Javascript
Because ECMAScript sounds horrible
Javascript

• Javascript is a general purpose programming language.

• It usually runs within a browser
  • Node.js runs Javascript in a server / application context

• Developed in the mid nineties as a simple way to provide interactivity to web pages.

• Originally developed by Brendan Eich working at Netscape

• Submitted to ECMA standards body in 1996

• ECMAScript 5.1 released in 2011
Javascript In A Browser

• REPL

  • Read-Eval-Print Loop

• All major browsers have a Javascript REPL system in the console
Javascript In A Browser

```javascript
1 + 2
3
a = "hello"
"hello"
b = ['a', 'b', 'c', 'd']
['a', 'b', 'c', 'd']
b[2]
"c"
o = {name: 'Mark', class: 'CS37'}
Object {name: "Mark", class: "CS37"}
o.name
"Mark"
document
#document
<!DOCTYPE html>
<html>
<head>...
</head>
<body>...
</body>
</html>
```
Documentation

http://ecma262-5.com/ELS5_HTML.htm

Data Types

• Basic Data Types
  • number
  • boolean
  • string
  • object
Data Types

- `typeof` unary operator
- lets us know what we’re dealing with
- If you’re evaluating a complex operation, you need parenthesis. Not because `typeof` is a function, but to make sure that there’s only one argument to `typeof`
Numbers

• Javascript has a single number datatype to deal with all numbers.

• No distinction between integers, floats, doubles, etc.

• All numbers are represented as floating point numbers, but if the fractional part is zero, they’re shown as integers.
Numbers

- Numbers stored in variables are converted objects when needed, to have methods and properties
  - Number.toString()
  - Number.toPrecision()

Strings

• A series of zero or more characters.

• Unicode support is pretty good.

• Browser support for full unicode support is spotty.
Strings

• String variables are also converted to objects as needed.

• String.toUpperCase()

• String.substring(start, end)

• Note the difference between .substring() and .length
  • One is a method, one is a property

Boolean

- Boolean for `true` and `false`.
- Comparisons
- Coerce other datatypes into Boolean.
- Note the behavior of the Boolean value for strings.
  - Empty string is `false`
  - Other strings are `true`. Even “false”!
Variables

- Variable names can be any combination of letters, numbers, an underscore (_), or $.

- Variable names cannot start with a number.

- Variables do not need to be declared.

- The `var` keyword can be used to declare and scope variables.

Variables

- Variables have global scope unless `var` is used to declare a variable.

```javascript
var foo = function() {
    numBaloons = 99;
}
foo();
console.log(numBaloons);
```

```javascript
var foo = function() {
    var numBaloons = 99;
}
foo();
console.log(numBaloons);
```

```javascript
Uncaught ReferenceError: numBaloons is not defined
```
Arrays

- Collection of values
- Created with \([n, n+1, \ldots k-1]\) syntax
- Array access with brackets: \(n[]\)
- Length property
- Standard Zero based indexing
Arrays

- Arrays can be collections of many different datatypes.
Arrays From Strings

- `String.split()` to create an array from a string.

```
> s = "983,Name,ID,OrderStats,9,15";
> "983,Name,ID,OrderStats,9,15"
> orderDetails = s.split("","");
> ["983", "Name", "ID", "OrderStats", "9", "15"]
> typeof orderDetails
> "object"
> orderDetails[1]
> "Name"
> 
```
Array Methods


• Lots of useful array methods.

• `.contains(<some value>)` // returns true or false

• `.join(<glue string>)` // joins all elements together with glue and returns a string.

• `.toString()` // Quick string representation of the array

• `.pop() .push() .shift() .unshift()` // Standard array methods

• `.sort()` // Sorts elements according to criteria

• `.splice()` // Adds or removes elements from an array
Array Assignment

- Assigning an array to another variable assigns a reference of the array to the variable, not a copy.
Array Assignment

- To make a copy of an array, use the `.slice(0)` method.
undefined

- Javascript has a special value for things that are not defined: `undefined`
- Out of bounds requests
- Un-initialized variables
- `undefined` is a property of the *global object*. Its type is `undefined`.

