CSCI 134 Fall 2021:
Nested Lists & List Methods

October 04, 2021

Shikha Singh, 9AM
Jeannie Albrecht, 10AM
Announcements & Logistics

- **Homework 4** is out on GLOW, due tonight at 10 pm
- **Lab 2** graded feedback was released last week, **Lab 3** will be returned on Wed
- **Lab 4** was released on Friday: has two parts!
  - Part 1 is due Wed/Thurs at 10 pm; Part 2 is due Oct 13/14 at 10 pm
  - No lab on Oct 11/12 (or herd meetings next weekend) due to reading period (but we will still have office hours next week!!)
  - (We’ll check about TA hours during reading period)
- **Midterm** reminder: Wed Oct 20th evening exam (more details forthcoming)
  - Time Option 1: **6 pm - 7:30 pm**
  - Time Option 2: **8 pm - 9:30 pm**
  - Let us know asap if you have any class conflicts or need accommodations (extra time accommodations should attend the early session if possible)

Do You Have Any Questions?
Homework Questions

• How to approach homework questions:
  • The best way to learn is to walk through the logic of the code using pencil and paper (without a computer)
  • Homework questions are the best practice for exams
  • But it is totally ok to test your answers with interactive python after using pencil and paper
  • Even for lab programs: always good to start by jotting your ideas and structure of code on paper, before programming
Lab Feedback: Things to Keep in Mind

- **Default return type**: Does your function always return something?
  - Suppose your function has the following input/output type
  - Input: list of strings; Output: returns a string
  - This function should always return a string and not None regardless of the input!

- Is your code as simple and elegant as can be? Or is it overly complicated?

- **Good programming style** does not necessarily mean fewest lines: it means good organization and readability
  - Variable names should be defined properly, and long names should be in camelCase
  - Make good use of white space and comments

- See CS134 Python Style guide: [https://www.cs.williams.edu/~cs134/resources/CS134_Fall2021_StyleGuide.pdf](https://www.cs.williams.edu/~cs134/resources/CS134_Fall2021_StyleGuide.pdf)
Last Time

- Discussed **file reading** using lists and strings
  - Used string methods `.strip()`, `.split()`
  - Used list methods `.append()`, `.extend()`, `.count()`
- Learned about **list comprehensions** as a way to simplify list accumulations
  - Leads to simpler, more succinct code
  - When a mapping or filter pattern comes up, list comprehensions are more elegant than defining an accumulation variable and using an explicit loop with list.append()
- Also began exploring lists of lists
Today’s Plan

• Review **CSV file reading** and accessing **lists of lists** (also mentioned in part 1 lab video this week)

• Use our knowledge about lists and loops to analyze interesting properties of our data
  • Focus on maintaining the state of variables when looping, and how to update state based on conditionals

• Review useful list methods:
  • Methods that don't modify lists: `.index()`, `.count()`
  • Methods that do modify lists:
    • `.append()`, `.extend()`, `.insert()`, `.remove()`, `.pop()`, `.sort()`
Recap: Working with CSVs

- Using a list comprehension for file reading:

```python
with open(filename) as roster:
    allStudents = [student.strip().split(',') for student in roster]
```

```
allStudents # list of lists

[['Appiah', 'Tazmin', '25AAA'],
 ['Bliven', 'Jocelyn M.', '25AAA'],
 ['Brockman', 'Annika M.', '24AAA'],
 ['Byun', 'Justin', '24AAA'],
 ['Cao', 'Yaoyue', '23AAA'],
 ['Casenave Barranguet', 'Lola', '25AAA'],
 ['Catlin', 'Tucker R.', '24AAA'],
 ['Chong', 'Andrew H.', '25AAA'],
]```
Recap: Lists of Lists!

- We have already seen lists of strings
- We can also have lists of lists!
- Sometimes called a **2D (two dimensional)** list
- Suppose we have a list of lists of strings
- \texttt{word} = \texttt{list[a][b]}
  - \texttt{a} is index into “outer” list (identifies **which list** we want)
  - \texttt{b} in index into “inner” list (identifies **the element within the list** we want)
- Let’s see an example!
Exercise: Students by Year

• Let's get to know our class better!

• Write a function `yearList` which takes in two arguments, `rosterList` (list of lists of string) and `year` (int) and returns the list of students in the class with that graduating year.

• Can we do this with a list comprehension?

