JavaScript: Introduction

Announcements

Week11 lab is now online

Week11 project is our “security experiment”

You’ll receive an email from us Wednesday afternoon / evening.

You will only need three hours (max) to participate in the experiment.

You’re graded on *participation*, not on *correctness*.

Please attend the Wednesday lecture, which is all about security!

Ancient JavaScript history!

Way back in 1995…

Web browsers had no way to run external code

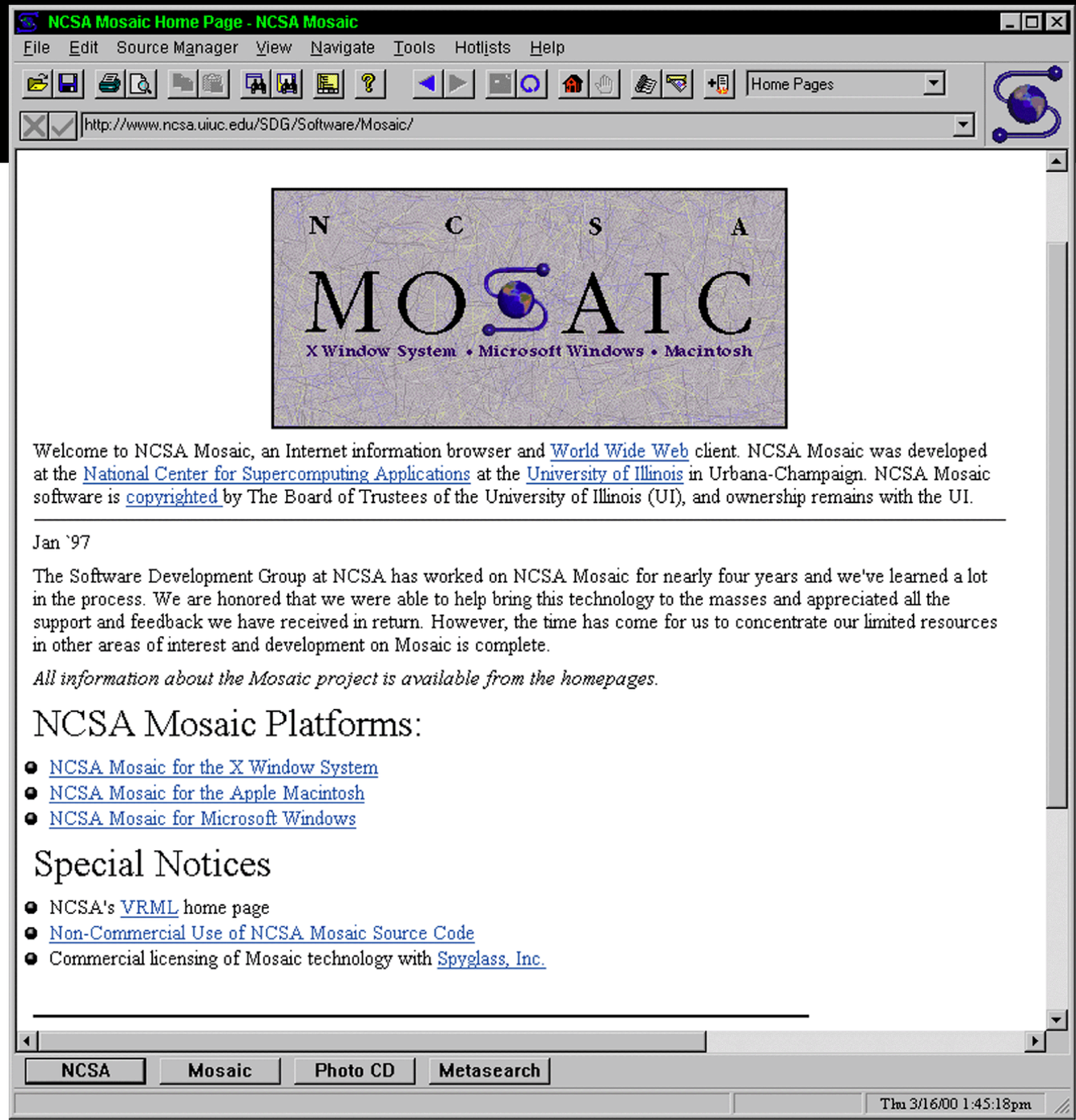
But there are tons of “mobile code systems” in the world