Objects

- Objects are very flexible data structures.

- A basic object:

  ```
  o = {id: 1, name: "an object", counter: 10};
  ```

- Create property names and values using key: value syntax.

- Separate multiple properties by commas.
Objects

```javascript
let o = {id: 1, name: "an object", counter: 10};
```

- Access properties via dot syntax

```
> o = {id: 1, name: "an object", counter: 10};
let Object {id: 1, name: "an object", counter: 10}
> o.id
1
> o.name
"an object"
> typeof(o.counter)
"number"
```
Objects

\[
o = \{id: 1, \text{name: "an object"}, \text{counter: 10}\};
\]

- Act as “Associative Arrays” or “Key / Value” arrays, or “Dictionary” array
- `arr["key"]` syntax
Objects

- Assigning to undefined properties creates them

```javascript
0
Object {id: 1, name: "an object", counter: 10}
o.desc = "A description"
"A description"
0
Object {id: 1, name: "an object", counter: 10, desc: "A description"}
```
null

- Null is a literal value representing an “empty” or non-existent value.

Operators

- Arithmetic Operators: + - / * % ++ --

- String concatenation: +

- Logical Operators: && || !

- Comparisons: < > <= >=

- Ternary Operator: condition ? true expr : false expr

- Bitwise Operators: << >> ^ ~
Control Structures

- if (condition) { stmt1 } else { stmt2 }
- while (condition) { statements }
- for (i = 0; i < 10; i++) { statements }
- Pretty much work like every other C or Java style language
Control Structures: forEach

- Arrays have a special `forEach` method for performing some action relating to each element of the array.

- The `forEach` method takes a `function` as an argument.

```javascript
// a = ["one", "two", "three"];
a.forEach(function(element, index, arr) {
    console.log(element.toUpperCase());
});
```
Basic I/O

- Alerts
- Log to Console
- Confirms
- Prompt
- DOM Manipulation
- Debugger
- No Direct Local File I/O!
alert( )

alert("Hello World");

- Display a modal dialog box with the specified text.
- Pauses execution of Javascript until dialog is dismissed.
console.log( )

```javascript
console.log("Hello World");
```

- Quick way to get some debugging out.

- Doesn’t block execution, so usually a better choice for debugging and testing than `alert()`.
confirm("Yes or No?");

• Ask for a true or false response from the user.
prompt( )

```javascript
prompt("Enter a number between 1 and 10");
console.log(i);
```

- Ask for user input as a text string.
Debugger

• Most browsers have a full featured interactive debugger built in.

• Breakpoints, watched expressions, step through execution, etc.

• Example.
Functions

• Multiple ways to define a function

```javascript
function echo(a) {
    return a;
}

echoTwo = function(a) {
    return a;
}

var echoThree = function(a) {
    return a;
}

console.log( echo("one") );
console.log( echoTwo("two") );
console.log( echoThree("three") );
```
Functions

Declares a named function without requiring assignment

Declares a *global* variable `echoTwo` and assigns an anonymous function to it

Declares a *local* variable `echoThree` and assigns an anonymous function to it

```javascript
function echo(a) {
    return a;
}

echoTwo = function(a) {
    return a;
}

var echoThree = function(a) {
    return a;
}

console.log( echo("one") );
console.log( echoTwo("two") );
console.log( echoThree("three") );
```
Functions

- Does any of this matter?
- What if we call the functions before they’re declared?

```javascript
function echo(a) {
    return a;
}

echoTwo = function(a) {
    return a;
}

var echoThree = function(a) {
    return a;
}

console.log( echo("one") );
console.log( echoTwo("two") );
console.log( echoThree("three") );
```
Functions

```javascript
console.log( echo("one") );
console.log( echoTwo("two") );
console.log( echoThree("three") );

function echo(a) {
    return a;
}

function echoTwo(a) {
    return a;
}

var echoThree = function(a) {
    return a;
}
```

Uncaught ReferenceError: echoTwo is not defined
Functions

• The first style has a symbol table entry created for it at parse time. So it can be referenced immediately during runtime.

