Intro to JavaScript

JavaScript History

- Client (generally browser-side) language invented at Netscape under the name LiveScript around 1995
- Netscape wanted a lightweight interpreted language to complement Java, similar to VB Script for Microsoft
- Name changed to JavaScript with introduction of Java to the browser; lots of confusion since between the two
- Standardized version is ECMAScript
- Now owned by Oracle

Adding JavaScript to a web page

- Inline code – place between <script> tag

```
<script type="text/javascript">
function inlineScript()
{
  var message = "Hello";
  alert(message);
}
</script>
```

Generally placed in the header
or at end of the <body> tag if you want
HTML to be rendered first before
processing the JavaScript

External JavaScript file

- Use the src attribute to reference an external JavaScript file

```
<script type="text/javascript"  src="demo.js">
```

Inside demo.js:

```
function printMessage()
{
  var message = "Hello";
  alert(message);
}
</script>
```

Variables

- Loosely typed; **case-sensitive - must be careful with case or programs will not work**
- Declaring a variable

```
var varName;     // Semicolon optional but recommended
```

- Primitive Data Types
  - Number (float and integer)
  - String (Can use “” or ‘’)
  - Boolean
  - Undefined
  - Null

```
var x = 10;
x = 3.5;
x = "hello";
alert(x);
```

Operators

- Usual arithmetic operators
  - +,-,*,/,%
- Convert strings to numbers using parse
  - var num = parseInt(“100”);
  - Var num2 = parseFloat(“3.5”);
- String concatenation with + and types converted automatically

```
var myString = "hello" + 23 + " there" + 10.5;
alert(myString);
```

Booleans

- true and false defined
- We also have “truthy” values
  - Numbers: 0 and NaN considered false, others considered true
  - Strings: “” considered false, others considered true
  - Null and Undefined considered false

  - Undefined is the value assigned to a declared variable with no value
  - Null is used to set an object to nothing

Determining the data type

- typeof variable 	returns the data type as a string

```
var myString = "hello";
alert(typeof myString);                    // String
```

Defining functions

```
function isEven(num)                // Separate multiple parameters by ,
{
        if ((num % 2) == 0)
                return true;
        else
                return false;
}

alert(isEven(3));
alert(isEven(4));
```

Functions are First Class

- We can assign a function to a variable using a function expression; we will use a similar format when we create our own objects

```
var myFunc = function(x)
{
        var z = x;
        z++;
        return z;
}

alert(myFunc(10)); // Outputs 11
```

```
var myFunc = function(x)
{
        var z = x;
        z++;
        return z;
}

function foo(f)
{
  alert(f(10));
}

foo(myFunc);
```

```
function addOne()
{
  return function(x)
  {
    return x+1;
  }
}

var f = addOne();
alert(f(10));
```

Operators

- Usual relational operators
  - ==, !=, <, >, <=, >=
- Also have Identity and Nonidentity
  - === and !==
  - True or false if values AND types match
- Logical Operators
  - &&, || , !

Control Statements

- if statement/ if else and switch statement the same as Java
- The ternary operator is also available
  - (condition) ? (return value if condition true) : (return value if condition false)
- While loop, do-while and for loop also the same
  - for (var i = 0; i < 100; i++)

Scoping

- Global scope for identifiers declared at the topmost level
- Local scope for identifiers declared inside a function
- Lexical/static scope for functions

```
var x = 10;

function foo1()
{
  var x = 20;   // Local variable x;
  foo2();
}

function foo2()
{
  alert("In foo 2 x= " + x);     // Outputs 10
}


alert("Outside: " + x);
foo1();
foo2();
alert("Outside: " + x);
```

No Block Scope

- If a variable is declared in a block, it is added to the scope of that block (e.g. global or a function) and does not go out of scope when the block exits

```
if (1 < 2)
{
  var x = 10;
}

alert("Outside: " + x);          // Legal and x = 10 here
```

