INTRODUCTION TO JAVASCRIPT

Created by **Abram Hindle**
abram dot hindle at ualberta dot ca
Department of Computing Science
University of Alberta
Edmonton, Alberta, Canada
Earth

JAVASCRIPT

Good resources:

- Javascript the good parts by Douglas Crockford (the guy who made JSON!)
    - **http://proquest.safaribooksonline.com/book/programming/javascript/9780596517748**
- W3Schools Javascript Tutorials
    - **http://www.w3schools.com/js/**
- JSFiddle - Web JS IDE
    - **http://jsfiddle.net/**
- **Video: Firefox Debugger**
- **New HTML5 APIs for JS**
- **HTML and Scripting from the spec**
- **ECMA-262: Ecmcascript spec**
- **Javascript spec**

JAVASCRIPT

Where did it come from?

- Netscape -- Brendan Eich
    - Called Mocha, LiveScript and eventually Javascript
- Inspired by Java (Sun)
- Inspired by C
- Inspired by Self

JAVASCRIPT

What kind of language is it?

- Dynamic
- Object Oriented
- Imperative
- Functional
- Prototype Driven
- Embeddable
- **ECMA-262**

JAVASCRIPT

Why?

It runs in browsers (with minimal compatibility issues).

It runs on webservers.

It runs in PDFs.

It is embedded everywhere.

It is getting better and better performance.

JAVASCRIPT ON WEBPAGES

- put the javascript within <script> tags.

```
<script type="text/javascript">
//<![CDATA[
// ^^^ do this for XHTML Compatibility if you are going to embed javascript
var someJS = "Put your javascript here";
//]]>
</script>


<script>
var someJS = "This works but isn't XHTML compatible";
</script>
<!-- you can embed oneliners within HTML! -->
<input value="test me!" type=button
       onclick="javascript:alert('hey look ma!');"/>
```

test me!

JAVASCRIPT SCOPE

- Scope: Lexicalish.
    - Javascript has a scope that feels lexical but really entire functions are a single block scope.
    - If statements and for loops do not create new scope
    - Variables are global by default unless they are pre-declared with var

```
// here's a comment
/* here's a variable assignment */
var variable = "A string value";
function init() {
    f = "Globally available don't do this!";
}
function testF() {
    return f;
}
function testVariable() {
    var variable = "A local value!";
    return variable;
}
init();
```

testF! testVariable! alert(variable)!

JAVASCRIPT SCOPE

- Scope: Closures.

```
// Good style dictates all variables should be defined with var
// this avoid confusion
var variable = "A string value";
// var causes variables to be defined in their initially scope
// and are made available to all functions defined within their scope.
function lexical() {
    var shared = "share!"; // defined in this scope
    return function() { // shared is also defined (and shared) in this function
        return shared;  // note how this function references shared
    };                  // did you also see we just returned an
}                       // anonymous function!
alert( // so now we call lexical() get a function and then
    ( lexical() )()  // call that function using ()
);
```

alert((lexical())());

JAVASCRIPT SCOPE

- Scope: Closures.

```
// Closures are functions who reference another scope
// First we make a named function called makeCounter
function makeCounter() {
    var counter = 0;
    // note that functions are values!
    var count = function() {
        return counter++; //post increment counter
    }
    return count;
}
var myCounter = makeCounter();
myCounter();
myCounter();
myCounter();
// myCounter returned 2
```

Example

JAVASCRIPT

- Functions and closures
    - functions can be named and take parameters
    - function can be anonymous and take parameters as well
    - function(makes an anonymous function

```
function cpsAdd(val1, val2, continuation) {
    continuation(val1 + val2);
}
function add1234(continuation) {
    cpsAdd(1,2, function(x) {
        cpsAdd(x,3, function(x) {
            // x is shadowed
            cpsAdd(x,4, continuation);
        });
    });
}
add1234(alert);
```

add1234(alert);

JAVASCRIPT

- Functions and closures
    - returning functions can be quite useful
        - or confusing

```
function cpsAddR(val1, val2) {
return function(continuation) {
   continuation(val1 + val2);
};
}
function addR(continuation) {
// if you don't have a return the last va
// is implicitly returned!
cpsAddR(4,5)( function(x) {
    cpsAddR(x,6)( function(y) {
        // both x and y are available her
        cpsAddR(x,y)( continuation );
    })
});
}
addR(alert);
```

addR(alert);

JAVASCRIPT NAMES

- Names
  - starts with a letter followed by underscores, letters or numbers.
  - can't be a **reserved word** like break or case or for or function or if or in etc.

```
var aString = "Strings";
var break = "not allowed!";
var BREAK = "This is allowed!";
var BrEAK = "Try not to abuse case sensitivity";
```