Tcl/Tk: Popular with Unix tool builders, still running today in many places

Java: Originally targeting television set-top boxes

Many others that never achieved commercial success

Meanwhile, inside Netscape…



Mocha ➜ LiveScript ➜ JavaScript

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 syntax

Easy to add small behaviors to web pages

One-liners: click a button, pop up a menu

M JavaScript

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/

Br **t:**

Li

Sa

Ea

Sc

Ea

Or

Why JavaScript in Comp[redacted]?

Every Comp[redacted] project is in Java; why should you know JavaScript?

1) It's the most widely available programming language in the world
   - In every browser, everywhere. It's many people's first language.

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 readable.

4) You'll might tweak the JavaScript to solve this week's project.

You already know big parts of JavaScript

Here's a JSON data structure:


```
{
  "authors": [
    {
      "name": "Alice Action",
      "email": "alice@[redacted]"
    },
    {
      "name": "Bob Bigbooté",
      "email": "bob@[redacted]"
    },
    {
      "name": "Charlie Chancery",
      "email": "charlie@[redacted]"
    },
    {
      "name": "Dorothy Danger",
      "email": "dorothy@[redacted]"
    },
    {
      "name": "Eve Ebullience",
      "email": "eve@[redacted]"
    }
  ],

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

You already know big parts of JavaScript

Here's a JavaScript program:

```
var x = {
  "authors": [
    {
      "name": "Alice Action",
      "email": "alice@[redacted]"
    },
    {
      "name": "Bob Bigbooté",
      "email": "bob@[redacted]"
    },
    {
      "name": "Charlie Chancery",
      "email": "charlie@[redacted]"
    },
    {
      "name": "Dorothy Danger",
      "email": "dorothy@[redacted]"
    },
    {
      "name": "Eve Ebullience",
      "email": "eve@[redacted]"
    }
  ],

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

You already know big parts of JavaScript

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

```
var x = {
  authors: [
    {
      name: "Alice Action",
      email: "alice@[redacted]"
    },
    {
      name: "Bob Bigbooté",
      email: "bob@[redacted]"
    },
    {
      name: "Charlie Chancery",
      email: "charlie@[redacted]"
    },
    {
      name: "Dorothy Danger",
      email: "dorothy@[redacted]"
    },
    {
      name: "Eve Ebullience",
      email: "eve@[redacted]"
    }
  ],

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

Lots of familiar built-in methods

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

Lots of familiar built-in methods

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

Easy access to fields inside objects

Lots of familiar built-in methods

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

“Arrays” also act like functional lists

Lots of familiar built-in methods

```
☚ 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("[redacted]") })
☛ [
 {
  body: "In a sports upset few could have imagined, the [redacted] defeated the New England
Patriots 34-7 in the Superbowl.",
  authors: [ "alice@[redacted]", "bob@[redacted]" ],
  title: "[redacted] Win Superbowl, Surprised NFL Fans React"
 },
 {
   body: "[redacted] bandmembers thought they were just making fun of Deflategate, but their prank
somehow propelled the [redacted] to victory."
   authors: [  "bob@[redacted].edu" ],
   title: "[redacted] 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

```
function makeArticle(bodyStr, authorsList, titleStr) {
    return {
        body: bodyStr,
        authors: authorsList,
        title: titleStr
    }
}
☚ makeArticle("No Way", ["Alice", "Bob"], "Dude")
☛ { body: "No Way", authors: [ "Alice", "Bob"], title: "Dude" }
```

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

```
function makeArticle(bodyStr, authorsList, titleStr) {
    return {
        body: bodyStr,
        authors: authorsList,
        title: titleStr
    }
}
👉 makeArticle("No Way", ["Alice", "Bob"], "Dude")
👈 { body: "No Way", authors: [ "Alice", "Bob"], title: "Dude" }
```

You can pass and return JavaScript objects anywhere a *value* might go.

