CS 134:
Files & List Comprehensions

Announcements & Logistics

- **Homework 4** will be released today at noon, due next Mon at 10 pm
- **Lab 3** due today 10 pm/ tomorrow 10 pm
  - Any questions?
- Mountain Day coming up (?)
    - Herd meetings will happen regardless of Mountain Day
- Lots of help hours! Come by to work on the lab
- Today's hours: 12:30-2:30 (Shikha), 1-3 pm (Jeannie), 4-6 pm and 7-11 (TAs)

Do You Have Any Questions?

Last Time

- Learned about list **accumulations**
- Discussed **nested** for loops
- Looked at **ranges** as an easy way to generate numerical sequences
- Learned about adding items to lists using **+** and **append()**
- Summarized important string and list methods and operations

Summarizing Mutability in Strings vs Lists

Strings are immutable

- Once you create a string, it cannot be changed!
- All functions that we have seen on strings ***return a new string*** and ***do not modify*** the original string

Lists are mutable

- Lists are mutable (or changeable) sequences
- You can concatenate items to a list using +, but this ***does not*** change the list
- You can append items using append() method, and this ***does*** change the list

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

Reading Files: Open

- `open(filename, mode)` returns a file object
    - `filename` is a path to a file as a string
    - `mode` is a string where
        - `'r'` - open for reading (default)
        - We will only look at this mode today
- Technically when you open a file, you must also close it to avoid memory leaks
- To avoid writing code to explicitly open and close files, we will use the `with open … as` code block, which keeps the file open within it, and closes the file after existing the block
- Today's focus: **iterable** file objects
    - We will see how to iterate over the lines of a file just as we iterated over strings and lists in previous lectures

Reading Files: `with … as`

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

Path to file on computer as a string

Variable name for your file object

Note. **(syntax)** Indentation defines the body of the with block where the file is open

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

⇔

```
with open(filename, 'r') as f:
        ... file operations involving f ...
        # f implicitly closed
        # when with is done.
```

Image Source: (http://cs111.wellesley.edu/spring19)

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
- A line in the file is determined by the special newline character `'\n'`
- For us visually, a line has the regular meaning
- Example: There is a text file `prideandprejudice.txt` within a directory `textfiles`, so we can iterate and print each line:

```
with open('textfiles/prideandprejudice.txt') as book:
    for line in book:
        print(line)
```

Variable name for your file object

Path to file on computer as a string

String Methods in File Reading

- When iterating over the lines of a file, the line variable will be a string ending in a special newline character `'\n’`
- How can we remove any leading/trailing white space (including ‘\n’)?
  - `line.strip()`
- Suppose the line in the file is a **space-separated** sequence of words. How can we collect each word in a list?
  - `line.split()`
- Suppose the line in the file is a **comma-separated** sequence of words. How can we remove commas and create a single “big” string with words separated by spaces instead of commas?
  - `' '.join(line.split(','))`

Useful List Methods: extend()

- We have already discussed **`myList.append(item)`** for adding items to a list one at a time
- `myList.extend([itemList])`: ***appends all the items*** in `itemList` to the end of **`myList`**
    - Method **modifies** the list it is called on, does not create a new list!

Example.

```
>>> myList = [1, 7, 3, 4, 5]
>>> myList.extend([6, 8]) # no return val
>>> myList
[1, 7, 3, 4, 5, 6, 8]
```

- Will see more list methods in the coming lectures, and continue to discuss mutability in more detail

Useful List Methods: count()

