CSCI 134 Fall 2021:
Files & List Comprehensions

September 29, 2021

Shikha Singh, 9AM
Jeannie Albrecht, 10AM
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?
- **Lab 2** graded feedback will be released today: instructions on viewing feedback is the same for all labs (on [http://cs.williams.edu/~cs134/labs-and-hws.html](http://cs.williams.edu/~cs134/labs-and-hws.html))
- **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()`
Today’s Plan

• Review useful string methods:
  • `split()`, `strip()`, `join()`, and more
• Discuss **file reading** using lists and strings
• Learn about **list comprehensions** as a way to simplify list accumulations
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.
### Summary: Sequence Operations (Strings, Lists, Ranges)

<table>
<thead>
<tr>
<th>Operation</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>x in seq</code></td>
<td>True if an item of seq is equal to x</td>
</tr>
<tr>
<td><code>x not in seq</code></td>
<td>False if an item of seq is equal to x</td>
</tr>
<tr>
<td><code>seq1 + seq2</code></td>
<td>The concatenation of seq1 and seq2*</td>
</tr>
<tr>
<td><code>seq*n, n*seq</code></td>
<td>n copies of seq concatenated</td>
</tr>
<tr>
<td><code>seq[i]</code></td>
<td>i’th item of seq, where origin is 0</td>
</tr>
<tr>
<td><code>seq[i:j]</code></td>
<td>slice of seq from i to j</td>
</tr>
<tr>
<td><code>seq[i:j:k]</code></td>
<td>slice of seq from i to j with step k</td>
</tr>
<tr>
<td><code>len(seq)</code></td>
<td>length of seq</td>
</tr>
<tr>
<td><code>min(seq)</code></td>
<td>smallest item of seq</td>
</tr>
<tr>
<td><code>max(seq)</code></td>
<td>largest item of seq</td>
</tr>
</tbody>
</table>

* Concatenation is not supported on range objects
String Methods
Useful in File Reading
Strings to Lists w/ `split()`

- `split()` is used to convert strings to lists

- The `split()` string method splits strings at “spaces” (the default separator) and returns a list of (sub)strings

- Can optionally specify other delimiters as well

```python
In [5]: phrase = "What a lovely day"

In [6]: phrase.split()
Out[6]: ['What', 'a', 'lovely', 'day']

In [7]: newPhrase = "What a *lovely* day!"  # multiple spaces or punctuations dont matter

In [8]: newPhrase.split()
Out[8]: ['What', 'a', '*lovely*', 'day!']

In [9]: commaSepSpells = "Impervius, Portus, Lumos, Reducio, Protego"  # comma separated strings

In [10]: commaSepSpells.split(',')
Out[10]: ['Impervius', ' Portus', ' Lumos', ' Reducio', ' Protego']
```
Strings to Lists w/ `list()` Function

- `list()` function, when given another sequence (range or string), returns a list of elements in the sequence
- Let's review how it works with **strings** and **ranges**

In [1]: `spell = "Avada Kedavra!"`

In [2]: `list(spell) # can turn a string into a list of its characters`

Out[2]: `['A', 'v', 'a', 'd', 'a', ' ', 'K', 'e', 'd', 'a', 'v', 'r', 'a', '!' ]`

In [16]: `list(range(-10, 10))`

Out[16]: `[-10, -9, -8, -7, -6, -5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9]`

In [21]: `list(range(3))`

Out[21]: `[0, 1, 2]`
List to Strings w/ \texttt{join()}

- \texttt{join()} is a string method that converts lists to strings
- Given a list of strings, the \texttt{join()} string method, when applied to a string \texttt{char}, concatenates the strings together with the string \texttt{char} between them

In [11]: wordList = ['Everybody', 'is', 'looking', 'forward', 'to', 'the', 'weekend']

In [12]: '*'\texttt{.join(wordList)}
Out[12]: 'Everybody*is*looking*forward*to*the*weekend'

In [13]: '_'\texttt{.join(wordList)}
Out[13]: 'Everybody_is_looking_forward_to_the_weekend'

In [14]: ' '\texttt{.join(wordList)}
Out[14]: 'Everybody is looking forward to the weekend'
Remove whitespace w/ `strip()`