• The other two have symbol table entries created at runtime, so aren’t available until after they’ve been executed.
Functions

//Function Declaration
function add(a,b) {return a + b};

//Function Expression
var add = function(a,b) {return a + b};

• So should we always use Function Declarations?

• Well, it depends…
• What is the console output here?

```javascript
function echo(a) {
  return a;
}

console.log( echo("one") );

function echo(a) {
  return a.toUpperCase();
}

console.log( echo("one") );
```
Functions

• Hmm, maybe not what we were expecting.

• Function Declarations are ‘hoisted’ to the top at parse time, so when executed, the last declared version wins.
Function Declarations

• Can only appear as block level elements.

• Are ‘hoisted’ to the top at parse time, before run time.

• Cannot be nested within non-function blocks.

• Are scoped by where they are declared, like `var`
Function Expressions

• Can be used anywhere an expression is valid.
  • Can be more flexible because of this.
• Are evaluated and assigned at run time.
Objects and Functions

- Functions can be added to objects as property variables.
- Many object “methods” are really properties with functions assigned to them.
Objects and Functions

```javascript
var doubleMe = function(x) {
    return 2 * x;
}

var halveMe = function(x) {
    return x/2;
}

var myLib = {
    version: 0.3,
    name: "My Test Library",
    double: doubleMe,
    half: halveMe
}

console.log( myLib.double(3) );
console.log( myLib.half(10) );
```
Objects and Functions

- Using anonymous function expressions instead.

```javascript
var myLib = {
    version: 0.4,
    name: "My Test Library",
    double: function(x) { return 2 * x; },
    half: function(x) { return x/2; }
}

console.log( myLib.double(3) );
console.log( myLib.half(10) );
```
Javascript in HTML

• Where does our Javascript live?

• Inline in an HTML document inside a `<script>` element

• Included in an external file via a `<script>` element.
Javascript in HTML

- The `<script>` element with inline content
- Within the `<script>` element, we’re parsing Javascript, not HTML
Javascript in HTML

- The `<script>` element with `src` attribute.
- Includes an external file with Javascript in it.
- No wrapping `<script>` tags within external files.

```html
<!doctype html>
<html>
<head>
  <title>js/jstest.html</title>
  <script src="jstest.js"></script>
</head>
<body>
  <div></div>
</body>
</html>

var answer = 42;
function calculateAnswer() {
  return answer;
}
console.log( calculateAnswer() );
```
The document Object

This is all well and good, but how about something involving a web page?
The document Object

- Browsers parse the HTML and CSS of a page, and build an object model in memory.
- The browser exposes this object to us for use with our Javascript as the `document` object.
The document Object

```html
<top frame>

  typeof document
  "object"

document
  #document
    <!DOCTYPE html>
    <html>
      <head>
        <title>Hello World</title>
      </head>
      <body>
        <p>A Basic HTML Page.</p>
      </body>
    </html>
```
The document Object

• The document object represents the root element of our DOM tree.

• It has child nodes, and each node has various attributes.

• Note the difference between .children and .childNodes
The document Object

- `document` elements are *objects*, so accessing their properties is done with the dot syntax
  - `object.property`
- `html.innerHTML` for example
The document Object

- The document object is *NOT* part of the Javascript language.

- It is an API defined by the W3C to interact with HTML and XML documents.

DOM Selection

• Starting with the `document` root and drilling down via `.children` is tedious. Can we get at elements some other way?

  • `document.getElementById("main")`
  
  • `document.getElementsByTagName("p")`
  
  • `document.getElementsByClassName("error")`
getElementsByById

• Gets an HTML Element object from the document based on an ID.

