Last Name: _____ First: _____ Netid: _____

CS 1110 Final, December 13th, 2022

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. You may use the backside of each page for extra room for your answers. However, if you do this, **please indicate clearly** on the page of the associated problem.

Question	Points	Score
1	2	
2	9	
3	11	
4	14	
5	16	
6	20	
7	12	
8	16	
Total:	100	

The Important First Question:

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

Last Name: _____ First: _____ Netid: _____

References

String Operations

Expression	Description
`len(s)`	**Returns**: Number of characters in `s`; it can be 0.
`a in s`	**Returns**: True if the substring `a` is in `s`; False otherwise.
`s.find(s1)`	**Returns**: Index of FIRST occurrence of `s1` in `s` (-1 if `s1` is not in s).
`s.count(s1)`	**Returns**: Number of (non-overlapping) occurrences of `s1` in `s`.
`s.islower()`	**Returns**: True if `s` is *has at least one letter* and all letters are lower case; it returns False otherwise (e.g. `'a123'` is True but `'123'` is False).
`s.isupper()`	**Returns**: True if `s` is *has at least one letter* and all letters are upper case; it returns False otherwise (e.g. `'A123'` is True but `'123'` is False).
`s.lower()`	**Returns**: A copy of `s` but with all letters converted to lower case (so `'A1b'` becomes `'a1b'`).
`s.upper()`	**Returns**: A copy of `s` but with all letters converted to upper case (so `'A1b'` becomes `'A1B'`).
`s.isalpha()`	**Returns**: True if `s` is *not empty* and its elements are all letters; it returns False otherwise.
`s.isdigit()`	**Returns**: True if `s` is *not empty* and its elements are all numbers; it returns False otherwise.
`s.isalnum()`	**Returns**: True if `s` is *not empty* and its elements are all letters or numbers; it returns False otherwise.

List Operations

Expression	Description
`len(x)`	**Returns**: Number of elements in list `x`; it can be 0.
`y in x`	**Returns**: True if `y` is in list `x`; False otherwise.
`x.index(y)`	**Returns**: Index of FIRST occurrence of `y` in `x` (error if `y` is not in `x`).
`x.count(y)`	**Returns**: the number of times `y` appears in list `x`.
`x.append(y)`	Adds `y` to the end of list `x`.
`x.insert(i,y)`	Inserts `y` at position `i` in `x`. Elements after `i` are shifted to the right.
`x.remove(y)`	Removes first item from the list equal to `y`. (error if `y` is not in `x`).

Dictionary Operations

Expression	Description
`len(d)`	**Returns**: number of keys in dictionary `d`; it can be 0.
`y in d`	**Returns**: True if `y` is a key in dictionary `d`; False otherwise.
`d[k] = v`	Assigns value `v` to the key `k` in dictionary `d`.
`del d[k]`	Deletes the key `k` (and its value) from the dictionary `d`.
`d.clear()`	Removes all keys (and values) from the dictionary `d`.

Last Name: _____ First: _____ Netid: _____

2. [9 points total] **Short Answer**

(a) [3 points] Consider the following assignment statements.

```
>>> a = [[1,2,3],[4,5,6],[7,8,9]]
>>> b = a[1:]
>>> b[0] = [10,11]
>>> b[1][0] = 99
```

What are the values `a` and `b` after all these assignments? Explain your answer.

`a` is `[[1,2,3],[4,5,6],[99,8,9]]`

`b` is `[[10,11],[99,8,9]]`

The second line (the slice) puts `[[4,5,6],[99,8,9]]`. The third line replaces the first element of this 2D list with another list. The last line goes inside of the list `[7,8,9]` and changes the first element. Because of how slicing works, the folder for this list is shared by both `a` and `b`, so changes to one affect another.

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

A *parameter* is a variable in the parentheses at the start of a function definition.

An *argument* is an expression in the parentheses of a function call.

A function call evaluates the arguments and plugs the result into the parameters before executing the function body.

(c) [3 points] What is a iterable? What is an iterator? How are they related?

An iterable any value that may be used in a for-loop. An iterator may also be used inside of a for-loop. However, once it is used you have to recreate it if you want to use it a second time. You can also use the function `next` to step through the elements of an iterator one at a time (you cannot do this with an iterable).