- `strip()` string method strips away whitespace & new line (\n) characters from the beginning/end of strings and **returns a new string**

```
In [1]: word = " ** Snowy Winters ** "
```

```
In [2]: word.strip()
```

```
Out[2]: ' ** Snowy Winters ** '
```

```
In [8]: "\nHello World\n".strip()
```

```
Out[8]: 'Hello World'
```

newline/return character is vertical whitespace
String Methods in Action

```python
word = 'Williams College'
word.split()  # ['Williams', 'College']
word.upper()  # 'WILLIAMS COLLEGE'
word.lower()  # 'williams college'
word.replace('iams', 'eslley')  # 'Willeslley College'
word.replace('tent', 'eselley')  # 'Williams College'

newWord = '   Spacey College   '
newWord.strip()  # 'Spacey College'

myList = ['Williams', 'College']
' '.join(myList)  # 'Williams College'
```

**Remember:** None of these operations change/affect the original string. They all return a new string!
Even More String Functions!

- **word.find(s)**
  - Return the first (or last) position (index) of string s in word. Returns -1 if not found.

- **char.isspace()**
  - Returns `True` if char is not empty and char is composed of white space (or lowercase, uppercase, alphabetic letters, digits, or either letters or digits).
  - Can also do: `islower()`, `isupper()`, `isalpha()`, `isdigit()`, `isalnum()`.

- **word.count(s)**
  - Returns the number of (non-overlapping) occurrences of s in word

- Many more: see `pydoc3 str`
Reading Data from Files
Reading Files: Open

- `open(filename, mode)` returns a file object
  - `filename` is a path to a file
  - `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 writing code to explicitly open and close, we will use the `with open` ... `as` block which keeps the file open within it
- Today’s focus: file objects are `iterable`
  - We will see how to iterate over the lines of a file
Reading Files: \texttt{with ... as}

\begin{verbatim}
with open\texttt{(filename)} as \texttt{inputFile}:
    # do something with file
\end{verbatim}

\textbf{Note. (syntax)} Indentation defines the body of the with block where the file is open

\texttt{f = open(filename, 'r')}  
... file operations involving \texttt{f} ...  
\texttt{f.close()}

\begin{verbatim}
with open\texttt{(filename, 'r')} as \texttt{f}:
    ... file operations involving \texttt{f} ...
    # \texttt{f} implicitly closed  
    # when with is done.
\end{verbatim}

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:

```python
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?
  - `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?
  - `' '.join(line.split(',,'))`
Useful List Methods: `extend()`

- `myList.extend([itemList])`: appends all the items in `itemList` to the end of `myList`
  - Method modifies the list it is called on

**Example.**

```python
>>> 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 discuss mutability in more detail
Useful List Methods: \texttt{count()}

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

\textbf{Example.}

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

- Number of words in *Pride and Prejudice*?
- Count of certain words?

```python
In [1]: wordList = []
with open('textfiles/prideandprejudice.txt') as book:
    for line in book:
        wordList.extend(line.strip().split())
len(wordList)
Out[1]: 122089

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
```

What does this do? What if we used `append`?
More Data Analysis

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

```python
# let's 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 come up

• **Mapping.** Iterate over a list and return a new list which 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 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)

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

List Comprehension for Filtering (only keep some elements)

```python
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:

```python
newSequence = [expression for item in sequence if conditional]
```
newSequence = [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 = [n**2 for n in range(10) if n%2 == 0]
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.

<table>
<thead>
<tr>
<th>Name</th>
<th>Age</th>
</tr>
</thead>
<tbody>
<tr>
<td>Harry</td>
<td>14</td>
</tr>
<tr>
<td>Hermoine</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

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

```python
filename = 'csv/roster02.csv'  # 9 am section
with open(filename) as roster:
    for student in roster:
        print(student.strip())
```

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
Coady,Tendai T.,25AAA
Cueto,Trishia,24AAA
Dabinett,Olivia A.,25AAA
Dohr,Peter R.,25AAA
Garrity-Rokous,Eamon,25AAA
George,Tyler J.,25AAA
Working with CSVs

• Using a list comprehension instead:

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

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

```python
[['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'],
 ['Coady', 'Tendai T.', '25AAA'],
 ['Cueto', 'Trishia', '24AAA'],
 ['Dabinett', 'Olivia A.', '25AAA'],
 ['Dohr', 'Peter R.', '25AAA'],
 ['Garrity-Rokous', 'Eamon', '25AAA'],
 ['George', 'Tyler J.', '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
• \texttt{word = list[a][b]}
  • \texttt{a} is index into “outer” list (identifies \texttt{which list} we want)
  • \texttt{b} in index into “inner” list (identifies \texttt{the element within the list} we want)
• Let’s see an example!