JavaScript Basics

CSE/ISE 102: Intro to Web Design
Stony Brook University

What is JavaScript?

- A *client-side* scripting language
- Common tasks performed by JavaScript in Web pages:
  - asking the browser to display information
  - making the Web page different depending on the browser
  - monitoring user events & specifying reactions
  - generating HTML code for parts of the page
  - modifying a page in response to events
  - checking correctness of input

JavaScript background

- Developed by Netscape, released 1995
- Supported by all major browsers
- The standard client-side scripting language
- JavaScript is not Java
  - two different languages
  - JavaScript ***is*** object oriented
- JavaScript programs are:
  - embedded inside Web pages
  - executed by browsers

JavaScript Reference

- W3Schools:

  http://www.w3schools.com/js/default.asp

Using JavaScript

- Embedded scripts
  - included directly inside a Web page in `<script>` tags
- External scripts
  - Can be shared by multiple pages
  - `<script>` tag includes a script URL

Script Placement

- Can go anywhere in the document
- Normally placed in the head or at the end of the body
  - it's recommended to place them just before the </body> tag

`<script>` Tags

- Embedded:

```
<script> JS Code </script>
```

- External:

```
<script src="myScript.js"></script>
```

JavaScript Basics

- JavaScript is case-sensitive
- Whitespace is ignored, unless part of a string (and enclosed in quotes)

Statements and Comments

- Statement: a command that tells the browser what to do
  - e.g., `alert("Thank you.");`
  - statements end with a semicolon
- Comment: explains the code, but is ignored by the computer
  - starts with // and continues to end of line

Variables

- A variable is a container for information
- Start by declaring a variable using `var`
- names must start with letter or underscore
  - can contain digits, but not spaces
- Use = to assign a value to a variable
- e.g., `var foo = 5;`

Data Types

- Variable values fall into several categories
  - undefined
  - null (no inherent value)
  - numbers
  - strings (enclosed in quotes)
  - booleans ('true' or 'false')
  - arrays (square bracket-ed list)

Comparison Operators

- == (is equal to)
- != (is not equal to)
- === (is identical to: equal and of same type)
- !== (is not identical to)
- `>  >=  <  <=`

Equal vs. Identical

- All values fall into one of several data types
- Some values may be similar, but not of the same type
  - e.g., 5 and "5" are equal, but not identical
  - "5" and "5" are equal and identical

Mathematical Operators

- += (adds the value to itself)
- ++ (increases value by 1)
- -- (decreases value by 1)

If/Else Tests

- if (<condition>)
  {
  // Stuff to do
  }
  else // optional
  {
  // alternate action(s) to take
  }

Loops

- for (*initialization* ; *test* ; *update*)
  {
  // Do something
  }
- e.g.,

```
for (var i = 0; i < 3; i++)
{
    alert(i);
}
```

Functions

- A block of code that runs when it is referenced
- Functions can take *arguments* (data) that affects how the function operates
  - commas separate multiple arguments

Predefined Functions

- alert(), confirm(), prompt()
- Date() — returns current date/time
- parseInt() — converts a String to a number
- setTimeout (functionName, delay)
  - executes a function after a delay (in ms)

Your Own Functions

- Syntax:

```
function name ( arguments )
{
   // function code goes here
}
```

Returning a Value

- Use the return keyword in a function to:
  - end that function immediately
  - send back a specified value
- e.g., `return arr.length;`

Variable Scope

- Defines where in a script a particular variable is available for use
  - global: can be used anywhere
  - local: only usable within parent function

Scope Rules

- If a variable is declared outside of a function, it is globally-scoped
- If a variable is declared inside a function:
  - use "var" to make it local
    e.g., var foo = "value";
  - omit "var" to make it global

Scope Collisions

```
function double (num)
{
   total = num + num;
   return total;
}
var total = 10;
var number = double(20);
alert(total); // returns 40, not 10
```

Random Values

- Use Math.random() for random numbers
  - returns a random value between 0 and 1
- Multiply result by size of desired range
- Use Math.floor() to convert value to integer

```
var m = Math.floor(Math.random() * 10 + 1);
```

JavaScript Arrays

- Ordered list of data
  - Stores several values under one name
    - types can be mixed
- Can be resized dynamically
- Can be created in two ways:

