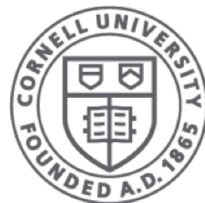
http://www.cs.cornell.edu/courses/cs1110/2018sp

Lecture 14: Nested Lists, Tuples, and Dictionaries

(Sections 11.1-11.5, 12.1-12)

CS 1110

Introduction to Computing Using Python



Cornell CIS
COMPUTING AND INFORMATION SCIENCE

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

Announcements

- A3 Tentative release date: Mon Mar 19-Thu Mar 22; tentative time for completion: somewhere between 1 and 2 weeks. Similar to A3 from Spring 2017.
- Prelim 1 Grading this weekend. Grades will come out before the drop deadline.

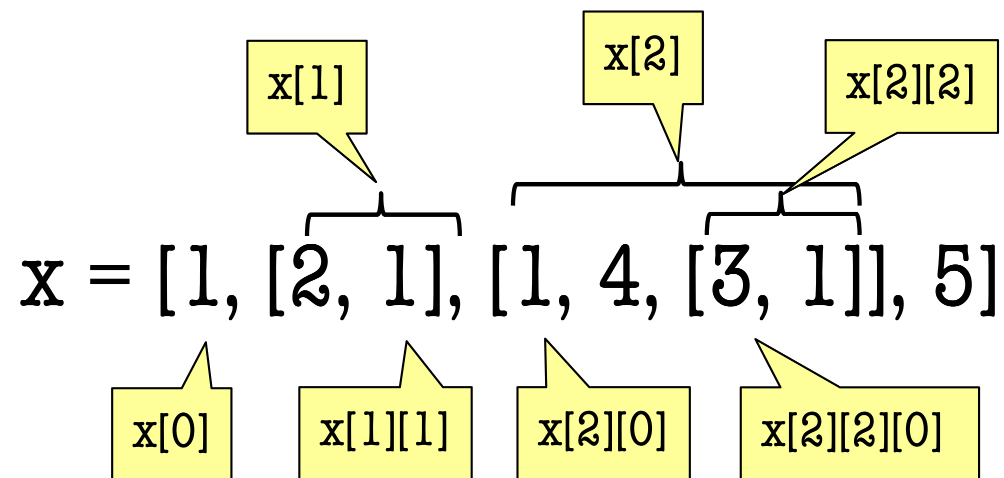
Next week: Recursion

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

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]
```



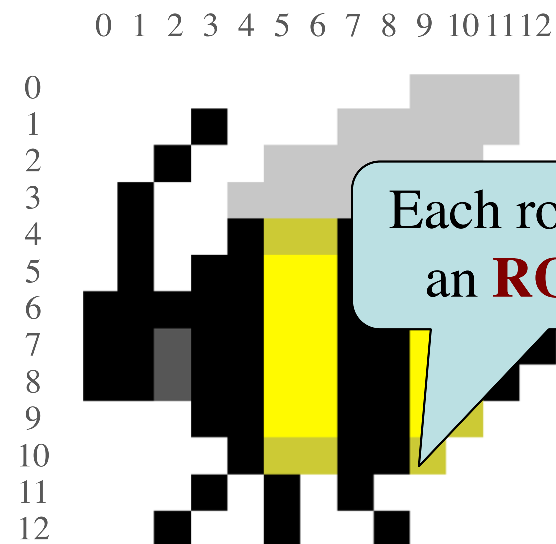
Two Dimensional Lists

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:

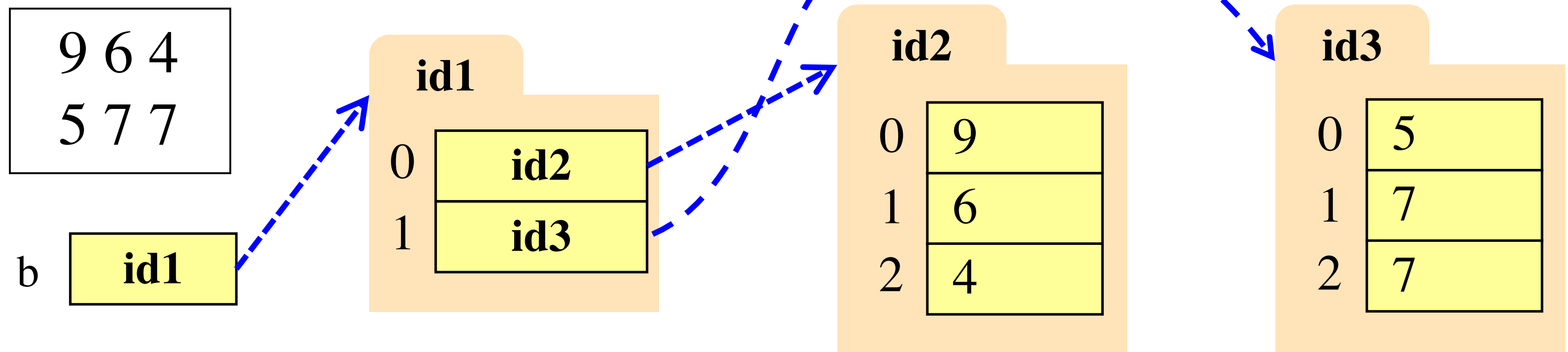
  `d[3][2] = 8`

- 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

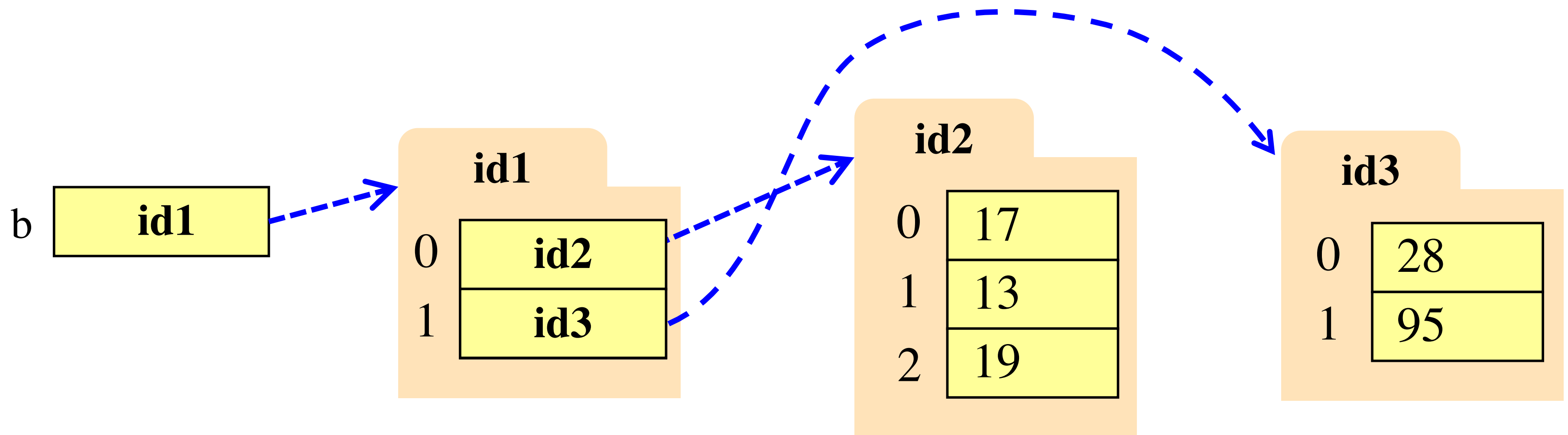
