

Lecture 14

More with Sequences

Announcements for This Lecture

Readings

- Today: Chapter 11
- Next Week: Sec. 5.8-5.10

- **Prelim, Oct 15th 7:30-9:00**
 - Material up to **TUESDAY**
 - Study guide is posted
- **Review session Wednesday**
 - Still checking place/time
 - Announcement on Piazza

Assignments

- A3 is due **tomorrow**
 - Turn in before you leave
- Opportunities for help
 - Consultants 4:30-9:30
 - Wil has OH 9-10, Fri
 - Max has OH 1-2, Fri
- Survey is posted in CMS
 - Due on day of exam
- A4 posted **after** the exam

Processing Lists: builtins

- `sum(x)` adds up all the elements in the list `x`
 - They must all be numbers!
- `min(x)` or `max(x)` find the min/max value in list `x`
 - They use the same ordering as `sort()`
- `range(a,b,c)` produces `[a,a+c,a+2*c,...,a+c*((b-a)/c)]`
 - Starts at `a`, increases by `c` each time, until `b` (or less)
 - The argument `c` is optional; `c = 1` by default
- `list(x)` converts `x` (such as a string) to a list
 - Example: `list('mimsy')` produces `['m', 'i', 'm', 's', 'y']`

The Map Function

- `map(<function>, <list>)`
 - Function has to have exactly **1 parameter**
 - Otherwise, get an error
 - Returns a new list
- Does the same thing as

```
def map(f,x):
```

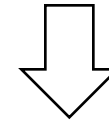
```
    result = [] # empty list
```

```
    for y in x:
```

```
        result.append(f(y))
```

```
    return result
```

`map(f, x)`



`[f(x[0]), f(x[1]), ..., f(x[n-1])]`

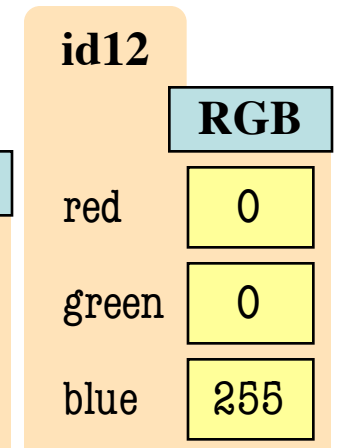
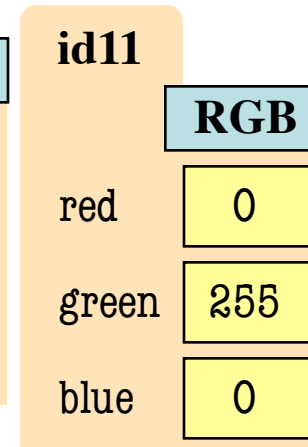
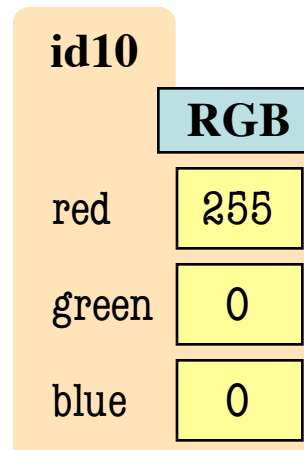
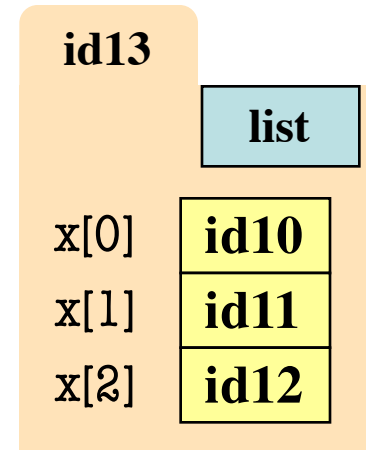
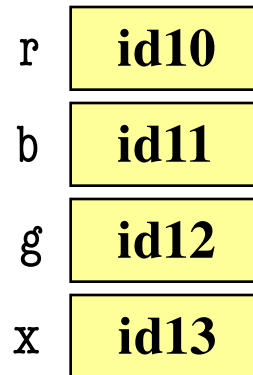
calls the function `f`
once for each item

```
map(len, ['a', 'bc', 'defg'])  
returns [1, 2, 4]
```

