Last Name: ____________ First: ____________ Netid: ____________

CS 1110 Final, December 8th, 2023

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	18	
4	16	
5	14	
6	12	
7	14	
8	15	
Total:	100	

The Important First Question:

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

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`.

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

   (a) [3 points] What is a *fruitful function*? What is a *procedure*? Give an example of each.

   Both fruitful functions and procedures are functions. However, a fruitful function returns a value while a procedure does not (it returns nothing). The built in function `round` is fruitful, while `print` is a procedure.

   (b) [3 points] Execute the four statements below. What is printed out? Explain your answer.

```
>>> a = [1,2]
>>> b = a[:]
>>> print (a == b)                  True
>>> print (a is b)                  False
```

   The operator `==` compares the contents of the lists while `is` compares the folder ids. Since slicing makes a copy, `a` and `b` are different folders.

   (c) [3 points] What are the names of the two sorting algorithms that take n^2 steps to complete? What is the difference between them?

   These algorithms are insertion sort and selection sort. They are similar, except that selection sort makes sure that all of the values sorted so far are less than or equal to the unsorted values.

3. [18 points total] **Testing and Exceptions**

(a) [10 points] Consider the following function specification.

```
def is_csv(value):
    """Returns True if value is a table of strings (CSV table); False otherwise.

    A table is a 2d list containing non-empty lists of the same length. An
    example of a CSV table would be [['a','b'],['c','d']]
    Precond: value is a non-empty list (it can be any list)"""
```

Do not implement this function. Instead, provide a list of at least **six 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. To be different, the examples should prioritize the most important things checked by this function.

There are many possible answers. The most important thing is that you focus on the types of content present in the list(s). The contents of the strings themselves is not that important (as all strings are treated the same). Given this fact, these were the main tests that we had in mind.

Input	**Output**	**Reason**
`['a']`	`False`	A list that does not contain lists.
`[['a'],'b']`	`False`	A list that combines lists with non-lists.
`[['a'],['b','c']]`	`False`	A list with elements of different lengths.
`[[],[]]`	`False`	A list containing only empty lists.
`[['a',2]]`	`False`	A lists of lists with non-strings in it.
`[['a','b']]`	`True`	A table with a single row.
`[['a','b'],['c','d']]`	`True`	A table with mutliple rows.

(b) [4 points] Given the function below, enforce its preconditions (but do **not** implement it) according to the specification. You should **not use assert statements**. Instead, write code that produces the errors specified. You may use the function above as a helper.

```
def flatten(value):
    """Returns a CSV table flattened into a list of strings.

    Example: flatten([['a','b'],['c','d']]) returns ['ab','cd']
    Precond: value is a valid CSV table. This function produces a TypeError if
    it is not a list, and a ValueError if it is a list but not a CSV table."""

    if type(value) != list:
        raise TypeError()
    if value == [] or not is_csv(value):
        raise ValueError()



    # Do NOT implement this function. Precondition only.
```

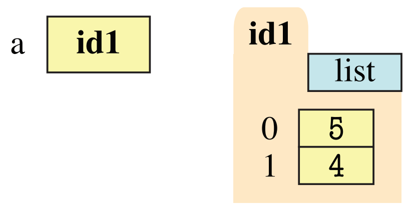
(c) [4 points] Implement the function below. You may use any of the previous two function as helpers. However, you **may not use if-statements**; you must use **try-except**.

```
def flatten_if_safe(value):
    """Returns the flattened list of value IF it is a CSV table.

    If value is a list but not a CSV table, this function returns the empty
    list. If it is not a list, this function returns None.
    Precond: value can be ANYTHING"""

    try:
        return flatten(value)
    except ValueError:
        return []
    except TypeError:
        return None
```

4. [16 points] **Call Frames**

Suppose we have a list `a = [5,4]`, together with the recursive function `reverse` shown below.

```
def reverse(seq):
    """Returns the reverse of seq"""
    right = seq[1:]
    if right == []:
        return seq

    left = seq[:1]
    return reverse(right)+left
```