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

Arguments don't have type declarations!

```
function makeArticle(bodyStr, authorsList, titleStr) {
    return {
        body: bodyStr,
        authors: authorsList,
        title: titleStr
    }
}
☚ makeArticle("No Way", ["Alice", "Bob"], "Dude")
☛ { body: "No Way", authors: [ "Alice", "Bob"], title: "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

```
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 like before, but 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

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

👈 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 a type’s prototype

Lets you add “missing” functionality wherever you need it

Also an opportunity for exciting new bugs…

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

👈 q = new Article("No Way", ["Alice", "Bob"], "Dude")
👉 Article{ body: "No Way", authors: [ "Alice", "Bob"], title: "Dude" }
👈 q.firstAuthor()
👉 "Alice"
```

JavaScript's dark corners: "equality"

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 primitive types might be equal? `a==b`

JavaScript's dark corners: "equality"

How do I compare A to B?

		Operand B					
		Undefined	Null	Number	String	Boolean	Object
Operand A	Undefined	`true`	`true`	`false`	`false`	`false`	`false`
	Null	`true`	`true`	`false`	`false`	`false`	`false`
	Number	`false`	`false`	`A === B`	`A === ToNumber(B)`	`A === ToNumber(B)`	`A == ToPrimitive(B)`
	String	`false`	`false`	`ToNumber(A) === B`	`A === B`	`ToNumber(A) === ToNumber(B)`	`A == ToPrimitive(B)`
	Boolean	`false`	`false`	`ToNumber(A) === B`	`ToNumber(A) === ToNumber(B)`	`A === B`	`ToNumber(A) == ToPrimitive(B)`
	Object	`false`	`false`	`ToPrimitive(A) == B`	`ToPrimitive(A) == B`	`ToPrimitive(A) == ToNumber(B)`	`A === B`

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 (?!)

All numbers in JavaScript are “double”

+0 and -0 are different!

NaN (not-a-number)

Also + and -Infinity

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 (?!)

All numbers in JavaScript are “double”

+0 and -0 are different!

NaN (not-a-number)

Also + and -Infinity

x	y	==	===	Object.is
undefined	undefined	true	true	true
null	null	true	true	true
true	true	true	true	true
false	false	true	true	true
"foo"	"foo"	true	true	true
{ foo: "bar" }	x	true	true	true
0	0	true	true	true
+0	-0	true	true	false
0	false	true	false	false
""	false	true	false	false
""	0	true	false	false
"0"	0	true	false	false
"17"	17	true	false	false
[1,2]	"1,2"	true	false	false
new String("foo")	"foo"	true	false	false
null	undefined	true	false	false
null	false	false	false	false
undefined	false	false	false	false
{ foo: "bar" }	{ foo: "bar" }	false	false	false
new String("foo")	new String("foo")	false	false	false
0	null	false	false	false
0	NaN	false	false	false
"foo"	NaN	false	false	false
NaN	NaN	false	false	true

JavaScript is sometimes absurd

Beautiful “Wat” talk by Gary Bernhardt

https://www.destroyallsoftware.com/talks/wat

Some absurdity with Ruby (yet another scripting language), mostly makes fun of JavaScript

Why is JavaScript so weird?

Some of the same thinking went into Python: “duck typing”

If you try to add a string to a number, JavaScript will try to help you

- “I know how to turn a number into a string, and then I can concatenate them for you!”
- Technical term: *automatic type coercion*

Java considers this an error (coercion must be manual)

Open question: are we doing any favors with “weird” features?

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, but easy to fix)
- http://stackoverflow.com/questions/22313797/java-8-nashorn-console-pretty-unusable

Useful online tutorial:

http://winterbe.com/posts/2014/04/05/java8-nashorn-tutorial/

You can call JavaScript from Java

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

```
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");
```

All your Java objects are visible from JavaScript

