Welcome to Zoom University

• Let me know if there are audiovisual or networking issues

• You are muted on entry
  • You can use the chat feature to get my attention
  • Use “???” or just call out
  • (you can also unmute yourself)

• Hmu on Slack (many of you have already joined) if I seem unaware of issues
Announcements

• Lab starts next week

• Office hours - see calendar on eecs485.org

• Project 1 due next Friday at midnight

• Please read the syllabus
Course Information

- Instructor: Kevin Leach kjleach@umich.edu
- GSI: Emily Bao yuweibao@umich.edu
- IA: Andrew Wei awei@umich.edu

- Meetings: All remote through Zoom
  - Lecture: https://umich.zoom.us/j/99643331691 MTWR 1:30 – 3:00 PM Eastern
  - Discussion: https://umich.zoom.us/j/99062495076 MW 3:00 – 4:00 PM

- Login with https://umich.zoom.us
  - Recordings available, but please attend live if you are able
Course Resources

• Piazza is the main form of communication [https://piazza.com/class/kbg0bsipvcw4w](https://piazza.com/class/kbg0bsipvcw4w)
  • Email staff if you are not enrolled

• Slack is for other communication ([https://eecs485su2020.slack.com](https://eecs485su2020.slack.com))
  • Invite link on Piazza (pls no sharesies)
  • Find group members, chat with group about projects
  • Ask Kevin questions not answered on Piazza

• Autograder.io ([https://autograder.io](https://autograder.io))
  • Projects submitted through autograder

• GradeScope ([https://gradescope.com](https://gradescope.com))
  • For quizzes (due before the next lecture)
Agenda

• **What is EECS 485 about?**

• Logistics

• The request response cycle

• Python conceptual model
1. Front end – what the user sees

2. Network – information travel

3. Back end – runs on Google servers

4. Storing and processing 50,000 e-mails for 1 billion+ users

5. Searching 50,000 e-mails in under 1 second
What is EECS 485 about?

• "How the web works"

• "Full stack" = frontend and backend
  • Frontend: runs in web browser (Chrome, Safari, Firefox)
  • Backend: runs on servers

• The Web and this class draw from many sub-disciplines of computer science
  • Interesting overlap with many other upper level CS courses
    381, 388, 445, 475, 489, 482, 486, 490, 491
What is EECS 485 *not* about?

• We don’t talk about networks
  • MAC addresses, Ethernet, Autonomous networks
  • These are topics for 489

• We don’t do graphics design
  • You’ll learn how to make a frontend, but this is not a UI course

• We don’t do database construction
  • You’ll use databases on your backend, but not how to build the database itself
Projects - what you'll build

• Instagram
  • 3 different ways
  • Projects 1-3

• Hadoop distributed compute engine
  • How big data is processed on many computers in parallel
  • Project 4

• Search algorithm of Google, circa mid 2000's
  • Project 5
Lecture/Lab – what you'll learn

• How the different parts of a website communicate with each other
  • Many different programs

• How to make many small computers coordinate to do one large task
  • Process and store large data sets
  • Service many users

• How search engines work

• How online advertising is sold

• And more!
Some technologies you'll use

• Linux
  • System administration for deployment
• Shell scripting
• HTML/CSS
• Python
  • Flask web framework with jinja2 template engine
  • Thread, process and socket libraries
• SQL
  • SQLite database
• JavaScript
  • Modern ES6 syntax
  • React/JS framework
• Hadoop
Agenda

• What is EECS 485 about?

• Logistics

• The request response cycle

• Python conceptual model
Waitlist

- We know there's a waitlist
  - Sorry!

- Wolverine Access handles the waitlist automatically

- When one enrolled student drops, Wolverine Access issues permission via email to one waitlisted student
  - Follow the instructions in the email

- Unfortunately, it's hard to predict how many students will drop
when the professor who banned phones and laptops in class now has to learn how to teach online 😏

From a principal’s publication, 1815: “Students today depend on paper too much. They don’t know how to write on a slate without getting chalk dust all over themselves. They can’t clean a slate properly. What will they do when they run out of paper?”
Attendance

- I would prefer you attend the lectures live as they are delivered
  - Pay more attention synchronously
  - Opportunity to ask questions before you forget

- Lecture and Discussion recordings available
  - However, we will have random quizzes on some lectures
    - (you’ll have 24 hours to turn these in remotely)

A meta-analysis of the relationship between class attendance in college and college grades reveals that attendance has strong relationships with both class grades (k = 69, N = 21,195, ρ = .44) and GPA (k = 33, N = 9,243, ρ = .44). These relationships make class attendance a better predictor of college grades than any other known predictor of academic performance, including scores on standardized admissions tests such as the SAT, high school GPA, study habits, and study skills. Results also show that class attendance explains large amounts of unique variance in college grades because of its relative independence from SAT scores and high school GPA and weak relationship with student characteristics such as conscientiousness and moti-