On the next two pages, diagram the evolution of the function call

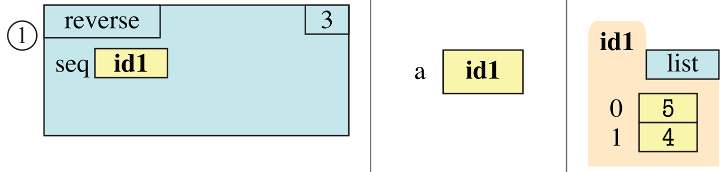
```
b = reverse(a)
```

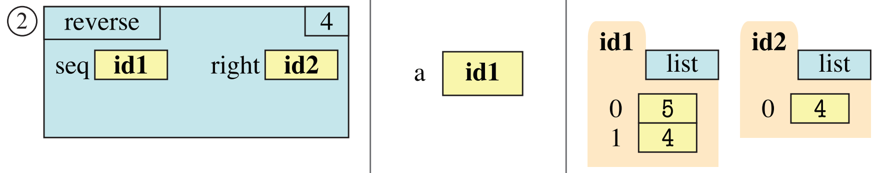
Diagram the state of the *entire call stack* for the function `reverse` when it starts, for each line executed, and when the frame is erased. If any other functions are called, you should do this for them as well (at the appropriate time). This will require a total of **ten** diagrams.

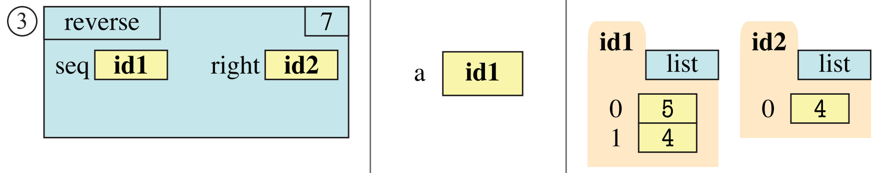
You should draw also the state of global space and the heap at each step. You can ignore the folders for the function definitions. Only draw folders for lists or objects. To help conserve time, you are allowed (and **encouraged**) to write "unchanged" if no changes were made to either a call frame, the global space, or the heap. In the case of the heap, you can point out that individual folder ids are unchanged.

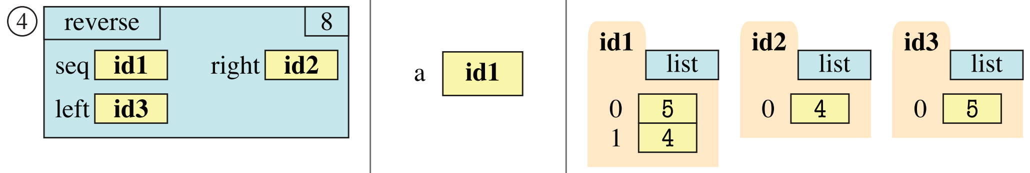
Last Name: ____________ First: ____________ Netid: ____________

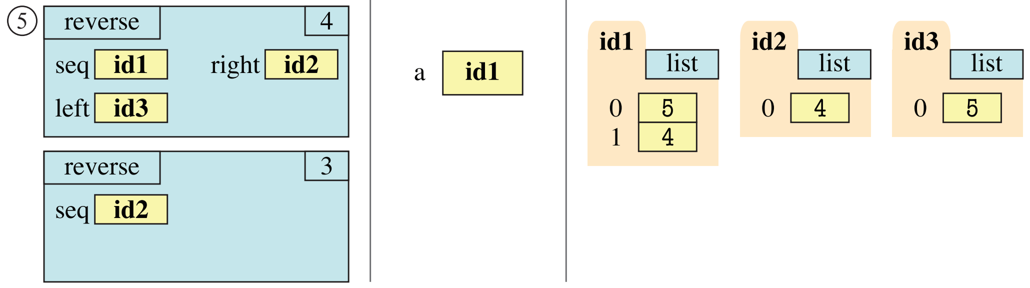
Call Stack **Global Space** **The Heap**



