Comp[redacted]: Web client development

[redacted]

First, a reminder

The web is:

Browsers (running JavaScript, loading HTML, images, etc.)

- Also mobile clients (iOS/Android/etc.)

Servers (written in all sorts of different programming languages)

Protocols (HTTP, possibly encrypted, etc.)

- Also higher-level things on top of HTTP, like sending JSON-formatted messages

Networks (the Internet)

JavaScript is:

A dynamically typed scripting language, not unlike Python

`var x = ...`; The type of x isn't known until runtime.

With lambdas everywhere and an unusual object system

Based on "prototypes" rather than "classes" like Java

With a "standard" set of APIs for dealing with the browser

Today's lecture: JavaScript and the browser

Digression: Traditional graphics programming

In the old days, graphics APIs exposed low-level things

Draw lines & rectangles, render text

High-performance 3D graphics was typically an unrelated API

UI widgets exposed higher-level things

Buttons, sliders, text boxes

Wildly varying APIs for dealing with system events (mouse, keyboard)

Event queues, callbacks, and more

Many current operating systems (e.g., Android, iOS) follow a similar pattern

The web is much nicer

Sophisticated fonts, layouts, style sheets, etc.
And non-programmer tools (Adobe DreamWeaver, etc.) to help out

Web 1.0: lay out a static HTML doc, decorate with JavaScript behaviors
onClick, onMouseOver, etc. placed discretely into the HTML

But that’s not really how it’s done today

What, exactly, is “Web 2.0”?

Much debate on what this term even means.

The web browser is hosting a dynamic application, not just a static page

- If you disable JavaScript, you don’t just get a limited page. You get nothing.

The web server responds to requests from client applications

- These might be JSON, XML, protobufs, whatever.

“Content” might come from users, so there are interesting security issues

- Discussed on Wednesday

Web pages aggregate stuff from multiple sources

- Advertising, “tracking”, “mashups”, etc.
- And again, lots more interesting security issues.

The two biggest Web 2.0 features

“Ajax” (asynchronous JavaScript): JS code can make its own network connections, fetch data, without forcing the page to reload.

DOM (document object model): JS code can reach deep into a web page and change its structure and contents, on the fly.

Notable: XMLHttpRequest invented in 1999 by Microsoft for Outlook Web Access

Rapidly adopted by other browsers.

Formed the basis for Google’s Gmail and other web apps to come.

Also of note, “HTML5”: attempts to standardize all of this

But older browsers, especially IE, don’t follow HTML5

Marc Andreessen's predictions (1995?)

Q: A quote of yours that I've always loved is that Netscape would render Windows "a poorly debugged set of device drivers."

Andreessen: In fairness, you have to give credit for that quote to Bob Metcalfe, the 3Com founder.

Q: Oh, it wasn't you? It's always attributed to you.

Andreessen: I used to say it, but it was a retweet on my part. [*Laughs*.] But yes, the idea we had then, which seems obvious today, was to lift the computing off of each user's device and perform it in the network instead.

http://www.wired.com/2012/04/ff_andreessen/

Putting it together

Example app: our RPN calculator

HTML: set up the layout of the screen, load the JavaScript

CSS (cascading style sheets): fonts, etc.

JavaScript: add behaviors

RPNCalc's HTML (generated by Java code)

```
<html>
<head>
    <title>Comp[redacted] RPN Calculator!</title>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link href="/mui-0.6.0/css/mui.css" rel="stylesheet" type="text/css">
    <link href="/commandline.css" rel="stylesheet" type="text/css">
    <script type="text/javascript" src="/mui-0.6.0/js/mui.min.js"></script>
    <script type="text/javascript" src="/jquery-1.12.4.min.js"></script>
    <script type="text/javascript" src="/rpncalc.js"></script>
</head>
<body>
<header id="header">
    <nav id="appbar" class="mui-container-fluid">
        <h1>Comp[redacted] RPN Calculator!</h1>
    </nav>
</header>
<div id="textOutput">
</div>
<div id="goButton">
    <button class="mui-btn mui-btn--fab mui-btn--primary">&gt;</button>
</div>
<div id="footer">
    <form>
        <div class="mui-textfield">
            <input id="commandLine" type="text" autocomplete="off" placeholder="type your commands here">
        </div>
        <input id="hello" type="text" value="hello" readonly hidden>
    </form>
</div>
</body>
</html>
```

