Lecture 14

Nested Lists and Dictionaries
Announcements for This Lecture

Readings
- Today: Chapter 11
- Next Week: Sec. 5.8-5.10

Assignments
- A3 is due today
  - Survey is posted in CMS
  - Late penalty 10%/day
- Opportunities for help
  - Consultants 4:30-9:30
  - Will be on Piazza until Mid.
- No lab next week
  - Tuesday part of Fall Break
  - No special lab for Wed

Prelim, Oct 12th 7:30-9:00
- Material up to TUESDAY
- Study guide is posted

Review session Wednesday
- Still checking place/time
- Announcement on Piazza
Lists of Objects

- List positions are variables
  - Can store base types
  - But cannot store folders
  - Can store folder identifiers
- Folders linking to folders
  - Top folder for the list
  - Other folders for contents
- Example:
  ```python
  >>> r = cornell.RED
  >>> b = cornell.BLUE
  >>> g = cornell.GREEN
  >>> x = [r, b, g]
  ```
Lists of Objects

• List positions are variables
  ▪ Can store base types
  ▪ But cannot store folders
  ▪ Can store folder identifiers

• Folders linking to folders
  ▪ Top folder for the list
  ▪ Other folders for contents

• Example:
  >>> r = cornell.RED
  >>> b = cornell.BLUE
  >>> g = cornell.GREEN
  >>> x = [r, b, g]

RGB
red
255
green
0
blue
0

RGB
red
0
green
255
blue
255

id10
id11
id12
list
x[0]
x[1]
x[2]
Nested Lists

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

\[ x = [1, [2, 1], [1, 4, [3, 1]], 5] \]

\[
\begin{align*}
  a &= [2, 1] \\
  b &= [3, 1] \\
  c &= [1, 4, b] \\
  x &= [1, a, c, 5]
\end{align*}
\]
## Two Dimensional Lists

### Table of Data

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

Each row, col has a value

### Images

Each row, col has an RGB value

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

\[ d = \begin{bmatrix} [5,4,7,3],[4,8,9,7],[5,1,2,3],[4,1,2,9],[6,7,8,0] \end{bmatrix} \]
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

- \( b = \begin{bmatrix} [9, 6, 4], [5, 7, 7] \end{bmatrix} \)

- \( b \) 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
Image Data: 2D Lists of Pixels

b[0][0] is a white pixel
Ragged Lists: Rows w/ Different Length

• $b = [[17,13,19],[28,95]]$

• Will see applications of this later
Slices and Multidimensional Lists

- Only “top-level” list is copied.
- Contents of the list are not altered
- \( b = [[9, 6], [4, 5], [7, 7]] \)

\[
x = b[:2]
\]
Slices and Multidimensional Lists

- Only “top-level” list is copied.
- Contents of the list are not altered
- \( b = [[9, 6], [4, 5], [7, 7]] \)

\[ x = b[:2] \]
Slices and Multidimensional Lists

- Create a nested list
  ```python
  >>> b = [[9,6],[4,5],[7,7]]
  ```

- Get a slice
  ```python
  >>> x = b[:2]
  ```

- Append to a row of x
  ```python
  >>> x[1].append(10)
  ```

- x now has nested list
  ```
  [[9, 6], [4, 5, 10]]
  ```

- What are the contents of the list (with name) in b?

  A: `[[9,6],[4,5],[7,7]]`
  B: `[[9,6],[4,5,10]]`
  C: `[[9,6],[4,5,10],[7,7]]`
  D: `[[9,6],[4,10],[7,7]]`
  E: I don’t know
Slices and Multidimensional Lists

• Create a nested list
  >>> b = [[9, 6], [4, 5], [7, 7]]
• Get a slice
  >>> x = b[:2]
• Append to a row of x
  >>> x[1].append(10)
• x now has nested list
  [[9, 6], [4, 5, 10]]

• What are the contents of the list (with name) in b?

A: [[9, 6], [4, 5], [7, 7]]
B: [[9, 6], [4, 5, 10]]
C: [[9, 6], [4, 5, 10], [7, 7]]
D: [[9, 6], [4, 10], [7, 7]]
E: I don’t know
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
        # Make a new list for this column
        # Add this row to accumulator table
    return result
Functions and 2D Lists

```
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):
        row = []  # Single row accumulator
        for n in range(numrows):
            row.append(table[n][m])  # Create a new row list
        result.append(row)  # Add result to table
    return result
```
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):
        row = []  # Single row accumulator
        for n in range(numrows):
            row.append(table[n][m])  # Create a new row list
        result.append(row)  # Add result to table
    return result
# Dictionaries (Type \texttt{dict})

## Description

- List of **key-value** pairs
  - Keys are unique
  - Values need not be
- Example: net-ids
  - net-ids are \textbf{unique} (a key)
  - names need not be (values)
  - \texttt{js1} is John Smith (class '13)
  - \texttt{js2} is John Smith (class '16)
- Many other applications

## Python Syntax

- Create with format: 
  \[
  \{k1:v1, k2:v2, \ldots\}
  \]
- Keys must be non-mutable
  - ints, floats, bools, strings
  - \textbf{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 elts. 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']`

```python
id8
dict
'js1': 'John'
'js2': 'John'
'wmw2': 'Walker'
d = {'js1': 'John', 'js2': 'John', 'wmw2': 'Walker'}
```

Key-Value order in folder is not important
Using Dictionaries (Type `dict`)

- **Access elts. 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['aal'] = 'Allen'`
  - Can delete keys
    - `del d['wmw2']`

```
d = {'js1':'John','js2':'John', 'wmw2':'Walker'}
```

Key-Value order in folder is not important
Using Dictionaries (Type `dict`)

- Access elts. 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['aal'] = 'Allen'`
  - Can delete keys
  - `del d['wmw2']`

\[
d = \{'js1':'John','js2':'John', 'wmw2':'Walker'\}
\]
Using Dictionaries (Type `dict`)

- Access elts. 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']`

```python
d = { 'js1': 'John', 'js2': 'John', 'wmw2': 'Walker' }
```

Deleting key deletes both
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()`

```python
for k in d:
    # Loops over keys
    print(k)  # key
    print(d[k])  # value

# To loop over values only
for v in d.values():
    print(v)  # value
```

See grades.py