JAVASCRIPT NUMBERS

- Numbers
  - Everything is a double :(
  - Expressed as an integer, a decimal or exponent

```
var aNumber = 10;
var aNumber = 11.11;
var aNumber = 1e-100;
var aNumber = 1E+100;
var nan = NaN;
var inf = Infinity;
var negativeInfinity = -Infinity;
```

JAVASCRIPT STRINGS

- Strings
    - Unicode by default
    - Define strings with " or ""
    - Any characters except control characters and \ and " or "" depending if you use " or ""

```
var aString = "";
var anotherString ="Hi how are you";
var escapesString = "\r\n\t\f\b\/\\\\'\"";
var snowMan = "\u2603";
snowMan.length === 1;
aString.length === 0;
```

JAVASCRIPT BOOLEANS

- Booleans
  - true
  - false
  - Unfortunately if statements have lot of truthy and falsey values
  - False values:

```
false
null
undefined
''
0
NaN
```

- True values are true and everything else.

JAVASCRIPT ARRAYS

- Arrays
  - Objected Oriented and full of methods
  - 0 indexed;
  - Indexable

```
var empty = [];
var arrayInitialized = [1,2,3,4,'5'];//mixed!
var arr = new Array(10);
arr[0] === undefined;
arr[0] = 'Assigned';
'Assigned' === arr[0];
arrayInitialized[4] === '5';
arrayInitialized.length === 5;
arrayInitialized.splice(3,1); // delete 4 from the array
```

JAVASCRIPT OBJECTS

- Objects
  - Everything is an object except booleans, and numbers and strings
  - numbers and strings smell like objects but are fake.
  - Objects have properties
  - properties are named by a string
  - property values are anything except undefined
  - Objects don't have a class
  - Pass by reference

JAVASCRIPT OBJECTS

```
var empty = {};
var abram = {
   "name":"Abram Hindle",
   "job":"Throwing Down JS",
   "favorite tea":"puerh"
};
var dog = {
   paws: 4 // note I didn't quote paws
};
dog.paws === 4;
abram["favorite tea"] === "puerh";
abram.name === "Abram Hindle";
abram["favorite tea"] = "oolong";
```

JAVASCRIPT OBJECTS

```
undefined.property; // Throws a type error
undefined && undefined.property // returns undefined
var empty = {};
empty.property === undefined;
var abram = {
   "name":"Abram Hindle",
   "job":"Throwing Down JS",
   "favorite tea":"puerh"
};
keys(abram); // produces ["name","job","favorite tea"]
//prototype!
var abramChild = Object.create(abram)
keys(abramChild); // produces []
abramChild.name === "Abram Hindle"; // inherits keys from abram
```

JAVASCRIPT OBJECTS

```
var abram = {
   "name":"Abram Hindle",
   "job":"Throwing Down JS",
   "favorite tea":"puerh",
   "sayName": function() {
      alert(this.name);
   }
};
abramChild = Object.create(abram);
abramChild.name = "Child";
function doit() {
    abram.sayName();
    abramChild.sayName();
}
```

doit()

JAVASCRIPT METHODS

```
var abram = {
   "name":"Abram Hindle",
   "job":"Throwing Down JS",
   "favorite tea":"puerh",
   "sayName": function() {
      alert(this.name);
   }
};
abramChild = Object.create(abram);
abramChild.name = "Child";
function doit() {
    abram.sayName();
    abramChild.sayName();
}
```

doit()

JAVASCRIPT METHODS

- There's a problem, if we use a function to construct our objects.
- The context of this does not get shared appropriately in inner functions since this will reference the function itself.

```
// we can use new on person now!
var Person = function() { // function is an object
   this.name = "Abram Hindle"; // this is the current function
   this.job = "Throwing Down JS";
   this["favorite tea"] = "puerh";
   var self = this; // you could do this if you like perl
   var that = this; // this is more idiomatic
   this.sayName = function() {
       var thatNameAccessor = function () {
           return that.name;
       };
       var thisNameAccessor = function () {
           return this.name;
       };
       alert("this:" + thisNameAccessor() +
             " that:" + thatNameAccessor());
   }
};
function doit2() {
    var p = new Person();
    p.sayName();
}
```

doit2()

