Lecture 12: Nested Lists and Dictionaries  
(Sections 11.1-11.5)

CS 1110  
Introduction to Computing Using Python

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

Announcements

- TA Office Hours: *Come to them!*
  - *Ask any question... or don't even ask a question!*
    - "I don't know what's going on..." ← say this!
  - Attend early!
- A3 is out
  - Show your code only to your partner or course staff
  - Look only at the code of your partner
- Masking policy:
  - Keep wearing masks in non-distanced educational settings: this does mean all in-person office/consulting hours

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:
  ```python
  {key1:value1, key2:value2, ...}
  ```
- Keys must be immutable
  - ints, floats, bools, strings
- Not lists or custom objects
- Values can be anything
- Example:
  ```python
  d = {'js1':'John Smith', 'js2':'John Smith', 'tm55':'Toni Morrison'}
  ```

Using Dictionaries (Type `dict`)

```python
>>> d = {'ec1': 'Ezra', 'ec2': 'Ezra', 'tm55': 'Toni'}
>>> d['ec1']
'Ezra'
```
Dictionaries are mutable (1, after)

1. Can reassign values
   * d['ec1'] = 'Ellis'

2. Can add new keys
   * d['psb26'] = 'Pearl'

3. Can delete keys
   * del d['tm55']

---

Dictionaries are mutable (2, before)

1. Can reassign values
   * d['ec1'] = 'Ellis'

2. Can add new keys
   * d['psb26'] = 'Pearl'

3. Can delete keys
   * del d['tm55']

---

Dictionaries are mutable (2, after)

1. Can reassign values
   * d['ec1'] = 'Ellis'

2. Can add new keys
   * d['psb26'] = 'Pearl'

3. Can delete keys
   * del d['tm55']

---

Dictionaries are mutable (3, before)

1. Can reassign values
   * d['ec1'] = 'Ellis'

2. Can add new keys
   * d['psb26'] = 'Pearl'

3. Can delete keys
   * del d['tm55']

---

Dictionaries are mutable (3, after)

1. Can reassign values
   * d['ec1'] = 'Ellis'

2. Can add new keys
   * d['psb26'] = 'Pearl'

3. Can delete keys
   * del d['tm55']

---

A student asked about iterating through a dict

students = {'ec1':'Ezra','ec2':'Ezra', 'tm55':'Toni'}

# loop variable iterates through each key
for netID in students:
    print(netID + ":" + students[netID])

prints to the screen:
ec1:Ezra
ec2:Ezra
tm55:Toni
Nested Lists

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

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

<table>
<thead>
<tr>
<th>Global Space</th>
<th>Heap</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>id1</td>
</tr>
<tr>
<td>c</td>
<td>id2</td>
</tr>
<tr>
<td>a</td>
<td>id3</td>
</tr>
<tr>
<td>x</td>
<td>id4</td>
</tr>
</tbody>
</table>

This is drawing accurate, but a little hard to reason about...

"Table-shaped" Two-Dimensional Lists

Table of Data

<table>
<thead>
<tr>
<th>E.g., student ID</th>
<th>E.g., lab number</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>5 4 7 3</td>
</tr>
<tr>
<td>1</td>
<td>4 8 9 7</td>
</tr>
<tr>
<td>2</td>
<td>5 1 2 3</td>
</tr>
<tr>
<td>3</td>
<td>4 1 2 9</td>
</tr>
<tr>
<td>4</td>
<td>6 7 8 0</td>
</tr>
</tbody>
</table>

Each row, column of the table stores data (a value). Here, the score of the student with ID 1 on lab 3

"Table-shaped" Two-Dimensional Lists

Table of Data

<table>
<thead>
<tr>
<th>E.g., shop ID</th>
<th>E.g., product ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>5 4 7 3</td>
</tr>
<tr>
<td>1</td>
<td>4 8 9 7</td>
</tr>
<tr>
<td>2</td>
<td>5 1 2 3</td>
</tr>
<tr>
<td>3</td>
<td>4 1 2 9</td>
</tr>
<tr>
<td>4</td>
<td>6 7 8 0</td>
</tr>
</tbody>
</table>

Each row, column of the table stores data (a value). Here, the number of units of product 3 sold by the shop with ID 1

"Table-shaped" Two-Dimensional Lists

Overview of Two-Dimensional Lists

<table>
<thead>
<tr>
<th>Column index</th>
<th>Row index</th>
<th>Really a list of lists, but convenient to think about it as a table, since all inner lists (rows) have the same number of elements.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Column index</td>
<td>Row index</td>
<td>Each row, col has a value. Store them as a list of lists (&quot;row-major order&quot;) tdlist = [[5,4,7,3], [4,8,9,7], [5,1,2,3], [4,1,2,9], [6,7,8,0]]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Column index</th>
<th>Row index</th>
<th>Number of rows of tdlist 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Column index</td>
<td>Row index</td>
<td>Number of cols in row 2 of tdlist 4</td>
</tr>
</tbody>
</table>

Access value at row 3, col 2
Assign value at row 3, col 2
Number of rows of tdlist
How Multidimensional Lists are Stored

- \( b \) holds id of a one-dimensional list
  - Has \( \text{len}(b) \) elements
- \( b[i] \) holds id of a list
  - Has \( \text{len}(b[i]) \) elements

Exercise 1

```python
def print_all_rows(my_table):
    """Prints all rows of the table, one row (list) on each line.
    Preconditions: my_table is a table of numbers
    my_table is not empty """
```

Data Wrangling: Transpose Idea

4 lists: 2 elements in each
How to transpose?
- 1st element of each list gets appended to 1st list
- 2nd element of each list gets appended to 2nd list

Exercise 2

```python
def print_all_elements(my_table):
    """Prints all elements of the table, one element on each line.
    Preconditions: my_table is a table of numbers
    my_table is not empty """
```

Data Wrangling: Transpose Idea

4 lists: 2 elements in each
How to transpose?
- 1st element of each list gets appended to 1st list
- 2nd element of each list gets appended to 2nd 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
    return new_table

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

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

Slices & Multidimensional Lists (Q2)
- 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