



<http://www.cs.cornell.edu/courses/cs1110/2021sp>

Lecture 12: Nested Lists and Dictionaries

(Sections 11.1-11.5)

CS 1110

Introduction to Computing Using Python

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

Announcements

- Be sure to go to section for Labs 11 & 12
- A3: first submission (“part A”) due Mar 24; final submission due Mar 28
- Definitive source for due dates is the course webpage, but we try to also put due dates on the Canvas calendar
- A2 grades and solutions available around Wednesday
- Next lecture will be a review session
- Tues 3/30 lecture will be open office hour
- Prelim 1 Study Guide available tonight. Be sure to read it!
- Exam logistics: seat number and Zoom link to be distributed via CMS by end of the week. Online exam takers will be contacted by proctor to do a **required** short mock exam *before* actual exam.

Nested Lists

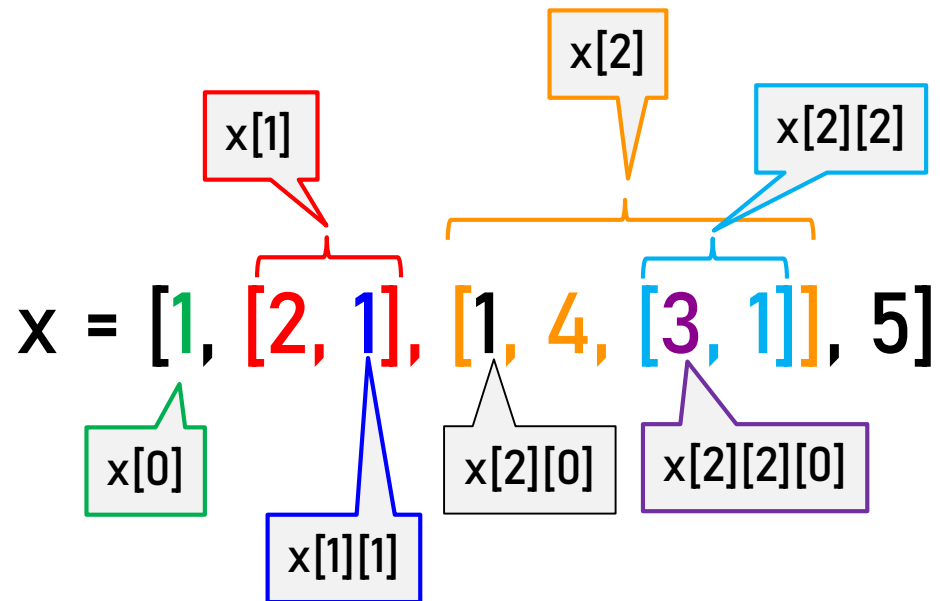
- Lists can hold any objects
- Lists are objects
- Therefore lists can hold other lists!

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

E.g., student ID

E.g., lab number

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

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

E.g., product ID

E.g., shop ID

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

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

Column index

Row index

Each row, col has a value

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.

Store them as *a list 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

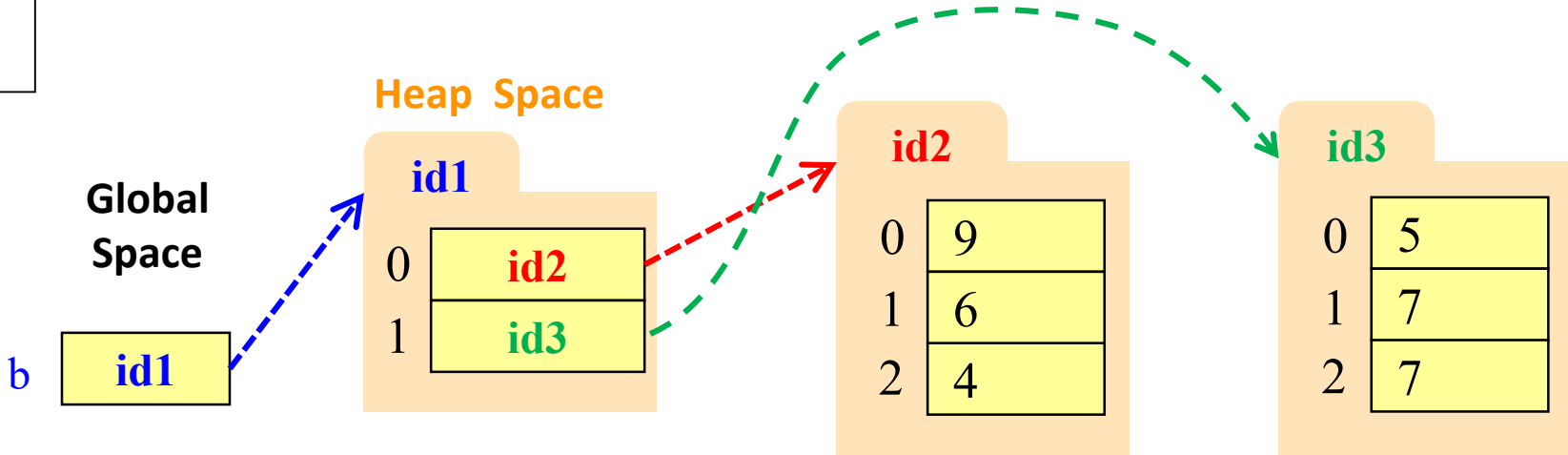
	0	1	2	3
0	5	4	7	3
1	4	8	9	7
2	5	1	2	3
3	4	1	2	9

```
>>> d = [[5,4,7,3],[4,8,9,7],[5,1,2,3],[4,1,2,9]]
>>> d[3][2]           Access value at row 3, col 2
2
>>> d[3][2] = 8      Assign value at row 3, col 2
>>> d
[[5, 4, 7, 3], [4, 8, 9, 7],[5, 1, 2, 3], [4, 1, 8, 9]]
>>> len(d)           Number of rows of d
4
>>> len(d[2])       Number of cols in row 2 of d
4
```

