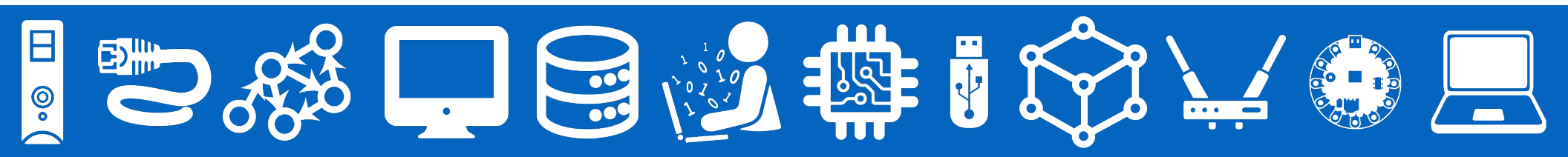


CSI 34:

Wrap-Up

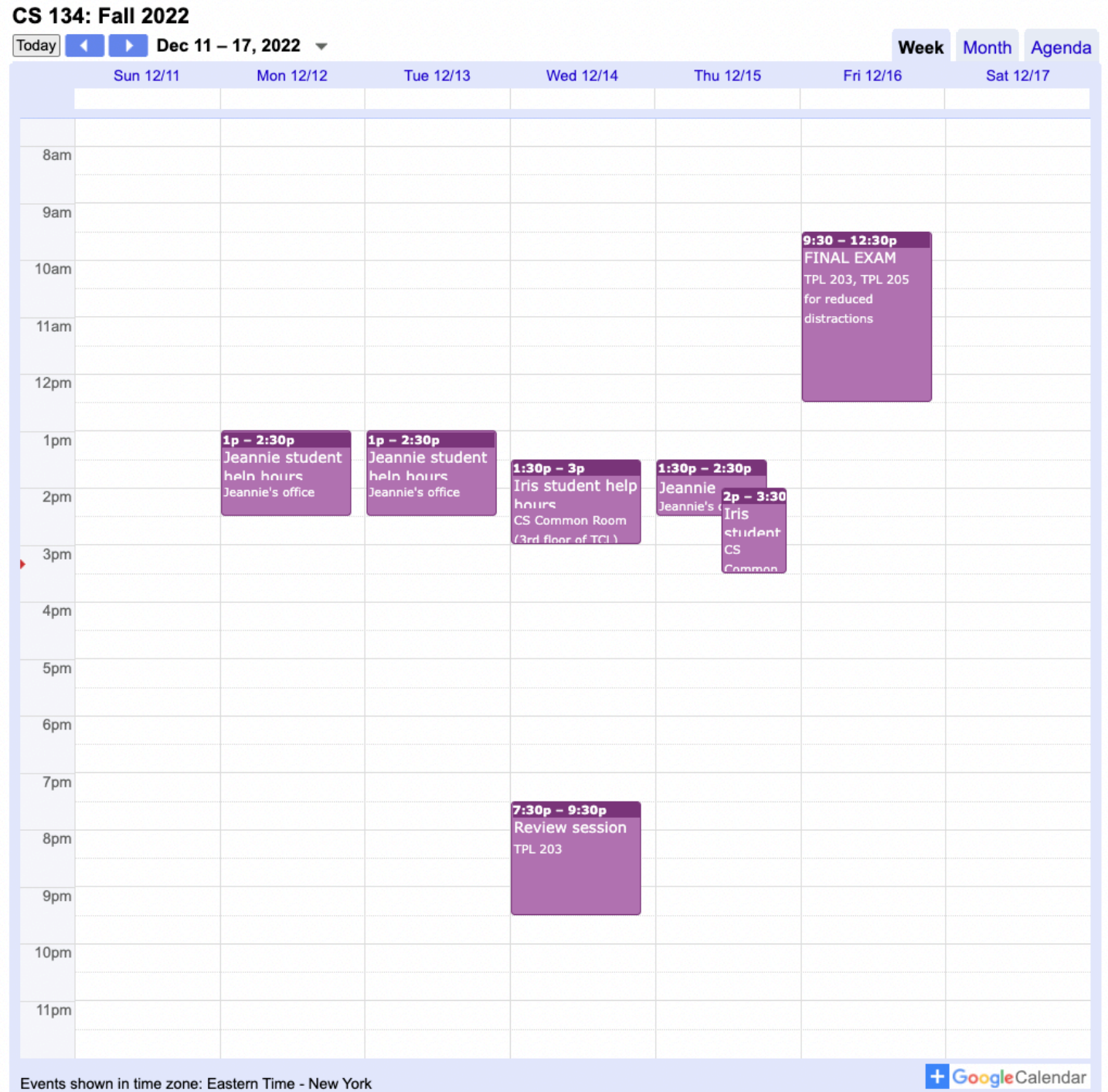


# Announcements & Logistics

- **Lab 9 Grading:** Coming soon!
- **Final exam:**
  - **Fri Dec 16 @ 9:30am in TPL 203**
    - **Reduced distractions/extra time TPL 205**
  - **2 hour closed book exam**
  - Cumulative w/ more weight on topics post-midterm topics
  - Practice problems are posted; review lecture slides, jupyter notebooks, HWs, and labs
  - Format will be very similar to midterm
- **Review session: Wed Dec 14 @ 7:30pm-9:30pm in TPL 203**
  - Very informal, come ask us questions

# Student Help Hours Next Week

(Check webpage  
for updates!)



# Last Time

- Reviewed OOP concepts using Python and Java as examples
  - A **class** vs an **instance** of the class
  - **Attributes** (instance variables) and `__slots__`
  - **Accessor** and **mutator** methods: getters, setters
  - **Scope**: public, private and protected (or `_` and `__` in Python)
  - Special methods and operator/function overloading

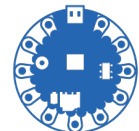
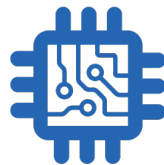
# Today's Plan

- Summarize **main topics** covered in CS 134 this semester
- Complete **course evals**
  - We'll end lecture early to leave time for you to fill out evals



# Optional Fun Stuff:

## Python & Webpages



What is a  
webpage?

“Ten movies streaming across that, that Internet, and what happens to your own personal Internet? I just the other day got... an Internet was sent by my staff at 10 o'clock in the morning on Friday. I got it yesterday [Tuesday]. Why? Because it got tangled up with all these things going on the Internet commercially. [...]

They want to deliver vast amounts of information over the Internet. And again, the Internet is not something that you just dump something on. It's not a big truck.

**It's a series of tubes.”**

**NO!**



US Senator Ted Stevens (R-Alaska) in 2006, Head of the committee regulating Net Neutrality



A webpage is just a publicly accessible file on a computer somewhere.



