Why Best Practices?

JavaScript has been secondary

JavaScript is taking over everything

JavaScript is different...

Don’t work in a vacuum.

Don’t believe everything you read.

It’s dangerous to go alone...

Take knowledge with you

Understanding Syntax

```
var x = 10;
var y = "10";
if (x == y) {
    //this is true...
}
```

◀ Why is this true??

```
function myModule()
{
    return
    {
        x: 3,
        y: 4
    }
}
```

- Why won't this work?

Prevent the .01%

“What is ‘this’?”

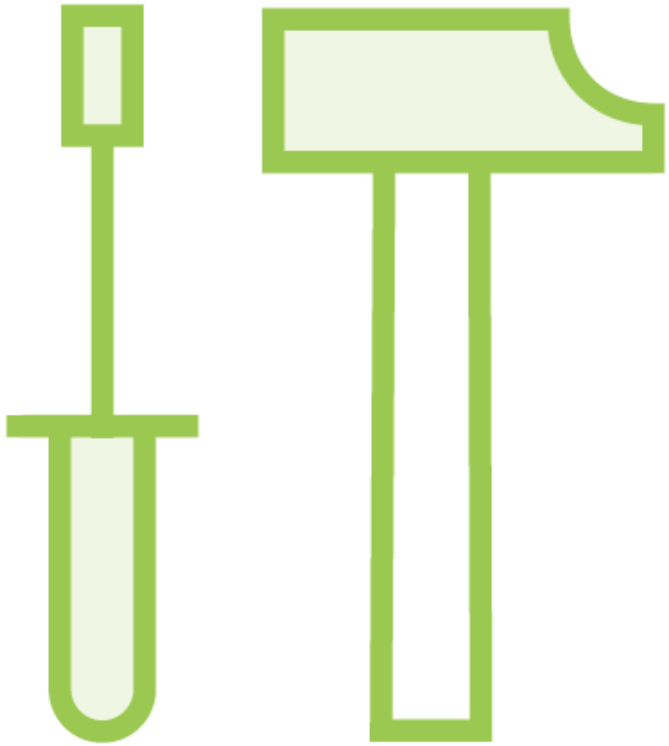
-Every JavaScript developer

“What was I thinking?”

-Every developer ever...

“That code is all wrong…”

Picking Your Packages



Tooling up

Automation is your friend

Rules of the road

Talk about why best practices

You decide

Semicolons

”Semicolons are optional in JavaScript”

Lot of people...

“Certain ECMAScript statements (…) must be terminated with semicolons.”

EcmaScript Standards

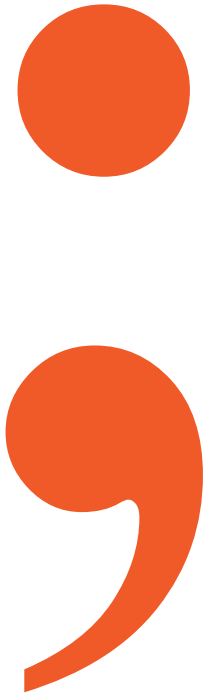
“For convenience, however, such semicolons may be omitted from the source text in certain situations.”

EcmaScript Standards

“These situations are described by saying that semicolons are automatically inserted....”

EcmaScript Standards

We need to understand how that works....



Three Rules...

“When, as a Script or Module is parsed from left to right, a token (called the offending token) is encountered that is not allowed by any production of the grammar, “

EcmaScript Standards

```
var a = 12

var b = 13


if(a){console.log(a)}


console.log(a+b)
```

- The offending token is separated from the previous token by at least one LineTerminator.

EcmaScript Standards

```
var a = 12;

var b = 13


if(a){console.log(a)}


console.log(a+b)
```

◀ Rule 1 a

```
var a = 12;
var b = 13;

if(a){console.log(a)}

console.log(a+b)
```

- Rule 1 a
- Rule 1 a

- The offending token is }

EcmaScript Standards

```
var a = 12;
var b = 13;

if(a){console.log(a);}

console.log(a+b)
```

- Rule 1 a
- Rule 1 a

- Rule 1 b

“When, as the Script or Module is parsed from left to right, the end of the input stream of tokens is encountered, then a semicolon is automatically inserted at the end of the input stream.“

EcmaScript Standards

```
var a = 12;
var b = 13;

if(a){console.log(a);}

console.log(a+b)
```

- Rule 1 a
- Rule 1 a

- Rule 1 b

```
var a = 12;
var b = 13;

if(a){console.log(a);}

console.log(a+b);
```

◀ Rule 1 a

◀ Rule 1 a

◀ Rule 1 b

◀ Rule 2

“When, as a Script or Module is parsed from left to right, a token (called the offending token) is encountered that is *not allowed* by any production of the grammar, “

EcmaScript Standards

```
var a = 12

var b = 13

var c = b + a


['menu', 'items', 'listed']

    .forEach(function (element)
    {

        console.log(element)

    })
```

◀ Rule 1 a

◀ Rule 1 a

◀ Rule 1 b

```
var a = 12

var b = 13

var c = b + a


(function(){

    console.log('inside my
iife');

    console.log('doing secret
                stuff...');

}())
```

“When, a token is encountered that is allowed by some production of the grammar, *but the production is a restricted production* and the token would be the first token of a restricted production, and the restricted token is separated from the previous token by *at least one LineTerminator*, then a *semicolon is automatically inserted before the restricted token*.“

