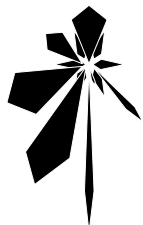


Introduction to JavaScript

Revision 1.1a
August 18, 2010

by
Aloysius Indrayanto



(C) 2010 AnemoneSoft.com

This document is multi-licensed under the Creative Commons Attribution Share-Alike (CC-BY-SA) license version 3.0 and the GNU Free Documentation License (GNU FDL) version 1.3 or later.

1. Introduction

JavaScript is not Java. It is completely different in both concept and design. JavaScript is the most widely-supported client-side scripting language on the internet. JavaScript is free to use by anyone without purchasing a license. On the other hand, Java is a much more complex general-purpose programming language developed by Sun Microsystems (it is now a subsidiary of Oracle Corporation). Java standard is controlled through the Java Community Process.

JavaScript is supported (with slight incompatibilities) by all major browsers, such as Internet Explorer, Mozilla Firefox, Google Chrome, Apple Safari, and Opera. JavaScript was invented by Brendan Eich at Netscape (with the release of Navigator 2.0) in 1996. JavaScript was approved as an international ISO (ISO/IEC 16262) standard in 1998.

JavaScript was designed to to be a lightweight programming language for developing interactive HTML pages. JavaScript is a client-side scripting language, hence, it is executed by the client's browser. Despite the fact that JavaScript is an interpreted language, some browsers may dynamically compile it in the background for better execution speed. Some features of JavaScript are:

- It can create, delete, and modify any HTML element;
- It can react to events occurred on many of the HTML elements;
- It can be used to store and retrieve cookies;
- It can be used to validate data;
- It supports object-oriented programming.

Note that this tutorial will only present the basics of using JavaScript in HTML pages. This tutorial will not discuss about object-oriented programming in JavaScript.

2. Requirements

An elementary knowledge in any programming language will be needed to easily understand the topic discussed in this tutorial. A recent enough browser (Internet Explorer 7.0+, Mozilla Firefox 3.0+, Google Chrome 3.0+, Apple Safari 3.0+, or Opera 9.5+) will be also needed to run the code snippets. This tutorial assumes that the browser to be used supports JavaScript 1.5+. Therefore, older versions of those browsers may still can be used, however, the correctness of the operations in the code snippets are not guaranteed.

3. JavaScript Basics

3.1. Syntax and Expressions

JavaScript code can be written both using ASCII and Unicode (UTF-16) character set. JavaScript is case sensitive. All JavaScript statements end in semi-colons ';', except for those that are followed with curly brackets '{' or '}'. JavaScript statements can be freely formatted using white spaces and line breaks as long as the formatting does not break a single coding element (a token). Therefore, the statements below are invalid statements:

```
var myNumber    = 25 . 5;  
var myLongText = 'An example of a very long text that  
                 needs more than one line to finish.';
```

while these statements are valid statements:

```
var myNumber    = 25.5;  
var myLongText = 'An example of a very long text that needs more than one line to finish.';  
var myLongText = 'An example of a very long text that'  
                 + ' needs more than one line to finish';
```

Single line comments are started with double slash '//'. Multi-line comments is put between slashes '/* */'. Example:

```
// This is a single line comment  
/* This is a multi-line comment that spans  
 * more than one lines  
 */
```

Identifiers in JavaScript must begin with a letter, an underscore '_', or a dollar sign '\$'. All characters after the first character can be letters, numbers, underscores, or dollar signs. None of JavaScript reserved words can be used as an identifier. Below are partial list of JavaScript's reserved words:

break	else	instanceof	true
case	false	new	try
catch	finally	null	typeof
continue	for	return	var
default	function	switch	void
delete	if	this	while
do	in	throw	with

It is also not recommended to use the names from JavaScript's built-in objects and functions as identifiers.

3.2. Operators

The table below lists most of JavaScript operators from the highest to the lowest order of precedence. Operators in the same group have the same level of precedence.

