Last Name: _____ First: _____ Netid: _____ Section _____

CS 1110 Final, December 9th, 2015

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	18	
7	10	
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) [2 points] What value does Python give in the evaluation below? Explain your answer.

```
>>> a = [1,2,3,4]
>>> b = a[:]
>>> a is b
???
```

   (b) [4 points] Describe the four steps that happen when you call a constructor.

   (c) [4 points] Name the three sorting algorithms introduced in class. Which one is the one we prefer to use most of the time and why?

Last Name: _____ First: _____ Netid: _____ Section _____

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

Some of you may have worked with the class `GImage` in Assignment 7. This is a subclass of `GRectangle` that allows you to display an image. It has the following primary attributes (there are others, but they are not important for this question).

Attribute	Invariant	Description
`x`	`float`	x-coordinate of the image center.
`y`	`float`	y-coordinate of the image center.
`width`	`float > 0`	Distance from left side to right side.
`height`	`float > 0`	Distance from bottom edge to top edge.
`source`	`str` or `None`	Name of the image file.

If `source` is `None`, then the image is blank when drawn.

Similar to what you did in Assignment 7, we want you to make a subclass of `GImage`. This subclass is called `GFilmStrip`. A filmstrip is a sequence of images used in animation. For example, in the illustration to the right, you can see a character in four different poses. Taken together, these poses make it look like the character is running. We can animate this running by changing the `source` attribute in `GImage` to be a different image each animation frame.


run1.png


run2.png


run3.png


run4.png

The subclass `GFilmStrip` provides new features to make this animation a little easier. Instead of a single source, it has a *list of sources*. The contents can still be `None`, just as in `GImage`. That would indicate a blank image in that animation frame.

The specification of this class is as follows:

```
class GFilmStrip(GImage):
    """Instance is an animated sequence of images.

    (MUTABLE) ATTRIBUTES:
        _images [list, whose entries are str or None]: the animation frames
        _frame [int in 0..len(_images)-1]: the current animation frame

    In addition to these invariants, the inherited attribute source must always
    be equal to animation frame in _images, specified by _frame."""
```

Implement this class on the next page. We have provided you with an outline of the class, but you will need to fill in the blanks where specified. While we have specified the preconditions for `__init__`, you will notice that none of the getters or setters have preconditions. That is because **you must figure out what those preconditions are, and enforce them**. Remember that your class must enforce all invariants.

Hint: The attributes inherited from `GImage` 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: `_images` and `_frame`.

If you have not finished Assignment 7 yet, you are free to use this class in that assignment.

```
from game2d import *
class GFilmStrip(GImage):
    """See specification on previous page."""
    def getSize(self):
        """Returns the number of images in this filmstrip"""


    def getFrame(self):
        """Returns the position number of the active animation frame"""


    def setFrame(self,frame):
        """Sets the frame number and makes that image the active one in source"""









    def getImage(self,n):
        """Returns the image at position n (where n is an index of _images)"""


    def setImage(self,n,image):
        """Sets the image at position n (where n is an index of _images)"""









    def __init__(                                          ):    # FILL IN
        """Creates a new GFilmstrip centered at (0,0) with all blank images.

        The initializer does not assign any images, but it must create the list of
        size elements. Initially the list only contains None values. Images are set
        later with setImage.

        This initializer has four parameters: width, height, size, and frame. size
        is the number of images that the fimstrip can hold, while frame is the
        starting frame.  The last parameter is OPTIONAL and defaults to 0.

        Preconditions: width and height are floats > 0. size is an int > 0, while
        frame is an int in 0..size-1."""
```

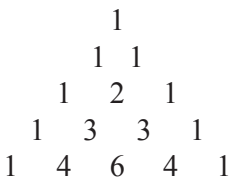
Last Name: _____ First: _____ Netid: _____ Section _____

4. [14 points] **Recursion and Iteration**

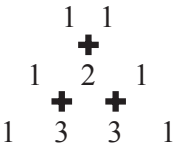
Some of you may remember Pascal's Triangle from your algebra class. This is the triangle that gives you the coefficients when you raise the polynomial $(x + 1)$ to a power. This triangle is shown to the right. We refer to each row of the triangle as a level, and a level can be represented as a list of ints. The row `[1]` is the 0th level, row `[1, 1]` is the 1st level, row `[1, 2, 1]` is the 2nd level and so on.

