CS 134:
Nested Lists & Writing to Files
Announcements & Logistics

- **Homework 4** is out on GLOW, due tonight at 11 pm
- **Lab 4** was released on Friday: has two parts!
  - Part 1 is due Wed/Thurs at 11 pm; Part 2 is due Mar 9/10 at 11 pm
- **Midterm** reminder: Thur Mar 17 evening exam (more details forthcoming regarding format)
  - Time Option 1: 6 pm - 7:30 pm
  - Time Option 2: 8 pm - 9:30 pm
  - Two rooms (one for reduced distractions/extra time)
  - Let us know asap if you have any class conflicts or need additional accommodations
  - Extra time accommodations should attend the early session if possible

Do You Have Any Questions?
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

• Explore more **CSV file reading** and accessing **lists of lists**

• 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
  
  • Help prepare for Lab 4

• Briefly look at writing/appending to files
Recap: Lists of Lists!

- We have already seen lists of strings.
- We can also have lists of lists (sometimes called a two-dimensional list).
- Often arise when using list comprehensions.
- Suppose we have a list of lists of strings called myList.
- word = myList[a][b] (# word is a string)
  - a is index into “outer” list (identifies which inner list we want).
  - b is index into “inner” list (identifies which element within the inner list).
- Be careful with lists of lists of strings vs lists of strings.

```python
def myList:
    return [['cat', 'frog'], ['dog', 'toad'], ['cow', 'duck']]
```

```python
myList[1][0] is 'dog'
myList[1][0] is 'f'
```
Lists of Lists and Comprehensions

• Suppose we want to create a list of lists of strings using our student data.

```python
In [9]:
filename = ...
s
allStudents = ...

with open(filename) as roster:
    for student in roster:
        allStudents.append(student.strip().split(','))
```

- **item**: Each element of the sequence.
- **sequence**: The sequence of elements.
- **expression results in a list**: The result of the expression is a list.

```python
allStudents = ...
```
Lists of Lists and Comprehensions

- Suppose we want to create a list of lists of strings using our student data.

```python
In [9]:
filename = 'csv/classnames.csv'
allStudents = []
with open(filename) as roster:
    for student in roster:
        allStudents.append(student.strip().split(',','))
```

```python
In [25]:
# with a list comprehension!
filename = 'csv/classnames.csv'
with open(filename) as roster:
    allStudents = [student.strip().split(',',')) for student in roster]
```

```python
In [26]:
allStudents  # list of lists of strings
```

```
Out[26]:
[['Aleman-Valencia', 'Karla', '25', 'ka14'],
 ['Batsaikhan', 'Munguldei', '25', 'mb34'],
 ['Berger', 'Marcello W.', '25', 'mwb3'],
 ['Bertolet', 'Jeremy S.', '24', 'jsb7'],
 ['Bhaskar', 'Monika A.', '25', 'mab13'],
 ['Blair', 'Maycie C.', '25', 'mcb12'],
 ['Brown', 'Courtney A.', '22', 'cab10'],
 ['Christ', 'Alexander M.', '22', 'amc11'],
 ['Gonzalez', 'Gabriela M.', '24', 'gmg7'],
 ['Herman', 'Adelaide A.', '25', 'aah6'],
 ['Hu', 'Jess', '25', 'jhh3'],
```
More List Comprehensions

allStudents:

- Generate list of only last names using allStudents

```
In [28]: # generate list of only student last names
   ...: lastNames = [s[0] for s in allStudents]
   ...: lastNames

Out[28]: ['Aleman-Valencia', 'Batsaikhan', 'Berger', 'Bertolet', 'Bhaskar',

- Generate list of only first names

```
In [29]: # List comprehension to generate a list of first names
   ...: # (without middle initial)
   ...: firstNames = [s[1].split()[0] for s in allStudents]
   ...: firstNames

Out[29]: ['Karla', 'Munguldei', 'Marcello', 'Jeremy', 'Monika',

split() first name, return first element (effectively removes middle initial)
Exercise: Student Fun Facts!

- Write a function `characterList` which takes in two arguments `rosterList` (list of lists) and `character` (a string) and returns the list of students in the class whose first name starts with character.

- Can we do this with a list comprehension?

```python
In [30]: def characterList(rosterList, character):
    """ Takes the student info as a list of lists and a string character and returns a list of students whose first name starts with character""
    return [name[1] for name in rosterList if name[1][0] == character]

In [31]: characterList(allStudents, "B")
Out[31]: ['Brandon', 'Bailey C.', 'Bernard V.']
```
Exercise: Student Fun Facts!

