Modern Web Programming

An Introduction to JavaScript

Matthew Perrins, David Artus

IBM Business Solutions Services

Hursley Labs, IBM UK

What is JavaScript

- Scripting language (object-oriented)
  - Lightweight programming language developed by Netscape
  - Interpreted, not compiled

- Designed to be embedded in browsers
  - Ideal for adding interactivity to HTML pages
  - Detect browser versions
  - Work with info from user via HTML forms
  - Create cookies
  - Validate form data
  - Read and write HTML elements

- Supported by all major browsers
  - Internet Explorer has JScript (started in IE3)
  - http://www.faqts.com/knowledge_base/view.phtml/aid/1380

- It’s free, no license required

What is JavaScript

- Syntax is similar to Java, but it’s not Java
  - In places it looks a lot like Java, but it really is <u>not</u> Java

- JavaScript code can be embedded within HTML code using the script tag:
  - But don’t do this

What is JavaScript

HelloWorld example program…

```
<html>
      <head><title>JavaScript HelloWorld!</title></head>
      <body>
        <script language="JavaScript">

          document.write('Javascript says "Hello World!"')

        </script>
      </body>
</html>
```

What is JavaScript

- Javascript can be located in the head, body or external file
  - Head section
    - Runs early, but loading script blocks other activity
  - Body section
    - Script executes when body loads
  - External
    - Allows scripts to run on several pages
    - Non-trivial JavaScript applications are best loaded from .js files
  - Examples:
    - http://www.w3schools.com/js/js_whereto.asp

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

JavaScript for Java Programmers

Deceptively familiar

- JavaScript clearly is part of the C heritage
  - The syntax is familiar – {} () ; ++ return
  - The logic is similar if(){} while()
  - Arrays []

```
var j = 1;
var i = 1;
while( i < 60 ){

    console.log( i + ",");

    i += j;
    j = i - j;

}
```

```
1, 2, 3, 5, 8, 13, 21, 34, 55
```

Snippet 2 - numbers

```
function pling(x){

    if ( x == 1){
         return 1;
     } else {
         return x * pling(x-1);
     }

}

console.log( " 3 yields " + pling(3) );
console.log( " 5 yields " + pling(5) );
console.log( " 100 yields " + pling(100) );
```

```
console.log( ( pling(3)     & pling(2)    ) );
console.log( ( pling(100)  |  pling(99) ) );
```

Learning points (1)

- JavaScript numbers are 64bit floating point

```
function pling(x){

    if ( x == 1){
        return 1;
     } else {
        return x * pling(x-1);
     }

}

console.log( " 3 yields " + pling(3));
console.log( " 5 yields " + pling(5));
console.log( " 100 yields " + pling(100));
```

64 bit floating point so can comfortably cope with 100!

```
console.log( " pling(3) & pling(2) "     +  ( pling(3)     & pling(2)    ) );
console.log( " pling(100) | pling(99) " +  ( pling(100)  |  pling(99) ) );
```

But bitwise operators are 32bit ints, so second expression yields 0

Snippet 3 - scope

```
greeting = "world";

function greet(greeting){

   console.log("Hello " + greeting);

}

function main(){

    greet(greeting);
    var greeting = "local"

}

main();
```

Learning points (2)

- Variables are declared in a scope
- The variable is visible in its scope even before its initialisation/declaration is reached

```
greeting = "world";

function greet(greeting){

   console.log("Hello " + greeting);

}

function main(){

    greet(greeting);
    var greeting = "local"

}

main();
```

Parameter here is a local variable which "hides" global variable.

Local variable here hides global for entire scope, when calling greet() local has not yet been initialised, so we see "undefined".