- `b = [[9, 6, 4], [5, 7, 7]]`



- 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]]`

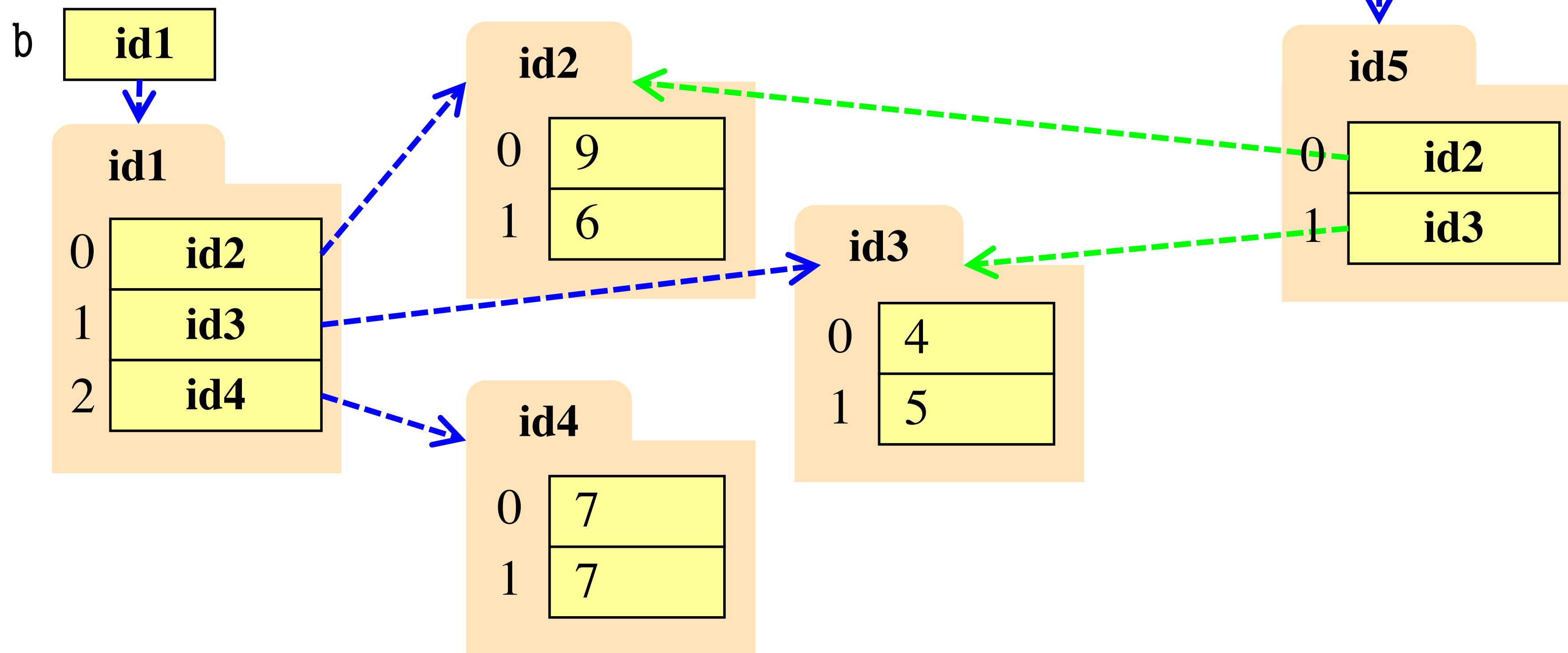


Slices and Multidimensional Lists

- Only "top-level" list is copied.
- Contents of the list are not altered
- `b = [[9, 6], [4, 5], [7, 7]]`

`x = b[:2]`

x id5



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 (A1)

- 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

Slices & Multidimensional Lists (A2)

- 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)
```