Lists of Objects

- List positions are variables
 - Can store base types
 - But cannot store folders
 - Can store folder identifiers
- Folders linking to folders
 - Top folder for the list
 - Other folders for contents
- Example:

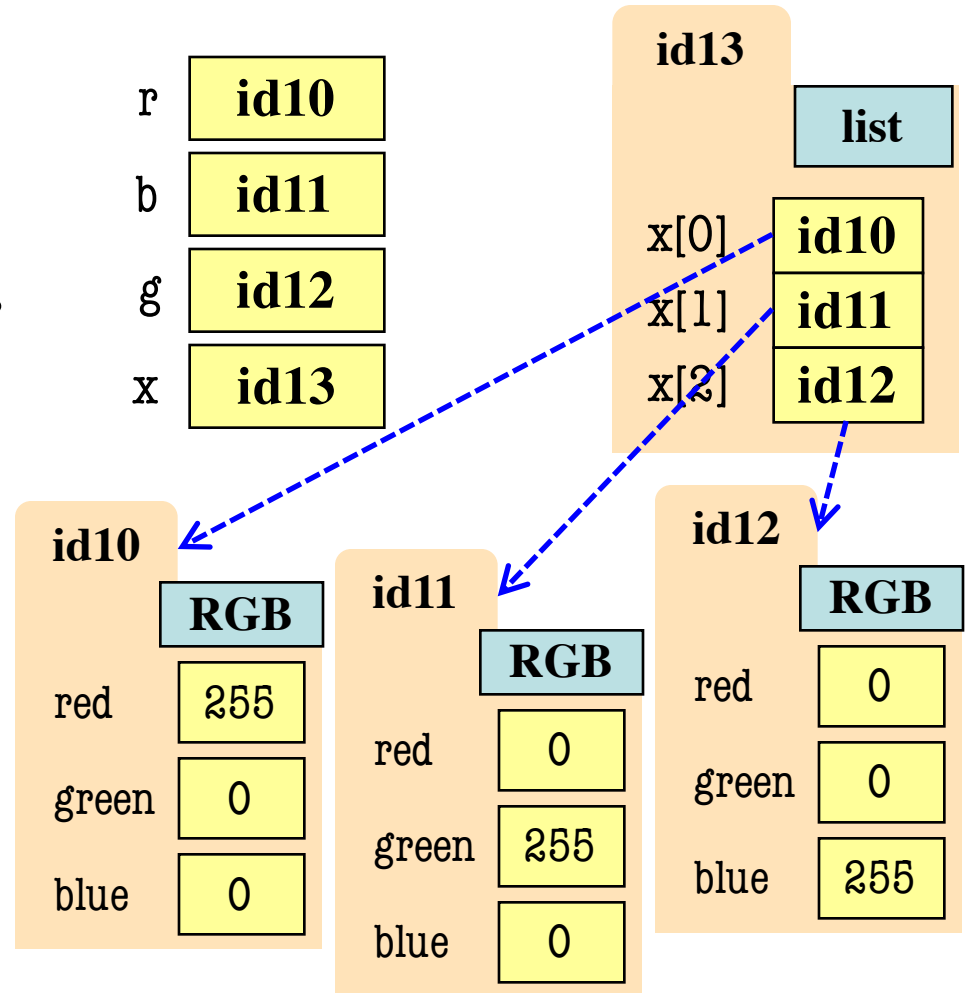
```
>>> r = colormodel.RED
>>> b = colormodel.BLUE
>>> g = colormodel.GREEN
>>> x = [r,b,g]
```



Lists of Objects

- List positions are variables
 - Can store base types
 - But cannot store folders
 - Can store folder identifiers
- Folders linking to folders
 - Top folder for the list
 - Other folders for contents
- Example:

```
>>> r = colormodel.RED
>>> b = colormodel.BLUE
>>> g = colormodel.GREEN
>>> x = [r,b,g]
```