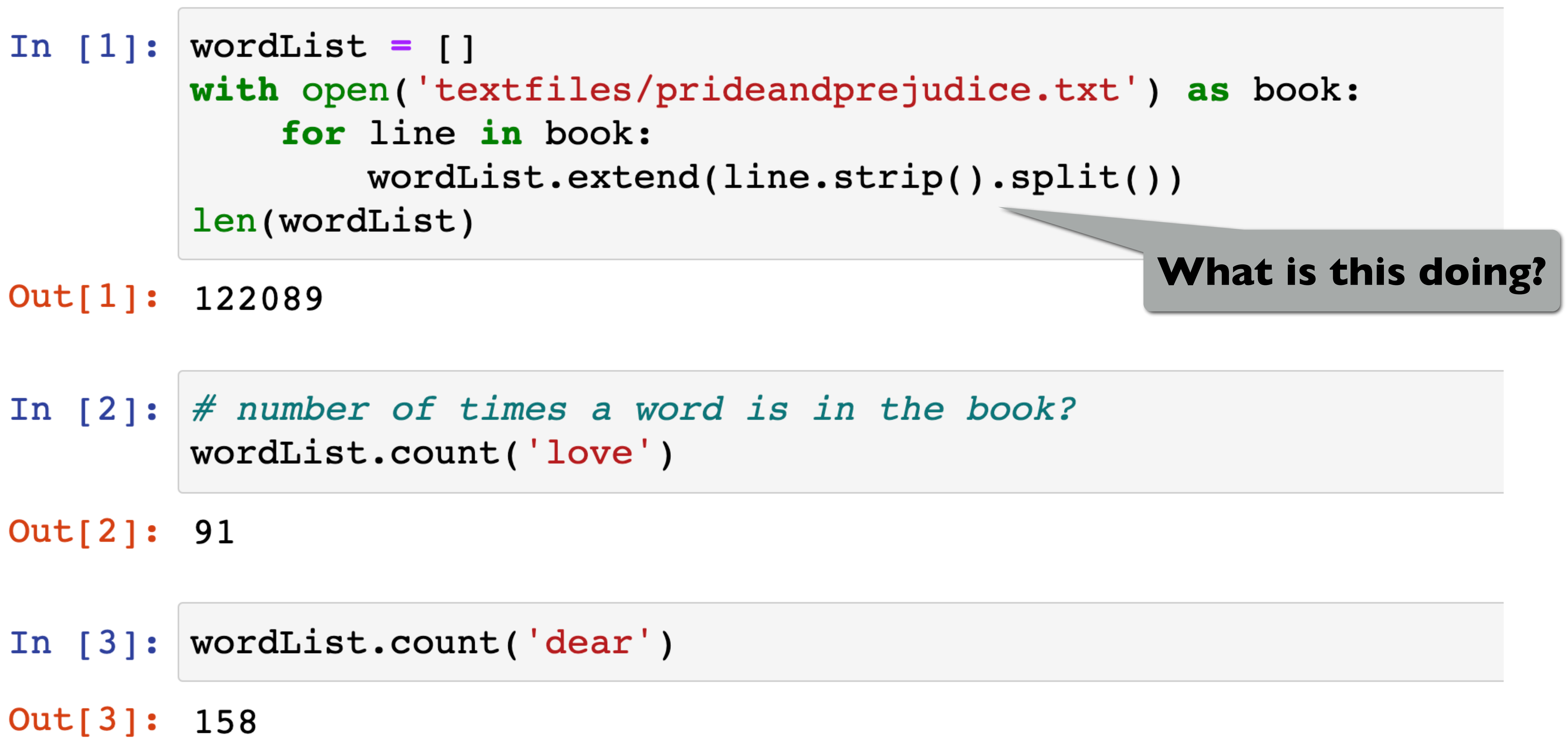
- `myList.count([item])`: counts and returns the number (an int) of times `item` appears in `myList`
- Method does not modify list it is called on

Example.

```
>>> myList = [2, 3, 2, 1, 2, 4, 1]
>>> c = myList.count(2)
>>> c
3
>>> myList
[2, 3, 2, 1, 2, 4, 1]
```

Analyzing Data Files

- How many words are in Pride and Prejudice?
- Can we count specific words?