RPNCalc's HTML (generated by Java code)

```
<html>
<head>
    <title>Comp[redacted] RPN Calculator!</title>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link href="/mui-0.6.0/css/mui.css" rel="stylesheet" type="text/css">
    <link href="/commandline.css" rel="stylesheet" type="text/css">
    <script type="text/javascript" src="/mui-0.6.0/js/mui.min.js"></script>
    <script type="text/javascript" src="/jquery-1.12.4.min.js"></script>
    <script type="text/javascript" src="/rpncalc.js"></script>
</head>
<body>
<header id="header">
    <nav id="appbar" class="mui-container-fluid">
        <h1>Comp[redacted] RPN Calculator!</h1>
    </nav>
</header>
<div id="textOutput">
</div>
<div id="goButton">
    <button class="mui-btn mui-btn--fab mui-btn--primary">&gt;</button>
</div>
<div id="footer">
    <form>
        <div class="mui-textfield">
            <input id="commandLine" type="text" autocomplete="off" placeholder="type your commands here">
        </div>
        <input id="hello" type="text" value="hello" readonly hidden>
    </form>
</div>
</body>
</html>
```

Load CSS stylesheets

RPNCalc's HTML (generated by Java code)

```
<html>
<head>
    <title>Comp[redacted] RPN Calculator!</title>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link href="/mui-0.6.0/css/mui.css" rel="stylesheet" type="text/css">
    <link href="/commandline.css" rel="stylesheet" type="text/css">
    <script type="text/javascript" src="/mui-0.6.0/js/mui.min.js"></script>
    <script type="text/javascript" src="/jquery-1.12.4.min.js"></script>
    <script type="text/javascript" src="/rpncalc.js"></script>
</head>
<body>
<header id="header">
    <nav id="appbar" class="mui-container-fluid">
        <h1>Comp[redacted] RPN Calculator!</h1>
    </nav>
</header>
<div id="textOutput">
</div>
<div id="goButton">
    <button class="mui-btn mui-btn--fab mui-btn--primary">&gt;</button>
</div>
<div id="footer">
    <form>
        <div class="mui-textfield">
            <input id="commandLine" type="text" autocomplete="off" placeholder="type your commands here">
        </div>
        <input id="hello" type="text" value="hello" readonly hidden>
    </form>
</div>
</body>
</html>
```

Load a bunch of JavaScript
(here local, could be remote)

RPNCalc's HTML (generated by Java code)

```
<html>
<head>
    <title>Comp[redacted] RPN Calculator!</title>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link href="/mui-0.6.0/css/mui.css" rel="stylesheet" type="text/css">
    <link href="/commandline.css" rel="stylesheet" type="text/css">
    <script type="text/javascript" src="/mui-0.6.0/js/mui.min.js"></script>
    <script type="text/javascript" src="/jquery-1.12.4.min.js"></script>
    <script type="text/javascript" src="/rpncalc.js"></script>
</head>
<body>
<header id="header">
    <nav id="appbar" class="mui-container-fluid">
        <h1>Comp[redacted] RPN Calculator!</h1>
    </nav>
</header>
<div id="textOutput">
</div>
<div id="goButton">
    <button class="mui-btn mui-btn--fab mui-btn--primary">&gt;</button>
</div>
<div id="footer">
    <form>
        <div class="mui-textfield">
            <input id="commandLine" type="text" autocomplete="off" placeholder="type your commands here">
        </div>
        <input id="hello" type="text" value="hello" readonly hidden>
    </form>
</div>
</body>
</html>
```

Every element has a "type", an optional "id", and optional "classes"

What's a "class" and an "id"?

Class: handles into the style sheets for fonts, spacing, etc.

```
.mui-form-control {
  -webkit-animation-duration: 0.0001s;
  animation-duration: 0.0001s;
  -webkit-animation-name: mui-node-inserted;
  animation-name: mui-node-inserted;
  display: block;
  background-color: transparent;
  color: rgba(0, 0, 0, 0.87);
  border: none;
  border-bottom: 1px solid rgba(0, 0, 0, 0.26);
  outline: none;
  height: 32px;
  width: 100%;
  font-size: 16px;
  padding: 0;
  box-shadow: none;
  border-radius: 0px;
  background-image: none;
}
```

Id: handles that JavaScript can grab onto for adding behaviors

```
document.getElementById('commandLine')
```

Do you need to understand CSS?

No, not really

Suffice to say that multiple rules can attach to any given element in a document.

CSS provides rules (and exceptions to the rules) for how to resolve all that.

There really isn’t a whole lot of magic behind CSS

But there are a lot of people complaining about how awful it is.