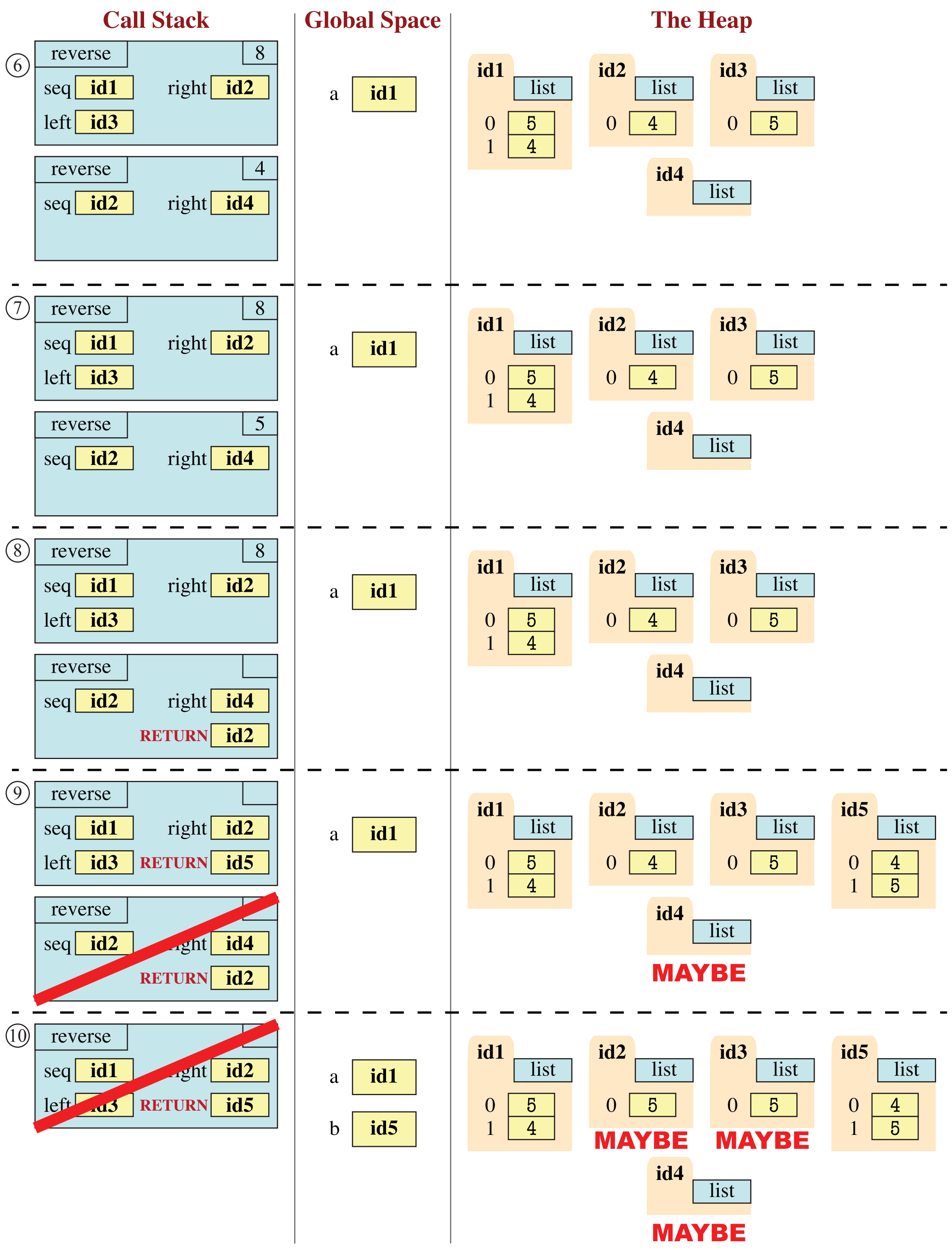








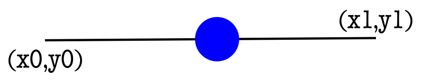
Last Name: ______________________ First: ______________________ Netid: ______________________



Last Name: ______________ First: ______________ Netid: ______________

5. [14 points] **Classes and Subclasses**

For this question, you are going to use the classes of Assignment 7 to make a `GSlider`. Shown to the right, this is a GUI element that you used to change the colors in Assignment 3. A slider is built up of two graphical objects, a `GPath` and a `GEllipse`. The former is the line that displays the path of the slider. The latter is the knob that the user drags to control the slider.



One easy way to make a GUI element composed of two different objects is to make the class a subclass on one and have the other as an attribute. That is what we have done on the next page. `GSlider` is a subclass of `GPath` but it has a `_knob` attribute for the `GEllipse`.

