What does JavaScript have to do with the Web?
Way back in 1995...

Web browsers had no way to run external code

But, there were tons of “mobile code systems” in the world

Tcl/Tk: Popular with Unix tool builders, still running today in many places
Java: Originally targeting television set-top boxes
  Released in ’95 with “HotJava” web browser
Many others that never achieved commercial success

Meanwhile, inside Netscape...
Brendan Eich’s original goals of JavaScript:
Lightweight execution inside the browser
Safety / security
Easy to learn / easy to write
Scheme-like language with C / Java-like syntax

Easy to add small behaviors to web pages
One-liners: click a button, pop up a menu
The big debate inside Netscape therefore became “why two languages? why not just Java?” The answer was that two languages were required to serve the two mostly-disjoint audiences in the programming ziggurat who most deserved dedicated programming languages: the component authors, who wrote in C++ or (we hoped) Java; and the “scripters”, amateur or pro, who would write code directly embedded in HTML.

Whether any existing language could be used, instead of inventing a new one, was also not something I decided. The diktat from upper engineering management was that the language must “look like Java”. That ruled out Perl, Python, and Tcl, along with Scheme. Later, in 1996, John Ousterhout came by to pitch Tk and lament the missed opportunity for Tcl.

I’m not proud, but I’m happy that I chose Scheme-ish first-class functions and Self-ish (albeit singular) prototypes as the main ingredients. The Java influences, especially y2k Date bugs but also the primitive vs. object distinction (e.g., string vs. String), were unfortunate.

https://brendaneich.com/tag/history/
Why JavaScript in Comp215?

Every Comp215 project is in Java, so why should you know JavaScript?

1) It’s the most widely available programming language in the world. It’s in every browser, everywhere, and for many people, is their gateway to “real” programming.

2) Browsers are everywhere, you should know how they work.

3) You’ll often want to modify somebody else’s JavaScript. And because you know Java, you’ll find JavaScript to be reasonably easy to read and tweak.
You already know big parts of JavaScript

Here’s a JSON data structure:

```json
{
  "authors": [  
    {
      "name": "Alice Action",
      "email": "alice@rice.edu"
    },
    {
      "name": "Bob Bigbooté",
      "email": "bob@rice.edu"
    },
    {
      "name": "Charlie Chancery",
      "email": "charlie@rice.edu"
    },
    {
      "name": "Dorothy Danger",
      "email": "dorothy@rice.edu"
    },
    {
      "name": "Eve Ebullience",
      "email": "eve@rice.edu"
    }
  ],
  "articles": [  
    {
      "title": "Rice Owls Win Superbowl, Surprised NFL Fans React",
      "authors": [ "alice@rice.edu", "bob@rice.edu" ],
      "body": "In a sports upset few could have imagined, the Rice Owls defeated the New England Patriots 34–7 in the Superbowl."
    },
    {
      "title": "Rice Band Deflated Footballs",
      "authors": [ "bob@rice.edu" ],
      "body": "Rice bandmembers thought they were just making fun of Deflategate, but their prank somehow propelled the Owls to victory."
    }
  ]
}
```
You already know big parts of JavaScript

Here’s a JavaScript program:

```javascript
var x = {
  "authors": [
    {
      "name": "Alice Action",
      "email": "alice@rice.edu"
    },
    {
      "name": "Bob Bigbooté",
      "email": "bob@rice.edu"
    },
    {
      "name": "Charlie Chancery",
      "email": "charlie@rice.edu"
    },
    {
      "name": "Dorothy Danger",
      "email": "dorothy@rice.edu"
    },
    {
      "name": "Eve Ebullience",
      "email": "eve@rice.edu"
    }
  ],
  "articles": [
    {
      "title": "Rice Owls Win Superbowl, Surprised NFL Fans React",
      "authors": [ "alice@rice.edu", "bob@rice.edu" ],
      "body": "In a sports upset few could have imagined, the Rice Owls defeated the New England Patriots 34–7 in the Superbowl."
    },
    {
      "title": "Rice Band Deflated Footballs",
      "authors": [ "bob@rice.edu" ],
      "body": "Rice bandmembers thought they were just making fun of Deflategate, but their prank somehow propelled the Owls to victory."
    }
  ]
};
```
You already know big parts of JavaScript

