Lecture 14: 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!

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

How Multidimensional Lists are Stored

- `b` holds `id` of a one-dimensional list
  - Has `len(b)` elements
- `b[i]` holds `id` of a one-dimensional list
  - Has `len(b[i])` elements

Slices & Multidimensional Lists (Q1)

- Create a nested list
  ```python
  >>> b = [[9,6],[4,8],[7,7]]
  ```
- Get a slice
  ```python
  >>> x = b[2]
  ```
- Append to a row of `x`
  ```python
  >>> x[1].append(10)
  ```

```
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)
  ```

```

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]]
```

Two Dimensional Lists

- Table of Data
- Images
  - Each row, col has a value
  - Each row, col has an RGB value

```python
b = [[9,6],[4,8],[7,7]]
c = [1, 4, b]
a = [2, 1]
x = [1, a, c, 5]
```

```
x[0]
x[1][1]
x[2][2][0]
x[2][0]
x[1]
x[2]
x[2][2]
x[1]
x[2]
x[2][2]
```

Three Dimensional Lists

- Table of Data
- Images
  - Each row, col has a value
  - Each row, col has an RGB value

```python
b = [[9,6],[4,8],[7,7]]
c = [1, 4, b]
a = [2, 1]
x = [1, a, c, 5]
```

```
x[0]
x[1][1]
x[2][2][0]
x[2][0]
x[1]
x[2]
x[2][2]
x[1]
x[2]
x[2][2]
```

Slices & Multidimensional Lists (Q1)

- Create a nested list
  ```python
  >>> b = [[9,6],[4,8],[7,7]]
  ```
- Get a slice
  ```python
  >>> x = b[2]
  ```
- Append to a row of `x`
  ```python
  >>> x[1].append(10)
  ```

```
A: [[9,6],[4,8],[7,7]]
B: [[9,6],[4,8],[10]]
C: [[9,6],[4,10],[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)
  ```

```
A: [[9,6],[4,5],[7,7]]
B: [[9,6],[4,5],[10]]
C: [[9,6],[4,10],[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\textsuperscript{st} element of each list gets appended to 1\textsuperscript{st} list
• 2\textsuperscript{nd} element of each list gets appended to 2\textsuperscript{nd} list

Data Wrangling: Transpose Code

```python
def transpose(orig_table):
    """Returns: copy of table with rows and columns swapped
    Precondition: table is a (non-nragged) 2D list"
    numrows = len(orig_table)
    numcols = len(orig_table[0]) # All rows have same no. cols
    new_table = [] # Result accumulator
    for m in list(range(numcols)):
        row = [] # Single row accumulator
        for n in range(numrows): # Build up new row
            row.append(orig_table[n][m])
        new_table.append(row) # Add new row to new table
    return new_table
```

Tuples

- Strings: immutable sequences of characters
- Lists: mutable sequences of any objects
- Tuples: immutable sequences of any objects

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

Conventionally use lists for:
- long sequences
- homogeneous sequences
- variable length sequences

Conventionally use tuples for:
- short sequences
- heterogeneous sequences
- fixed length sequences

Returning multiple values

- Can use lists/tuples to \textbf{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 \texttt{dict})

<table>
<thead>
<tr>
<th>Description</th>
<th>Python Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>List of key-value pairs</td>
<td>Create with format: {k1:v1, k2:v2, ...}</td>
</tr>
<tr>
<td>Keys are unique</td>
<td>Keys must be \textbf{immutable}</td>
</tr>
<tr>
<td>Values need not be</td>
<td>* ints, floats, bools, strings</td>
</tr>
<tr>
<td>* Not lists or custom objects</td>
<td>Values can be anything</td>
</tr>
<tr>
<td>Example: net-ids are unique (a key)</td>
<td>Example:</td>
</tr>
<tr>
<td>* names need not be (values)</td>
<td>d = {'ec1':'Ezra', 'ec2':'Ezra', 'aa1':'Allen'}</td>
</tr>
<tr>
<td>* js1 is John Smith (class '13)</td>
<td>* del d['aia63']</td>
</tr>
<tr>
<td>* js2 is John Smith (class '16)</td>
<td></td>
</tr>
</tbody>
</table>

Using Dictionaries (Type \texttt{dict})

• Dictionaries are \textbf{mutable}
  - Can reassign values
d["ec1"] = 'Ellis'
  - Can add new keys
d["aa1"] = 'Allen'
  - Can delete keys
del d["aia63"]

Deleting key deletes both