CS 134:
Files & List Comprehensions
Announcements & Logistics

- **Homework 4** due next Mon at 11 pm
- **Lab 4** posted today
  - Two week lab!
  - Automated feedback returned after Part 1 (due next week at usual time)
    - You can fix your mistakes!
  - We’ll grade everything after you submit Part 2 on second week
  - Gain experience with lists, strings, file reading, lists of lists

Do You Have Any Questions?
Last Time

- Learned about adding items to lists using `+`, `append()`, and `extend()`
- Began thinking about side effects of mutability in lists
- Summarized important string and list methods and operations (so far)
- Sequence operators and functions
- String methods
- List methods
- Looked at `ranges` as an easy way to generate numerical sequences

```
In [4]: list(range(10))
Out[4]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```
Today’s Plan

- Discuss **file reading** using lists and strings
- Learn about **list comprehensions** as a way to simplify list accumulations
- Introduce lists of lists (aka 2D lists)
Reading Data from Files
Working with Files in Python

- File I/O is a very common and important operation

- `open(filename, mode)` is a built-in Python function for working with files
  - `filename` is a path to a file as a string
  - `mode` is a string where
    - `'r'` - open for reading (default)
    - `'w'` - open for writing (will overwrite previous contents)
    - `'a'` - open for appending (will not overwrite previous contents)

- Whenever you open a file, you must also close it to avoid memory leaks

- We will use the `with open ... as` code block, which keeps the file open within it, and automatically closes the file after exiting the block

- We can iterate over the lines of a text file just as we iterated over strings and lists in previous lectures
Opening Files: `with ... as`

```python
with open(filename) as inputFile:
    # do something with file
```

```python
f = open(filename, 'r')
... file operations involving f ...
f.close()
```

**Note.** *(syntax)* Indentation defines the body of the `with` block where the file is open.
Iterating over Lines in a File

- Within a `with open(filename) as inputFile:` block, we can iterate over the lines in the file just as we would iterate over any sequence such as lists, strings, or ranges.

- The end of a line in the text file is determined by the special newline character `\n`.

- Example: We have a text file `mountains.txt` within a directory `textfiles`, so we can iterate and print each line as follows:

```python
In [1]:
# read input file and print each line
with open('textfiles/mountains.txt') as book:
    for line in book:
        print(line.strip())
```

```
O, proudly rise the monarchs of our mountain land,
With their kingly forest robes, to the sky,
Where Alma Mater dwelleth with her chosen band,
And the peaceful river floweth gently by.

The mountains! The mountains! We greet them with a song,
Whose echoes rebounding their woodland heights along,
Shall mingle with anthems that winds and fountains sing,
Till hill and valley gaily gaily ring.
```
Common File Type: CSVs

- A CSV (Comma Separated Values) file is a specific type of plain text file that stores “tabular” data.
- Each row of a table is a line in the text file, with each column on the row separated by commas.
- This format is a common import and export format for spreadsheets and databases.

<table>
<thead>
<tr>
<th>Name</th>
<th>Age</th>
</tr>
</thead>
<tbody>
<tr>
<td>Harry</td>
<td>14</td>
</tr>
<tr>
<td>Hermione</td>
<td>14</td>
</tr>
<tr>
<td>Dumbledore</td>
<td>60</td>
</tr>
</tbody>
</table>

**CSV form:**
Name,Age
Harry,14
Hermoine,14
Dumbledore,60
Working with CSVs

- Since CSVs are just text files, we can process them in the same way
- Might require additional post-processing/splitting using string methods

```python
In [2]: filename = 'csv/classnames.csv'
with open(filename) as roster:
    for line in roster:
        print(line.strip())
```

Lastname, firstname, unix

Aleman-Valencia, Karla, kal14
Batsaikhan, Munguldei, mb34
Berger, Marcello W., mwb3
Bertolet, Jeremy S., jsb7
Bhaskar, Monika A., mab13
Blair, Maycie C., mcb12
Brown, Courtney A., cab10
Christ, Alexander M., amc11
Gonzalez, Gabriela M., gmg7
Herman, Adelaide A., aah6
Hu, Jess, jhh3
Huang, Will, wh4
Jain, Divij, dj4
Kirtane, Jahnavi N., jnk1
Kluev, Varya A., vak1
Klugman, Pat T., ptk2
Knight Garcia I, Grace P., gpk1
Useful String and List Methods in File Reading

• Now that we know how to read files, we can use our favorite list and string methods to work with the data

• `line.strip()`: Remove any leading/trailing white space or “
”

• `line.split(', ')`: Separate a `comma-separated` sequence of words

• `' '.join(line.split(','))`: Create a single “big” string with words separated by spaces instead of commas

• `myList.extend()`: Create lists of words while iterating over the file

• `myList.count(ele)`: Count the occurrence of various elements

• ...and so on!
Data Analysis

- Some examples (more on Jupyter!)

