CS 1110:
Introduction to Computing Using Python

Lecture 14

Nested Lists and Dictionaries

[Andersen, Gries, Lee, Marschner, Van Loan, White]

Announcements

- Prelim = not Thursday or Friday
- might be on weekend or next week
- Dean of Faculty supports our request to extend the drop deadline (but no official decision yet)

Next week: Recursion

- Tuesday and Thursday: Recursion.
- Reading: 5.8-5.10

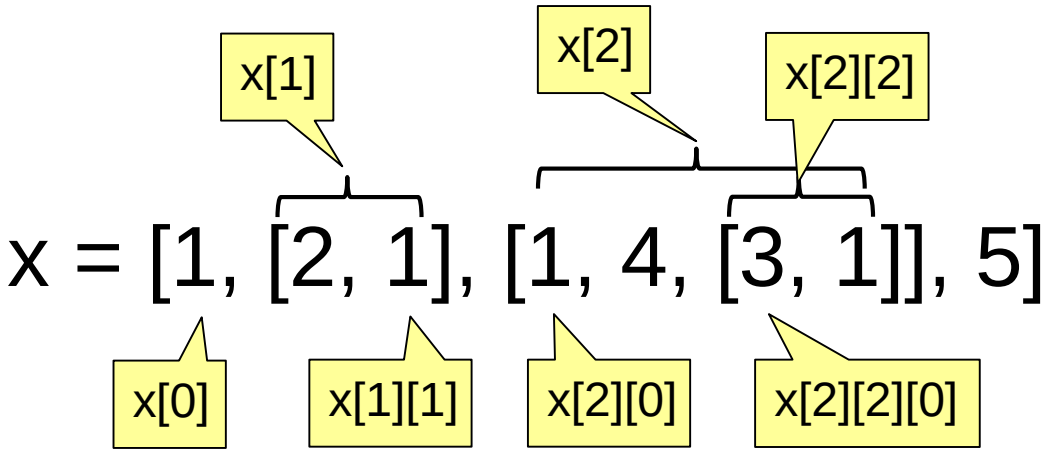
Announcements: Lab 8

- Lab 8 has been released
- Could be useful as extra for loop practice.
- Prelim does not depend on any material introduced in this lab.