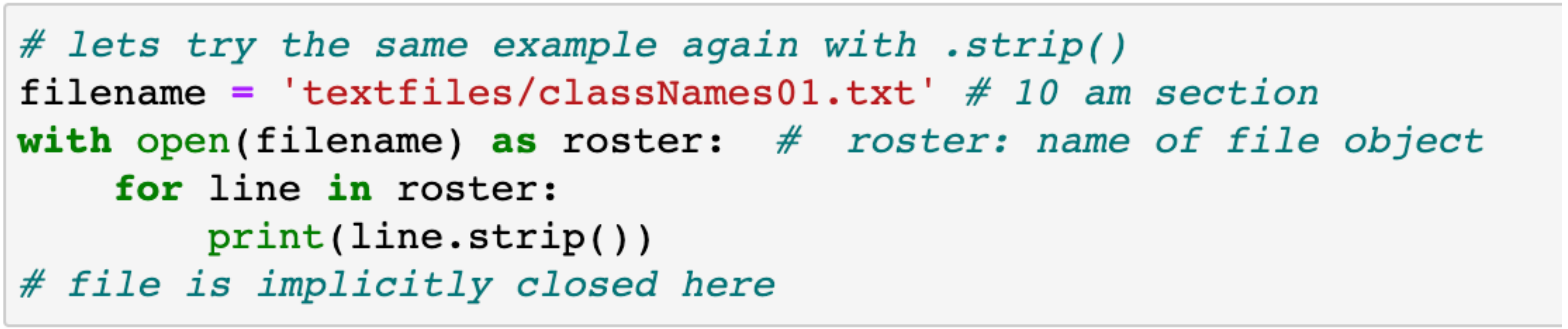
```
In [1]:  wordList = []
         with open('textfiles/prideandprejudice.txt') as book:
             for line in book:
                 wordList.extend(line.strip().split())
         len(wordList)

Out[1]:  122089
```

What is this doing?

```
In [2]:  # number of times a word is in the book?
         wordList.count('love')

Out[2]:  91

In [3]:  wordList.count('dear')

Out[3]:  158
```

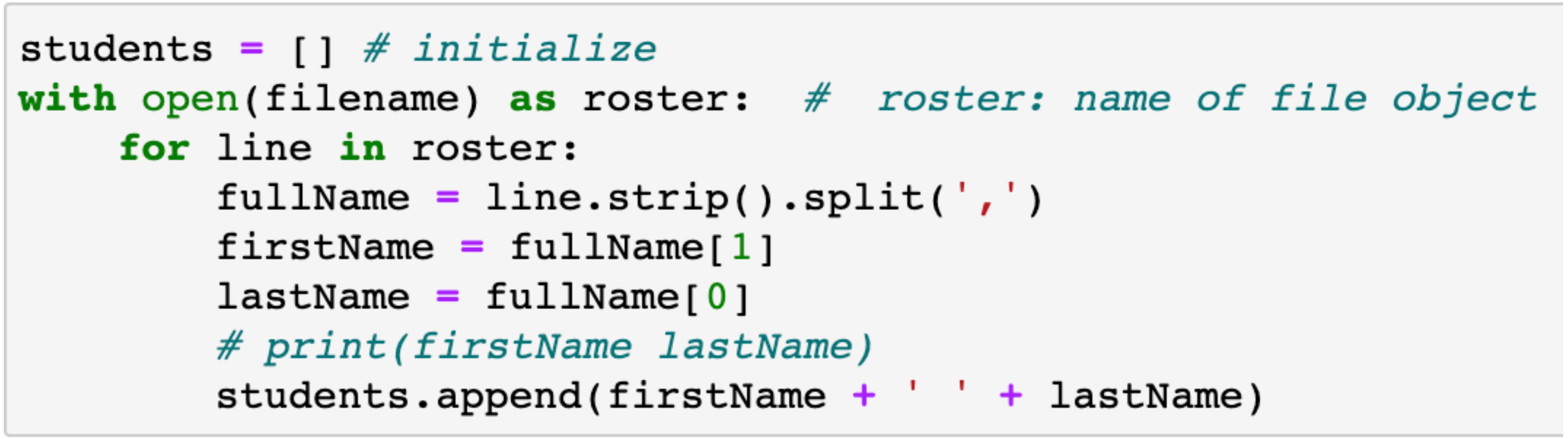
More Data Analysis

- Suppose we want to simply print each line in the file



```python
# lets try the same example again with .strip()
filename = 'textfiles/classNames01.txt' # 10 am section
with open(filename) as roster:  #  roster: name of file object
    for line in roster:
        print(line.strip())
# file is implicitly closed here
```

- Prints in lastName,firstName format
- How do we create a list of 'firstName (MI) lastName'?



```python
students = [] # initialize
with open(filename) as roster:  #  roster: name of file object
    for line in roster:
        fullName = line.strip().split(',')
        firstName = fullName[1]
        lastName = fullName[0]
        # print(firstName lastName)
        students.append(firstName + ' ' + lastName)
```

