Announcements

- The final is at 10:30 am, Sat 6/4, in this room
  - Open book, open notes
  - No electronic devices
  - No food

- Assignment 7 due 10pm tomorrow
- No late Assignment 7’s
- Fill out course evaluations
  - https://eval.ucdavis.edu
- Office hours Friday 12-2, 2131 Kemper Hall

Asynchronous programming

- Pattern we see in different forms:
  - Make request for data
  - Specify callback function
- Callback is run when data is received
- Used for any request that might take a lot of time
- Why? Neither the server nor the browser should get hung up waiting for data
  - Browser should respond to user commands, run animations, etc
  - Server should respond to URLs

Example: Database access

```
db = new sqlite3.Database("polls.db");
var request =
  "SELECT * FROM Polls WHERE population = males";
db.get(request, function(err, row) {
  if (!error & row != undefined) {
    console.log(row);
  }
});
```

What is `db`? What data is this trying to get? Where is the callback function? When does it get run? Would this code run on the server or the browser?

Bad example: API request

```
requestString = "http://elections.huffingtonpost.com/pollster/api/polls?jsonp=1&state=CA&after=2016-04-20";
request (requestString, function(error, resp, body) { });
console.log(body);
```

Never prints out a result. Why not? How to fix?

Problem: API request

```
requestString = "http://elections.huffingtonpost.com/pollster/api/polls?jsonp=1&state=CA&after=2016-04-20";
request (requestString, function(error, resp, body) {
  console.log(body);
});
```

Action to be taken has to be inside the callback function
Server control flow

- Browser HTTP requests to API
- API HTTP responses
- Database outputs
- Queue of incoming work
- Run appropriate event handler
- Database
- SQL queries
- HTTP requests to browser

Browser control flow

- Server HTTP responses
- API JSONp responses
- Buttons and other UI
- Animation timer
- Queue of incoming work
- Run appropriate event handler
- HTTP requests to server

The server

```javascript
var staticFileServer = new static.Server('./public');
function handler (request, response) {
  var pathname = url.parse(request.url).pathname;
  if (pathname == '/query')
    queries.queryServer(request, response, search);
  else staticFileServer.serve(request, response);
}
```

- The event handler when the server gets an HTTP request
- Static page handler told to work in directory “public”
- Request is object containing incoming request

Examples

- Browser -> API
  - JSONp used in Weather App
- Browser -> Server
  - XMLHttpRequest
- Server -> API
  - Node request function
- Server -> Database
  - Node db object

Other events

- These are not callback functions for data requests, but they are asynchronous event handlers
- User Interface events
  - onclick
  - onchange
  - mouseover, mouseout
- Animation
  - Uses setInterval function and animation object
  - Calls an animation function every 30ms, eg.

The response object

- The response object is like a pre-addressed envelope, addressed to the machine that made the request
- You are the server, Netflix is the browser, in this metaphor.
The response object

- The static server puts the requested static file into the response object, and sends it off.
- The dynamic query server computes a response, usually in JSON but also potentially in HTML, CSS, XML..., puts that into the response object, and sends it off.
- Response object passed to function that will fill it in.
- Often this requires doing an API or database request, so it won’t be done immediately but in a callback.

Using closure to pass response object

function doAPIQuery(response) {
  requestString = "http://apiSite.com?q=who";
  request(requestString, function (error, resp, body) {
    if (!error &.& resp.statusCode == 200)
      fillResponse(body, response);
  });
}

Here the anonymous callback is defined inside doAPIQuery(), which means it has access to all the variables of doAPIQuery, even though it is called much later.

So it can pass the response object on to fillResponse()

XMLHttpRequest

var req = new XMLHttpRequest();
req.onreadystatechange = function() {  
  if (req.readyState === XMLHttpRequest.DONE) {
    if (req.status === 200)
      console.log(req.response);
  }
  var reqURL = "/query?" + req.responseText;
  req.open('GET', reqURL, true);
  req.send(null);
}

Slightly different from previous two examples

Why only part of the URL?