Last Name: _____ First: _____ Netid: _____

3. [11 points total] **Testing and Exceptions**

(a) [7 points] Consider the following function specification.

```
def swapcase(s):
    """Returns a copy of s with the case of each letter swapped.

    Characters that are not letters are not affected.

    Precondition: s is a string"""
```

Do not implement this function. Instead, provide a list of at least **five test cases** to test this function. For each test case provide: (1) the function input, (2) the expected output, and (3) an explanation of what makes this test *significantly* different.

There are many possible answers. However, these were the main tests that we had in mind.

Input	Output	Reason
`''`	`''`	The empty list is always important
`'a'`	`'A'`	String with only lower case
`'A'`	`'a'`	String with only upper case
`'Aa'`	`'aA'`	String with a mix of cases
`'Aa!'`	`'aA!'`	String with non letter characters

(b) [4 points] Suppose you are given the following function definitions.

```
def first(n):
    try:
        x = second(n)
    except ArithmeticError:
        x = -n
    return x
```

```
def second(n):
    try:
        y = third(n)
    except ZeroDivisionError:
        y = 100
    return y
```

On the next page, complete the function `third` so that the following are all true.

- `first(0)` returns 100
- `first(n)` **crashes** whenever `n` < 0
- `first(n)` returns `-n` whenever `n` is odd
- `first(n)` returns `n//2` whenever `n` is even.

As an important restriction `third` is **not allowed to have any return statement other than the one provided**. You have to produce the functionality on the previous page by raising exceptions. You should use the exceptions `Exception`, `ArithmeticError`, and `ZeroDivisionError`. `ArithmeticError` is a subclass of `Exception` and `ZeroDivisionError` is a subclass of `ArithmeticError`.

Hint: Your answer should just consist of if-statements and statements that create errors.

```
def third(n):
    """A function that raises a lot of exceptions.

    See the previous page for the output.
    Precondition: n is an int."""


    if n == 0:
        raise ZeroDivisionError()
    elif n < 0:
        raise Exception()
    elif n % 2 == 1:
        raise ArithmeticError()



    # This is the ONLY return allowed in this function
    return n // 2
```

4. [14 points] **Call Frames**

Throughout this course, we have made heavy use of the `range` function. This function returns an iterable (not an iterator) that can be used in for-loops.

