lst = ['ant', 'bat', 'cat', 'dog']
for i in range(len(lst)):
    print(i, lst[i])
I is for ...

- **Identity**
  - Who are you? Computer Science Student
- **Invariant**
  - Reasoning formally and informally about loops
- **Internet**
  - Network of networks
  - Far more than that!
Lynn Conway

See Wikipedia and lynnconway.com

• Helped invent dynamic scheduling early '60s IBM
• Transgender, fired in ‘68
• IBM apologized in 2020 (52 years later)

• Joined Xerox Parc in 1979
• Revolutionized VLSI design with Carver Mead

• Joined U. Michigan 1985
• NAE '89, IEEE Pioneer ‘09
• Professor and Dean, retired '98

“If you want to change the future, start living as if you are already there.”
Announcements

• APT-2 due Thursday!
  • Remember you get 24-hour grace period, can’t turn in after that!

• Assignment 2 Turtles out – due Tues Feb 15

• Exam 1 handed back today via Gradescope
  • Regrade request in Gradescope by 2/15

• Lab 5 Friday – Prelab out now
Plan for the Day

- Accumulator Pattern
- Range
- Loop Index
- Loop Tracing
- Files
The Accumulator Pattern

• Pattern you will see with a lot of loops
• Here is the pattern:

  • Initialize a variable
  • loop over a sequence (list or string)
    • Accumulate (add a little more to variable)
  • Do something with variable (result)
Example of Accumulator Pattern

```python
def sumlist(lst):
    total = 0
    for num in lst:
        total += num
    return total
```
Example of Accumulator Pattern

```python
def sumlist(lst):
    total = 0
    for num in lst:
        total += num
    return total
```
Example of Accumulator Pattern

```python
def sumlist(lst):
    total = 0
    for num in lst:
        total += num
    return total
```

`lsta = [3, 7, 8, 2, 6]`

`print(sumlist(lsta))`

Output:
Example of Accumulator Pattern

def sumlist(lst):
    total = 0
    for num in lst:
        total += num
    return total

lsta = [3, 7, 8, 2, 6]
print(sumlist(lsta))

Output: 26
Example 2: Accumulator Pattern

```python
def numLetters(word):
    total = 0
    for letter in word:
        total += 1
    return total
```
Example 2: Accumulator Pattern

```python
def numLetters(word):
    total = 0
    for letter in word:
        total += 1
    return total
```

- Initialize variable
- Loop over characters in word
- Update total, Add 1 to it
- Return variable
Example 2: Accumulator Pattern

```python
def numLetters(word):
    total = 0
    for letter in word:
        total += 1
    return total
```

```python
word = "giraffe"
print(numLetters(word))
```

Output:
Example 2: Accumulator Pattern

```python
def numLetters(word):
    total = 0
    for letter in word:
        total += 1
    return total
```

```python```
word = "giraffe"
print(numLetters(word))
```

Output:
```
7
```
REVIEW: Looping over Sequences

• Let’s explore this:
  • Given a sentence:
    • “Duke Computer Science is so much fun!”
  • How do we create this sentence?
    • “Dk Cmptr Scnc s s mch fn!”
  • Input is sentence. Output has vowels removed
Accumulator Pattern: NoVowels

- “For each character, if it’s not a vowel add it to the output string”
- Accumulator pattern: change a variable in a loop
  - Accumulate a value while iterating through loop

```python
def noVowels(phrase):
    ret = ""
    for ch in phrase:
        if not isVowel1(ch):
            ret = ret + ch
    return ret
```
Accumulator Pattern: NoVowels

- “For each character, if it’s not a vowel add it to the output string”
- Accumulator pattern: change a variable in a loop
  - Accumulate a value while iterating through loop

```python
def noVowels(phrase):
    ret = ""
    for ch in phrase:
        if not isVowel(ch):
            ret = ret + ch
    return ret
```
range() Sequence

- `range(y)` – starts at 0 and goes up to but doesn't include `y`: 0 … (y-1)
- `range(x, y)`: x … (y-1)
- Sounds familiar? Slicing!
- Sequence that provides access to int values
Example

range(5)
list(range(5))
range(5)[0]
range(5)[4]
range(5)[5]
range(5,10)
list(range(5,10))
range(5,10)[3]
for x in range(3):
    print(x)
## Example

