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**
  - Assign -4 to element 0
- **k = 3**
- **x[k] = 2 * x[0]**
  - Assign -8 to x[3]
- **x[k-2] = 6**
  - Assign 6 to x[1]
<table>
<thead>
<tr>
<th>Lists vs.</th>
<th>Tuples vs.</th>
<th>Strings</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Creation</strong>&lt;br&gt;x = [a1, a2, a3, \ldots]&lt;br&gt;Can contain anything</td>
<td><strong>Creation</strong>&lt;br&gt;x = (a1, a2, a3, \ldots)&lt;br&gt;Can contain anything</td>
<td><strong>Creation</strong>&lt;br&gt;x = 'Hello'&lt;br&gt;Only contains chars</td>
</tr>
<tr>
<td><strong>len(x)</strong> is length</td>
<td><strong>len(x)</strong> is length</td>
<td><strong>len(x)</strong> is length</td>
</tr>
<tr>
<td><strong>Supports slicing</strong>&lt;br&gt;Example: x[1:2]&lt;br&gt;x[i] is an element</td>
<td><strong>Supports slicing</strong>&lt;br&gt;Example: x[1:2]&lt;br&gt;x[i] is an element</td>
<td><strong>Supports slicing</strong>&lt;br&gt;Example: x[1:2]&lt;br&gt;x[i] is a substring</td>
</tr>
<tr>
<td><strong>Can concatenate</strong>&lt;br&gt;y = x + [1, 2]&lt;br&gt;Makes a new list</td>
<td><strong>Can concatenate</strong>&lt;br&gt;y = x + (1, 2)&lt;br&gt;Makes a new tuple</td>
<td><strong>Can concatenate</strong>&lt;br&gt;y = x + 'World'&lt;br&gt;Makes a new string</td>
</tr>
<tr>
<td><strong>Is mutable</strong></td>
<td><strong>Is not mutable</strong></td>
<td><strong>Is not mutable</strong></td>
</tr>
</tbody>
</table>
# Lists vs. Tuples vs. Strings

<table>
<thead>
<tr>
<th>Lists</th>
<th>Tuples</th>
<th>Strings</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Creation</strong></td>
<td>$x = [a_1, a_2, a_3, \ldots]$</td>
<td>$x = (a_1, a_2, a_3, \ldots)$</td>
</tr>
<tr>
<td>Can contain anything</td>
<td>Can contain anything</td>
<td><strong>Is not mutable</strong></td>
</tr>
<tr>
<td><strong>len(x)</strong> is length</td>
<td><strong>len(x)</strong> is length</td>
<td><strong>len(x)</strong> is length</td>
</tr>
<tr>
<td>Supports slicing</td>
<td>Supports slicing</td>
<td>Supports slicing</td>
</tr>
<tr>
<td><strong>Example</strong>: $x[1:2]$</td>
<td><strong>Example</strong>: $x[1:2]$</td>
<td><strong>Example</strong>: $x[1:2]$</td>
</tr>
<tr>
<td>$x[i]$ is an element</td>
<td>$x[i]$ is an element</td>
<td>$x[i]$ is a substring</td>
</tr>
<tr>
<td><strong>Can concatenate</strong></td>
<td><strong>Can concatenate</strong></td>
<td><strong>Can concatenate</strong></td>
</tr>
<tr>
<td>$y = x + [1, 2]$</td>
<td>$y = x + (1, 2)$</td>
<td>$y = x + \text{'World'}$</td>
</tr>
<tr>
<td>Makes a new list</td>
<td>Makes a new tuple</td>
<td>Makes a new string</td>
</tr>
<tr>
<td><strong>Is mutable</strong></td>
<td><strong>Is not mutable</strong></td>
<td><strong>Is not mutable</strong></td>
</tr>
</tbody>
</table>

Did not use this semester, but work almost like lists do.
Modified Question 4 from Fall 2011

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 '
```python
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 '0'+str(scores[i])+''
        for n in range(scores[i]):    # Loop over number of elements in scores[i]
            row = 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 \( d \): \( \text{len}(d[r]) \)
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 \( m \times m \) 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 `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 `colormodel.PINK`.
- Inner squares have checkerboard pattern of `colormodel.RED` and `colormodel.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:
  ```python
  GRectangle(pos=(0,0),size=(10,10))
  ```
- 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 = []; c = 0  # Make a new list to represent columns
    while c < m:  # Place col c of bricks
        row = []; r = 0  # Make a new list to represent rows
        while r < m:
            color = colormodel.RED
            if r == 0 or r == m-1 or c == 0 or c == m-1:
                color = colormodel.PINK
            elif r+c % 2 == 0:
                color = colormodel.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:

\[
\begin{array}{ccc}
3 & 1 & 2 \\
2 & 1 & 7 & 8 & 5 \\
5 \\
6 & 8 \\
\end{array}
\quad \text{instead of} \quad
\begin{array}{cc}
1 & 2 & 3 \\
1 & 7 & 8 & 5 & 2 \\
5 \\
8 & 6 \\
\end{array}
\]

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.**

```python
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""
```
**Modified Question 4 from Fall 2011**

```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""

    # invariant: rows 0..r−1 of b have been rotated
    r = 0
    while r < len(b):
        first = b[r][0]  # Rotate row r one position to the left;
        c = 1
        while c < len(b[r]):
            b[r][c-1]= b[r][c];
            c= c+1
        # post: b[r][1..] has been moved to b[r][0..]
        b[r][len(b[r])-1]= first;
        r = r+1

    # post: rows 0..b.length−1 of b has been rotated
```