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]
```



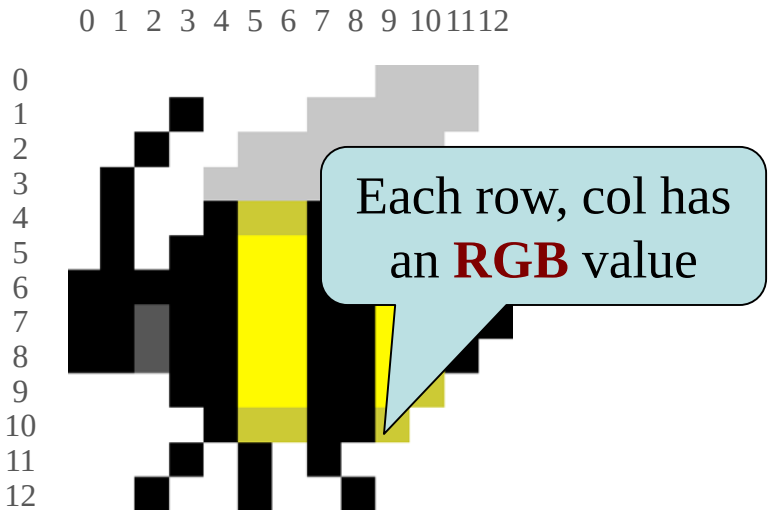
Two Dimensional Lists

Table of Data

	0	1	2	3
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

Each row, col has a value

Images



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]]

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

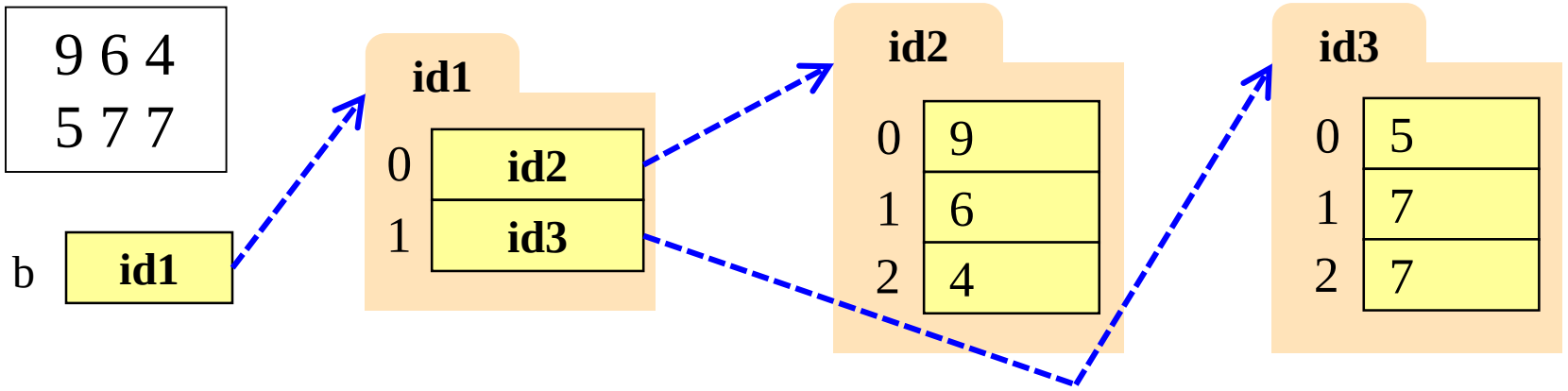
- Number of rows of d:
  - len(d)
- Number of cols in row r of d:
  - len(d[r])

d

	0	1	2	3
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

How Multidimensional Lists are Stored

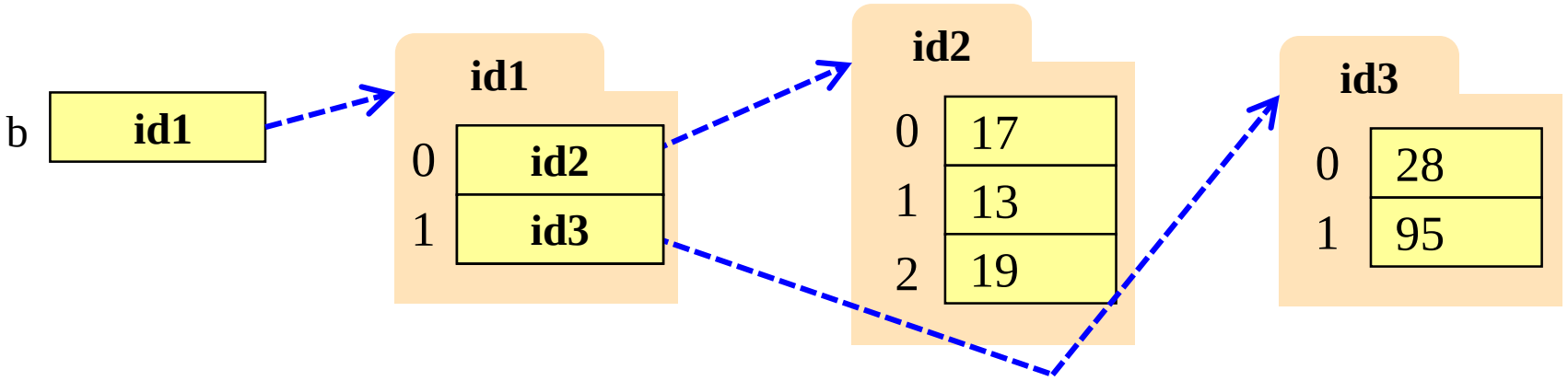
- b = [[9, 6, 4], [5, 7, 7]]



- 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

Ragged Lists: Rows w/ Different Length

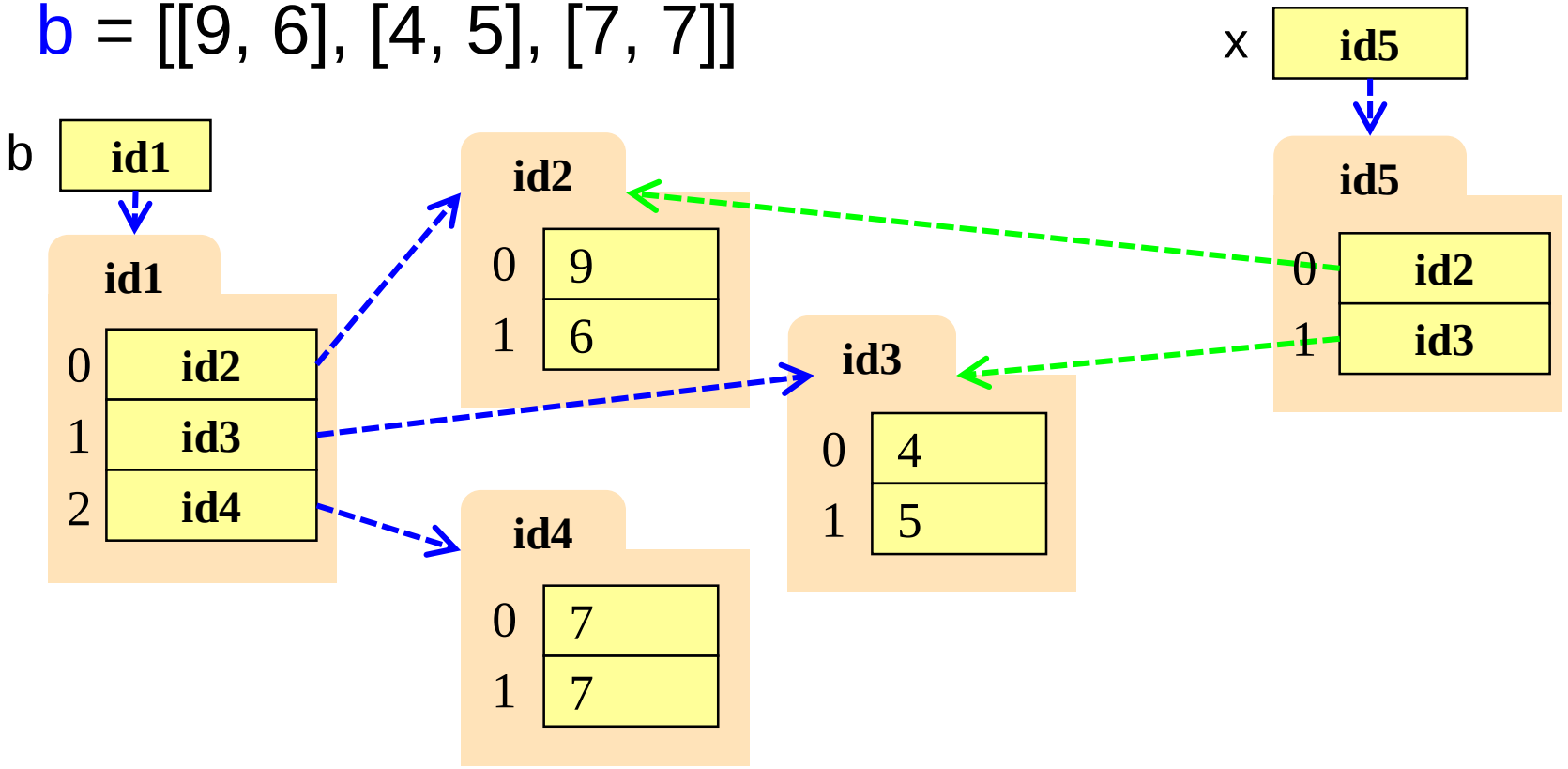
- b = [[17,13,19],[28,95]]



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 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)
```