When in doubt, beat on it until you get what you want

Lots of web pages with examples, reference guides, and of course you can read the “source code” to web sites that you like.

StackOverflow is incredibly useful here.

Do you need to understand JavaScript?

We introduced the JavaScript language earlier

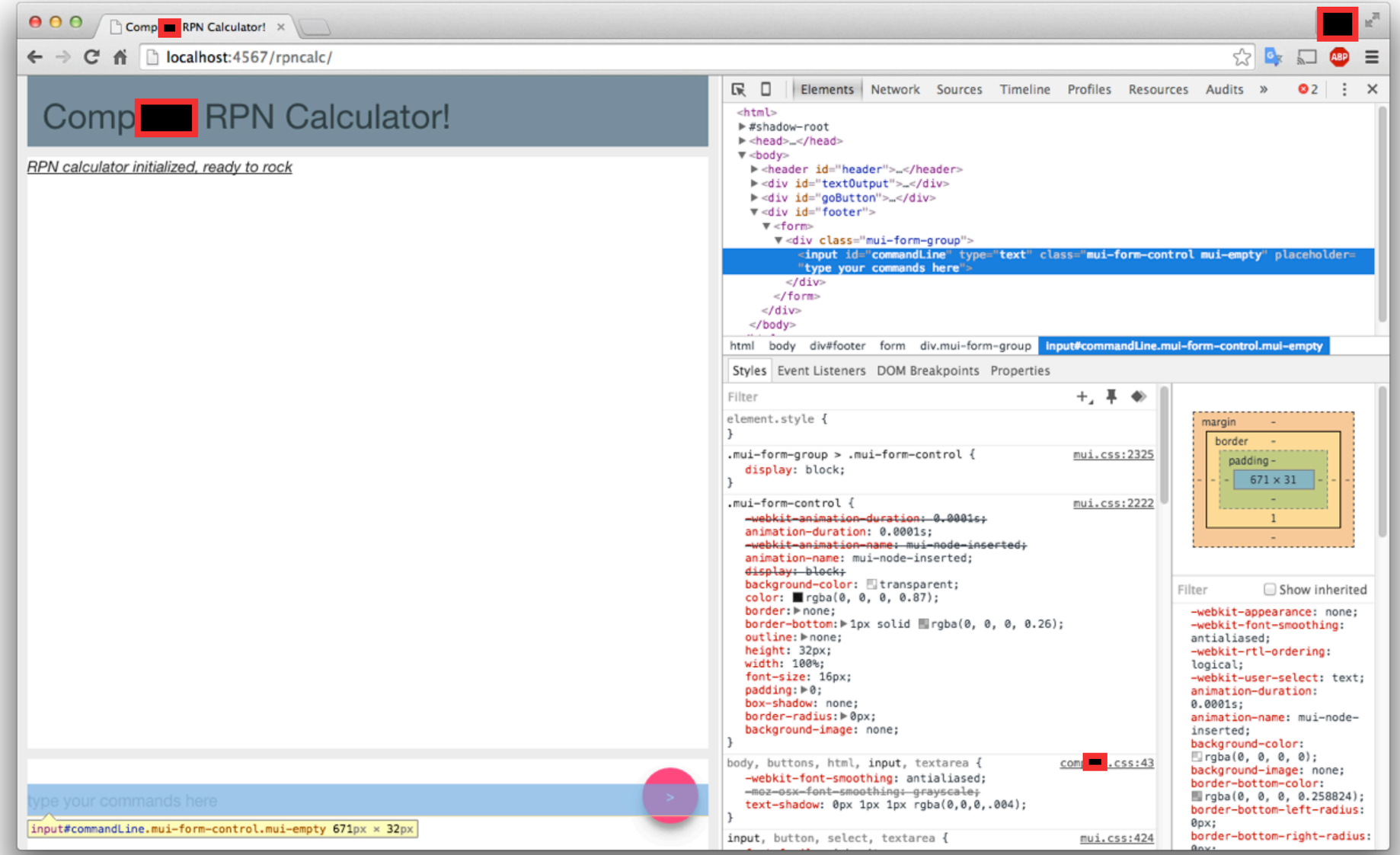
Today: JavaScript's view of the browser environment: the Document Object Model (DOM)

Several top-level objects and everything else is below that

Learning your way around the DOM

See the real-time status of the DOM (not just the HTML that made it)

