Lecture 26: Lists of Lists
def mystery(data, pos, n):
    result = []
    for i in range(0, n):
        for j in range(0, n):
            result.append(data[i + pos][j + pos])
    return result

Suppose that a variable called grid has been declared as follows:
grid = [[8, 2, 7, 8, 2, 1], [1, 5, 1, 7, 4, 7],
        [5, 9, 6, 7, 3, 2], [7, 8, 7, 7, 7, 9],
        [4, 2, 6, 9, 2, 3], [2, 2, 8, 1, 1, 3]]
which means it will store the following 6-by-6 grid of values:

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>2</td>
<td>7</td>
<td>8</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>1</td>
<td>7</td>
<td>4</td>
<td>7</td>
</tr>
<tr>
<td>5</td>
<td>9</td>
<td>6</td>
<td>7</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>9</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>6</td>
<td>9</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>8</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
</tbody>
</table>

For each call at right, indicate what value is returned. If the function call results in an error, write error instead.

<table>
<thead>
<tr>
<th>Function Call</th>
<th>Contents of List Returned</th>
</tr>
</thead>
<tbody>
<tr>
<td>mystery(grid, 2, 2)</td>
<td>__________________________</td>
</tr>
<tr>
<td>mystery(grid, 0, 2)</td>
<td>__________________________</td>
</tr>
<tr>
<td>mystery(grid, 3, 3)</td>
<td>__________________________</td>
</tr>
</tbody>
</table>
Find Lucky 7

Write a function \( \text{lucky}(m) \) that takes a rectangular list \( m \) and looks for the number 7 in \( m \). If found, \( \text{lucky} \) returns a list containing the row and column position of 7, and if not found, returns an empty list.

Example:
\[
z = [[20, 3, 6], [4, 12, 18], [6, 13, 5], [15, 7, 8]]
\]
\[
\text{lucky}(z)
\]

Returns
\[
[3, 1]
\]
Mountain peak

Write a program that reads elevation data from a file, draws it on a DrawingPanel and finds the path from the highest elevation to the edge of the region.

Data:

34 76 87 9 34 8 22 33 33 33 45 65 43 22
5 7 88 0 56 76 76 77 4 45 55 55 4 5
...

## Mountain peak

Consider the data:

<p>| | | | | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>34</td>
<td>76</td>
<td>87</td>
<td>9</td>
<td>34</td>
<td>8</td>
<td>22</td>
<td>33</td>
<td>33</td>
<td>33</td>
<td>45</td>
<td>65</td>
<td>43</td>
<td>22</td>
</tr>
<tr>
<td>5</td>
<td>7</td>
<td>88</td>
<td>0</td>
<td>56</td>
<td>76</td>
<td>76</td>
<td>77</td>
<td>4</td>
<td>45</td>
<td>55</td>
<td>55</td>
<td>4</td>
<td>5</td>
</tr>
</tbody>
</table>

...  

Each line is a row of elevations ➔ we will create a list of lists of elevations

First steps:

1. create a mapping of the data representation to DrawingPanel components
2. read in the data
3. draw an image of the elevation data using DrawingPanel components
Each row becomes a row of rectangles in the DrawingPanel
Each rectangle is 1 pixel wide

If the mountain data is stored in a list of lists `data`, how large do we make the DrawingPanel?

```python
p = DrawingPanel( width ??, height ??)
```
Each row becomes a row of rectangles in the DrawingPanel
Each rectangle is 1 pixel wide
The elevation is mapped to a level of black or white, a shade of grey, creating a greyscale image:
  - white is weakest (higher in elevation)
  - black is strongest (lower in elevation)

```
p.canvas.create_rectangle(x, y, x + 1, y + 1,
  outline=map elevation to a shade of grey)
```
Mapping indices to arguments

data =
[ [34, 76, 87, 9, 34, 8, 22, 33, 33, 33, 45, 65, 43, 22]
 [5, 7, 88, 0, 56, 76, 76, 77, 4, 45, 55, 55, 4, 5]...
]

For the first row:

p.canvas.create_rectangle(0, 0, 1, 1, outline = color of 34)
p.canvas.create_rectangle(1, 0, 2, 1, outline = color of 76)
p.canvas.create_rectangle(2, 0, 3, 1, outline = color of 87)
p.canvas.create_rectangle(3, 0, 4, 1, outline = color of 9)
....
Mapping to indices to arguments

data =
[ [34, 76, 87, 9, 34, 8, 22, 33, 33, 33, 45, 65, 43, 22],
  [5, 7, 88, 0, 56, 76, 76, 77, 4, 45, 55, 55, 4, 5]...]

for row in range(0, len(data)):
    for col in range(0, len(data[row])):
        color = get_color(data[row][col])
        p.canvas.create_rectangle(col, row, col + 1, row + 1, outline = color)

p.canvas.create_rectangle(0, 0, 1, 1, outline = color of 34-- data[0][0])
p.canvas.create_rectangle(1, 0, 2, 1, outline = color of 76-- data[0][1])
p.canvas.create_rectangle(2, 0, 3, 1, outline = color of 87-- data[0][2])
p.canvas.create_rectangle(3, 0, 4, 1, outline = color of 9-- data[0][3])
...