Snippet 4 – privacy and objects

```
var anObject = {

    x : 7,
    y : 8,

    add: function() {
        return this.x + this.y;
    }
};

console.log( "x " + anObject.x );
console.log( "x " + anObject [“x”] );
console.log( “add " + anObject [“add”] ());
```

Objects

- No encapsulation, attributes are directly visible
- Some conventions: attributes beginning with _ are treated as private
  - No language enforcement

```
var anObject = {

    x : 7,
    y : 8,

    add: function() {
        return this.x + this.y;
    }
};

console.log( "x " + anObject.x );
console.log( "x " + anObject [“x”] );
console.log( “add " + anObject [“add”] ());
```

Can refer to object attributes as associative array

Objects can gain new behavoiour

- Functions are objects
- They can be assigned like any other value

```
var anObject = {

    x : 7,
    y : 8,

    add: function() {
        return this.x + this.y;
    }
};

var x = 100;  var y = x++;

anObject.multiply = function() {
                return this.x * this.y;
};

console.log( "multiply " + anObject.multiply() );
```

Snippet 5 - arrays

```
anObject = [100];
anObject.push("101");
anObject[4] = 102;

console.log( "0 " + anObject[0]);
console.log( "1 " + anObject[1]);

for ( var i = 0; i < anObject.length ; i++ ){
   console.log(“item “ + I + “ “ + anObject[i]);
}
```

Arrays are untyped and may not be contiguous

- All Array objects have useful methods such as push(), pop(), join(), concat()
- JavaScript 1.6 onwards offers forEach()
  - Otherwise may need to check if items are defined

```
anObject = [100];
anObject.push("101");
anObject[4] = 102;

console.log( "0 " + anObject[0]);
console.log( "1 " + anObject[1]);

for ( var i = 0; i < anObject.length ; i++ ){
   console.log(anObject[i]);
}

anObject.forEach( function(item){
    console.log(item);
});
```

Items [2] and [3] undefined

Snippet 6 - context

```
 var anObject = {
     x : 7,
     y : 8,
       add: function() {
            return this.x + this.y;
        }
};

 var x = 100;  var y = x++;

 var fn = anObject.add;

 console.log( "fn " + fn() );
```

Objects

- Invocation has a context
  - Here this.x and this.y are interpreted in the contect of anObject

```
var anObject = {

    x : 7,
    y : 8,

    add: function() {
        return this.x + this.y;
    }
};

var x = 100;  var y = x++;

console.log( "add " + anObject.add() );
```

Objects and functions are not directly related

- We can get a “pointer” to the add function
- And invoke it in a different context.

```
 var anObject = {
     x : 7,
     y : 8,
       add: function() {
            return this.x + this.y;
        }
};

 var x = 100;  var y = x++;

 var fn = anObject.add;

 console.log( "fn " + fn() );    // prints 201
```

Understanding this is important for Dojo programming. We want to run functions when users take UI action, controlling those function’s context is critical.

Snippet 7 – amusing?

http://www.youtube.com/watch?v=kXEgk1Hdze0

```
console.log(Array(16).join(" go " + 2) + " Batman " );


console.log(Array(16).join(" go " - 2) + " Batman " );
```

JavaScript + has two meanings, here we contenate strings: 2 is converted to string

But – is used for numerics, “go” is not a valid number so we get “not a number” or NaN

JavaScript basics

Arithmetic Operators

Operator	Description	Example	Result

+	Addition	x=2
		y=2
		x+y
			4

-	Subtraction	x=5
		y=2
		x-y
			3

*	Multiplication	x=5
		y=4
		x*y
			20

/	Division	15/5 5/2	3 2.5
%	Modulus (division remainder)	5%2 10%8 10%2	1 2 0
++	Increment	x=5 x++	x=6
--	Decrement	x=5 x--	x=4

Assignment Operators

Operator	Example	Is The Same As
=	**x=y**	**x=y**
+=	**x+=y**	**x=x+y**
-=	**x-=y**	**x=x-y**
***=**	**x*=y**	**x=x*y**
/=	**x/=y**	**x=x/y**
%=	**x%=y**	**x=x%y**

Comparison Operators

Operator	Description	Example
==	**is equal to**	**5==8 returns false**
===	**is equal to (checks for both value and type)**	**x=5** **y="5"** **x==y returns true** **x===y returns false**
!=	**is not equal**	**5!=8 returns true**
>	**is greater than**	**5>8 returns false**
<	**is less than**	**5<8 returns true**
>=	**is greater than or equal to**	**5>=8 returns false**
<=	**is less than or equal to**	**5<=8 returns true**

