ECS 162
Web Programming

5/3

Midterm Mon May 6

- SCANTRON
- Open notes. No computers/phones.
- Recommend: make a few pages of good notes.
  - Making notes uncovers things you need to study
  - Include code snippets to illustrate syntax of commands
- Last year's test, on "labs" page
- Comics on their own page
- You'll get an email with your assigned seat on Sunday night.

Programming problems

- Fill in some functions that are part of a short Javascript program, probably using a CORS API or doing an AJAX request (so, something with an XMLHttpRequest).
- Make request, specify callback
- When response comes back, extract information from JSON and modify DOM elements

Other topics

- buttons, textboxes, images, paragraphs, divs, etc.
- CSS, including flexbox
- media queries
- Javascript data types, conversions, equality
- objects
- functions
- variable scope
- closures
- string and array methods and properties
- server, handling static files and queries

HTML

- Browser executes the HTML by constructing (initializing) the Document Object Model (DOM)
- Link CSS in head, Javascript usually right before the end of the body
- Later Javascript code modifies the DOM (not the HTML)
- Elements can be hidden but not removed by changing their display property

CSS

- CSS properties control how elements are displayed
- Browser uses CSS as it renders the DOM
- Font stacks

  font-family: ‘Helvetica Neue’, Helvetica, Arial, sans-serif;
- Using selectors (id, class, element type)
- CSS cascade rules
  - Most specific
  - If equally specific, last applied

Media queries

- Media queries for fundamental changes to the CSS.
- Base media queries only on width of the viewport, etc, avoid special behavior for specific devices.

```
/* on small screens */
@media (max-width: 480px) {
   #menuIcon { display: block; }  /* show menu icon */
   nav { display: none; }         /* hide nav bar */
}
```

Changing styles in Javascript

```
div.narrow {
        width: 200px; }
div.wide {
        width: 60%; }
```

- Best practice: change class names in Javascript, let those determine the styles. Then we can apply media queries to different classes to handle different UI modes properly.

Color

- In CSS:

  color: #ff8020;

  means all the red, about half the green, and a little blue -- an intense reddish orange.

```
background-color: #99ff99;
// text color
color: #000000;
```

Default CSS layout

- Inline elements follow one another like words in text
  - <img> is inline, also text things like <strong>
- Block elements stack on top of each other, like paragraphs
  - <p>,<div>,<header>
  - Width fills container, height shrinks to fit contents
- Can set width of block element to something smaller

Size units

- Reference pixel size, designed to subtend same width in user's visual field, irrespective of device - margin: 10px;
- Percent of container size - width: 20%;
- Percent of viewport size - height: 100vh;
- For text, em, which sizes font relative to normal font size on that device, which is designed to be small but readable - font-size: 1.3 em;

Flexbox

- Setting

  ```
  display: flex;
  ```

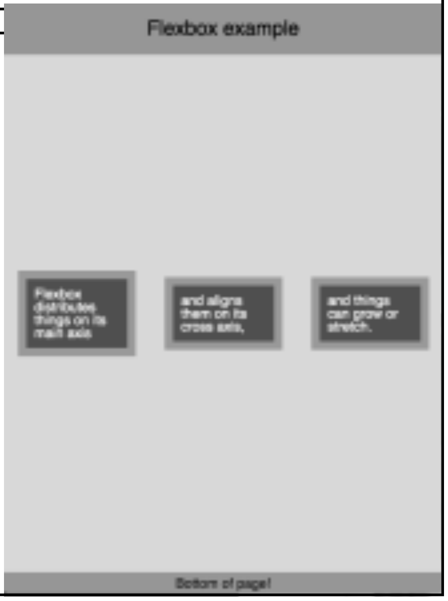
  makes a box a flexbox container

  ```
  flex-direction: row;    or
  flex-direction: column;
  ```

  indicates main layout direction for its contents, the other direction is the cross direction.
- Two main approaches to distributing items.

