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
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**

• **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
```python
import turtle

def drawPicture(turt):
    t.forward(50)
    t.left(90)
    t.forward(80)
    t.forward(100)
    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

```python
import turtle

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

```python
import turtle

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

```python
import turtle

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

- Create turtle
Example: Simple.py parts

```python
import turtle

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

• Call function to draw
Example: Simple.py parts

```python
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)
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?

```python
import turtle

def draw(turt):
    colors = ['red', 'purple', 'blue', 'green', 'yellow', 'orange']
    turt.speed(0)
    for x in range(360):
        turt.pencolor(colors[x % 6])
        turt.width(x/100 + 1)
        turt.forward(x)
        turt.left(59)

if __name__ == '__main__':
    win = turtle.Screen()
    t = turtle.Turtle()
    draw(t)
    win.exitonclick()
```
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](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. \textit{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

```
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])
```
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>
</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

---

```
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

```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>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>
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></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

```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</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

### Table

<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>

### Code Snippet

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

```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>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>
Examples of Processing Data

• Lecture 1: count letters in Bible

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

- Ngram informs how words evolve
- From dove to dived
- [https://www.youtube.com/watch?v=tFW7orQsBuo](https://www.youtube.com/watch?v=tFW7orQsBuo)
Sequences, Repetition

• Parameters? What are they to this query?

What can the URL tell you?
Sequences, Repetition

- Parameters? What are they to this query?

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
lineCount function

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()}, \texttt{f.read()}, \texttt{f.close()}
  • Just like: \texttt{st.lower()}, \texttt{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()`
```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
```