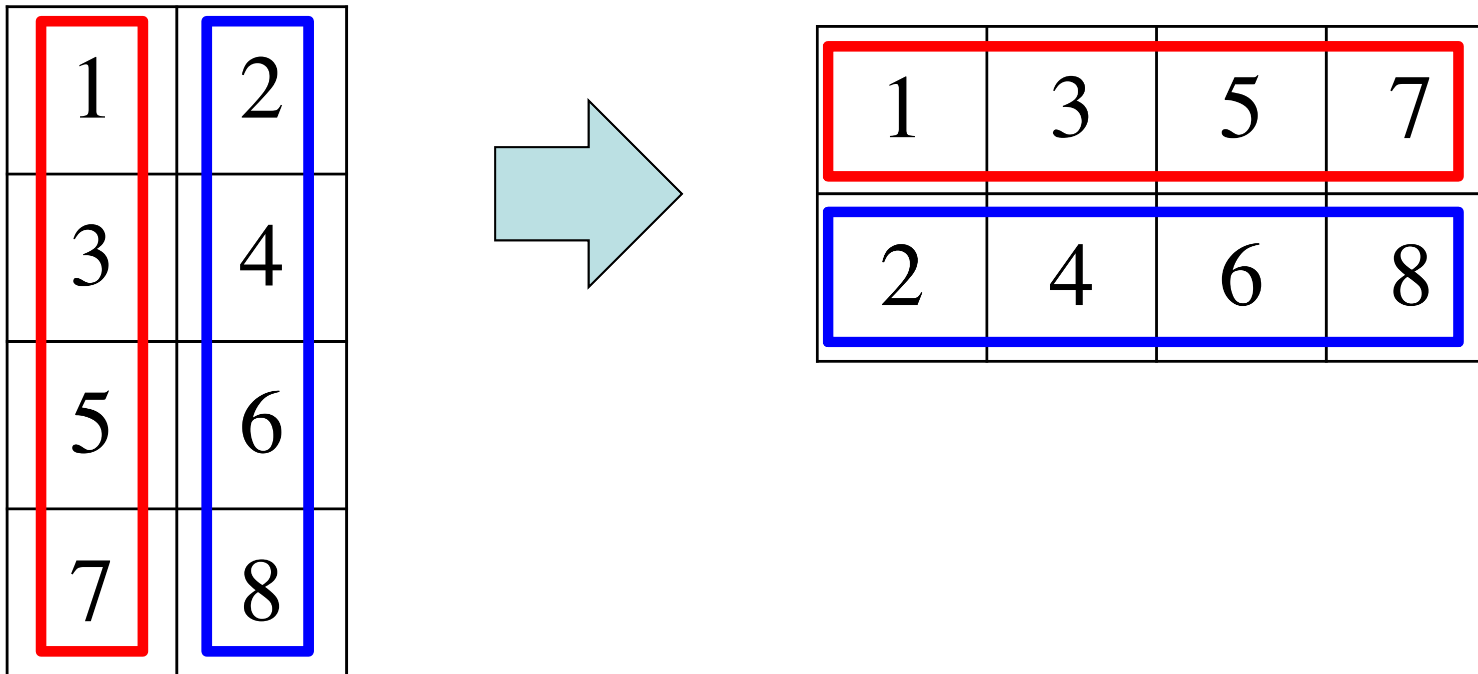
- x now has nested list

```
[[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



4 lists: 2 elements in each   2 lists: 4 elements in each

How to transpose?

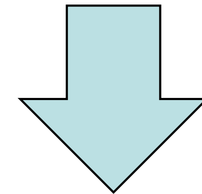
- 1st element of each list gets appended to 1st list
- 2nd element of each list gets appended to 2nd list

Data Wrangling: Transpose Code