Grades

• To pass EECS 485, your average project score must be a passing score, and your average exam score (not including quizzes) must be a passing score (default: 70% or above)
  • Professionalism must also be > 0%

<table>
<thead>
<tr>
<th>Component</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>Projects (5 x 10%)</td>
<td>50%</td>
</tr>
<tr>
<td>Exam 1</td>
<td>20%</td>
</tr>
<tr>
<td>Exam 2</td>
<td>20%</td>
</tr>
<tr>
<td>GradeScope Quizzes</td>
<td>5%</td>
</tr>
<tr>
<td>Professionalism</td>
<td>5%</td>
</tr>
</tbody>
</table>
GradeScope Quiz

• There is a quiz on GradesCope to complete after random lectures
  • They’re due before the next lecture
  • No late quizzes accepted
  • Today’s codeword: Independence Day (won’t count for credit, just for testing GradeScope)

• Format:
  • Code word
  • Short answer / multiple choice
  • Summary
  • Activity

• Why?
  • To help you stay on top of lecture material in a remote setting

• 3-5 lowest quizzes are dropped (i.e., we’ll probably keep the top 10)
Autograder

• Linked from eecs485.org
• 6 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
  • More than half the points

• Private testcases
  • Visible after the deadline
  • Not published
Style grading

• Automatic style grading
  • Common in industry. We will use the same tools.
• HTML
  • html5validator
• Python
  • pycodestyle
  • pydocstyle
  • pylint
• JavaScript
  • eslint with AirBnB coding standard
• **Pro-tip:** run the tools early and often!
Collaboration

• We will solicit peer evaluations on group projects
  • Members who contribute less than their share may receive a lower grade on the project
  • Non-contributing members may receive a zero
• For those retaking the course: if you submitted a project in a previous term, you may not be in a group for that same project this term.

<table>
<thead>
<tr>
<th>Task</th>
<th>Collaboration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project 1</td>
<td>No</td>
</tr>
<tr>
<td>Projects 2-5</td>
<td>Groups of 2 - 3</td>
</tr>
<tr>
<td>Exams</td>
<td>No</td>
</tr>
<tr>
<td>Quizzes</td>
<td>Yes, but written answers in own words</td>
</tr>
</tbody>
</table>
Discussion / Lab

• Starts next Monday
  • Emily will run these

• Hands-on coding and project help

• *Not* intended for asking individual coding questions
  • e.g., don’t try to share your own project submissions
Expectations in a ULCS: Independence

• In many ways, 485 is a grad class
  • Group work
  • Big projects
  • Increased independence

• You'll have a lot to learn on your own
  • HTML
  • CSS
  • Python (we’ll see some in lecture)
  • Flask framework
  • JavaScript (we’ll see some in lecture)
  • React JS framework
  • Independent learning is the biggest difference between 485 and other classes
  • Consider this fair warning!

• Expect breadth: the web encompasses many pieces of other areas of computer science
Getting help

• Piazza
  • For questions not including code

• Staff office hours
  • Details on eecs485.org
  • Can help with code

• Professor office hours
  • Details on eecs485.org

• Coursewide Slack
  • DM Kevin, talk with others
How to do well in 485

• The exams won't be a surprise if you keep up with quizzes and projects

• Learn to work well with your partner

• P1-P3 are similar, P4-P5 will require more effort and time
Agenda

• What is EECS 485 about?

• Logistics

• The request response cycle

• Python conceptual model
The request response cycle

- The **request response cycle** is how two computers (or programs) communicate with each other over the Internet.

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

(more details in 489)
The request response cycle

• A client requests a web page

• A server responds with (usually) an HTML file
  • It might create the content on-the-fly

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

• The client renders the HTML
What does a server respond with?

• A server might respond with different kinds of files. Common examples:
  • HTML
  • CSS
  • JavaScript – we’ll talk about this one later
  • Text
  • JSON
  • XML

**Basically:** the server produces a big string that gets *interpreted* by the client
HTML: Hyper Text Markup Language

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

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

https://www.w3schools.com/TAGS/default.ASP
CSS

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

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

```html
<!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 is only HTML/CSS
  • No programming language on the server
  • Same content every time the page is loaded