Creating Arrays, v1

- Use the Array constructor:
  - var colors = new Array();
  - var colors = new Array(20); // size 20
  - var colors = new Array("red", "blue", green");

Creating Arrays, v2

- Use *array literal* notation:
  - var names = [ ];
  - var colors = ["red", "blue", "green"];

Array Manipulation

- Use `length` to get the number of values
  - var size = myArray.length;
- Use bracket notation to get/set values:
  - Note that indices start at 0
  - var next = nyArray[3]; // 4th element
  - myArray[0] = 45; // set first element

Push() and Pop()

- push() adds values to the end of an array
  - colors.push("brown", "orange");
  - returns the new array length
- pop() removes/returns last element
  - var item = colors.pop();

shift() and unshift()

- shift() removes/returns first item in an array
  - var item = colors.shift();
- unshift() adds any number of items to the front of an array, and returns new length
  - count = colors.unshift("black");

Reordering Arrays

- `reverse()` changes an array by flipping the order of its contents
- `sort()` orders the elements of an array in ascending order
  - works based on string equivalents
  - fix this by passing in your own `compare()` function as an argument (returns +, -, 0)

Array Searching

- `indexOf()` and `lastIndexOf()` return the first/last index of a given value
  - first argument: value to look for
  - second argument: starting index (optional)
  - returns -1 if not found

String Operations

- length — number of characters in the string
- toUpperCase()/toLowerCase()
- trim() — removes leading/trailing spaces
- charAt(i) — returns character at index i
- concat(str) — get new combined string
- indexOf() and lastIndexOf()

Extracting Substrings

- slice(), substr(), substring()
  - one argument: starting point of extraction
  - second argument (slice and substring()) = stopping point
  - second argument (substr()) = # of characters to extract

The Browser Object

- JS lets you access and manipulate parts of the browser window

property/method	description
`event`	state of an event
`history`	list of visited URLs
`location`	access URI in status bar
`status`	set status bar text
`alert()`	displays an alert box
`close()`	closes current window
`confirm()`	displays dialog w/ OK, Cancel buttons
`focus()`	sets focus on current window

Common Events

- `onblur`
- `onchange`
- `onclick`
- `onerror`
- `onfocus`
- `onkeydown`
- `onkeypress`
- `onkeyup`
- `onload`
- `onmousedown`
- `onmousemove`
- `onmouseout`
- `onmouseover`
- `onmouseup`
- `onsubmit`

Applying Event Handlers

- Three techniques:
  1. as an HTML attribute
  2. as a method attached to the element
  3. using `addEventListener`
- Last 2 require the `window` object

Event Handlers 1

- As an HTML attribute:

```
<body onclick="myFunction();">
```

- This is antiquated, and should be avoided, just like inline CSS styles
  - blurs line between semantic and behavioral layers

Event Handlers 2

- As a method:

```
window.onclick = myFunction;
```

- We can only bind one event at a time

```
window.onclick = myFunction;
// someOtherFunction replaces myFunction
window.onclick = someOtherFunction;
```

Event Handlers 3

- addEventListener

```
window.addEventListener("click", myFunction);
```

- Omit "on" from the event handler here

Using JavaScript

Meet the DOM

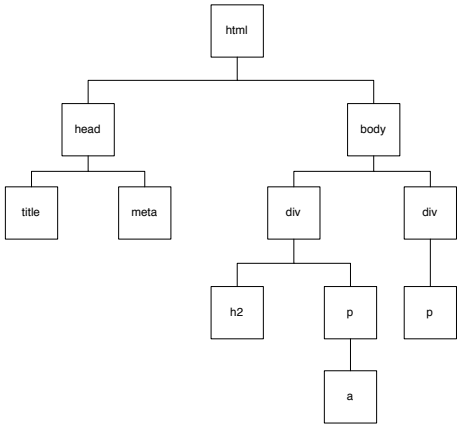
- DOM = Document Object Model
  - allows us to access, manipulate the contents of a document
  - structured map of an HTML document
- We can use JavaScript to access and manipulate everything via the DOM

DOM Contents

- Each page element is called a *node*
- The DOM is made up of 3 types of nodes:
  1. element nodes
  2. attribute nodes
  3. text nodes

```
<html>
<head>
   <title>Document Title</title>
   <meta charset="utf-8">
</head>
<body>
   <div>
      <h2>Subhead</h2>
      <p>Text with a <a href="foo.html">link</a>.</p>
   </div>
   <div>
      <p>More text here.</p>
   </div>
</body>
</html>
```

The Node Tree



Accessing DOM Nodes

- `document` object identifies the page itself
  - has a set of standard properties and methods
  - navigate to the element we're after by chaining properties and methods together
    - use periods to separate parts of a route
  - There are several ways to access nodes...