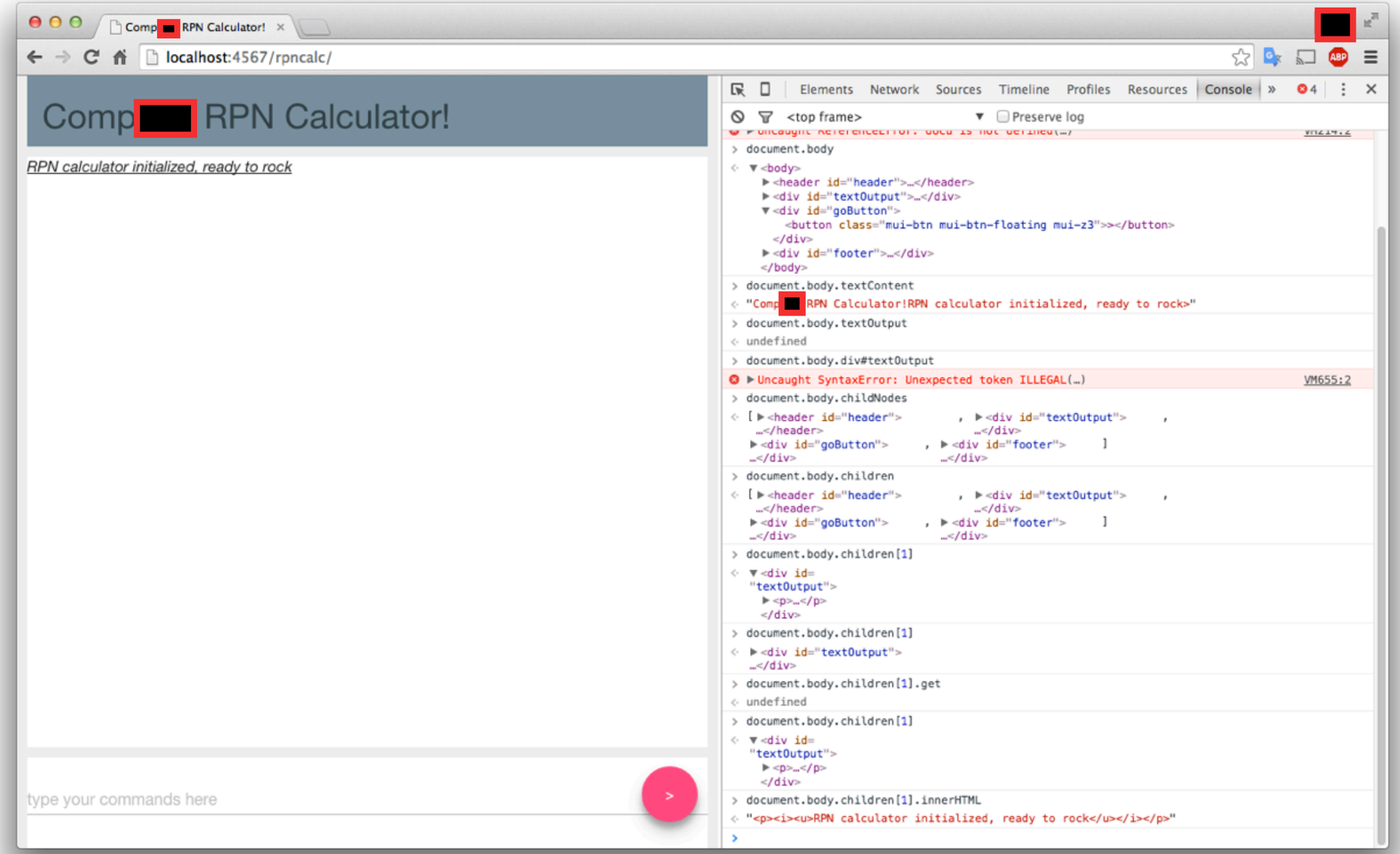
And see what CSS styles are attached to each object



Using the JavaScript Console

Type commands and see what they return

Helpful to do this while staring at the reference docs in another window



Where do you get started?

Lots of general documentation online, tons of StackOverflow posts, etc.

Mozilla's intro is as good as anything:

- https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction

Key concepts: document, element, window

document: the top-level view of the browser page

element: any individual widget in the page, as found above

window: the container where the document lives

- Common trick: setting window.location to a new URL causes the whole page to be reloaded

But… "real" JavaScript programmers use libraries that help them.