Nested Lists

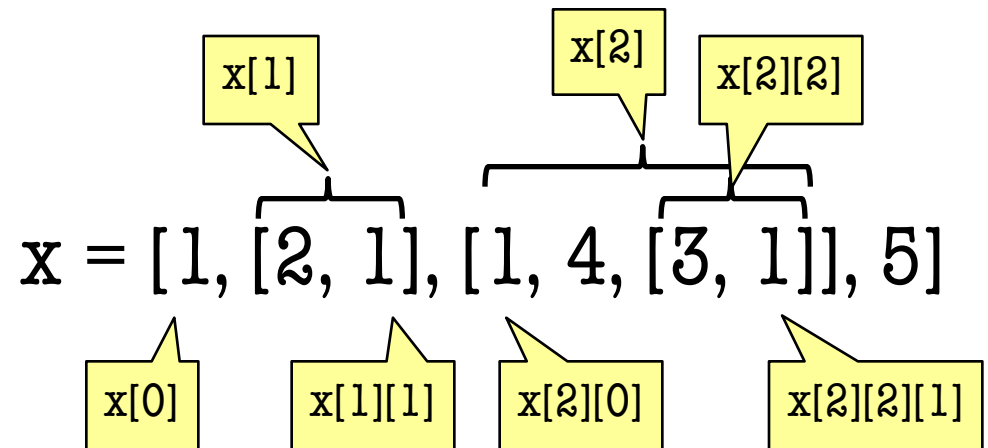
- Lists can hold any objects
- Lists are objects
- Therefore lists can hold other lists!

