CSE 115
Introduction to Computer Science I
Midterm

Midterm will be returned no later than Monday.

We may make midterm pickup available before then.

Stay tuned.
# Midterm results

<table>
<thead>
<tr>
<th></th>
<th>Module 1</th>
<th>Module 2</th>
<th>Overall</th>
</tr>
</thead>
<tbody>
<tr>
<td>avg</td>
<td>68.4 / 80</td>
<td>60.3 / 80</td>
<td>128.7 / 160</td>
</tr>
<tr>
<td>avg %</td>
<td>85.5%</td>
<td>75.4%</td>
<td>80.5%</td>
</tr>
<tr>
<td>median</td>
<td>72 / 80</td>
<td>66 / 80</td>
<td>137 / 160</td>
</tr>
<tr>
<td>median %</td>
<td>90.0%</td>
<td>82.5%</td>
<td>85.6%</td>
</tr>
</tbody>
</table>
Road map

▶ Review ◀

JSON

Chat App - Part 1

AJAX

Chat App - Part 2
var myDiv = document.getElementById("myDiv");
myDiv.innerHTML = "Content added from JavaScript";
Web Server

Client
Sends requests to server

Client
Sends requests to server

Client
Sends requests to server

Web Server
Software runs continuously and waits for requests from clients

Responds to requests
import bottle

@bottle.route("/")
def any_name():
    return bottle.static_file("index.html", root="")

@bottle.route("/myCode.js")
def another_name():
    return bottle.static_file("myCode.js", root="")

bottle.run(host="0.0.0.0", port=8080, debug=True)
Client
Sends requests to server at "/"

```html
<html>
<head></head>
<body>
<h1>First Web Page</h1>
<p>My content</p>
<div id="myDiv"></div>
<script src="myCode.js"></script>
</body>
</html>
```

Server responds with index.html

```python
import bottle

@index
def any_name():
    return bottle.static_file("index.html", root='')

@index
def another_name():
    return bottle.static_file("myCode.js", root='')

bottle.run(host="0.0.0.0", port=8080, debug=True)
```

```
var myDiv = document.getElementById("myDiv");
myDiv.innerHTML = "Content added from JavaScript"
```

myCode.js runs in the browser and the HTML is modified

Client sends requests to server at "/"

Server responds with index.html

index.html requires myCode.js and a second request is sent

Server responds with myCode.js
Road map

Review

▶ JSON ◀

Chat App - Part 1

AJAX

Chat App - Part 2
We've seen `json.loads` to convert from a JSON string to Python type.

To complete the conversions we have:
- `json.dumps` to convert Python types to JSON strings.
- `JSON.stringify` to convert JavaScript types to a JSON string.
- `JSON.parse` to convert a JSON string to JavaScript type.

Whenever we send data over the Internet we'll convert it to a JSON string.
Road map

Review

JSON

▶ Chat App - Part 1 ◀

AJAX

Chat App - Part 2
Chat App - Overview

Goal: Build an app where users can send chat messages to all other users

To build this app we will need
- An HTML file that will be displayed to the user
- (JavaScript) A function that will modify the HTML to display the chat history to the user
- (JavaScript) A function that will allow the user to submit a new chat message
- (Python) A server that will host our HTML, JavaScript, and chat messages
- (Python) Function to save and load the chat history in a file that will persist ever if the server restarts
- A way for the JavaScript front-end to communicate with the Python back-end
Chat App - Overview

Note: There are many possible ways to build this app

We'll walk through only one possible design, though there are many other solutions
Chat App - File Structure

We'll create the following files for our chat app

- index.html
- chat.js
- server.py
- chat.txt
- chat.py
Chat App - index.html

<html>
<head>
    <script src="chat.js"></script>
</head>
<body onload="loadChat();">

Message: <input type="text" id="message">
<button onClick="sendMessage();">Send</button>
<hr/>
<div id="chat"></div><br/>