JQuery

Example raw JavaScript (which won't work in IE6):

```
var goButton = document.getElementById("goButton")
goButton.addEventListener(“click", fetchQuery)
```

JQuery (and other libraries like it) simplify this, runs everywhere:

```
$("#goButton").on("click", fetchQuery)
```

Notable: the window doesn’t load right away

Raw JavaScript:

```
window.onload = function() {
    printParagraph("<i><u>RPN calculator initialized, ready to rock</u></i>")

    // other objects are in the DOM, ready for more callbacks to be registered

}
```

JQuery:

```
$(document).ready(function() {
    printParagraph("<i><u>RPN calculator initialized, ready to rock</u></i>")

    $("#goButton").on("click", fetchQuery)
    $("#commandLine").keydown(function (event) {
        if(event.keyCode == 13) {
            event.preventDefault() // prevent carriage-return from triggering a page reload
            fetchQuery()
        }
    })
})
```

Asynchronous requests

Of course, they’re not all compatible

Again, this is where JQuery and other libraries try to paper over the difference

```
var xhrObject = (function(){
  // adapted from MooTools
  var XMLHTTP = function(){
    return new XMLHttpRequest();
  }

  var MSXML2 = function(){
    return new ActiveXObject('MSXML2.XMLHTTP');
  }

  var MSXML = function(){
    return new ActiveXObject('Microsoft.XMLHTTP');
  }

  try {
    XMLHTTP()
    return XMLHTTP
  } catch (e){
    try {
      MSXML2()
      return MSXML2
    } catch (e){
      MSXML()
      return MSXML
    }
  }
})();
```

Code snippet here from another JQuery-like library: tries three different APIs

JQuery networking

Build a JavaScript object that describes the request, includes callbacks

```
function dispatchQuery(input) {
    console.log("dispatching query: " + input)
    $.ajax( {
        url: "/rpnserver/",
        type: "GET",
        data: {'input': input},
        success: function(data) {
            console.log("success: " + data)
            printParagraph(JSON.parse(data).response)
        },
        error: function(data) {
            console.log("error: " + data)
            printParagraph('<b>Bah! ' + data + ' error!</b>')
        }
    })
}
```

JQuery networking

Build a JavaScript object that describes the requ

```
function dispatchQuery(input) {
    console.log("dispatching query: " + input)
    $.ajax( {
        url: "/rpnserver/",
        type: "GET",
        data: {'input': input},
        success: function(data) {
            console.log("success: " + data)
            printParagraph(JSON.parse(data).response)
        },
        error: function(data) {
            console.log("error: " + data)
            printParagraph('<b>Bah! ' + data + ' error!</b>')
        }
    })
}
```

Outbound request (just a JS key/value object, gets converted to URL “query string”)

JQuery networking

Build a JavaScript object that describes the request, includes callbacks

```
function dispatchQuery(input) {
    console.log("dispatching query: " + input)
    $.ajax( {
        url: "/rpnserver/",
        type: "GET",
        data: {'input': input},
        success: function(data) {
            console.log("success: " + data)
            printParagraph(JSON.parse(data).response)
        },
        error: function(data) {
            console.log("error: " + data)
            printParagraph('<b>Bah! ' + data + ' error!</b>')
        }
    })
}
```

Lambda to handle the response when it comes back

JQuery networking

Build a JavaScript object that describes the request, includes callbacks

```
function dispatchQuery(input) {
    console.log("dispatching query: " + input)
    $.ajax( {
        url: "/rpnserver/",
        type: "GET",
        data: {'input': input},
        success: function(data) {
            console.log("success: " + data)
            printParagraph(JSON.parse(data).response)
        },
        error: function(data) {
            console.log("error: " + data)
            printParagraph('<b>Bah! ' + data + ' error!</
        }
    })
}
```

built-in safe JSON parsing (in recent JS, not in older browsers)