As you can see to the right, we compute each level by looking at the previous level. We put new 1s at the start and end of the list. For the values in-between, we add two adjacent elements from the previous list. As a result, each level has exactly one more element than the previous level.

Implement the function below which computes the specified level of the Pascal triangle. You do not need to enforce the precondition.



Rule:



Important: Your solution *must* use recursion, but you are also allowed to use a loop as well. However, you are restricted to **exactly one loop**. If you need more than one loop, you are not using recursion properly.

```
def pascal_level(n):
    """Returns: the nth level of the pascal triangle
    Precondition: n is an int >= 0."""
```

Last Name: _____ First: _____ Netid: _____ Section _____

5. [14 points] **Call Frames**

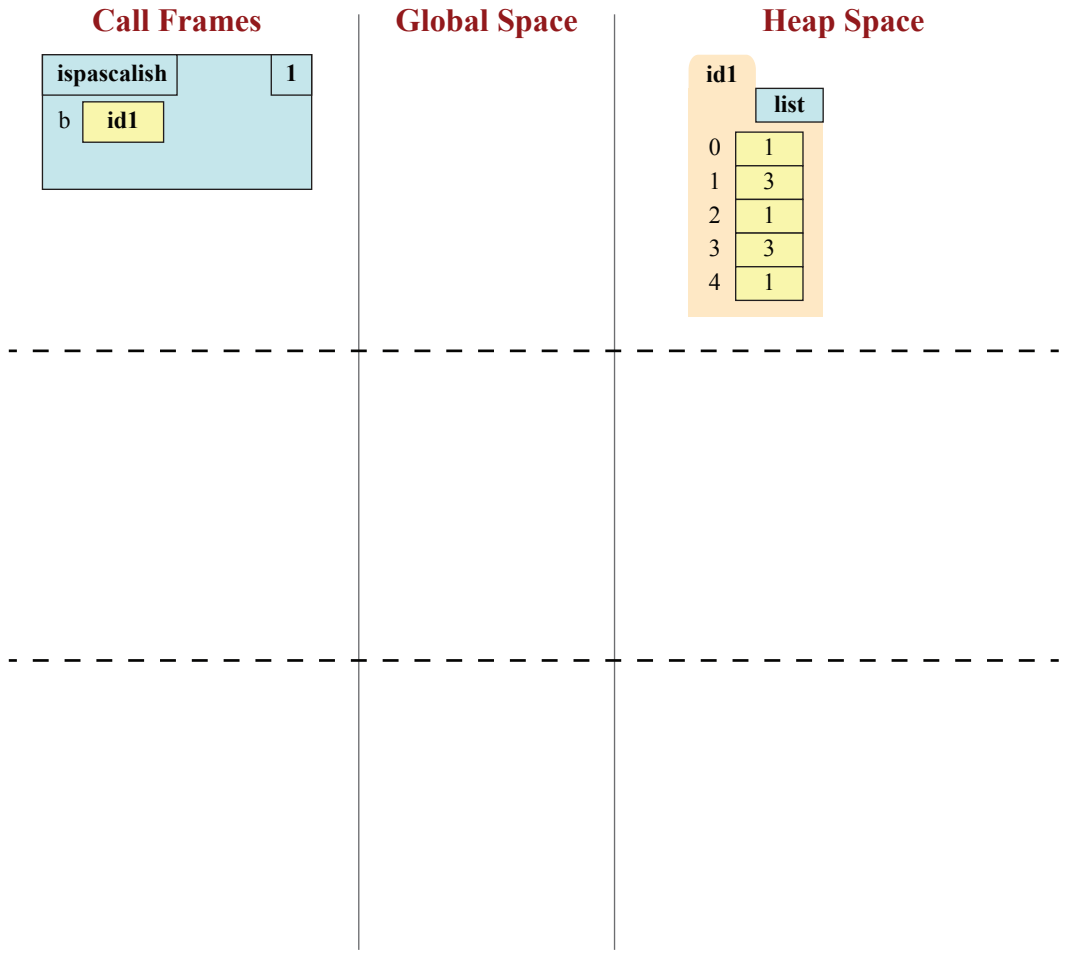
Each level of Pascal's triangle is a palindrome; it is the same backwards and forwards. It also has the additional property that values are always increasing as you go towards the center. The function below tests whether or not a list has these two properties. For example, while the lists `[2, 3, 2]` and `[1, 4, 4, 1]` are Pascal-ish, the lists `[2, 3]` and `[4, 1, 4]` are not.

```
def ispascalish(b)
    if len(b) < 2:                                          1
        return True                                         2

    if (len(b) > 2 and b[0] >= b[1]):                       3
        return False                                        4

    return b[0] == b[-1] and ispascalish(b[1:-1])           5
```

