How Multidimensional Lists are Stored

- \( \text{sum}(x) \) adds up all the elements in the list \( x \)
  * They must all be numbers!
- \( \text{min}(x) \) or \( \text{max}(x) \) find the min/max value in list \( x \)
  * They use the same ordering as \( \text{sort()} \)
- \( \text{range}(a,b,c) \) produces \([a,a+c,a+2c,\ldots,a+(b-a)/c]\)
  * Starts at \( a \), increases by \( c \) each time, until \( b \) (or less)
  * The argument \( c \) is optional; \( c = 1 \) by default
- \( \text{list}(x) \) converts \( x \) (such as a string) to a list
  * Example: \( \text{list}(\text{"minsy"}) \) produces \([\text{"m"}, \text{"i"}, \text{"n"}, \text{"s"}, \text{"y"}]\)

Two Dimensional Lists

<table>
<thead>
<tr>
<th>Table of Data</th>
<th>Images</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 2 3</td>
<td>0 1 2 3</td>
</tr>
<tr>
<td>0 5 4 7 3</td>
<td>0 5 4 7 3</td>
</tr>
<tr>
<td>1 4 8 9 7</td>
<td>1 4 8 9 7</td>
</tr>
<tr>
<td>2 5 1 2</td>
<td>2 5 1 2</td>
</tr>
<tr>
<td>3 4 1 2 9</td>
<td>3 4 1 2 9</td>
</tr>
<tr>
<td>4 6 7 8 0</td>
<td>4 6 7 8 0</td>
</tr>
</tbody>
</table>

Each row, col has a value

Store them as lists of lists (row-major order)

\( d = [(5,4,7,3),(4,8,9,7),(5,1,2,3),(4,1,2,9),(6,7,8,0)] \)

The Map Function

- \( \text{map}(\text{function}, \text{list}) \)
  * Function has to have exactly 1 parameter
  * Otherwise, get an error
  * Returns a new list
- Does the same thing as
  \( \text{def map(f,x):} \)
  \( \text{result} = [] \) # empty list
  \( \text{for } y \text{ in } x: \)
  \( \text{result.append(f(y))} \)
  \( \text{return result} \)

Overview of Two-Dimensional Lists

- Access value at row 3, col 2:
  \( d[3][2] \)
  \( \begin{bmatrix} 0 & 4 & 7 & 3 \\ 1 & 4 & 8 & 9 \\ 2 & 5 & 1 & 2 \\ 3 & 4 & 1 & 2 \\ 4 & 6 & 7 & 8 \\ \end{bmatrix} \)
- 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

- \( b = [(9, 6, 4), [6, 7, 7]] \)
- \( b \) holds name of a one-dimensional list
  * Has \( \text{len}(b) \) elements
  * Its elements are (the names of) 1D lists
- \( b[1] \) holds the name of a one-dimensional list (of ints)
  * Has \( \text{len}(b[1]) \) elements

Image Data: 2D Lists of Pixels

- \( d[0][0] \) is a white pixel
- \( d[1][0] \) is a red pixel
- \( d[2][0] \) is a green pixel
- \( d[3][0] \) is a blue pixel
- Each row, col has an RGB value
- Each row, col has an RGB value
Slices and Multidimensional Lists
• Only “top-level” list is copied.
• Contents of the list are not altered
  \[ b = [ [9, 6], [4, 5], [7, 7]] \]

Functions and 2D Lists
```python
def transpose(table):
    """Returns copy of table with rows and columns swapped
    Precondition: table is a (non-ragged) 2d List"
    numrows = len(table)
    result = [[]] # Result accumulator
    for m in range(numcols):
        row = [] # Single row accumulator
        for n in range(numrows):
            row.append(table[n][m]) # Build up row
        result.append(row) # Add result to table
    return result
```

Dictionaries (Type dict)
Description
• List of key-value pairs
  • Keys are unique
  • Values need not be
• Example: net-ids
  • net-ids are unique (a key)
  • names need not be (values)
  • js1 is John Smith (class '13)
  • js2 is John Smith (class '16)
• Many other applications

Python Syntax
• Create with format: \{k1:v1, k2:v2, …\}
  • Keys must be non-mutable
    * ints, floats, bools, strings
  • Not lists or custom objects
• Values can be anything
  • Example:
    \[ d = \{'js1':John Smith, 'js2':John Smith, 'wmw2':Walker White'\} \]

Using Dictionaries (Type dict)
• Access els. like a list
  • \[ d['js1']\] evaluates to 'John'
  • But cannot slice ranges!
• Dictionaries are mutable
  • Can reassign values
    \[ d['js1'] = 'Jane' \]
  • Can add new keys
    \[ d['aa1'] = 'Allen' \]
  • Can delete keys
    \[ del d['wmw2'] \]

Using Dictionaries and For-Loops
• Dictionaries != sequences
  • Cannot slice them
• Different inside for loop
  • Loop variable gets the key
  • Then use key to get value
• Has methods to convert dictionary to a sequence
  • Seq of keys: \[ d.keys() \]
  • Seq of values: \[ d.values() \]
  • key-value pairs: \[ d.items() \]