AJAX requests don’t let you give the beginning of the URL because they can only go to the URL containing the current Web page.

For instance, an AJAX request from http://45.55.29.158:8082/CAmap.html can only go to a URL beginning with http://45.55.29.158:8082

This is because of the Same Origin Policy

Same Origin Policy

D3 idea

- Attach data to SVG or DOM objects.
- Use it to set attributes or style attributes such as size, position and color.
- Special syntax to create an array of objects to go along with an array of data.

// make empty list of divs in container
var divList = d3.select("#blocks").selectAll("div");
var divs = divList.data(districts); // bind empty list to array
divs.enter().append("div"); // add one div per array element
D3 function arguments

```javascript
divs.attr("backgroundColor", "rgb(200,200,150)");
divs.style("height", function(d) { return (d*5)+"px";
});
divs.attr("class", function(d,i) { return "delegateBox dist"+(i+1)");
```

- We are setting the color, height and class
- The bound data shows up as the first argument `d` of the anonymous function argument
- Index `i` in the original list is the second argument

Examples of D3 arrays

- Delegate boxes
- Map regions
- Statewide delegate boxes
- Covering bars showing Hillary votes

Examples of D3 arrays

```javascript
start = [255,150,180]; end = [20,20,255];
frac = .35; color = [0,0,0];
for (var i=0; i<3; i++) {
    color[i] = Math.round((end[i]-start[i])*(frac + start[i]));
}
```

```
var swatch = document.getElementById("swatch");
swatch.style.backgroundColor = "rgb("+color.join(",")+");
```

Adding elements to the DOM

- The DOM is a tree; have to add our new element as the child of some parent

```javascript
function removeFromDOM(id) {
    e = document.getElementById(id);
    if (e !== null) {
        p = e.parentNode;
        p.removeChild(e);
    }
}
```
CREATE TABLE AirportFood (city TEXT, airport TEXT, state TEXT, chipotle BOOLEAN)

<table>
<thead>
<tr>
<th>City</th>
<th>Airport</th>
<th>State</th>
<th>Chipotle in airport</th>
</tr>
</thead>
<tbody>
<tr>
<td>San Francisco</td>
<td>SFO</td>
<td>CA</td>
<td>TRUE</td>
</tr>
<tr>
<td>Sacramento</td>
<td>SMF</td>
<td>CA</td>
<td>FALSE</td>
</tr>
</tbody>
</table>

SQL query

SELECT columns FROM table WHERE Boolean

- Always returns a list of objects
- One list element per row, properties of objects are the selected columns

```
SELECT airport FROM AirportFood WHERE state = 'CA'
```

```
[ {"airport": "sfo"}, {"airport": "smf"} ]
```

```
SELECT * FROM AirportFood WHERE chipotle=TRUE
```

```
[ {"city": "San Francisco", "airport": "sfo", "state": "CA", "chipotle": true } ]
```

sqlite3

- As far as SQL is concerned, it always returns a list
- Sqlite3 gives us three interfaces to this list

```
db = new sqlite3("dbfile.db");

db.all("SELECT * FROM AirportFood WHERE chipotle=TRUE", function(err, list) {
    for (i=0, i<list.length, i++) {
        row = list[i];
        console.log(row.city);
    }
});
```

Example

- List of rows
- Rows are objects
- Object properties are columns
Final thoughts

- Web software is constantly in a state of transition
- What languages and modules we used is not so important; two years from now something else (hopefully better) will be popular
- Important concepts:
  - Asynchronous programming
  - Databases
  - Security
  - DOM
  - Objects

Things to check out

- Books
  - Data visualization on the Web
  - Javascript: The good parts
  - Design for hackers
- Tools
  - Browser: Jquery
  - Server: Express
  - A gazillion more…. 

Thank you!

- We’re constantly developing this course.

- Thank you for helping improve it!