$a = [2, 1]$

$b = [3, 1]$

$c = [1, 4, b]$

$x = [1, a, c, 5]$



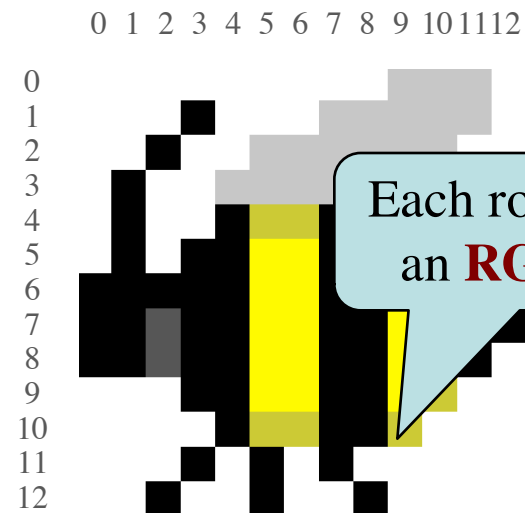
Two Dimensional Lists

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`

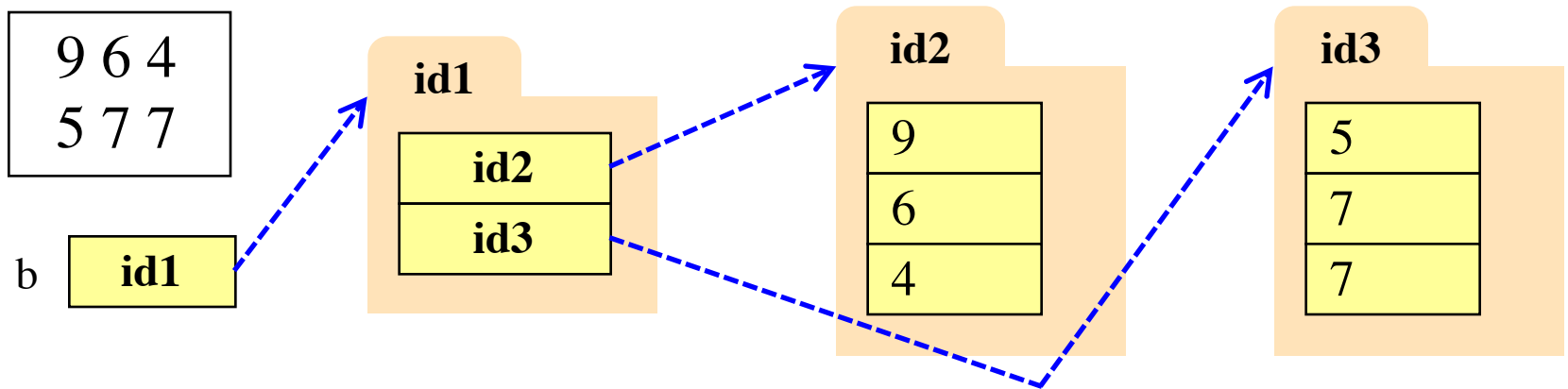
- **An odd symmetry**

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

	0	1	2	3	
d	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	

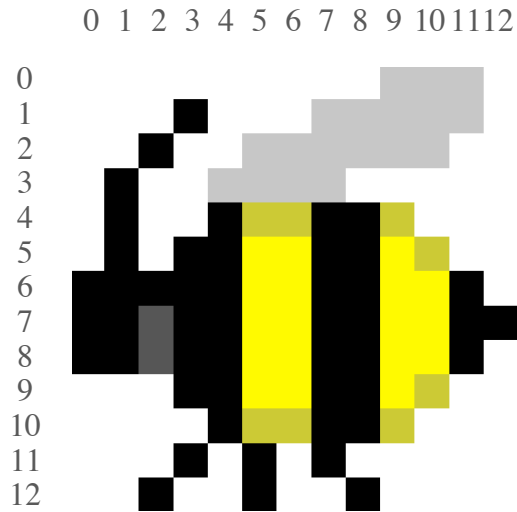
How Multidimensional Lists are Stored

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

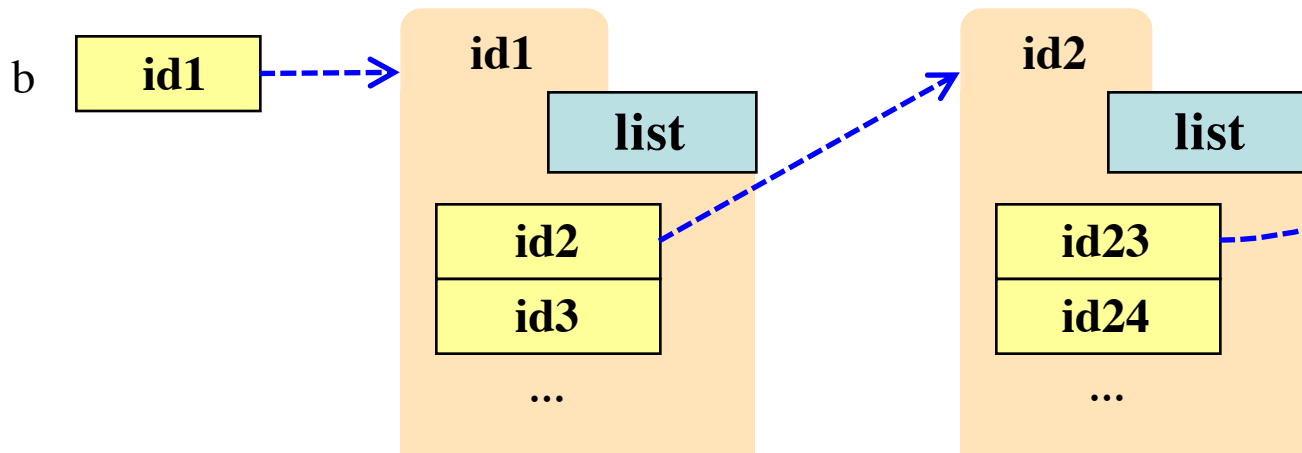
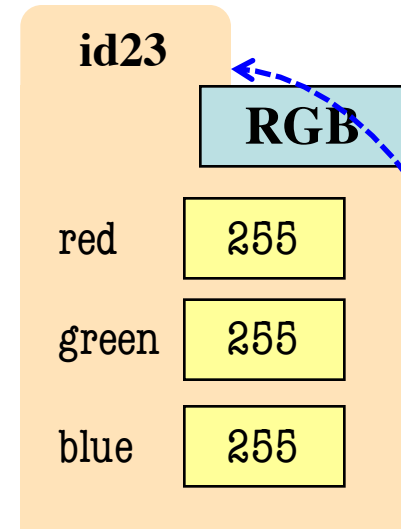


- b holds name of a one-dimensional list
 - Has $\text{len}(b)$ elements
 - Its elements are (the names of) 1D lists
- $b[i]$ holds the name of a one-dimensional list (of ints)
 - Has $\text{len}(b[i])$ elements