For this question, you only need to pay attention to the `points` attribute of `GPath`; all other attributes can be left to their default values (e.g. the line color can be left as the default `'black'`). The `points` attribute is an **even-length** list of numbers (ints **or** floats) expressing the points the path goes through. For example, a line segment from (0,1) to (5,-3) would have `points` attribute [0,1,5,-3].

For the class `GEllipse` you only need to know the following attributes.

Attribute	Invariant	Description
`x`	`float or int`	x-coordinate of the ellipse center.
`y`	`float or int`	y-coordinate of the ellipse center.
`width`	`float > 0 or int > 0`	The width along the central horizontal axis.
`height`	`float > 0 or int > 0`	The height along the central vertical axis.
`fillcolor`	`str`	The interior color (given as the name, e.g. `'blue'`).

In addition, remember that both `GPath` and `GEllipse` use keyword arguments in their constructors. So to create a red ellipse of radius 10 at the origin, you would specify

```
ellipse = GEllipse(x=0,y=0,width=20,height=20,fillcolor='red')
```

Implementing mouse control is too messy for an exam, so we control the slider with an additional attribute called `_value`. This attribute must be in sync with the knob. When it changes, the knob moves (we will not worry about what to do when the knob moves first). This is expressed by the specification on the next page.

With this in mind, implement this class on the next page. We have provided 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. All type-based preconditions should be enforced by `isinstance`.

Hint: The attributes in `GPath` and `GEllipse` 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 new attributes: `_knob` and `_value`.

```
class GSlider(GPath):
    """A class representing a graphical slider."""
    # MUTABLE ATTRIBUTES:
    # Invariant: _value is a float in 0 to 1, inclusive
    # IMMUTABLE ATTRIBUTES:
    # Invariant: _knob is a GEllipse
    # ADDITIONAL INVARIANT
    # If the slider path is from (x0,y0) to (x1,y1), the _knob is located at
    # x = (x1-x0)*_value+x0,  y =(y1-y0)*_value+y0
    # DEFINE GETTERS/SETTERS AS APPROPRIATE. SPECIFICATIONS NOT NEEDED.
    def setValue(self,value):
        """Sets the slider value"""
        assert isinstance(value, float)
        assert 0 <= value and value <= 1
        self._value = value
        self._knob.x = (self.points[2]-self.points[0])*value+self.points[0]
        self._knob.y = (self.points[3]-self.points[1])*value+self.points[1]

    def getValue(self):
        """Gets the slider value"""
        return self._value

    def getKnob(self):
        """Gets the slider knob"""
        return self._knob

    def __init__(self,    x0,    y0,    x1,    y1,    radius = 10): # Fill in parameters
        """Initializes a slider whose path is from (x0,y0) to (x1,y1)

        The initial value is 0. The slider line is black, which is the default color
        (no need to set it). The knob has the given radius and a fillcolor of 'blue'.
        Precond: x0, y0, x1, and y1 (in that order) are all ints
        Precond: radius is an int >= 1 (OPTIONAL; default 10)"""
        assert isinstance(x0,int) and isinstance(x1,int)
        assert isinstance(y0,int) and isinstance(y1,int)
        assert isinstance(radius,int) and radius >= 1
        super().__init__(points=[x0,y0,x1,y1])
        self._knob = GEllipse(x=x0,y=y0,width=2*radius,height=2*radius,fillcolor='blue')
        self._value = 0



    def draw(self,                    view                    ):     # Fill in parameters
        """Draws the knob ON TOP of the slider path.

        Precond: view is a GView object"""
        assert isinstance(view,GView)
        super().draw(view)
        self._knob.draw(view)
```

6. [12 points] **Nested Lists**

A triangular list is a ragged list whose rows start (end) at size one and increase (decrease) by one each row. Unlike a table, not all rows have the same number of columns. But the length of the largest column is equal to the number of rows.