```python
def yearList(rosterList, year):
    """Takes the student info as a list of lists and a year (22-25) and returns a list of students graduating that year""
    pass
```
Exercise: Students by Year

- Write a function `yearList` which takes in two arguments, `rosterList` (list of lists of string) and `year` (int) and returns the list of students in the class with that graduating year.

```python
def yearList(rosterList, year):
    """Takes the student info as a list of lists and a year (22-25) and returns a list of students graduating that year""
    return [s[1] for s in allStudents if s[-1][:-2] == str(year)]
```

keep second string in s (student’s first name)
list of strings
list of lists of strings
string that is the first 2 characters [:2] of the last element [-1] of the list of strings
Exercise: Most Vowels

- **Student Fun Facts!** Which student in the class has the most number of vowels in their name?
- How do we approach this problem?
  - What do we need to keep track of?
  - When do we update and how?
  - Useful to think about/write down a strategy BEFORE coding!
  - Not a list of lists this time...just a list of strings

```python
def mostVowels(wordList):
    '''Takes a list of strings wordList and returns a list of strings in wordList that contain the most number of vowels'''
    pass
```
**Exercise: Most Vowels**

- **Student Fun Facts!** Which student in the class has the most number of vowels in their name?

```python
def mostVowels(wordList):
    '''
    Takes a list of strings wordList and returns a list of strings from wordList that contain the most # vowels'''
    maxSoFar = 0  # initialize counter
    result = []
    for word in wordList:
        count = countVowels(word)
        if count > maxSoFar:
            # update: found a better word
            maxSoFar = count
            result = [word]
        elif count == maxSoFar:
            # why do we need this?
            result.append(word)
    return result
```
Exercise: Least Vowels

- **More Fun Facts!** Which student in the class has the least number of vowels in their name?

- What needs to change?
  - `maxSoFar` becomes `minSoFar`
  - What do we initialize `minSoFar` to?

```python
def leastVowels(wordList):
    ''' Takes a list of strings wordList and returns a list of strings in wordList that contain the least number of vowels'''
    pass
    # what needs to change?
```
Writing to Files

- We know how to **read from** files
- We can also **write to** files
- We can write (or save) all the results that we are computing into a file (a persistent structure). To open a **new** file for writing, we use `open` with the mode `'w'`.

```python
fYears = len(yearList(allStudents, 25))
sophYears = len(yearList(allStudents, 24))
jYears = len(yearList(allStudents, 23))
sYears = len(yearList(allStudents, 22))
mostVowelNames = ','.join(mostVowels(firstNames))
leastVowelNames = ','.join(leastVowels(firstNames))

with open('studentFacts.txt', 'w') as sFile:
    sFile.write('Fun facts about CS134 students:
')  # need newlines
    sFile.write('Students with most vowels in their name: {}
'.format(mostVowelNames))
    sFile.write('Students with least vowels in their name: {}
'.format(leastVowelNames))
    sFile.write('No. of first years in CS134: {}
'.format(fYears))
    sFile.write('No. of sophomores in CS134: {}
'.format(sophYears))
    sFile.write('No. of juniors in CS134: {}
'.format(jYears))
    sFile.write('No. of seniors in CS134: {}
'.format(sYears))
```
Format Printing for Python Strings

• A quick way to build strings with particular form is to use the `.format` function on them

Syntax:  `myString.format(*args)`

`*args` means it takes zero or more arguments

• For every pair of braces `{}`, format consumes one argument

• Argument is implicitly converted to a string (with `str`) and concatenated with the remaining parts of the format string

• Especially useful in printing: called format printing

```
In [8]: "Hello, you {} world{}".format("silly","!")  # creates a new string
Out[8]: 'Hello, you silly world!

