EECS 485 Web Systems Introduction

eecs485.org

Slides by Andrew DeOrio

Announcements

- Please read the syllabus and eecs485.org
- Lab starts this week
- Project 1 due soon

About DeOrio

- Triple Wolverine: undergrad (EE), masters (EE) and PhD (CSE)
- Research in engineering education and interdisciplinary computing
- Taught 280, 281, 370, 485, Creative Process
- Worked at Intel, automotive, startups
- Chickens!
- Goats!

About EECS 485

- "How the web works"
- Full stack
  - Data-center scale backend systems
  - In-browser front end software systems
- Expect breadth
  - The web encompasses many other areas of computer science
  - Overlap with 388, 390, 445, 475, 489, 482, 486, 490, 491, et al.

Some things you'll build

- Instagram
- Hadoop distributed compute engine
- Google, circa mid 2000's

Some technologies you'll use

- Linux
  - System administration for deployment
- Shell scripting
- HTML/CSS
- Python
  - Flask web framework with jinja2 template engine
  - Thread and socket libraries
- SQL
  - SQLite database
- JavaScript
  - React/JS framework
- Hadoop

Independent learning

- Independent learning is a goal
  - By the end of this course, we hope you'll feel comfortable learning a new web technology independently using developer documentation
  - We're still here to help!

- You'll have a lot to learn on your own
  - HTML, CSS
  - Python (we'll help in lecture)
  - Flask framework
  - JavaScript (we'll help in lecture)
  - React JS framework

Agenda

- About EECS 485
- **Logistics**
- The request response cycle
- Python conceptual model

Course format

- Lectures
  - Mostly in-person
  - Live and studio recordings released each week
  - No attendance required
  - Attend any section at any time
- Labs
  - In-person
  - One lab will be recorded
  - No attendance required
  - Attend any section at any time
- Office hours
  - Mix of in-person and remote
  - See calendar on eecs485.org

Communication

- eecs485.org
  - Links to all course resources
- edstem.org
  - Best for technical questions and project updates
- eecs485staff@umich.edu
  - Best for non-technical questions
- awdeorio@umich.edu (individual professor email addresses)
  - Best for confidential matters

Exams

- Put the dates in your calendar
  - See eecs485.org
  - No make-up exam
- In-person, paper
- Cheat sheet OK
- Format
  - Part multiple choice
  - Part code and free response
- Content
  - Concepts from lecture
  - Code from projects

Labs

- **YES** we have lab this week
- If you're on the waitlist, attend any lab or watch the recording
  - Materials are posted on eecs485.org

- Expect first segment project intro and lecture review
- Second segment hands-on help with projects

Projects

- Project 1: Templated Static Site Generator. An Instagram clone implemented with a templated static site generator.
- Project 2: Server-side Dynamic Pages. An Instagram clone implemented with server-side dynamic pages.
- Project 3: Client-side Dynamic Pages. An Instagram clone implemented with client-side dynamic pages.
- Project 4: MapReduce. A single machine, multi-process, multi-threaded server that executes user-submitted MapReduce jobs.
- Project 5: Search Engine. A scalable search engine similar to Google or Bing.

Project groups

- Project 1: Independent
- Projects 2-5: Groups of 2-3
- You can modify your group between projects
- In exceptional cases, you may request group dissolution
- For those retaking the course: you may work in a group or you may reuse your old code, but not both.
- We can help you find a group (optional). Watch for email.

Project grading

- Autograder linked from eecs485.org
- 3 submissions per day per group
- We grade the best submission

- Public testcases
  - Visible on autograder before the deadline
  - Full testcase source code published
  - Includes style grading
  - Most of the points
- Private testcases
  - Visible after the deadline
  - Not published

Grades

Projects (5 x 10%)	50%
Midterm	25%
Final	25%

- Draft letter grade after the midterm

Agenda

- About EECS 485
- Logistics
- **The request response cycle**
- Python conceptual model

The request response cycle

- The request response cycle is how two computers communicate with each other on the web

1. A client requests some data
2. A server responds to the request

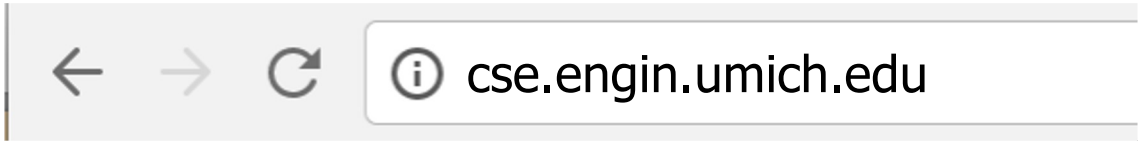
client

internet

server

The request response cycle

- A client requests a web page



- A server responds with an HTML file
  - It might create the content on-the-fly
-

```
<!DOCTYPE html>
...
```

What does a server respond with?

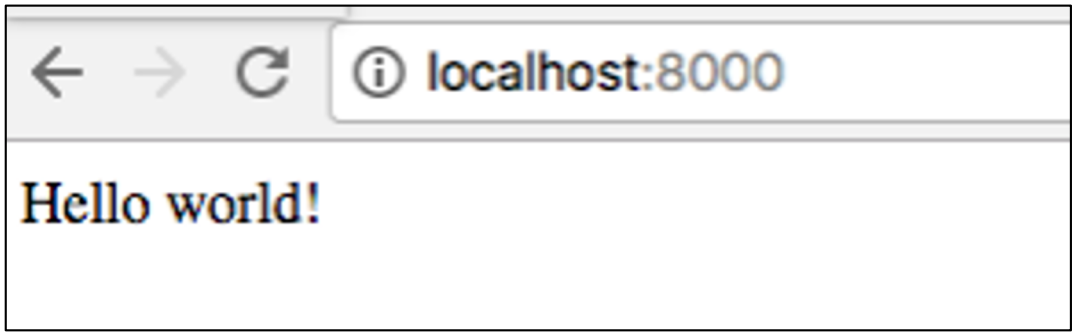
- A server might respond with different kinds of files
  - HTML
  - CSS
  - JavaScript

- We'll talk about JavaScript later in the semester

HTML

- HTML describes the content on a page
- Example: `index.html`

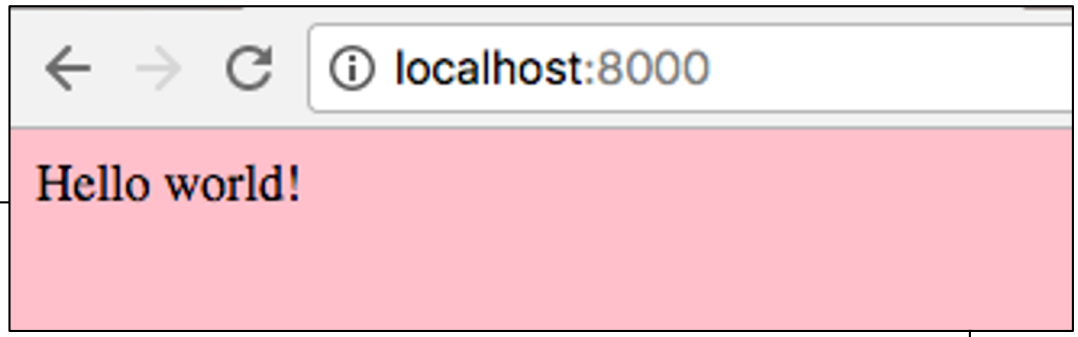
```
<!DOCTYPE html>
<html lang="en">
  <body>
    Hello world!
  </body>
</html>
```



CSS

- CSS describes the layout or style of a page.
- Link to CSS in HTML
- Example `style.css`

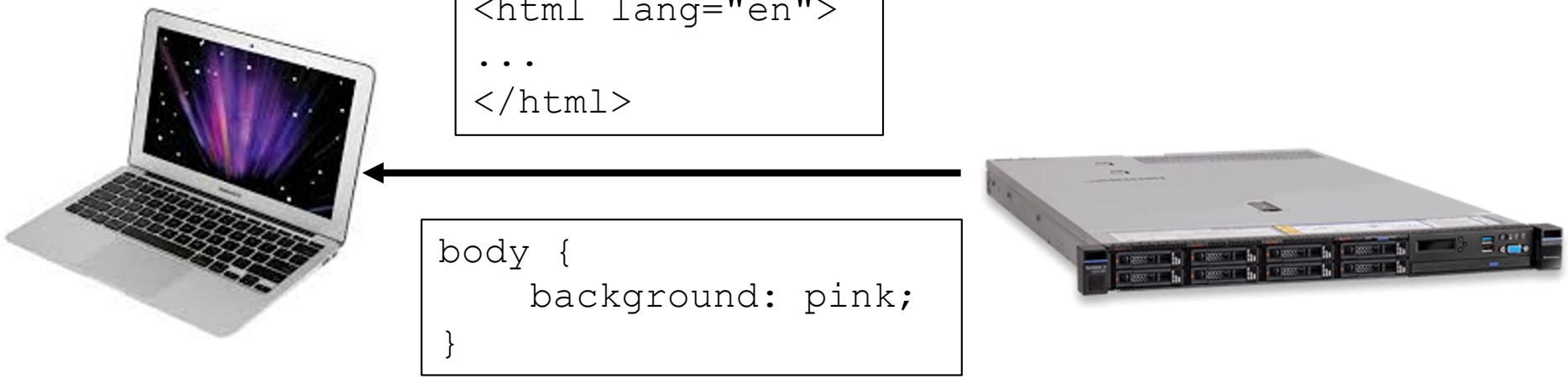
```
body {
    background: pink;
}
```




```
<!DOCTYPE html>
<html lang="en">
  <head>
    <link rel="stylesheet" type="text/css" href="/style.css">
  </head>
  <body>
    Hello world!
  </body>
</html>
```

Static pages

- A *static page* doesn't change
  - Often HTML, CSS, Images, etc.
  - No programming language on the server
  - Same content every time the page is loaded



Static file server

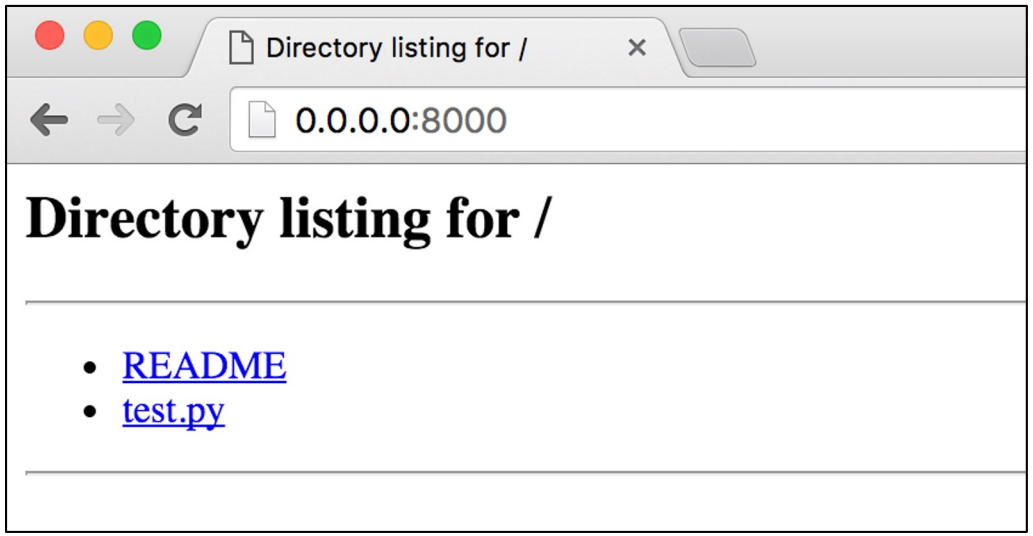
- A static file server sends files that don't change
  - HTML, CSS, images, etc.
- Server loads the file from disk and sends to client