Variable Declarations

- If a variable is assigned a value but no such variable exists then it is created with global scope

```
function foo()
{
   x = 10;
}

foo();
alert("Outside: " + x);
```

```
function foo()
{
   var x = 10;
}

foo();
alert("Outside: " + x);
```

Arrays

- Arrays are an object type

- var myArray = new Array();
- var myArray = [];   // Same thing
- var myArray = [1, 2, 3];

- myArray.length   // Access length of array
- myArray[0] = 3;
- alert(myArray[2]);
- myArray.push(val);   // Add to end

Useful array methods:

concat(a1,a2) – concatenates arrays
join(separator) – Converts to string
pop() – Removes last element
reverse() - Reverses array

String methods

- Strings also an object (along with Date, Number)
- Some methods:
  - charAt(index)
  - indexOf(searchString)
  - replace(toReplace, replaceWith)
  - split(separator)
  - substr(startIndex, numChars)
  - toLowerCase()
  - toUpperCase()

```
var myString = "10,20,30,40";
var myArray = myString.split(",");
alert(myArray.join(":"));
```

Creating an object

```
var myObj = new Object();           // Calls the object constructor
myObj.x = 10;                  // Creates properties on the fly
myObj.y = 100;
```

- Can also use object literal notation:

```
var myObj = {
        x : 10,
        y: 20
};
```

- Objects can be returned or sent in as a parameter

Custom Objects

- Use a function that serves as a constructor with properties and sub-functions (methods)

```
function Point(x, y)
{
  this.x = x;                    // Public instance vars
  this.y = y;
  var d = 10;                    // Internal "private" variable
  this.getDistance = function(p)          // A method
  {
          var x2 = Math.pow(p.x - this.x, 2);
          var y2 = Math.pow(p.y - this.y, 2);
          d = Math.sqrt(x2 + y2);
          return d;
  };
}

var point1 = new Point(0,0);
var point2 = new Point(1,1);
var distance = point1.getDistance(point2);
alert(distance);                                    // 1.41
```

One flaw: has to create all of the Point items (x, y, d, getDistance) for every Point created

OK for vars but not for getDistance

Prototype Object

- Avoid the problem on the previous slide with the prototype object
- Every function has a **prototype** property
  - Change made to the prototype object is seen by all instances of that data type
  - Similar to declaring something static; can create a member function once and assign it to the prototype property

```
function Point(x, y)
{
  this.x = x;
  this.y = y;
}
Point.prototype.getDistance  = function(p)
  {
          var x2 = Math.pow(p.x - this.x, 2);
          var y2 = Math.pow(p.y - this.y, 2);
          return Math.sqrt(x2 + y2);
  };
```

```
var point1 = new Point(0,0);
var point2 = new Point(1,1);
var distance = point1.getDistance(point2);
alert(distance);
```

For “static” variables could use Point.val = 100;

Overriding the prototype

- Possible to override the prototype based on scoping rules

```
function Point(x, y)
{
  this.x = x;                     // Public instance vars
  this.y = y;
}
Point.prototype.getDistance = function(p)
  {
                var x2 = Math.pow(p.x - this.x, 2);
                var y2 = Math.pow(p.y - this.y, 2);
                return Math.sqrt(x2 + y2);
  };


var point1 = new Point(0,0);
var point2 = new Point(1,1);
point1.getDistance = function(p)
{
  return 10;
};
var distance = point1.getDistance(point2);
alert(distance);
```

Inheritance

- Inheritance is possible using the call() method which invokes a method as though it were a method of the object specified by the first parameter

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

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

function Employee(first,last,position)
{
                Person.call(this, first, last);
                this.position = position;
}
```