```
nashorn> var rpn = new edu.[redacted].rpn.RPNCalculator()
nashorn> rpn.calc("2 5 +")
7.0
```

Why might I want JavaScript + Java?

High-level debugger: look at your objects, call their methods
Call Java functions, even when your program is running!

Java9 will do this sort of thing directly (no need for JavaScript)
“Read-Eval-Print-Loop” (REPL) is a powerful tool, coding paradigm.
You had this with Python as well.

Java11 is “deprecating” Nashorn
You can integrate other JS engines with Java if you really need it.
V8 JavaScript engine: https://github.com/eclipsesource/J2V8

Why use an embedded JS engine?

1) Maybe there's a JS library that's helpful for you

2) Maybe you want to have code in your JSON input

Example possibilities:

- Code to edit JSON before loading
- Code to "procedurally" generate your input
- Code to create "armies" of monsters, all slightly different

3) Extensibility for other programmers

Emacs text editor (elisp), Adobe Lightroom (Lua), etc.

Others can add features to your system without needing your source code!

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 to restrict JS-to-Java visibility.

Performance. Less of a big deal than it used to be.

Nashorn compiles to Java bytecode. (~60% of native speed)

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.

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 (here: ArgueJS)

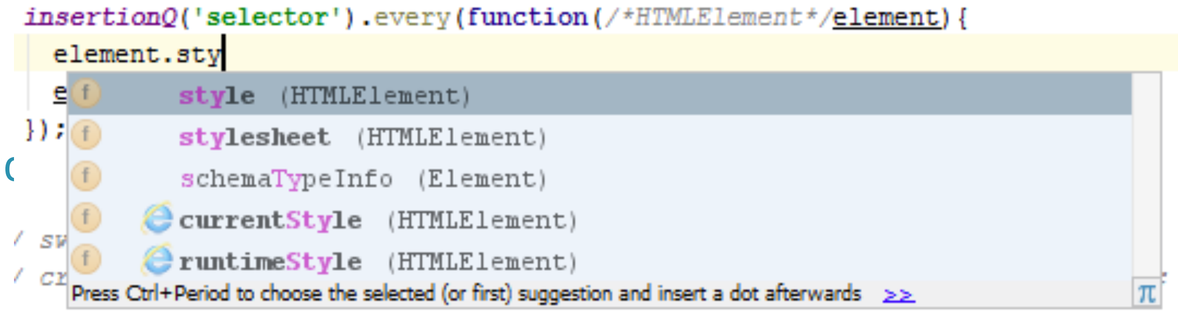
```
function range( ) {
  arguments = __({start: Number, stop: Number, step: Number})

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

JavaScript: 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 Erro
```



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

2) There are (ugly) libraries for this (here: 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)

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

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

TypeScript (from Microsoft)

```
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

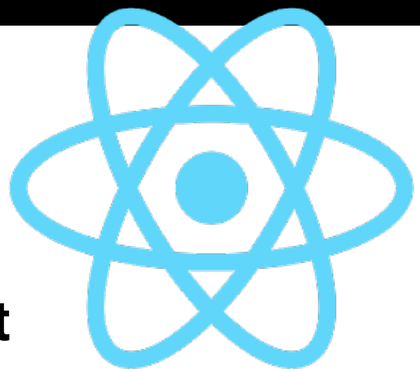
JavaScript is probably closer to Python than Java

http://stackoverflow.com/questions/1786522/how-different-are-the-semantics-between-python-and-javascript

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 libraries like AngularJS, JQuery, React do a lot of the work



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

Even for languages you might not have considered
ClojureScript: Scheme/LISP-ish language, targets JavaScript
Whalesong: Racket (Scheme-ish) to JavaScript
Haste: Haskell to JavaScript
Pyret: Python-ish teaching language, targets JavaScript

How crazy does this get? Google Closure

Google built a compiler from JavaScript to JavaScript

Among other things, Closure has:

A code optimizer (dead code elimination, etc.)

A type checker (catches possible type confusion)

Why did they do all this?

Compression: why send code that won't run?