How Multidimensional Lists are Stored

9	6	4
5	7	7

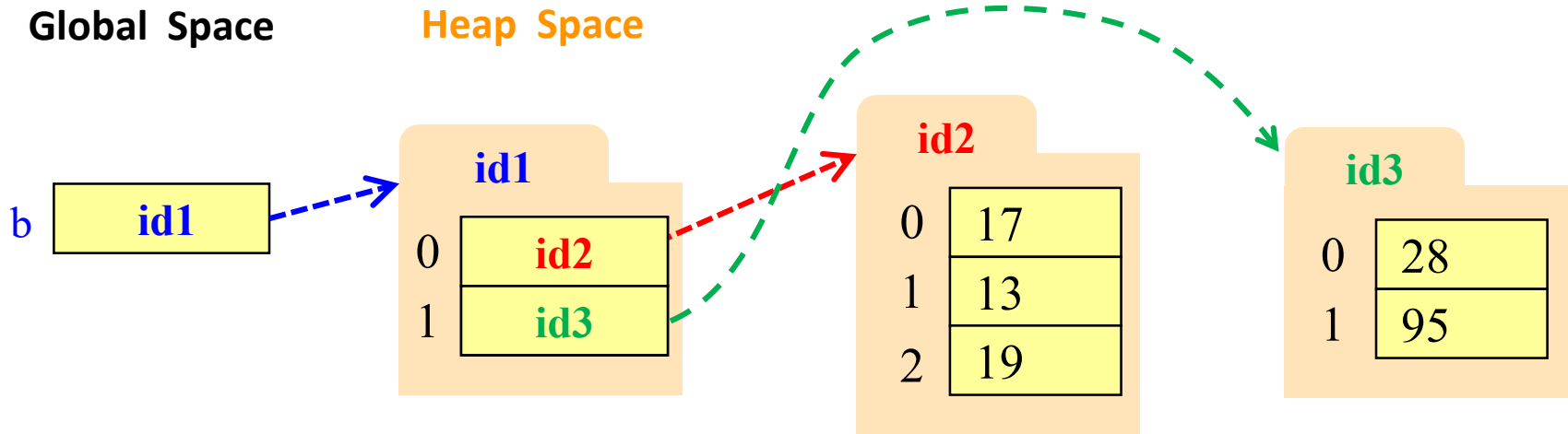
$b = [[9, 6, 4], [5, 7, 7]]$



- `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

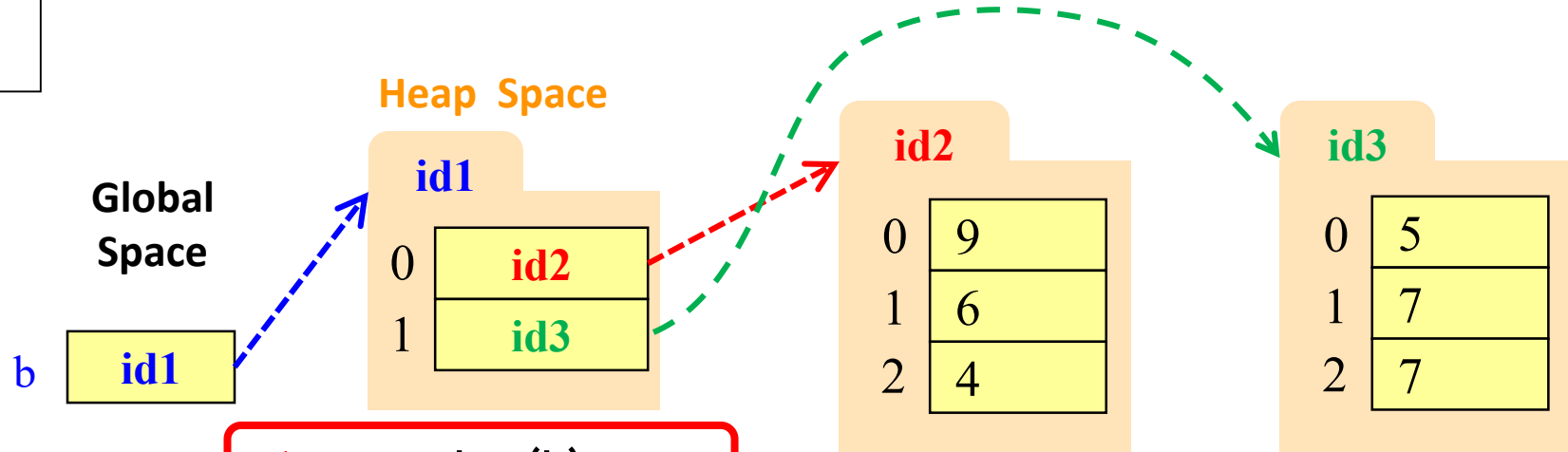
- $b = [[17,13,19],[28,95]]$



How to access every element of nested list?

9	6	4
5	7	7

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



There are $\text{len}(b)$ rows

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

A loop to go row to row. Then at each row, set a loop to go column to column.
→ Nested loops!

Row i has $\text{len}(b(i))$ elements

Exercise 1

```
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
```

```
    """
```

Exercise 2

```
def print_all_elements(my_table):
```

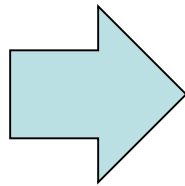
```
    """Prints all elements of the table,  
    one element on each line.
```

```
    Preconditions: my_table is a table of numbers  
                  my_table is not empty
```

```
    """
```

Data Wrangling: Transpose Idea

1	2
3	4
5	6
7	8



1	3	5	7
2	4	6	8

2 lists: 4 elements in each

