Review 4

Lists and Sequences
Overview of List Syntax

- `x = [0, 0, 0, 0]`: Create list of length 4 with all zeroes
- `x.append(2)`: Append 2 to end of list `x` (now length 5)
- `3 in x`: Evaluates to `False` (3 not in `x`)
- `x[2] = 5`: Assign 5 to element 2
- `x[0] = -4`: and -4 to element 0
- `k = 3`: Assign -8 to `x[3]` and 6 to `x[1]`
<table>
<thead>
<tr>
<th>Lists vs.</th>
<th>Tuples vs.</th>
<th>Strings</th>
</tr>
</thead>
</table>
| **Creation**  
\[x = [a_1, a_2, a_3, \ldots]\]  
Can contain anything  
**len**(*x*) is length  
**Supports slicing**  
*Example:* *x*[1:2]  
*x*[i] is an element  
**Can concatenate**  
*y* = *x* + [1, 2]  
Makes a new list  
**Is mutable**  
*x*.append(5)  
**Creation**  
\[(a_1, a_2, a_3, \ldots)\]  
Can contain anything  
**len**(*x*) is length  
**Supports slicing**  
*Example:* *x*[1:2]  
*x*[i] is an element  
**Can concatenate**  
*y* = *x* + (1, 2)  
Makes a new tuple  
**Is not mutable**  
**Creation**  
*'Hello'*  
Only contains chars  
**len**(*x*) is length  
**Supports slicing**  
*Example:* *x*[1:2]  
*x*[i] is a substring  
**Can concatenate**  
*y* = *x* + 'World'  
Makes a new string  
**Is not mutable** |
Each element in the list `scores` contains the number of students who received score `i` on a test. For example, if 30 students got 85, then `scores[85]` is 30. Write the body of function `histogram`, which returns a histogram as a list of strings. (You need not write loop invariants.) For example, if `scores = [7, 0, 4, 3, 2, 0, ...]` then the first elements of the resulting string list are:

```
'00 *******'
'01 '
'02 ****'
'03 ***'
'04 *
'05 '
```
def histogram(scores):
    """Return a list of Strings (call it s) in which each s[i] contains:
    (1) i, as a two-digit integer (with leading zeros if necessary)
    (2) a blank,
    (3) n asterisks '★', where n is scores[i].
    Precondition: scores is a list of nonnegative integers, len(scores) < 100"""

# IMPLEMENT ME
def histogram(scores):
    """Return a list of Strings (call it s) in which each s[i] contains:
    (1) i, as a two-digit integer (with leading zeros if necessary)
    (2) a blank,
    (3) n asterisks '*', where n is scores[i].
Precondition: scores is a list of nonnegative integers, len(scores) < 100"""
    s = []    # List to contain the result.
    for i in range(len(scores)):    # Need the value i, not the elements of scores
        # Row is the string for this row
        row = str(scores[i])+' ' if scores[0] > 10 else 'O'+str(scores[i])+' '    
        for n in range(scores[i]):    # Loop over number of elements in scores[i]
            row = row+'*'    # Add another * to the row
        s.append(row)    # Add row to the list
    return s
## Overview of Two-Dimensional Lists

- **Access value at row 3, col 2:**
  \[ d[3][2] \]

- **Assign value at row 3, col 2:**
  \[ d[3][2] = 8 \]

- **An odd symmetry**
  - Number of rows of \( d \):  \( \text{len}(d) \)
  - Number of cols in row \( r \) of \( d \):  \( \text{len}(d[r]) \)

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>5</td>
<td>4</td>
<td>7</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>8</td>
<td>9</td>
<td>7</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>1</td>
<td>2</td>
<td>9</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>0</td>
</tr>
</tbody>
</table>
How Multidimensional Lists are Stored

- \( b = [[[9, 6, 4], [5, 7, 7]]] \)

- \( b \) holds name of a one-dimensional list
  - Has \( \text{len}(b) \) elements
  - Its elements are (the names of) 1D lists

- \( b[i] \) holds the name of a one-dimensional list (of ints)
  - Has \( \text{len}(b[i]) \) elements
Recall drawing GRectangles in A7. Write method placeSquares, whose requirements appear below. It draws square bricks as shown to the right and returns them as a 2d list of GRectangle.

```python
def placeSquares(self, m):
    """Create a list of m x m squares (GRectangle), as specified below, adding the squares to the GUI, and return the list."""
```

Method Requirements:

- There are \( m \) columns and rows of squares; precondition: \( 0 < m \).
- Each square has side length \( \text{BRICK\_SIDE} \); there is no space between them.
- The bottom-left square is at the bottom-left corner (0,0) of the GUI. Squares in columns and rows 0 and \( m-1 \) have color 'pink'.
- Inner squares have checkerboard pattern of 'red' and 'green', as shown (bottom-left one is green; one next to it, red).
Recall drawing GRectangles in A7. Write method placeSquares, whose requirements appear below. It draws square bricks as shown to the right and returns them as a 2d list of GRectangle.

```python
def placeSquares(self, m):
    """Create a list of m x m squares (GRectangle), as specified on last slide, adding them to the GUI, and return the list."""
```

API Reminders:

- GRectangle has attributes pos (a 2 element tuple), size (a 2 element tuple), fillcolor, and linecolor
- You construct a GRectangle with keyword arguments: `GRectangle(pos=(0,0), size=(10,10), color='blue')`
- You add to the GUI with `self.view.add(...)"""
def placeSquares(self, m):
    """Place the m x n Bricks, as requested on the exam and return the list"""
    bricks = []  # Make a new list to represent columns
    while c < m:  # Place col c of bricks
        row = []  # Make a new list to represent rows
        while r < m:
            color = 'red'
            if r == 0 or r == m-1 or c == 0 or c == m-1:
                color = 'pink'
            elif r+c % 2 == 0:
                color = 'green'
            brick = GRectangle(pos=(r*BRICK_SIDE, c*BRICK_SIDE), fillcolor=color,
                                size=(BRICK_SIDE, BRICK_SIDE), linecolor=color)
            row.append(brick)
            self.view.add(brick); r = r+1
        bricks.append(row)
        c = c+1
    return bricks
Ragged Lists: Rows w/ Different Length

- $b = [[17, 13, 19], [28, 95]]$

- To create a ragged list
  - Create $b$ as an empty list ($b = []$)
  - Create each row as a list ($r1 = [17, 13, 19]$; $r2 = [28, 95]$)
  - Append lists to $b$ ($b$.append(r1); $b$.append(r2))
Someone messed up a method to create certain arrays for us. For example (and this is only an example), they produced the array:

```
3 1 2 1 2 3
2 1 7 8 5 instead of 1 7 8 5 2
5 the array 5
6 8 8 6
```

Thus, they put the last value of each row at the beginning instead of the end. Write a procedure that fixes this by rotating each row one position to the left; each element is moved one position earlier, and the first element is placed in the last position. Do not use recursion. **DO NOT RETURN A VALUE.**

def rotate(b):
    """Rotate each row one position to the left, as explained above.
    Precondition: b is a list, might be ragged, and each row has >= 1 value"""
```python
def rotate(b):
    """Rotate each row one position to the left, as explained on the previous slide.
    Precondition: b is a list, might be ragged, and each row has >= 1 value""
    # Process each row
    for r in range(len(b)):
        # Remember the first element so we can put it at the end
        first = b[r][0]
        # Start at second element and shift each to the left
        for c in range(1, len(b[r])):
            b[r][c-1] = b[r][c]
        # Put the first element at the end
        b[r][len(b[r])-1] = first
```
def rotate(b):

    """Rotate each row one position to the left, as explained on the previous slide.
    Precondition: b is a list, might be ragged, and each row has >= 1 value"

    # Process each row
    for r in range(len(b)):
        # Remember the first element so we can put it at the end
        first = b[r][0]
        # Start at second element and shift each to the left
        for c in range(1,len(b[r])):
            b[r][c-1] = b[r][c]
        # Put the first element at the end
        b[r][len(b[r])-1] = first

    return b

Watch this in the Python Tutor
def reduce(matrix, row, col):
    """ Returns a copy of the matrix, missing the given row and column.
    Precondition: matrix is a table of numbers, row is an index (int) for a row,
    while col is an index (int) for a column"

\[
\begin{bmatrix}
1 & 5 & 0 \\
2 & 3 & -4 \\
1 & 0 & 2 \\
1 & 1 & -1 \\
\end{bmatrix}
\quad = \quad
\begin{bmatrix}
1 & 0 \\
2 & -4 \\
1 & -1 \\
\end{bmatrix}
\]
def reduce(matrix, row, col):
    """ Returns a copy of the matrix, missing the given row and column.  
    Precondition: matrix is a table of numbers, row is an index (int) for a row, 
    while col is an index (int) for a column""
    rows = len(matrix)
cols = len(matrix[0])
copy = []  # Accumulator for table
    for r in range(rows):
        if r != row:
            copyrow = []  # Accumulator for row
                for c in range(cols):
                    if c != col:
                        copyrow.append(matrix[r][c])
            copy.append(copyrow)
    return copy