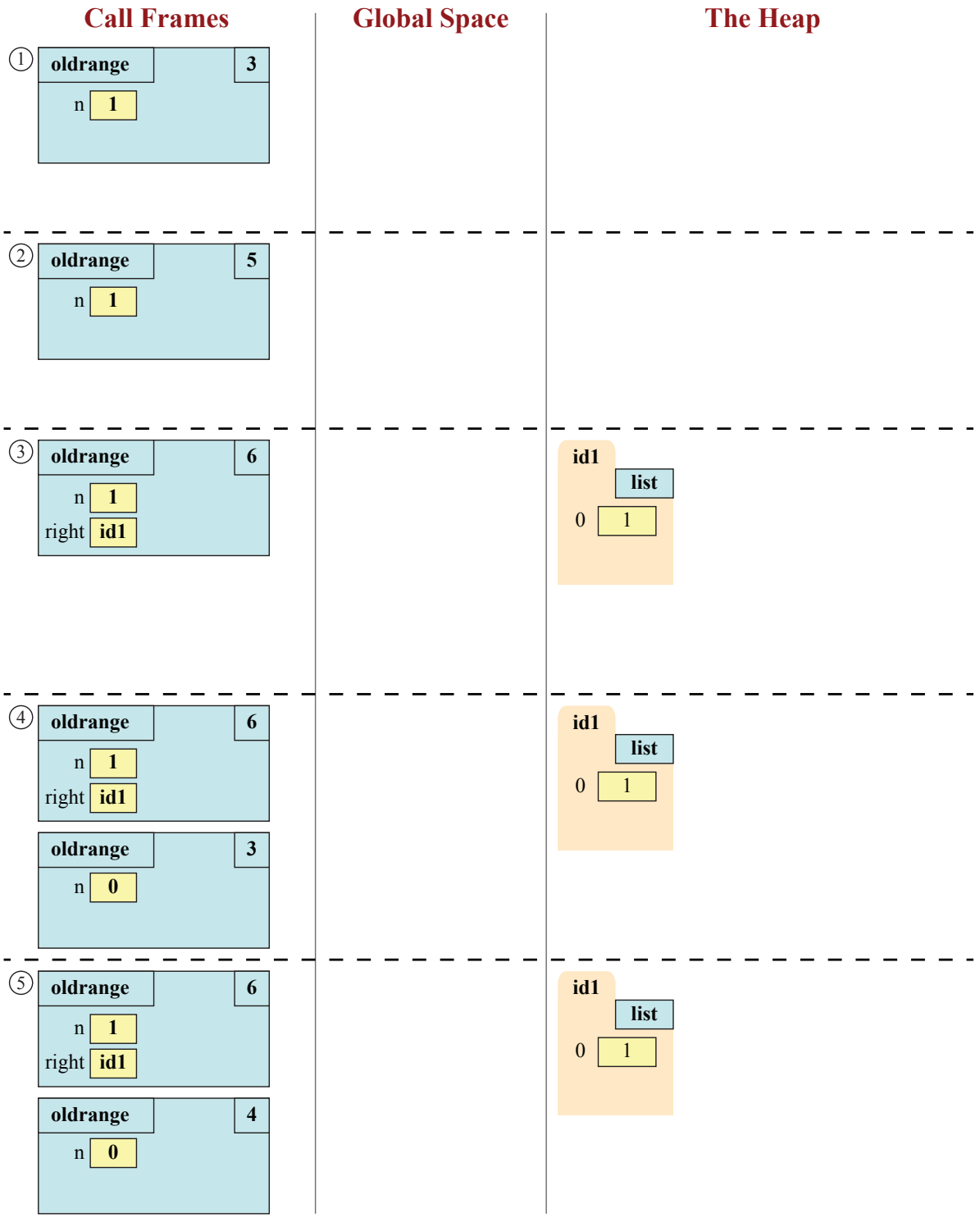
However, in Python 2, this function actually returned a list. Such a function would be simple for us to implement, as we could use resursion. This would look something like the function `oldrange` to the right (note that this version includes `n` in the list). On the next two pages, diagram the execution of the assignment statement

```
1 def oldrange(n):
2     """Returns[0,..,n] (inclusive)"""
3     if n == 0:
4         return [0]
5     right = [n]
6     left  = oldrange(n-1)
7     return left+right
```