</body>
</html>
Chat App - index.html

Download the JavaScript portion of the app
Chat App - index.html

```html
<html>
<head>
    <script src="chat.js"></script>
</head>
<body onload="loadChat();">
    Message: <input type="text" id="message">
    <button onClick="sendMessage();">Send</button>
    <hr/>
    <div id="chat"></div><br/>
</body>
</html>

Instead of adding the script tag at the bottom of the body so it runs after the page loads we'll download it earlier and run it using the onload property to call a JavaScript function
Chat App - index.html

Add an empty div where our JavaScript can write the chat history
Add 2 new HTML elements

- An input with a type of text gives the user a text box
- A button that calls one of our JavaScript functions whenever the user clicks it
Chat App - chat.js

function loadChat(){
    ...
}

function sendMessage(){
    ...
}

We'll define these later, but chat.js will need these two functions:

loadChat()
• Gets the chat history from the server and displays in the chat div

sendMessage()
• Reads the value in the text box and sends it to the server as a new chat message
• Updates the chat history
Chat App - chat.txt

This file will store all of the chat history

We won't add anything to this file manually, but we'll manually create it as an empty file

This file will store one chat message per line
Chat App - chat.py

```python
filename = "chat.txt"

def get_chat():
    full_chat = []
    with open(filename) as file:
        for line in file:
            full_chat.append({"message": line.rstrip("\n\r")})
    return full_chat

def add_message(message):
    with open(filename, "a") as file:
        file.write(message + "\n")
```
Chat App - chat.py

filename = "chat.txt"

define a filename variable outside of any function (at the module level)

This variable can be used by any functions in this file

This allows us to switch files by making only one change in our code
def add_message(message):
    with open(filename, "a") as file:
        file.write(message + "\n")

We'll call this function each time we get a new message from a user

Open chat.txt in append mode and write the message followed by a new line
def get_chat():
    full_chat = []
    with open(filename) as file:
        for line in file:
            full_chat.append({"message": line.rstrip("\n\r")})
    return full_chat

Read all the chat messages from a file and add them to a list as a dictionary with a key "message"

Our JavaScript code will expect this same format
import bottle
import json
import chat

@bottle.route('/
')
def index():
    return bottle.static_file("index.html", root="")

@bottle.route('/chat.js')
def static():
    return bottle.static_file("chat.js", root="")

@bottle.route('/chat')
def get_chat():
    return json.dumps(chat.get_chat())

@bottle.post('/send')
def do_chat():
    content = bottle.request.body.read().decode()
    content = json.loads(content)
    chat.add_message(content['message'])
    return json.dumps(chat.get_chat())

bottle.run(host="0.0.0.0", port=8080, debug=True)
Chat App - Server

```python
import bottle
import json
import chat
```

We'll need bottle and json so we import those package.

Since we have python code across 2 files we also need to import our chat file:

- Since there files are in the same directory we can import using it's name, without the ".py"
- We can now call `chat.add_message(message)` and `chat.get_chat()` just like we call function from built-in modules.
@bottle.route('/')
def index():
    return bottle.static_file("index.html", root='')

@bottle.route('/chat.js')
def static():
    return bottle.static_file("chat.js", root='')

Host our 2 front end files

The user will access our root path "/" to get index.html

In our HTML we have a script tag referencing chat.js which will trigger a second HTTP requests with a path of "/chat.js"
... 
@bottle.route('/chat')
def get_chat():
    return json.dumps(chat.get_chat())

@bottle.post('/send')
def do_chat():
    content = bottle.request.body.read().decode()
    content = json.loads(content)
    chat.add_message(content['message'])
    return json.dumps(chat.get_chat())

...

Set up paths that will be accessed from our JavaScript code

We need to convert to/from JSON strings whenever data is being sent/received from the front-end
@bottle.post('/send')
def do_chat():
    content = bottle.request.body.read().decode()
    content = json.loads(content)
    chat.add_message(content['message'])
    return json.dumps(chat.get_chat())

We label this as a post path to limit this to HTTP POST requests

POST is a way to say that this path expects the user to send information to the server

This information will be stored in the body of the request
Chat App - Server