Operator	Description	Operands	Association
<i>Group 1</i>			
.	property access (dot notation)	objects	L to R
[]	array index	array[integer, string]	L to R
()	function call	function(arguments)	L to R
new	create new object	constructor call	R to L
<i>Group 2</i>			
++, --	increment and decrement operators	lvalue	R to L
new	create new object	constructor call	R to L
+, -	unary plus and negation	number	R to L
!	logical complement, logical negation, unary	boolean	R to L
delete	undefine a property	lvalue, unary	R to L
typeof	return data type	any, unary	R to L
void	return a value of undefined	any, unary	R to L
<i>Group 3</i>			
*, /, %	mutiplication, division, modulo (remainder)	numbers	L to R
<i>Group 4</i>			
+, -	addition, subtraction	numbers	L to R
+	concatenation	strings	L to R
<i>Group 5</i>			
<, <=, >, >=	comparision operators: less than, less than or equal to, greater than, greater than or equal to	numbers, strings	L to R
instanceof	test for object type	object instanceof constructor	L to R
in	check for existence of property	string in object	L to R
<i>Group 6</i>			
==, !=	equality, inequality	any	L to R
===, !==	identity, non-identity	any	L to R
<i>Group 7</i>			
&&	logical and	boolean	L to R
<i>Group 8</i>			
	logical or	boolean	L to R
<i>Group 9</i>			
? :	conditional (ternary)	boolean ? any : any	R to L
<i>Group 10</i>			
=	assignment	lvalue = any	R to L
*, /=, %=, +=, -=	assignment with operation	lvalue *= any	R to L

From <http://www.daaq.net/old/javascript/index.php?page=js+expressions&parent=core+javascript>

Despite grouping operators based on their order of precedence, JavaScript operators can also be grouped based on their functions.

1. Arithmetic operators:

- **Addition '+'**. This operator adds its two numeric operands.
- **Subtraction '-'**. This operator subtracts the value of the second numeric operand from the value of the first numeric operand.
- **Multiplication '*'**. This operator multiplies the values of two numeric operands.
- **Division '/'**. This operator divides the value of its first numeric operand by the value of its second numeric operand. Division by zero yields Infinity or -Infinity. The expression $0/0$ evaluates to Nan. Division of integers may yield a floating-point result.
- **Modulo '%'**. This operator returns the remainder value from division between the values of its first numeric operand and its second numeric operand.
- **Unary minus '-'**. This operator negates the value of its numeric operand that immediately following it (converts it to its negative equivalent).
- **Unary plus '+'**. This operator exist to balance the unary minus. This operator really does nothing.
- **Unary increment '++'**. This operator increments the numeric value of its operand by one. If placed before the operand (pre-increment), it returns the incremented value. If placed after the operand (post-increment), it returns the original value prior incrementing the operand.
- **Unary decrement '--'**. This operator decrements the numeric value of its operand by one. If placed before the operand (pre-decrement), it returns the decremented value. If placed after the operand (post-decrement), it returns the original value prior decrementing the operand.
- **Assignment '='**. This operator assigns the value of the right operand into the left operand.
- **Assignment with operation '+=', '-=', '*=', '/=', and '%='**. These operators modify the value of the left operand using the value of the right operand.

2. Relational operators:

- **Equality** '=='. This operator compares its two operands and returns `true` if they are equal, `false` otherwise.
- **Inequality** '!='. This operator compares its two operands and returns `true` if they are *not* equal, `false` otherwise.
- **Comparison** '<', '<=', '>', and '>='. These operators test the relative order of their two operands. They return `true` if the relative orders are correct, `false` otherwise.
- **Operator in**. This operator tests if the first operand is a property of the second operand.
- **Operator instanceof**. This operator tests if the first operand is an object of the class stated in the second operand.

3. String operators:

- **Concatenation** '+'. This operator concatenates its two string operands.
- **Assignment** '='. This operator assigns the value of the right operand into the left operand.

4. Logical operators:

- **Logical NOT** '!'. This operator negates the value of its boolean operand (`true` to `false` and `false` to `true`).
- **Logical AND** '&&'. This operator compares the values of its two boolean operands and returns `true` if both of the operands are `true`.
- **Logical OR** '||'. This operator compares the values of its two boolean operands and returns `true` if one of the operands is `true`.

5. Operator typeof. This operator returns the data type of its operand. The operator will return one of the following strings:

- `undefined`
- `number`
- `string`
- `boolean`
- `function`
- `object` (for objects, arrays, and `null`).

