

Compsci 101

Turtle, Bagels, Loop Tracing, Files

Part 1 of 4

Susan Rodger
Nicki Washington
February 23, 2021



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!

Prof. Beth Trushkowsky

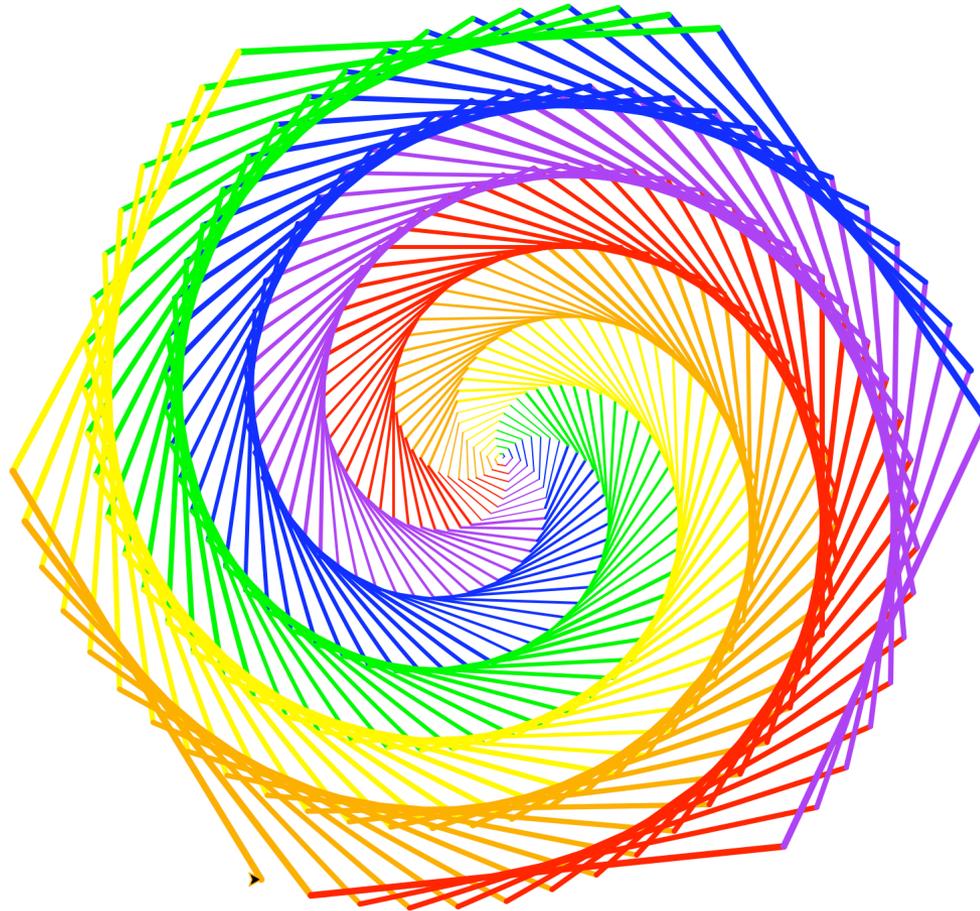
- Computer Science BS, Duke '07
- Computer Science MS '10, PhD '14, UC Berkeley
- Associate Professor, CS, Harvey Mudd College
- Her research involves leveraging human intelligence via crowdsourcing to create hybrid human/machine query processing systems.



PFTD

- Turtle
- Bagels APT
- Trace through loops
- Files

Run Turtle, Run



Turtle Programming

- **Must:**
 - Import turtle module
 - Create window/Screen
 - Last thing - exit on click
 - Create turtles to use, name/type/value
- **Review Turtle commands and concepts**
 - http://bit.ly/turtle_tutorial for more, and book
- **See Snowpeople.py, ColorMyWorld.py, and Spiro.py for some ideas**
 - Color, Position, Leaving Turtle where started
 - Many more commands than this



Put yourself in the turtle t...