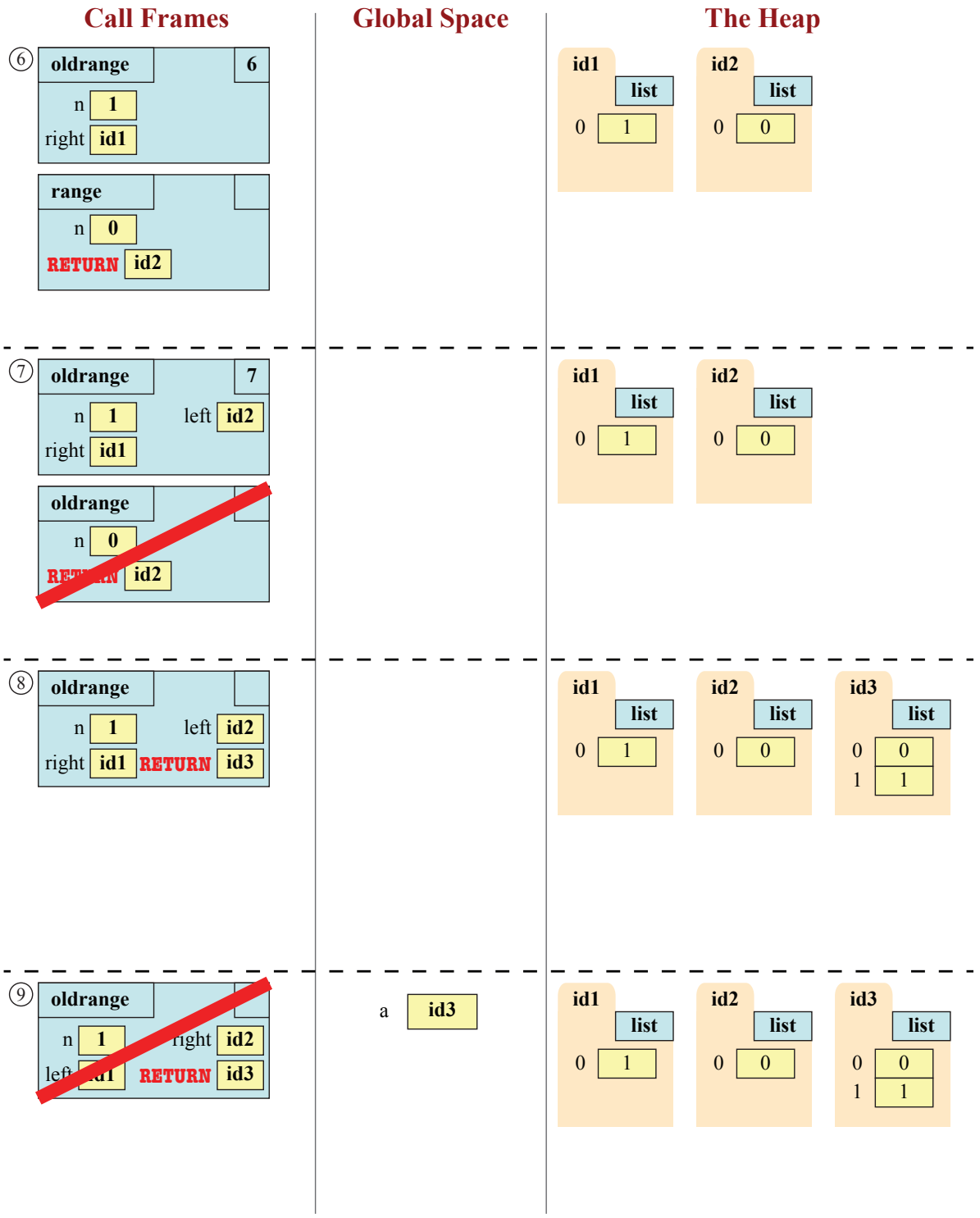
```
>>> a = oldrange(1)
```

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.

Last Name: _____ First: _____ Netid: _____



Last Name: _____ First: _____ Netid: _____



Last Name: _____ First: _____ Netid: _____

5. [16 points] **Nested Lists**

Implement the function below according to its specification. You may use any Python features that we have learned in class.

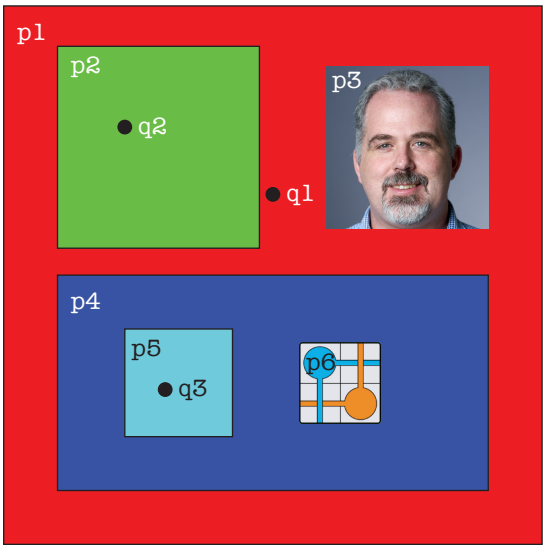
```
def pad_square(ragged):
    """MODIFIES the ragged 2D list by adding zeros to make a square table

    The number of rows (and columns) of the new table should be the maximum of
    the longest row and the longest column. Zeros should be added to both the
    rows and columns to fill out anything missing.

    Ex: If a = [[1,2],[3,4,5]], pad_square(a) changes a to [[1,2,0],[3,4,5],[0,0,0]]
    That is because the longest row is 3, and the longest column is 2. So we pad all
    row to 3 columns, and add a new row to make 3 rows.

    Ex: If a = [[1,2],[3,4],[5,6]], pad_square(a) changes a to [[1,2,0],[3,4,0],[5,6,0]]
    That is because the longest row is 2, and the longest column is 3. So we pad all
    rows to 3 columns, but we do not need to add a new row.

    Precondition: ragged is a ragged 2D list of integers"""

    # Find the size of the square
    maxlen = 0
    for row in ragged:
        if len(row) > maxlen:
            maxlen = len(row)
    maxlen = max(maxlen,len(ragged))

    # Pad the rows (add new columns)
    for row in ragged:
        for x in range(maxlen-len(row)):
            row.append(0)

    # Pad the columns (add new rows)
    for x in range(maxlen - len(ragged)):
        newrow = []
        for y in range(maxlen):
            newrow.append(0)
        ragged.append(newrow)
```

Last Name: _____ First: _____ Netid: _____

6. [20 points] **Classes and Subclasses**

For the next two questions, you will be working with a new class called `GPanel`. The class is a subclass of `GRectangle` from Assignment 7. This class is very similar to an important class in Java that you will see if you continue on to CS 2110.