```
def transpose(orig_table):
    """Returns: copy of table with rows and columns swapped

    Precondition: table is a (non-ragged) 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 list(range(numrows)):
            row.append(old_table[n][m]) # Build up new row
        new_table.append(row) # Add new row to new table
    return new_table
```

1	2
3	4
5	6



1	3	5
2	4	6

Tuples

strings:
immutable sequences of **characters**

lists:
mutable sequences of **any objects**

tuples:
immutable sequences of **any objects**

"tuple" generalizes "pair," "triple," "quadruple," …

- 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 **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

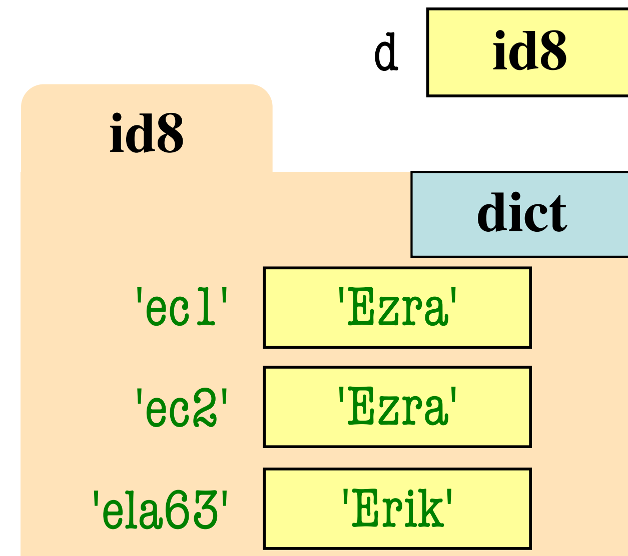
- Create with format: `{k1:v1, k2:v2, …}`
- Keys must be **immutable**
  - ints, floats, bools, strings
  - **Not** lists or custom objects
- Values can be anything
- Example:

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

Using Dictionaries (Type `dict`)

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

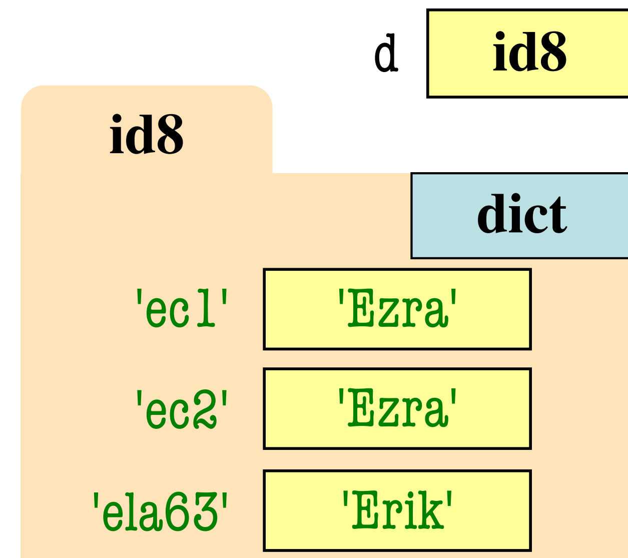
