Lists and Dictionaries

CS 8: Introduction to Computer Science, Spring 2019
Lecture #14

Ziad Matni, Ph.D.
Dept. of Computer Science, UCSB

Administrative

- **Hw07 out today – DUE ON TUESDAY 5/4**
- **Hw08 out on Thursday – DUE ON THURSDAY 5/6 (last day of lecture)**

- **Lab06** – issued and due by next week Thursday (5/6) by 11:59 PM
  - A little involved, so feel free to pair-up (optional)
  - Go to the lab next Monday to work on it

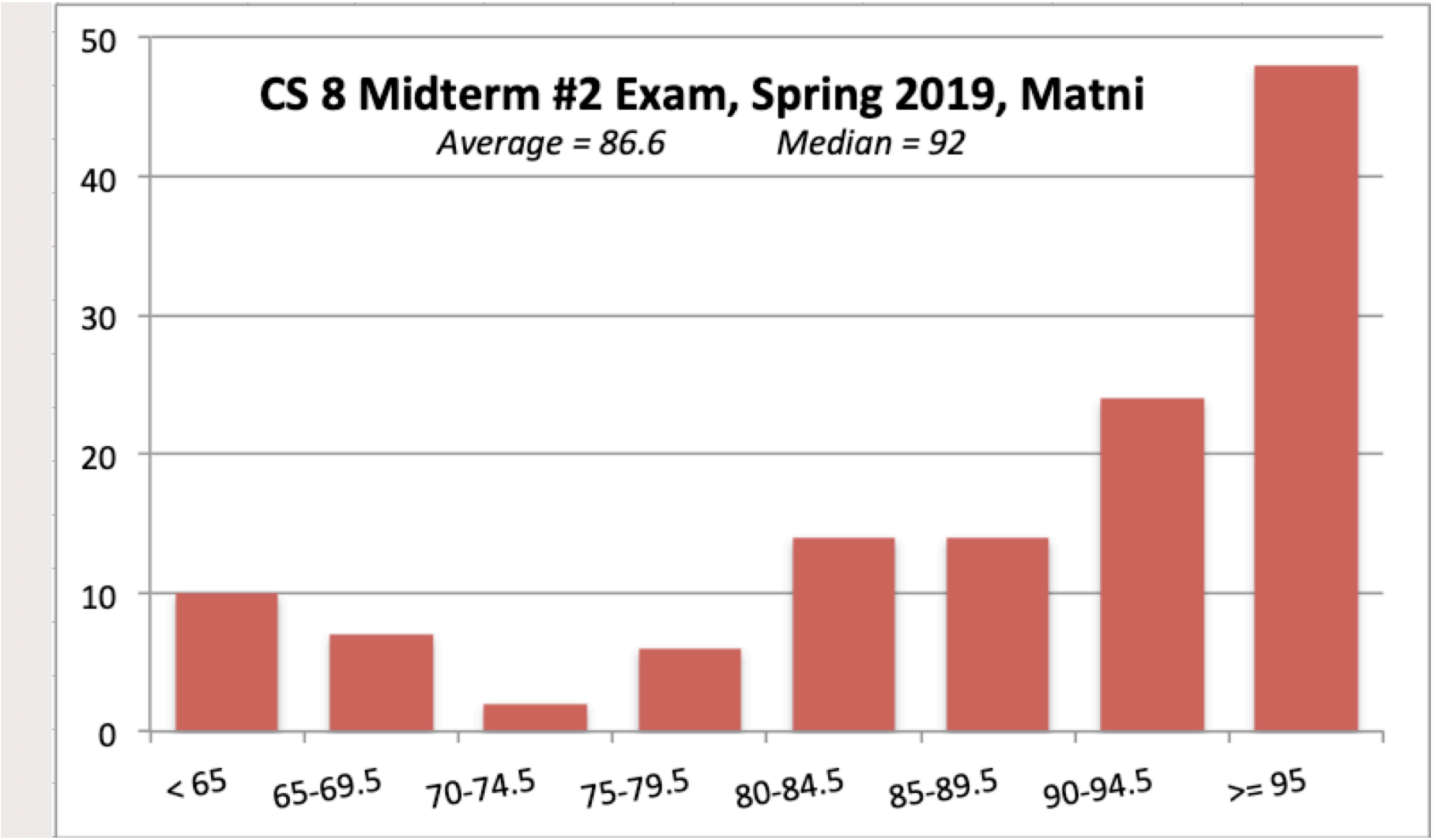
- You are still working on Project #1… right?
  - Due on Sunday!

Administrative

- **MIDTERM #2 is graded!**
  - Grades on GauchoSpace
  - To review your midterms: **same arrangement as with Midterm #1**

- If you're in the **8 AM** lab – go to **Chong Liu's** office hours
- If you're in the **9 AM** lab – go to **Brian Young's** office hours
- If you're in the **10 AM** lab – go to **Shane Masuda's** office hours
- If you're in the **11 AM** lab – go to **Prof. Matni's** office hours

- Average was 86.6 and median was 92 (very high!)


CS 8 Midterm #2 Exam, Spring 2019, Matni
Average = 86.6
Median = 92
50
40
30
20
10
0
< 65
65-69.5
70-74.5
75-79.5
80-84.5
85-89.5
90-94.5
>= 95

Study Session for Final Exam

- Any interest?

- Would take place on Friday, June 7th
- Location: TBA

Lecture Outline

- Doing more with Lists
  - Simple Example: Find the **Median** of multiple numbers
- Dictionaries

What Operations Can We do with LISTS?

- Find min, max, sum
  - `max(list), min(list), sum(list)`
  - How does this work with numbers? strings?
- Add item to a list (at the end)
  - `list.append(item)`
- Sort a list
  - `list.sort()`
- Reverse a list
  - `list.reverse()`
- Remove item from a list
  - `list.remove(item)`
- Count how many of something
  - `list.count(item)`
- Add item to a list (anywhere: before some index)
  - `list.insert(index, item)`
- Remove LAST item from a list
  - `list.pop()`

Calculating Means and Medians Using **Lists**

- Mean (Average) = (max – min) / sum
- Median (middle item) is more complex…
  - This isn't in any list function, so we have to develop it ourselves

Example:

1	5	2	10	8	7	7	6	3

sort it first and then find the middle value…

1	2	3	5	6	7	7	8	10

Median = 6

If there's an even number of entities, then employ an average calculation…

1	2	3	5	6	7	7	8

Median = 5.5

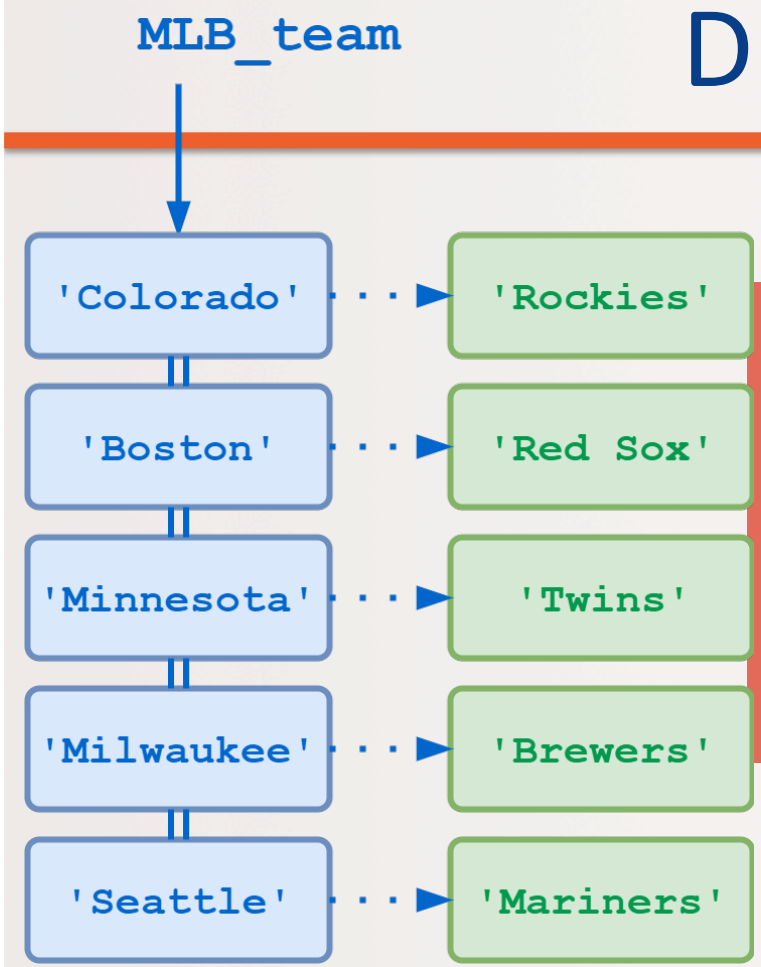
"Find the Median" Algorithm

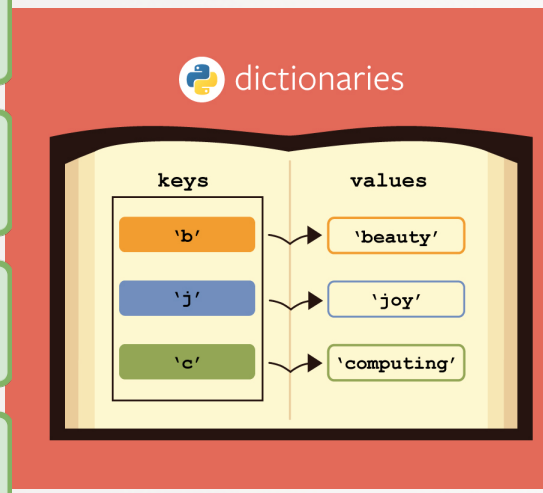
1. ***Sort*** *the list first*
2. *Determine the* ***length*** *of the list (why?)*
3. *Find the* ***middle*** *of the list (length/2)*
   a) *If the length is an* ***odd*** *number, then there's only 1 middle*
   b) *If the length is an* ***even*** *number, then identify the middle 2 and get their average*

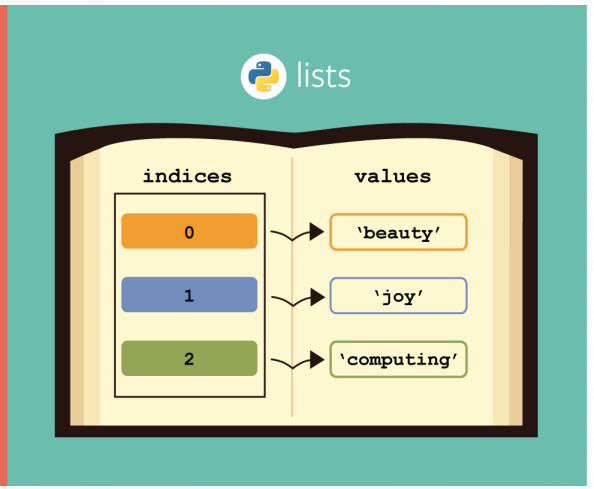
“Find the Median” Function

```
def median(alist):
    # Make a copy so we won't change "alist" itself (why worry about that?)
    CopyList = alist
    CopyList.sort()       # guess what this does??

    if len(CopyList)%2 == 0:    # if there is an even no. of things in the list,
                                # then, we should identify the middle 2 numbers
        rightMiddle = len(CopyList)//2   # That's the *position* of the right-middle no.
        leftMiddle = rightMiddle - 1     # That's the *position* of the left-middle no.
        median = (CopyList[leftMiddle] + CopyList[rightMiddle])/2

    else:                       # if there is an odd no. of things in the list,
                                # then, it's easier: just find the middle number
        index_of_middle = len(CopyList)//2
        median = CopyList[index_of_middle]

    return median
```

Dictionaries







Images from: realpython.com and bjc.edc.org

Dictionaries

- Popular data structures in Python
- **Unordered *associative*** collections
  - Basically like **lists**, but you can access each value by a **key**

instead of an index position

- Use curly braces, **{ }** to define a dictionary

```
ages = { 'sam':19, 'alice':20 }
```

KEY VALUE

NOTE THE SYNTAX
and the use of the colon

key*:*value

Let's try it!