Image Data: 2D Lists of Pixels

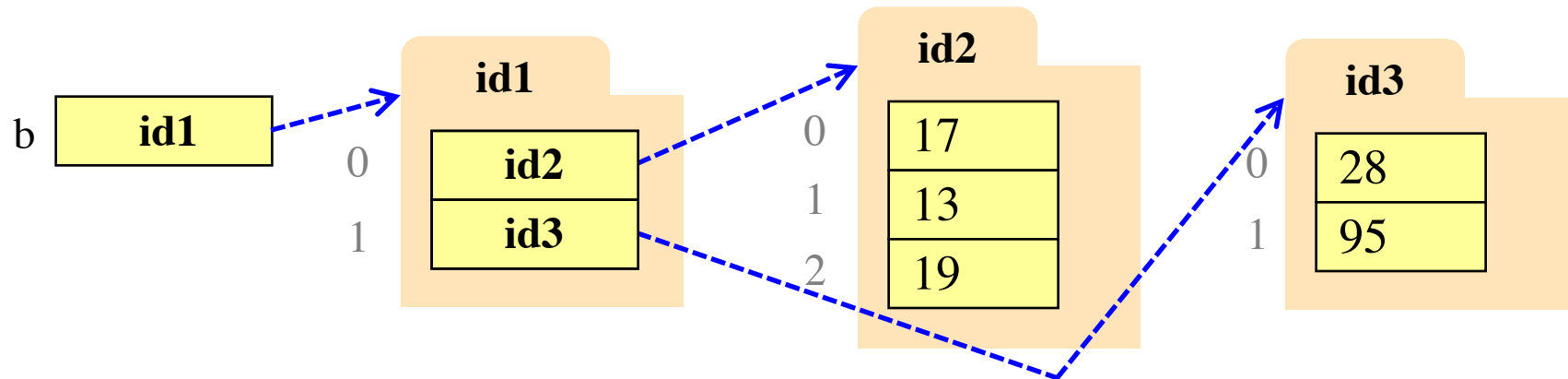


b[0][0] is a white pixel



Ragged Lists: Rows w/ Different Length

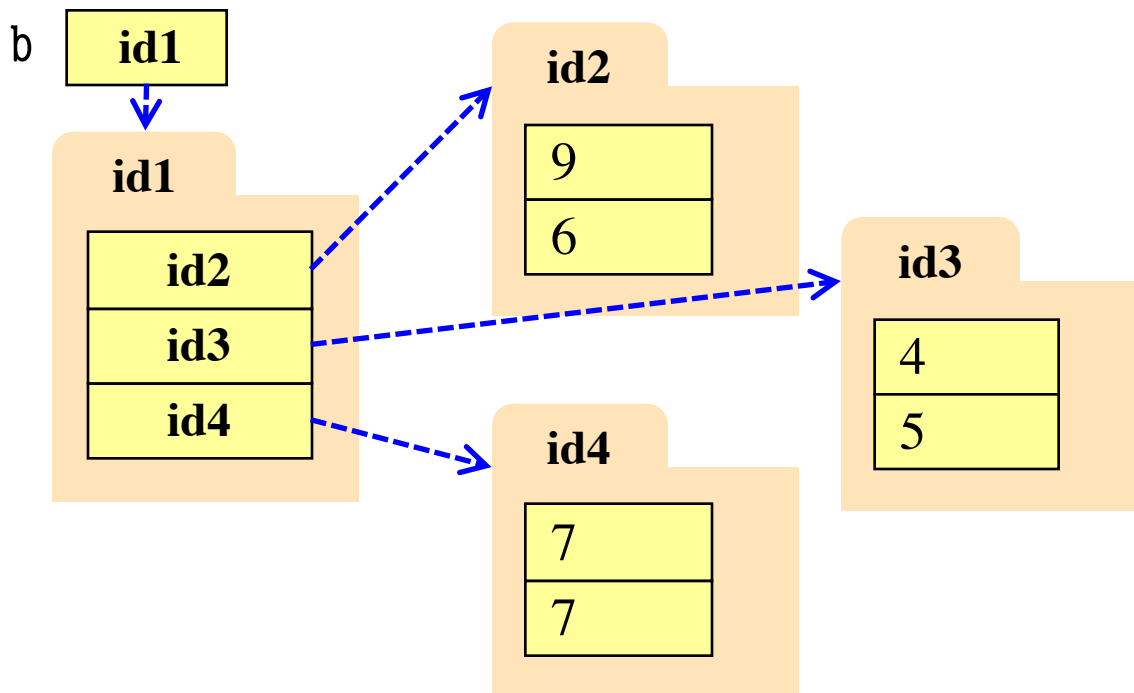
- $b = [[17, 13, 19], [28, 95]]$



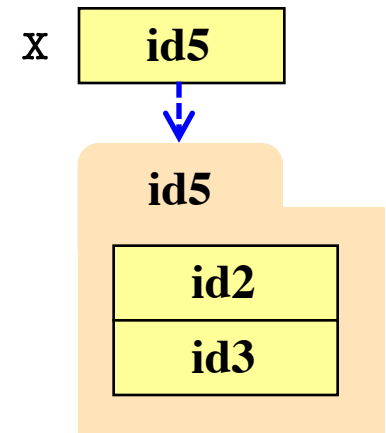
- Will see applications of this later

Slices and Multidimensional Lists

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

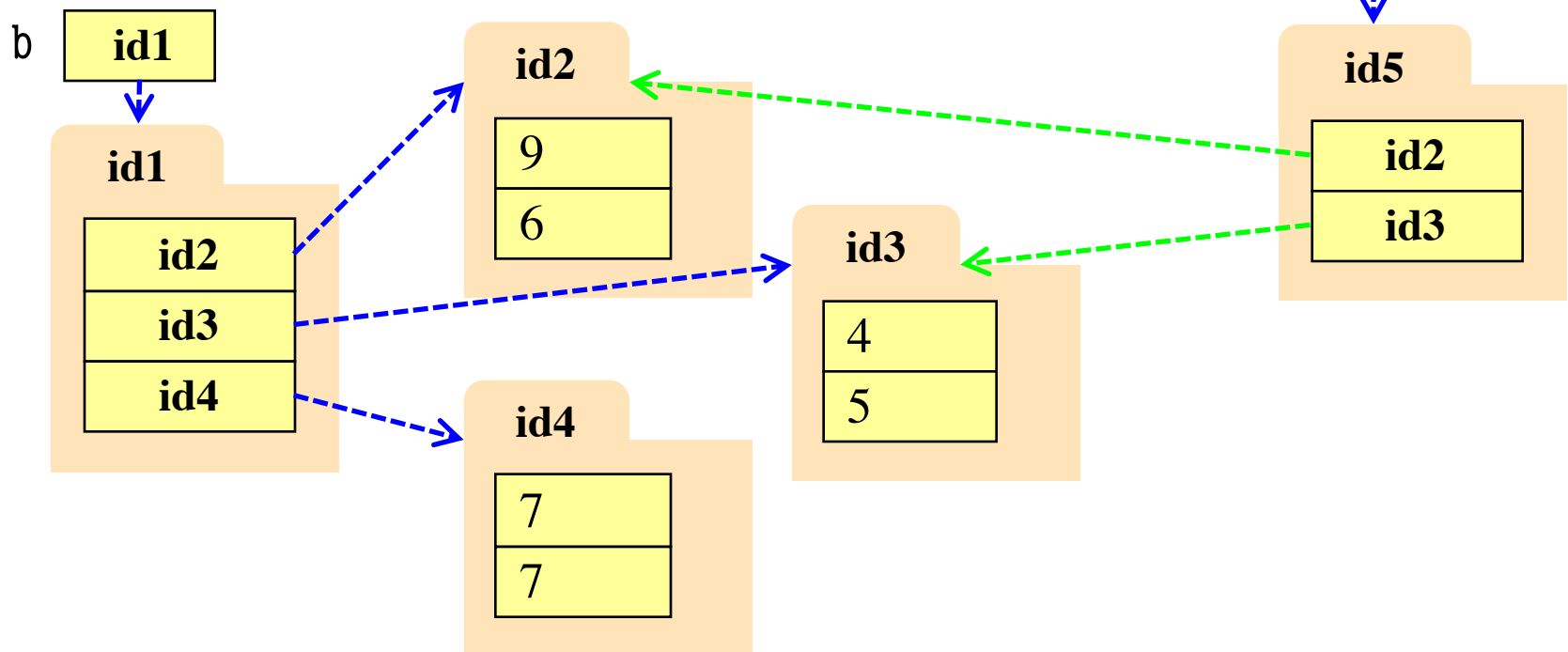


$x = b[:2]$



Slices and Multidimensional Lists

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



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 are the contents of the list (with name) 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 