<table>
<thead>
<tr>
<th>Python Expression</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>range(5)</code></td>
<td><code>range(0, 5)</code></td>
</tr>
<tr>
<td><code>list(range(5))</code></td>
<td><code>[0, 1, 2, 3, 4]</code></td>
</tr>
<tr>
<td><code>range(5)[0]</code></td>
<td><code>0</code></td>
</tr>
<tr>
<td><code>range(5)[4]</code></td>
<td><code>4</code></td>
</tr>
<tr>
<td><code>range(5)[5]</code></td>
<td><code>ERROR!!!!!!</code></td>
</tr>
<tr>
<td><code>range(5,10)</code></td>
<td><code>range(5,10)</code></td>
</tr>
<tr>
<td><code>list(range(5,10))</code></td>
<td><code>[5, 6, 7, 8, 9]</code></td>
</tr>
<tr>
<td><code>range(5,10)[3]</code></td>
<td><code>8</code></td>
</tr>
<tr>
<td><code>for x in range(3): print(x)</code></td>
<td><code>0</code> <code>1</code> <code>2</code></td>
</tr>
</tbody>
</table>
Range Examples

• Access all the values in a list to print them
  • Use the “for each in sequence” pattern

```python
lst = ["ant", "bat", "cat", "dog"]
for s in lst:
    print(s)
```
Range Examples

• Access all the values in a list to print them
  • Use the “for each in sequence” pattern

```python
lst = ["ant", "bat", "cat", "dog"]
for s in lst:
    print(s)
```

Output: “ant”
“bat”
“cat”
“dog”
Range Examples

• Access all the values in a list to print them
  • Use an index to access $i^{\text{th}}$ element

```python
lst = ['ant', 'bat', 'cat', 'dog']
for i in range(len(lst)):
    print(i, lst[i])
```
Range Examples

• Access all the values in a list to print them
  • Use an index to access $i^{th}$ element

```python
lst = ["ant", "bat", "cat", "dog"]
for i in range(len(lst)):
    print(i, lst[i])
```

Output: 0 ant
        1 bat
        2 cat
        3 dog
Repetition with Range

- Sometimes rather than looping over a sequence of values you want to repeat # times
  - Do this 4 times
  - Do that 250 times

- Can do this with the Python range function!
  - If don’t care about the value in the range (e.g. “Do this four times”), can do:

```python
for _ in range(4):
    CODE
```
Code-Tracing a Loop

1. Find the changing variables/expressions
2. Create table, columns are variables/expressions
   1. First column is loop variable
   2. Add columns to help track everything else
3. Each row is an iteration of the loop
   1. Before execute code block, copy down each variable’s value
   2. Execute code block, update a value in the row as it changes
Code-Tracing a Loop

1. Find the changing variables/expressions
2. Create table, columns are variables/expressions
   1. First column is loop variable
   2. Add columns to help track everything else

def mystery(lst):
    idxMax = 0
    for i in range(len(lst)):
        if lst[idxMax] < lst[i]:
            idxMax = i

    return idxMax

What should be the table's columns?
Code-Tracing a Loop

1. Find the changing variables
2. Create table, columns are the variables
   1. First column is loop variable
   2. Add columns to help track everything else

```python
def mystery(lst):
    idxMax = 0
    for i in range(len(lst)):
        if lst[idxMax] < lst[i]:
            idxMax = i
    return idxMax
```
Fill in table

1. Before execute code block, copy down each variable’s value
2. Execute code block, update a value in the row as it changes

```python
def mystery(lst):
    idxMax = 0
    for i in range(len(lst)):
        if lst[idxMax] < lst[i]:
            idxMax = i
    return idxMax

mystery([2, 12, 4, 15, 15])
```

<table>
<thead>
<tr>
<th>i</th>
<th>idxMax</th>
<th>lst[idxMax]</th>
<th>lst[i]</th>
<th>lst[idxMax] &lt; lst[i]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
1. Before execute code block, copy down each variable’s value

2. Execute code block, update a value in the row as it changes

---

```python
def mystery(lst):
    idxMax = 0
    for i in range(len(lst)):
        if lst[idxMax] < lst[i]:
            idxMax = i
    return idxMax

mystery([2, 12, 4, 15, 15])
```

---

<table>
<thead>
<tr>
<th>i</th>
<th>idxMax</th>
<th>lst[idxMax]</th>
<th>lst[i]</th>
<th>lst[idxMax] &lt; lst[i]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Fill in table