```html
<!DOCTYPE html>
<html lang="en">
  ...
</html>

body {
  background: pink;
}
```
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

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

• What is EECS 485 about?

• Logistics

• The request response cycle

• Python conceptual model
## Python vs. C++

<table>
<thead>
<tr>
<th>C++</th>
<th>Python</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Compiled</strong></td>
<td><strong>Interpreted</strong></td>
</tr>
<tr>
<td>Static typing</td>
<td>Dynamic typing</td>
</tr>
<tr>
<td>Default pass by value</td>
<td>Default pass by pointer</td>
</tr>
<tr>
<td>Manual memory management</td>
<td>Automatic memory management with garbage collection</td>
</tr>
</tbody>
</table>
Compiled vs. interpreted

• Language implementations can be compiled or interpreted
• Compiled: Program is converted into low-level machine code before execution. Examples include C/C++
Compiled vs interpreted

• Language implementations can be compiled or interpreted
• Interpreted: program is executed by an interpreter, which is another program

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

```
x = 0
x = x + 1
print(x)
add.py
```

• `python` is a program whose input is Python source code (plain text) and whose output is that 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](http://pythontutor.com/visualize.html)
## Python vs. C++

<table>
<thead>
<tr>
<th></th>
<th>C++</th>
<th>Python</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Type</strong></td>
<td>Compiled</td>
<td>Interpreted</td>
</tr>
<tr>
<td><strong>Typing</strong></td>
<td>Static typing</td>
<td>Dynamic typing</td>
</tr>
<tr>
<td><strong>Pass by</strong></td>
<td>Default pass by value</td>
<td>Default pass by pointer</td>
</tr>
<tr>
<td><strong>Memory</strong></td>
<td>Manual memory management</td>
<td>Automatic memory management with garbage collection</td>
</tr>
</tbody>
</table>
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 strongly, statically typed
• **Strong** typing means that the type of a value doesn't suddenly change
• **Static** typing means that compile time variables have a type

```c
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 strongly, dynamically typed
• Strong typing means that the type of a value doesn't suddenly change.
• Dynamic typing means that runtime objects (values) have a type

>>> 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
## Python vs. C++

<table>
<thead>
<tr>
<th></th>
<th>C++</th>
<th>Python</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Compiled</td>
<td>Interpreted</td>
</tr>
<tr>
<td></td>
<td>Static typing</td>
<td>Dynamic typing</td>
</tr>
<tr>
<td>Default pass by value</td>
<td>Default pass by pointer</td>
<td></td>
</tr>
<tr>
<td>Manual memory management</td>
<td>Automatic memory management with garbage collection</td>
<td></td>
</tr>
</tbody>
</table>
C/C++ objects

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

```cpp
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

About Python objects and references:

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

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

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

```python
>>> def increment(x):
    return x + 1
>>> increment(5)
6
```
What is the output?

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

```python
>>> def g(a):
    a = [1, 2, 3]
>>> x = [1, 2]
>>> g(x)
>>> x
```
What is the output?

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

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

```python
>>> def g(a):
    a = [1, 2, 3]
>>> x = [1, 2]
>>> g(x)
>>> x
[1, 2]
```
# Python vs. C++

<table>
<thead>
<tr>
<th>C++</th>
<th>Python</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compiled</td>
<td>Interpreted</td>
</tr>
<tr>
<td>Static typing</td>
<td>Dynamic typing</td>
</tr>
<tr>
<td>Default pass by value</td>
<td>Default pass by pointer</td>
</tr>
<tr>
<td>Manual memory</td>
<td>Automatic memory management with garbage</td>
</tr>
<tr>
<td>management</td>
<td>collection</td>
</tr>
</tbody>
</table>
Python object allocation

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

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

• Keep track of the number of references to each object
• If the number of references == 0, then deallocating

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

• Can deallocate `[1, 2]`
Garbage Collection

• Work with partner: why won't reference counting work here?

```python
x = [2]
y = [3, x]
x.append(y)

x = 0
y = 0
```
Standard library

• Python ships with a huge standard library
  • [https://docs.python.org/3/tutorial/stdlib.html](https://docs.python.org/3/tutorial/stdlib.html)

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

• There are many web-related modules, which make Python great for web programming
3rd party libraries

- Python is extensible with 3rd party libraries hosted on the public [PyPI](https://pypi.org) 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
  
  ```bash
  $ pip install jinja2 click
  ```

- Many more web-related libraries and frameworks, again makes Python great for web programming
Python 2 vs. Python 3

• Lots of differences between Python 2 vs. Python 3
  • Python 2 is officially obsolete!

• Multiple versions may be installed
  $ python2 --version
  Python 2.7.13
  $ python3 --version
  Python 3.6.2
  $ python --version
  Python 2.7.13

• We'll use Python 3 in EECS 485
Tools

• See tutorials on pdb and pytest on eecs485.org

• pdb: the Python debugger
  • pdb++ (pdbpp): really helpful extensions to pdb

• pytest: unit testing utility
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