By Element Name

- Use document.getElementsByTagName()
- e.g., document.getElementsByTagName("p") returns a collection (nodeList) of every paragraph on the page, in order
  - access specific elements by index, like an array: paragraphs[0], etc.
  - Use "*" to get every document element
- use namedItem() to get a specific named item

By id Attribute Value

- `getElementById()`
- returns a single element

```
var photo =
document.getElementById("lead-photo");
```

By class or Selector

- `getElementsByClassName()`
  - returns a nodeList
- querySelectorAll() (also returns a nodeList)

```
var s = document.querySelectorAll(".sidebar p");
```

Accessing an Attribute Value

- Use the `getAttribute()` method
  - supply the attribute name as an argument

```
bigImage = document.getElementById("lead-image");
alert(bigImage.getAttribute("src");
```

Manipulating Nodes

- The DOM provides several built-in methods for manipulating elements, their attributes, and their contents

`setAttribute()`

- Changes the value of an attribute
- Takes two arguments:
  - name of attribute to be changed
  - new attribute value
- e.g., bigImage.setAttribute("src", "foo.jpg");
- Can be used for many things

`innerHTML()`

- Lets us access and change the text and markup inside an element

```
var inDiv = document.getElementsByClassName("intro");
inDiv[0].innerHTML = "<p>This is our text.</p>";
```

`style`

- Add, remove, or modify an element's CSS style
- Works similar to inline styling

```
document.getElementById("intro").style.color = "#fff";
```

- Hyphenated CSS property names (e.g., `background-color`) are converted to camel case (`backgroundColor`)
- `style` can also retrieve a style value

Adding and Removing Elements

- The DOM also allows us to change the document structure on the fly by adding and removing nodes
- This is a more precise technique than adding content with `innerHTML`

Creating Elements

- **Note:** this creates a *reference* to an element; it doesn't add the element (yet)!
- createElement()
  - argument is the new element to create

```
var newDiv =
document.createElement("div");
```

Entering Text

- Use createTextNode() to enter text into an element we've created
- var t = document.createTextNode("blah");

Putting Things Together

- Use appendChild() to add a node to an existing element
  - call this method on the parent node
  - argument is the node to add

```
// Get source/parent element
var ourDiv = document.getElementById("our-div");

// Create our new nodes
var newPara = document.createElement("p");
var copy = document.createTextNode("Hello");

// Set up the new paragraph node
newPara.appendChild(copy);

// Add it to the parent element
ourDiv.appendChild(newPara);
```

Adding to Existing Elements

- insertBefore()
  - first argument: node to be inserted
  - second argument: element it gets inserted in front of
- ourDiv.insertBefore(newHeading, para);

Replacing Nodes

- replaceChild() replaces one node with another
  - first argument: new child to insert
  - second argument: node to be replaced
- ourDiv.replaceChild(newImage, swapOut);

Removing Elements

- Removes a node or an entire branch from the document tree
  - needs references to parent and element to be removed
- parent.removeChild(removeMe);

`document` Object Properties

- title — text in title bar
- url — get complete document URL
- domain — get just the document domain

JavaScript Libraries

JavaScript Libraries

- You don't have to write everything yourself
- There are many prewritten collections (*libraries*) of functions and methods that you can use in your scripts

Some Useful Libraries

- jQuery
  - spinoffs include jQuery UI and jQuery Mobile
- Dojo — for developing Web apps with Ajax
- Prototype — adds Ajax support to Ruby on Rails
- MooTools
- Yahoo! User Interface Library (YUI)

Using a Library

1. Download the JavaScript (.js) file
2. Set your script tag to point to the library
3. Go ahead and call the library's functions