Chapter 2
JavaScript Core Features - Overview

Adapted from
JavaScript: The Complete Reference 2nd Edition
by
Thomas Powell & Fritz Schneider
© 2004 Thomas Powell, Fritz Schneider, McGraw-Hill

Basic Features

• Script Execution order
  - Top to bottom
  - `<head>` before `<body>`
  - Can't forward reference outside a `<script>` tag
• JavaScript is case sensitive
  - HTML is not, XHTML is
  - "Camelback" style `document.lastModified`
  - IE's JScript is a little less case sensitive than standard ECMAScript and Netscape's JavaScript
  - Remember `onClick`, `ONCLICK`, `onclick` doesn't count since that is HTML

Basic Features Contd.

• Whitespace
  - Whitespace is generally ignored in JavaScript statements but not always consider
    • `x = x + 1` same as `x = x + 1`
    • `s = typeof x;` is same as `s = typeof x;` but it not the same as `s = typeof x;` or `s = type of x;`
  - Return character can cause havoc
  - Given white space support by JavaScript some developers favor "crunching"
Basic Features Contd.

- **Statements**
  - A script is made up of individual statements
  - JavaScript statements are terminated by returns or semi-colons (;)
  - So $x = x+1; \text{ same as } x = x+1$
  - Prefer to use semi-colons because if you reduce returns you run into problems
  - $x=x+1 \text{ alert}(x)$ throws an error while $x=x+1; \text{alert}(x);$ does not.

Blocks

- To group together statements we can create a block using curly braces ({ }). In some sense this creates one large statement
- Blocks are used with functions as well as larger decision structures like if statements

```
function add(x,y)
{
    var result = x+y;
    return result;
}
```

Variables

- Variables store data in a program
- The name of a variable should be unique well formed identifier starting with a letter and followed by letters or digits
- Variable names should not contain special characters or white space
- Variable names should be well considered
  - X versus sum
  - Some rules of programming might not follow on the Web?
Variables Contd.

- Define a variable using the var statement
  ```javascript
  var x;
  ```
- If undefined a variable will be defined on its first use
- Variables can be assigned at declaration time
  ```javascript
  var x = 5;
  ```
- Commas can be used to define many variables at once
  ```javascript
  var x, y = 5, z;
  ```

Basic Data Types

- Every variable has a data type that indicates what kind of data the variable holds
- Basic data types in JavaScript
  - Strings ("thomas", 'x', "Who are you?")
    - Strings may include special escaped characters
    - Strings may contain some formatting characters
      - "There are some newlines \n\n\n and tabs \t\t\t yes!"
  - Numbers (5, -345, 56.7, -456.45677)
    - Numbers in JavaScript tend not to be complex (e.g. higher math)
  - Booleans (true, false)
- Also consider the values null and undefined as types

Weak Typing

- JavaScript is a weakly type language meaning that the contents of a variable can change from one type to another.
  - Some languages are more strongly type in that you must declare the type of a variable and stick with it.
- Example of dynamic & weak typing a variable initially holding a string can later hold a number
  ```javascript
  x = "hello"; x = 5; x = false;
  ```
- While weak typing seems beneficial to a programmer it can lead to problems
Type Conversion

- Consider the following example of weak typing in action
  ```javascript
  document.write(4*3);
document.write("\n");
document.write("5" + 5);
document.write("\n");
document.write("5" - 3);
document.write("\n");
document.write(5 * "5");
  ```

- You may run into significant problems with type conversion between numbers and strings using functions like `parseFloat()` to deal with these problems.
  - Prompt demo

Dealing with Type

- You can also use the `typeof` operator to figure out type:
  ```javascript
  var x = "5";
  alert(typeof x);
  ```

- Be aware that using operators like equality or even `+` may not produce expected results:
  ```javascript
  x="5";
y="5";
alert(x == y)
  ```

  Produces a rather interesting result. We see the inclusion of a type equality operator (`===`) to deal with this.

Composite Types

- JavaScript supports more advanced types made up of a collection of basic types.
- Arrays
  - An ordered set of values grouped together with a single identifier
- Defining arrays:
  ```javascript
  var myArray = [1, 5, 1968, 3];
  var myArray2 = ["Thomas", true, 3, -47];
  var myArray3 = new Array();
  var myArray4 = new Array(10);
  ```
### Arrays

- Access arrays by index value
  - `var myArray = new Array(4)`
  - `myArray[3] = "Hello";

- Arrays in JavaScript are 0 based given
  - `var myArray2 = ["Thomas", true, 3, -47];`
  - `myArray2[0] is "Thomas", myArray[1] is true and so on`
  - `Given new Array(4) you have an array with an index running from 0 – 3`
  - `To access an array length you can use arrayName.length`
    * `alert(myArray2.length);`

### Objects

- Underneath everything in JavaScript are objects.
- An object is a collection of data types as well as functions in one package
- The various data types called properties and functions called methods are accessed using a dot notation.

  ```plaintext
  objectName.propertyName
  ```

- We have actually been using these ideas already, for example `document.write("hello")` says using the `document` object invoke the `write()` method and give it the string “hello” this results in output to the string

### Working with Objects

- There are many types of objects in JavaScript
  - Built-in objects (primarily type related)
  - Browser objects (navigator, window, etc.)
  - Document objects (forms, images, etc.)
  - User defined objects
- Given the need to use objects so often shortcuts are employed such as the `with` statement

  ```javascript
  with (document) {
      write("This is easier");
      write("This is even easier");
  }
  ```

- We also see the use of the short cut identifier `this` when objects reference themselves
Expressions and Operators

- Make expressions using operators in JavaScript
- Basic Arithmetic
  - + (addition), - (subtraction/unary negation), / (division), * (multiplication), % (modulus)
- Increment decrement
  - ++ (add one) -- (subtract one)
- Comparison
  - >, <, >=, <=, != (inequality), == (equality), === (type equality)
- Logical
  - && (and) || (or) ! (not)

More Operators

- Bitwise operators (&, |, ^)
  - Not commonly used in JavaScript except maybe cookies?
  - Shift operators (>> right shift, << left shift)
- String Operator
  - + serves both as addition and string concatenation
    - document.write("JavaScript" + " is " + " great!");
  - You should get familiar with this use of +
- Be aware of operator precedence
  - Use parenthesis liberally to force evaluations
    - var x = 4 + 5 * 8 versus x = (4+5) * 8

Flow Control

- Basic program execution control handled in JavaScript using the if statement
  - if (expression) true-case; else false-case;
    - if (x > 10)
      - alert("x bigger than 10");
    - else
      - alert("x smaller than 10");
More on If Statements

- You can use {} with if statements to execute program blocks rather than single statements
  ```javascript
  if (x > 10) {
    alert("x is bigger than 10");
    alert("Yes it really is bigger");
  }
  ```
- Be careful with ;'s and if statements
  ```javascript
  if (x > 10);
  alert("I am always run!?");
  ```

Switch Statements

- If statements can get messy so you might consider using a switch statement instead
  ```javascript
  switch (condition) {
    case (value) : statement(s)
    break;
    …
    default: statement(s);
  }
  ```
- The switch statement is not supported by very old JavaScript aware browsers (pre-JavaScript 1.2), but today this is not such an important issue

Switch Example

```javascript
var x=3;
switch (x) {
  case 1: alert('x is 1');
          break;
  case 2: alert('x is 2');
          break;
  case 3: alert('x is 3');
          break;
  case 4: alert('x is 4');
          break;
  default: alert('x is not 1, 2, 3 or 4');
}
```
Loops

• JavaScript supports three types of loops: while, do/while, and for

• Syntax of while:

while(condition)
    statement(s)

• Example:

var x=0;
while (x < 10)
    document.write(x);
    document.write("<br />");
    x = x + 1;
} document.write("Done");

Do Loop

• The difference between loops is often when the loop condition check is made, for example

var x=0;
do {
    document.write(x);
    x = x + 1;
} while (x < 10);

• In the case of do loops the loop always executes at least once since the check happens at the end of the loop

For Loop

• The most compact loop format is the for loop which initializes, checks, and increments/decrements all in a single statement

for (x=0; x < 10; x++)
    document.write(x);

• With all loops we need to exercise some care to avoid infinite loops. See example
For/In Loop

• One special form of the for loop is useful with looking at the properties of an object. This is the for/in loop.

```javascript
for (var aProp in window) {
    document.write(aProp);
    document.write("<br/>");
}
```

• We will find this construct useful later on when looking at what we can do with a particular object we are using.

Loop Control

• We can control the execution of loops with two statements: break and continue

• break jumps out of a loop (one level of braces)
• continue returns to the loop increment

```javascript
var x = 0;
while (x < 10) {
    x = x + 1;
    if (x == 3)
        continue;
    document.write("x = "+x);
    if (x == 5)
        break;
}
document.write("Loop done");
```

Functions

• Functions are useful to segment code and create a set of statements that will be used over and over again. The basic syntax is

```javascript
function name(parameter list) {
    function statement(s)
    return;
}
```

• For example

```javascript
function add(x, y) {
    var sum = x + y;
    return sum;
}
```
Functions Contd.

- We can then invoke a function using the function name with ()'s:
  ```javascript
  var result = add(2, 3);
  ```
- We can also pass variable values as well as literals:
  ```javascript
  var a = 3, b=5;
  var result;
  result = add(a,b);
  ```
- Variables are passed to function by value so you must use return to send things back.
- You can return a value or not from a function and you can have as many return statements as you like.

Input/Output in JavaScript

- Special dialog forms
  - Alert
    ```javascript
    alert("Hey there JavaScript coder!");
    ```
  - Confirm
    ```javascript
    if (confirm('Do you like cheese?'))
      alert("Cheese lover");
    else
      alert("Cheese hater");
    ```
  - Prompts
    ```javascript
    var theirname = prompt("What's your name?", "");
    ```

Input/Output in JavaScript Contd.

- Writing to the HTML document
  ```javascript
  document.write();
  document.writeln();
  ```
- Writing should be done before or as the document loads.
- In traditional JavaScript the document is static after that, though with the DOM everything is rewritable.
- Since we are writing to an (X)HTML document you may write out tags and you will have to consider the white space handling rules of (X)HTML.
Comments and Formatting

- When writing JavaScript commenting is useful
- Two methods — C and C++ style
  - /* This is a multiple line style comment */
  - // This is a single line comment
- Security concern — who is reading your comments?
- Formatting for reading or for speed?

Summary

- JavaScript supports a basic syntax very similar to C
- It is a weakly typed language
- It has a limited set of data types
- It is very object flavored but it does not force object-oriented programming on programmers
- It forgoes many features of programming languages that wouldn’t make sense in the Web environment (file I/O, complex Math, etc.)