JAVASCRIPT METHODS

- There's a problem, if we use a function to construct our objects.
- The context of this does not get shared appropriately in inner functions since this will reference the function itself.

```
// look a constructor!
var Animal = function(name) { // function is an object
   this.name = name; // this is the current function
   this.odd = (Math.random() > 0.5);
   var that = this;
   this.likesNumber = function(x) {
       return (x%2 == 1)?this.odd:!this.odd;
   }
};
function doit3() {
   var animal = new Animal(prompt("Name your Animal"));
   var num = prompt("A number your animal might like!");
   alert( animal.likesNumber( num ) ? "Yes!"+animal.name+" loves it" :
                                      "No "+animal.name+" hates it!" );
}
```

doit3()

JAVASCRIPT METHODS

- Objects have a prototype object that they are prototyped from
- When you change a prototype it will be available in all instances immediately

```
// look a constructor!
var Counter = function(name) { // function is an object
   this.name = name; // this is the current function
   this.count = 0;
   this.inc = function() { ++this.count; }
};
function counterTest() {
   var counter1 = new Counter("sheep");
   var counter2 = new Counter("people");
   for (var i = 0 ; i < 10; i++) {
       counter1.inc();
       counter2.inc();
   }
   // Dynamically Add a method to all counters in the system
   Counter.prototype.dec = function() { --this.count };
   counter2.dec();
   alert("Counter1:"+counter1.count+" Counter2:"+counter2.count);
}
```

counterTest()

HTML DOM

- **JS HTML DOM Tutorial**
-

HTML DOM ELEMENT

```
<h1>A header title</h1>
<div>
Hi!
<a href="http://google.ca">Click me!</a>
</div>
```

A HEADER TITLE

Hi! ***Click me!***

HTML DOM ELEMENT

- Document (it's a tree with children nodes!)
  - Root Element: HTML (document.children[0])
    - Element: Head (document.children[0].children[0])
    - Element: Body (document.children[0].children[1])
      - Element: h1 (document.children[0].children[1].children[0])
        - Text: A header title
      - Element: div
        - Text: Hi!
        - Element: a ; attribute href
          - Text: Click me!

HTML DOM ELEMENTS

- Document

```
document
```

- Get children

```
document.children // a list (only Elements)
document.childNodes // a list of Nodes (includes te
// oh look you can dynamically make elements!
var elm = document.createElement("div");
// textContent returns text of childNodes!
elm.textContent = "Here's some text in that div";
elm.childNodes.length === 1; // TextNode is a node
elm.children.length === 0; // no Element children
// Append it to the body
document.children[0].children[1].appendChild(elm);
```

HTML DOM ELEMENTS

- Get Specific Children

```
// Get all DIVs
var divs = document.getElementsByTagName("div");
// gets all elements with class divider
var dividers = document.getElementsByClassName("divider");
// get the element with the ID main
var main = document.getElementById('main');
// get the element by name
var ups = document.getElementsByName('up');
```

HTML DOM ELEMENTS

- Create Children

```
function addToExample() {
  // gets all elements with class divider
  var example = document.getElementsByClassName("example")[0];
  var elm = document.createElement("div");
  elm.textContent = "Add me!";
  example.appendChild(elm);
}
```

```
<div class="example">
Hi! I'm Example!
</div>
```

addToExample

Hi! I'm Example!

HTML DOM ELEMENTS

- Manipulate Styles

```
function modifyStyle() {
  // gets all elements with class divider
  var examples = document.getElementsByClassName("example2");
  examples.map( function(example) {
     example.style.borderWidth = int(10*Math.random())+"px";
  });
}
```

```
<div class="example2">Hi! I'm Example2!</div>
<div class="example2">So am I!</div>
```

modifyStyle

Hi! I'm Example2!
So am I!

HTML DOM ELEMENTS

- Explore with Firefox and Chromium
  - Start Developer Tools and Go to the Console
  - Experiment with a webpage like **http://metafilter.com/**
  - Change dom elements
  - Console lets you try out javascript and see the results

HTML DOM RESOURCES

- **W3CSchools Javascript DOM**
- **W3CSchools Javascript HTML DOM Examples**
- **DOM Intro and Reference from MDN**
- **Mozilla Network DOM Refs**
- **W3C What is a DOM**

JQUERY

- **JQuery**
  - Cross Browser Javascript Object Model and Utility library
  - Monkey patches itself into Javascript via prototypes
  - let's you you deal with the page when it is ready

```
$( document ).ready(function() {
    // Do everything you have to now that the page has loaded
}
```

JQUERY

- Selectors: Like CSS but in Javascript and saves you a lot of time
  - $ is a function

```
// How many slides in this document?
alert( $( "section" ).length );
```

How many Sections?