```
d = {'ec1':'Ezra','ec2':'Ezra',
     'ela63':'Erik'}
```



Using Dictionaries (Type dict)

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

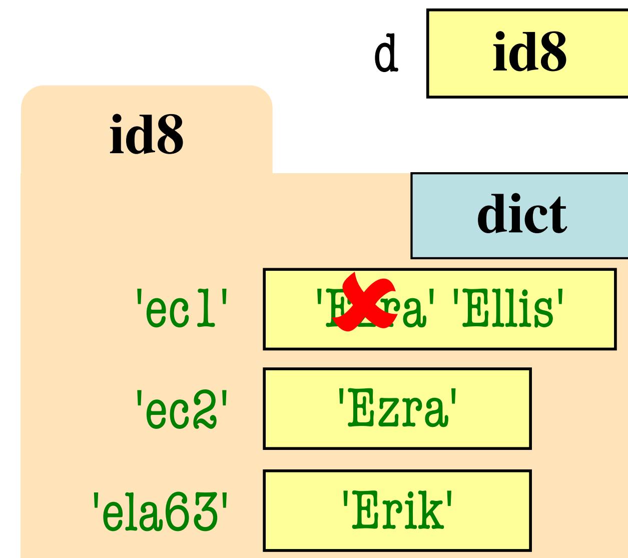
```
d = {'ec1':'Ezra','ec2':'Ezra',
     'ela63':'Erik'}
```



Using Dictionaries (Type dict)

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

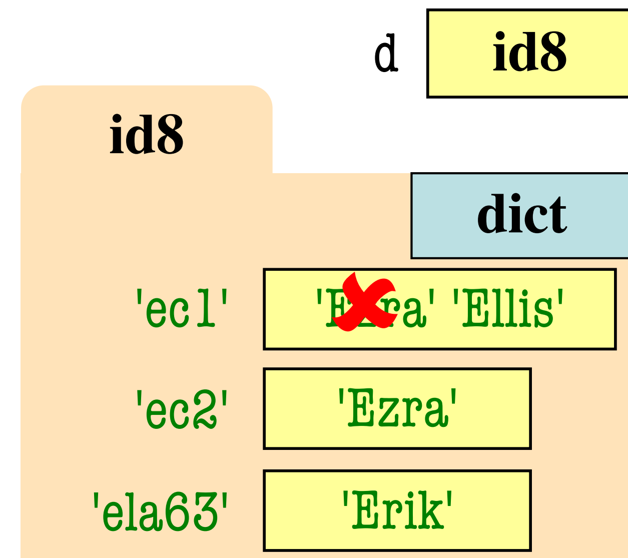
```
d = {'ec1':'Ezra','ec2':'Ezra',
     'ela63':'Erik'}
```



Using Dictionaries (Type dict)

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

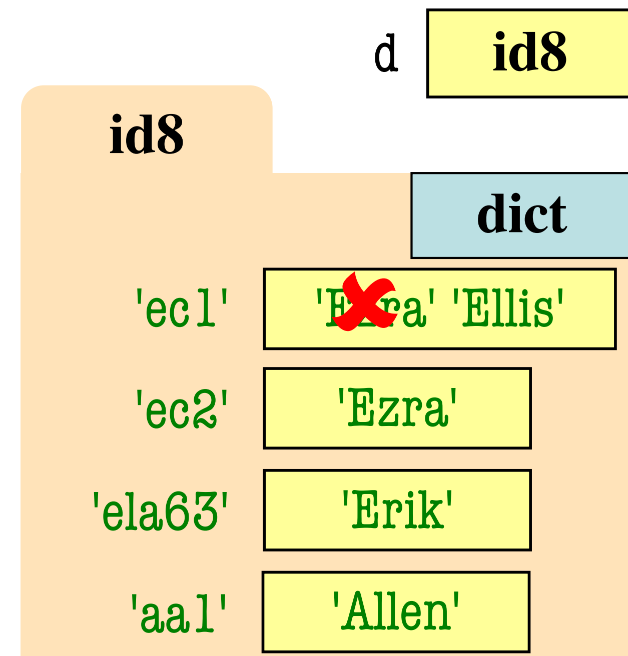
```
d = {'ec1':'Ezra','ec2':'Ezra',
     'ela63':'Erik'}
```



Using Dictionaries (Type dict)

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

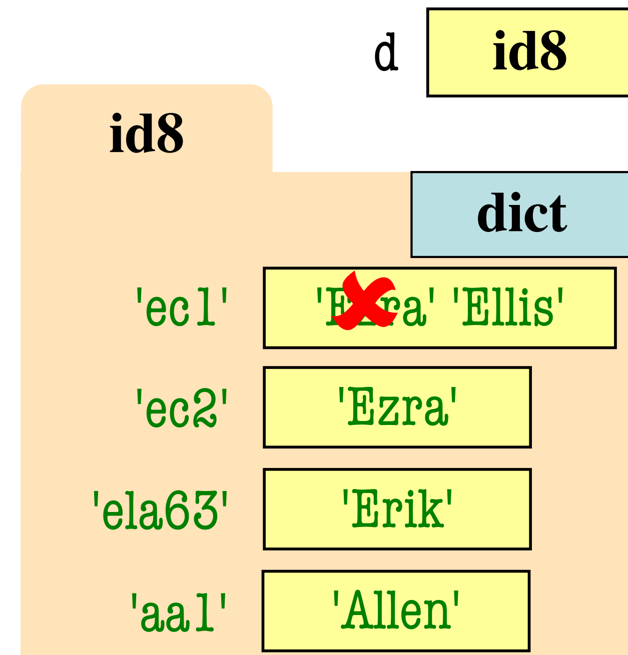
```
d = {'ec1':'Ezra','ec2':'Ezra',
     'ela63':'Erik','aa1':'Allen'}
```



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']

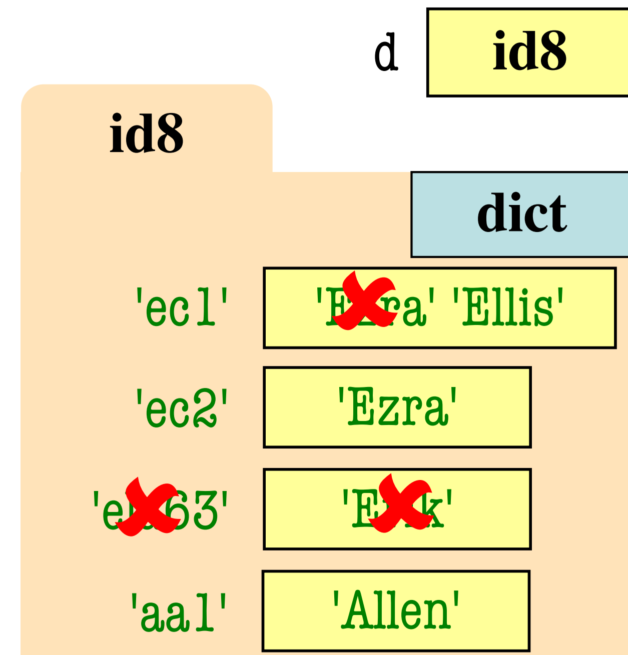
```
d = {'ec1':'Ezra','ec2':'Ezra',
     'ela63':'Erik','aa1':'Allen'}
```



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