

Compsci 101

Accumulator Pattern, Loop Tracing, Files

Susan Rodger

February 14, 2023

```
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

- **Assignment 2 Turtles due Thurs!**
- **Lab 5 Friday – Prelab coming out Wed or Thur**
- **Coming, APT-3 out Thursday**
- **Coming, APT-1 QUIZ (Feb 23-27)**
 - Timed APTs, take when you want during these dates
 - Your own work!
- **DO NOT discuss Exam 1 until it is handed back**
 - Will be handed back on Gradescope

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)

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)



Build a new
list or string
or number

Example of Accumulator Pattern

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


Example of Accumulator Pattern

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

Initialize variable

Loop over
numbers in lst

update total,
Add num to it

return variable

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
```

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
```

total

0

num

3

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

Output:

Example of Accumulator Pattern

```
def sumlist(lst):
```

```
    total = 0
```

total

3

```
    for num in lst:
```



```
        total += num
```

num

3

```
    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
```

total

3

num

7

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

Output:

Example of Accumulator Pattern

```
def sumlist(lst):
```

```
    total = 0
```

total

10

```
    for num in lst:
```



```
        total += num
```

num

7

```
    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
```

total

10

num

8

```
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
```

total

18

num

8



```
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
```

total

18

num

2

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

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

Output:

Example of Accumulator Pattern

```
def sumlist(lst):
```

```
    total = 0
```

total

20

```
    for num in lst:
```



```
        total += num
```

num

2

```
    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
```

total

20

num

6

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

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

Output:

Example of Accumulator Pattern

```
def sumlist(lst):
```

```
    total = 0
```

total

26

```
    for num in lst:
```



```
        total += num
```

num

6

```
    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
```

total

26

num

6



```
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
```

total	26
num	6

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

Output:
26

We implemented the sum function

Example 2: Accumulator Pattern

```
]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
```

Initialize variable

Loop over
characters in word

update total,
Add 1 to it

return variable

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

```
def numLetters(word):
```



```
    total = 0
```

total

0

```
    for letter in word:
```

```
        total += 1
```

```
    return total
```

```
word = "card"
```

```
print(numLetters(word))
```

Output:

Example 2: Accumulator Pattern

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

total 0

letter 'c'

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

Output:

Example 2: Accumulator Pattern

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



total	1
letter	'c'

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

Output:

Example 2: Accumulator Pattern

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



total 1

letter 'a'

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



Output:

Example 2: Accumulator Pattern

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



total	2
letter	'a'

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

Output:

Example 2: Accumulator Pattern

```
def numLetters(word):
```

```
    total = 0
```

total

2

```
    for letter in word:
```

```
        total += 1
```

letter

'r'

```
    return total
```

```
word = "card"
```

```
print(numLetters(word))
```

Output:

Example 2: Accumulator Pattern

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



total

3

letter

'r'

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

Output:

Example 2: Accumulator Pattern

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

total 3
letter 'd'

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

Output:

Example 2: Accumulator Pattern

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



total	4
letter	'd'

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

Output:

Example 2: Accumulator Pattern

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

total	4
letter	'd'

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

Output:

Example 2: Accumulator Pattern

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

total 4

letter 'd'

word = "card"

→ print(numLetters(word))

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 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

```
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

Initialize before loop

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

Loop over characters
in phrase

Update inside loop

Do something with
value after loop

range() Sequence

- Range generates a sequence of values
- `range(y)` – starts at 0 and goes up to but doesn't include `y`: `0 ... (y-1)`
 - `y` is an integer
- `range(x, y) : x ... (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

Not a list, but generates a sequence of numbers!

`range(5)`

`range(0, 5)`

`list(range(5))`

`[0, 1, 2, 3, 4]`

Convert to a list

`range(5)[0]`

`0`

`range(5)[4]`

`4`

`range(5)[5]`

ERROR!!!!!!

`range(5,10)`

`range(5,10)`

`list(range(5,10))`

`[5, 6, 7, 8, 9]`

`range(5,10)[3]`

`8`

`for x in range(3):`

`0`

`print(x)`

`1`

`2`

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

```
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

```
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

```
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

```
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:

```
for _ in range(4):  
    CODE
```


WOTO-1 – Accumulator, Range
<http://bit.ly/101s23-0214-1>

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

```
def mystery(lst):
```

```
    idxMax = 0
```

Other variable

```
    for i in range(len(lst)):
```

```
        if lst[idxMax] < lst[i]:
```

```
            idxMax = i
```