```python
@bottle.post('/send')
def do_chat():
    content = bottle.request.body.read().decode()
    content = json.loads(content)
    chat.add_message(content['message'])
    return json.dumps(chat.get_chat())
```

To read the data in the HTTP request containing a new chat message, we will read the variable "bottle.request.body" which contains the body of the request in a similar format as the response of an HTTP request when we connected to APIs.

This path expects the body of the request to be a JSON string in the format `{"message": <message_content>}`

It is important that both our Python and JavaScript code use the same format.
Chat App - Server

...  
bottle.run(host="0.0.0.0", port=8080, debug=True)

And finally, start the server
Now we have our server setup that will
• Serve our HTML and JavaScript files
• Accept new chat message and save them to a file at the path "/send"
• Return the entire chat history at the path "/chat"
• The chat history will persist even if the server restarts

What we need next is to write our JavaScript functions that will make requests to the paths
Chat App - Check Point

However, the only way we have to make HTTP requests from the front end is to
• Type a URL in our browser
• Add script tags in the HTML that download JavaScript files

How do we make requests to "/send" and "/chat" after the page loads?

How do we make a POST requests with a body containing a chat message?
Road map

Review

JSON

Chat App - Part 1

▶︎ AJAX ◀

Chat App - Part 2
AJAX

Asynchronous JavaScript
• Because everything related to web development needs its own acronym..

A way to make HTTP request from JavaScript after the page is fully loaded

Can make HTTP GET requests (Request content from a server)

Can make HTTP POST requests (Send content to a server)
There are many different ways to setup an AJAX call

We setup the call in a function that takes

- The path where we want to send the requests (matches the paths in the annotations of the bottle server)
- A callback function. This function will be called when the server responds to our requests with the response as an argument

```javascript
function ajaxGetRequest(path, callback){
    var request = new XMLHttpRequest();
    request.onreadystatechange = function(){
        if (this.readyState === 4 && this.status === 200){
            callback(this.response);
        }
    }
    request.open("GET", path);
    request.send();
}
```
AJAX - HTTP GET Request

```javascript
function ajaxGetRequest(path, callback){
    var request = new XMLHttpRequest();
    request.onreadystatechange = function(){
        if (this.readyState === 4 && this.status === 200){
            callback(this.response);
        }
    }
    request.open("GET", path);
    request.send();
}
```

To avoid being distracted by the details, you may paste this function in your JavaScript where it's needed and call this function whenever you need to make an AJAX request.
function ajaxPostRequest(path, data, callback){
    var request = new XMLHttpRequest();
    request.onreadystatechange = function(){
        if (this.readyState === 4 && this.status === 200){
            callback(this.response);
        }
    }
    request.open("POST", path);
    request.send(data);
}

To make a POST request most of the code is the same

The major difference is that we have a third parameter named data which must be a string containing the data that will be in the body of the request
AJAX - HTTP POST Request