There are lots and lots of “little” libraries

Sometimes you need simple things

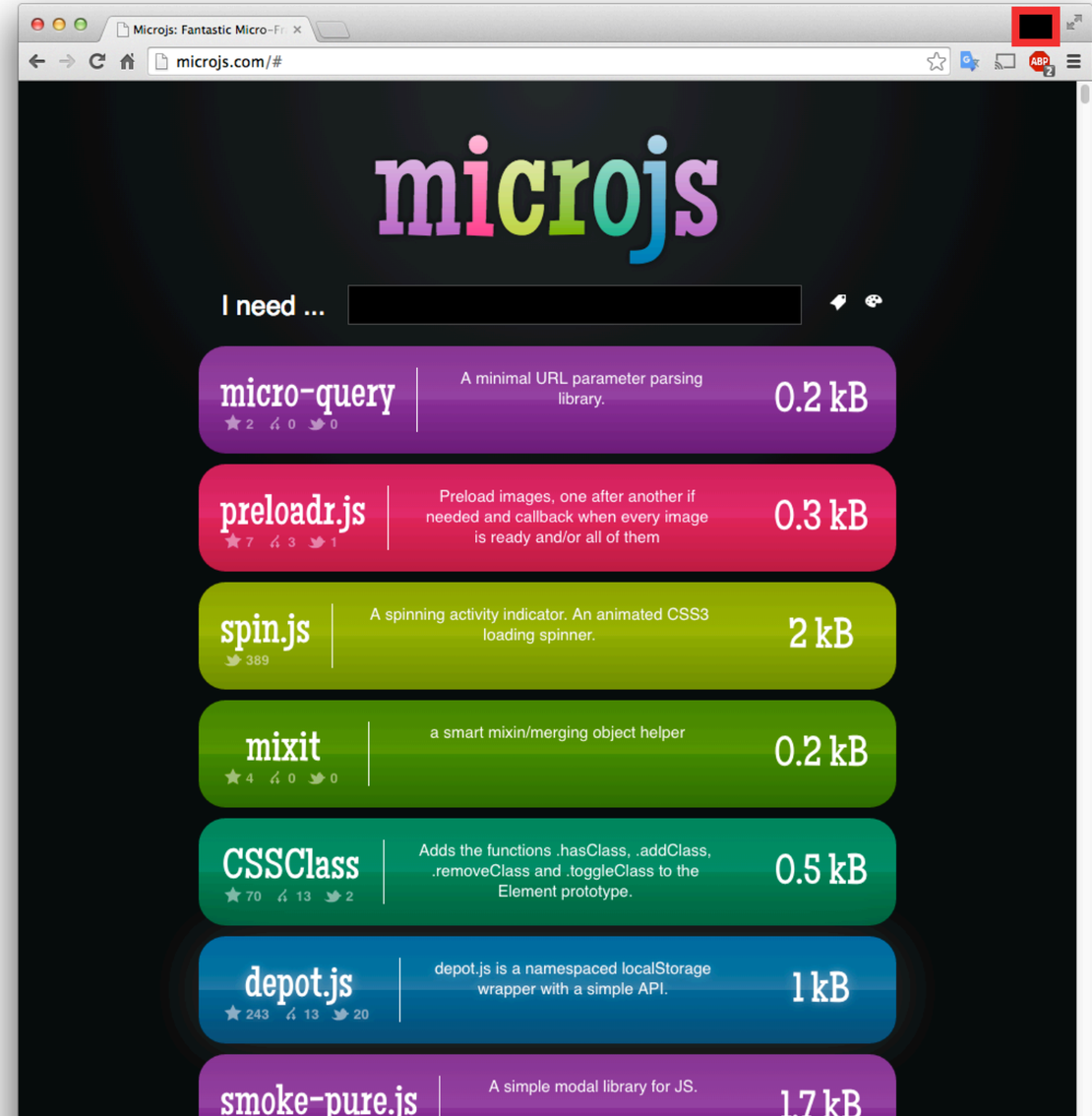
So go get a simple library!

Many “big” sites use “big” frameworks

React, AngularJS, etc.

Or they just write in some *other* language.

Any advice I give you today will be completely obsolete in two years!



How do “real” web sites do it?

Code inside HTML vs. HTML inside code

Code inside HTML

ColdFusion, PHP, etc.: HTML with “escapes” to hide code inside it

Security risks: can an attacker inject code?

Testing and scalability: code is entangled with data, hard to unit test

Old-school JavaScript was typically used this way for small behaviors

HTML inside code

You’ve seen this with the j2html library

Sometimes more verbose, but easier to test

Less security risk

Impossible for non-programmers to edit

Facebook's React

https://facebook.github.io/react/

A couple features of note:
EcmaScript 6 = JavaScript 6 = JavaScript with new lambda syntax and classes!
http://es6-features.org/

JSX: Facebook extension to have inline HTML (versus code to convert JSON)
https://facebook.github.io/react/docs/jsx-in-depth.html

Code is compiled down to older JavaScript for older browsers!

JavaScript evolves quickly

Java evolves slowly: two major revisions in the past 20 years

JavaScript and languages on top (like JSX) evolve fast
Something new every single year!

The principles are all the same
Everything you've learned this semester (filters, maps, etc.) is universal