List Patterns: Map & Filter

- 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 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**

Image Source: (http://cs111.wellesley.edu/spring19)

List Comprehensions

List Comprehension for Mapping (perform an op on each element)

```
newSequence = [expression for item in sequence]
```

List Comprehension for Filtering (only keep some elements)

```
newSequence = [item for item in sequence if conditional]
```

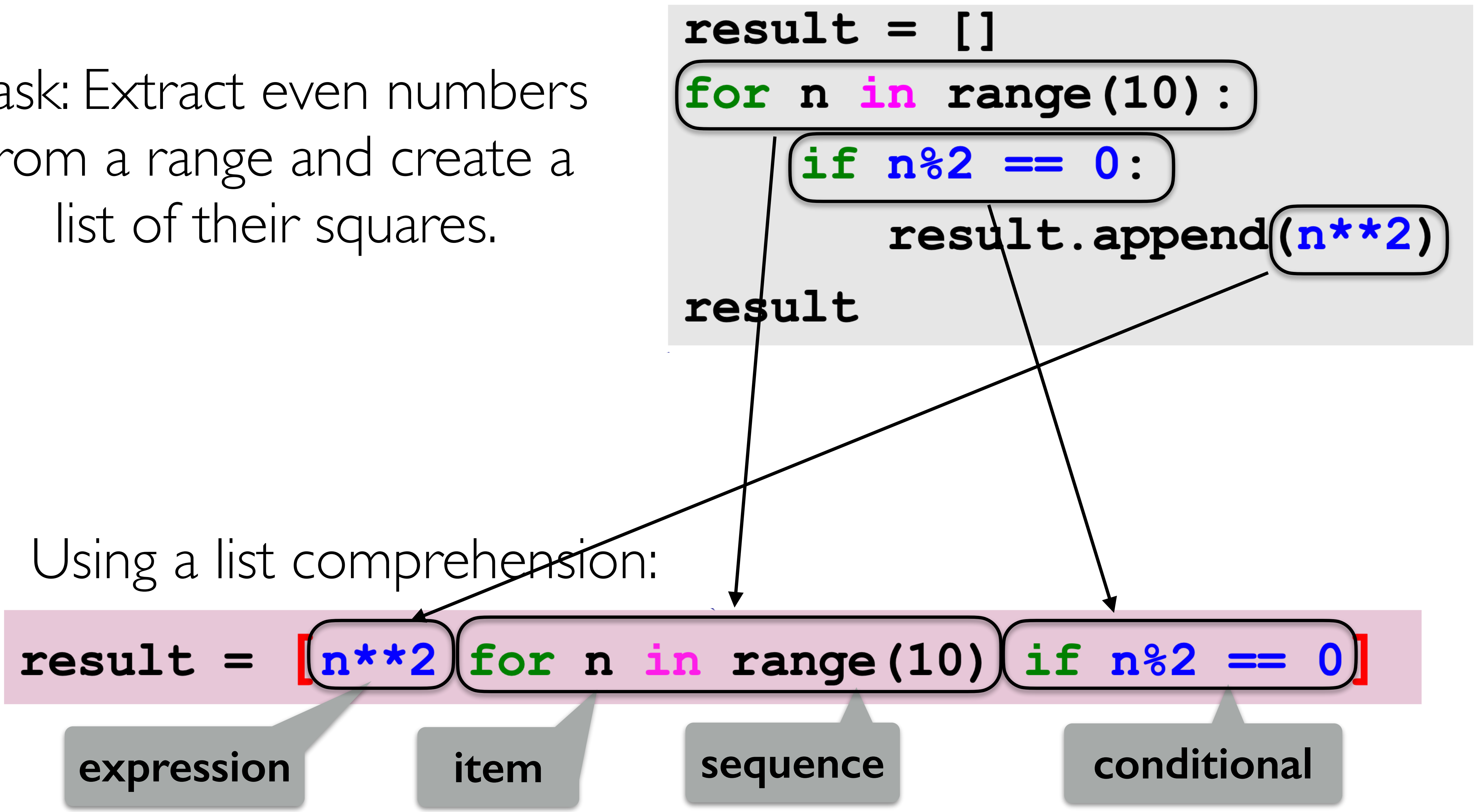
- Important points:
    - List comprehensions always start with an expression (even a variable like "item" is an expression)
    - We never use append() in list comprehensions
    - We can combine mapping and filtering into a single list comprehension:

```
newSequence = [expression for item in sequence if conditional]
```

List Comprehensions: Mapping & Filtering

```
newSequence = [expression for item in sequence if conditional]
```

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



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

Using a list comprehension:

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

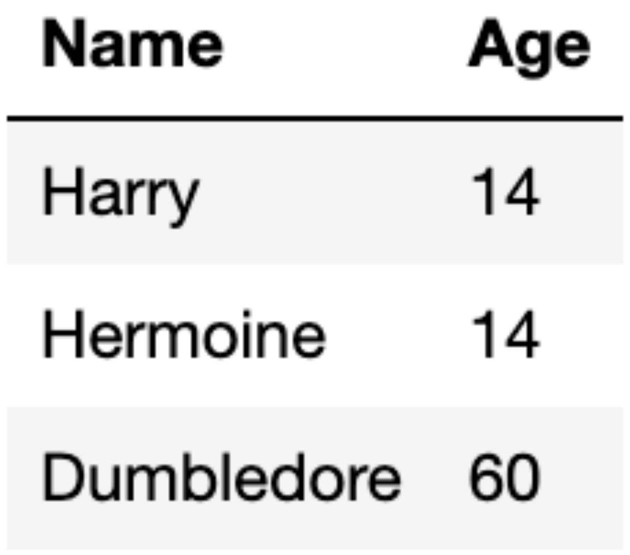
Image Source: (http://cs111.wellesley.edu/spring19)

More Data Analysis

- Let's use some of the functions we've written recently and list comprehensions to answer some more questions about data
- (See Jupyter notebook!)

Common File Type: CSVs

- A CSV (Comma Separated Values) file is a 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 the most common import and export format for spreadsheets and databases.



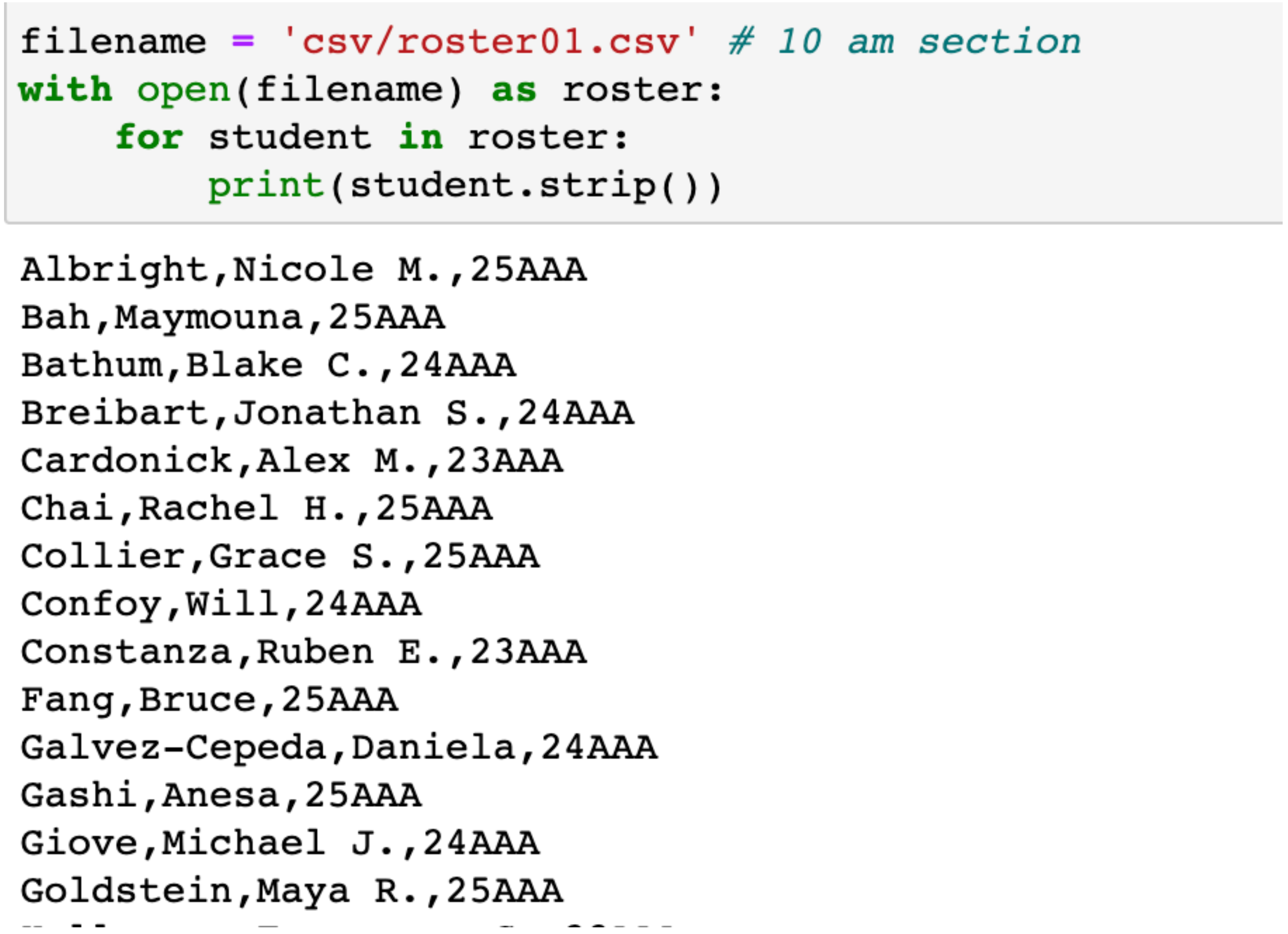
Name	Age
Harry	14
Hermoine	14
Dumbledore	60