Here's a JavaScript program, optional quotation marks removed:

```javascript
var x = {
    authors: [
        {
            name: "Alice Action",
            email: "alice@rice.edu"
        },
        {
            name: "Bob Bigbooté",
            email: "bob@rice.edu"
        },
        {
            name: "Charlie Chancery",
            email: "charlie@rice.edu"
        },
        {
            name: "Dorothy Danger",
            email: "dorothy@rice.edu"
        },
        {
            name: "Eve Ebullience",
            email: "eve@rice.edu"
        }
    ],
    articles: [
        {
            title: "Rice Owls Win Superbowl, Surprised NFL Fans React",
            authors: [ "alice@rice.edu", "bob@rice.edu" ],
            body: "In a sports upset few could have imagined, the Rice Owls defeated the New England Patriots 34-7 in the Superbowl."
        },
        {
            title: "Rice Band Deflated Footballs",
            authors: [ "bob@rice.edu" ],
            body: "Rice bandmembers thought they were just making fun of Deflategate, but their prank somehow propelled the Owls to victory."
        }
    ]
};
```
Lots of familiar built-in methods

```javascript
x.authors.map(function(x) { return "XXX" + x.name + "XXX" })

["XXXAlice ActionXXX", "XXXBob BigbootéXXX", "XXXCharlie ChanceryXXX", "XXXDorothy DangerXXX", "XXXEve EbullienceXXX"]

x.articles.filter(function(a) { return a.title.includes("Rice") })

[
{
    body: "In a sports upset few could have imagined, the Rice Owls defeated the New England Patriots 34-7 in the Superbowl.",
    authors: ["alice@rice.edu", "bob@rice.edu"],
    title: "Rice Owls Win Superbowl, Surprised NFL Fans React"
},
{
    body: "Rice bandmembers thought they were just making fun of Deflategate, but their prank somehow propelled the Owls to victory."
    authors: ["bob@rice.edu"],
    title: "Rice Band Deflated Footballs"
}
]```
Easy access to fields inside objects

```javascript
x.authors.map(function(x) { return "XXX" + x.name + "XXX" })
```

```javascript
["XXXAlice ActionXXX", "XXXBob BigbootéXXX", "XXXCharlie ChanceryXXX", "XXXDorothy DangerXXX", "XXXEve EbullienceXXX"]
```

```javascript
x.articles.filter(function(a) { return a.title.includes("Rice") })
```

```javascript
[
{
  body: "In a sports upset few could have imagined, the Rice Owls defeated the New England Patriots 34-7 in the Superbowl."
  authors: [ "alice@rice.edu", "bob@rice.edu" ],
  title: "Rice Owls Win Superbowl, Surprised NFL Fans React"
},
{
  body: "Rice bandmembers thought they were just making fun of Deflategate, but their prank somehow propelled the Owls to victory."
  authors: [ "bob@rice.edu" ],
  title: "Rice Band Deflated Footballs"
}
]```
“Arrays” also act like lists

```javascript
x.authors.map(function(x) { return "XXX" + x.name + "XXX"; })

["XXXAlice ActionXXX", "XXxBob BigbootéXXX", "XXXCharlie ChanceryXXX", "XXXDorothy DangerXXX", "XXXEve EbullienceXXX"]

x.articles.filter(function(a) { return a.title.includes("Rice"); })

[
{
  body: "In a sports upset few could have imagined, the Rice Owls defeated the New England Patriots 34–7 in the Superbowl."
  authors: [ "alice@rice.edu", "bob@rice.edu" ],
  title: "Rice Owls Win Superbowl, Surprised NFL Fans React"
},
{
  body: "Rice bandmembers thought they were just making fun of Deflategate, but their prank somehow propelled the Owls to victory."
  authors: [ "bob@rice.edu" ],
  title: "Rice Band Deflated Footballs"
]```
Lots of familiar built-in methods

```javascript
xhr.authors.map(function(x) { return "XXX" + x.name + "XXX" })
> ["XXXAlice ActionXXX", "XXXBob BigbootéXXX", "XXXCharlie ChanceryXXX", "XXXDorothy DangerXXX", "XXXEve EbullienceXXX"]

xhr.articles.filter(function(a) { return a.title.includes("Rice") })
> [
  {
    body: "In a sports upset few could have imagined, the Rice Owls defeated the New England Patriots 34–7 in the Superbowl. ",
    authors: [ "alice@rice.edu", "bob@rice.edu" ],
    title: "Rice Owls Win Superbowl, Surprised NFL Fans React"
  },
  {
    body: "Rice bandmembers thought they were just making fun of Deflategate, but their prank somehow propelled the Owls to victory."
    authors: [ "bob@rice.edu" ],
    title: "Rice Band Deflated Footballs"
  }
]```

Lambdas (even in 1996)!
JavaScript objects are totally different

Java: we define classes, objects are *instances* of classes
JavaScript: no classes at all, just data

Version 1: We just return a new JavaScript object

```javascript
function makeArticle(bodyStr, authorsList, titleStr) {
    return {
        body: bodyStr,
        authors: authorsList,
        title: titleStr
    }
}
```

```javascript
makeArticle("No Way", ["Alice", "Bob"], "Dude")
```
JavaScript objects are totally different

