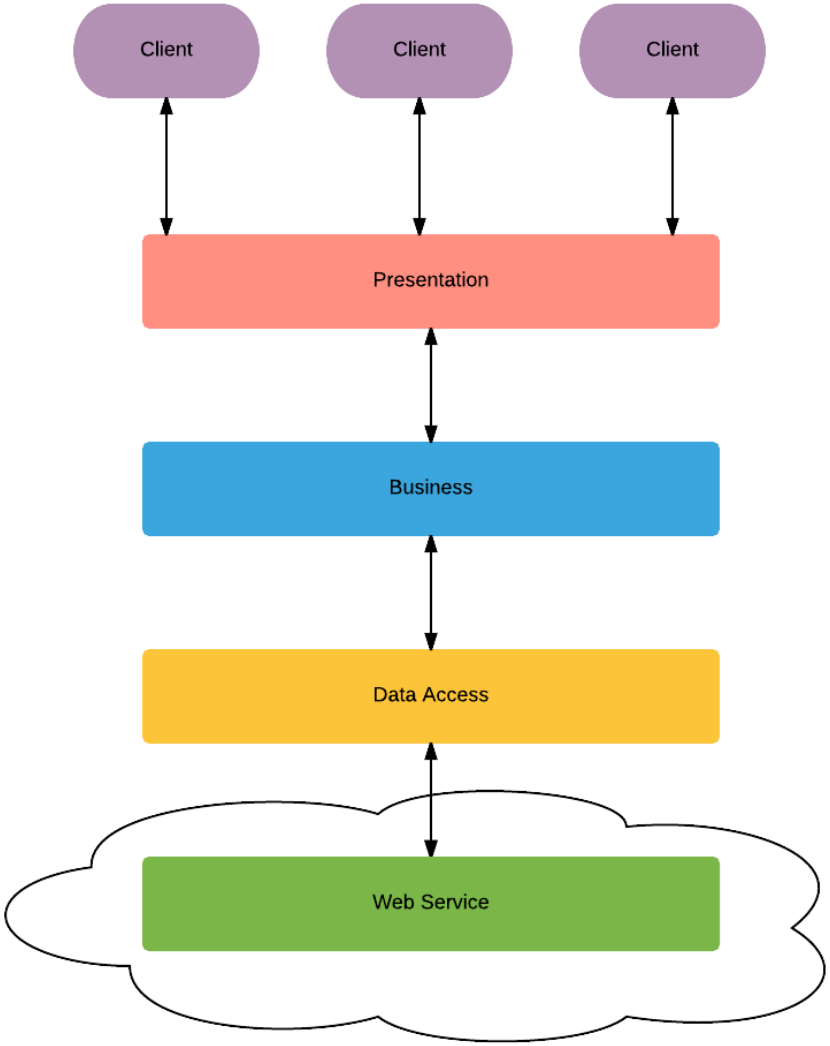
RAWDATA
Section 3

Troels Andreasen & Henrik Bulskov

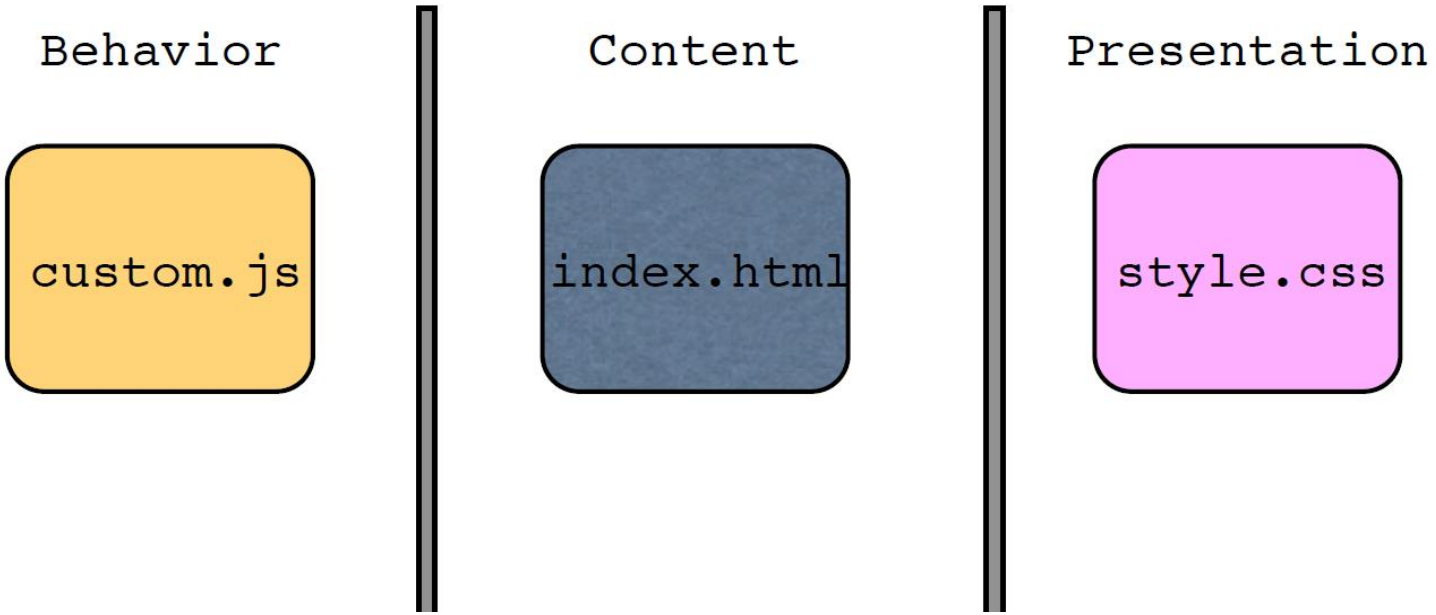
What to do in section 4?

- JavaScript
  - Functions
  - JQuery
- Single Page Applications
  - Databinding
  - Modularity
- Responsiveness
  - Adaptive applications
  - Bootstrap

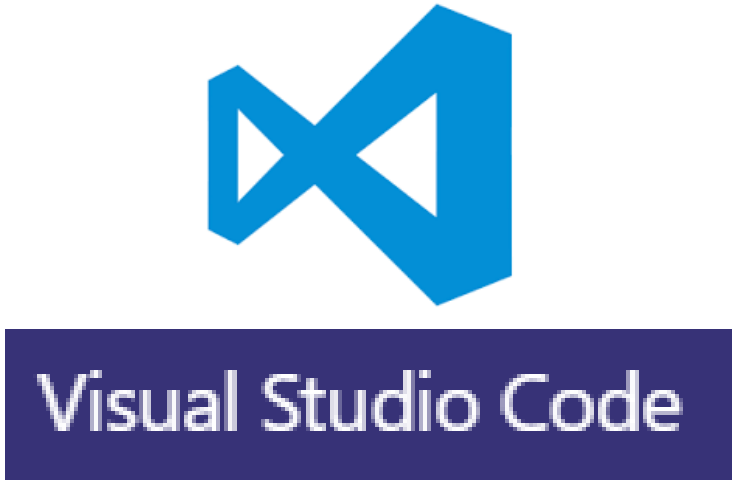
System Development



Unobtrusive Design



Development Tools



What we'll dig into…

JS Environments

- Mostly browsers
- Node.js (https://nodejs.org/en/)
- https://repl.it/languages/javascript
- https://jsfiddle.net/

JavaScript

- JavaScript is a high-level, dynamic, untyped, and interpreted programming language.