...
2) Read in the data

```python
from drawingpanel import *
from random import *

def main():
    file = open("mountaindata.dat")
    lines = file.readlines()

    data = []
    for line in lines:
        data.append(line.split())
    p = DrawingPanel(len(data[0]), len(data))
    draw_image(p, data)
```
3) Draw the elevation image

# draws the passed in data on the passed in drawing panel.
# The data is a list of lists of numbers representing
# elevation data.

def draw_image(p, data):
    for row in range(0, len(data)):
        # data[row] -> [3, 5, 76, 3]
        for col in range(0, len(data[row])):
            color = get_color(int(data[row][col]))
            p.canvas.create_rectangle(col, row, col + 1, row + 1, outline=color)
Mountain peak

data =
[ [34, 76, 87, 9, 34, 8, 22, 33, 33, 33, 45, 65, 43, 22]
  [5, 7, 88, 0, 56, 76, 76, 77, 4, 45, 55, 55, 4, 5]
  ...
]

Next steps:
4) Find the peak
5) Find the steepest path down
6) Draw the path in yellow
4) Find the peak

data =
[ [34, 76, 87, 9, 34, 8, 22, 33, 33, 33, 45, 65, 43, 22]
  [5, 7, 88, 0, 56, 76, 76, 77, 4, 45, 55, 55, 4, 5]
  ...
]

Find the largest elevation in the list of lists. Write `find_peak(data)`

Return a tuple of the location in the 2d list
5) Find the steepest path

```python
data =
[['2537', '2483', '2475', '2480', '2518', '2532', '2480', '2478', '2431'],
 ['2541', '2549', '2614', '2700', '2647', '2746', '2690', '2621', '2550'],
 ['2525', '2525', '2640', '2769', '2802', '2883', '2856', '2694', '2631'],
 ['2514', '2505', '2526', '2614', '2717', '2715', '2867', '2836', '2771'],
 ['2506', '2482', '2480', '2528', '2518', '2561', '2586', '2662', '2654'],
 ['2527', '2477', '2464', '2459', '2452', '2475', '2480', '2500', '2518'],
 ['2544', '2505', '2488', '2454', '2442', '2445', '2446', '2467', '2470'],
 ['2528', '2486', '2464', '2446', '2434', '2436', '2442', '2444', '2450'],
 ['2464', '2505', '2482', '2456', '2433', '2463', '2462', '2489', '2467'],
 ['2532', '2541', '2519', '2515', '2496', '2502', '2529', '2519', '2553']]
```

How do we determine the steepest path?

We would need to compare the peak to each neighbor.
5) Find the steepest path down

\[
\text{data} = \\
[[2537, 2483, 2475, 2480, 2518, 2532, 2480, 2478, 2431] \\
[2541, 2549, 2614, 2700, 2647, 2746, 2690, 2621, 2550] \\
[2525, 2525, 2640, 2769, 2802, 2883, 2856, 2694, 2631] \\
[2514, 2505, 2526, 2614, 2717, 2715, 2867, 2836, 2771] \\
[2506, 2482, 2480, 2528, 2518, 2561, 2586, 2662, 2654]] \\
... \\
\]

We will simplify this problem. Look at only three neighbors:
  - up
  - down
  - front

If peak is at location data[r][c], define each above.
5) Find the steepest path down

data =
[["2537", "2483", "2475", "2480", "2518", "2532", "2480", "2478", "2431"]
['2541', '2549', '2614', '2700', '2647', '2746', '2690', '2621', '2550']
['2525', '2525', '2640', '2769', '2802', '2883', '2856', '2694', '2631']
['2514', '2505', '2526', '2614', '2717', '2715', '2867', '2836', '2771']
...]

Compare and find the smallest of the three to create the next path element.

What happens if there are ties?
5) Find the steepest path down

Rules for ties.
If up == down but < front, choose randomly between them.
  up = 2550
  down = 2550
  front = 2690
If front ties with up or down, choose front.
  up = 2690           up = 2550
  down = 2550         down = 2690
  front = 2550        front = 2550
5) Pseudocode for find_path

initialize current location \( \leftarrow \) (this is both a row and column)
make an empty list for path
while location is still within the list bounds
   assign up, front and down
   if (up < down and down < front)
      append up location to path
   else if (down < up and down < front)
      append down location to path
   else if (down == up and up < front)
      chose randomly between down and up
      append one of them to path
   else
      append front location to path
   update current location based on the chosen next location for path
return path
6) Pseudocode for draw_path

For each tuple in the path

   Using the column and row given in the tuple, draw
   a rectangle that is one pixel wide and filled in with yellow