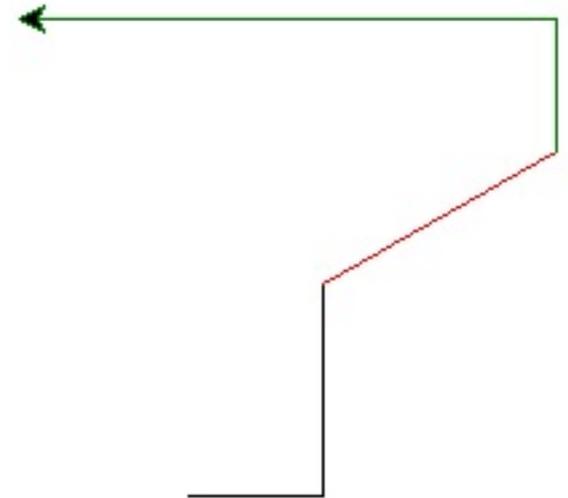
```
t.forward(50)           # turtle moves forward
                        # drawing a line
t.left(90)              # turtle turns to its left
t.pencolor("blue")     # change pen color
t.forward(100)         # turtle moves forward
                        # drawing line, new color
```

Example: simple.py

```
import turtle

def drawPicture(turt):
    t.forward(50)
    t.left(90)
    t.forward(80)
    t.pencolor('red')
    t.right(60)
    t.forward(100)
    t.pencolor('green')
    t.left(60)
    t.forward(50)
    t.left(90)
    t.forward(200)

if __name__ == '__main__':
    win = turtle.Screen()
    t = turtle.Turtle()
    drawPicture(t)
    win.exitonclick()
```



Example: Simple.py parts

```
import turtle ←
```

- Import at the top

```
]if __name__ == '__main__':  
    win = turtle.Screen()  
    t = turtle.Turtle()  
    drawPicture(t)  
]    win.exitonclick()
```

Example: Simple.py parts

```
import turtle
```

```
]if __name__ == '__main__':
```

```
→ win = turtle.Screen()    • Create window  
  t = turtle.Turtle()  
  drawPicture(t)  
] win.exitonclick()
```

Example: Simple.py parts

```
import turtle
```

```
]if __name__ == '__main__':
```

```
    win = turtle.Screen()
```

```
→ t = turtle.Turtle()
```

```
    drawPicture(t)
```

```
]    win.exitonclick()
```

- Create turtle

Example: Simple.py parts

```
import turtle
```

```
]if __name__ == '__main__':  
    win = turtle.Screen()  
    t = turtle.Turtle()  
    → drawPicture(t)  
    win.exitonclick()  
]
```

- Call function to draw

Example: Simple.py parts

```
import turtle
```

```
if __name__ == '__main__':  
    win = turtle.Screen()  
    t = turtle.Turtle()  
    drawPicture(t)  
    win.exitonclick()
```

- Close canvas when click on it

Example: Simple.py DrawPicture

```
def drawPicture(turt):  
    turt.forward(50)  
    turt.left(90)  
    turt.forward(80)  
    turt.pencolor('red')  
    turt.right(60)  
    turt.forward(100)  
    turt.pencolor('green')  
    turt.left(60)  
    turt.forward(50)  
    turt.left(90)  
    turt.forward(200)
```



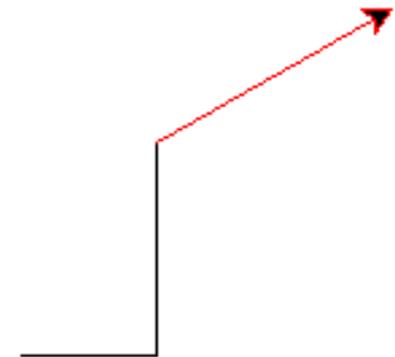
Example: Simple.py DrawPicture

```
def drawPicture(turt):  
    turt.forward(50)  
    turt.left(90)  
    turt.forward(80)  
    turt.pencolor('red')  
    turt.right(60)  
    turt.forward(100)  
    turt.pencolor('green')  
    turt.left(60)  
    turt.forward(50)  
    turt.left(90)  
    turt.forward(200)
```



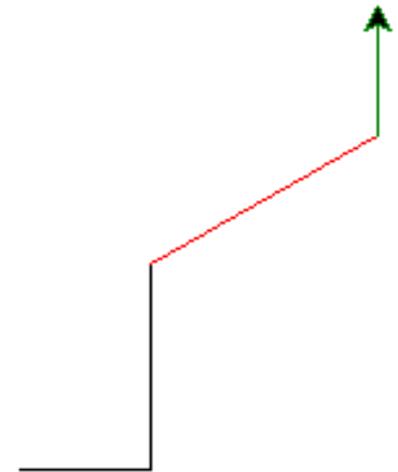
Example: Simple.py DrawPicture

```
def drawPicture(turt):  
    turt.forward(50)  
    turt.left(90)  
    turt.forward(80)  
    turt.pencolor('red')  
    turt.right(60)  
    turt.forward(100)  
    turt.pencolor('green')  
    turt.left(60)  
    turt.forward(50)  
    turt.left(90)  
    turt.forward(200)
```



