Lecture 13

More with Sequences
Announcements for This Lecture

Readings

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

Assignments

• A2 has been graded
  ▪ Pick up in Gates 216
  ▪ **Mean**: 33, **StdDev**: 8
  ▪ Grades explained in Piazza
• A3 is due on **FRIDAY**
  ▪ Turn in before you leave
  ▪ Will post survey today
  ▪ Survey due next week
• A4 posted **after** the exam

Prelim, Oct 16\(^{th}\) 7:30-9:00

• Material up to **TODAY**
• Study guide is posted
• Will Review on Thursday
  ▪ Will cover what is on exam
  ▪ Set up practice problems
Processing Lists: builtins

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

- **map(⟨function⟩, ⟨list⟩)**
  - Function has to have exactly 1 parameter
  - Otherwise, get an error
  - Returns a new list
- Does the same thing as

```python
def map(f, x):
    result = []  # empty list
    for y in x:
        result.append(f(y))
    return result
```

Examples:
- `map(f, x)`
- `map(len, ['a', 'bc', 'defg'])`
  - Returns `[1, 2, 4]`
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 = colormodel.RED
  >>> b = colormodel.BLUE
  >>> g = colormodel.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:
  ```python
  >>> r = colormodel.RED
  >>> b = colormodel.BLUE
  >>> g = colormodel.GREEN
  >>> x = [r, b, g]
  ```
Nested Lists

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

\[
a = [2, 1] \\
b = [3, 1] \\
c = [1, 4, b] \\
x = [1, a, c, 5]
\]
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>

Images

Each row, col has a value

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 \( d \): \( \text{len}(d[r]) \)

\[
\begin{array}{cccc}
0 & 1 & 2 & 3 \\
\hline
0 & 5 & 4 & 7 & 3 \\
1 & 4 & 8 & 9 & 7 \\
2 & 5 & 1 & 2 & 3 \\
3 & 4 & 1 & 2 & 9 \\
4 & 6 & 7 & 8 & 0 \\
\end{array}
\]
How Multidimensional Lists are Stored

• \( b = \left[ [9, 6, 4], [5, 7, 7] \right] \)

• \( b \) 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

id1

id1

list

id2

id3

...

id2

id23

id24

...

RGB

red 255
green 255
blue 255
Ragged Lists: Rows w/ Different Length

- \( b = \begin{bmatrix} 17, 13, 19 \end{bmatrix}, \begin{bmatrix} 28, 95 \end{bmatrix} \)

- Will see applications of this later
Slices and Multidimensional Lists

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

\[
\text{id1} \quad \text{id2} \quad \text{id3} \\
\quad \text{id4} \\
\quad \text{id2} \\
\quad \text{id3} \\
\quad \text{id4} \\
\quad \text{id5}
\]

\[ x = b[:2] \]
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
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
  ```python
  [[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)
    numcols = len(table[0])  # All rows have same no. cols
    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:
  ```python
  {k1:v1, k2:v2, ...}
  ```
- Keys must be non-mutable
  - ints, floats, bools, strings
  - **Not** lists or custom objects
- Values can be anything
- Example:
  ```python
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']`

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

Key-Value order in folder is not important

10/7/14

More Sequences

18
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['aa1'] = 'Allen'
  - Can delete keys
    - del d['wmw2']

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

- Access elts. like a list
  - \texttt{d['js1']} evaluates to 'John'
  - But cannot slice ranges!
- Dictionaries are \textbf{mutable}
  - Can reassign values
  - \texttt{d['js1']} = 'Jane'
  - Can add new keys
  - \texttt{d['aal']} = 'Allen'
  - Can delete keys
  - \texttt{del d['wmw2']}

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

```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
Dictionaries and Lists

- The values can be lists
  - Works just like 2D lists
  - But first index is a key

- **Example:**
  ```python
  >>> d = {}  # Empty dict
  >>> d['wmw2'] = [9, 6]
  >>> d['aa1'] = [4]
  >>> d['aa1'].append(5)
  ```

- **We will use this in A4!**