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

Data Wrangling: Transpose Code

```
def transpose(table):
```

```
    """Returns: copy of table with rows and columns swapped  
    Precondition: table is a (non-ragged) 2d List"""
```

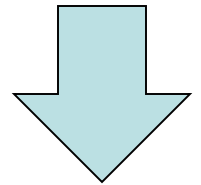
```
    n_rows = len(table)
```

```
    n_cols = len(table[0]) # All rows have same no. cols
```

```
    new_table = [] # Result accumulator
```

```
    return new_table
```

1	2
3	4
5	6



1	3	5
2	4	6

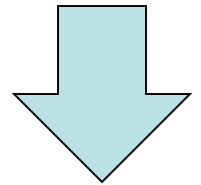
```
d = [[1,2],[3,4],[5,6]]
```

```
d_v2 = transpose(d)
```

Data Wrangling: Transpose Code

```
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  
  
    for c in range(n_cols):  
        row = [] # Single row accumulator  
        for r in range(n_rows):  
            row.append(table[r][c]) # 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

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

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

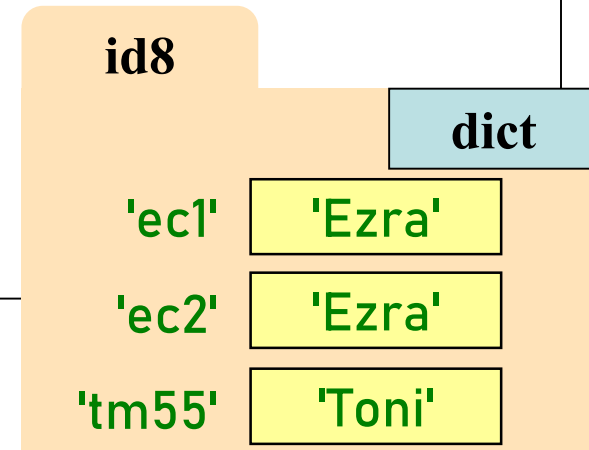
Using Dictionaries (Type dict)

```
>>> d = {'ec1': 'Ezra', 'ec2': 'Ezra', 'tm55': 'Toni'}
>>> d['ec1']
'Ezra'
>>> d[0]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 0
>>> d[:1]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unhashable type: 'slice'
>>>
```

Global Space

d id8

Heap Space



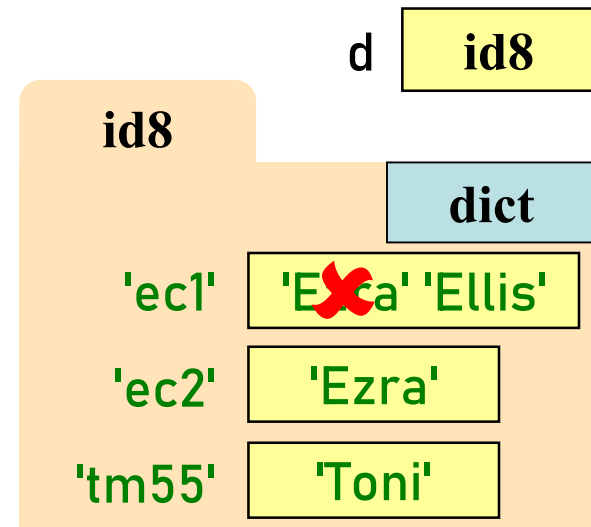
- Can access elements like a list
- Must use the key, not an index
- Cannot slice ranges

Using Dictionaries (Type dict)

- Dictionaries are **mutable**

- Can reassign values
- `d['ec1'] = 'Ellis'`

```
d = {'ec1':'Ezra','ec2':'Ezra',  
      'tm55':'Toni'}
```

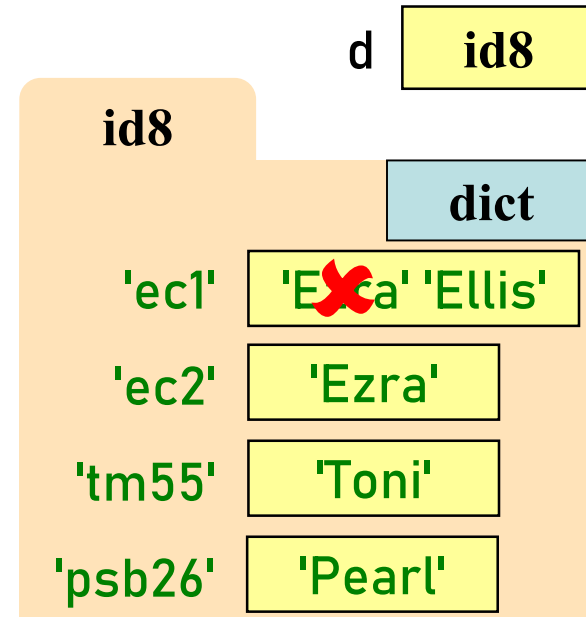


Using Dictionaries (Type dict)

- Dictionaries are **mutable**

- Can reassign values
- `d['ec1'] = 'Ellis'`
- Can add new keys
- `d['psb26'] = 'Pearl'`

```
d = {'ec1':'Ezra','ec2':'Ezra',  
     'tm55':'Toni','psb26':'Pearl'}
```

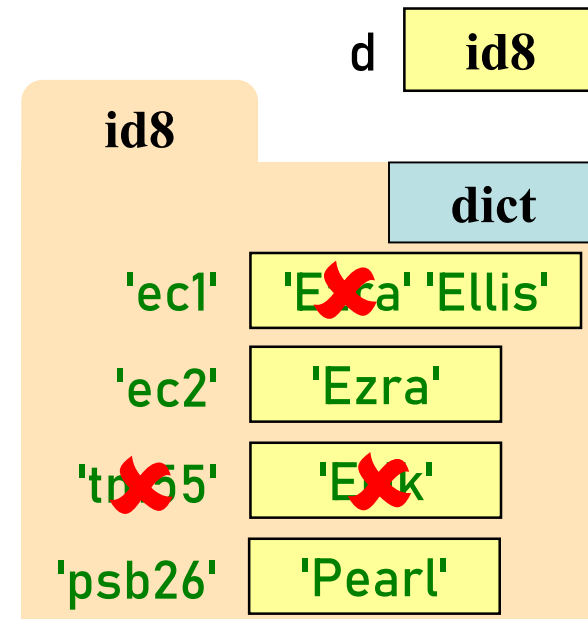


Using Dictionaries (Type dict)

- Dictionaries are **mutable**

- Can reassign values
- `d['ec1'] = 'Ellis'`
- Can add new keys
- `d['psb26'] = 'Pearl'`
- Can delete keys
- `del d['tm55']`

```
d = {'ec1':'Ezra','ec2':'Ezra',  
     'psb26':'Pearl'}
```



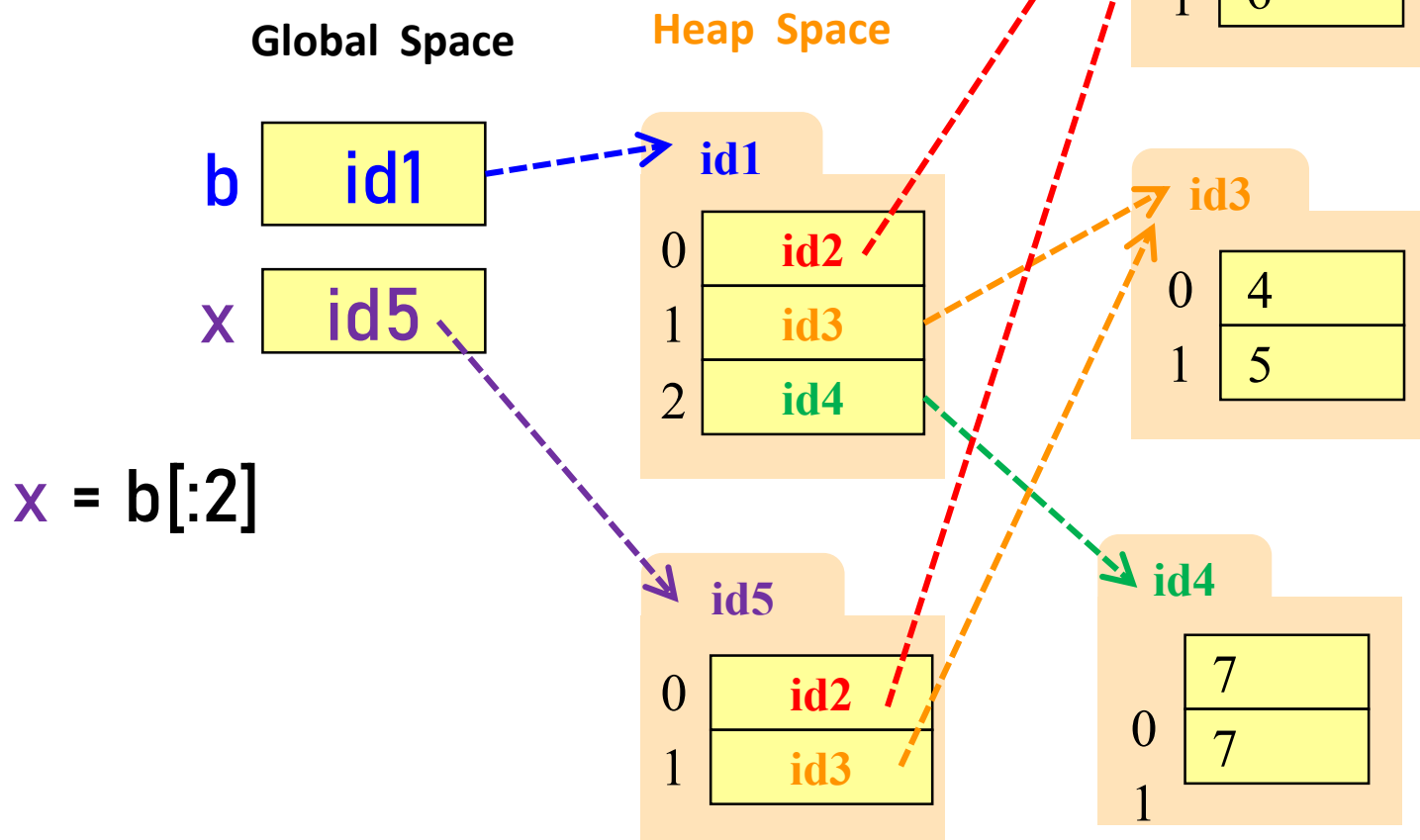
Be sure to read
Textbook 11.1-11.5 for
additional examples!

Deleting key deletes both
key and value

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 & 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