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 Tonight! Remember 24-hour grace period
• Assignment 2 Turtles out – due Thurs Oct 6
• DO NOT discuss Exam 1 until it is handed back
  • Earliest would be next week
  • Will be handed back on Gradescope
• Lab 4 Friday – Prelab out now
  • No lab Friday Oct 7
• Coming, APT-3 and Assignment 3 out next week
• Coming, APT-1 QUIZ (Oct 13-17)
  • Timed test, take when you want during these dates
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

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

Output:
Example of Accumulator Pattern

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

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

```python
print(sumlist(lsta))
```

Output:

```
26
```
Example of Accumulator Pattern

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

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

Output:

```
0
3
```
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:

```
3
3
```

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:

```
3
7
```

Output: 26
Example of Accumulator Pattern

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

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

Output:
Example of Accumulator Pattern

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

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

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:
Example of Accumulator Pattern

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

```python
lst = [3, 7, 8, 2, 6]
print(sumlist(lst))
```

Output:

18

2
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:
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:
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 of Accumulator Pattern

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

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

Output:

```
26
```

```python
lst = [3, 7, 8, 2, 6]
print(sumlist(lst))
```

```
Output:
26
```
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:
```
26
```

We implemented the sum function

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
```
Example 2: Accumulator Pattern

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

word = "card"
print(numLetters(word))

Output:
Example 2: Accumulator Pattern

```python
# Define the function to count the number of letters in a word
def numLetters(word):
    total = 0
    for letter in word:
        total += 1
    return total

# Assign a word and print the result
word = "card"
print(numLetters(word))
```

Output: 0
Example 2: Accumulator Pattern

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

word = "card"
print(numLetters(word))

Output:

<table>
<thead>
<tr>
<th>total</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>letter</td>
<td>'c'</td>
</tr>
</tbody>
</table>
Example 2: Accumulator Pattern

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

word = "card"
print(numLetters(word))
```

Output:

```
1 'c'
```
Example 2: Accumulator Pattern

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

word = "card"
print(numLetters(word))
```

Output:

```
total
letter
1
'a'
```

Output:
Example 2: Accumulator Pattern

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

word = "card"
print(numLetters(word))
```

Output:

```
total: 2
letter: 'a'
```

Output:
Example 2: Accumulator Pattern

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

word = "card"
print(numLetters(word))
```

Output:

```
2
'r'
```
Example 2: Accumulator Pattern

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

word = "card"
print(numLetters(word))
```

Output:

```
3
'r'
```
Example 2: Accumulator Pattern

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

word = "card"
print(numLetters(word))

Output:
Example 2: Accumulator Pattern

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

word = "card"
print(numLetters(word))
```

Output:

```
4
'd'
```

9/29/22 Compsci 101, Fall 2022 34
Example 2: Accumulator Pattern

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

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

Output: 4
Example 2: Accumulator Pattern

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

word = "card"
print(numLetters(word))
```

Output:

```
total  4
letter 'd'
```

We implemented the len function

Output: 4
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 Cmprtr 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
20 def noVowels(phrase):
21     ret = ""
22     for ch in phrase:
23         if not isVowel1(ch):
24             ret = ret + ch
25     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 isVowel1(ch):
            ret = ret + ch
    return ret
```
range() Sequence

• Range generates a sequence of values
• \texttt{range(y)} – starts at 0 and goes up to but doesn't include \(y\): \(0 \ldots (y-1)\)
  • \(y\) is an integer
• \texttt{range(x, y)}: \(x \ldots (y-1)\)
  • \(x\) and \(y\) are integers
• Sequence that provides access to int values
• "up to but not including" sounds familiar? Slicing!
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>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>0</td>
</tr>
<tr>
<td><code>range(5)[4]</code></td>
<td>4</td>
</tr>
<tr>
<td><code>range(5)[5]</code></td>
<td>ERROR!!!!!!</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>8</td>
</tr>
</tbody>
</table>

for x in range(3):
  print(x)

- Not a list, but generates a sequence of numbers!
- `range(x)` generates a sequence that can be used in a for loop
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 $i$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 \( i^{\text{th}} \) 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
```

range(0,3)  
Think of as [0, 1, 2, 3]  
But it is NOT a list
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
```
WOTO-1 – Accumulator, Range
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

```python
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>
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></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></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

```
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>lst[0] is 2</td>
<td>lst[0] is 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

```
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></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

```
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>lst[0] is 2</td>
<td>lst[1] is 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

<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>1</td>
<td>12</td>
<td>True</td>
</tr>
</tbody>
</table>

```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])
```
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>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>lst[1] is 12</td>
<td>lst[2] is 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 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></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

```
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>1</td>
<td>2</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>lst[1] is 12</td>
<td>lst[3] is 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>
</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
```

```python
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>4</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

```
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>
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>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>

return 3
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>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>
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 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>
What is always true about the loop?

1. \( \text{lst}[\text{idxMax}] \geq \text{lst}[k] \) for all \( k \leq i \)
2. \( i < \text{len}(\text{lst}) \)
3. \( \text{idxMax} < \text{len}(\text{lst}) \)

```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>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>
WOTO-2 Loop Tracing

- 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
  • Ngram informs how words evolve
  • Shows number of times phrases occur in books over the years
  • https://books.google.com/ngrams

• Funny video on irregular words
  • https://www.youtube.com/watch?v=tFW7orQsBuo
Studying Language Evolution

- friend vs enemy
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):
    """
    return # 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: `len("word")`
  • To get file object use `open("data.txt")`
  • What is returned? Integer value, file object

• Often methods (aka function) applied to object
  • `f.readlines()`, `f.read()`, `f.close()`
  • Just like: `st.lower()`, `st.count("e")`