In [9]: print("Hello, {}":".format("you silly world!"))
Hello, you silly world!.
```
Appending to Files

- If a file already has something in it, opening it in `w` mode again will erase all of its past contents.
- If we need to *append* something to an *existing* file without erasing the contents, we open it in append `a` mode.

```python
with open('studentFacts.txt', 'a') as sFile:
    sFile.write('Goodbye.
')```
Summary of List Methods

Methods that do not modify list:
.index(), .count()
Useful List Methods: `index()`

- **myList.index(item)**: returns the first index (int) of item in myList if it is present, else throws an error
  - Method does not modify the list it is called on

**Example.**

```python
>>> myList = ['1', '7', '3', '4', '5']
>>> myList.index('3')  # return index of 3
2
>>> myList.index('10')
ValueError
```
Useful List Methods: count()

- `myList.count([item])`: counts and returns the number of times `item` appears in `myList` (as an `int`)
- Method does not modify list it is called on

**Example.**

```python
>>> myList = [2, 3, 2, 1, 2, 4, 1]
>>> c = myList.count(2)
>>> c
3
```
Summary of List Methods

Methods that DO modify list:

.append(), .extend(), .pop(), .insert(), .remove(), .sort()
Aside: List Assignment

```
myList[index] = item : assignment can modify a list directly
```

Example.

```
myList[1] = 7  # assign 7 to index 1 of myList
```

```
myList Before
[1, 2, 3, 4]

myList After
[1, 7, 3, 4]
```
append(

`myList.append(item)` : appends item to end of list

Example.

`myList.append(5)`  # insert 5 at the end of the list

```
myList Before
[1, 7, 3, 4]

myList After
[1, 7, 3, 4, 5]
```
extend()

`myList.extend([itemList])`: appends all the items in itemList to the end of myList

**Example.**

```python
myList.extend([6, 8])  # insert both 6 and 8 at the end of the list
```

**myList Before**

```
[1, 7, 3, 4, 5]
```

**myList After**

```
[1, 7, 3, 4, 5, 6, 8]
```
**pop()**

**myList.pop(index):** Removes the item at a given index (int) and returns it. If no index is given, removes and returns the *last item* from the list.

**Example.**

```python
val = myList.pop(3)                  # returns
```  

returns `val = 4`

**myList Before**

\[1, 7, 3, 4, 5, 6, 8]\n
**myList After**

\[1, 7, 3, 5, 6, 8]\n
**pop()**

`myList.pop(index)`: Removes the item at a given index (int) and returns it. If no index is given, removes and returns the last item from the list.

**Example.**

```python
val = myList.pop()
val = 8
```

- **myList Before**: `[1, 7, 3, 5, 6, 8]`
- **myList After**: `[1, 7, 3, 5, 6]`
- **No Index**: `returns`
myList.insert(index, item): inserts item at index (int) in myList, all items to the right of index shift over to make room.

Example.

myList.insert(0, 11)  # insert 11 at index 0

Before

[1, 7, 3, 5, 6]

After

[11, 1, 7, 3, 5, 6]
myList.insert(index, item): inserts item at index (int) in myList, all items to the right of index shift over to make room

Example.

myList.insert(10, 12)  # insert 12 at index 10

[11, 1, 7, 3, 5, 6]  [11, 1, 7, 3, 5, 6, 12]
remove()"

`myList.remove(item)` removes item from myList, all items to the right removed item shift to the left by one.

(Unlike `pop()`, item is not returned!)