EcmaScript Standards

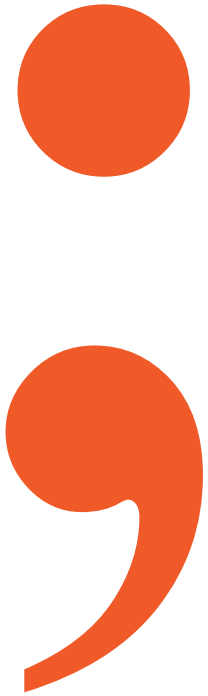
Restricted Production

continue, break, return, or throw....

```
function returnObject()
{
    if(someTrueThing)
    {
        return
        {
            hi: 'hello'
        }
    }
}
```

“Use semicolons in conjunction with JSHint (or ESLint) to prevent potential issues”

Jonathan Mills



1. Consistency with other languages
2. Prevents the .01% issues...

Linting

Linting Code

A linter scans your code to detect potential problems and errors.

Linting Code

A linter scans your code to detect potential problems and errors.

JS Lint

Created by Douglas Crockford in 2002

Preconfigured

Not very configurable...

JS Hint

Fork of JSLint

Much more configurable

Built in package support

Not extensible...

ESLint

The most recent

Custom rules support

Lots of configuration

JS Hint

Getting Started

In the browser.

In your editor.

In the command Line.

With a build tool.

Curly Braces

```
function service()
{
    var get = function()
    {
        console.log('get');
    }
    var set = function()
    {
        console.log('set');
    }
    return
    {
        get: get,
        set: set
    }
}
```

```
function service()
{
    var get = function()
    {
        console.log('get');
    }
    var set = function()
    {
        console.log('set');
    }
    return {
        get: get,
        set: set
    }
}
```

```
function service(){
    var get = function() {
        console.log('get');
    }
    var set = function() {
        console.log('set');
    }
    return {
        get: get,
        set: set
    }
}
```

Equality

How do I compare things?

==

If variables are two different types, it will convert them to the same type...

There will be no type conversion...

Use === as the default

The see if a var exists, use typeof undefined

Variables

Hoisting

Hoisting is JavaScript's default behavior of moving all declarations to the top of the current scope.

“A var statement declares variables that are scoped to the running execution context’s VariableEnvironment. Var variables are created when their containing Lexical Environment is instantiated and are initialized to undefined when created.”

EcmaScript Standards

“A var statement declares variables that are scoped to the running execution context’s VariableEnvironment. Var *variables are created when their containing Lexical Environment is instantiated* and are initialized to undefined when created.”

EcmaScript Standards

“A var statement declares variables that are scoped to the running execution context’s VariableEnvironment. Var variables are created when their containing Lexical Environment is instantiated and are *initialized to undefined when created.*”

EcmaScript Standards

```
console.log(myVariable);

var myVariable = 10;
```

Variables:

```
console.log(myVariable);

var myVariable = 10;
```

Variables:
myVariable = undefined;

```
console.log(myVariable);

var myVariable = 10;
```

Variables:
myVariable = 10;

```
var myVariable = 10;

function func(){

  myVariable = 25;

  var myVariable;

}

func();

console.log(myVariable);
```

Variables: Global
none

Variables: func
none

```
var myVariable = 10;

function func(){

  myVariable = 25;

  var myVariable;

}

func();

console.log(myVariable);
```

Variables: Global
myVariable = 10

Variables: func
none

```
var myVariable = 10;

function func(){

  myVariable = 25;

  var myVariable;

}

func();

console.log(myVariable);
```

Variables: Global
    myVariable = 10

Variables: func
    myVariable = undefined

```
var myVariable = 10;

function func(){

  myVariable = 25;

  var myVariable;

}

func();

console.log(myVariable);
```

Variables: Global
myVariable = 10

Variables: func
myVariable = 25

```
var myVariable = 10;

function func(){

  var myVariable;

  myVariable = 25;

 }

func();

console.log(myVariable);
```

All var declarations go to the top of your scope!

Functions

Functions...

Declarations

Expressions

```
var myVariable = 10;

function func(){

  var myVariable;

  myVariable = 25;

 }

func();

console.log(myVariable);
```

All var declarations go to the top of your scope!

Summary

- Syntax in Javascript
- Semicolons...
- Linting
- Equality
- Variables
- Functions

Consistency is key

Global Variables

Strict Mode

“JavaScript is trying to help… Don’t let it.”

-Me

Read Only Properties

Deleting Stuff

Dupes

Number Types

The ‘with’ Statement

“with violates lexical scope, making program analysis (e.g. for security) hard to infeasible.”

This

JavaScript Behaviors

Strict Mode

Clean Errors instead of hiding problems

Callbacks

Promises

Using ES6 and Babel

Async / Await

Async Patterns

Callbacks

Named Functions

Promises

Babel

Async / Await in ES7

Production Code

Jonathan Mills

@jonathanfmills www.jonathanfmills.com

Introduction

Node.js Best Practices

Npm

Environment

Cross Platform Gotchas

NPM

Environment

Cross Platform

Simplify Your Environment

JavaScript Best Practices

Jonathan Mills

@jonathanfmills www.jonathanfmills.com