Useful expression
to track

Loop
variable

```
    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

```
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])
```

i	idxMax	lst[idxMax]	lst[i]	lst[idxMax] < lst[i]

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])
```

i	idxMax	lst[idxMax]	lst[i]	lst[idxMax] < lst[i]

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])
```

#1

i	idxMax	lst[idxMax]	lst[i]	lst[idxMax] < lst[i]
	0			

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])
```

i	idxMax	lst[idxMax]	lst[i]	lst[idxMax] < lst[i]
0	0			

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])
```

i	idxMax	lst[idxMax]	lst[i]	lst[idxMax] < lst[i]
0	0	lst[0] is 2	lst[0] is 2	False

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])
```

#1

i	idxMax	lst[idxMax]	lst[i]	lst[idxMax] < lst[i]
0	0	2	2	False
1	0			

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])
```

i	idxMax	lst[idxMax]	lst[i]	lst[idxMax] < lst[i]
0	0	2	2	False
1	0	lst[0] is 2	lst[1] is 12	True

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])
```



#2

i	idxMax	lst[idxMax]	lst[i]	lst[idxMax] < lst[i]
0	0	2	2	False
1	0 1	2	12	True

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])
```

i	idxMax	lst[idxMax]	lst[i]	lst[idxMax] < lst[i]
0	0	2	2	False
1	0 1	2	12	True
2	1			

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])

```

i	idxMax	lst[idxMax]	lst[i]	lst[idxMax] < lst[i]
0	0	2	2	False
1	0 1	2	12	True
2	1	lst[1] is 12	lst[2] is 4	False

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])

```

i	idxMax	lst[idxMax]	lst[i]	lst[idxMax] < lst[i]
0	0	2	2	False
1	0 1	2	12	True
2	1	12	4	False
3	1			

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])
```

i	idxMax	lst[idxMax]	lst[i]	lst[idxMax] < lst[i]
0	0	2	2	False
1	0 1	2	12	True
2	1	12	4	False
3	1	lst[1] is 12	lst[3] is 15	True

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])

#2

i	idxMax	lst[idxMax]	lst[i]	lst[idxMax] < lst[i]
0	0	2	2	False
1	1	2	12	True
2	1	12	4	False
3	3	12	15	True

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])
```

i	idxMax	lst[idxMax]	lst[i]	lst[idxMax] < lst[i]
0	0	2	2	False
1	0 1	2	12	True
2	1	12	4	False
3	1 3	12	15	True
4	3			

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])
```

i	idxMax	lst[idxMax]	lst[i]	lst[idxMax] < lst[i]
0	0	2	2	False
1	0 1	2	12	True
2	1	12	4	False
3	1 3	12	15	True
4	3	15	15	False

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])

i	idxMax	lst[idxMax]	lst[i]	lst[idxMax] < lst[i]
0	0	2	2	False
1	0 1	2	12	True
2	1	12	4	False
3	1 3	12	15	True
4	3	15	15	False

return 3

What is always true about the loop?

```
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])

i	idxMax	lst[idxMax]	lst[i]	lst[idxMax] < lst[i]
0	0	2	2	False
1	1	2	12	True
2	1	12	4	False
3	3	12	15	True
4	3	15	15	False

What is always true about the loop?

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

```
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])`

<code>i</code>	<code>idxMax</code>	<code>lst[idxMax]</code>	<code>lst[i]</code>	<code>lst[idxMax] < lst[i]</code>
0	0	2	2	False
1	0 1	2	12	True
2	1	12	4	False
3	1 3	12	15	True
4	3	15	15	False

What is always true about the loop?

