On your way in...(on the side table)

Hand-in:
1. Homework 2
   • Two Piles: <50 and >= 50

Pick-Up:
1. POGIL 18: More Lists and Strings
Welcome to CS 134!

Introduction to Computer Science
Iris Howley

-Strings-

Spring 2019
Student Tech Panel - Landing an Internship!

• Tuesday (tomorrow) at 7pm
• Location: Weston Hall (Admissions Office)

• Have you ever been interested in a career in Software Engineering, Research, or just Computer Science at Williams in general?
• Come listen to a panel of upperclassmen talk about their experiences with: Technical interviews, Interning at Google, Williams Research, working at Tech Start-Ups, and more!

• Pizza and refreshments will be provided!
Python Documentation

• > pydoc3 list

• > pydoc3 string

• You need to be at the Terminal, not in interactive python
  • Interactive python starts with this: ‘>>>’

• Also, python.org documentation:
  • https://docs.python.org/3/index.html
Welcome! This is the documentation for Python 3.7.2.

Parts of the documentation:

- **What's new in Python 3.7?**
  - or all "What's new" documents since 2.0

- **Tutorial**
  - start here

- **Library Reference**
  - keep this under your pillow

- **Language Reference**
  - describes syntax and language elements

- **Python Setup and Usage**
  - how to use Python on different platforms

- **Python HOWTOs**
  - in-depth documents on specific topics

- **Installing Python Modules**
  - installing from the Python Package Index & other sources

- **Distributing Python Modules**
  - publishing modules for installation by others

- **Extending and Embedding**
  - tutorial for C/C++ programmers

- **Python/C API**
  - reference for C/C++ programmers

- **FAQs**
  - frequently asked questions (with answers!)
Python Documentation

- 4.8. Intermezzo: Coding Style
- 5. Data Structures
  - 5.1. More on Lists
    - 5.1.1. Using Lists as Stacks
    - 5.1.2. Using Lists as Queues
    - 5.1.3. List Comprehensions
    - 5.1.4. Nested List Comprehensions
  - 5.2. The def statement
  - 5.3. Tuples and Sequences
  - 5.4. Sets
  - 5.5. Dictionaries
  - 5.6. Looping Techniques
  - 5.7. More on Conditions
  - 5.8. Comparing Sequences and Other Types
5.1. More on Lists

The list data type has some more methods. Here are all of the methods of list objects:

- `list.append(x)`
  - Add an item to the end of the list. Equivalent to `a[len(a):] = [x]`.

- `list.extend(iterable)`
  - Extend the list by appending all the items from the iterable. Equivalent to `a[len(a):] = iterable`.

- `list.insert(i, x)`
  - Insert an item at a given position. The first argument is the index of the element before which to insert, so `a.insert(0, x)` inserts at the front of the list, and `a.insert(len(a), x)` is equivalent to `a.append(x)`.

- `list.remove(x)`
  - Remove the first item from the list whose value is equal to `x`. It raises a `ValueError` if there is no such item.

- `list.pop([i])`
  - Remove the item at the given position in the list, and return it. If no index is specified, `a.pop()` removes and returns the last item in the list. (The square brackets around the `i` in the method signature denote that the parameter is optional, not that you should type square brackets at that position. You will see this notation frequently in the Python Library Reference.)

- `list.clear()`
  - Remove all items from the list. Equivalent to `del a[:]`.

- `list.index(a[, start[, end]])`
Starter Lab Feedback Now Available

• cd ~/cs134/starter

• git pull

• more GradeSheet.txt
  • (q to quit)
In plain English, what is an algorithm for cutting the names off the end of all strings that take the above form?

(Think the sandwich instructions from first day of class)

Take a couple minutes to discuss with a partner.
TODAY’S LESSON

Strings

(A sequence of characters.)
A Tale of Two Sequences

Accessing elements by index

- $l = [18, 20, 5, 16]$
- $l[1] = 20$
- $l[-1] = 16$

- $s = ‘hello’$
- $s[1] = ‘e’$
- $s[-1] = ‘o’$
A Tale of Two Sequences

Lengths of sequences

**l = [18, 20, 5, 16]**

- `len(l)`
  - 4

**s = ‘hello’**

- `len(s)`
  - 5
A Tale of Two Sequences
Slice notation for lists and strings

<table>
<thead>
<tr>
<th>l</th>
<th>h</th>
</tr>
</thead>
<tbody>
<tr>
<td>l = [18, 20, 5, 16, 'd']</td>
<td>h = &quot;Hello&quot;</td>
</tr>
<tr>
<td>l[2:] = [5, 16, 'd']</td>
<td>h[2:] = 'llo'</td>
</tr>
<tr>
<td>l[:2] = [18, 20]</td>
<td>h[::2] = 'He'</td>
</tr>
<tr>
<td>l[:2] = [18, 20, 5]</td>
<td>h[:2] = 'Hel'</td>
</tr>
<tr>
<td>l[2:] = [16, 'd']</td>
<td>h[2:] = 'lo'</td>
</tr>
</tbody>
</table>

We call this ‘slice notation’ or ‘string slicing’
A Tale of Two Sequences

Slice notation - step

- \( l = [18, 20, 5, 16] \)
- \( l[:::-1] \)
  - \([16, 5, 20, 18]\)

- \( s = \text{‘hello’} \)
- \( s[:::-1] \)
  - \( \text{‘olleh’} \)
- \( s[:::2] \)
  - \( \text{‘hlo’} \)
A Tale of Two Sequences
Slice notation – copying a sequence