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 are the contents of the list (with name) 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

Functions and 2D Lists

```
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
```

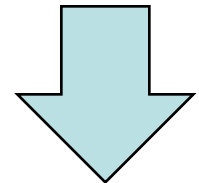
```
        for n in range(numrows):
```

```
            row.append(table[n][m]) # Build up row
```

```
        result.append(row) # Add result to table
```

```
    return result
```

1	2
3	4
5	6



1	3	5
2	4	6

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)
- Many other applications

Python Syntax

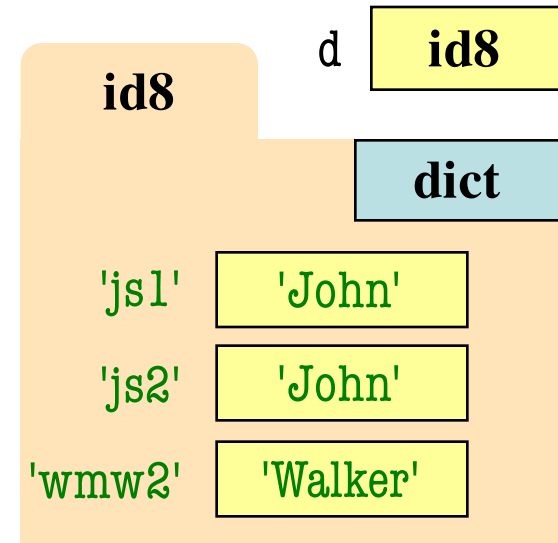
- Create with format:
`{k1:v1, k2:v2, ...}`