• Since ID must be unique, this method returns a single element, not an array of elements.
getElementById

```html
<!doctype html>
<head>
  <title>js/getElementById.html</title>
  <link rel="stylesheet" type="text/css" href="getElements.css" />
</head>

<body>
  <div id="main">
    <div id="first" class="item">
      First Block
    </div>
    <div id="second" class="item">
      Second Block
    </div>
    <div id="third" class="item selected">
      Third Block
    </div>
  </div>
</body>
</html>
```
Updating the DOM

• Now that we can get an element, can we do something with it?

```html
<!doctype html>
<head>
  <title>js/getElementById.html</title>
  <link rel="stylesheet" type="text/css" href="getElements.css" />
  <script src="getElementById.js"></script>
</head>

<body>
  <div id="main">
    <div id="first" class="item">First Block</div>
    <div id="second" class="item">Second Block</div>
    <div id="third" class="item">Third Block</div>
  </div>
</body>
</html>

d2 = document.getElementById('second');
d2.classList.add("selected");
Updating the DOM

• Hmm nothing happened. Why? Check the console.
Updating the DOM

• Uncaught TypeError: Cannot read property 'classList' of null?? But how can d2 be null?

```html
<!doctype html>
<head>
<title>js/getElementById.html</title>
<link rel="stylesheet" type="text/css" href="getElements.css"/>
<script src="getElementById.js"></script>
</head>
<body>
<div id="main">
  <div id="first" class="item">First Block</div>
  <div id="second" class="item">Second Block</div>
  <div id="third" class="item">Third Block</div>
</div>
</body>
</html>
```
Waiting for the DOM to load

- The browser waits for no DOM
- The browser parses the file, loads the `getElementById.js` file, and executes it all before the rest of the HTML is parsed and the DOM is created.
Waiting for the DOM to load

• What if we just move the `<script>` element down to the bottom?
Waiting for the DOM to load

• Works!
Waiting for the DOM to load

• That seems… hackish. Isn’t there a “right” way to do this?

• Well, it’s perfectly valid. `<script>` elements do not have to go in the `<head>`, although they frequently do.

• However, `<script>` elements that aren’t in the `<head>` tend to get overlooked later, so we try to put them there if we can.
Events

• The web browser is an Event Driven application.

• Documents load, links are clicked, HTTP requests are made and completed.

• Each of these is an event, and we can register event listeners (function) which will be called as these events occur.

• These are called *callbacks*. 
Events

• `object.addEventListener('event', callback);`

• The object can be any object that responds to event listeners, such as an Element, the Document, or maybe the Window.
Events

- A basic example of a ‘click’ event handler.

```html
<!doctype html>
<head>
  <title>js/events.html</title>
  <link rel="stylesheet" type="text/css" href="getElements.css" />
</head>

<body>
  <div id="main">
    <div id="first" class="item">First Block</div>
    <div id="second" class="item">Second Block</div>
    <div id="third" class="item">Third Block</div>
  </div>

  <script>
    clickCount = 0;
    d1 = document.getElementById('first');
    d1.addEventListener('click', function() {
      console.log("Clicked " + ++clickCount + " times.");
    });
  </script>
</body>
</html>
```
Events

• Is it really that simple? What about IE, doesn’t that always mess us up?

• Well, yes. Of course it does.

• `object.addEventListener()` didn’t come to IE until 9

• Earlier methods for adding event listeners were directly in markup, or via `object.event = callback;`
window load Event

• There’s also a window object that the DOM API provides for us.

• The Window object supports the load event, and we can register our own callback with this.

• The load event fires once the DOM has completed loading.
window.load Event

```html
<!doctype html>
<head>
  <title>js/window-load.html</title>
  <link rel="stylesheet" type="text/css" href="getElements.css" />
  <script src="window-load.js"></script>
</head>

<body>
  <div id="main">
    <div id="first" class="item">First Block</div>
    <div id="second" class="item">Second Block</div>
    <div id="third" class="item">Third Block</div>
  </div>

  window.addEventListener('load', function() {
    d2 = document.getElementById('second');
    d2.classList.add('selected');
  });
</body>
</html>
```
window load Event

• Works!
window load Event

• Since addEventListener doesn’t work with IE 8 or older, to provide a more robust solution you’d have to do browser capabilities detection.