- 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 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)

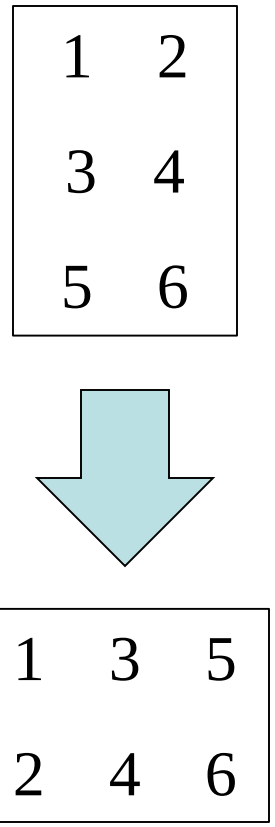
- x now has nested list

  [[9, 6], [4, 5, 10]]

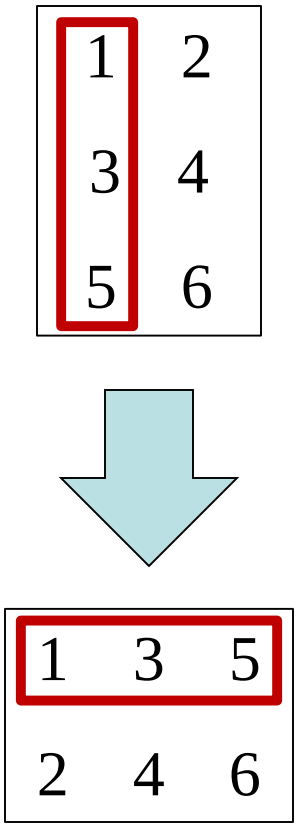
- What is now in b?