Learn more: <https://youtu.be/AEaKrq3SpW8>

# HTML

- **H**yper**T**ext **M**arkup **L**anguage
- Specifies how to format text for your Internet Browser
  - Different tags/symbols specify how computer should display text
  - Markup language, not a programming language!

# Try This...

CS134 Simple Page

Not Secure | cs.williams.edu/~cs134/basic.html


Hello CS 134! This is a simple web page.

Looking for Jeannie? Click [here](#).



Looking for Iris? Click [here](#).

Looking for Pixel? Click [here](#).

Here are some images.



**Williams**



Back

Forward

Reload

Save As...

Print...

Cast...

Search Images with Google Lens

Create QR Code for this Page

Translate to English

LastPass

**View Page Source**

Inspect

- Right-click a webpage
- "View Page Source"

# Try This...

```
<html>
  <head>
    <title>CS134 Simple Page</title>
  </head>

  <body>
    Hello CS 134!  This is a simple web page.

    <br><br>
    Looking for Jeannie?  Click <a href="http://www.cs.williams.edu/~jeannie">here</a>.

    <br><br>
    Looking for Iris?  Click <a href="http://www.cs.williams.edu/~iris">here</a>.

    <br><br>
    Looking for Pixel?  Click <a href="https://www.cs.williams.edu/~iris/website/img/HAILab.jpg">
    here</a>.

    <br><br>
    Here are some images.
    <br><br>
    
    <br><br>
    
    <br><br>
    

  </body>
</html>
```

- Copy/Paste/Save with .html file extension in a text editor (like VS Code)

# Try This...

```
<html>
  <head>
    <title>CS134 Simple Page</title>
  </head>

  <body>
    <font color="blue">Hello CS 134! This is a simple web page.</font>

    <br><br>
    Looking for Jeannie? Click <a href="http://www.cs.williams.edu/~jeannie">here</a>.

    <br><br>
    Looking for Iris? Click <a href="http://www.cs.williams.edu/~iris">here</a>.

    <br><br>
    Looking for Pixel? Click <a href="https://www.cs.williams.edu/~iris/website/img/HAILab.jpg">
    here</a>.

    <br><br>
    Here are some images.
    <br><br>
    
    <br><br>
    
    <br><br>
    

  </body>
</html>
```

- Make a small change. Save and view file in a web browser.