3.3. Data Types and Variables

Data types in JavaScript can be grouped into two categories, the *primitive* data type and the *composite* data type. Primitive data type includes:

- **Numbers.** This data type can represent both integer and floating-point values, for example:
 - 0
 - -75
 - 10000000
 - 5.00
 - 1.0e25
 - 7.5e-3
 - 0xFF0000

There are some numeric literals of this type:

- `Number.POSITIVE_INFINITY` or `Infinity` which represents a number larger than the largest positive number that JavaScript can represent.
- `Number.NEGATIVE_INFINITY` or `-Infinity` which represents a number smaller than the smallest negative number that JavaScript can represent.
- `Number.NaN` or `NaN` which represents something that is not a number, for example, the result of `0/0`.
- `Number.MAX_VALUE` which represents the largest number that JavaScript can represent.
- `Number.MIN_VALUE` which represents the smallest non-zero number that JavaScript can represent.

There are also some functions that can be used to check if a variable is an infinity or a NaN:

- `isFinite()`
- `isNaN()`
- **Strings.** This data type represents a sequence of characters that form a text. A string literal is defined by enclosing a sequence of characters in matching single or double quotes. An empty string is a string with nothing in it. JavaScript support escaped characters such as:

- `\0` for the null character;
 - `\b` for backspace;
 - `\t` for tab;
 - `\n` for new line (a combination of `\r` for carriage return and `\f` for form feed);
 - `\"` for double quote;
 - `\'` for single quote or apostrophe;
 - `\\` for backslash;
 - `\x99` for a two-digit hexadecimal number specifying a character in ASCII;
 - `\u9999` for a four-digit hexadecimal number specifying a character in UTF-16.
- **Booleans.** There are only two boolean values, `true` and `false`. Boolean values usually come from the results of logical comparisons. If used in arithmetic, JavaScript will automatically convert `true` to 1 and `false` to 0. On the other hand, JavaScript will treat any non-zero value as `true`, and a zero value as `false`.

Composite data type includes:

- **Arrays.** In JavaScript, array index starts with zero. JavaScript does not support multi-dimensional arrays, however, arrays can be nested. The code snippet below shows how to initialize and fill an array.

```
// Method one: using array constructor
var myArray = new Array();
myArray[0] = 'One';
myArray[1] = 'Two';

// Method two: using array literals
var myArray = [ 'One', 'Two' ];
```

The code snippet below shows how to nest arrays.

```
// Method one: using array constructor
var myArray = new Array();
myArray[0] = new Array();
myArray[1] = new Array();
myArray[0][0] = 'One A';
myArray[0][1] = 'One B';
myArray[1][0] = 'Two A';
myArray[1][1] = 'Two B';

// Method two: using array literals
var myArray = [ [ 'One A', 'One B' ], [ 'Two A', 'Two B' ] ];
```

The size of an array can be obtained by reading the `length` property of an array.

```
var myArray = [ 'One', 'Two' ];
var myLength = myArray.length; // Will contain the number 2
```

- **Objects.** In JavaScript, an object is a collection of named values (called *properties* of the object). An object can also be associated with functions (called *methods* of the object). Objects in JavaScript can be treated as associative arrays as demonstrated in the code snippet below.

```
var myObject      = new Object();
    myObject.name = 'Mr. Z';

var myName = myObject['name']; // Will contain the string 'Mr. Z'
```

JavaScript has many predefined objects with their own properties and methods, such as Math object, Date object, etc.

- **Functions.** Unlike other programming languages, in JavaScript a function is a data type. Hence, a function can be treated as a variable that contains a value (the function body). Therefore, a function can be passed as a parameter in a function call. A function's value (the function body) can be replaced by assigning a new value. The code snippet below shows how to define a function and then replace it. Note that `alert()` is a JavaScript built-in function to pop-up a message dialog to the user.

```
// Define a function with one parameter
function printHello(name)
{
    alert('Hello \'' + name + '\!'');
}

// Call the function
printHello('John Doe');

// Replace the function
printHello = function()
{
    alert('Boo!');
}

// Call the function again
printHello('John Doe');
```

- **Object literals.** There are two object literals in JavaScript:
 - `null` that states that something has no value (but still a valid object).
 - `undefined` that states that something has never been assigned a value. An `undefined` is not `null` and not zero. Note that `undefined` is not a reserved words but a predefined global variable. Hence, ensure that your code does not overwrite it.

In JavaScript, variables can be defined using the `var` keyword. Scope of variables in JavaScript:

- **Global variables.** All variables that are defined in the script, anywhere in the HTML document, are global variables as long they are not declared inside a function body. All predefined variables and objects are in global scope.
- **Local variables.** All variables that are defined inside a function body is local to that function only.