Java: we define classes, objects are instances of classes
JavaScript: no classes at all, just data
Version 2: We can use the “new” keyword

```javascript
function Article(bodyStr, authorsList, titleStr) {
  this.body = bodyStr;
  this.authors = authorsList;
  this.title = titleStr;
}

new Article("No Way", ["Alice", "Bob"], "Dude")
Article{ body: "No Way", authors: [ "Alice", "Bob"], title: "Dude" }
```

This feels just like before, except JavaScript “knows” it’s an Article
Adding methods to an object

Version 0: just write functions that take objects as parameters

Version 1: stick lambdas inside the object

Version 2: write “prototypes” that apply to all objects of a type

```javascript
function Article(bodyStr, authorsList, titleStr) {
  this.body = bodyStr;
  this.authors = authorsList;
  this.title = titleStr;
}
Article.prototype.firstAuthor = function() { return this.authors[0] }
```

```javascript
q = new Article("No Way", ["Alice", "Bob"], "Dude")
». Article{ body: "No Way", authors: [ "Alice", "Bob"], title: "Dude" }
». q.firstAuthor()
». "Alice"
```
All objects are “open” to extension

Anybody can add a method to any type’s prototype

Lets you add “missing” functionality wherever you need it

But, also an opportunity for exciting new bugs...

```javascript
function Article(bodyStr, authorsList, titleStr) {
    this.body = bodyStr;
    this.authors = authorsList;
    this.title = titleStr;
}
Article.prototype.firstAuthor = function() { return this.authors[0] }
```

```javascript
q = new Article("No Way", ["Alice", "Bob"], "Dude")
Article{ body: "No Way", authors: [ "Alice", "Bob"], title: "Dude" }
q.firstAuthor()
"Alice"
```
Equality in Java is simple
Two references (or primitive types) are equal? \(a==b\)
Want to compare their internal meaning? \(a.equals(b)\)

In JavaScript, equality is insanely complicated
Two references (or primitive types) are equal? \(a===b\)
Two primitives types might be equal? \(a==b\)

<table>
<thead>
<tr>
<th>Operand A</th>
<th>Undefined</th>
<th>Null</th>
<th>Number</th>
<th>String</th>
<th>Boolean</th>
<th>Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>Undefined</td>
<td>true</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>Null</td>
<td>true</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>Number</td>
<td>false</td>
<td>false</td>
<td>A === B</td>
<td>A === ToNumber(B)</td>
<td>A === ToNumber(B)</td>
<td>A === ToPrimitive(B)</td>
</tr>
<tr>
<td>String</td>
<td>false</td>
<td>false</td>
<td>ToNumber(A) === B</td>
<td>A === B</td>
<td>ToNumber(A) === ToNumber(B)</td>
<td>A === ToPrimitive(B)</td>
</tr>
<tr>
<td>Boolean</td>
<td>false</td>
<td>false</td>
<td>ToNumber(A) === B</td>
<td>ToNumber(A) === ToNumber(B)</td>
<td>A === B</td>
<td>ToNumber(A) === ToPrimitive(B)</td>
</tr>
<tr>
<td>Object</td>
<td>false</td>
<td>false</td>
<td>ToPrimitive(A) === B</td>
<td>ToPrimitive(A) === ToNumber(B)</td>
<td>ToPrimitive(A) === ToNumber(B)</td>
<td>A === B</td>
</tr>
</tbody>
</table>
JavaScript “equality”

This causes all sorts of confusion!
“Undefined” - if you ask for a field that isn’t in an object
“null” - not the same thing as “undefined”

All numbers in JavaScript are “double”
+0 and -0 are different!
NaN (not-a-number)
Also + and -Infinity
JavaScript is sometimes absurd

Beautiful “Wat” talk by Gary Bernhardt

https://www.youtube.com/watch?v=20BySC_6HyY

Some absurdity with Ruby (yet another scripting language), mostly makes fun of JavaScript
Even if you’re not using a browser

JavaScript is *really* everywhere, even in Java
Java8 includes the “Nashorn” (German for rhinoceros) JavaScript engine

“jrunscript” from your shell starts it up

(doesn’t do all the nice arrow-key handling, although easy to fix)

http://stackoverflow.com/questions/22313797/java-8-nashorn-console-pretty-unusable
Easy to support JavaScript within Java

From Java: create a Nashorn “engine” and ask it to run things for you

```java
ScriptEngineManager factory = new ScriptEngineManager();
ScriptEngine engine = factory.getEngineByName("nashorn");
engine.eval("var x = {a: "Alice", b: "Bob"}"); // or use json.Builder!
String result = (String) engine.eval("x.a");
```

From JavaScript: interact easily with Java classes & objects

```javascript
var rpn = new edu.rice.rpn.RPNCalculator()
var sum = rpn.calc("3 4 5 6 + + +")
```
Why use an embedded JS engine?