1. Before execute code block, copy down each variable’s value
2. Execute code block, update a value in the row as it changes

```python
def mystery(lst):
    idxMax = 0
    for i in range(len(lst)):
        if lst[idxMax] < lst[i]:
            idxMax = i
    return idxMax

mystery([2, 12, 4, 15, 15])
```

<table>
<thead>
<tr>
<th>i</th>
<th>idxMax</th>
<th>lst[idxMax]</th>
<th>lst[i]</th>
<th>lst[idxMax] &lt; lst[i]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>False</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Fill in table

1. Before execute code block, copy down each variable's value

2. Execute code block, update a value in the row as it changes

```python
def mystery(lst):
    idxMax = 0
    for i in range(len(lst)):
        if lst[idxMax] < lst[i]:
            idxMax = i
    return idxMax
```

mystery([2, 12, 4, 15, 15])

<table>
<thead>
<tr>
<th>i</th>
<th>idxMax</th>
<th>lst[idxMax]</th>
<th>lst[i]</th>
<th>idxMax &lt; lst[i]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>False</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Fill in table

1. Before execute code block, copy down each variable’s value
2. Execute code block, update a value in the row as it changes

```python
def mystery(lst):
    idxMax = 0
    for i in range(len(lst)):
        if lst[idxMax] < lst[i]:
            idxMax = i
    return idxMax
mystery([2, 12, 4, 15, 15])
```

<table>
<thead>
<tr>
<th>i</th>
<th>idxMax</th>
<th>lst[idxMax]</th>
<th>lst[i]</th>
<th>lst[idxMax] &lt; lst[i]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>False</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>2</td>
<td>12</td>
<td>True</td>
</tr>
</tbody>
</table>
Fill in table

1. Before execute code block, copy down each variable’s value

2. Execute code block, update a value in the row as it changes

```python
def mystery(lst):
    idxMax = 0
    for i in range(len(lst)):
        if lst[idxMax] < lst[i]:
            idxMax = i
    return idxMax
```

```
mystery([2, 12, 4, 15, 15])
```

<table>
<thead>
<tr>
<th>i</th>
<th>idxMax</th>
<th>lst[idxMax]</th>
<th>lst[i]</th>
<th>lst[idxMax] &lt; lst[i]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>False</td>
</tr>
<tr>
<td>1</td>
<td>0 1</td>
<td>2</td>
<td>12</td>
<td>True</td>
</tr>
</tbody>
</table>
1. Before execute code block, copy down each variable’s value

2. Execute code block, update a value in the row as it changes

```python
def mystery(lst):
    idxMax = 0
    for i in range(len(lst)):
        if lst[idxMax] < lst[i]:
            idxMax = i
    return idxMax

mystery([2, 12, 4, 15, 15])
```

<table>
<thead>
<tr>
<th>i</th>
<th>idxMax</th>
<th>lst[idxMax]</th>
<th>lst[i]</th>
<th>lst[idxMax] &lt; lst[i]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>False</td>
</tr>
<tr>
<td>1</td>
<td>0 1</td>
<td>2</td>
<td>12</td>
<td>True</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
1. Before execute code block, copy down each variable’s value
2. Execute code block, update a value in the row as it changes

```python
def mystery(lst):
    idxMax = 0
    for i in range(len(lst)):
        if lst[idxMax] < lst[i]:
            idxMax = i
    return idxMax

mystery([2, 12, 4, 15, 15])
```

<table>
<thead>
<tr>
<th>i</th>
<th>idxMax</th>
<th>lst[idxMax]</th>
<th>lst[i]</th>
<th>lst[idxMax] &lt; lst[i]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>False</td>
</tr>
<tr>
<td>1</td>
<td>0 1</td>
<td>2</td>
<td>12</td>
<td>True</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>12</td>
<td>4</td>
<td>False</td>
</tr>
</tbody>
</table>
1. Before execute code block, copy down each variable's value

2. Execute code block, update a value in the row as it changes

```python
def mystery(lst):
    idxMax = 0
    for i in range(len(lst)):
        if lst[idxMax] < lst[i]:
            idxMax = i
    return idxMax

mystery([2, 12, 4, 15, 15])
```

<table>
<thead>
<tr>
<th>i</th>
<th>idxMax</th>
<th>lst[idxMax]</th>
<th>lst[i]</th>
<th>lst[idxMax] &lt; lst[i]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>False</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>2</td>
<td>12</td>
<td>True</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>12</td>
<td>4</td>
<td>False</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
1. Before execute code block, copy down each variable’s value

