Python Lists

CS 8: Introduction to Computer Science
Lecture #8

Ziad Matni
Dept. of Computer Science, UCSB

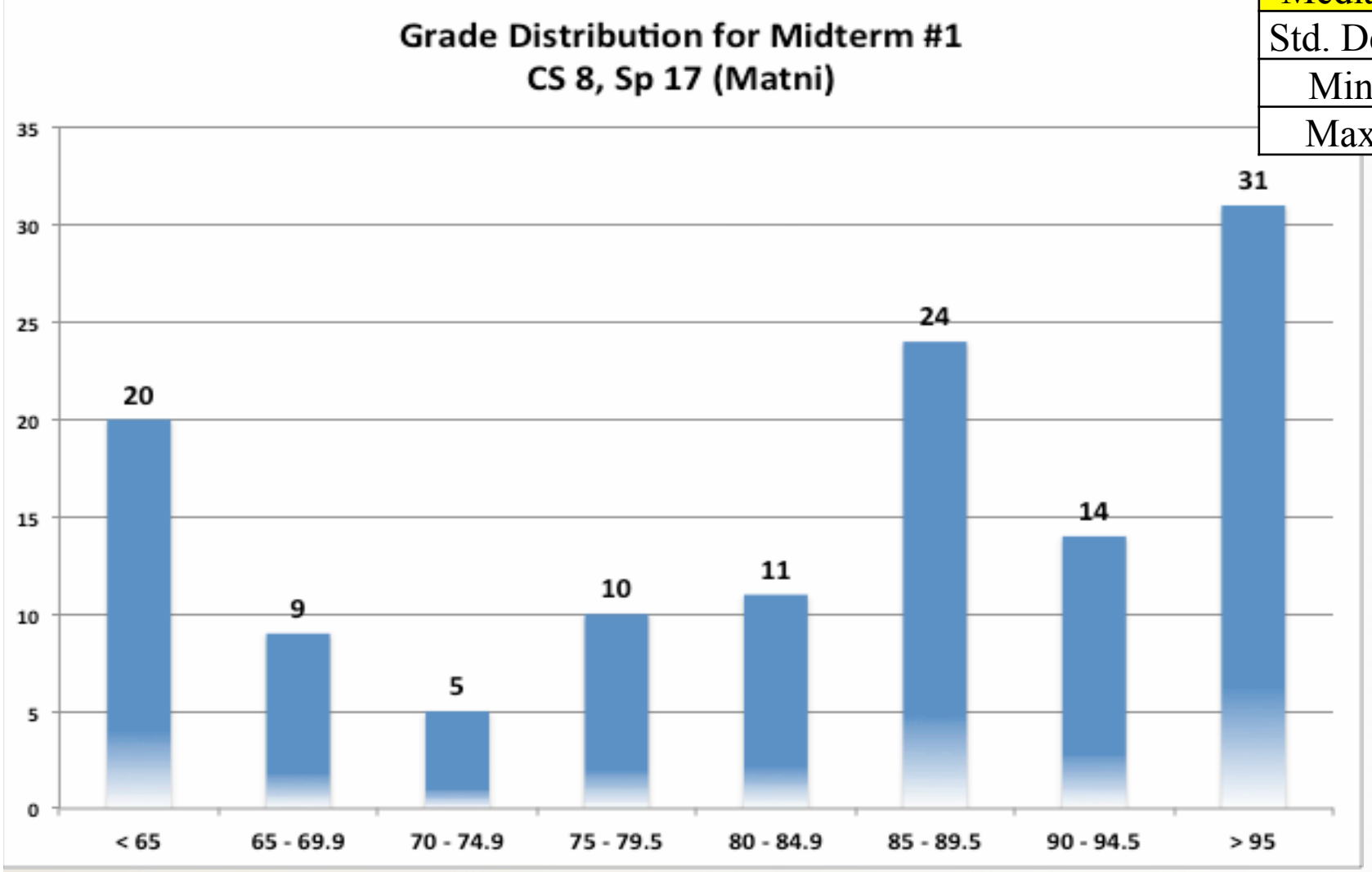
Administrative

- **Midterm is graded!**
  - Grades are online

- **Don't forget your TAs' and Instructor's office hours!! ☺**

Midterm #1 Results

Average	**81.4**
Median	**86**
Std. Dev.	16.5
Min	30
Max	103



Starting chapter 4

Sequential Data Types

- Data types that are made up of other data types
- *Example*:

  Strings are made up of character elements

- Strings are **immutable**
  - You can't exchange a character in strings by simple assignment
  - *Example*:

    Let's say, **s = 'book'**, you cannot issue **s[3] = 'm'** and expect the string **s = 'boom'**
    (it won't work that way, you'd have to do other manipulation)

Lists – More Versatile Sequences

- **Lists** are another sequential data type

- But **unlike strings**, lists …
  - can hold **any** type of data (not just characters)
  - *are* mutable – legal to change **list elements**

Lists – More Versatile Sequences

- Use square brackets, `[ ]` to define a list

```
fruit = ['apple', 'pear', 'orange', 'lemon']
```

- And use `[ ]` to access elements too

```
fruit[2]    >>> 'orange'
```

  - Indexing works the same as strings
    - i.e. start with [0]
  - Index slicing works the same as with strings too
    - E.g. fruit[1:] = ['pear', 'orange', 'lemon']
    - E.g. fruit[:1] = ['apple', 'pear']

List Examples

```
>>> li = ['abcd', 2, 3, 'efg', True, 7]
>>> li
['abcd', 2, 3, 'efg', True, 7]

>>> li[0]
'abcd'

>>> li[1] - li[2]
-1

>>> li[1] + li[0]
TypeError: cannot concatenate 'str' and 'int' objects
```

Note: mixed data types can be placed inside 1 list

DEMO! Let's try it!

Other Built-In List Functions

See table 4.2 in textbook: all used as ***listname.function( )***

- append
- insert
- pop
- sort
- reverse
- index
- count
- remove

DEMO!
Let's try it!

Other Operations Involving Lists

- Built-in functions like **`len`** (same as strings)
  - Use **`max`** and **`min`** for extremes (work for strings too)
  - And **`sum`** (only if all elements are number types)

- Test membership in lists,
  just like you can with strings: **`in`**, **`not in`**

More Operations Involving Lists

- But unlike strings, can use built-in **`del`** operator:

```
fruit >>> ['apple', 'pear', 'orange']
del fruit[1]
fruit >>> ['apple', 'orange']
```

- Also can use **`[ ]`** with **`=`** to change elements too

  (can't do that with strings…)

```
fruit[0] = 'tangerine'
fruit >>> ['tangerine', 'orange']
```

List Operations: + and *

- **+** concatentates (but both operands must be lists)

```
nums = [20, -92, 4]
nums + 9 >>> TypeError
nums + [9] >>> [20, -92, 4, 9]
```

- ***** repeats (one operand is a list, other is an int)

```
nums * [2] >>> TypeError
nums * 2  >>> [20, -92, 4, 20, -92, 4]
```

- Note: can make a list of lists, but still just 1 `nums`

```
[nums] * 2  >>> [[20, -92, 4], [20, -92, 4]]
```

  - Explained next slide

Actually, Lists Hold References

- Look at prior example a different way to see this

```
[nums, nums] == [nums] * 2 >>> True
```

- Now give a name for the list of list references

```
numList = [nums, nums]
numList >>> [[20, -92, 4], [20, -92, 4]]
```

Actually, Lists Hold References

- Delete an item from original list – see result!

```
del(nums[0])
numList >>> [[-92, 4], [-92, 4]]
```

- WHY ARE ALL OF AFFECTED?!?!?!

- Look at p. 124 in textbook (especially Fig. 4.4)

Finding extreme values

- Usually able to use built-in functions **`max`**, **`min`**
  - **But what if we didn't have such functions?**
  - **Or what if they don't fit our problem (e.g. max odd)?**

- Basic algorithm applies to any extreme

```
Store value (or index) of first list item
Loop through remaining items:
    If current more extreme than stored item:
        Replace stored extreme item (or index)
```

  - Assumes there is at least one item in the list

Find-the-Maximum Algorithm

```
1. Store value of first list item
2. Loop through remaining items:
      If current item > than stored item:
         Replace stored extreme item
```

```
def getMax(alist):
   maxSoFar = alist[0]
   for item in alist:
     if item > maxSoFar:
          maxSoFar = item
return maxSoFar
```

Calculating Means and Medians

- Mean (Average) = (max – min) / sum

- Median (middle item) is more complex…

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…

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*
3. *Find the middle of the list (length/2)*
    1. *If the length is an odd number, then there’s only 1 middle*
    2. *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
    copylist = alist
    copylist.sort()

    if len(copylist)%2 == 0:   # if length of list is even
        rightmid = len(copylist)//2
        leftmid = rightmid - 1
        median = (copylist[leftmid] + copylist[rightmid])/2

    else:  # if length of list is odd
        mid = len(copylist)//2
        median = copylist[mid]

    return median
```

YOUR TO-DOs

- ❑ Finish reading **Chapter 4**

- ❑ Finish **Homework4** (due **Thursday 5/4**)
- ❑ Begin **Lab4**
- ❑ Keep working on **Project1** (due **Friday 5/12**)

- ❑ Wash your hands