```javascript
window.addEventListener('load', function()
{
    d2 = document.getElementById('second');
    d2.classList.add('selected');
});
```
window load Event

• IE 8 supported a different method, the `object.attachEvent` method.

• Even older browsers only support a single “onload” property.

• If only someone would write a library that did all this for us...

```javascript
var ready = function(myFunction) {
  if (window.attachEvent) {
    window.attachEvent('onload', myFunction);
    console.log("IE");
  } else if (window.addEventListener) {
    window.addEventListener('load', myFunction);
    console.log("Modern");
  } else {
    console.log("Legacy");
    if (window.onload) {
      var currofnload = window.onload;
      var newonload = function() {
        currofnload();
        myFunction();
      };
      window.onload = newonload;
    } else {
      window.onload = myFunction;
    }
  }
}
```
Putting Pieces Together

Demo
<!doctype html>
<head>
    <title>js/click-count.html</title>
    <link rel="stylesheet" type="text/css" href="click-count.css"/>
    <script src="click-count.js"></script>
</head>

<body>
    <div id="main">
    </div>
</body>
</html>
```javascript
var addCount = function(event) {
    var curCount = Number(this.textContent);
    curCount++;
    this.textContent = curCount.toString();
}

window.addEventListener('load', function() {
    var numBoxes = 9;
    main = document.getElementById('main');
    for (i = 0; i < numBoxes; i++) {
        var newBox = document.createElement('div');
        newBox.textContent = "0";
        newBox.addEventListener('click', addCount);
        main.appendChild(newBox);
    }
});
```
click-jump.html
```javascript
var addCount = function(event) {
    var curCount = Number(this.textContent);
    curCount++;
    this.textContent = curCount.toString();

    if (curCount == 1) {
        this.style.position = "absolute";
    }

    var max_x = window.innerWidth - 110;
    var max_y = window.innerHeight - 60;
    var newX = Math.random() * max_x;
    var newY = Math.random() * max_y;
    newX = Math.floor(newX);
    newY = Math.floor(newY);

    this.style.top = newY.toString() + "px";
    this.style.left = newX.toString() + "px";
}
```
Inspiration

- Case study on copying stuff from other people.
- https://account.arizona.edu/welcome
Updating Styles

• Many ways of updating DOM elements involves changing its CSS Style Attributes
  • Positioning an element somewhere
  • Changing font styles
  • Changing colors, borders, etc
An HTMLElement object has a style property

The style property is itself an object, having properties for all the CSS properties appropriate to that element

```
element.style.color = "#cccccc";
```
element.style

size = 2;
```
element.style.fontSize = size.toString() + "em";
```

- All values are strings. If you have to assign numerical values, you need to convert them to strings.

- You also need to make sure the value is a complete and valid value for the property, including any units.
element.style

```javascript
element.style.fontSize = "2em";
```

- You can't have javascript variable names that contain a dash... its the subtraction operator.

- CSS properties with dashes in their name become camelCased

  ```javascript
  font-size   ->   fontSize
  ```
Timing Events

• Browsers implement Javascript in a threaded environment.

• Events can be queued to fire at a later time.

• `window.setTimeout()`

• `window.setInterval()`

setTimeout()

```html
<!doctype html>
<head>
  <title>js/timeout.html</title>
  <link rel="stylesheet" type="text/css" href="timeout.css" />
</head>

<body>
  <div id="main">0</div>

  <script>
    var counter = function() {
      var d = document.getElementById('main');
      var curCount = Number(d.textContent);
      curCount++;
      d.textContent = curCount.toString();
    }
    window.setTimeout(counter, 1000);
  </script>

