CS134:
Files & List Comprehensions
Announcements & Logistics

- **Homework 4** due next Mon at 10 pm
- **Lab 2** feedback coming soon
- **Lab 3** due today/tomorrow
  - Lots of student help hours today/tomorrow if you need help!

Do You Have Any Questions?
Last Time

• Learned about **nested for loops**
• Summarized important **string and list methods and operations**
  • Sequence operators and functions: +, [], [:], *, etc
  • Work on strings and lists!
• String methods: `.lower()`, `.upper()`, `.join()`, `.split()`
• List methods: `.append()`, `.extend()`
Today’s Plan

• Review adding items to lists using +, append(), and extend()

• Begin thinking about side effects of mutability in lists

• Briefly discuss ranges: as an easy way to generate numerical sequences

• Discuss file reading using lists and strings (like readWords() from lab)

• Learn about list comprehensions as a way to syntactically simplify list accumulations
Recap: Modifying Lists

- Lists are **mutable** data structures
  - We can *change* them (delete things from them, add things to them, etc.)
- List **concatenation** (using `+`) *creates a new list* and *does not modify* (or *mutate*) any existing list
  - **Important point:** Concatenating to a list using `+` returns a new list!

- Alternatively we can **append to a list** using a special list method
  - The list method `myList.append(item)` **modifies** the list `myList` by adding `item` to it at the end
  - Often more efficient to append rather than concatenate! (But we have to be very careful when modifying the list)
  - **Important point:**Appending to a list modifies the existing list!
Adding elements to a List

• Here are a few examples that show how to use the list `.append()` method vs `+` operator to add items to the end of an existing list:

```python
# Original list
numList = [1, 2, 3, 4, 5]

# Concatenation
numList + [6]
# [1, 2, 3, 4, 5, 6]  # this is a new list!

# Append method
numList.append(6)
# numList has been updated to include 6
# [1, 2, 3, 4, 5, 6]
```
More Useful List Methods

- `myList.extend(itemList)`: *appends all items* in `itemList` to the end of `myList` (modifying `myList`)
- `myList.count(item)`: counts and returns the number (int) of times `item` appears in `myList`
- `myList.index(item)`: returns the first index (int) of `item` in `myList` if it is present, else throws an error

```python
myList = [1, 7, 3, 4, 5]

myList.extend([6, 4])

myList

[1, 7, 3, 4, 5, 6, 4]

myList.count(4)

2

myList.index(3)

2

myList.index(10)

ValueError
<ipython-input-38-14d2e386c720:4> 1 myList.index(10)

ValueError: 10 is not in list
Summarizing Mutability in Strings vs Lists

Strings are immutable

- Once you create a string, it cannot be changed!
- All operations 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
Ranges
Moving on: Ranges (another sequence!)

• Python provides an easy way to iterate over numerical sequences using **ranges, another sequence data type**

• When the `range()` function is given two integer arguments, it returns a **range object** of all integers starting at the first and up to, *but not including*, the second; if the first integer is 0, it may be omitted.

• To see the values included in the range, we can pass our range to the `list()` function which returns a **list** of them

```
range(0, 10)
```

```
range(0, 10)
```

```
type(range(0, 10))
```

```
range
```

```
list(range(0, 10))
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
list(range(10))
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```
Moving on: Ranges (another sequence!)

• Python provides an easy way to iterate over numerical sequences using **ranges, another sequence data type**

• When the `range()` function is given two integer arguments, it returns a **range object** of all integers starting at the first and up to, **but not including**, the second; if the first integer is 0, it may be omitted.

• To see the values included in the range, we can pass our range to the `list()` function which returns a **list** of them.

```
range(0, 10)  # Argument 1 is not provided, defaults to 0
range(0, 10)  # Argument 1 is not provided, defaults to 0
range(0, 10)  # Argument 1 is not provided, defaults to 0

A range is a type of sequence in Python (like string and list)

To see elements in range, pass range to list() function
```

```
list(range(0, 10))  # Provided
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
list(range(0, 10))  # Provided
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
list(range(10))  # Provided
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
list(range(0, 10))  # Provided
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

First argument omitted, defaults to 0
Loops and Ranges to Print Patterns

- Sometimes we might use a **for loop**, not to iterate over a sequence, but just to **repeat** a task. The following loops print a pattern to the screen. (Look closely at the indentation!)