2. Execute code block, update a value in the row as it changes

```python
def mystery(lst):
    idxMax = 0
    for i in range(len(lst)):
        if lst[idxMax] < lst[i]:
            idxMax = i
    return idxMax
```

mystery([2, 12, 4, 15, 15])

<table>
<thead>
<tr>
<th>i</th>
<th>idxMax</th>
<th>lst[idxMax]</th>
<th>lst[i]</th>
<th>lst[idxMax] &lt; lst[i]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>False</td>
</tr>
<tr>
<td>1</td>
<td>0 1</td>
<td>2</td>
<td>12</td>
<td>True</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>12</td>
<td>4</td>
<td>False</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>12</td>
<td>15</td>
<td>True</td>
</tr>
</tbody>
</table>
1. Before execute code block, copy down each variable’s value

2. Execute code block, update a value in the row as it changes

```python
def mystery(lst):
    idxMax = 0
    for i in range(len(lst)):
        if lst[idxMax] < lst[i]:
            idxMax = i
    return idxMax
```

```
mystery([2, 12, 4, 15, 15])
```

<table>
<thead>
<tr>
<th>i</th>
<th>idxMax</th>
<th>lst[idxMax]</th>
<th>lst[i]</th>
<th>lst[idxMax] &lt; lst[i]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>False</td>
</tr>
<tr>
<td>1</td>
<td>0 1</td>
<td>2</td>
<td>12</td>
<td>True</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>12</td>
<td>4</td>
<td>False</td>
</tr>
<tr>
<td>3</td>
<td>1 3</td>
<td>12</td>
<td>15</td>
<td>True</td>
</tr>
</tbody>
</table>
1. Before execute code block, copy down each variable’s value

2. Execute code block, update a value in the row as it changes

```python
def mystery(lst):
    idxMax = 0
    for i in range(len(lst)):
        if lst[idxMax] < lst[i]:
            idxMax = i
    return idxMax

mystery([2, 12, 4, 15, 15])
```

<table>
<thead>
<tr>
<th>i</th>
<th>idxMax</th>
<th>lst[idxMax]</th>
<th>lst[i]</th>
<th>lst[idxMax] &lt; lst[i]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>False</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>2</td>
<td>12</td>
<td>True</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>12</td>
<td>4</td>
<td>False</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>12</td>
<td>15</td>
<td>True</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
1. Before execute code block, copy down each variable’s value

2. Execute code block, update a value in the row as it changes

```python
def mystery(lst):
    idxMax = 0
    for i in range(len(lst)):
        if lst[idxMax] < lst[i]:
            idxMax = i
    return idxMax

mystery([2, 12, 4, 15, 15])
```

<table>
<thead>
<tr>
<th>i</th>
<th>idxMax</th>
<th>lst[idxMax]</th>
<th>lst[i]</th>
<th>lst[idxMax] &lt; lst[i]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>False</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>2</td>
<td>12</td>
<td>True</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>12</td>
<td>4</td>
<td>False</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>12</td>
<td>15</td>
<td>True</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>15</td>
<td>15</td>
<td>False</td>
</tr>
</tbody>
</table>
What is always true about the loop?

```python
def mystery(lst):
    idxMax = 0
    for i in range(len(lst)):
        if lst[idxMax] < lst[i]:
            idxMax = i
    return idxMax
```

mystery([2, 12, 4, 15, 15])

<table>
<thead>
<tr>
<th>i</th>
<th>idxMax</th>
<th>lst[idxMax]</th>
<th>lst[i]</th>
<th>lst[idxMax] &lt; lst[i]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>False</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>2</td>
<td>12</td>
<td>True</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>12</td>
<td>4</td>
<td>False</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>12</td>
<td>15</td>
<td>True</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>15</td>
<td>15</td>
<td>False</td>
</tr>
</tbody>
</table>
What is always true about the loop?

