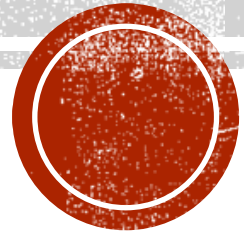
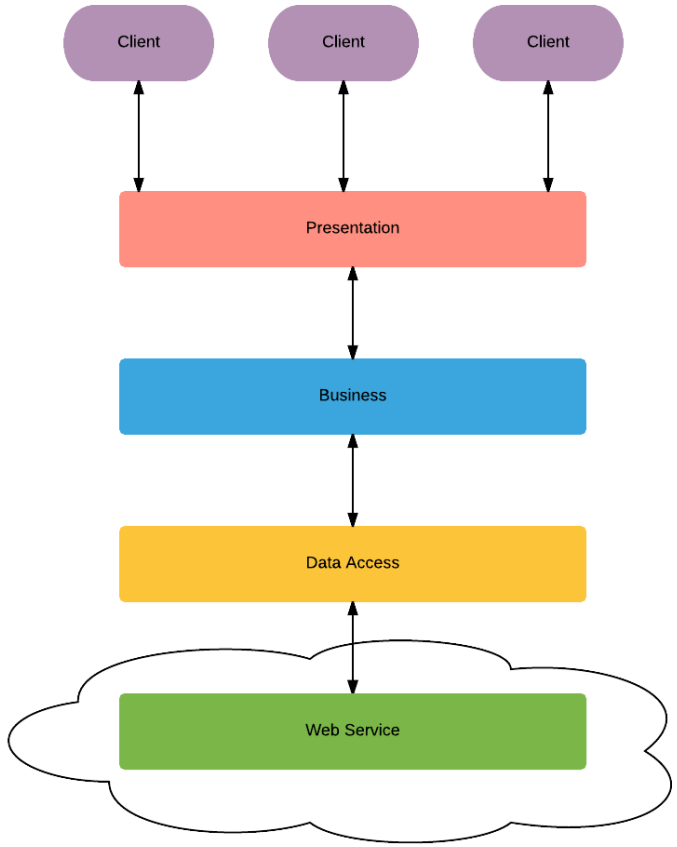
RAWDATA SECTION 3

Troels Andreasen & Henrik Bulskov

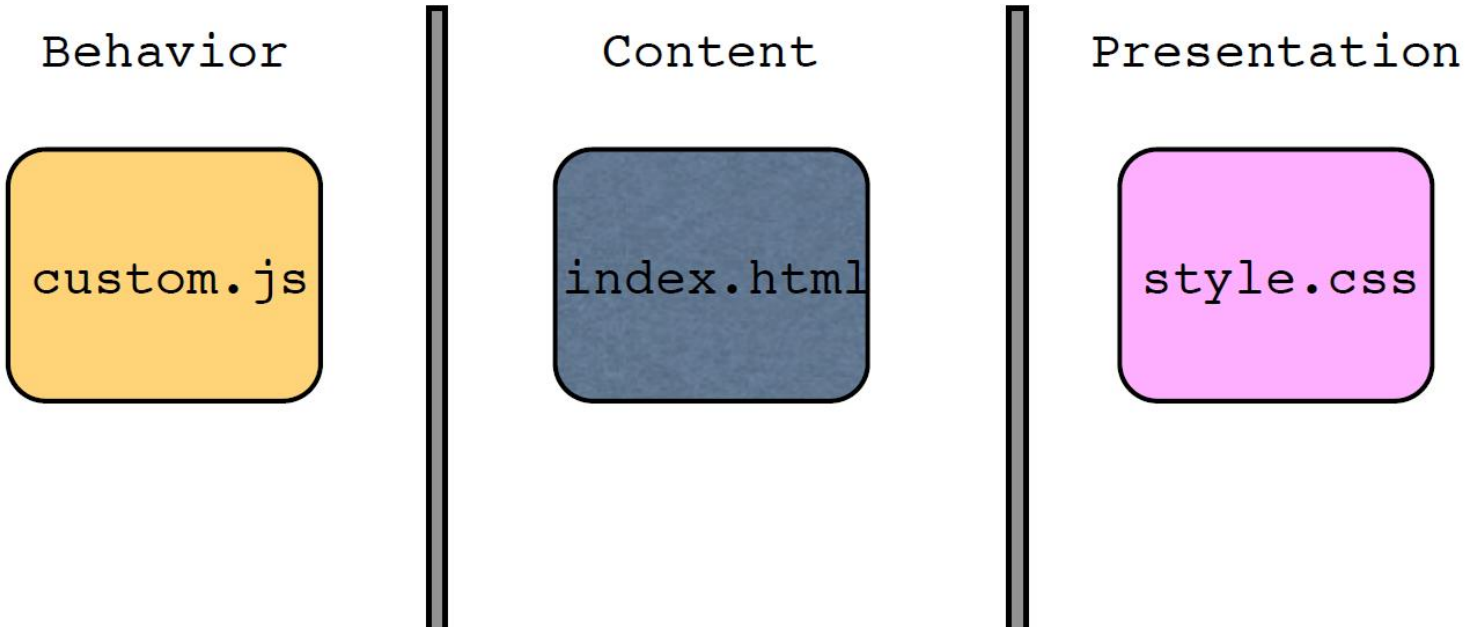


WHAT TO DO IN SECTION 3?