</body>
</html>
```
setInterval()

- `setTimeout()` only fires a single time.
- To fire on an interval, use `setInterval()`, or continually call `setTimeout()`.
- Demo
CAN YOU PASS THE SALT?

I SAID—
I KNOW! I'M DEVELOPING A SYSTEM TO PASS YOU ARBITRARY CONDIMENTS. IT'S BEEN 20 MINUTES! IT'LL SAVE TIME IN THE LONG RUN!
Classes

Oops, sorry, there are no classes.
Class Like Thingies

• Javascript has no “Class” concept.

• Objects are based on building on a prototype.

• “Instances” are not tied to a particular static Class definition.

• funcitons?
functions and new

- Classes are just functions!
- Create new instances with the new keyword.

```javascript
function Droid(type, name) {
    this.type = type;
    this.name = name;
}

var r2 = new Droid('astromech', 'R2D2');
var c3 = new Droid('protocol', 'C3PO');

console.log(r2);
```
prototypes

- Methods can be added through the special `.constructor` property of objects.

```javascript
function Droid(type, name) {
  this.type = type;
  this.name = name;
}
Droid.prototype = {
  getName: function() { return this.name },
  getType: function() { return this.type }
}
var r2 = new Droid('astromech', 'R2D2');
var c3 = new Droid('protocol', 'C3PO');
console.log(r2);
console.log(r2.getName());
```
prototypes

• Don’t like the behavior of something? Re-define it on the fly

```javascript
function Droid(type, name) {
    this.type = type;
    this.name = name;
}

Droid.prototype = {
    getName: function() { return this.name },
    getType: function() { return this.type }
}

var r2 = new Droid('astromech', 'R2D2');
var c3 = new Droid('protocol', 'C3PO');

console.log(r2.getName());

Droid.prototype.getName =
    function() { return this.name.toLowerCase() };

console.log(r2.getName());
```
myQuery

- jQuery is a very popular Javascript toolkit which abstracts away some of the underlying complexity.
- Can we build our own simple toolkit?
- Of course we can…
- jQuery doesn’t own $
Basic Selection

- Using `document.getElementById()` isn’t too bad, but it sure is a lot of typing.

- Can we use the `$('selector')` pattern?

```javascript
var $ = function myQuery(selector) {
  // See if selector starts with a #. If so we're looking for an ID
  if (selector[0] == '#') {
    // Strip off the # sign
    var selector = selector.substring(1, selector.length);
    var element = document.getElementById(selector);
    return element;
  }
}
```
Basic Selection

```javascript
var $ = function myQuery(selector) {
    // See if selector starts with a #. If so we're looking for an ID
    if (selector[0] == '#') {
        // Strip off the # sign
        var selector = selector.substring(1, selector.length);
        var element = document.getElementById(selector);
        return element;
    }
}
```

```html
> a = $('#post1');
> <article id="post1">Foo</article>
> a = $('#main');
> <section id="main">...</section>
> </p>
```
Returning Objects

```javascript
function myQuery(selector) {
    this.element = null;
    this.selector = selector;

    // See if selector starts with a #.
    // If so we're looking for an ID
    if (selector[0] == '#') {
        // Strip off the # sign
        var selector = selector.substring(1, selector.length);
        var element = document.getElementById(selector);

        myQobj = new myQuery(selector);
        myQobj.element = element;
        return myQobj;
    }
}

var $ = myQuery;
```
prototype Methods

```javascript
myQuery.prototype = {
    getElement: function() {
        return this.element;
    },
    getSelector: function() {
        return this.selector;
    },
}
```

Function Chaining

• Supports function chaining.

• The return value from the function call is an object, which has methods we can call.

• Don’t need intermediate variables.
jQuery

• This is basically what jQuery does.
• More methods and selector types.
• There’s a lot more edge cases handled, and checks made.
• jQuery ‘plugins’ just add their own function calls to the jQuery prototype property.

http://code.jquery.com/jquery-2.2.1.js
And now for something moderately different