Dictionaries – Key/Value Pairs

- Use the familiar **[ ]** to **access**, **set** or **delete** *by key*

```
>>> print(ages['alice'])
    20

>>> ages['pete'] = 24    # adds new item in this case
>>> del(ages['pete'])    # bye bye pete
```

- In Dictionaries, we don't use **indexing** like we did with lists
  - That's because values are **not** stored in a discernible order
  - How do find things in a dictionary? We go by the **key**.

Let's try it!

Useful Functions for Dictionaries

Assume: `Ages = {'Britta':33, 'Annie':20, 'Jeff':42 }`

Show all the keys

- `Ages.keys() = ['Britta', 'Annie', 'Jeff']`

Show all the values

- `Ages.values() = [33, 20, 42]`

FYI: Although *these look like lists*, they are actually different kinds of data types:
dict_keys and ***dict_values***

You can always try using **type()** to figure out the data type you're using!

Another Useful Dictionary Function

Assume: `Ages = {'Britta':33, 'Annie':20, 'Jeff':42 }`

Show all the items in the dictionary as a **list of tuples**

- `Ages.items() =`

```
[('Britta', 33), ('Annie', 20), ('Jeff', 42)]
```

What Will These Do, if:

```
ages = { 'sam':19, 'alice':20, 'ben': 22, 'bert': 44 }
```

```
for item in ages:
    print(item)
```

```
for item in ages.keys():
    print(item)
```

```
for item in ages.values():
    print(item)
```

```
for item in ages.items():
    print(item)
```

```
for item in ages.items():
    print(item[0])
```

Let's try it!

Application Example: **Finding the Mode**

- Number that occurs **most often** within a set of numbers

- ***Example:***

  Consider the set of numbers: 1, 3, 2, 3, 5, 1, 6, 1

  The mode is 1.

- Given a list **nums = [1, 3, 2, 3, 5, 1, 6, 1]**, how do I find the mode?
  - I'll have to make a count of all the elements
  - The element with the highest count is the "mode"

Find the **Mode** of a List: *The Algorithm*

Simple (without coding detail) algorithm/plan:

We'll create a dictionary to store all the **numbers** in the list WITH their **frequency counts** (i.e. how often they appear):

- Go thru each number in the list, and:
  - Put it in the dictionary (as *key*) and mark the count (as *value*) as 1
  - If you see that number again, increment the *value*

..cont…

Find the **Mode** of a List: *The Algorithm*

…cont…

- When this is done, look at all the *values* you've collected and search for the **BIGGEST** one (why?)

- Now that you have the maximum value, look for the key that it's associated with – THAT'S YOUR MODE! ☺
  - Careful: there may be cases where you have MORE than 1 mode!

Finding The **Mode** Of A List

```
def mode(alist):
    countdict = {}                  # Start with a blank dictionary
   for item in alist:
     if item in countdict:          # Is it already in the dictionary?
         countdict[item] += 1       # if so, increment its "value"
     else:                          # If it ISN'T in the dictionary…
         countdict[item] = 1        # Put it in there! Give it "value" = 1
```

…Continued next slide

Finding mode (cont.)

Continued…

```
countlist = countdict.values()    # Make a values list
maxcount = max(countlist)         # Get the biggest value

modelist = [ ]              # make a list of the modes (why a list?)
for item in countdict:      # Go thru the dictionary you've created
   if countdict[item] == maxcount:  # If you find the "biggest value"
          modelist.append(item)     # Add the "biggest value" key

return modelist
```

YOUR TO-DOs

- ❑ **Homework 7**
- ❑ **Lab 6**
- ❑ **Project**

</LECTURE>