Lecture 13: Nested Lists, Tuples, and Dictionaries
(Sections 11.1-11.5, 12.1-12)

CS 1110
Introduction to Computing Using Python

[E. Andersen, A. Bracy, D. Gries, L. Lee, S. Marschner, C. Van Loan, W. White]
Nested Lists

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

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

\[
b = [3, 1]
\]

\[
c = [1, 4, b]
\]

\[
a = [2, 1]
\]

\[
x = [1, a, c, 5]
\]
## 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>

Each row, col has a value

### Images

Each row, col has an RGB value

---

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

\[
d = \begin{bmatrix} 5, 4, 7, 3 \end{bmatrix}, \begin{bmatrix} 4, 8, 9, 7 \end{bmatrix}, \begin{bmatrix} 5, 1, 2, 3 \end{bmatrix}, \begin{bmatrix} 4, 1, 2, 9 \end{bmatrix}, \begin{bmatrix} 6, 7, 8, 0 \end{bmatrix}\]
Overview of Two-Dimensional Lists

<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>
</tbody>
</table>

```python
>>> d = [[5, 4, 7, 3], [4, 8, 9, 7], [5, 1, 2, 3], [4, 1, 2, 9]]
>>> d[3][2] = 8
Access value at row 3, col 2
>>> len(d)
Number of rows of d
4
>>> len(d[2])
Number of cols in row 2 of d
4
>>> d
[[5, 4, 7, 3], [4, 8, 9, 7], [5, 1, 2, 3], [4, 1, 8, 9]]
```
How Multidimensional Lists are Stored

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

- \( b \) holds id of a one-dimensional list
  - Has \( \text{len}(b) \) elements
- \( b[i] \) holds id of a one-dimensional list
  - Has \( \text{len}(b[i]) \) elements
Ragged Lists: Rows w/ Different Length

- $b = [[17, 13, 19], [28, 95]]$
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 & Multidimensional Lists (Q1)

- 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)

What is now in x?

A: [['9', '6', '10']]
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
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)
```

What is now in x?

- A: `[[9,6,10]]`
- 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 & Multidimensional Lists (Q2)

- 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 is now 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
• 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 is now 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
Data Wrangling: Transpose Idea

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

4 lists: 2 elements in each

2 lists: 4 elements in each

How to transpose?

- 1\(^{st}\) element of each list gets appended to 1\(^{st}\) list
- 2\(^{nd}\) element of each list gets appended to 2\(^{nd}\) list
def transpose(table):
    """Returns: copy of table with rows and columns swapped

    Precondition: table is a (non-ragged) 2d List"
    n_rows = len(table)
    n_cols = len(table[0])  # All rows have same no. cols
    new_table = []  # Result accumulator
    for c in range(n_cols):
        row = []  # Single row accumulator
        for r in range(n_rows):
            row.append(table[r][c])  # Build up new row
        new_table.append(row)  # Add new row to new table
    return new_table

d = [[1,2],[3,4],[5,6]]
d_v2 = transpose(d)
Tuples

- Tuples fall between strings and lists
  - write them with just commas: 42, 4.0, 'x'
  - often enclosed in parentheses: (42, 4.0, ‘x’)

Use **lists** for:
- long sequences
- homogeneous sequences
- variable length sequences

Use **tuples** for:
- short sequences
- heterogeneous sequences
- fixed length sequences

* “tuple” generalizes “pair,” “triple,” “quadruple,” …
Returning multiple values

- Can use lists/tuples to **return** multiple values

```
INCHES_PER_FOOT = 12

def to_feet_and_inches(height_in_inches):
    feet = height_in_inches // INCHES_PER_FOOT
    inches = height_in_inches % INCHES_PER_FOOT
    return (feet, inches)

all_inches = 68
(ft, ins) = to_feet_and_inches(all_inches)
print('You are ' + str(ft) + ' feet, ' + str(ins) + ' inches.')
```
## 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)

### Python Syntax
- Create with format: `{key1:value1, key2:value2, ... }`
- Keys must be **immutable**
  - ints, floats, bools, strings
  - **Not** lists or custom objects
- Values can be anything
- **Example:**
  ```python
d = {'ec1':'Ezra Cornell',
      'ec2':'Ezra Cornell',
      'tm55':'Toni Morrison'}
  ```
Using Dictionaries (Type `dict`)

• Can access elements like a list
• Must use the key, not an index
• Cannot slice ranges

```python
>>> d = {'ec1': 'Ezra', 'ec2': 'Ezra', 'tm55': 'Toni'}
>>> d['ec1']
'Ezra'
>>> d[0]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 0
>>> d[:1]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unhashable type: 'slice'
```
Using Dictionaries (Type `dict`)

- Dictionaries are **mutable**
  - Can reassign values
  - `d['ec1'] = 'Ellis'`

```
d = {'ec1':'Ezra','ec2':'Ezra', 'tm55':'Toni'}
```
Using Dictionaries (Type `dict`)

- Dictionaries are **mutable**
  - Can reassign values
  - `d['ec1'] = 'Ellis'`

```
d = {'ec1':'Ezra','ec2':'Ezra',
     'tm55':'Toni'}
```
Using Dictionaries (Type `dict`)

- Dictionaries are **mutable**
  - Can reassign values
    - `d['ec1'] = 'Ellis'`
  - Can add new keys
    - `d['psb26'] = 'Pearl'`
  - `d = {'ec1':'Ezra','ec2':'Ezra', 'tm55':'Toni'}`
Dictionaries are **mutable**
- Can reassign values
  - `d['ec1'] = 'Ellis'`
- Can add new keys
  - `d['psb26'] = 'Pearl'`

```python
d = {'ec1':'Ezra','ec2':'Ezra', 'tm55':'Toni','psb26':'Pearl'}
```
Using Dictionaries (Type `dict`)

- Dictionaries are **mutable**
  - Can reassign values
  - `d['ec1'] = 'Ellis'`
  - Can add new keys
  - `d['psb26'] = 'Pearl'`
  - Can delete keys
  - `del d['tm55']`

```python
id8 = {
    'ec1': 'Ezra',
    'ec2': 'Ezra',
    'tm55': 'Toni',
    'psb26': 'Pearl'
}
```
Using Dictionaries (Type `dict`)

- Dictionaries are **mutable**
  - Can reassign values
    - `d['ec1'] = 'Ellis'`
  - Can add new keys
    - `d['psb26'] = 'Pearl'`
  - Can delete keys
    - `del d['tm55']`

```
d = {'ec1':'Ezra','ec2':'Ezra', 'psb26':'Pearl'}
```

Deleting key deletes both

```
  id8
    dict
      'ec1' ['Ezra', 'Ellis']
      'ec2' ['Ezra']
      'psb26' ['Pearl']
```