**Example.**

`myList.remove(12)`  # remove 12 from myList

```
myList Before
[11, 1, 7, 3, 5, 6, 12]

myList After
[11, 1, 7, 3, 5, 6]
```
sort()

`myList.sort(item):` sorts the list in place in ascending order

**Example.**

`myList.sort()`  # sort by mutating `myList`

<table>
<thead>
<tr>
<th>myList Before</th>
<th>myList After</th>
</tr>
</thead>
<tbody>
<tr>
<td>[11, 1, 7, 3, 5, 6]</td>
<td>[1, 3, 5, 6, 7, 11]</td>
</tr>
</tbody>
</table>
sort() vs sorted()

- **sort method** is only for lists and sorts by mutating the list in place.
- Python provides a built in **function sorted** that can be used to sort any sequence (strings, lists, tuples). It **returns a new sorted sequence**, and does NOT modify the original sequence!

**Example.**

```python
list1 = [6, 3, 4]; list2 = [6, 3, 4]
list1.sort() # sort by mutating list1
sorted(list2) # returns a new sorted list
```

<table>
<thead>
<tr>
<th>list1 Before</th>
<th>list1 After</th>
<th>list2 Before</th>
<th>list2 After</th>
</tr>
</thead>
<tbody>
<tr>
<td>[6, 3, 4]</td>
<td>[3, 4, 6]</td>
<td>[6, 3, 4]</td>
<td>[6, 3, 4]</td>
</tr>
</tbody>
</table>

Does not change!
Sorting Strings

- Strings are sorted based on the ASCII values of their characters.
- ASCII stands for “American Standard Code for Information Interchange”.
- Common character encoding scheme for electronic communication (that is, anything sent on the Internet!)
- Special characters come first, followed by capital letters, then lowercase.
- Characters encoded using integers from 0-127.
- Can use Python functions `ord(str)` and `chr(int)` to work with these values.
# ASCII TABLE

