CSE 115
Introduction to Computer Science I
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

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

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

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

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

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

Server responds with myCode.js

myCode.js runs in the browser and the HTML is modified
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)
function ajaxGetRequest(path, callback) {
    var request = new XMLHttpRequest();
    request.onreadystatechange = function () {
        if (request.readyState === 4 && request.status === 200) {
            callback(request.response);
        }
    }
    request.open("GET", path);
    request.send();
}

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
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
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.
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);
}

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
AJAX - Calling the Functions

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.
Chat App - chat.js

function renderChat(response){
    var chat = "";
    for(var data of JSON.parse(response).reverse()){
        chat = chat + data.message + "\n";
    }
    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

<body onload="setInterval(loadChat, 2000);">...
Chat App - Enter to Send

Message: <input type="text" id="message" onKeyPress="checkEnter(event);"/>

```javascript
function checkEnter(keyUpEvent){
    if(keyUpEvent.keyCode === 13){
        sendMessage();
    }
}
```

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?
Chat App

User

Browser
Navigates to the app's URL

index.html downloaded

The Internet

HTTP Request for path "/"

@bottle.route("/")
# return static file:
# index.html

HTTP Request for path "/chat.js"

@bottle.route("/chat.js")
# return static file:
# chat.js

Server

AJAX HTTP GET Request for path "/chat"

@bottle.route('/chat')
# call get_chat() in the
# chat.py file

JSON formatted chat history
-Convert to HTML and set as the innerHTML of the chat div
Chat App

User

User enters a chat message and clicks button

The Internet

AJAX HTTP POST Request for path "/send"

Server

@bottle.route('/send')
# - read the sent message
# - call add_message from the
# chat.py file
# - the message is appended to
# chat.txt

JSON formatted chat history
- Converted to HTML and set as the innerHTML of the chat div

Handle POST requests until the user leaves the site
Music Rating App

User

Browser
Navigates to the app's URL

index.html downloaded

The Internet

HTTP Request for path "/"

@bottle.route("/")
# return static file:
# index.html

HTTP Request for path "/myCode"

@bottle.route("/myCode.js")
# return static file:
# myCode.js

AJAX HTTP GET Request for path "/songs"

@bottle.route('/songs')
# call get_songs() in the
# ratings.py file

Server

myCode.js downloaded

JSON formatted songs and ratings
-Convert to HTML and set as the innerHTML of the songs div

JSON formatted songs and ratings and set as the innerHTML of the songs div
Chat App

User

User enters a new song and clicks button

The Internet

AJAX HTTP POST Request for path "/add_song"

@bottle.route('/add_song')
# -read the new song
# -call add_song from the 
#  ratings.py file
# -the song is appended to 
#  songs.csv

Server

@bottle.route('/rate_song')
# -call rate_song from the 
#  ratings.py file
# -the rating is appended to 
#  ratings.csv

Handle POST requests until the user leaves the site

User clicks a rating button

JSON formatted songs and ratings
-Convert to HTML and set as the innerHTML of the songs div

JSON formatted songs and ratings
-Convert to HTML and set as the innerHTML of the songs div
Music Rating App - Expansions

Prevent Multiple Ratings
• Users can rate the same songs as many times as they can click (or write a program to spam ratings)
• Discussion: How would we prevent this?

Update Titles and Artists
• If the user to add a song uses the wrong title/artist, it cannot be updated later
• Could make it so any user can edit these fields

Reviews
• Add reviews to the ratings so users can share their opinions instead of just numbers
Music Rating App - Expansions

Style
• Add star ratings instead of displaying all ratings for each song
• Add color and CSS

Sorting (module 4 foreshadow)
• Sort the songs based on average rating, artist, or number of ratings

Security (module 4 foreshadow)
• This site is not secure!
• Vulnerable to HTML/JavaScript injection
• No encryption of HTTP requests
• Preventing multiple ratings without compromising privacy