Logical Operators

Comparison operators can be used in conditional statements to compare values and take action depending on the result:

```
if (age<18) x="Too young";
```

Operator	Description	Example
&&	**and**	**x=6** **y=3** **(x < 10 && y > 1) returns true**
||	**or**	**x=6** **y=3** **(x==5 || y==5) returns false**
!	**not**	**x=6** **y=3** **!(x==y) returns true**

`! & Bitwise`

- Prefix *logical not* operator.
- If the operand is truthy, the result is **`false`**. Otherwise, the result is **`true`**.
- **`!!`** produces booleans.

Bitwise

- **`&   |   ^   >>   >>>   <<`**

- The bitwise operators convert the operand to a 32-bit signed integer, and turn the result back into 64-bit floating point.

Statements

- *expression*
- **`if`**
- **`switch`**
- **`while`**
- **`do`**
- **`for`**
- **`break`**
- **`continue`**
- **`return`**
- **`try/throw`**

For statement

- Iterate through all of the elements of an array:

```
for (var i = 0; i < array.length; i++) {

  // within the loop,
  // i is the index of the current member
  // array[i] is the current element

}
```

For statement

- Iterate through all of the members of an object:

```
for (var name in object) {
    if (object.hasOwnProperty(name)) {

        // within the loop,
        // name is the key of current member
        // object[name] is the current value

    }
}
```

Switch statement

- Multiway branch
- The switch value does not need to a number. It can be a string.
- The case values can be expressions.

Switch statement

```
switch (expression) {
case ';':
case ',':
case '.':
    punctuation();
    break;
default:
    noneOfTheAbove();
}
```

Throw statement

```
throw new Error(reason);

throw {
    name: exceptionName,
    message: reason
};
```

Try statement

```
try {
    ...
} catch (e) {
    switch (e.name) {
    case 'Error':
        ...
        break;
    default:
        throw e;
    }
}
```

With statement

- Intended as a short-hand
- Ambiguous
- Error-prone
- Don't use it

```
with (o) {
     foo = null;
}
```

❑ `o.foo = null;`

❑ `foo = null;`

Variables

- Variables are untyped
- Should declare variables before they’re used
  - If you don’t they are global
  - Favour restricted scopes – inside functions

- Example – functions within functions …

```
function doThing() {
    var candyBarPrice = 2.50;
    var taxRate = .075;
    var candyBarsTax = candyBarPrice * taxRate ;
     function printTax() { console.log(candyBarsTax ); }
    …
```

Strings

- Strings are sequences of keyboard characters enclosed in quotes
  - **`“Hello World” or ‘Hello World’`**

- Variables can hold strings
  - **`var greeting = “Hello World”`**

- String can be empty, i.e., contain no characters
  - **`var myAnswer = “”`**

- Use ‘\’ (escape symbol) to enter prohibited characters
  - \b for backspace, \n for newline, \t for tab, \” for double quote

JavaScript Functions – Syntax

- JS function syntax

```
function myFunctionName (list of parameters) {

     ….JS code here…

}
```

Note: no function overloading. You can call functions with too many parameters or too few. Hence may need to check if you have a valid parameter before you use it.

```
function myfunctionName( arg1 ) {
    if ( arg1 && arg1.someProperty) {
        /* do stuff with arg1’s property */
    }
}
```

Deals with arg1 being undefined or null

JavaScript Functions

- Built-In Functions
  - Prompt
  - Alert
  - Confirm
- Useful as a debugging tool
  - Probably more useful to use debugger and console.log
  - UI frameworks will probably use different dialogue capabilities

Coming Up

- Inheritance
- Modules
- Debugging
- Efficiency
- JSON

Modern Web Programming

An Introduction to JavaScript

Matthew Perrins, David Artus

IBM Business Solutions Services

Hursley Labs, IBM UK