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



SYSTEM DEVELOPMENT



UNOBTRUSIVE DESIGN

Visual Studio

Visual Studio for Mac

Visual Studio Code

Rider JetBrains

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 **Arrow-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'';

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
  }
}
```

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

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

OBJECT CONSTRUCTION

```
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 FUNCTIONS

PROTOTYPE-BASED LANGUAGE

- JavaScript is often described as a prototype-based language — to provide inheritance, objects can have a prototype object, which acts as a template object that it inherits methods and properties from.
- An object's prototype object may also have a prototype object, which it inherits methods and properties from, and so on. This is often referred to as a prototype chain and explains why different objects have properties and methods defined on other objects available to them.

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

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

OBJECT PROTOTYPE

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!!!

- Let keyword
  - Introduce block scope

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

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

THE ARGUMENTS OBJECT

NAMED VS. ANONYMOUS FUNCTIONS

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

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

FUNCTION CLOSURE

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 parenteses
- 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

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

Private member

Public member

Accessing public member with 'this'

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.

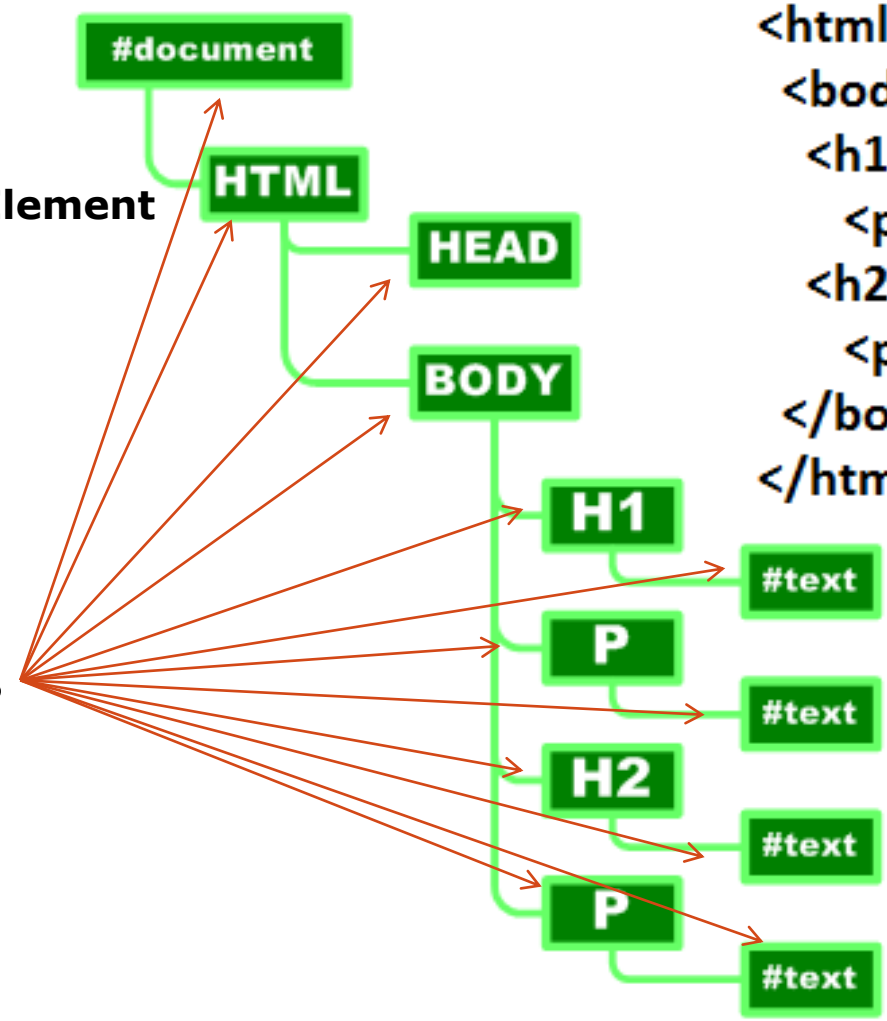
DOCUMENT TREE STRUCTURE

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

CSS SELECTORS

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

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;
}
```

WEB PAGES & ASP.NET

PREPARE ASP.NET FOR SERVING FILES

- Add a folder called wwwroot to your Web project
  - This folder serves as the root of your web server, thus all you files must be located here
- Modify your Startup.cs by adding the static file middleware by the statement app.UseFileServer() in the Configure method.
  - Place it just before the UseRouting() statement.
- Add your files to the wwwroot folder and the should be accessible (at the root of your server).
  - You can have any structure inside your wwwroot.
  - Some very common folders are js, css and images for separating javascript, style sheet, and images files, respectively.
  - If you do not provide a file in your path the FileServer will look for one of the default files: default.htm, default.html, index.htm, or index.html

https://docs.microsoft.com/en-us/aspnet/core/fundamentals/static-files?view=aspnetcore-3.0

WORKING WITH EXTERNAL RESOURCES

- Building modern web applications often lean on a number of different frameworks
- For the structure and layout of pages often grid systems are used
  - Bootstrap, FlexBox Grid
- For the styling of pages
  - Bootstrap, Pure.CSS
- For the behavior and interaction of web pages (single page app)
  - React, Vue, Angular, Ember, Knockout

WORKING WITH EXTERNAL RESOURCES

- To include external frameworks you can basically use two different methods:
  - Reference the resources via CDN
  - Keep a local copy of the resource on your server
- For the latter a number of package manager tools are available to handle the download for you:
  - Yarn
  - NPM
  - Bower
  - Libman (dot.net tool)

LIBMAN

- Use it from within Visual Studio
- Use it from the terminal (command line)
  - Install the LibMan CLI

    Run in terminal: dotnet tool install -g Microsoft.Web.LibraryManager.CLI

    (https://github.com/aspnet/LibraryManager/wiki/Using-LibMan-CLI)
  - Use unpkg (access to everything on npm)