- Keys must be non-mutable
 - ints, floats, bools, strings
 - **Not** lists or custom objects
- Values can be anything
- Example:

```
d = {'js1':'John Smith',  
     'js2':'John Smith',  
     'wmw2':'Walker White'}
```

Using Dictionaries (Type dict)

- Access elts. like a list
 - `d['js1']` evaluates to `'John'`
 - But cannot slice ranges!
- Dictionaries are **mutable**
 - Can reassign values
 - `d['js1'] = 'Jane'`
 - Can add new keys
 - `d['aa1'] = 'Allen'`
 - Can delete keys
 - `del d['wmw2']`

```
d = {'js1':'John','js2':'John',  
     'wmw2':'Walker'}
```

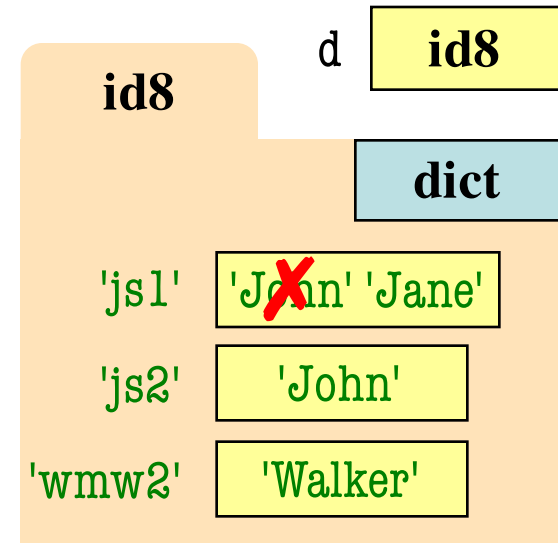


Key-Value order in folder is not important

Using Dictionaries (Type dict)