JavaScript has no block scope as demonstrated by the code snippet below.

```
var myVar = 99;

if(true) {
  var myVar = 25;
  alert(myVar); // Will pop-up a message dialog containing 25
}

alert(myVar); // Will pop-up a message dialog containing 25
```

3.4. Conditional Statements

JavaScript supports two conditional statements, the `if` statement and the `switch` statement. The code snippets below are examples of how to use all the three variants of the `if` statements.

```
var myVal = 100;

if(myVal > 50) {
  // <your statements here>
}
```

```
var myVal = 100;

if(myVal > 50) {
  // <your statements here>
}
else {
  // <your statements here>
}
```

```
var myVal = 100;

if(myVal > 25) {
  // <your statements here>
}
else if(myVal > 50) {
  // <your statements here>
}
else if(myVal > 75) {
  // <your statements here>
}
else {
  // <your statements here>
}
```

The code snippet below shows an example of how to use the `switch` statement.

```
var myIndex = 2;
switch(myIndex) {
  case 0:
    // <your statements here>
    break;
  case 1:
    // <your statements here>
    break;
  case 2:
    // <your statements here>
    break;
  default:
    // <your statements here>
    break;
}
```

Each case branch must be ended with a `break` statement, otherwise the code flow will fall-through the next branch. The code flow will go to the `default` branch if there is no match in all of the case branches.

3.5. Iterative Statements

JavaScript supports four iterative statements, the `while` statement, the `do...while` statement, the `for` statement, and the `for...in` statement. The code snippet below shows an example of how to use the `while` statement.

```
// Pop-up three message dialog, each containing the value 0, 1, and finally 2
var myVar = 0;
while(myVar < 3) {
  alert(myVar);
  ++myVar;
}
```

The code snippet below shows an example of how to use the `do...while` statement.

```
// Pop-up a message dialog containing the value 3
var myVar = 3;
do {
  alert(myVar);
  ++myVar;
} while(myVar < 3)
```

The code snippet below shows an example of how to use the `for` statement.

```
// Pop-up three message dialog, each containing the value 0, 2, and finally 4
for(var myCnt = 0; myCnt <= 4; myCnt += 2) {
  alert(myCnt);
}
```

The code snippet below shows an example of how to use the `for...in` statement.

```
// Pop-up two message dialog, each containing the element index and value
var myArray = [ 'One', 'Two' ];
for(var pos in myArray) {
  alert('Element #' + pos + ' contains \'' + myArray[pos] + '\');
}
```

When the `for...in` statement is used to iterate an array, it will iterate the indexes of the array and not the content of the array. When the `for...in` statement is used to iterate an object, it will iterate the properties of the object.

Note that the use of `for...in` statement *may conflict* with some JavaScript framework/library (such as Prototype.JS) that augments the JavaScript array with some methods. In this case, the `for...in` statement will also iterate those additional methods.

The `break` statement can be used to break the current loop and jump to the next statement after the current loop. The `continue` statement can be used to skip the current iteration back to the beginning of the loop.

3.6. Functions

In JavaScript a function is also a data type. A function can be defined as follows:

```
function functionName(/* argument list */)
{
    // <your statements here>
    return;
}
```

A nested/local function (function inside function) is allowed. However, a function cannot be defined inside a block (for example inside an `if` conditional block). The code snippet below shows examples of functions.

```
function printHello(name)
{
    alert('Hello \'' + name + '\\'!');
}

function isInRange(value)
{
    if(value >= 1 && value <= 10) return true;
    return false;
}

function calcFactorial(value)
{
    if(value <= 1) return 1;
    return value * calcFactorial(value - 1);
}
```

There is a predefined function parameter named `arguments`. It is basically both an object and an array. As an object, it contains some information about the function call. As an array, it contains all the arguments passed in the function call (indexed from zero). There are two properties provided by the `arguments` object that are particularly interesting, `arguments.callee` and `arguments.callee.caller`. The code snippet below demonstrate the use of those properties.

```
function functionOne()
{
    alert('The number of arguments = ' + arguments.length);
    for(var i = 0; i < arguments.length; ++i) {
        alert('Argument #' + i + ' contains ' + arguments[i]);
    }

    alert(arguments.callee);
    alert(arguments.callee.caller);
}

function functionTwo()
{
    functionOne(7, 5);
}

functionTwo();
```

The code snippet below, re-implement the `calcFactorial()` function using `arguments.callee`.

```
function calcFactorial(value)
{
    if(value <= 1) return 1;
    return value * arguments.callee(value - 1);
}

alert(calcFactorial(7)); // Will pop-up a message dialog containing 5040
```

4. Basic Usage of JavaScript in HTML

4.1. Embedding JavaScript

There are three methods to embed JavaScript in an HTML page:

- Embedded between the `<script></script>` tag;
- Loaded from external URL/file using the `src` attribute of the `<script></script>` tag;
- Associated with an HTML event handler.