```python
In [4]:
# if we want to create one big list of the words, we can accumulate
# in a list using the extend() method
wordList = []
with open('textfiles/mountains.txt') as book:
    for line in book:
        wordList.extend(line.strip().split())

In [6]:
wordList

Out[6]:
['O,',
 'proudly',
 'rise',
 'the',
 'monarchs',
 'of',
 'our',
 'mountain',

In [7]:
len(wordList)

Out[7]: 133

In [5]:
# number of times a word ('mountains!') is in the song?
wordList.count('mountains!')

Out[5]: 4
```
Data Analysis w/ CSVs

• Convert our last, first, unix CSV (snippet shown below) into a list of names

```python
In [8]:
students = []
filename = "csv/classNames.csv"
with open(filename) as roster:
    for line in roster:
        fullName = line.strip().split(',
firstName = fullName[1]
lastName = fullName[0]
# print(firstName, lastName)
students.append(firstName + ' ' + lastName)
```

```python
In [9]:
students
```

Out[9]:

```
['Karla Aleman-Valencia',
'Munguldei Batsaikhan',
'Marcello W. Berger',
'Jeremy S. Bertolet',
'Monica A. Bhaskar',
'Maycie C. Blair',
'Courtney A. Brown',
'Alexander M. Christ',
'Gabriela M. Gonzalez',
'Adelaide A. Herman',
'Jess Hu',
```
List Patterns: Map & Filter

- When working with files, it is common to store data in lists
  - When processing lists, there are common patterns that appear

- **Mapping**: Iterate over a list and return a new list that results from performing an operation on each element of a given sequence (list)
  - E.g., take a list of integers `numList` and return a new list which contains the square of each number in `numList`

- **Filtering**: Iterate over a list and return a new list that results from keeping only those elements of the list that satisfy some condition
  - E.g., take a list of integers `numList` and return a new list which contains only the even numbers in `numList`

- Python allows us to implement these patterns succinctly using list comprehensions
List Comprehensions

Mapping List Comprehension (perform operation on each element)

\[
\text{newList} = [\text{expression for item in sequence}]
\]

Filtering List Comprehension (only keep some elements)

\[
\text{newList} = [\text{item for item in sequence if conditional}]
\]

Important points:

- List comprehensions always start with an expression (even a variable name like “item” is an expression!)
- We never use `append()` in list comprehensions
- We can combine mapping and filtering into a single list comprehension:

\[
\text{newList} = [\text{expression for item in sequence if conditional}]
\]
Dissecting List Comprehensions

newList = [expression for item in sequence if conditional]

Task: Extract even numbers from a range and create a list of their squares.

Using a list comprehension:

result = []
for n in range(10):
    if n%2 == 0:
        result.append(n**2)

result = [n**2 for n in range(10) if n%2 == 0]

expression: n**2
item: n
sequence: range(10)
conditional: n%2 == 0

All list comprehensions can be rewritten using a for loop!
Using List Comprehensions

- **List comprehensions** are often convenient when working with files.

- Recall our list of student names from before:

  ```
  In [9]: students
  Out[9]: ['Karla Aleman-Valencia',
           'Munguldei Batsaikhan',
           'Marcello W. Berger',
  ```

- Example: How can we find the list of student names that begin with a vowel? (Hint: we’ll use our `isVowel()` function again from before)

  - Idea:
    - Iterate over students (list of strings)
    - For each name in list, check if first letter is a vowel
    - If it is, add name to result list
Using List Comprehensions

- **List comprehensions** are often convenient when working with files.

- Recall our list of student names from before.

  ```python
  In [9]: students
  Out[9]: ['Karla Aleman-Valencia',
           'Munguldei Batsaikhan',
           'Marcello W. Berger',
          ]
  ```

- Example: How can we find the list of student names that begin with a vowel? (Hint: we’ll use our `isVowel()` function again from before.)

  ```python
  In [21]: vowelNames = [name for name in students if isVowel(name[0])]
     vowelNames
  Out[21]: ['Alexander M. Chinn',
           'Adelaide A. Herman',
           'Owen A. Kolean',
           'Andrew W. Loftus',
           'Abraham S. Park',
           'Isabella G. Polanco',
           'Alison Y. Zhang',
           'Elissa J. Berger',
           'Avery G. Freund',
           'Annie H. Gustafson',
           'Oliver E. Hall',
           'Eddie G. Loyd']
  ```
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).

```python
myList = [['cat', 'frog'],
          ['dog', 'toad'],
          ['cow', 'duck']]
```

`myList[1][0]`?

`'dog'`
We Don’t Talk About Bruno Data Types

- Python is a loosely typed programming language
- We don’t explicitly declare data types of variables
- But like Bruno, the creepy uncle in Encanto who lurks behind the walls and predicts the future, data types are always there
- It’s important to make sure we pay attention to what a function expects, especially with lists and strings! (remember this in Lab 4)
- Lists of lists of strings versus list of strings:

```python
myList = [ ['cat', 'frog'], ['dog', 'toad'], ['cow', 'duck'] ]
myList[1][0] is 'dog'

myList = [ 'cat', 'frog', ['dog', 'toad'], ['cow', 'duck'] ]
myList[1][0] is 'f'
```
Data Analysis

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

```python
In [14]: allStudents # a list of lists of strings!
```

```
```
More Examples!

• Let’s write some functions to answer questions about student names
• Student fun facts! :-)