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Hex</th>
<th>Char</th>
<th>Decimal</th>
<th>Hex</th>
<th>Char</th>
<th>Decimal</th>
<th>Hex</th>
<th>Char</th>
<th>Decimal</th>
<th>Hex</th>
<th>Char</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>[NULL]</td>
<td>32</td>
<td>20</td>
<td>[SPACE]</td>
<td>64</td>
<td>40</td>
<td>@</td>
<td>96</td>
<td>60</td>
<td>`</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>[START OF HEADING]</td>
<td>33</td>
<td>21</td>
<td>!</td>
<td>65</td>
<td>41</td>
<td>A</td>
<td>97</td>
<td>61</td>
<td>a</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>[START OF TEXT]</td>
<td>34</td>
<td>22</td>
<td>&quot;</td>
<td>66</td>
<td>42</td>
<td>B</td>
<td>98</td>
<td>62</td>
<td>b</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>[END OF TEXT]</td>
<td>35</td>
<td>23</td>
<td>#</td>
<td>67</td>
<td>43</td>
<td>C</td>
<td>99</td>
<td>63</td>
<td>c</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>[END OF TRANSMISSION]</td>
<td>36</td>
<td>24</td>
<td>$</td>
<td>68</td>
<td>44</td>
<td>D</td>
<td>100</td>
<td>64</td>
<td>d</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>[ENQUIRY]</td>
<td>37</td>
<td>25</td>
<td>%</td>
<td>69</td>
<td>45</td>
<td>E</td>
<td>101</td>
<td>65</td>
<td>e</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>[ACKNOWLEDGE]</td>
<td>38</td>
<td>26</td>
<td>&amp;</td>
<td>70</td>
<td>46</td>
<td>F</td>
<td>102</td>
<td>66</td>
<td>f</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>[BEL]</td>
<td>39</td>
<td>27</td>
<td>'</td>
<td>71</td>
<td>47</td>
<td>G</td>
<td>103</td>
<td>67</td>
<td>g</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>[BACKSPACE]</td>
<td>40</td>
<td>28</td>
<td>(</td>
<td>72</td>
<td>48</td>
<td>H</td>
<td>104</td>
<td>68</td>
<td>h</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>[HORIZONTAL TAB]</td>
<td>41</td>
<td>29</td>
<td>)</td>
<td>73</td>
<td>49</td>
<td>I</td>
<td>105</td>
<td>69</td>
<td>i</td>
</tr>
<tr>
<td>11</td>
<td>B</td>
<td>[VERTICAL TAB]</td>
<td>43</td>
<td>2B</td>
<td>+</td>
<td>75</td>
<td>4B</td>
<td>K</td>
<td>107</td>
<td>6B</td>
<td>k</td>
</tr>
<tr>
<td>12</td>
<td>C</td>
<td>[FORM FEED]</td>
<td>44</td>
<td>2C</td>
<td>,</td>
<td>76</td>
<td>4C</td>
<td>L</td>
<td>108</td>
<td>6C</td>
<td>l</td>
</tr>
<tr>
<td>13</td>
<td>D</td>
<td>[CARRIAGE RETURN]</td>
<td>45</td>
<td>2D</td>
<td>.</td>
<td>77</td>
<td>4D</td>
<td>M</td>
<td>109</td>
<td>6D</td>
<td>m</td>
</tr>
<tr>
<td>14</td>
<td>E</td>
<td>[SHIFT OUT]</td>
<td>46</td>
<td>2E</td>
<td>,</td>
<td>78</td>
<td>4E</td>
<td>N</td>
<td>110</td>
<td>6E</td>
<td>n</td>
</tr>
<tr>
<td>15</td>
<td>F</td>
<td>[SHIFT IN]</td>
<td>47</td>
<td>2F</td>
<td>/</td>
<td>79</td>
<td>4F</td>
<td>O</td>
<td>111</td>
<td>6F</td>
<td>o</td>
</tr>
<tr>
<td>16</td>
<td>10</td>
<td>[DATA LINK ESCAPE]</td>
<td>48</td>
<td>30</td>
<td>0</td>
<td>80</td>
<td>50</td>
<td>P</td>
<td>112</td>
<td>70</td>
<td>p</td>
</tr>
<tr>
<td>17</td>
<td>11</td>
<td>[DEVICE CONTROL 1]</td>
<td>49</td>
<td>31</td>
<td>1</td>
<td>81</td>
<td>51</td>
<td>Q</td>
<td>113</td>
<td>71</td>
<td>q</td>
</tr>
<tr>
<td>18</td>
<td>12</td>
<td>[DEVICE CONTROL 2]</td>
<td>50</td>
<td>32</td>
<td>2</td>
<td>82</td>
<td>52</td>
<td>R</td>
<td>114</td>
<td>72</td>
<td>r</td>
</tr>
<tr>
<td>19</td>
<td>13</td>
<td>[DEVICE CONTROL 3]</td>
<td>51</td>
<td>33</td>
<td>3</td>
<td>83</td>
<td>53</td>
<td>S</td>
<td>115</td>
<td>73</td>
<td>s</td>
</tr>
<tr>
<td>20</td>
<td>14</td>
<td>[DEVICE CONTROL 4]</td>
<td>52</td>
<td>34</td>
<td>4</td>
<td>84</td>
<td>54</td>
<td>T</td>
<td>116</td>
<td>74</td>
<td>t</td>
</tr>
<tr>
<td>21</td>
<td>15</td>
<td>[NEGATIVE ACKNOWLEDGE]</td>
<td>53</td>
<td>35</td>
<td>5</td>
<td>85</td>
<td>55</td>
<td>U</td>
<td>117</td>
<td>75</td>
<td>u</td>
</tr>
<tr>
<td>22</td>
<td>16</td>
<td>[SYNCHRONOUS Idle]</td>
<td>54</td>
<td>36</td>
<td>6</td>
<td>86</td>
<td>56</td>
<td>V</td>
<td>118</td>
<td>76</td>
<td>v</td>
</tr>
<tr>
<td>23</td>
<td>17</td>
<td>[ENG OF TRANS. BLOCK]</td>
<td>55</td>
<td>37</td>
<td>7</td>
<td>87</td>
<td>57</td>
<td>W</td>
<td>119</td>
<td>77</td>
<td>w</td>
</tr>
<tr>
<td>24</td>
<td>18</td>
<td>[CANCEL]</td>
<td>56</td>
<td>38</td>
<td>8</td>
<td>88</td>
<td>58</td>
<td>X</td>
<td>120</td>
<td>78</td>
<td>x</td>
</tr>
<tr>
<td>25</td>
<td>19</td>
<td>[END OF MEDIUM]</td>
<td>57</td>
<td>39</td>
<td>9</td>
<td>89</td>
<td>59</td>
<td>Y</td>
<td>121</td>
<td>79</td>
<td>y</td>
</tr>
<tr>
<td>26</td>
<td>1A</td>
<td>[SUBSTITUTE]</td>
<td>58</td>
<td>3A</td>
<td>:</td>
<td>90</td>
<td>5A</td>
<td>Z</td>
<td>122</td>
<td>7A</td>
<td>z</td>
</tr>
<tr>
<td>27</td>
<td>1B</td>
<td>[ESCAPE]</td>
<td>59</td>
<td>3B</td>
<td>;</td>
<td>91</td>
<td>5B</td>
<td>[</td>
<td>123</td>
<td>7B</td>
<td>{</td>
</tr>
<tr>
<td>28</td>
<td>1C</td>
<td>[FILE SEPARATOR]</td>
<td>60</td>
<td>3C</td>
<td>&lt;</td>
<td>92</td>
<td>5C</td>
<td>]</td>
<td>124</td>
<td>7C</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>1D</td>
<td>[GROUP SEPARATOR]</td>
<td>61</td>
<td>3D</td>
<td>=</td>
<td>93</td>
<td>5D</td>
<td>\</td>
<td>125</td>
<td>7D</td>
<td>}</td>
</tr>
<tr>
<td>30</td>
<td>1E</td>
<td>[RECORD SEPARATOR]</td>
<td>62</td>
<td>3E</td>
<td>&gt;</td>
<td>94</td>
<td>5E</td>
<td>^</td>
<td>126</td>
<td>7E</td>
<td>~</td>
</tr>
<tr>
<td>31</td>
<td>1F</td>
<td>[UNIT SEPARATOR]</td>
<td>63</td>
<td>3F</td>
<td>?</td>
<td>95</td>
<td>5F</td>
<td>_</td>
<td>127</td>
<td>7F</td>
<td>[DEL]</td>
</tr>
</tbody>
</table>
5. Data Structures