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

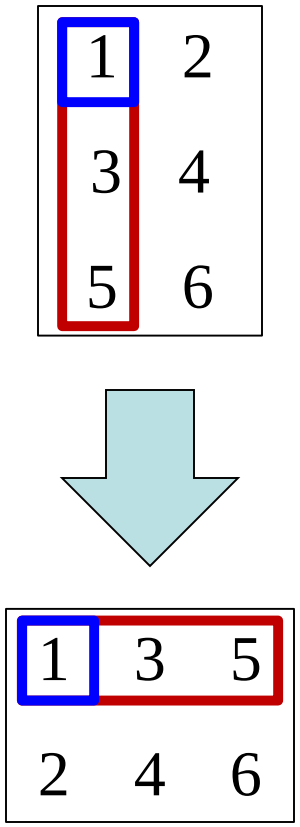
Data Wrangling: Transpose



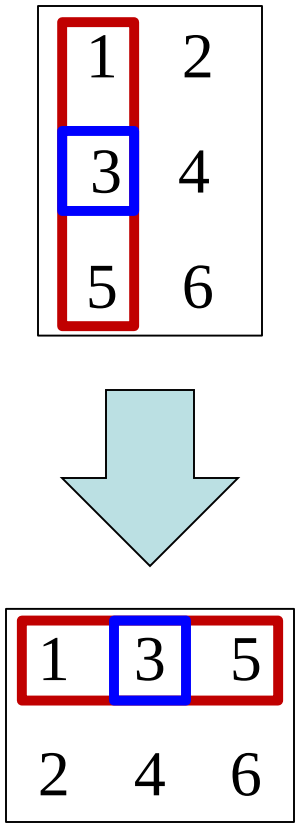
Data Wrangling: Transpose



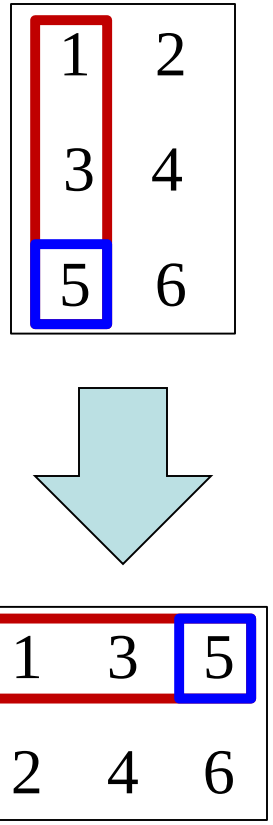
Data Wrangling: Transpose



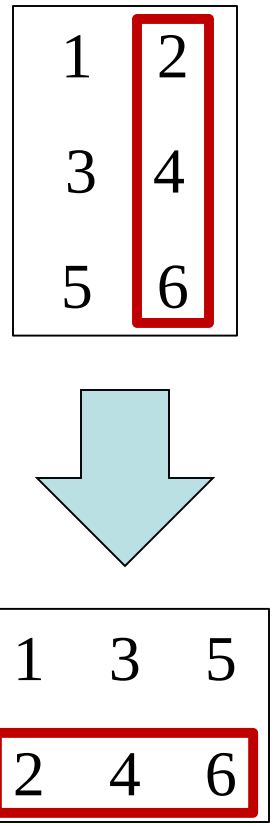
Data Wrangling: Transpose



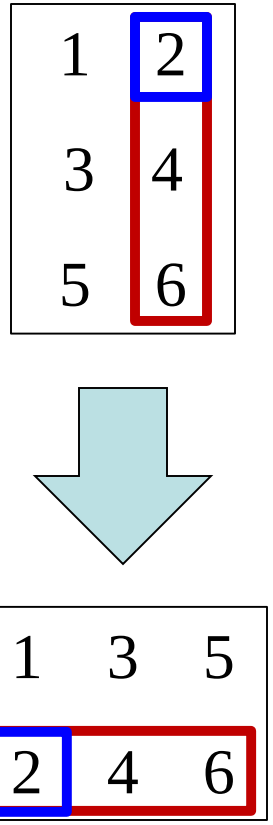
Data Wrangling: Transpose



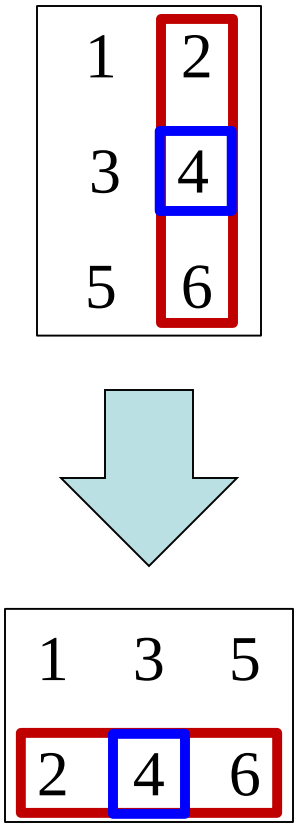
Data Wrangling: Transpose



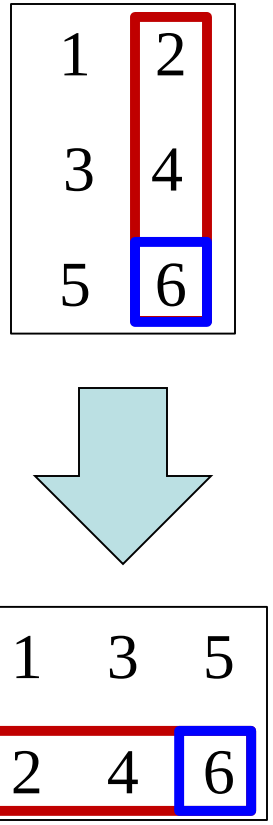
Data Wrangling: Transpose



Data Wrangling: Transpose



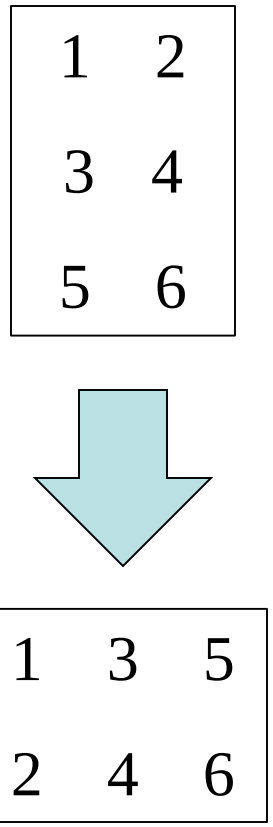
Data Wrangling: Transpose



Data Wrangling: Transpose

```
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
```



Data Wrangling

	0	1	2	3
0		Qual 1	Qual 2	Qual 3
1	Andrew	01.02.03	27.06.08	06.04.07
2	Ben	31.08.01		05.07.04
3	Carl		18.04.03	09.12.09

Andrew	Qual 1	01.02.03
Andrew	Qual 2	27.06.08
Andrew	Qual 3	06.04.07
Ben	Qual 1	31.08.01
Ben	Qual 3	05.07.04
Carl	Qual 2	18.04.03
Carl	Qual 3	09.12.09

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

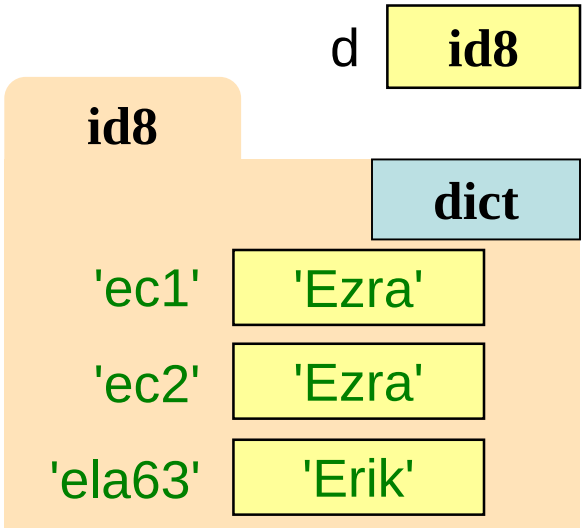
- Create with format:
  {k1:v1, k2:v2, …}
- Keys must be **immutable**
  - ints, floats, bools, strings
  - **Not** lists or custom objects
- Values can be anything
- Example:

```
d = {'ec1':'Ezra Cornell',
     'ec2':'Ezra Cornell',
     'ela63':'Erik Andersen'}
```

Using Dictionaries (Type dict)

- Access elts. like a list
  - d['ec1'] evaluates to 'Ezra'
  - But cannot slice ranges!

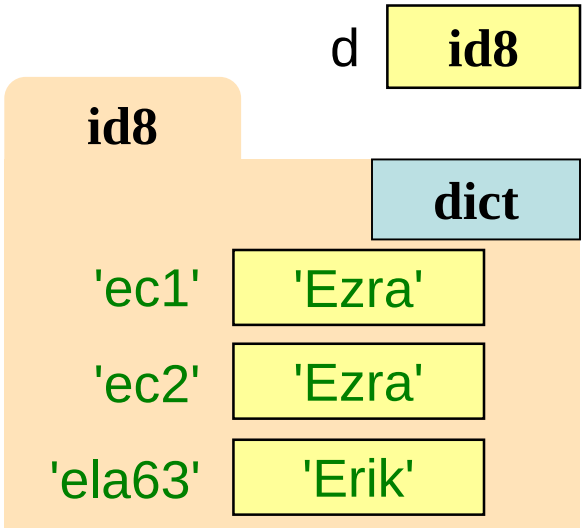
```
d = {'ec1':'Ezra','ec2':'Ezra',
     'ela63':'Erik'}
```



Using Dictionaries (Type dict)

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

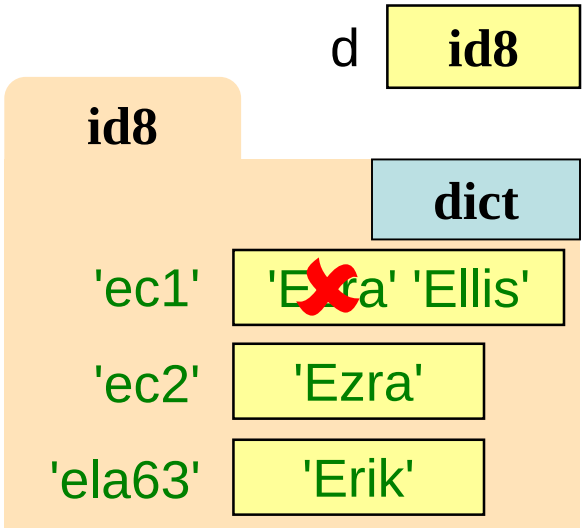
```
d = {'ec1':'Ezra','ec2':'Ezra',
     'ela63':'Erik'}
```



Using Dictionaries (Type dict)

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

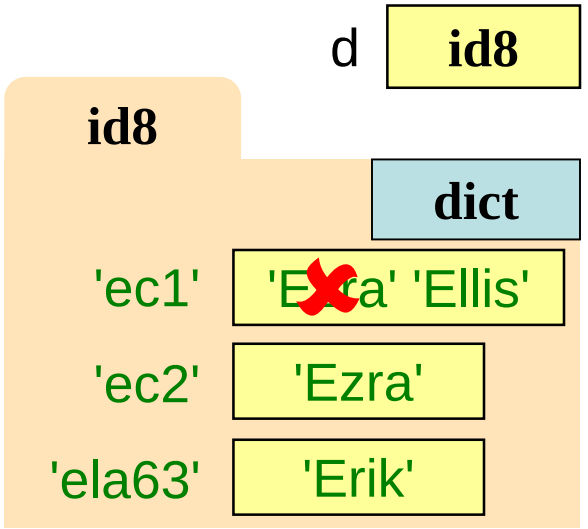
```
d = {'ec1':'Ezra','ec2':'Ezra',
     'ela63':'Erik'}
```



Using Dictionaries (Type dict)

- Dictionaries are **mutable**
  - Can reassign values
  - d['ec1'] = 'Ellis'
  - Can add new keys
  - d['aa1'] = 'Allen'

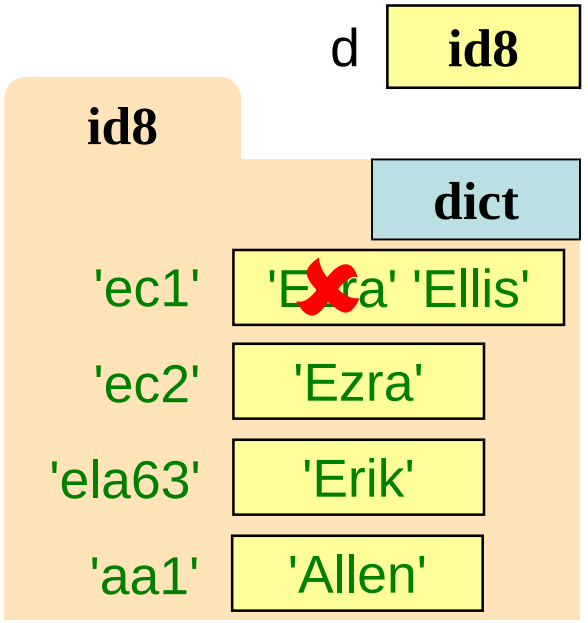
d = {'ec1':'Ezra','ec2':'Ezra',
'ela63':'Erik'}



Using Dictionaries (Type dict)

- Dictionaries are **mutable**
  - Can reassign values
  - d['ec1'] = 'Ellis'
  - Can add new keys
  - d['aa1'] = 'Allen'

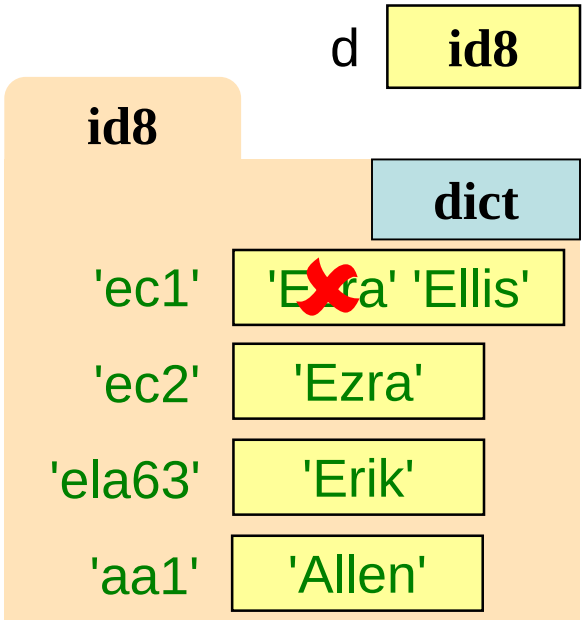
```
d = {'ec1':'Ezra','ec2':'Ezra',
     'ela63':'Erik','aa1':'Allen'}
```



Using Dictionaries (Type dict)

- Dictionaries are **mutable**
  - Can reassign values
  - d['ec1'] = 'Ellis'
  - Can add new keys
  - d['aa1'] = 'Allen'
  - Can delete keys
  - del d['ela63']

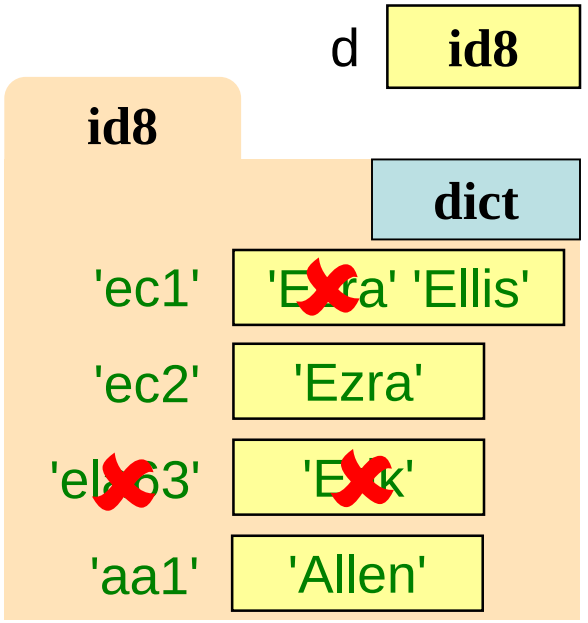
d = {'ec1':'Ezra','ec2':'Ezra',
'ela63':'Erik','aa1':'Allen'}



Using Dictionaries (Type dict)

- Dictionaries are **mutable**
  - Can reassign values
  - d['ec1'] = 'Ellis'
  - Can add new keys
  - d['aa1'] = 'Allen'
  - Can delete keys
  - del d['ela63']

d = {'ec1':'Ezra','ec2':'Ezra',
'aa1':'Allen'}



Deleting key deletes both

New Way to do Test Cases

```
test_cases = {4:3, 5:3, 6:6, -2:0, 20:7}


for item in test_cases:
        seq = orig[:]   # this creates a copy of the list orig
        print "\tTesting input " + str(seq) + ", " + str(item)
        output = lab08.lesser_than(seq, item)
        ct.assert_equals(test_cases[item], output)
        ct.assert_equals(seq, orig) # make sure argument
list wasn't changed
```