1) Maybe there’s a JS library that’s really awesome and you need it.

2) Maybe you need something more general than your JSON input.

Example adventure game possibilities:
- Code to edit JSON before loading.
- Code to “procedurally” create mazes.
- Code to create “armies” of monsters, all slightly different.
- Code to simplify maze for test cases.

3) Executable config files = ultimate extensibility

Emacs text editor (elisp), \LaTeX documents (\LaTeX language), ...

4) Debugging

It’s immensely helpful to have a REPL to issue debug commands
(REPL = Read-Eval-Print Loop)

You might expose “scripting” to a user without them needing to change
(or even have source code) to your “core” application.
Why not use an embedded JS engine?

**Security.** Executable file formats are dangerous!
Nashorn can connect to any class, call any method: bad news for security!
   There’s a “ClassFilter” variant that lets you restrict JS-to-Java visibility.

**Performance.** Less of a big deal than it used to be.
Nashorn compiles to Java bytecode. Hits ~60% of “native” Java performance.
*Very* fast JS engines in the browsers (Google V8, Mozilla SpiderMonkey, ...)

**Bugs!** JavaScript doesn’t catch bugs as early as Java.
Not a problem for “small” things, but “small” things tend to grow bigger.
JavaScript is growing up. ‘use strict’ command eliminates many errors.
Lack of any type declarations helps you early-on, hurts you later.
Cross-language debugging? Painful.
See also, JSLint: https://github.com/douglascrockford/JSLint
JavaScript: What about type declarations?

You're expecting an “array”, you get something else. What do you do?

0) Add tons of error checking

typedFunction = function(paramsList, f){
    if (!(paramsList instanceof Array)) throw Error("...")
}

1) Put the types in comments (some IDEs will then do autocomplete)

2) There are (ugly) libraries for this (ArgueJS)

function range() {
    arguments = __({start: Number, stop: Number, step: Number})

    for(var i = arguments.start; i < arguments.stop; i += arguments.step)
        console.log(i);
}
JavaScript language variants

TypeScript, CoffeeScript, Flow, ...
Each language adds some features to JavaScript, then emits vanilla JavaScript

Flow (from Facebook)

```javascript
function foo(x: string, y: number): string {
  return x.length * y;
}

foo("Hello", 42);
```

TypeScript (from Microsoft)

```javascript
interface Person {
  firstname: string;
  lastname: string;
}

function greeter(person : Person) {
  return "Hello, " + person.firstname + " " + person.lastname;
}

var user = {firstname: "Jane", lastname: "User"};
```
JavaScript vs. Python

JavaScript is probably closer to Python than to Java
https://blog.glyphobet.net/essay/2557

If you know Python well, the biggest change is how objects work
How do “real” coders write JavaScript?

Most just write JavaScript as-is, or within giant JS frameworks
Web coding frameworks like AngularJS or JQuery do a lot of the work

Beyond TypeScript, Flow, etc., lots of other languages target JavaScript
Google’s Dart: more Java-like than JavaScript-like (e.g., has classes)
GWT (Google Web Toolkit): Java to JavaScript compiler
Ceylon, Scala, and Kotlin: can target the JVM or JavaScript
pyjamas: Python to JavaScript compiler

Even for language you might not have thought about
ClojureScript: Scheme/LISP-ish language to JavaScript
Whalesong: Racket (another Scheme dialect) to JavaScript
Haste: Haskell to JavaScript
How crazy does this get? Google Closure

Google built a compiler from JavaScript to JavaScript
https://developers.google.com/closure/

Among other things, Closure has:
A full-blown code optimizer (dead code elimination, etc.)
A full-blown type checker (catches possible type confusion)

Why did they do all of this?
Compression: why waste time/energy transmitting code that won’t run?
Bug detection: they really, really don’t want Gmail crashing in your browser
Even crazier? asm.js

asm.js: A subset of JavaScript, easy for “real” compilers to target
No dynamic memory allocation (big static arrays), no fancy method dispatch (dumb function calls), basically JavaScript as a “portable assembly language”

Takes advantage of newer JavaScript features like “typed arrays” (i.e., real integers)

emscripten: LLVM backend that targets JavaScript

SpiderMonkey (Firefox) JS engine can get within 20% of native C code
asm.js now also optimized in IE10 and Chrome 28+

Older JavaScript engines can run asm.js code without modification
If you want to learn more about JavaScript

Zillions of books out there
Some focus on the language (e.g., Crockford)
Some focus on libraries (e.g., node.js web server)
Most focus on web browsers
   Lecture coming soon... (Friday?)
JavaScript is evolving rapidly
   Nashorn supports v5.1
   Cool new stuff coming in v6/v7
http://es6-features.org
Live coding: playing with JavaScript