# Try This...



# HTML

- `<h1>Text goes here</h1>` ➔ Makes a level 1 heading
- Guess: there's also an `<h2></h2>`, and `<h3></h3>`, and ...
- `<b>Text goes here</b>` ➔ Makes the text bold (also `<strong>`)
- `<i>Text goes here</i>` ➔ Makes the text italic (also `<em>`)
- `<a href="http://url-here.edu">Link Text here</a>` ➔ Makes a hyperlink
- `<font face="courier">Text goes here</font>` ➔ Changes the font
- `<font size="+2">Text goes here</font>` ➔ Changes font size
- `<font color="green">Text goes here</font>` ➔ Changes font color
- `<p>Text goes here</p>` ➔ Paragraph definition (~2 newlines)
- `<br>` ➔ Line break (~1 newline)

# HTML Header

- `<html>` ➡ Defines what markup language is being used
- `<head>` Text & Tags in here are part of the header `</head>`
- `<title>` This title appears in the web browser `</title>`
- `<body>` Text & Tags in here are part of the body text `</body>`
- `</html>` ➡ Ends HTML file

```
<html>
  <head>
    <title>CS134 Simple Page</title>
  </head>

  <body>
    Hello CS 134! This is a simple web page.

    <br><br>
    Looking for Jeannie? Click <a href="http://www.cs.williams.edu/~jeannie">here</a>.

    <br><br>
    Looking for Iris? Click <a href="http://www.cs.williams.edu/~iris">here</a>.
```



# Pulling Source Code from Web Pages

```
terminal% pip install requests
```

```
>>> import requests
>>> r = requests.get('http://www.cs.williams.edu/~cs134/basic.html')
>>> r.text

'<html>\n  <head>\n    <title>CS134 Simple Page</title>\n  </\n  head>\n\n  <body>\n    Hello CS 134!  This is a simple web page.\n\n    <br><br>\n    Looking for Jeannie?  Click <a href="http://\nwww.cs.williams.edu/~jeannie">here</a>.\n\n    <br><br>\n    Looking for Iris?  Click <a href="http://www.cs.williams.edu/\n~iris">here</a>.\n\n    <br><br>\n    Looking for Pixel?  Click <a\nhref="https://www.cs.williams.edu/~iris/website/img/\nHAILab.jpg">here</a>.\n\n    <br><br>\n    Here are some images.\n\n    <br><br>\n    \n\n    <br><br>\n    <img\nsrc="http://sysnet.cs.williams.edu/williams.gif" alt="seal">\n\n    <br><br>\n    \n\n  </body>\n</html>\n\n  \n\n  \n'
```

<http://docs.python-requests.org/en/master/user/quickstart/>

# Processing Source Code from Web Pages

- If you want to parse the HTML text from a string, the Beautiful Soup module is recommended:
  - <https://beautiful-soup-4.readthedocs.io/en/latest/>
- `terminal% pip install beautifulsoup4`

# Processing Source Code from Web Pages

```
>>> from bs4 import BeautifulSoup
>>> soup = BeautifulSoup(r.text, 'html.parser')
>>> print(soup.prettify())
```

---

```
<html>
<head>
  <title>
    CS134 Simple Page
  </title>
</head>
<body>
Hello CS 134!  This is a simple web page.
<br/>
<br/>
Looking for Jeannie?  Click
<a href="http://www.cs.williams.edu/~jeannie">
  here
</a>
.
<br/>
<br/>
Looking for Iris?  Click
<a href="http://www.cs.williams.edu/~iris">
  here
</a>
```