CSV form:

```
Name,Age
Harry,14
Hermoine,14
Dumbledore,60
```

Working with CSVs

- Let's start by looking at our data file:

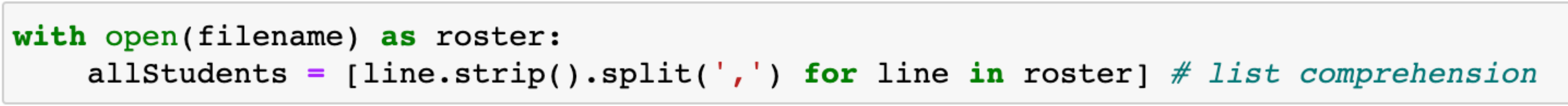


```python
filename = 'csv/roster01.csv' # 10 am section
with open(filename) as roster:
    for student in roster:
        print(student.strip())
```

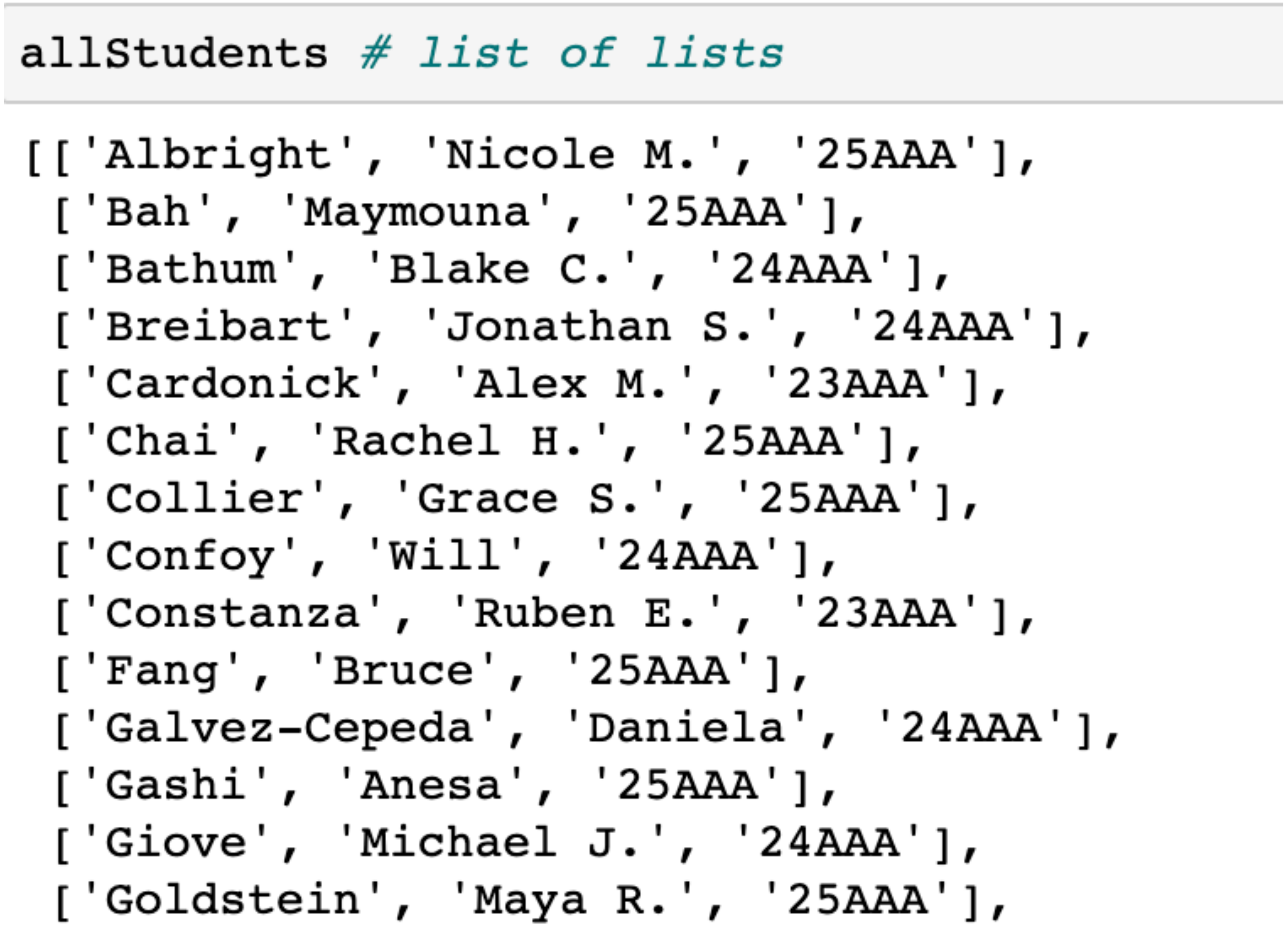
```
Albright,Nicole M.,25AAA
Bah,Maymouna,25AAA
Bathum,Blake C.,24AAA
Breibart,Jonathan S.,24AAA
Cardonick,Alex M.,23AAA
Chai,Rachel H.,25AAA
Collier,Grace S.,25AAA
Confoy,Will,24AAA
Constanza,Ruben E.,23AAA
Fang,Bruce,25AAA
Galvez-Cepeda,Daniela,24AAA
Gashi,Anesa,25AAA
Giove,Michael J.,24AAA
Goldstein,Maya R.,25AAA
```

Working with CSVs

- Using a list comprehension instead:



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



```python
allStudents # list of lists
```

```
[['Albright', 'Nicole M.', '25AAA'],
 ['Bah', 'Maymouna', '25AAA'],
 ['Bathum', 'Blake C.', '24AAA'],
 ['Breibart', 'Jonathan S.', '24AAA'],
 ['Cardonick', 'Alex M.', '23AAA'],
 ['Chai', 'Rachel H.', '25AAA'],
 ['Collier', 'Grace S.', '25AAA'],
 ['Confoy', 'Will', '24AAA'],
 ['Constanza', 'Ruben E.', '23AAA'],
 ['Fang', 'Bruce', '25AAA'],
 ['Galvez-Cepeda', 'Daniela', '24AAA'],
 ['Gashi', 'Anesa', '25AAA'],
 ['Giove', 'Michael J.', '24AAA'],
 ['Goldstein', 'Maya R.', '25AAA'],
```

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
- word = list[a][b]
    - a is index into "outer" list (identifies which list we want)
    - b in index into "inner" list (identifies the element within the list we want)
- Let's see an example!