```javascript
function ajaxPostRequest(path, data, callback){
  var request = new XMLHttpRequest();
  request.onreadystatechange = function(){
    if (this.readyState === 4 && this.status === 200){
      callback(this.response);
    }
  };
  request.open("POST", path);
  request.send(data);
}
```

As with the AJAX GET request you may paste this function where needed so we don't get distracted by this syntax and these details.
To make an AJAX POST request we need to call the `ajaxPostRequest` function with a path, data, and a callback function.

When the function `called_on_button_press` is called it will send an AJAX POST request:
1. To the path "/some_path" which must match a path of the same name in the bottle server.
2. With a body of "Button pressed" (We will send JSON strings in our apps).
3. When the server responds to this request our `action_on_response` function will be called with the response from the server.
function action_on_response(response){
    console.log("The server responded with: ", response);
}

function called_on_button_press(){
    ajaxPostRequest("/some_path", "Button pressed", action_on_response);
}

Note that we do not use parentheses when passing the
action_on_response function as an argument

We are passing the entire function as an argument. Not the
evaluation of a call of this function

The function will be called latter by the AJAX function

[Calling ajaxGetRequest works the same way except we don't pass a
data argument]
Road map

Review

JSON

Chat App - Part 1

AJAX

Chat App - Part 2
Chat App Continued

Now that we have a way to communicate with our server after the page is loaded we can finish our chat app.

To do this we will make an AJAX get request after the page loads to get and display the current chat history.

Then we will make an AJAX POST request each time the user clicks the button to send a message.
function renderChat(response){
    var chat = "";
    for(var data of JSON.parse(response).reverse()){
        chat = chat + data.message + "<br>";
    }
    document.getElementById("chat").innerHTML = chat;
}

function loadChat(){
    ajaxGetRequest("/chat", renderChat);
}

Recall
• loadChat is called after the HTML is finished loading
• The response from the server at "/chat" is a JSON string representing a list of objects where each object contains a key "message"
Chat App - chat.js

function renderChat(response){
    var chat = "";
    for(var data of JSON.parse(response).reverse()){
        chat = chat + data.message + "</br>";
    }
    document.getElementById("chat").innerHTML = chat;
}

function loadChat(){
    ajaxGetRequest("/chat", renderChat);
}
...

loadChat initiates the AJAX GET request at the path "/chat" to get the current chat history from the server

The callback function is renderChat which parses the JSON string and iterates over the array while accumulating a string storing HTML

The callback then sets this HTML to the div with the id "chat"
function sendMessage(){
    var messageElement = document.getElementById("message");

    var message = messageElement.value;
    messageElement.value = "";
    var toSend = JSON.stringify({"message": message});

    ajaxPostRequest("/send", toSend, renderChat);
}

Recall
• Send message is called when the user clicks the send button
• There is a text box with the id "message" on the page
• The "/send" path on our server expects a JSON string representing an object with a key of "message"
Chat App - chat.js

```javascript
function sendMessage()
    var messageElement = document.getElementById("message");

    var message = messageElement.value;
    messageElement.value = ";
    var toSend = JSON.stringify({"message": message});

    ajaxPostRequest("/send", toSend, renderChat);
}
```

When the sendMessage function is called (the button is clicked) it will initiate an AJAX POST request to the "/send" path

The data to be sent is pulled from the text box input by accessing its "value" property. This property contains the text that the user has entered

The server responds with the updated chat history so our callback the same renderChat function as we used for the GET request
Chat App

We now have a fully functional chat app that uses JavaScript and AJAX calls to communicate with a python web server that saves the chat history in a persistent file.
Chat App - Expansions

Our app works just fine, but it could benefit from improvements. Here are few ideas that could expand this app

Make it pretty
• The app has no style. We could use CSS, Bootstrap, etc to improve the aesthetics of the app

Live updates
• Users only see new messages when they either send a message or refresh the page
• We could improve this by using polling, long-polling, or web sockets

Keyboard shortcuts
• Allow users to send a message by hitting the enter key
Chat App - Polling

Instead of loading the content once, call setInterval
• Calls loadChat every 2000 ms (2 seconds)

User will see live chat with at most a 2 seconds delay

Not the best way to get updates from the server, but it is the simplest
• Creates a lot of traffic
• tradeoff between delays and server traffic
• ex: 100 users @ 2 second polling = 50 requests per second
Use the onKeyPress attribute to call a new JavaScript function whenever a key is pressed

Use checkEnter to check if the enter key is pressed
- Every key has a key code
- the key code for enter is 13
Now that we have the foundation of app we can build upon with significantly less effort than it took to create the app

What can you build?