Bug detection: they *really, really* don't want Gmail to crash

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”

- Uses newer JavaScript features like “typed arrays” (i.e., real integers)

LLVM backend that targets JavaScript

SpiderMonkey (Firefox) JS engine can get within 20% of native C code

asm.js optimizations also supported now in MS Edge, Chrome

Older JavaScript engines can run asm.js code without modification (slowly)

Learn more about JavaScript

Zillions of books out there

Some focus on the language (Crockford)

Some focus on libraries (e.g., node.js web server)

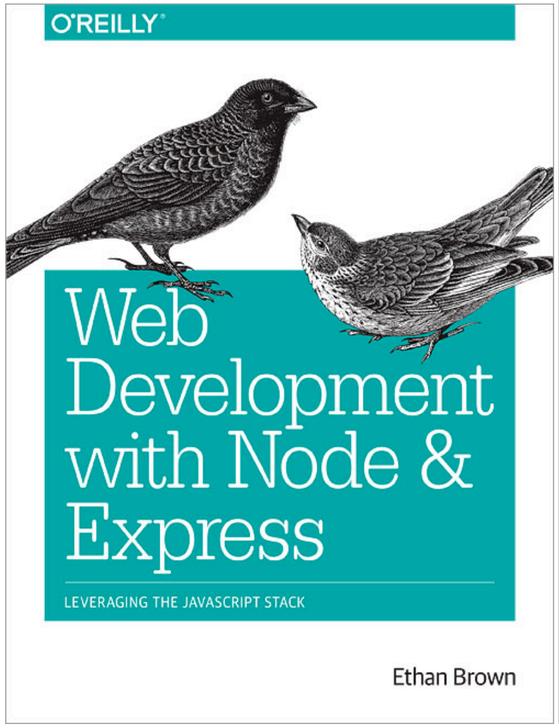
Most focus on web browsers

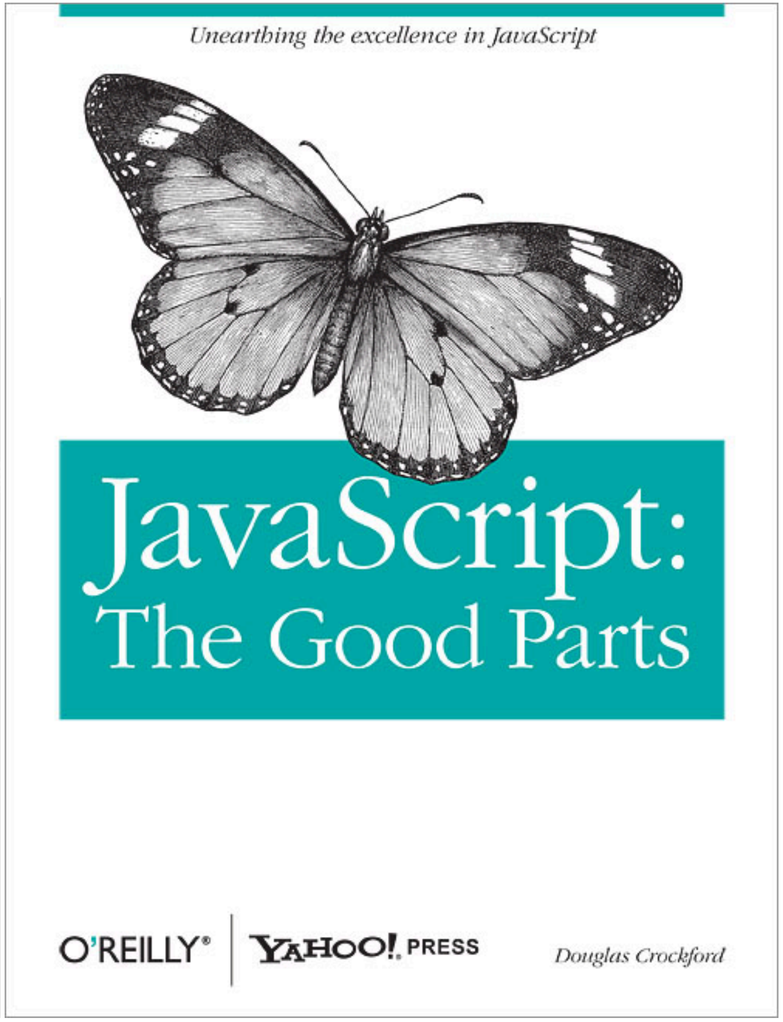
JavaScript is evolving rapidly

Nashorn supports v5.1

Cool new stuff coming in v6/v7

http://es6-features.org





Live coding: playing with JavaScript