# Processing Source Code from Web Pages

```
>>> soup.title
```

```
<title>CS134 Simple Page</title>
```

```
>>> soup.title.name
```

```
'title'
```

```
>>> soup.title.string
```

```
'CS134 Simple Page'
```

```
>>> soup.title.parent.name
```

```
'head'
```

```
>>> soup.img
```

```

```

# Processing Source Code from Web Pages

```
>>> soup.a
```

```
<a href="http://www.cs.williams.edu/~jeannie">here</a>
```

```
>>> soup.find_all('a')
```

```
[<a href="http://www.cs.williams.edu/~jeannie">here</a>,  
  <a href="http://www.cs.williams.edu/~iris">here</a>,  
  <a href="https://www.cs.williams.edu/~iris/website/img/  
HAILab.jpg">here</a>]
```

# Extracting All URLs

```
for link in soup.find_all('a'):  
    print(link.get("href"))
```

---

`http://www.cs.williams.edu/~jeannie`

`http://www.cs.williams.edu/~iris`

`https://www.cs.williams.edu/~iris/website/img/HAILab.jpg`

# See beautifulsoup4 documentation

Beautiful Soup

latest

Search docs

- Beautiful Soup Documentation
- Quick Start
- Installing Beautiful Soup
- Making the soup
- Kinds of objects
- Navigating the tree
- Searching the tree
- Modifying the tree
- Output
- Specifying the parser to use
- Encodings
- Line numbers
- Comparing objects for equality
- Copying Beautiful Soup objects
- Parsing only part of a document
- Troubleshooting
- Translating this documentation
- Beautiful Soup 3

Beautiful Soup then parses the document using the best available parser. It will use an HTML parser unless you specifically tell it to use an XML parser. (See [Parsing XML](#).)

## Kinds of objects

Beautiful Soup transforms a complex HTML document into a complex tree of Python objects. But

Lots more beautifulsoup4 can do!  
Learning the importance of documentation!  
<https://beautiful-soup-4.readthedocs.io/en/latest/>

```
soup = BeautifulSoup('<b class="boldest">Extremely bold</b>')  
tag = soup.b  
type(tag)  
# <class 'bs4.element.Tag'>
```

Tags have a lot of attributes and methods, and I'll cover most of them in [Navigating the tree](#) and [Searching the tree](#). For now, the most important features of a tag are its name and attributes.

### Name

Every tag has a name, accessible as `.name`:

```
tag.name  
# u'b'
```

If you change a tag's name, the change will be reflected in any HTML markup generated by Beautiful Soup:

```
tag.name = "blockquote"  
tag  
# <blockquote class="boldest">Extremely bold</blockquote>
```

### Attributes

# What are we doing?!

- So now we can *scrape* HTML data from webpages...
- ...and parse the data so we can pull out meaningful text...

## Why might we want to pull source code from the web?

- Maybe you're:
  - building a web crawler, documenting all the webpages on the Internet so their text can be searchable...
  - a sports recruiter and you need to pull wins/losses data from local amateur leagues...
  - a designer building software to make stock market transactions based on the weather...
  - a PR firm tracking in vivo mentions of particular products or brands
  - a humanitarian gathering evidence on organized crime groups
  - an AI researcher trying to generate new paint color names



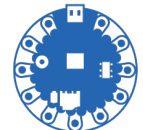
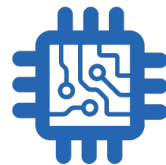
# What are we doing?!

- Python has **lots** more accessible modules that do other fun things
  - Play music
  - Process images
  - Generate text
  - Statistical operations
  - Among others!

# Take-away

- Python is a **powerful tool** that:
  - Processes, manipulates, organizes data
  - Accesses data
  - Creates beautiful things: art, solutions, puzzles, ...
  - Expands human capabilities
- **But also: communicates** complex computational ideas

# Course Wrap-Up



# CSI 34 in a Nutshell



- We have covered many topics this semester!
- We started out learning the basics of Python and programming in general
- **Pre-midterm**
  - **Types & Operators** ( int, float, %, //, /, concatenation, etc)
  - **Functions** (variable scope, return vs print, defining vs calling functions)
  - **Booleans and conditionals** (if elif else)
  - **Iteration:** for loops, while loops, nested loops, accumulation variables in loops
  - **Sequences:** strings (string methods, in/not in, iteration, etc) , lists (list methods, append, extend, etc), ranges, tuples, lists of lists
  - **File reading:** with ... as , strip(), split()
  - **Mutability** and **aliasing**

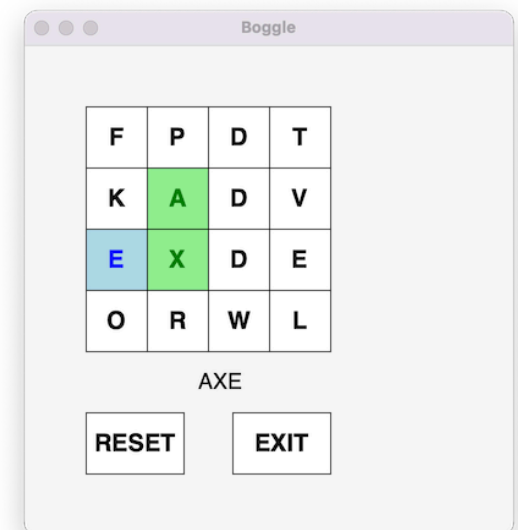
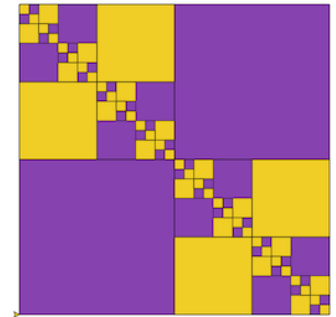
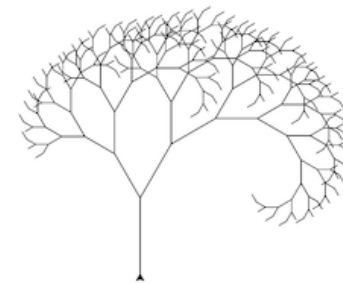
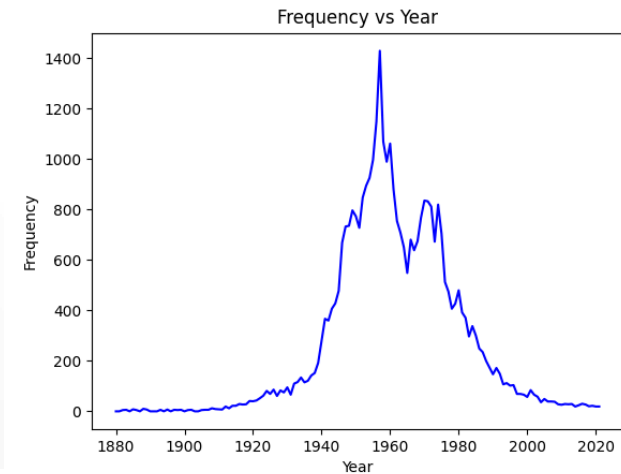
# CS 134 in a Nutshell

- Then we moved on to more advanced CS topics
- **Post-midterm**
  - **Data structures:** More tuples, dictionaries, sets
    - **Sorting** data with key functions
  - **Recursion:** recursive methods and classes
    - **Graphical recursion** with **turtle** graphics
  - **Classes, Objects, and OOP**
    - attributes, `__slots__`, special methods, getters, setters, inheritance
    - “Bigger” OOP Examples: Tic-Tac-Toe, Boggle, LinkedList
  - **Advanced topics:**
    - Efficiency (Big-O), Searching and sorting, Iterators, Python vs. Java

# Labs

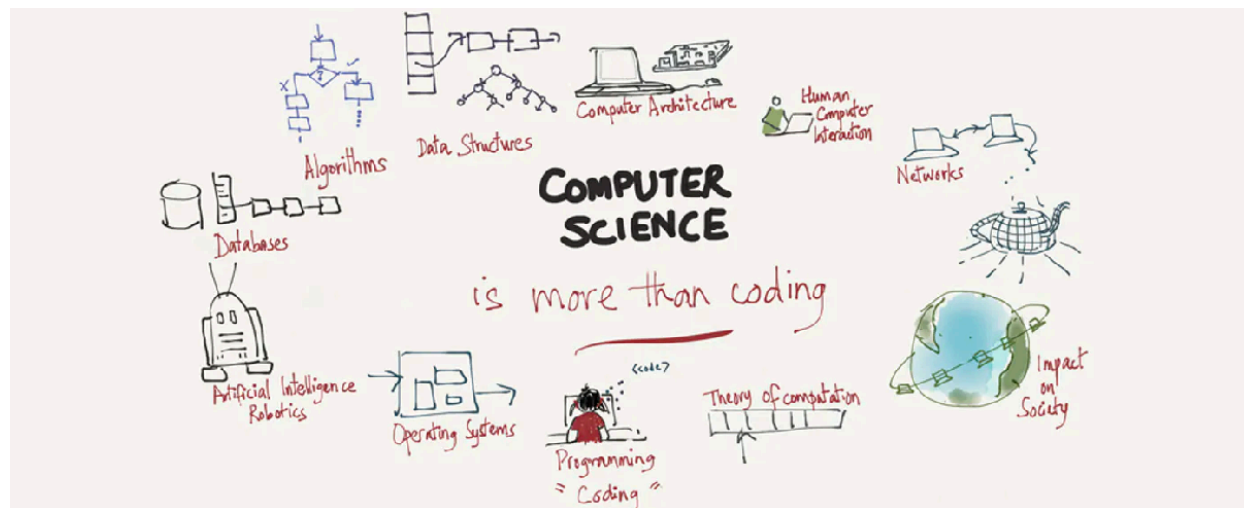
- Hello, World!
- Day of the week (conditionals)
- Word puzzles (strings and loops)
- Voting algorithms (lists and loops)
- Debugging
- Name popularity (dictionaries and plotting)
- Recursion
- Autocomplete (classes and methods)
- Boggle (OOP, more classes and inheritance)
- Selection sort (Java)

puzzle



# Takeaway: What is Computer Science?

- Computer science  $\neq$  computer programming!
- Computer science is the study of what computers [can] do; programming is the practice of making computers do useful things
- Programming is a big part of computer science, but **there is much more to CS** than just writing programs!
- Another part of CS is **computational thinking**



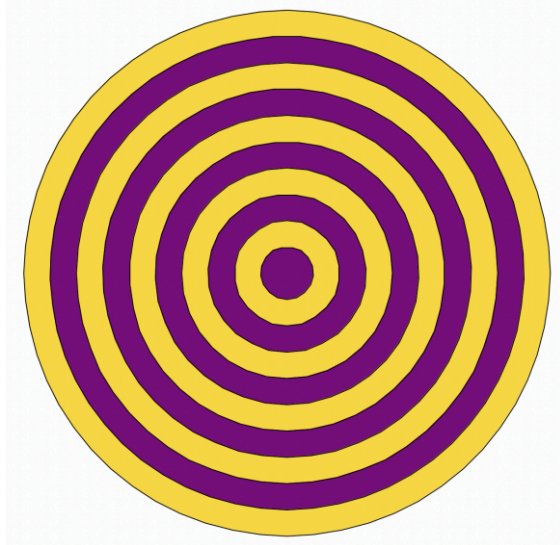
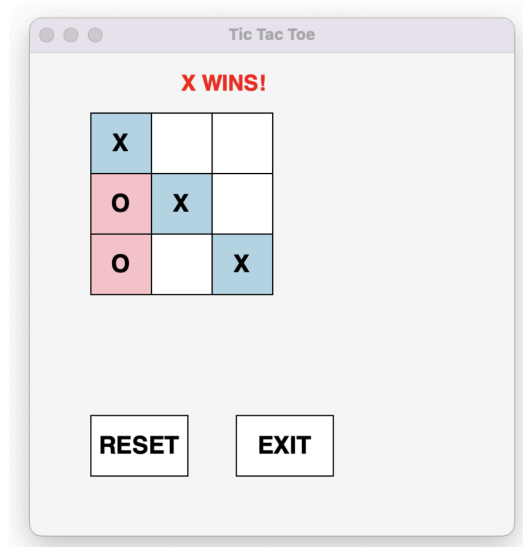
# Take away: Computational Thinking

- Computational thinking allows us to develop solutions for complex problems. We present these solutions such that a computer, a human, or both, can understand.
- Four pillars of CT:
  - **Decomposition** - break down a complex problem into smaller parts
  - **Pattern recognition** – look for similarities among and within problems
  - **Abstraction** – focus on important information only, ignore irrelevant details
  - **Algorithms** - develop a step-by-step solution to the problem
- A computer can perform a billion of operations per second, but computers only do exactly what you tell them to do!
- In this course we will learn **learned** how to 1) use CT to develop algorithms for solving problems, and 2) implement our algorithms through computer programs



# Goals from Lecture I

- Abstraction and modularity
- Representing knowledge with data structures
- Iteration and recursion as computational tools
- Divide and conquer problem solving strategies
- Testing and debugging
- Organizing and dealing with data
- Plotting and visualizing data
- Playing with python graphics
- Transitioning from Python to Java (and beyond!)