This chapter describes some things you've learned about already, in more detail, and adds some new things as well.

5.1. More on Lists

The list data type has some more methods. Here are all of the methods of list objects:

```python
list.append(x)
    Add an item to the end of the list. Equivalent to `a[len(a):] = [x]`.

list.extend(iterable)
    Extend the list by appending all the items from the iterable. Equivalent to `a[len(a):] = iterable`.

list.insert(i, x)
    Insert an item at a given position. The first argument is the index of the element before which to insert, so `a.insert(0, x)` inserts at the front of the list, and `a.insert(len(a), x)` is equivalent to `a.append(x)`.

list.remove(x)
    Remove the first item from the list whose value is equal to `x`. It raises a `ValueError` if there is no such item.

list.pop(i)
    Remove the item at the given position in the list, and return it. If no index is specified, `a.pop()` removes and returns the last item in the list. (The square brackets around the `i` in the method signature denote that the parameter is optional, not that you should type square brackets at that position. You will see this notation frequently in the Python Library Reference.)

list.clear()
    Remove all items from the list. Equivalent to `del a[:]`.

list.index(x[, start[, end]])
    Return zero-based index in the list of the first item whose value is equal to `x`. Raises a `ValueError` if there is no such item.

    The optional arguments `start` and `end` are interpreted as in the slice notation and are used to limit the search to a particular subsequence of the list. The returned index is computed relative to the beginning of the full sequence rather than the `start` argument.

list.count(x)
    Return the number of times `x` appears in the list.
```