CSc 110, Spring 2017

Lecture 25: Lists of Lists
Lists of lists

• List definition
  \[ \text{value, value, ... value} \]

• The \textbf{value} can be of type list:
  \[
  \text{list} = [[1, 2, 3], [4, 5, 6, 7]]
  \]

How can you access 2?

\[
\text{list[0][1]}
\]

How can you find the length of the second inner list ([4, 5, 6])?

\[
\text{len(list[1])}
\]
List of tuples vs. List of lists

List of tuples:

```python
>>> all_months = [('january', 31), ('february', 28), ('march', 31),
                 ('april', 30), ('may', 31), ('june', 30),
                 ('july', 31), ('august', 31), ('september', 30),
                 ('october', 31), ('november', 30), ('december', 31)]
```

List of lists:

```python
>>> all_months = [['january', 31], ['february', 28], ['march', 31],
                 ['april', 30], ['may', 31], ['june', 30],
                 ['july', 31], ['august', 31], ['september', 30],
                 ['october', 31], ['november', 30], ['december', 31]]
```
List of lists

- List of lists:

```python
>>> all_months = [['january', 31], ['february', 28], ['march', 31], ['april', 30], ['may', 31], ['june', 30], ['july', 31], ['august', 31], ['september', 30], ['october', 31], ['november', 30], ['december', 31]]
```

Print the number of days in each month of `all_months`:

```python
>>> for j in range(0, len(all_months)):
    print(all_months[j][1])
```
Creating lists of lists

- list = [[0] * 4] * 5 will NOT create a list of 5 different lists
  This will create a list of lists that all reference the SAME 4 element list.


```python
>>> list = [[0] * 4] * 5
>>> list
[[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0]]
>>> list[0][1] = 88
>>> list
[[0, 88, 0, 0], [0, 88, 0, 0], [0, 88, 0, 0], [0, 88, 0, 0], [0, 88, 0, 0]]
```
Creating lists of lists

• Instead, write this:

```
list = [[0] * 4, [0] * 4, [0] * 4, [0] * 4, [0] * 4]
```

• Or this:

```
list = []
for i in range(0, 5):
    list.append([0] * 4)
```
Lists of lists

- A 2-dimentional list is rectangular if each row has the same length:
  \[
  rlist = [[2, 4], [20, 40], [100, 400], [2000, 4000]]
  \]

- Refer to a rectangular list's size using the number of rows and columns.

- The list above is a 4 x 2 rectangular list
Rectangular lists

Example of a 4 x 3 list:

```python
>>> rlist = [[0] * 3, [0] * 3, [0] * 3, [0] * 3]
>>> rlist
[[0, 0, 0], [0, 0, 0], [0, 0, 0], [0, 0, 0]]
```

Conceptually, we think of this as a matrix or grid:

```
0 0 0
0 0 0
0 0 0
0 0 0
```
Rectangular lists

Assign the second column of each row to 7:

```python
>>> rlist
[[0, 0, 0], [0, 0, 0], [0, 0, 0], [0, 0, 0]]
```

The grid now looks like this:

```
0 7 0
0 7 0
0 7 0
0 7 0
0 7 0
```
Rectangular lists

Write a function \( \text{mtable}(n) \) that takes an integer as a parameter and returns a rectangular list that is a multiplication table of size \( nxn \).

The call \( \text{mtable}(3) \) would produce the grid on the right.

Express each value in terms of row and column index:

```python
def mtable(n):
    t = []
    for r in range(0, n):
        row = [0] * n
        for c in range(0, n):
            row[c] = (r + 1) * (c + 1)
        t.append(row)
    return t
```

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>6</td>
<td>9</td>
</tr>
</tbody>
</table>
Greatest column sum

Write a function `greatest_column_sum(m)` that takes a rectangular list, finds the column with the greatest sum, and returns a list with the column number and the sum

Example:

```python
n = [[10, 3, 7], [4, 12, 18], [6, 13, 5], [15, 2, 8]]
greatest_column_sum(n)
```

Returns

`[2, 38]`
Greatest column sum

Data grid:

<table>
<thead>
<tr>
<th></th>
<th>10</th>
<th>3</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>12</td>
<td>18</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>13</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>2</td>
<td>8</td>
<td></td>
</tr>
</tbody>
</table>

What drives the outer loop, row or column?
Greatest column sum

# Finds the column with the greatest sum in a 2-d list.
# Returns a list of the corresponding column number and the sum.
def greatest_column_sum(m):

gr_sum = 0
gr_col = 0

for col in range(0, len(m[0])):
    sum = 0
    for row in range(0, len(m)):
        sum += m[row][col]
    if (sum > gr_sum):
        gr_sum = sum
        gr_col = col
return [gr_col, gr_sum]
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:

```
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
...
```

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
1) Create a mapping of the data representation to the DrawingPanel object:

```
[ [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],
  ...]
```

Each elevation will be represented as a pixel-wide rectangle in the DrawingPanel object.
The rectangle will be filled in by a color computed from the elevation.
```
p.canvas.create_rectangle(x, y, x + 1, y + 1,
                         outline = map elevation to a color)
```
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 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

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)
Suppose that a variable called grid has been declared as follows:

\[
\begin{array}{cccccc}
8 & 2 & 7 & 8 & 2 & 1 \\
1 & 5 & 1 & 7 & 4 & 7 \\
5 & 9 & 6 & 7 & 3 & 2 \\
7 & 8 & 7 & 7 & 7 & 9 \\
4 & 2 & 6 & 9 & 2 & 3 \\
2 & 2 & 8 & 1 & 1 & 3 \\
\end{array}
\]

which means it will store the following 6-by-6 grid of values:

Suppose that a variable called grid has been declared as follows:

\[
\begin{array}{cccccc}
8 & 2 & 7 & 8 & 2 & 1 \\
1 & 5 & 1 & 7 & 4 & 7 \\
5 & 9 & 6 & 7 & 3 & 2 \\
7 & 8 & 7 & 7 & 7 & 9 \\
4 & 2 & 6 & 9 & 2 & 3 \\
2 & 2 & 8 & 1 & 1 & 3 \\
\end{array}
\]

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>