# Beyond CS I 34

- For those interested in continuing on the CS path:
  - Obvious next step: take **CSI36** + **Math 200**
  - Practice more Java over winter break: redo our labs in Java!
- In general, if you enjoy **puzzles and programming**, there are many ways to practice these skills:
  - Try Project Euler: Math + CS puzzles
  - MIT course: The missing semester of your CS education
- Staying connected with CS as non-majors:
  - Can still take CS I 36 and other courses!
  - Come talk to us for more ideas

# Course Evals Logistics

- Two parts: **(1) SCS form**, **(2) Blue sheets** (both online)
- Your feedback helps us improve the course and shape the CS curriculum
  - Your responses are **confidential** and we only receive anonymized comments after we submit our grades
  - We appreciate your constructive feedback
- **SCS forms** are used for evaluation, **blue sheets are open-ended** comments directed only to your instructor

*To access the online evaluations, log into **Glow** ([glow.williams.edu](http://glow.williams.edu)) using your regular Williams username and password (the same ones you use for your Williams email account). On your Glow dashboard you'll see a course called "**Course Evaluations**." Click on this and then follow the instructions you see on the screen. If you have trouble finding the evaluation, you can ask a neighbor for help or reach out to [ir@williams.edu](mailto:ir@williams.edu).*

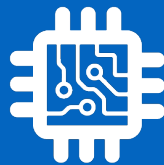
# Thank you!

## WE MADE IT!

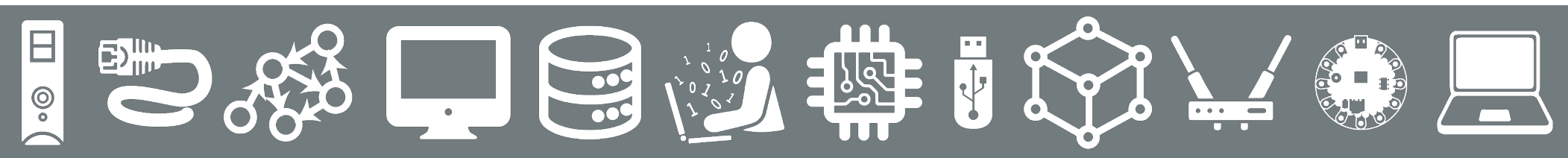
- You all should be proud of how much you've learned!
- **Thank you** for your patience and enthusiasm during these somewhat crazy times
- Good luck on finals and have a great break!



# The end!



# *Leftover Slides*



# HTML Tables

- `<table>` ➡ Begins the table
- `<tr>` ➡ Begins a row
  - `<td>Text in cell 1</td>` ➡ Adds a column *within* the row
  - `<td>Text in cell 2</td>` ➡ Adds a column *within* the row
  - `<td>Text in cell 3</td>` ➡ Adds a column *within* the row
- `</tr>` ➡ Ends a row
- `<tr>` ➡ Begins 2nd row
  - `<td>Text in cell 4</td>` ➡ Adds a column *within* 2nd row
- `</tr>` ➡ Ends 2nd row
- `</table>` ➡ Ends the table

# HTML Bulleted Lists

`<ol>` ➔ Begins numbered list (i.e., **o**rdered **l**ist)

1. `<li>`Text goes here`</li>`

2. `<li>`Another numbered bullet item`</li>`

`</ol>` ➔ Ends numbered list

`<ul>` ➔ Begins bulleted list(i.e., **u**nordered **l**ist)

• `<li>`Text goes here`</li>`

• `<li>`Another numbered bullet item`</li>`

• `<li>`Yet another numbered bullet item`</li>`

`</ul>` ➔ Ends bulleted list