Source: Wikipedia

Comparing C# and JS

C#	JavaScript
• Strongly-Typed	• Loosely-typed
• Static	• Dynamic
• Classical Inheritance	• Prototypal
• Classes	• Functions
• Constructors	• Functions
• Methods	• Functions

http://www.crockford.com/javascript/inheritance.html

Comparing C# and JS

C#	JavaScript ES6
• Strongly-Typed	• Loosely-typed
• Static	• Dynamic
• Classical Inheritance	• Prototypal
• Classes	• Functions Classes
• Constructors	• Functions Constructors
• Methods	• Functions Array-fn

Types in JS

- Number
- String
- Boolean
- Symbol (new in ES6)
- Object
  - Function
  - Array
  - Date
  - RegExp
- null
- undefined

The Basics

- **Strings**: textual content. wrapped in quotation marks (single or double).
  - 'hello, my name is Karl'
  - "hello, my name is Karl"
- **Numbers**: integer (2) or floating point (2.4) or octal (012) or hexadecimal (0xff) or exponent literal (1e+2)
- **Booleans**: true or false

The Basics

- **Arrays**: simple lists. *indexed* starting with 0
  - ['Karl', 'Sara', 'Ben', 'Lucia']
  - ['Karl', 2, 55]
  - [['Karl', 'Sara'], ['Ben', 'Lucia']]
- **Objects**: lists of key-value pairs
  - {firstName: 'Karl', lastName: 'Swedberg'}
  - {parents: ['Karl', 'Sara'], kids: ['Ben', 'Lucia']}

Variables

- Always declare your variables!
- If you don't, they will be placed in the **global scope**
  - **bad:** myName = 'Karl';
  - **good: var** myName = 'Karl';
  - **still good: var** myName = 'Karl'; // more stuff
    myName = 'Joe';
- ”use strict”;

Variables ES6

- Let keyword
  - Introduce block scope

Operators

- All the ones you know… plus

=== and !==

Objects

- JavaScript objects can be thought of as simple collections of name-value pairs
- Create an empty object:

```
var obj = new Object();
var obj = {};
```

- Object literal syntax:

```
var obj = {
  name: "Carrot",
  "for": "Max",
  details: {
    color: "orange",
    size: 12
  }
}
```

Object Construction

```
function Person(name, age) {
    this.name = name;
    this.age = age;
}

// Define an object
var p = new Person("Peter", 21);
```

Object Prototype

```
function Person(firstName, lastName) {
    this.firstName = firstName;
    this.lastName = lastName;
    this.getFullName = function () {
        return this.firstName + " "
        + this.lastName;
    };
}

var p1 = new Person("Bruce", "Willis");
var p2 = new Person("Chuck", "Norris");
```

Object Prototype

```
function Person(first, last) {
    this.firstName = first;
    this.lastName = last;
}

Person.prototype.fullName = function () {
    return this.firstName + ' ' + this.lastName;
};
```

Functions

- Looks like the ones you now, but it’s not…
- Functions parameters
- Return values
- Objects
- “this”
- Closures and scope

Scope

- JS has function scope!!!

- Except if you use the ES6 let keyword – so do that!!!

Functions

- Functions parameters
  - A function can take 0 or more named parameters
- Return values
  - Will always return something
  - If you don’t return something, undefined is returned

The Basics: Functions

- Functions allow you to **define** a block of code, name that block, and then **call** it later as many times as you want.

- **function** myFunction( ) { /* code goes here */ }

- **myFunction**( ) // calling the function *myFunction*

The arguments Object

- Every function has an arguments object a collection of the arguments passed to the function when it is called
- an "array-like object" in that it is indexed and has a **length** property but can't attach array methods to it
- can be looped through
- allows for variable number of arguments

The arguments Object

```
function sum() {
    var result = 0;
    for (var i = 0; i < arguments.length; i++) {
        result += arguments[i];
    }
    return result;
}
```

Named vs. Anonymous Functions

- Named:
  - function **foo**() { } // function **declaration**
  - var foo = function **foo**() { }; // function **expression**
- Anonymous:
  - var foo = function() { }; // function **expression**