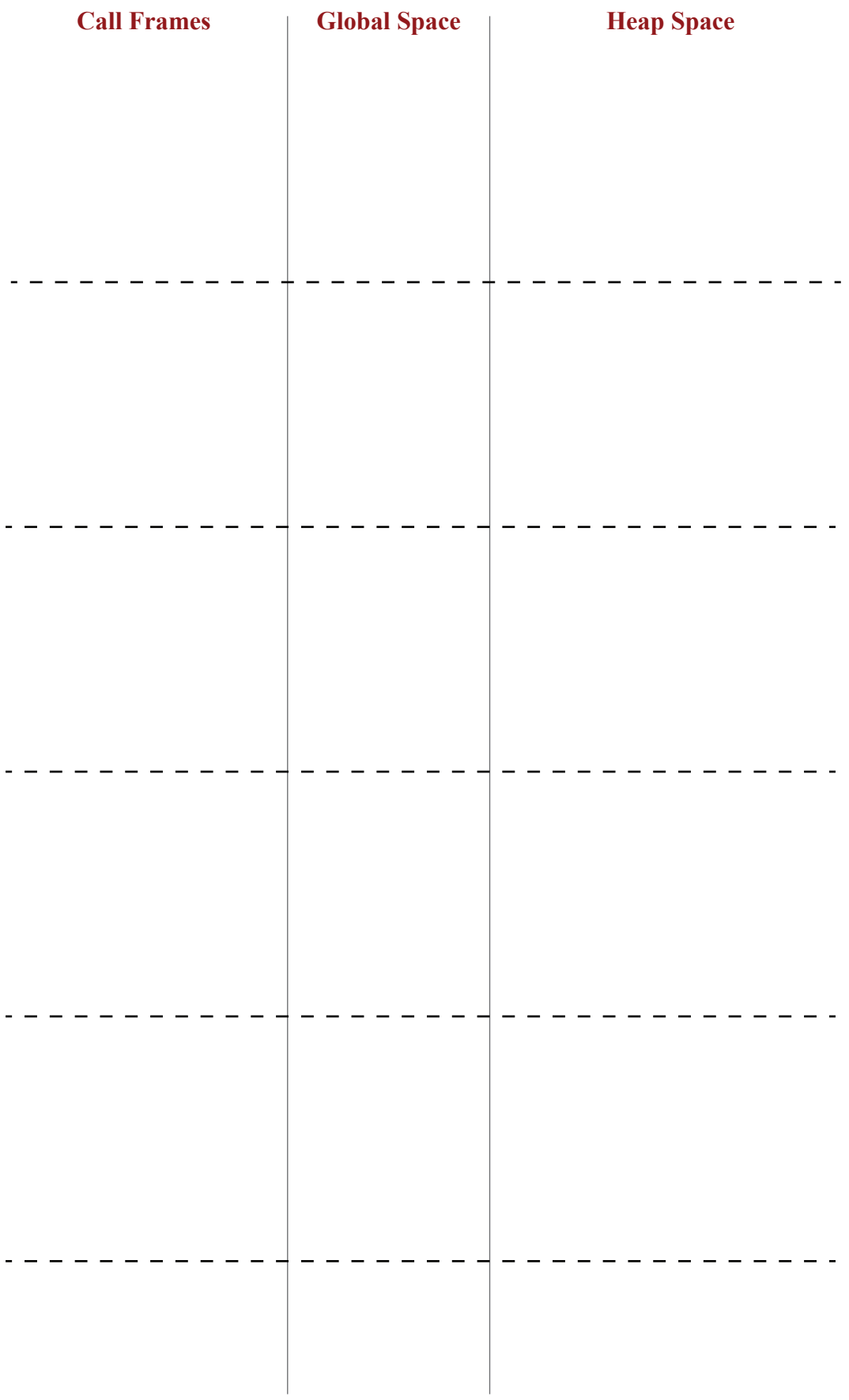
On this page and the next, diagram the execution of the assignment statement

```
>>> x = ispascalish([1,3,1,3,1])
```