• Write a function `mostVowels` that can be used to compute the list of students with the most vowels in their first name. (Hint: use `countVowels()`.)

```
In [32]: 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:
           result.append(word)
   return result

In [33]: # which student(s) has most vowels in their name?
mostVowelNames = mostVowels(firstNames)
mostVowelNames

Out[33]: ['Adelaide', 'Giuliana']
```
Exercise: Student Fun Facts!

- Write a function `leastVowels` that can be used to compute the list of students with the least vowels in their first name. (Hint: use `countVowels()`.)

```python
In [35]: def leastVowels(wordList):
    """Takes a list of strings wordList and returns a list of strings in wordList that contain the least number of vowels'"""
    minSoFar = len(wordList[0])  # initialize counter
    result = []
    for word in wordList:
        count = countVowels(word)
        if count < minSoFar:
            # update: found a better word
            minSoFar = count
            result = [word]
        elif count == minSoFar:
            result.append(word)
    return result

In [36]: leastVowels(firstNames)
Out[36]: ['Jess', 'Will', 'Pat', 'Chan', 'Sam', 'Dan', 'Will', 'Tyler', 'Zach', 'Josh', 'Harry']
```
Exercise: Student Fun Facts!

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

```python
In [58]: 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 [name[1] + " " + name[0] for name in rosterList if name[2] == str(year)]

In [59]: juniors = yearList(allStudents, 23)
juniors
Out[59]: ['Brandon Paguada',
         'Bailey C. Burger-Moore',
         'Claudia V. Cantin',
         'Kaiser A. Garcia',
         'Oliver E. Hall',
         'Marla Khishigsuren',
         'Sebastian X. Van Der Weide']
```
An Aside: Writing to Files

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

```python
In [65]:
   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: {}
    sFile.write('Students with least vowels in their name: {}
    sFile.write('No. of first years in CS134: {}
    sFile.write('No. of sophomores in CS134: {}
    sFile.write('No. of juniors in CS134: {}
    sFile.write('No. of seniors in CS134: {}')
```

Format Printing for Python Strings

• A convenient way to build strings with particular form is to use the `.format()` string method

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* and concatenated with the remaining parts of the format string

• Especially useful in printing to files

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 \texttt{w} mode again will erase all of its past contents.

- We can also \texttt{append} something to an \texttt{existing} file without erasing the contents. To do that we open in append \texttt{a} mode.

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

```
In [63]: cat studentFacts.txt

Fun facts about CS134 students:
Students with most vowels in their name: Adelaide, Giuliani.
No. of first years in CS134: 48.
No. of sophmores in CS134: 19.
No. of juniors in CS134: 7
No. of seniors in CS134: 3
Goodbye.
```
Lab 4
Lab 4 Goals

• In Lab 4 you will implement several voting algorithms and helpful functions for manipulating election data

• Lab 4 will give you experience with:
  • Lists of strings
  • Lists of lists of strings
  • Loops
  • Using string and list methods
  • File reading

• Pay close attention to expected input (lists of strings, list of lists of strings, etc) and expected output
Ballot Data

- Ballot data is represented in various text files
- Each line represents a single voter’s ranked choices

```python
In [44]: # different types of coffee
    filename = 'csv/coffee.csv'
    with open(filename) as coffeeTypes:
        allCoffee = []
        for coffee in coffeeTypes:
            allCoffee.append(coffee.strip().split(','))
    allCoffee
```

```
Out[44]: [['kona', 'dickason', 'ambrosia', 'wonderbar', 'house'],
         ['kona', 'house', 'ambrosia', 'wonderbar', 'dickason'],
         ['kona', 'ambrosia', 'dickason', 'wonderbar', 'house'],
         ['kona', 'ambrosia', 'wonderbar', 'dickason', 'house'],
         ['house', 'kona', 'dickason', 'wonderbar', 'ambrosia'],
         ['kona', 'house', 'dickason', 'ambrosia', 'wonderbar'],
         ['kona', 'house', 'dickason', 'ambrosia', 'wonderbar'],
         ['dickason', 'ambrosia', 'wonderbar', 'kona', 'house'],
         ['house', 'kona', 'ambrosia', 'dickason', 'wonderbar'],
         ['ambrosia', 'house', 'wonderbar', 'kona', 'dickason'],
         ['wonderbar', 'ambrosia', 'kona', 'house', 'dickason'],
         ['house', 'wonderbar', 'kona', 'ambrosia', 'dickason']]
```
Working with Ballot Data

You’ll use string and list methods to process the data and implement several different voting algorithms.