1. lst[idxMax] is always the largest value seen so far, up through value of i

```python
def mystery(lst):
    idxMax = 0
    for i in range(len(lst)):
        if lst[idxMax] < lst[i]:
            idxMax = i
    return idxMax

mystery([2, 12, 4, 15, 15])
```

<table>
<thead>
<tr>
<th>i</th>
<th>idxMax</th>
<th>lst[idxMax]</th>
<th>lst[i]</th>
<th>lst[idxMax] &lt; lst[i]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>False</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>2</td>
<td>12</td>
<td>True</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>12</td>
<td>4</td>
<td>False</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>12</td>
<td>15</td>
<td>True</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>15</td>
<td>15</td>
<td>False</td>
</tr>
</tbody>
</table>
def mystery(lst):
    idxMax = 0
    for i in range(len(lst)):
        if lst[idxMax] < lst[i]:
            idxMax = i
    return idxMax

mystery([2, 12, 4, 15, 15])

<table>
<thead>
<tr>
<th>i</th>
<th>idxMax</th>
<th>lst[idxMax]</th>
<th>lst[i]</th>
<th>lst[idxMax] &lt; lst[i]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>False</td>
</tr>
<tr>
<td>1</td>
<td>0 1</td>
<td>2</td>
<td>12</td>
<td>True</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>12</td>
<td>4</td>
<td>False</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>12</td>
<td>15</td>
<td>True</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>15</td>
<td>15</td>
<td>False</td>
</tr>
</tbody>
</table>
• Remember the steps
• (1) Find the changing variable/expressions,
• (2) Create the table with these as the column
• (3) Each row is an iteration of the loop
Examples of Processing Data

• Lecture 1: count letters in Bible

• Another example: Google Ngram viewer
  • https://books.google.com/ngrams
Studying Language Evolution

• Ngram informs how words evolve
• From friend vs enemy
• https://www.youtube.com/watch?v=tFW7orQsBuo
Sequences, Repetition

- Parameters? What are they to this query?
  - https://books.google.com/ngrams/graph?content=friend%2Cenemy&year_start=1900&year_end=2019&corpus=26&smoothing=3&direct_url=t1%3B%2Cfriend%3B%3B.t1%3B%2Cenemy%3B%3B
Sequences, Repetition

- Parameters? What are they to this query?
- https://books.google.com/ngrams/graph?content=friend%2Cenemy&year_start=1900&year_end=2019&corpus=26&smoothing=3&direct_url=t1%3B%2Cfriend%3B%3B.t1%3B%2Cenemy%3B%3B

What can the URL tell you?
Sequences, Repetition

- Parameters? What are they to this query?
  https://books.google.com/ngrams/graph?content=friend%2Cenemy&year_start=1900&year_end=2019&corpus=26&smoothing=3&direct_url=t1%3B%2Cfriend%3B%3B.t1%3B%2Cenemy%3B%3B
Processing Data

- How do we find the longest word in any text?
- How do we find the word that occurs the most?
- How is this related to how Google Search works?

- Text files can be viewed as sequences
  - Sequences of lines
  - Each line is a string
  - Some clean-up because of ‘\n’
File Pattern: One line at a time

- Simplest and reasonably efficient Python pattern
  - Open, loop, close, return/process
  - LineCounter.py

- File as sequence
  - One line at-a-time

- Asymmetry in Open vs Close steps
def lineCount(fname):
    """
    # lines in file fname
    """
    f = open(fname)
    lc = 0
    for line in f:
        lc = lc + 1
    f.close()
    return lc
def altCount(fname):
    
    return # lines in file fname
    
    f = open(fname)
    lc = len(f.readlines())
    f.close()
    return lc
```python
if __name__ == "__main__":
    name = "data/poe.txt"
    pc = lineCount(name)
    print("# lines:",pc)
    pc2 = altCount(name)
    print("# lines:",pc2)
```
File Objects

• A file is an object, like a string
  • Functions applied to object: \texttt{len(“word”)}
  • To get file object use \texttt{open(“data.txt”)}
  • What is returned? Integer value, file object

• Often methods (aka function) applied to object
  • \texttt{f.readlines(), f.read(), f.close()}
  • Just like: \texttt{st.lower(), st.count(“e”)}
STOPPED HERE

- DID NOT DO SLIDES BELOW
Text File Processing Pattern

• See module `FileStuff.py`
  • If newline `\n` is read, call `.strip()`
  • If want to break line into “words”, call `.split()`

• Process the list returned by `.split()`
  • May need to convert strings to int or float or …

• The `for line in f:` pattern is efficient
  • Contrast list returned by `f.readlines()`
```python
def avgWord(fname):
    f = open(fname, encoding="utf-8")
    totalWords = 0
    totalLen = 0
    for line in f:
        line = line.strip()  # remove newline
        data = line.split()
        for word in data:
            totalWords = totalWords + 1
            totalLen = totalLen + len(word)
    f.close()
    return totalLen/totalWords
```