You need a new diagram every time a frame is added or erased, or an instruction counter changes. **However, do not draw a frame for `len`.** We have done the first step for you. There are **eight** more diagrams to draw. You may write *unchanged* if a space does not change.



Last Name: _____ First: _____ Netid: _____ Section _____



6. [18 points total] **2-Dimensional Lists**

A matrix is a rectangular table of floats. It is a nonempty 2D list of floats where all rows have the same length. You add matrices by adding all of floats in the same position. For example,

$$\begin{matrix}[[1.5, & 2.0], \\ [2.5, & -1.0]]\end{matrix} + \begin{matrix}[[5.0, & 0.0], \\ [0.0, & 5.0]]\end{matrix} = \begin{matrix}[[6.5, & 2.0], \\ [2.5, & 4.0]]\end{matrix}$$

The *dimension* of a matrix is $n \times m$ where n is the number of rows and m the number of columns. Implement the two functions below. **You do not need to enforce the preconditions**.

(a) [7 points]

```
def sum_matrix(matrix1, matrix2):
    """Returns: new matrix that is sum of matrix1 and matrix2.
    Precondition: matrix1 and matrix2 are matrices with the same dimension"""
```

(b) [11 points]

```
def ismatrix(m):
    """Returns: true if m is a matrix (nonempty, rectangular with floats).
    Precondition: NONE (m can be anything)"""
```

7. [10 points] **Testing**

   Consider the function ismatrix from the previous question. It the space below, provide at least **six different test cases** to verify that this function is working correctly. For each test case provide: (1) the function input, (2) the expected output, and (3) an explanation of what makes this test *significantly* different.

8. [16 points total] **Loop Invariants**

   On the next page are two variations of the Dutch National Flag algorithm shown in class. The one on the left has been completed for you. The second algorithm on the right is identical to the first except that it has a different loop invariant. It is currently missing the code for initialization, the loop condition, and the body of the loop (all other code is provided).

   (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) [12 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. It is not enough for your code to be correct. **To get credit**, you must satisfy the loop invariant.

```
def dnf(b,h,k):
    """Returns: Divisions i, j of dnf
    Rearranges the list into negatives,
    0s, and positives according to DNF.
    Pre: h, k are positions in list b."""
    # Make invariant true at start

    i = h-1

    t = h-1

    j = k+1

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


    while t < j-1:
        if b[t+1] < 0:
            swap(b,i+1,t+1)

            t = t+1

            i = i+1

        elif b[t+1] == 0:
            t = t+1

        else:
            swap(b,j-1,t+1)

            j = j-1



    # post: b[h..i] < 0, b[i+1..j-1] = 0,
    # and b[j..k] > 0
    return (i,j)
```

```
def dnf(b,h,k):
    """Returns: Divisions i, j of dnf
    Rearranges the list into negatives,
    0s, and positives according to DNF.
    Pre: h, k are positions in list b."""
    # Make invariant true at start




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


    while ______________:




    # post: b[h..i-1] < 0, b[i..j] = 0,
    # and b[j+1..k] > 0
    return (i,j)
```