```python
# what does this print?  # what does this print?

for i in range(5):
    print('$' * i)
for j in range(5):
    print('*' * j)
```

What are the values of i and j???
Iterating Over Ranges

# what does this print?

```python
for i in range(5):
    print('$' * i)
for j in range(5):
    print('*' * j)
```

# what does this print?

```python
for i in range(5):
    print('>' * i)
    for j in range(i):
        print('*' * j)
```
Iterating Over Ranges

```python
# what does this print?
for i in range(5):
    print('$' * i)
for j in range(5):
    print('*' * j)
```

```python
# what does this print?
for i in range(5):
    print('$' * i)
    for j in range(i):
        print('*' * i)
```

i, not j!
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)

• Using `open()` within a `with ... as` code block, we can `iterate` over the `lines of a text file` just as we iterated over strings and lists in previous lectures
Opening Files: \texttt{with} \ ... \ \texttt{as}

\begin{itemize}
  \item Path to file on computer as a string
  \item \texttt{with open(\textit{filename}) as \textit{inputFile}:}
  \item \hspace{1cm} \# do something with file
  \item Variable name for your file
  \item \textbf{Note. (syntax)} Indentation defines the body of the with block where the file is open. File automatically closed after with...as block.
\end{itemize}
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`. We can iterate and print each line as follows:

```python
# 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.

**CSV form:**
- Name,Age
- Marcel the Shell,4
- Nana Connie,70
- Mario,55
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
filename = 'csv/classnames.csv'
with open(filename) as roster:
    for line in roster:
        print(line.strip())
```

Acosta, RJ
Adelman, Jackson C.
Agha, Harris
Alcock, Nick R.
Aragon, Valeria
Arian, M Aditta
Atli, Emir C.
Berrutti Bartesaghi, Martina
Bhatia, Anjali K.
Bossman, Tryphena
Brant, Nora E.
Cass, Ryan T.
Chang, Daniel Y.
Chang, Kayla
Chen, Will J.
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 ‘“\n”
  • `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
# 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())

wordList
['0,',
 'proudly',
 'rise',
 'the',
 'monarchs',
 'of',
 'our',
 'mountain',

len(wordList)
133

# number of times a word ('mountains!') is in the song?
wordList.count('mountains!')
4
```
Data Analysis w/ CSVs

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

```python
students = [] # initialize empty list
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)
```

Final result: a list of strings

```
['RJ Acosta',
 'Jackson C. Adelman',
 'Harris Agha',
 'Nick R. Alcock',
 'Valeria Aragon',
 'M Aditta Arian',
 'Emir C. Atli',
 'Martina Berrutti Bartesaghi',
 'Anjali K. Bhatia',
]```

lastname, firstname

string parsing to find first and last names; then append string to list
List Comprehensions
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 original 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 elements of the original 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)

```python
newList = [expression for item in sequence]
```

Filtering List Comprehension (only keep some elements)

```python
newList = [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()` inside of list comprehensions
  - We can combine mapping and filtering into a single list comprehension:

```python
newList = [expression for item in sequence if conditional]
```
Dissecting List Comprehensions

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

Using a list comprehension:

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

```
students
['RJ Acosta',
 'Jackson C. Adelman',
 'Harris Agha',
 'Nick R. Alcock',
```

- Example: How can we find the list of student names that begin with a vowel? (Hint: we'll use our `isVowel()` function again)
  - 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
students
['RJ Acosta',
 'Jackson C. Adelman',
 'Harris Agha',
 'Nick R. Alcock',
]

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

```python
# Which student names start with a vowel?
vowelNames = [name for name in students if isVowel(name[0])]
vowelNames
['Emir C. Atli',
 'Anjali K. Bhatia',
 'Alex W. Choi',
 'Ethan Cooper',
 'Edith N. Edwards-Mizel',
 'Amir H. Estejab',
]```
The end!

CS 134:
Files & List Comprehensions