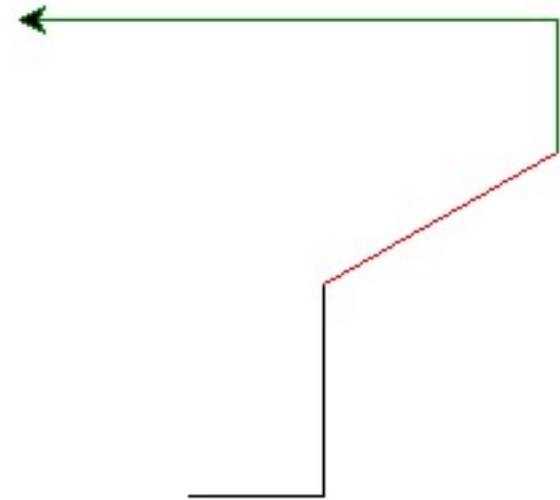
Example: Simple.py DrawPicture

```
def drawPicture(turt):  
    turt.forward(50)  
    turt.left(90)  
    turt.forward(80)  
    turt.pencolor('red')  
    turt.right(60)  
    turt.forward(100)  
    turt.pencolor('green')  
    turt.left(60)  
    turt.forward(50)  
    turt.left(90)  
    turt.forward(200)
```



Example: Simple.py DrawPicture

```
def drawPicture(turt):  
    turt.forward(50)  
    turt.left(90)  
    turt.forward(80)  
    turt.pencolor('red')  
    turt.right(60)  
    turt.forward(100)  
    turt.pencolor('green')  
    turt.left(60)  
    turt.forward(50)  
    turt.left(90)  
    turt.forward(200)
```



Compsci 101

Turtle, Bagels, Loop Tracing, Files

Part 2 of 4

Susan Rodger
Nicki Washington
February 23, 2021



What are key concepts in Spiro.py?

```
8 import turtle
9
10 def draw(turt):
11     colors = ['red', 'purple', 'blue', 'green', 'yellow', 'orange']
12     turt.speed(0)
13     for x in range(360):
14         turt.pencolor(colors[x % 6])
15         turt.width(x/100 + 1)
16         turt.forward(x)
17         turt.left(59)
18
19 if __name__ == '__main__':
20     win = turtle.Screen()
21     t = turtle.Turtle()
22     draw(t)
23     win.exitonclick()
```

