Python Lists 2

CS 8: Introduction to Computer Science
Lecture #9

Ziad Matni
Dept. of Computer Science, UCSB

Administrative

Tutoring/Review Session Available!

- Friday, **5/5** at **2:00 PM** in **PHELPS 3526**
  - T.A. Sourav M. will go over some of the basics of loops, conditionals, and functions with plenty of examples

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

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[:]  # Can also use: copylist = list(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
```

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

Specifically, for finding the maximum value in a list
(and without using the max() function):

```
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
```

Another way to create: `list()`

- With no arguments, creates an empty list

```
list() >>> []
```

- Or pass any sequence as an argument

```
list(range(3)) >>> [0, 1, 2]
list('cat') >>> ['c', 'a', 't']
```

- Makes a copy of another list (alternate to using [:])

```
nums = [-92, 4]
numsCopy = list(nums)
nums[0] = 7
nums >>> [7, 4]
numsCopy >>> [-92, 4]
```

Let's try it!

Making A List By `Split`-ting A String

- A handy string method named **`split`** returns a **list** of substrings

- Default *delimiter* is white space – consecutive spaces, tabs, and/or newline characters

Let's try it!

- Can specify a different delimiter

```
>>> 'dog,cat,wolf, ,panther'.split(',')
['dog', 'cat', 'wolf', ' ', 'panther']
```

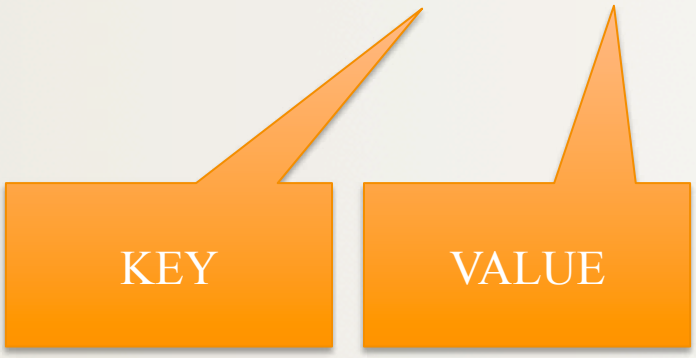
Dictionaries

- Unordered *associative* collections
  - Basically 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 }
```



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*

```
ages['alice']   >>>  20
ages['pete'] = 24 #  adds new item in this case
del(ages['pete']) #  bye bye pete
```

Let's try it!

- In Dictionaries, we don't use **indexing** like we did with lists
  - That's because values are not stored in discernible order

Useful Functions for Dictionaries

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

- Show all the keys
  - `MyDict.keys() = ['Britta', 'Annie', 'Jeff']`
- Show all the values
  - `MyDict.values() = [33, 20, 42]`

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

Tuples

- Yet another type of Python data structure
- Like a list, EXCEPT:
  - It's immutable
  - You can't add elements to a tuple
- Example: **('CS8', 125)** is a tuple
  - Note the use of **( )**, instead of **[ ]**

Another Useful Dictionary Function

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

- Show all the items in the dictionary as a **list of tuples**
  - `MyDict.items() =`

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

Modes

- 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?

Find the Mode: The Algorithm

1. Create an "empty" dictionary (as initialization)
2. Go through all the numbers in the list, one at a time
    1. If the number is **not** in the dictionary,
       then create an entry in the dictionary with that number as *key* and *value* = 1.
    2. If the number is in the dictionary,
       then find that entry ; add 1 to its *value* (accumulation)
3. When done with going through the numbers, create a new list that is made up of all the values in the dictionary
4. Calculate the maximum in that new list
5. Go through all the numbers in that new list and compare each one to the maximum
6. That when you find a match, you've found the mode! (return mode)

Finding The **Mode** Of A List

- First note: there might be more than one mode!

```
def mode(alist):   # Listing 4.6 (and start of 4.7)
    countdict = {}

    for item in alist:
        if item in countdict:
            countdict[item] = countdict[item]+1
        else:
            countdict[item] = 1
```

  - Continued next slide

Finding mode (cont.)

- Rest of Listing 4.7:

```
countlist = countdict.values()
maxcount = max(countlist)

modelist = [ ]   # in case there is more than one
for item in countdict:
    if countdict[item] == maxcount:
        modelist.append(item)

return modelist
```

YOUR TO-DOs

- Finish reading **Chapter 4**

3 THINGS TO FINISH NEXT WEEK!!!

- Finish **Homework5** (due **Thursday 5/11**)
- Finish **Lab4** (due **Tuesday 5/9**)
- Keep working on **Project1** (due **Friday 5/12**)

- Let the sunshine in

</LECTURE>