Using justify & align

```
body {   display: flex;
           flex-direction: column; }
main {   display: flex;
           flex-direction: row;
         justify-content: space-around;
           align-items: center;
           flex-grow: 1;}
div {  display: flex;
        flex-directon: row;
        align-items: center;    }
p { width: 100px;   }
```



Flexbox example

Flexbox distributes things on its main axis

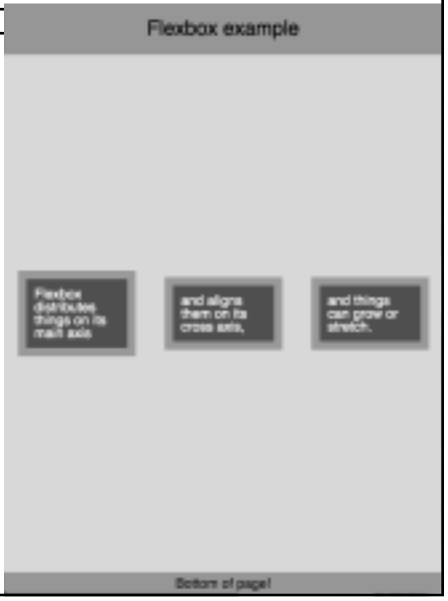
and aligns them on its cross axis,

and things can grow or stretch.

Bottom of page!

Using grow & shrink

```
body {   display: flex;
           flex-direction: column; }
main {   display: flex;
           flex-direction: row;
         justify-content: space-around;
           align-items: center;
           flex-grow: 1;}
div {  display: flex;
        flex-directon: row;
        align-items: center;    }
p { width: 100px;   }
```



Flexbox example

Flexbox distributes things on its main axis

and aligns them on its cross axis,

and things can grow or stretch.

Bottom of page!

Using using shrink and grow

```
body {   display: flex;
          flex-direction: column; }
main {   display: flex;
          flex-direction: row;
        justify-content: space-around;
          align-items: center;
          flex-grow: 1;}
div {  display: flex;
       flex-directon: row;
       align-items: center;   }
p { width: 100px;  }
```

How does flexbox determine the width of the divs?

Using using shrink and grow

```
body {   display: flex;
          flex-direction: column; }
main {   display: flex;
          flex-direction: row;
        justify-content: space-around;
          align-items: center;
          flex-grow: 1;}
div {  display: flex;
       flex-directon: row;
       align-items: center;   }
p { width: 100px;  }
```

How does flexbox determine the width of the divs?

They get the width of their contents, plus margin and padding.

What if we took out the css on the paragraph?

Using using shrink and grow

```
body {   display: flex;
         flex-direction: column; }
main {   display: flex;
         flex-direction: row;
        justify-content: space-around;
         align-items: center;
         flex-grow: 1;}
div {  display: flex;
       flex-directon: row;
       align-items: center;   }
p { width: 100px;  }
```

How does flexbox determine the width of the divs?

They get the width of their contents, plus margin and padding.

What if we took out the css on the paragraph?

The paragraphs, and divs, would expand to fill the width of main.

Javascript

- All numbers are really floating point
- Operations and comparisons do automatic string/number conversion, except for "==="

```
3.0 == "3" // true!
3.0 === "3" // false!
6+"cars" === "6cars" // true!
```

Functions

- Two kinds of declarations

```
function makeTile () { .... }
let makeTile = function () { .... }
```

- Second form uses an anonymous function on RHS
- Functions are objects, and can have properties and methods.

```
if (makeTile.tileID  == undefined) {
     makeTile.tileID  = 0; } else { makeTile.tileID++  }
```

Scope

- Variables defined with let are visible throughout the function; variables defined with let are visible throughout their block (bounded by {}).
- Variables defined with var are visible throughout their functions; generally not needed.
- Global variables, and code to be run on initialization, should be outside of any function, usually at the top of the file.
- When is initialization for browser code? For server code?

Scope

- Variables defined with let are visible throughout the function; variables defined with let are visible throughout their block (bounded by {}).
- Variables defined with var are visible throughout their functions; generally not needed.
- Global variables, and code to be run on initialization, should be outside of any function, usually at the top of the file.
- When is initialization for browser code? For server code?