Import turtle

1 – slowest
10 – fastest
0 – No animation

Create screen/window

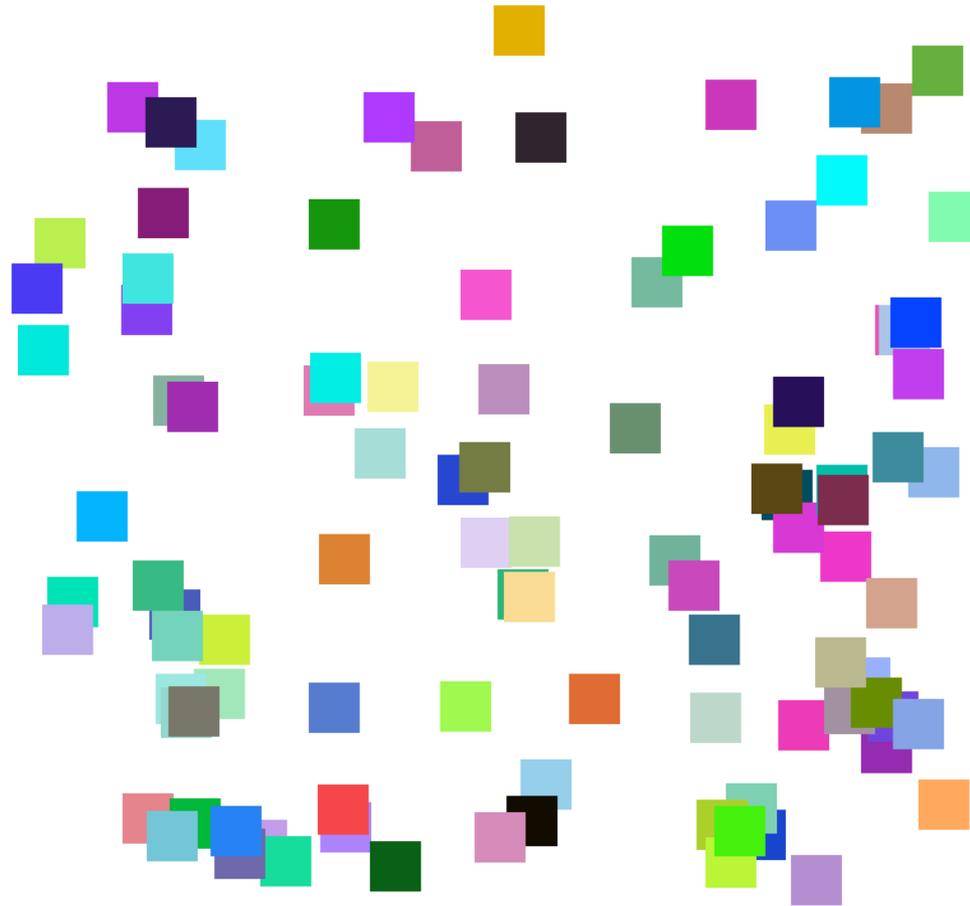
Create turtle

Close on click

Useful turtle functions

- **forward(n)/backward(n)** – move turtle n pixels
- **left(n)/right(n)** – turn turtle n degrees
- **pendown()/pendup()** – whether actually drawing
- **setposition(x, y)** – puts turtle in this (x,y) coordinate (a.k.a. **goto**, **setpos**)
- **sethead(n)** – points turtle in this direction (n=0 is east)
- Many more in documentation!
 - <https://docs.python.org/3/library/turtle.html>

ColorMyWorld.py



Turtle Concepts

- **Create a screen so you can ..**
 - Exit On Click
 - Some other Screen Functions
- **Create a turtle so you can ...**
 - Move and draw using the turtle
- **Drawing Concepts**
 - Pen [up and down]
 - Fill
 - Color
 - Position

Compsci 101

Turtle, Bagels, Loop Tracing, Files

Part 3 of 4

Susan Rodger
Nicki Washington
February 23, 2021



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
```

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

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