A `GPanel` is just a `GRectangle` that can contain other instances of `GRectangles` (which can include objects that are a `GImage` – as it is a subclass of `GRectangle` – or even a `GPanel`). For example, in the image to the right, the `GPanel p1` contains the `GRectangle p2`, the `GImage p3`, and the `GPanel p4`. Furthermore, the `GPanel p4` contains the `GRectangle p5` and the `GImage p6`. The points `q1`, `q2`, and `q3` are part of the next question, and are not important just yet.



On the next page, you will implement the class `GPanel`. We have provided all of the specifications, but all method headers are incomplete. You are to complete all of these methods according to their specifications **and assert all preconditions**.

In order to implement this class, you will need to remember the attributes and methods of `GRectangle`. They are as follows.

Attribute	Invariant	Description
`x`	`float`	x-coordinate of the center of the rectangle.
`y`	`float`	y-coordinate of the center of the rectangle.
`left`	`float < x`	x-coordinate of the left edge of the rectangle.
`right`	`float > x`	x-coordinate of the right edge of the rectangle.
`top`	`float > y`	y-coordinate of the top edge of the rectangle.
`bottom`	`float < y`	y-coordinate of the bottom edge of the rectangle.
`width`	`float > 0`	The width along the horizontal axis.
`height`	`float > 0`	The height along the vertical axis.
`linecolor`	instance of `RGB`	The color of the rectangle border.
`fillcolor`	instance of `RGB`	The color of the rectangle interior.

Method	Description
`contains(x,y)`	**Returns**: True if the point (x,y) is in the rectangle; False otherwise
`draw(view)`	Draws the rectangle to the specified `GView` instance `view`.

There are other attributes, but they can be ignored for this problem. In particular, avoid using attributes that you might have remembered from Assignment 7, but are not listed in the tables above. When enforcing precoditions, you must use `isinstance` over `type`.

Finally, recall how the constructor for `GRectangle` works. You provide it with a list of keyword arguments that initialize various attributes. For example, to create a red square centered at (0,0), use the constructor call

```
GRectangle(x=0,y=0,width=10,height=10,fillcolor=introcs.RGB_RED)
```

The constructor for `GPanel` *does not* work this way. Please read its specification carefully.

Hint: The preconditions indicate every method parameter *other* than `self`.

```
from game2d import *
import introcs
class GPanel(GRectangle):
    """Class is a panel that can store GRectangles (including other GPanels)."""
    # INSTANCE ATTRIBUTES (in addition to those inherited from GRectangle):
    # Attribute _contents: The panel contents, which are called "entries"
    # Invariant _contents: a list of GRectangle (or subclass of GRectangle) objects
    # NOTE: Because _contents is a list, we replace set with add/remove/clear

    def addContents( self,  rect ):                                    # Fill in
        """Adds rect as an entry of this GPanel.

        Precond: rect is an instance of GRectangle, and is contained inside
        of the GPanel (left is >= panel's left, right is <= panel's right, etc)."""

        assert isinstance(rect,GRectangle)
        assert rect.left >= self.left and rect.right <= self.right
        assert rect.bottom >= self.bottom and rect.top <= self.top
        self._contents.append(rect)


    def removeContents( self,  rect ):                                 # Fill in
        """Removes rect as an entry of this GPanel.

        Precond: rect is an entry of this GPanel."""

        assert rect in self._contents
        while rect in self._contents:
            self._contents.remove(rect)


    def getContents( self ):                                           # Fill in
        """Returns a COPY of the list of entries in this GPanel.
        The entries do not need to be copied; only the list."""

        return self._contents[:]


    def clear( self ):                                                 # Fill in
        """Removes all entries from this GPanel"""

        self._contents = []
```