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

```
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])

i	idxMax	lst[idxMax]	lst[i]	lst[idxMax] < lst[i]
0	0	2	2	False
1	0 1	2	12	True
2	1	12	4	False
3	1 3	12	15	True
4	3	15	15	False

WOTO-2 Loop Tracing

<http://bit.ly/101s23-0214-2>

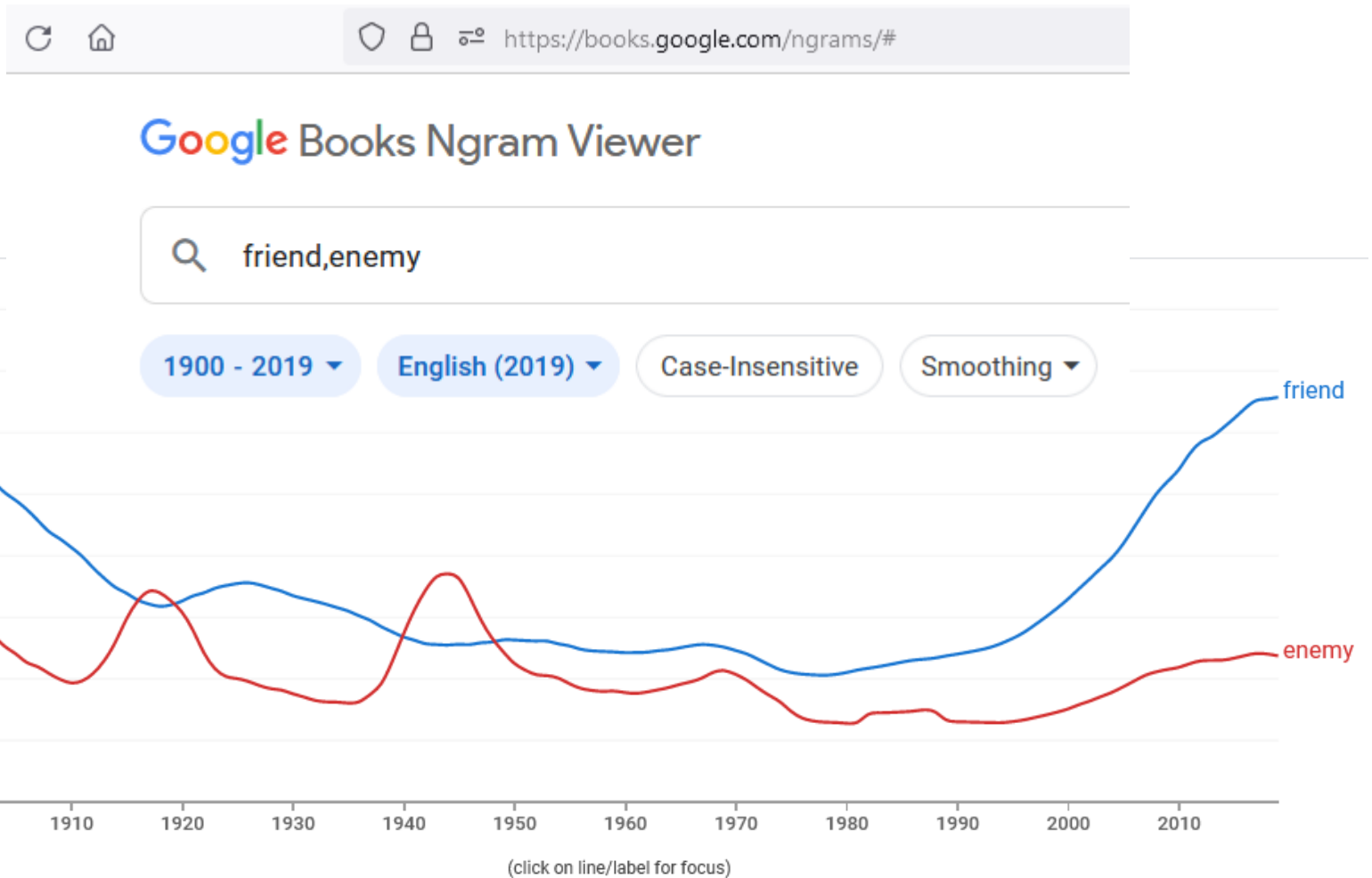
- **Remember the steps**
- **(1) Find the changing variable/expressions,**
- **(2) Create the table with these as the column**
- **(3) Each row is an iteration of the loop**

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

```
7 def lineCount(fname):  
8     """  
9     return # lines in file fname  
10    """  
11    f = open(fname)  
12    lc = 0  
13    for line in f:  
14        lc = lc + 1  
15  
16    f.close()  
17    return lc
```

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**

```
7  def lineCount(fname):
8      """
9      return # lines in file fname
10     """
11     f = open(fname)
12     lc = 0
13     for line in f:
14         lc = lc + 1
15
16     f.close()
17     return lc
```

lineCount function

```
7  def lineCount(fname):  
8      """  
9      return # lines in file fname  
10     """  
11     f = open(fname)  
12     lc = 0  
13     for line in f:  
14         lc = lc + 1  
15  
16     f.close()  
17     return lc
```

altCount function

```
19  def altCount(fname):  
20      """  
21      return # lines in file fname  
22      """  
23      f = open(fname)  
24      lc = len(f.readlines())  
25      f.close()  
26      return lc
```

main

```
28 ▶ if __name__ == "__main__":  
29     name = "data/poe.txt"  
30     pc = lineCount(name)  
31     print("# lines:",pc)  
32     pc2 = altCount(name)  
33     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")`

WOTO-3 Files

<http://bit.ly/101s23-0214-3>