```
            idxMax = i
```

```
    return idxMax
```

Other variable

Useful expression
to track

Loop
variable

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]

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			

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

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	2	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	12	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	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])

#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

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	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]` 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])

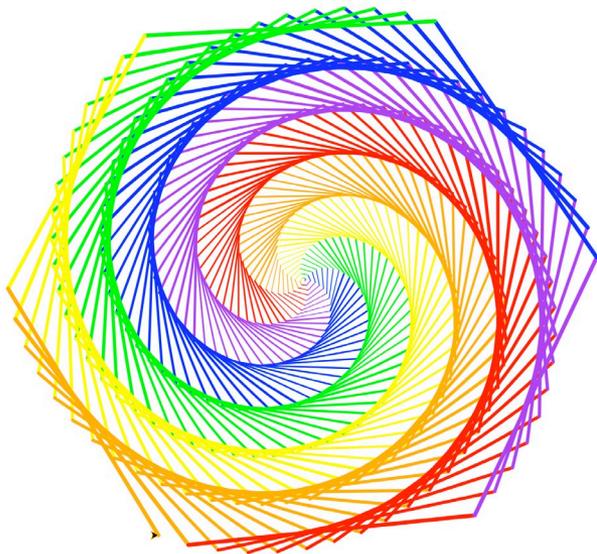
i	idxMax	lst[idxMax]	lst[i]	lst[idxMax] < lst[i]
0	0	2	2	False
1	0 1	2	12	True
2	1	12	4	False
3	1 3	12	15	True
4	3	15	15	False

Compsci 101

Turtle, Bagels, Loop Tracing, Files

Part 4 of 4

Susan Rodger
February 23, 2021

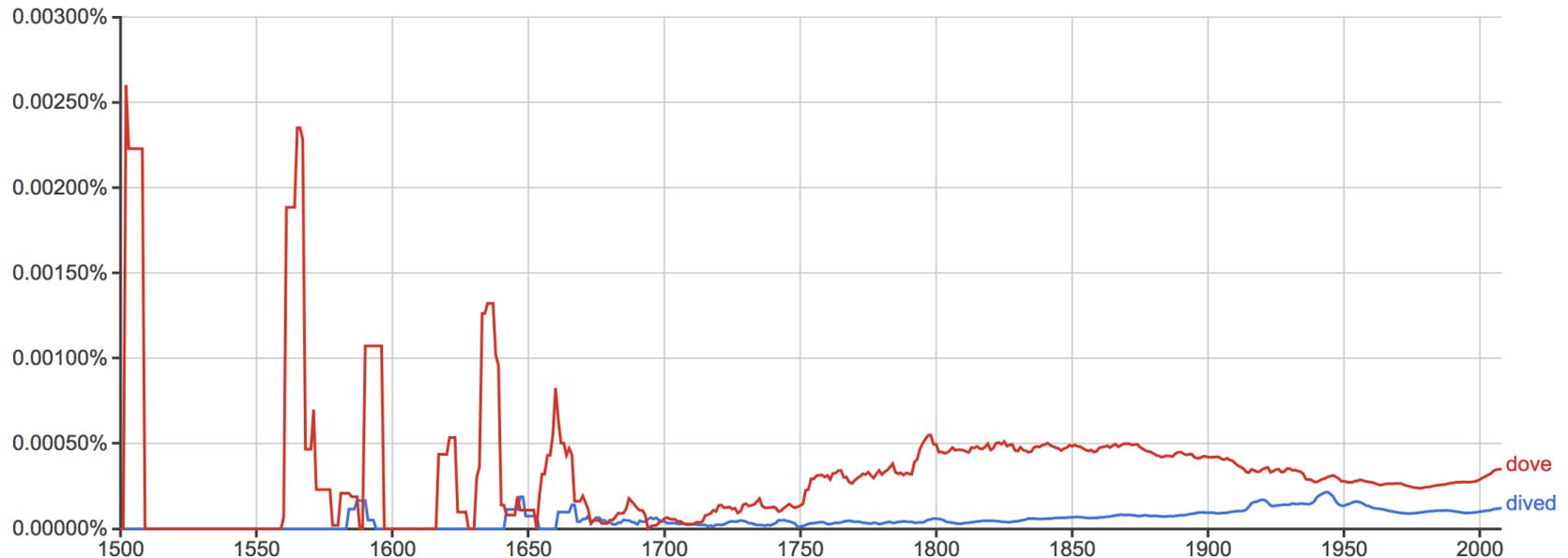


Examples of Processing Data

- 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 dove to dived
- <https://www.youtube.com/watch?v=tFW7orQsBuo>



Sequences, Repetition

- Parameters? What are they to this query?
 - https://books.google.com/ngrams/graph?content=terrorism%2Cpatriot&year_start=1800&year_end=2000&corpus=15&smoothing=3



What can the URL tell you?

Sequences, Repetition

- Parameters? What are they to this query?

Search words

https://books.google.com/ngrams/graph?content=terrorism%2Cpatriot&year_start=1800&year_end=2000&corpus=15&smoothing=3

Year start search

Year end search



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'



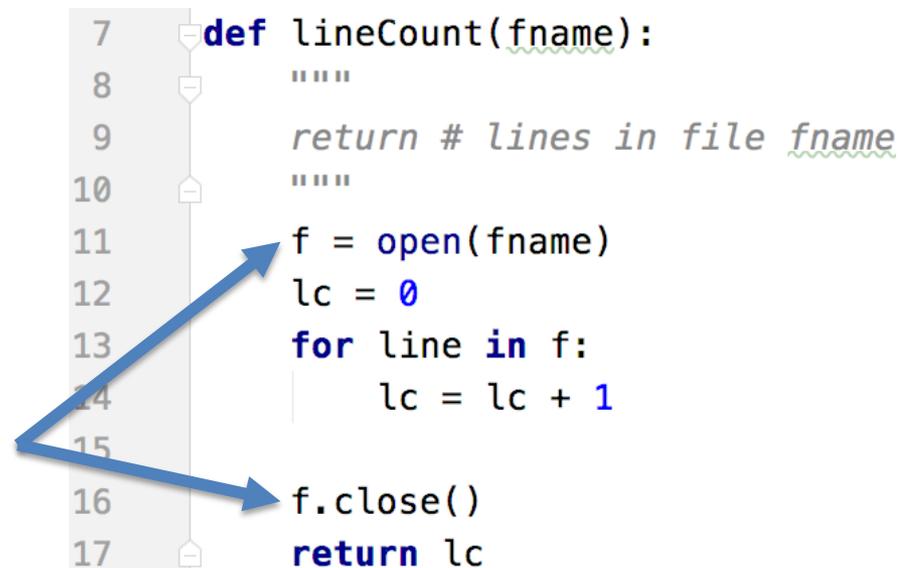
File Pattern: One line at a time

- 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

```
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")`

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()**

FileStuff.py: avgWord

```
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
```