begin{bmatrix} 1 & \cancel{5} & 0 \\ 2 & \cancel{3} & -4 \\ \cancel{1} & \cancel{0} & \cancel{2} \\ 1 & \cancel{1} & -1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 2 & -4 \\ 1 & -1 \end{bmatrix}$$

Reduction to a 3x2 matrix

Implement the function below. Remember that tables are implemented in Python using row-major order. In addition, the row and column indices start at 0.

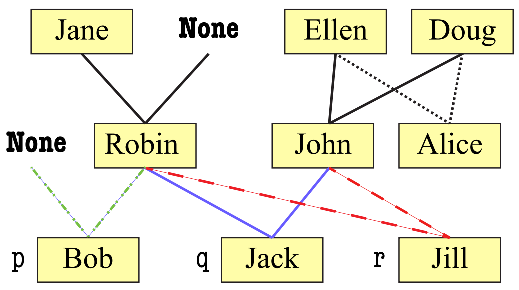
```
def reduce(matrix,row,col):
    """Returns: a copy of the matrix, missing the given row and column.

    Preconditions: matrix is a table of numbers, row is an index (int) for a
    row, while col is an index (int) for a column"""
```

7. [14 points] **Recursion**

You may recall the class `Person` from the second prelim, whose specification is on the next page. For this problem, you do not need to know how to use the constructor for `Person`. You only need to use the three attributes mentioned in the specification.

The family trees we saw before looked really simple because each set of parents only had one child. However, this is not required by the class specification. If we wanted to show a family tree with brothers, sisters, aunts, and uncles, it might look something like the diagram below.

```
class Person(object):
    """Instance is a person/family tree
    INSTANCE ATTRIBUTES:
       name: First name [nonempty str]
       mom: Mom's side [Person or None]
       dad: Dad's side [Person or None]
    """
    ...
```



We say that two people are *related* if they have a common person in their family tree (including themselves). A recursive way of saying this is that **either they are the same person, or one of them is related to an ancestor (parent, grandparent, etc.) of another**.

For example, Jack and Jill share the same parents, so they are related. Jack and Bob are related because they share their mother, Robin. Jill and Alice are related because of Jill's grandparents. Robin and Jack are related because Robin herself is in Jack's family tree. However, John and Robin are *not* related even though they had children together.

Using this knowledge, implement the function below using recursion. Note the precondition; this actually makes the function a lot simpler.

```
def related(p,q):
    """Return: True if Persons p and q are related
    If either p or q is None, or they are not related, it returns False
    Preconditions: p, q are each either a Person or None"""
```

Last Name: ______________ First: ______________ Netid: ________ Section ________

8. [16 points] **Call Frames**

Throughout this course, we have made heavy use of the `range` function. If we wanted to, we could actually implement this function with recursion. This would look like something to the right.
On this page and the next, diagram the execution of the assignment statement

```
>>> a = range(1)
```

```
1 def range(n):
2     if n == 0:
3         return [0]
4     right = [n]
5     left = range(n-1)
6     return left+right
```

You should draw a new diagram every time a call frame is added or erased, or an instruction counter changes. There are a total of **nine** diagrams to draw. You may write *unchanged* in any of the three spaces if the space does not change at that step.

Call Frames	Global Space	Heap Space

Last Name: ______________ First: ______________ Netid: ________ Section ________

Call Frames	Global Space	Heap Space