Scope

- Variables defined with let are visible throughout the function; variables defined with let are visible throughout their block (bounded by {}).
- Variables defined with var are visible throughout their functions; generally not needed.
- Global variables, and code to be run on initialization, should be outside of any function, usually at the top of the file.
- When is initialization for browser code? For server code?

loading of script; startup of server

Question

```
let element = document.createElement("button")
document.querySelector("body").appendChild(element);
element.textContent = 0;
let count = 0;
makeButtonFunction();

function makeButtonFunction () {
    element.onclick = function () {
        count++;
        element.textContent = count;
    }}
```

How can we avoid having count be global?

Question

```
var element = document.createElement("button")
document.querySelector("body").appendChild(element);
element.textContent = 0;
makeButtonFunction();

function makeButtonFunction () {
    let count = 0;
    element.onclick = function () {
        count++;
        element.textContent = count;
    }}
```

Put it in a closure!

Objects

- When defined as literals or with assignment to methods and properties, everything is public

```
let car = { "make": "Lexus", "price": 38,000 }
car.reportPrice = function () {
     console.log(this.price);   }
console.log(car.make);
```

- When defined using class and new, allows private data and methods via the scope of the constructor function.

JSON

- The JSON data format is a Javascript literal

```
let dataString = ’ {"car": "Toyota"} '; // JSON
yourCar = JSON.parse(dataString);
// yourCar is an object
let anotherString = JSON.stringify(yourCar);
dataString == anotherString;  // true!
```

- JSON cannot include methods

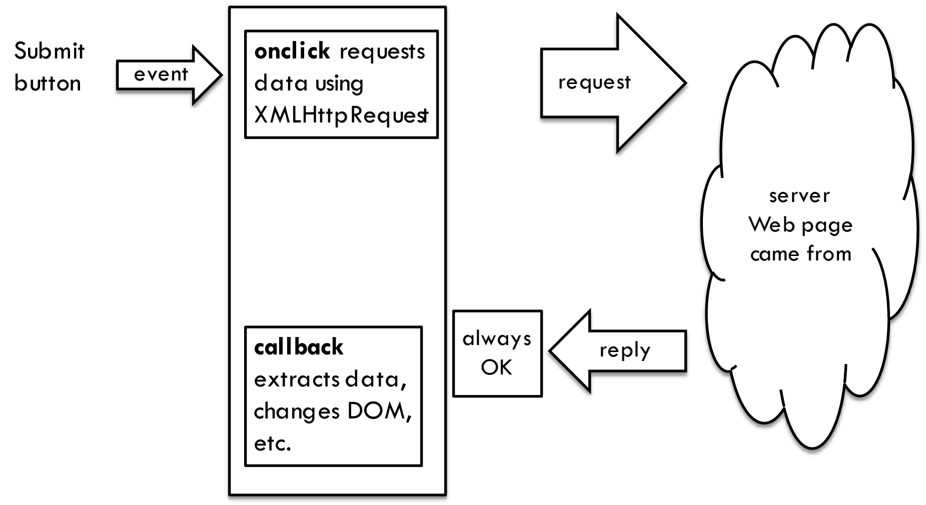
JSON Objects can be complex

```
movieData = {"total": 2, "movies": [
   { "id": "770672122", "title": "Toy Story 3", "year":
     2010, "mpaa_rating": "G", "runtime": 103,
   "critics_consensus": "Deftly blending comedy, adventure, and
     honest emotion, Toy Story 3 is a rare second sequel that
     really works.",
    "release_dates": {
         "theater": "2010-06-18",
        "dvd": "2010-11-02"
        }, …
rating = movieData.movies[0].mpaa_rating;
```