Function Closure

```
function makeAdder(a) {
  return function(b) {
    return a + b;
  };
}
var x = makeAdder(5);
var y = makeAdder(20);
x(6); // ?
y(7); // ?
```

ES6 Functions

- Default values

```
var getProduct = function(productId = 1000) {
    console.log(productId);
};
getProduct();
```

- Rest and spread operator
  - rest
  - spread

```
function (productId, ...categories) {
```

```
var prices = [12, 20, 18];
var maxPrice = Math.max(...prices);
```

- Arrow functions

```
var getPrice = () => 5.99;
```

```
document.addEventListener('click', () => console.log(this));
```

Patterns

- Spaghetti and Ravioli
  - Separation Patterns
  - Avoiding Globals

- Object Literals
- Module Pattern
  - Anonymous Closures
  - Private/Public Members
  - Immediate Invocation

- Revealing Module Pattern
  - Refinements to Module Pattern

Problems with Spaghetti Code

- Mixing of Concerns
- No clear separation of functionality or purpose
- Variables/functions added into global scope
- Potential for duplicate function names
- Not modular
- Not easy to maintain
- No sense of a “container”

Some Examples of Spaghetti Code with JavaScript

- Script all over the page
- Objects are extended in many places in no discernible pattern
- Everything is a global function
- Functions are called in odd places
- Everything is a global
- Heavy JavaScript logic inside HTML attributes
  - Obtrusive JavaScript
  - http://en.wikipedia.org/wiki/Unobtrusive_JavaScript

Namespaces

- Encapsulate your code under a namespace
- Avoid collisions
- First and easy step towards good design

```
var my = my || {};
```

```
my.sum = function () {
    var sum = 0;
    for (var i = 0; i < arguments.length; i++) {
        sum += parseInt(arguments[i], 10);
    }
    return sum;
}
```

Object Literals

- Benefits
- Quick and easy
- All members are available
- Challenges
  - “this” problems
  - Best suited for simple view models and data

```
my.model = {
    firstName: "Peter",
    lastName: "Smith",
    fullName: function() {
        return this.firstName + " " + this.lastName;
    }
}
```

Module Pattern

- Anonymous Closures
  - Functions for encapsulation
- Immediate function invocation
- Private and public members

Anonymous Closure

- Function expression instead of function definition
  - Wrapped in parentheses
- Scoped
  - All vars and functions are enclosed

```
(function () {
    var x = 5;
    //...
})();
```

Immediate Function Invocation

- Create a module
- Immediately available

```
my.model = (function () {
    var sep = " ";
    return {
        firstName: "Peter",
        lastName: "Smith",
        fullName: function() {
            return this.firstName + sep + this.lastName;
        }
    }
})();
```