JQUERY

- Selectors: Like CSS but in Javascript and saves you a lot of time
  - $ is a function
  - They let you operate on multiple objects
  - JQuery also tries to avoid assignment

```
function note( event )
{
   alert("Clicked");
}
/* For all list items add a click listener */
function selectExample()
{
   $("li").click( note );
}
// for all list items remove that listener we added
function removeSelectExample()
{
   $("li").off("click","",note);
}
```

Toggles Off

JQUERY

- Selectors: Like CSS but in Javascript and saves you a lot of time
  - $ is a function
  - They let you operate on multiple objects
  - JQuery also tries to avoid assignment

```
function hideIt()
{
   $("li").hide();
}
function showIt()
{
   $("li").show();
}
```

Hide SHow

JQUERY

- Comes with some animations as well

```
function hideIt2()
{
   $("li").fadeOut();
}
function showIt2()
{
   $("li").fadeIn();
}
```

Hide SHow

JQUERY

- You can replace elements as well

```
function censor()
{
   $( "button.censor" ).html("CENSORED");
}


<button class="censor" onclick="javascript:censor()"> I think _____
  about ____ </button>
```

I think _____ about ____

AJAX

- Asynchronous JavaScript and XML
    - Client Side
    - Allows Javascript to make HTTP requests and interpret the results without redirecting the browser.
    - Enables heavy-clients and lightweight webservices
    - Can be used to avoid presentation responsibility on the webservice.
    - JSON is a common replacement for XML
    - Twitter.com is heavy on Ajax

AJAX

- Disadvantages of AJAX ridden websites
  - History and Back button
  - Bookmarks
  - Browser Portability
  - Same origin policy -- Ajax is not cross domain.

AJAX

```
function getSomeJSON() {
   var xhr = new XMLHttpRequest();
   xhr.open('GET', 'some.json');
   // This is a call back
   xhr.onreadystatechange = function(){
       // readystate tells you how the transfer is going
       // 4 is done
       if( xhr.readyState === 4 ){
           // This is the HTTP Code
           if(xhr.status === 200){
               alert(xhr.responseText);
           } else {
               alert("There was an error " + xhr.status);
           }
       }
   };
   // finally send it
   xhr.send(null);
}
```

getSomeJSON()

AJAX

Let's make a generic GET

```
function getJSON( url, successCallback ) {
   var xhr = new XMLHttpRequest();
   xhr.open('GET', url);
   // This is a call back
   xhr.onreadystatechange = function(){
       // readystate tells you how the transfer is going
       // 4 is done
       if( xhr.readyState === 4 ){
           // This is the HTTP Code
           if(xhr.status === 200){
               successCallback( xhr.responseText );
           } else {
               alert("There was an error " + xhr.status);
           }
       }
   };
   // finally send it
   xhr.send(null);
}
```

AJAX

Now let's try to do something dynamic! With callbacks

window.setInterval Lets run a function at a set interval.

window.setTimeout Lets you run a function after a period of time.

```
var myInterval;
function startGetting() {
    myInterval = window.setInterval( function() { //callback
        var now = new Date();
        var s = 1 + (now.getSeconds() % 3);
        var url = s + ".json";
        getJSON( url, function( ourJSON ) { //another
            $("#ajaxy").text( ourJSON );    //callback
        });
    },1000); // 1 second or 1000 ms
}
```

```
<div id="ajaxy">AJAXY</div>
```

Start Getting

AJAXY

AJAX + JSON + JQUERY

Now let's try to do something dynamic! With callbacks

window.setInterval Lets run a function at a set interval.

window.setTimeout Lets you run a function after a period of time.

```
function startGettingJQuery() {
    var myInterval = window.setInterval( function() { //callback
        var now = new Date();
        var s = 1 + (now.getSeconds() % 3);
        var url = s + ".json";
        $.getJSON( url, function( data ) {
            // JSON Parsing
            $("#ajaxy2").text( data.message );
        });
    },1000); // 1 second or 1000 ms
}
```

```
<div id="ajaxy2">AJAXY2</div>
```

Start Getting

AJAXY2

JSON

Strict subset of Javascript see **http://json.org** for details

- JSON.parse -- Parses a JSON text
- JSON.stringfy -- Turns an object into a JSON string

```
function hotDogs() {
    var obj = { "food":"hotdog",
                "condiments":["ketchup","mustard","cheese"],
                "sausage":"weiner"
    };
    $("#hotdog").text( JSON.stringify( obj, null, " " ) ); //pretty print
    var newObj = JSON.parse($("#hotdog").text());
    $("#sausage").text( newObj.sausage );
}


<div id="sausage">sausage</div>
<pre><span id="hotdog">hotdog</span></pre>
```

Parse Some JSON!

sausage

```
hotdog
```

LICENSE

LICENSE

The source code to this slide deck is: