University of Regina

CS 215
Web Oriented Programming

JavaScript Fundamentals

Dr. Orland Hoeber
orland.hoeber@uregina.ca
http://www.cs.uregina.ca/~hoeber/cs215/

Readings

- Chapters 13 - 16

Origins & Evolution of JavaScript

- Originally developed by Netscape
- Became a joint venture between Netscape and Sun in 1995

- A language standard was developed by the European Computer Manufacturers Association (ECMA-262)

- The official name of the language is ECMAScript, but most still call it JavaScript

Three Elements of the Language

- Core JavaScript
  - core programming constructs such as operators, expressions, statements, sub-programs
- Client-Side JavaScript
  - collection of objects that support the control of a browser and interaction with users
- Server-Side JavaScript
  - collection of objects that make the language useful on a Web server

Java and JavaScript

- The only relation between Java and JavaScript is the similarity in syntax
  - JavaScript is dynamically typed
  - JavaScript is object-based, but not object-oriented
    - has objects, but these are not proper classes
    - no class-based inheritance or polymorphism
    - the objects are simply collections of properties and functions (more like structs in C than classes in Java)

Browsers and JavaScript

- When JavaScript is embedded within an HTML document
  - the browser executes the JavaScript at that point
  - the output of the JavaScript is inserted in the document
  - the browser will continue to render the document when the execution is complete
- JavaScript can be embedded both in the head and body of the document
  - head: code that will produce content only when requested (e.g., function definitions)
  - body: code that is to be executed once at the specific location

Embedding JavaScript in HTML

- There are two methods:
  - explicit embedding
    - JavaScript code explicitly resides in the HTML document

```
<script type="text/javascript">
…
</script>
```

  - implicit embedding
    - JavaScript code is stored in a separate file and linked to the HTML document in the desired location (head or body)

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

Problems with Explicit Embedding

- There are two problems with explicit embedding
  - some browsers do not recognize the <script> tag and display the contents as document content
  - embedded JavaScript can confuse XHTML validators

```
<script type="text/javascript">
<!--
    document.write("Hello World.");
// -->
</script>
```

- The best solution to avoid such problems is to use implicit embedding
  - also allows for the maintenance of the JavaScript code separate from the HTML markup and easier re-use of code

JavaScript and Semicolons

- The JavaScript interpreter tries to make semicolons unnecessary
  - built in the philosophy of forgiving Web browsers
  - this can cause all kinds of problems if you don't put in the semicolons yourself, and when you break lines of code for readability

- Solution:
  - always terminate statements with semicolons
  - when spreading a statement over multiple lines, ensure that the incomplete lines do not form complete statements

Primitive Types

- There are five primitive data types in JavaScript
  - Number
  - String
  - Boolean
  - Undefined
  - Null

- The first three also have wrapper objects (with the same name as the data type) that provide properties and methods for operating on these data types
- Because the language coerces values between the primitive type and the associated object type automatically, you can treat the primitives as if they were objects of the same type

Primitive Literals

- Numeric literals
  - represented internally in double-precision floating-point form
    - no difference between integers and floating-point numbers
- String literals
  - zero or more characters encapsulated in either single or double quotes
    - escape sequences use the back-slash (e.g., \n, \t, \’, \\)
- Boolean literal
  - can only take values of true or false
- Null literal
  - a variable that has not been explicitly declared or assigned a value
- Undefined literal
  - a variable that has been declared but not assigned a value

Variable Declaration

- Variables can be either implicitly or explicitly declared
  - implicit declaration (assign it a value)

```
temp = x;
```

  - explicit declaration (var statement)

```
var temp = x;
var a, b;
```

- A good programming practice is to always explicitly declare your variables

Numeric Operators

- All of the numeric operators that you expect are present: +, -, /, *, %, ++, --
  - rules for prefix and postfix unary operators are the same as expected
  - rules for precedence and associativity are the same as all C-derived programming languages

```
var a=2, b=4, c, d;
c = 3 + a * b;
// * first, so c==11 (not 24)
d = b / a / 2;
// associates left, so d==1 (not 4)
```

Objects to Support Numeric Computing

- There are two objects to help with numeric computing
  - Math
    - trigonometric functions (sin, cos, etc.)
    - other common mathematical functions (floor, round, max, etc.)
  - Number
    - useful properties of numbers and constants (MAX_VALUE, MIN_VALUE, PI, etc.)
    - functions for formatting numbers (toFixed, toPrecision, toString)
- Use dot notation to access the functions and properties

```
var x = Number.MAX_VALUE;
var y = Math.cos(x);
y = y.toFixed(3);
```

NaN

- Any arithmetic operation that results in an error (e.g., division by zero) or produces a value that cannot be represented as a double-precision floating-point number becomes NaN
  - stands for Not a Number
  - any comparison against a number or another NaN fails
  - must use isNaN() function to test for these

Coercions

- The string and number literals can automatically be coerced to the other type
  - string concatenation (+) coerces numbers to strings
  - numeric operations (other than +) coerce strings to numbers

```
“August “ + 1977
1977 + “ August”
7 * “3”
7 + “3”
```

Explicit Type Conversions

- String and Number constructors

```
var str_value = String(value);
var num_value = Number(value);
```

- toString method

```
var num = 6;
var str_value = num.toString();
```

- parseInt and parseFloat methods

```
var str = “6.23 metres”;
var int_value = parseInt(str);
var float_value = parseFloat(str);
```

String Properties and Methods

- One property and a bunch of methods to support string processing
  - property
    - length e.g., var len = str1.length;
  - methods
    - charAt(position) e.g., str.charAt(3)
    - indexOf(string) e.g., str.indexOf('B')
    - substring(from, to) e.g., str.substring(1, 3)
    - toLowerCase() e.g., str.toLowerCase()

  - all indexing starts at position zero

Date Object

- Since performing calculations on dates can be somewhat complex, JavaScript includes a built-in Date object
  - created without parameters makes it the current date/time

    ```
    var now = new Date();
    ```

  - methods to extract information
    - toLocaleString – returns a string of the date
    - getDate – returns the day of the month
    - getMonth – returns the month of the year (0 – 11)
    - getDay – returns the day of the week (0 – 6)
    - getFullYear – returns the year
    - getTime – returns the number of milliseconds since January 1, 1970
    - getHours – returns the hour (0 – 23)
    - getMinutes – returns the minutes (0 – 59)
    - getMilliseconds – returns the millisecond (0 – 999)

Basic Output

- The output from JavaScript can be done at the Document or Window object levels
  - at the Document object level, the output is inserted into the HTML document at the appropriate location

```
…
<body>
…
<script type="text/javascript">
<!--
var x = Math.sqrt(3);
document.write("<p>The square root of 3 is" + x.toFixed(5) +
"</p>");
// -->
</script>
…
</body>
…
```

Basic Output

- At the Window object level, there are three methods
  - alert
  - confirm
  - prompt
- Since the default object for JavaScript is the Window object currently being displayed, we can simply use these methods as if they were built-in functions

```
var x = Math.sqrt(3);
alert(“The square root of 3 is” + x.toFixed(5));
```

Basic Output

- These are not the best ways to get output from JavaScript to the user
  - using document.write requires that you insert HTML code into your JavaScript output
    - difficult to debug
    - difficult to validate as XHTML
  - using alert, confirm, and prompt produce pop-up windows
    - modal (must respond)
    - no control over style or formatting
- There is a much better way that creates or overwrites HTML elements that we'll see next week

Control Expressions

- The control expressions and statements are similar to their counterparts in Java and C/C++
- Expressions
  - primitive
    - numbers are true, unless they are zero
    - strings are true, unless they are empty or “0”
  - relational operators
    - includes the usual six: ==, !=, <, <=, >, >=
    - special ones: === and !==
      - for these, no coercions are performed
      - only are true if the value and type are the same

Control Expressions

- Compound Expressions
  - the usual compound operators apply: &&, ||, !
  - && and || are short-circuit operators
- Operator Precedence
  - the precedence of operators is what you would expect
  - see the textbook for details

Control Statements

- Selection Statement (if-then and if-then-else)

```
if (a < b) {
  document.write(“<p>a is less than b</p>”);
} else {
  document.write(<p>a is not less than b</p>”);
}
```

- Compound Selection Statement (switch)

```
switch (x) {
  case “a”:
          document.write(“<p>x is a</p>”);
          break;
  case “b”:
          document.write(“<p>x is b</p>”);
          break;
  default:
          document.write(“<p>x is neither a or b</p>”);
}
```

Loop Statements

- While

```
x = 5;
while (x > 0) {
  document.write("<p>x = " + x-- + "</p>");
}
```

- For

```
for (count = 0; count < 5; count++) {
  document.write("<p>count = " + count + "</p>");
}
```

- Do-While

```
var count = 0;
do {
  count ++;
  sum = sum + count;
} while (count < 5);
```

Arrays

- Arrays are objects with special functionality
- Can be created with a new operator, or directly with literal array values

```
var my_list = new Array (1, 2, “three”, “four”);
var my_list = new Array (200);
var my_list = [1, 2, “three”, “four”];
```

- Arrays are indexed from 0 (like String)
- Arrays are dynamically sized as elements are added
- Elements are accessed using square brackets, and the length is read/write

```
my_list[47] = 200;
var y = my_list.length;
my_list.length = 5;
```

Arrays Methods

- There are a collection of useful methods included with the Array objects
  - join(“ : ”);
    - converts all objects to strings and joins them together using the parameter string as the separator
  - reverse();
    - reverses the order of the elements in the array
  - sort();
    - sorts the elements of the array alphabetically
  - concat(a, b);
    - concatenates additional parameter elements to the end of the array
  - slice(x, y);
    - returns a portion of the array from x to y
  - toString();
    - converts all objects to strings and returns these in a comma-separated list
  - push(a); pop();
    - add/remove elements from the end
  - shift(a); unshift();
    - add/remove elements from the beginning

Functions

- Functions are defined in a similar way to other C-based languages

```
function my_function (a, b) {
  return (a + b);
}
```

- If there is no return, the return has no parameter, or the end of the function is reached, then undefined is returned
  - in these cases, the function can be called without assigning it to a value:

```
format_data(my_data);
```

Function Parameters

- All function parameters are pass-by-value
  - because Object variables are actually references, passing in objects performs the pass-by-value on the reference
  - this means that Objects (and Arrays) that are used as parameters are pass-by-reference

- There is no type-checking of the parameters
- There is no checking of the number of parameters passed in the function call
  - if more, extras are ignored
  - if less, missing parameters are undefined
  - arguments.length can be used to verify the correct number

Functions as Parameters

- Previously, I said that Array.sort will perform alphabetical sorting of the array
- This default behaviour can be overwritten by passing in a sorting function
  - special function that compares values and returns a negative value, 0, or a positive value representing the sort order of the two objects

```
function num_order(a,b) {
  return b – a;
}
…
var num_list = [3, 7, 2, 14, 9];
num_list.sort(num_order);
```

Object Creation

- Objects can be created with a **new** expression
  - results in a call to a constructor method
  - the constructor creates the properties that characterize the new object
  - in order to use custom objects, the work is in creating the constructor (which we'll see soon)

- Accessing properties: dot or array notation
- New properties can dynamically be added to an object just by assigning a value to the property name
  - since the object is already created, you don't need to declare the new property using var

Object Creation and Deletion

- We can create a blank object using the built-in Object constructor

```
var my_object = new Object();
my_object.name = “Bob”;
my_object.age = 25;
…
document.write (my_object.name + “ is “ + my_object.age);
…
delete my_object;
```

Constructors

- Constructors are special functions that are used to create objects of the same name
  - use the reserved word **this** to reference the object

```
function Car(newMake, newModel, newYear){
  this.make = newMake;
  this.model = newModel;
  this.year = newYear;
}
```

  - must be called with **new**

```
var myCar = new Car (“Audi”, “A4”, 2014);
```

  - methods can also be added to the object by the constructor
    - if you have a function defined as displayCar();

```
this.display = displayCar;
```

Regular Expressions

- JavaScript includes powerful pattern matching via regular expressions
  - supported in two objects: RegExp and String
- The regular expressions in JavaScript are the same as in most other modern languages (derived from Perl)
  - describe patterns of strings
  - a pattern is delimited with slashes

```
/hello/
/.ell./
/[0-9]/
/[a-zA-Z]/
```

Pre-defined Character Classes

- Character classes are put in square brackets, with dashes indicating logical sequences of characters or numbers
- Match to a single character (unless told to do otherwise)
- Character classes can be inverted by starting it with a ^
- There is a set of useful pre-defined character classes:
  - . – any character
  - \d – a digit
  - \D – not a digit
  - \w – a word character
  - \W – not a word character
  - \s – a white space character
  - \S – not a white space character

```
/[a-zA-Z]\d[a-zA-Z]\s\d[a-zA-Z]\d/
/\w\w\w/
/[^abc]/
```

Quantifiers and Anchors

- There are three special quantifiers:
  - * -- zero or more repetitions
  - + -- one or more repetitions
  - ? -- zero or one

```
/x*y+z?/
/\d+.\d*/
/[a-zA-Z]\d[a-zA-Z]\s?\d[a-zA-Z]\d/
```

- There are two anchors:
  - ^ -- beginning of the string
  - $ -- end of the string

```
/^A/
/end$/
```

Pattern Modifiers

- Modifiers can be added after the closing / in the regular expression
  - i -- matches both upper and lower case
  - g -- matches all instances (global match)

```
/Apple/i
/[a-z]\d[a-z]\s?\d[a-z]\d/i

/The/g
```

Regular Expressions and String Object

- These regular expressions can be used in the String object
  - search (returns the index of the pattern)

    ```
    var str = "Rabbits are furry animals";
    var position = str.search(/bits/);
    ```

  - replace (replaces the pattern with a new string)

    ```
    str.replace(/bits/, "bots");
    str.replace(/\sa/g, " i");
    ```

  - match (extracts matches into an array)

    ```
    var str = "I have 4 apples and 3 oranges";
    var digit_array = str.match(/\d/g);
    var substr_array = str.match(/(\d)([^\d]+)(\d)/);
    ```

  - split (splits the string into an array based on the pattern)

    ```
    var split_array = str.split(/[a-c]/);
    ```

Debugging JavaScript

- Debugging JavaScript is not as easy as with other programming languages
  - depends on the browser
  - at the very least, you should configure your browser to show JavaScript errors
  - there are also some reasonably good debugging consoles (either built-in, or as add-ins)
    - IE – Tools/Developer Tools
    - Safari – Develop/Error Console
    - Firefox – FireBug add-in
- FireBug is a very useful add-on for Firefox that can support debugging JavaScript
- W3Schools also has some advice:
  - https://www.w3schools.com/js/js_debugging.asp

Homework

- Keep up with your readings on JavaScript
- Next topic: JavaScript, DOM, & Events
- The third assignment will be posted next week
  - due Thursday Oct 26 @ 11:55 PM
- The midterm is scheduled for Tuesday Oct 24