```
var eddie = new Employee("Eddie","Haskell","Schmoozer");
alert(eddie.firstName + " " + eddie.lastName + " " + eddie.position);
```

But this doesn’t work yet:
alert(eddie.getFullName() + " " + eddie.position);

Inheritance via the prototype

- We can chain an instance of a base type to the sub-type's prototype and get inheritance as JavaScript searches through the chain for a matching method

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

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

function Employee(first,last,position)
{
        Person.call(this, first, last);
        this.position = position;
}
```

```
Employee.prototype = new Person();

var eddie = new Employee("Eddie","Haskell","Schmoozer");
alert(eddie.getFullName() + " " + eddie.position);
```

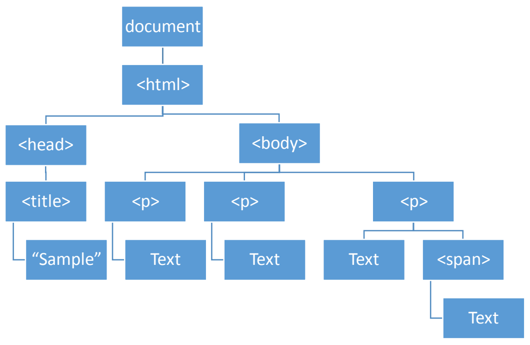
Programming the Browser

- Already using alert() but more formally it is window.alert(message);
- Also:

```
var name = prompt("Enter your name", "Default Text");
var result = confirm("Ready to end class?");   // Shows OK/Cancel
location.href = "http://newURL";
location.reload(true);
var childwindow = open("URL","Title","height=1024, width=400");
```

Navigating the Document Object Model (DOM)

```
<html>
<head>
<title>Sample</title>
</head>
<body>
<p>Some text</p>
<p>More text</p>
<p>Text with a <span>span element</span></p>
</body>
</html>
```



Nodes are Document, Element, Attribute, Text

Navigating the Document Object Model (DOM)

```
<html>
<head>
<title>Sample</title>
</head>
<body>
<p>Some text</p>
<p>More text</p>
<p>Text with a <span id="foo">span element</span></p>
</body>
</html>
```

Several ways to access elements, one is by ID:

```
var span = document.getElementById("foo");
var txt = span.childNodes[0].nodeValue;
span.childNodes[0].nodeValue = "twist";
```

Creating Elements

- Creating a div (division or section) and text elements:

```
var el = document.createElement("div");
el.id = "myDiv";
var text = document.createTextNode("Hello world!");
el.appendChild(text);
el.setAttribute("align","center");                    // Set attributes
document.body.appendChild(el);  // Adds to end of the body
```

Adding multiple DIV’s

```
var divs = Array();
var i;
for (i = 0; i < 10; i++)
{
        divs[i] = document.createElement("div");
        divs[i].id = "myDiv" + i;
        var text = document.createTextNode("Hello world!");
        divs[i].appendChild(text);
        divs[i].setAttribute("align","center");                    // Set attributes
        document.body.appendChild(divs[i]);  // Adds to end of the body
}
```

Can set properties using innerHTML

```
var el = document.getElementById("foo");
el.innerHTML = "<B>New Value</B>";
```

Some say this is bad because it can lead to invalid markup

Accessing Forms

- Example:

```
<html>
<head>
<title>Javascript Demo</title>
</head>
<body>

<p>Text with a <span id="foo">span element</span></p>

<form name="theForm" action="">
<input type=text name="myTextBox" id="text1" value="4">
</form>

<script type="text/javascript" src="demo.js">
</script>

</body>
</html>
```

```
var el = document.getElementById("text1");
el.value = parseInt(el.value) + 1;

el = document.theForm.myTextBox;
el.value = parseInt(el.value) + 1;
```

Scripting Buttons – Click Event

```
<html>
<head>
<title>Javascript Demo</title>
</head>

<body>
<form name="theForm" action="">
<input type=text id="text1" value="2">
<button type="button" id="button1" onClick="addOne()">Click Me</button>
</form>

<script type="text/javascript" src="demo.js">
</script>

</body>
</html>
```

```
function addOne()
{
        var el = document.getElementById("text1");
        el.value = parseInt(el.value) + 1;
}
```