- You'll use a static file server in project 1

Static file server in Python

```
$ python3 -m http.server
Serving HTTP on 0.0.0.0 port 8000 ...
```

- Now, navigate to http://0.0.0.0:8000
  - or http://localhost:8000



Static file server internals

- Server process waits for connection from client
- Receives a request
- Looks in content directory, computes file name `./index.html`
- Loads file from disk
- Writes response to client: 200 OK, followed by bytes for `./index.html`
- Pseudo code for Python's `http.server`

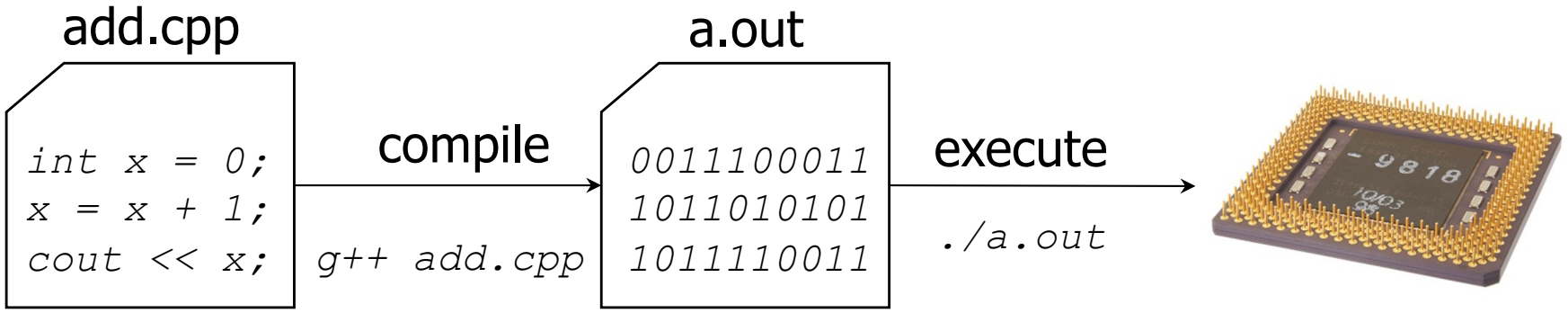
```
while not shutdown_request:
  if request:
    with open(request.filename) as fh:
      content = fh.read()
      copy(content, request.client)
```

Agenda

- About EECS 485
- Logistics
- The request response cycle
- **Python conceptual model**

Compiled language implementation

- Compiled language implementation: Program is converted into low-level machine code before execution
  - Examples: C, C++, Go



Interpreted language implementation

- Interpreted: program is executed by an interpreter, which is another program

```
$ python3 add.py
1
```

add.py

```
x = 0
x = x + 1
print(x)
```

- `python` is a program whose input is Python source code (plain text) and whose output is the output of the program described by the Python source code
- `python` is written in C
- An alternative interpreter, `pypy`, is written in Python!