Invorke immediately

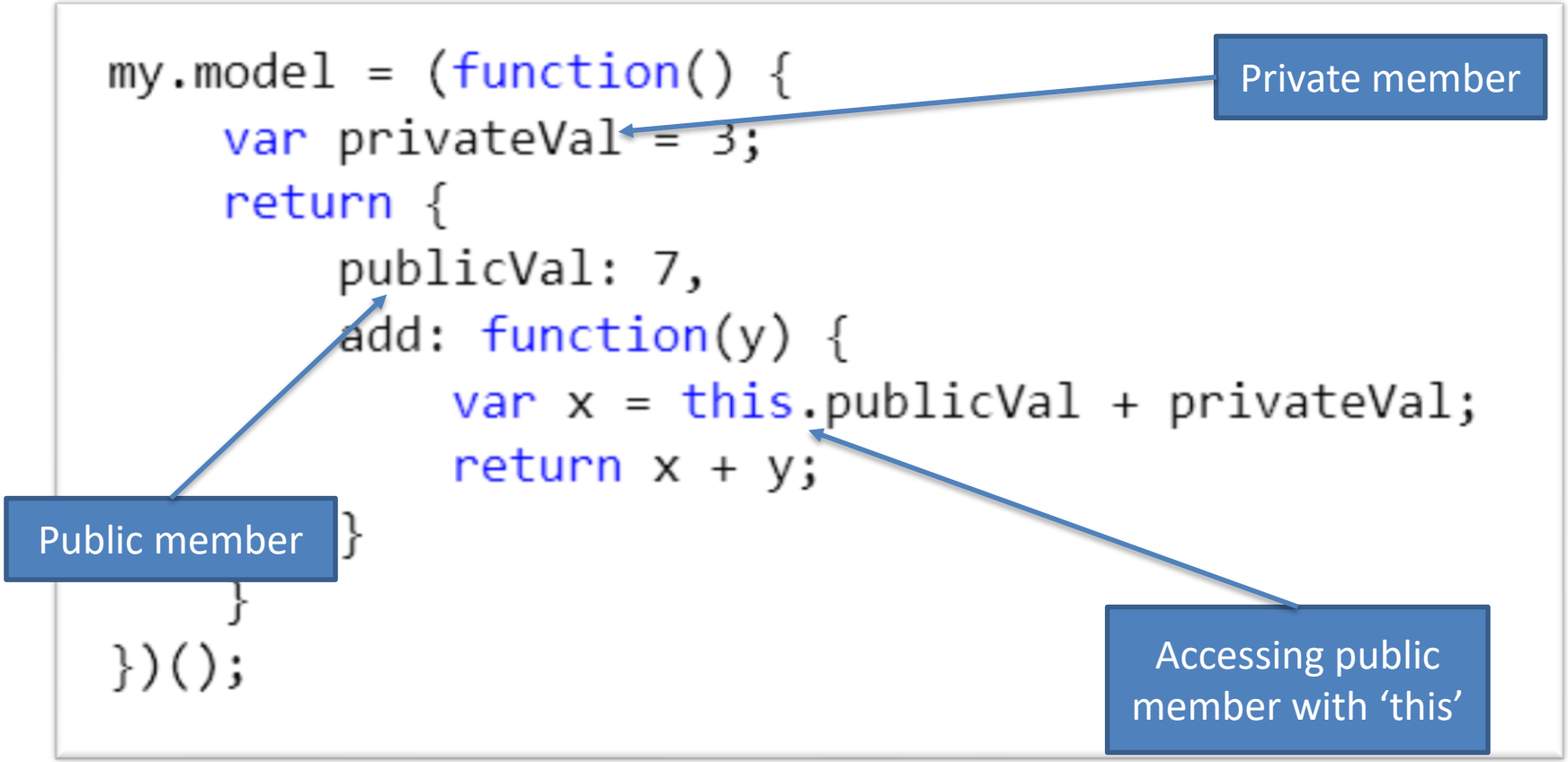
The Module Pattern

- Benefits:
  - Modularize code into re-useable objects
  - Variables/functions taken out of global namespace
  - Expose only public members
  - Hide plumbing code
- Challenges:
  - Access public and private members differently

Private/Public Members

```
my.model = (function() {
    var privateVal = 3;
    return {
        publicVal: 7,
        add: function(y) {
            var x = this.publicVal + privateVal;
            return x + y;
        }
    }
})();
```

Private/Public Members



The Revealing Module Pattern

- All the Benefits of the Module Patterns +
  - Makes it clear what is public vs private
  - Helps clarify "this"
  - Reveal private functions with different names

Revealing

```
my.model = (function () {

    var privateVal = 3;
    var add = function (y) {
        return privateVal + y;
    }

    return {
        someVal: privateVal,
        add: add
    }
})();
```

Private members

Public members

Global Object

- In the browser environment, the global object is **window.** It collects all functions and variables that are global in scope.
- Usually implied.
- Comes with some useful properties and methods:
  - location
  - parseInt(); parseFloat()
  - isNaN()
  - encodeURI(); decodeURI()
  - setTimeout(); clearTimeout()
  - setInterval(); clearInterval()

JavaScript and DOM

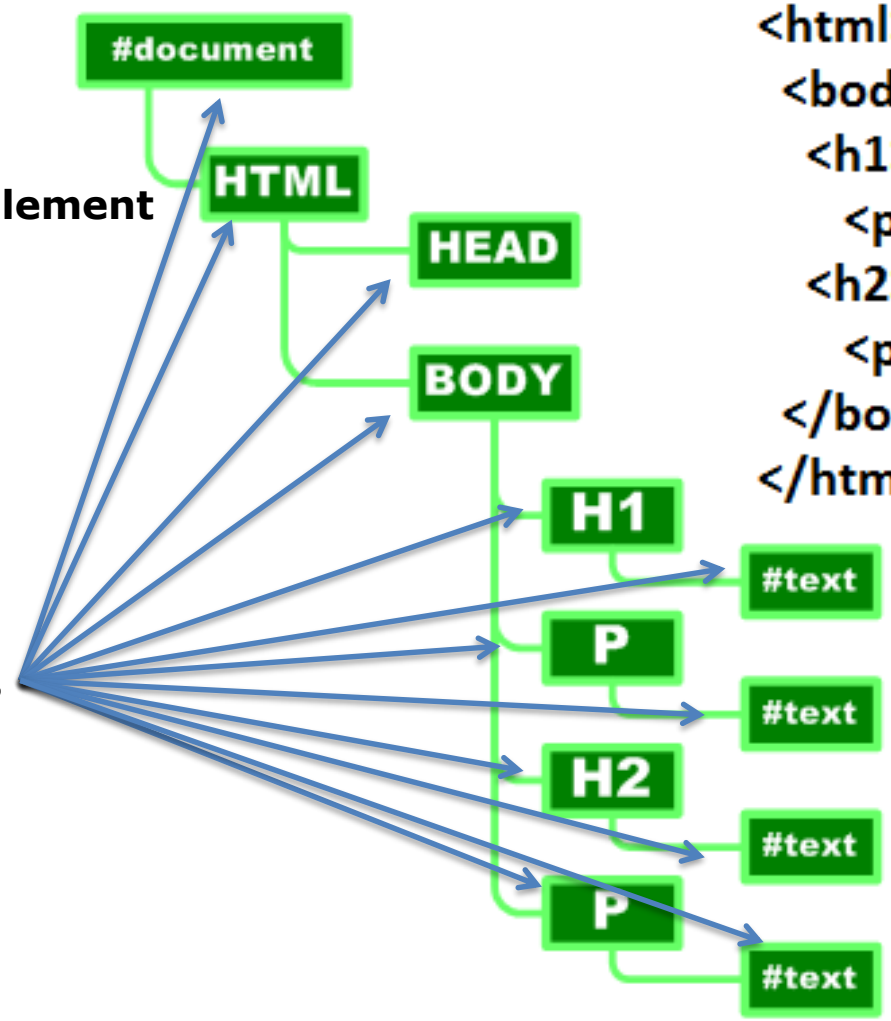
- JavaScript understands HTML and can directly access it.
- JavaScript uses the HTML Document Object Model to manipulate HTML.
- The DOM is a hierarchy of HTML things.
- Use the DOM to build an “address” to refer to HTML elements in a web page.
- Levels of the DOM are dot-separated in the syntax.

Document Tree Structure

document

document.documentElement

document.body



Dom nodes

```
<html>
 <body>
  <h1>Heading 1</h1>
    <p>Some text</p>
  <h2>Heading 1</h2>
    <p>Some text</p>
 </body>
</html>
```

DOM nodes

In a DOM tree, almost everything you'll come across is a node.

- Every element is at its most basic level a node in the DOM tree.
- Every attribute is a node.
- Every piece of text is a node.
- Comments
- Special characters (like © a copyright symbol)
- DOCTYPE declaration

All are nodes

jQuery

- jQuery is a fast, small, and feature-rich JavaScript library. It makes things like HTML document traversal and manipulation, event handling, animation, and Ajax much simpler with an easy-to-use API that works across a multitude of browsers.

A few of jQuery's Benefits

- Lets you move quickly from beginner to advanced
- Improves developer efficiency
- Excellent documentation
- Unobtrusive from the ground up
- Reduces browser inconsistencies
- At its core, a simple concept

CSS Selectors

- element {}
- #id {}
- .class {}
- **selector1**, **selector2** {}
- ancestor **descendant** {}
- parent > **child** {}
- :nth-child() {}

CSS Selectors (in jQuery)

- $('element')
- $('#id')
- $('.class')
- $('**selector1**, **selector2**')
- $('ancestor **descendant**')
- $('parent > **child**')
- $(':nth-child(n)')

CSS Attribute Selectors

- $('input[name=firstname\\[\\]]')
- $('[**title**]') has the attribute
- $('[attr**=**"val"]') attr equals val
- $('[attr**!=**"val"]') attr does not equal val
- $('[attr**~=**"val"]') attr has val as one of space-sep. vals
- $('[attr**^=**"val"]') attr begins with val
- $('[attr**$=**"val"]') attr ends with val
- $('[attr***=**"val"]') attr has val anywhere within

Including jQuery

```
<script src="//code.jquery.com/jquery-1.11.3.min.js"></script>
<script src="//code.jquery.com/jquery-migrate-1.2.1.min.js"></script>
```

DOM Element Selection

```
var elem = document.getElementById("myelem");
elem.innerHTML = "Blah blah";
```

```
var elem = $("#myelem");
elem.text("blah blah");
```

jQuery Selectors

- Simple selectors

  $('div') $('.myclass')
- Multiple elements

  $('div, p, li')
- Numeric selection

  $('li:eq(5)') $('li:first') $('li:nth-child(2)')
- Stacking selector filters

  a[title = "jQuery"][href ^= "http://"]

Events

- blur
- focus
- load
- resize
- scroll
- unload
- beforeunload
- click
- dblclick
- mousedown
- mouseup
- mousemove
- mouseover
- mouseout
- change
- select
- submit
- keydown
- keypress
- keyup
- error

Event Binding

```
document.getElementById('button').addEventListener('click', function() {
    var val = document.getElementById('in').value;
    document.getElementById('out').innerText = val;
}
```

```
$('#button').on('click', function () {
    $('#out').html($('#in').val());
});
```

Modular Programming

- Moving from spaghetti to ravioli coding will create a number of modules
- You need to add them in the right order in your HTML file
- One solution is to use asynchronous module definition (AMD), i.e. load the files when they are needed
- ES6 introduce a module system
  - But it is not supported by browsers yet, so to use it transpilation is necessary

require.js

- Is a file and module loader lib

  http://requirejs.org/

- You can require and define files and modules

3 steps to use require.js with SPA

1. Include require.js in your HTML file

```
<script data-main=”js/main" src=”js/lib/require.js"></script>
```

2. Configure require, normally in the data-main file

```
(function () {
    requirejs.config({
        baseUrl: 'Scripts',
        paths: {
            knockout: 'lib/knockout-3.4.0',
            jquery: 'lib/jquery-2.2.3.min',
            text: 'lib/text',
            bootstrap: 'lib/bootstrap.min'
        }
    });
})();
```

3 steps to use require.js with SPA

3. Use require to load your files

```
define(['knockout', 'app/config'], function (ko, config)
{
        ...
});
```

```
require(['knockout', 'app/viewmodel', 'app/config'],
        function (ko, vm, config) {
    ...
});
```

```
requirejs.config({
    baseUrl: 'Scripts',
    paths: {
        knockout: 'lib/knockout-3.4.0',
        jquery: 'lib/jquery-2.2.3.min',
        text: 'lib/text',
        bootstrap: 'lib/bootstrap.min'
    }
```