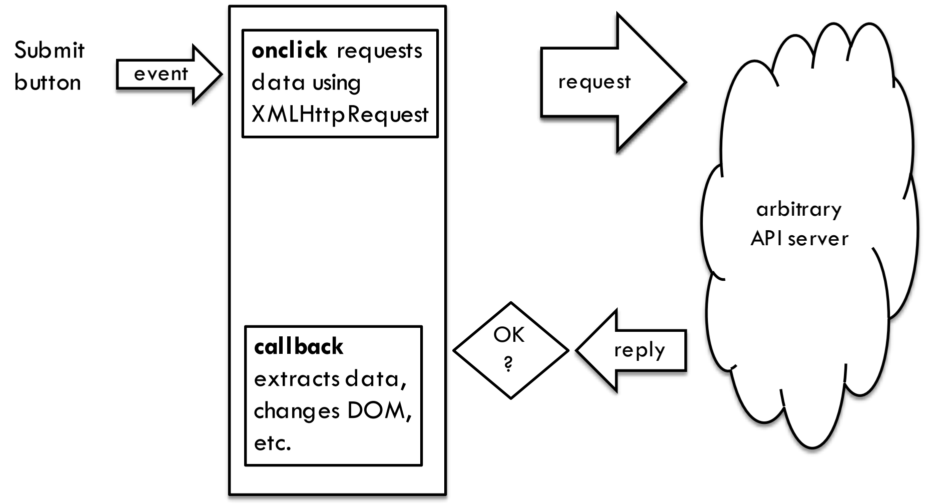
Same Origin Policy

- In general, Web pages can only send queries to the server they came from.
- In an AJAX query, we only give the query part of the URL, not the server name.
- CORS is an exception. The browser sends a CORS request to the API server it requests.
- If the API server labels the response as public, then the browser passes it back to our Javascript code; otherwise error!
- This deters random Javascript code from trying to get to other servers we are logged into.

AJAX query


Submit button
event
onclick requests data using XMLHttpRequest
request
server Web page came from
callback extracts data, changes DOM, etc.
always OK
reply

CORS API interaction


Submit button
event
onclick requests data using XMLHttpRequest
request
arbitrary API server
callback extracts data, changes DOM, etc.
OK ?
reply

Animation

- Javascript animations change the display at intervals, using a timer function such as setInterval();
- The timer function calls a callback function after waiting for some number of milliseconds:

```
let timer = setInterval( function() {
        pos = moveTurtle(pos);
        pos = pos+1;}
        }, 80);
```

Closure

- A function defined inside another function has access to all the local variables.
- When the outer function exits, it's local variables are stored in a closure.
- If there was more than one inner function created, they all have access to the same closure.
- We often use this to pass information to callback functions (**which get called later!**) from the functions in which they are defined.
- This is also how Javascript implements static variables and private data for objects.

Class constructor

```
class Weather {
  constructor (t,w) {
     this.fahrenheit = t;
     this.wind = w;
     this.celsius = function() {
       return (t-32)*5/9;
     }
  }
}
```

Not a great piece of code

```
davisWeather = new Weather(77, 22);
davisWeather.celsius() // returns 25

davisWeather.fahrenheit = 86;
davisWeather.celsius() // returns?
```

Tricky problem

```
class Weather {
   constructor  (t,w) {
       this.fahrenheit  = t;
       this.wind  = w;
       this.celsius  = function() {
         return (t-32)*5/9;
       }
   }
}
```

The closure of this.celsius is the constructor; it has access to the private variable t, which is in the closure of the constructor.

But t does not change when this.fahrenheit does!

Correct!

```
class Weather {
   constructor  (t,w) {
       this.fahrenheit  = t;
       this.wind  = w;
       this.celsius  = function() {
         return (this.fahrenheit-32)*5/9;
       }
   }
}
```

Server

- A server is a computer, on the internet, running software that responds to HTTP requests.
- The HTTP requests often consist mainly of a URL, either the name of a file to download (eg. palindrome.js), or a query that will be answered with a JSON string.
- In express, we configure the server by stringing together middleware functions to make a pipeline. Each stage typically either can respond to the HTTP request or passes it on to the next one.

Server

- As an app programmer, we usually spend our time in the custom middleware functions that respond to queries.