**Nested Lists**

- Lists can hold any objects
- Lists are objects
- Therefore lists can hold other lists!

\[ a = [2, 1] \]
\[ b = [3, 1] \]
\[ c = [1, 4, b] \]
\[ x = [1, a, c, 8] \]

\[ x[0] \]
\[ x[1][1] \]
\[ x[2][2][1] \]
\[ x[2][0] \]
\[ x[1] \]
\[ x[2] \]
\[ x[2][2] \]

**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></td>
</tr>
<tr>
<td>0 5 4 7 3</td>
<td></td>
</tr>
<tr>
<td>1 4 8 9 7</td>
<td></td>
</tr>
<tr>
<td>2 5 1 2 3</td>
<td></td>
</tr>
<tr>
<td>3 4 1 2 9</td>
<td></td>
</tr>
<tr>
<td>4 6 7 8 0</td>
<td></td>
</tr>
</tbody>
</table>

Each row, col has a value
Each row, col has an RGB value

Store them as lists of lists (row-major order)
\[ d = [[5, 4, 7, 3], [4, 8, 9, 7], [5, 1, 2, 3], [4, 1, 2, 9], [6, 7, 8, 0]] \]

**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

**Slices and Multidimensional Lists**

- Only “top-level” list is copied.
- Contents of the list are not altered
- \( x = b[2] \)
- \( b = [[9, 6], [4, 5], [7, 7]] \)
- \( x \)
def transpose(table):
    #"""Returns: copy of table with rows and columns swapped
    Precondition: table is a (non-ragged) 2d List"""
    numrows = len(table)  # Need number of rows
    numcols = len(table[0])  # All rows have same no. cols
    result = []  # Result (new table) accumulator
    for m in range(numcols):
        # Get the column elements at position m
        # Add this row to accumulator table
        row.append(table[n][m])  # Create a new row list
    return result

Dictionaries (Type dict)

<table>
<thead>
<tr>
<th>Description</th>
<th>Python Syntax</th>
</tr>
</thead>
</table>

- 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

Using Dictionaries (Type dict)

- Access elts. like a list
  - d[j1] evaluates to ’John’
  - But cannot slice ranges!
- Dictionaries are mutable
  - Can reassign values
  - d[j1] = ’Jane’
  - Can add new keys
  - d[aa1] = ’Allen’
  - Can delete keys
  - del d[’wmw2’]

Using Dictionaries (Type dict)

d = {’js1’:’John’, ’js2’:’John’, ’wmw2’:’Walker’}

d = {}  # Construct empty dict

d[’js1’] = ”John”  # Add key

d[’js2’] = ”John”  # Add key

d[’js1’] = ”Jane”  # Reassign value

d[’js2’] = ”Jane”

d[’js1’] = ”Jane”

d[’js2’] = ”Jane”

d.pop(’js1’)  # Delete key

d.popitem()  # Delete key and value pair

deletes both

def dict():
    return d

def keys():
    return d.keys()

def values():
    return d.values()

Dictionaries and For-Loops

- Dictionaries != sequences
  - Cannot slice them
- Different inside for loop
  - Loop variable gets the key
  - Then use key to get value
- Can extract iterators with dictionary methods
  - Key iterator: d.keys()
  - Value iterator: d.values()
  - key-value pairs: d.items()