Interactive interpreter

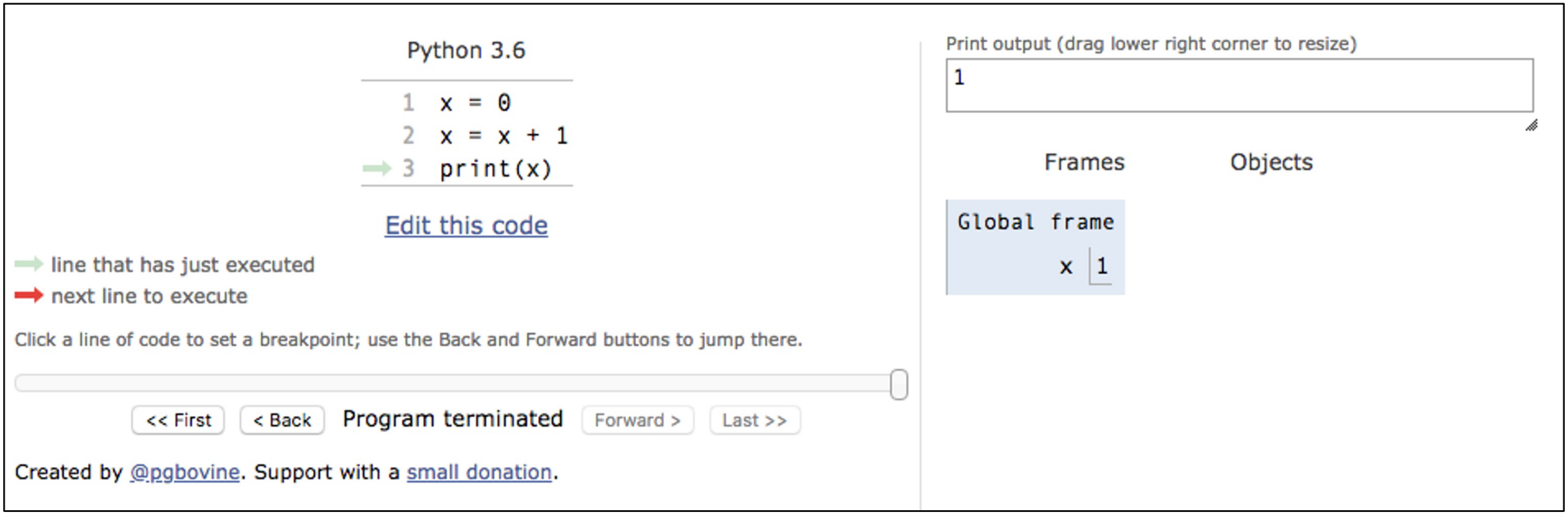
- You can use an interpreter interactively
- Great for debugging

```
$ python3
>>> x = 0
>>> x = x + 1
>>> x
1
```

- See the debugging tutorials on eecs485.org for more pro-tips

Visualizing Python programs

- **Pro-tip** visualize Python examples using PythonTutor
  - http://pythontutor.com/visualize.html



Data model

- *Objects* are Python's abstraction for data
- All data in a Python program are represented by objects
- Examples:
  - `x = 1`
  - `y = [1, 2, 3]`
  - `z = {"hello": 3, "world": 5}`
- All objects have
  - Identity (similar to memory address)
  - Type
  - Value
- Types include numbers, strings, lists, tuples, dictionaries, sets ...

C/C++ type system

- C/C++ is statically typed
- **Static typing:** Compile time variables have a type
  - Compiler checks for mistakes

```
int main() {
  int x = 0;
  x = "hello world";  // Compile error
}
```

- `x` refers to an integer object with value `0`
- `x` cannot be assigned to a different type

Python type system

- Python is dynamically typed
- **Dynamic typing:** Runtime objects (values) have a type
  - No compiler, no type checks at compile time

```
>>> x = 0
```

- `x` is a reference to an object
- `0` is an integer object

```
>>> x = "hello world"
```

- `x` is still a reference to an object. It now refers to a different object.
- `"hello world"` is a string object

C/C++ objects

Memory diagram

- Operations in C++ work with **values** of **objects** in memory
- Assignment means **copying** the **value**

```
int main() {
  int x = 12;
  int y = 34;
  cout << &x; //0x768
  cout << &y; //0x928
  x = y;
  cout << &x; //0x768
  cout << &y; //0x928
}
```

Python objects and references

Memory diagram

- Operations in Python work with **references** to **objects** in memory
- A Python reference is like a C/C++ pointer
- Assignment means **copying** the **pointer**

```
>>> x = [1, 2]
>>> y = [3, 4]
>>> id(x)
768
>>> id(y)
928
>>> x = y
>>> id(x)
928
```

C/C++ object allocation

Memory diagram

- Static (global) and automatic (local) variable allocation is managed by the compiler
  - Deallocated automatically
- Dynamic variable allocation is managed by the programmer
  - Deallocated manually

```
int SIZE = 3;
int main() {
  int x = 12;
  int *p = new int(34);
  delete p;
}
```

Python object allocation

Memory diagram

- Objects are allocated on assignment in a private heap
- Objects are deallocated automatically

```
>>> x = [1, 2]
>>> y = [3, 4]
>>> x = y
```

Reference counting and garbage collection

- How to automatically deallocate objects?
  - Identify objects that are no longer referenced and free them
  - Two concurrent methods in Python
- Reference counting
  - Keep track of the number of references to each object
  - If the number of references == 0, then deallocate
- Garbage collection
  - Periodically detect reference cycles
  - Deallocate "isolated" cycles of objects

Reference counting

- Keep track of the number of references to each object
- If the number of references == 0, then deallocate

```
>>> x = [1 ,2]   # ref count = 1
>>> y = x        # ref count = 2
>>> x = None     # ref count = 1
>>> y = None     # ref count = 0
```

- Immediately deallocate `[1,2]`

Garbage collection

- Periodically detect reference cycles
- Deallocate isolated cycles of objects

```
>>> x = {}       # x is a dictionary
>>> y = {}       # y is a dictionary
>>> x["y"] = y   # x references y
>>> y["x"] = x   # y references x
>>> x = None
>>> y = None
```

- Ref counts are both 1, even though neither `x` nor `y` is used by the program anymore
  - Garbage collector detects this case

Python functions

- Python functions start a new scope, just like C/C++
- Python blocks delimited by whitespace, unlike C/C++'s braces `{ }`

```
>>> def increment(x):
        return x + 1
>>> increment(5)
6
```

What is the output?

```
>>> def f(a):
        a.append(3)
>>> x = [1, 2]
>>> f(x)
>>> x
```

```
>>> def g(a):
        a = [1, 2, 3]
>>> x = [1, 2]
>>> g(x)
>>> x
```

What is the output?

```
>>> def f(a):
        a.append(3)
>>> x = [1, 2]
>>> f(x)
>>> x
[1, 2, 3]
```

```
>>> def g(a):
        a = [1, 2, 3]
>>> x = [1, 2]
>>> g(x)
>>> x
[1, 2]
```

- References are like pointers in C/C++ not C-style references

Python standard library

- Python ships with a huge standard library
  - https://docs.python.org/3/tutorial/stdlib.html

- Here are a few that will be useful for project 1
  - `pathlib`: common filename manipulations
  - `sys`: system-specific functions, e.g., `exit`
  - `shutil`: high-level file manipulations
  - `json`: JSON encoder and decoder

- Many web-related libraries
  - Python is great for web programming

Python third party libraries

- Python is extensible with 3rd party libraries hosted on the public PyPI repository

- Here are few that will be useful for project 1
  - `jinja2`: a template engine
  - `click`: command line utility option and argument parsing
- Use `pip` to install

```
$ pip install jinja2 click
```

- Many web-related libraries and frameworks
  - Python is great for web programming

Python tools

- `pdb`: the Python debugger
  - pdb+ (`pdbp`): really helpful extensions to `pdb`
- `pytest`: unit testing utility
- See tutorials on `pdb` and `pytest` on eecs485.org

Summary

- Please read syllabus and eecs485.org
- *Request response cycle* is how two computers communicate: client requests and server responds
- *Static page* is the same every time the page is loaded
- *Static file server* sends a copy of a file from disk
- Python programs are run by an *interpreter*
- Python is *dynamically typed*
- Python *references* work a lot like C/C++ pointers

Your to-do list

- Learn HTML
  - https://www.w3schools.com/html/html_intro.asp
- Learn (a little) CSS
  - https://www.w3schools.com/html/html_css.asp
- Learn Python
  - https://docs.python.org/3/tutorial/index.html
- Get started on Project 1
  - Link on eecs485.org