- Access elts. like a list
 - `d['js1']` evaluates to `'John'`
 - But cannot slice ranges!
- Dictionaries are **mutable**
 - Can reassign values
 - `d['js1'] = 'Jane'`
 - Can add new keys
 - `d['aa1'] = 'Allen'`
 - Can delete keys
 - `del d['wmw2']`

```
d = {'js1':'John','js2':'John',  
     'wmw2':'Walker'}
```

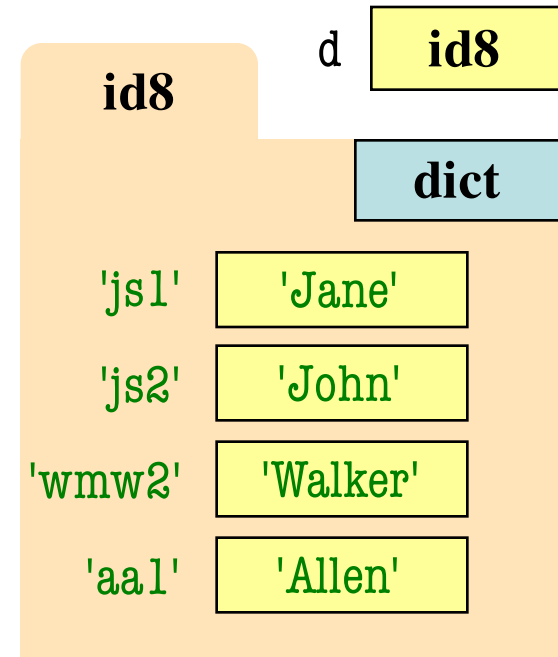


Key-Value order in folder is not important

Using Dictionaries (Type dict)

- Access elts. like a list
 - `d['js1']` evaluates to `'John'`
 - But cannot slice ranges!
- Dictionaries are **mutable**
 - Can reassign values
 - `d['js1'] = 'Jane'`
 - Can add new keys
 - `d['aa1'] = 'Allen'`
 - Can delete keys
 - `del d['wmw2']`

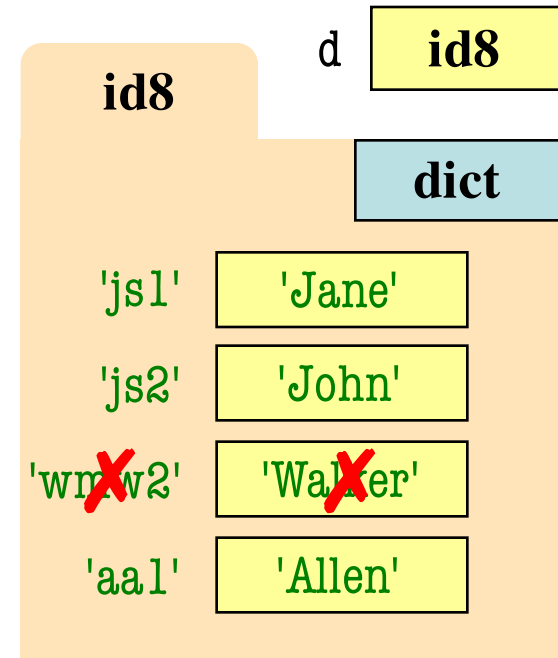
```
d = {'js1':'John','js2':'John',  
     'wmw2':'Walker'}
```



Using Dictionaries (Type dict)

- Access elts. like a list
 - `d['js1']` evaluates to `'John'`
 - But cannot slice ranges!
- Dictionaries are **mutable**
 - Can reassign values
 - `d['js1'] = 'Jane'`
 - Can add new keys
 - `d['aa1'] = 'Allen'`
 - Can delete keys
 - `del d['wmw2']`

```
d = {'js1':'John','js2':'John',  
     'wmw2':'Walker'}
```



Deleting key deletes both

Dictionaries and For-Loops

- Dictionaries != sequences
 - Cannot slice them
- *Different* inside for loop
 - Loop variable gets the key
 - Then use key to get value
- Has **methods** to *convert* dictionary to a sequence
 - Seq of keys: `d.keys()`
 - Seq of values: `d.values()`
 - key-value pairs: `d.items()`

```
for k in d:  
    # Loops over keys  
    print k    # key  
    print d[k] # value
```

```
# To loop over values only  
for v in d.values():  
    print v    # value
```

See `grades.py`