Last Name: _____ First: _____ Netid: _____

```
# Class GPanel (CONTINUED).


def __init__( self,  x,  y,  w,  h,  color = introcs.RGB_WHITE ):          # Fill in
    """Initializes a new GPanel of the given dimensions and color.

    Parameters x and y specify the center of the panel.
    Parameters w and h are the width and height of the panel.
    Parameter color is the fillcolor, and is introcs.RGB_WHITE
    by default. A new GPanel has NO entries to start.

    Precond: x, y, w, and h are floats with w, h > 0.
    color is an RGB object (class provided by introcs)."""

    assert isinstance(x, float) and isinstance(y,float)
    assert isinstance(w, float) and isinstance(h,float)
    assert w > 0 and h > 0
    assert isinstance(color,introcs.RGB)

    super().__init__(x=x,y=y,width=w,height=h,fillcolor=color)
    self._contents = []




def draw( self,  view ):                                                   # Fill in
    """Draws this GPanel AND ITS CONTENTS to the parameter view.

    The drawing order is important for this method. FIRST it draws the GPanel
    itself. Then it draws all of the entries, in the order that they are given
    in the list (so entries at the end of the list are on top).

    Precond: view is an instance of GView."""

    assert isinstance(view,GView)

    super().draw(view)
    for item in self._contents:
        item.draw(view)
```

Last Name: _____ First: _____ Netid: _____

7. [12 points] **Recursion**

We need one more method in `GPanel`. This method takes a point `(x,y)` and returns the top most entry containing that point. For example, in the image in question 6, `p1.selected(q1.x,q1.y)` returns `p1`, `p1.selected(q2.x,q2.y)` returns `p2`, and `p1.selected(q3.x,q3.y)` returns `p5`.

Hint: This is **not** a divide-and-conquer problem. There is a recursive definition in the specification below. Because of how the recursive definition is worded, you will need to use **both** recursion and iteration in this problem (why?). You will find the methods in `GRectangle` (listed in question 6) to be useful.

You do not need to assert the preconditions for this problem.

```
def selected(self,x,y):
    """Returns the topmost entry containing (x,y)

    If (x,y) is not inside this GPanel, this method returns None. Otherwise, it
    returns the entry that is topmost AMONG THOSE CONTAINING (x,y).

    Remember that objects are drawn back to front. We define the topmost entry
    of a GRectangle as follows:
    - The topmost of a GRectangle that is NOT a GPanel is the GRectangle itself
    - The topmost of an EMPTY GPanel is the GPanel itself
    - The topmost of a GPanel is the topmost of the final entry in the list
    Again, this function only considers entries that contain (x,y) when
    computing the topmost.

    Precondition: x and y are floats."""

    # Handle case if point outside
    if not self.contains((x,y)):
        return None

    # In case no entry contains (x,y)
    choice = self

    # Loop through all entries
    for item in self._contents:
        if item.contains((x,y)):
            if isinstance(item,GPanel):
                choice = item.selected(x,y)
            else:
                choice = item

    # Return what was found
    return choice
```

Last Name: _____ First: _____ Netid: _____

8. [16 points total] **Generators**

Implement the generators below using anything learned in class. However, note the precondition of `fold_head`. Iterables that are not lists or strings cannot be sliced and have no length.

(a) [7 points]

```
def emit_vowels(s):
    """Generates the vowels of s in order

    Vowels are 'aeiouy', but a 'y' as the first character is not a vowel
    Example: emit_vowels('examination') generates 'e', 'a', 'i', 'a', 'i', and 'o'.

    Precond: s is a string of lowercase letters"""

    vowels = 'aeiouy'

    # Track the first vowel
    first = True
    for ch in s:
        if ch != 'y' or not first:
            if ch in vowels:
                yield ch
        first = False
```

(b) [9 points]

```
def fold_head(input,n):
    """Generates every element of input STARTING at n, added together with the
    elements at positions 0..n-1

    If input has less than n elements it generates only the sum of the elements.
    Example: Let a = (1,2,3,4). fold_head(a,1) generates 3, 4, 5, while fold_head(a,2)
    generates 6, 7. fold_head(a,3), fold_head(a,4), and fold_head(a,5) all generate 10.

    Precond: input is a nonempty iteratable of integers; n >= 0 is an integer"""

    sum = 0
    pos = 0

    # Track position with pos
    for item in input:
        if pos < n:
            sum += item
        else:
            yield item+sum

        pos += 1

    # Must guarantee one output
    if pos <= n:
        yield sum
```