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>m = l</td>
<td>n = l</td>
<td>o = l[:]</td>
</tr>
<tr>
<td>l == m</td>
<td>l == n</td>
<td>l == o</td>
</tr>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>l is m</td>
<td>l is n</td>
<td>l is o</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>False</td>
</tr>
</tbody>
</table>

l[:] returns a *copy* of l (similar to l and n)

l and m are tied to the same balloon!
l and n are tied to different balloons!
Slice Notation

- `s[start:end:step]`  # start through not past end, by step
- `s[start:end]`  # items start through end-1
- `s[start:]`  # items start through the rest of the list
- `s[:end]`  # items from the beginning through end-1
- `s[:]`  # a copy of the whole list
A Tale of Two Sequences

Sorting lists and strings

• \( l = [18,20,5,16] \)
• \( m = \text{sorted}(l) \)
• \( l \)
  • \([18,20,5,16]\)
• \( m \)
  • \([5,15,18,20]\)

• \( s = \text{‘hello’} \)
• \( t = \text{sorted}(s) \)
• \( s \)
  • \( \text{‘hello’} \)
• \( t \)
  • \([‘e’, ‘h’, ‘l’, ‘l’, ‘o’]\)

If you want to use these modified strings & lists, you need to attach them to a variable!
A Tale of Two Sequences

• What do we do with \( t=\['e', 'h', 'l', 'l', 'o'\] \)?
  - We can turn it back into a string!

• `''.join(t)
  - `ehllo`

• `';'.join(sorted('hello'))`
  - `e;h;l;l;o`

We now have the tools to alphabetize a string. How do we do that?
Canonical Form

• (alphabetizing a string)

>>> s = 'hello'
This character says what to join the list with
>>> t = ''.join(sorted(s))
>>> t
  'ehllo'
A Tale of Two Sequences

• \( l = [18, 20, 5, 16] \)
• \( l.append('dogge') \)

\[
\begin{align*}
l & \quad [18, 20, 5, 16, 'dogge'] \\
\end{align*}
\]

A String is not a List!

• \( s = 'hello' \)
• \( s.append('world') \)

• \( s += '!' \)
• \( 'hello!' \)
String Functions

\[ s = ' Csci 134 ' \]

- \( s\).lower()  
  \( ' csci 134 ' \)
- \( s\).upper()  
  \( ' CSCI 134 ' \)
- >>> s
  \( ' Csci 134 ' \)
- \( s\).strip()  
  \( 'Csci 134' \)
- \( s\).replace(' ', '')  
  \( 'Csci134' \)
String Functions

More built-in string functions described in python.org documentation:

https://docs.python.org/3/library/stdtypes.html#text-sequence-type-str
Canonical Form

Formatting a string in such a way that similar words can be compared
• (i.e. lowercase and sorting)
• What about h = ‘Hello’? What should its canonical form be?

>>> h = ‘Hello'
>>> i = ''.join(sorted(h))
>>> i
‘Hello’

The capital H will make it difficult to compare words!
More Strings & Lists

• Look at POGIL 18 – More Lists & Strings

• It goes through a few more String functions that might be useful

• Let’s look at Question 8
• What does [:] do?
QUESTIONS?
Leftover Slides
Reading in from a File

• [https://docs.python.org/3/tutorial/inputoutput.html?highlight=read%20from%20file#reading-and-writing-files](https://docs.python.org/3/tutorial/inputoutput.html?highlight=read%20from%20file#reading-and-writing-files)

• `with open('workfile.txt') as f:
  for line in f:
    print(line)  # do something with line here!
• # when done with file, unindent`
Reading in from a File
Multiple ways to do the same task in python

- `f = open("workfile.txt","r")`
- `for line in f:
  - print(line)  # do something with line here!`
- `# when done with file, close file`
- `f.close()`
__all__ special variable

• If the variable starts/ends with "__ _" it’s a special python variable
• We saw this with __name__

• __all__ is another special variable
• Whatever is stored in __all__ is imported when the user types:
  • from _____ import *
• Any function/variable/etc. that’s not included in __all__ can be imported explicitly
  • from <module name> import <not-included-in-star-variable/function>
A Tale of Two Sortings...

• \( l = [18, 20, 5, 16] \)
• \( l.sort() \)
• \( l \rightarrow [5, 16, 18, 20] \)

• \( m = [18, 20, 5, 16] \)
• \( \text{sorted}(m) \)
• \( m \rightarrow [18, 20, 5, 16] \)

.sort() sorts the list itself
.sorted() returns a copy of the sorted list
Mutability
A Balloon Metaphor...

>>> 3
A Balloon Metaphor...

>>> x = 3
A Balloon Metaphor...

>>> x = x + 1

NUMBERS ARE IMMUTABLE
A Balloon Metaphor...

>>> [5,16,18]
A Balloon Metaphor...

```python
>>> mylist = [5,16,18]
```
A Balloon Metaphor...

```python
>>> mylist.append('dogge')
```

LISTS ARE MUTABLE

mylist
A Tale of Two Mutabilities...

- $x = 3$
- $y = 3$
- $x == y$
  - True
- $x$ is $y$
  - True

- $l = [1,2,3]$
- $m = [1,2,3]$
- $l == m$
  - True
- $l$ is $m$
  - False
A Balloon Metaphor...

```python
>>> l = [1,2,3]
>>> m = [1,2,3]
```
A Tale of Two Mutabilities...

- $x = 3$
- $y = x$
- $x == y$
  - True
- $x$ is $y$
  - True

- $l = [1,2,3]$
- $m = l$
- $l == m$
  - True
- $l$ is $m$
  - True
A Balloon Metaphor...

```python
>>> l = [1, 2, 3]
>>> m = l
```