Triangular lists come in two forms. In an *upper triangular list*, the largest row is the first one. In a *lower triangular list*, the largest row is the last one. Examples of these are shown below.

$$\begin{array}{l} [\ [1,\ 2,\ 4,\ 5], \\ \quad [0,\ 6,\ -2], \\ \quad [3,\ -1], \\ \quad [8]\ ] \end{array} \qquad \begin{array}{l} [\ [5], \\ \quad [4,\ -2], \\ \quad [2,\ 6,\ -1], \\ \quad [1,\ 0,\ 3,\ 8]\ ] \end{array}$$

Upper Triangular List **Lower Triangular List**

The function `rot_ccw` below takes an upper triangular list and rotates it counter-clockwise. This is similar to transpose (shown in class) where rows are swapped with columns. But the rows in the rotated copy are arranged **in reverse order from the columns in the original**. For example, `rot_ccw` takes the upper triangular matrix above on the left and returns the lower triangular matrix on the right. With this explanation, implement the function below.

Hint: Do *not* think about looping over the original triangular list. That will confuse you. Think about looping over the list you are returning (the one you are building), and then figure out how to get the elements you want from the original list.

```python
def rot_ccw(tri):
    """Returns a new lower triangular list from rotating tri counter-clockwise

    Precondition: tri is an upper triangular list"""

    # Get the basic dimensions
    nrows = len(tri)
    ncols = 1

    # Build the new list one row at a time
    result = []
    for r in range(nrows):
        row = []

        # Add the elements for each row
        for c in range(ncols):
            # Find the right element to add
            item = tri[c][nrows-ncols]
            row.append(item)

        result.append(row)
        # Number of elements in row gets larger
        ncols = ncols+1

    return result
```

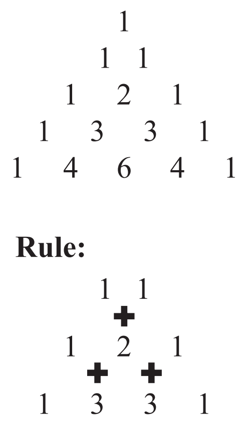
Last Name: ________________ First: ________________ Netid: ________________

7. [14 points] **Recursion and Iteration**

Some of you may remember Pascal's Triangle from your algebra class. This triangle 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 shown on the right, we compute each level from the previous one. We put new 1s at the start and end of the level. For the values in-between, we add two adjacent elements from the previous level. 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.



Important: Your solution must use recursion, but you are also allowed to use a loop as well. Note that this question is *not* a divide-and-conquer problem. Read the paragraph above to see the recursive definition.

```
def pascal_level(n):
    """Returns: the nth level of the pascal triangle

    Precondition: n is an int >= 0."""

    # Must handle both base cases separately
    if n == 0:
        return [1]
    elif n == 1:
        return [1,1]

    # Recursive call to access previous level
    prev = pascal_level(n-1)

    # Start the current level
    result = [1]

    # Compute the interior points
    for x in range(len(last)-1):
        result.append(last[x]+last[x+1])

    # Add the final value and return
    result.append(1)
    return result
```

8. [15 points total] **Generators**

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

   (a) [6 points]

```
def bythrees(stream):
    """Generates every third element of the iterable stream

    Ex: bythrees([1,5,7,0,2,4,3]) generates 1, 0, 3
    Ex: bythrees([1]) generates 1
    Precond: stream is iterable (can be used in a for-loop)"""
    # HINT: Look at the precondition. stream is NOT a sequence

    # Track our current position
    pos = 0
    for item in stream:
        if pos % 3 == 0:
            yield item
        pos += 1
```

   (b) [9 points]

```
def fibfrom(n):
    """Generates the Fibonnacci sequence, starting with the nth value.

    The Fibonnaci sequence starts with 1 and 1, and adds the previous two values
    to get the next value. This sequence is infinite, so the generator is too.
    Ex: fibfrom(0) generates 1, 1, 2, 3, 5, 8, 13, 21, 34, ...
    Ex: fibfrom(4) generates 5, 8, 13, 21, 34, ...
    Precond: n >= 0 is an int"""
    # HINT: You do not need recursion for this. Do not use recursion.

    # Track current value, previous value, position
    prev = 0
    curr = 1
    pos = 0

    # Loop forever
    while True:
        if pos > n:
            yield curr

        # Update the loop variables
        temp = curr
        curr = curr+prev
        prev = temp
        pos = pos + 1
```