4.1.1. Embedded between the `<script></script>` Tag

A script tag can be put anywhere in the document, both in the header or the body of the document. One HTML document can have as many script tags as needed. Script tags cannot be nested. All code in the script tags are processed in the order in which they appear in the document. Therefore, if a script refers to an HTML element, ensure that the element is already defined before the script is processed (executed). The code snippet below shows an example of how to embed JavaScript using this method.

```
<script type='text/javascript'>
    alert('Hello world from JavaScript!');
</script>
```

4.1.2. Loaded from External File

It would be a very good idea to separate complicated scripts in their own files. The code snippet below shows an example of how to embed JavaScript using this method.

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

In this case, the JavaScript code stored in the external file `myscript.js` must be written without the `<script></script>` tag. All path will be resolved relative to the location of the HTML file that loads the script files.

4.1.3. Associated with an HTML Event Handler

The code snippet below shows an example of how to associate a piece of JavaScript code with an HTML event handler.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"  
    'http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd'>  
  
<html xmlns='http://www.w3.org/1999/xhtml' lang='en' xml:lang='en'>  
  <head>  
    <script type='text/javascript' src='prototype-c161.js'></script>  
  </head>  
  <body style='text-align:center;'>  
    <p onclick="alert('Hello world from JavaScript!');">Click me!</p>  
  </body>  
</html>
```

Note that one will need to use different type of quote if one wants to include string in the JavaScript code. Escaping the quote such as in the examples below will not work!

```
<p onclick="alert(\"Hello world from JavaScript!\");">Click me!</p>
```

```
<p onclick='alert(\'Hello world from JavaScript!\');'>Click me!</p>
```

Personally, I would prefer to use single quotes for all HTML attribute values and JavaScript strings whenever possible.

4.2. Event Handlers

The table below displays a partial list of event handlers supported by JavaScript. Some events can happen to many HTML elements. However, some other can only happen to some specific HTML elements.

Event Handler	Description
<i>Mouse Events</i>	
onmousedown	A mouse button has been pressed.
onmouseup	A mouse button has been released.
onclick	A mouse button has been clicked.
ondblclick	A mouse button has been double-clicked.
onmousemove	The mouse has been moved.
onmouseover	The mouse pointer has entered an element.
onmouseout	The mouse pointer has left an element.
<i>Keyboard Events</i>	
onkeydown	A key has been pressed.
onkeyup	A key has been released.
onkeypress	onkeydown followed by onkeyup.
<i>Special Events</i>	
onload	The document has completely loaded.
onunload	The document is unloaded (navigated away).
onfocus	An element gains focus.
onblur	An element loses focus.
onchange	An element's content has been changed.
onsubmit	A form submit command is issued.
onreset	A form reset command is issued.

From <http://webdevelopersjournal.com/articles/jsevents1/jsevents1.html>

Some events provides additional information. In this case, additional information related to an event can be captured by using the Event object. However, there are incompatibilities between browsers about how the Event object can be accessed, for example Mozilla-based browsers prefers to pass the object as an argument to the event handler while Internet Explorer prefers to use a global event object. The code snippet below shows a generic method to access the object.

```
function myEventHandler(e)
{
    if(!e && typeof(event) != 'undefined') e = event;
    // <use the Event object 'e' here>
}
```

Due to the global event object comes from the window object (all properties and methods of the window object can be accessed as if they have global scope), the code snippet below can also be used.

```
function myEventHandler(e)
{
    if(!e && window.event != undefined) e = event;
    // <use the Event object 'e' here>
}
```

To complicate the matter, there are also incompatibilities in the implementation of Event objects between browsers. The code snippet below shows a possible example of how to get the key code on a keypress event.

```
function myKeyPressEventHandler(e)
{
    // Get the keycode
    var code;
    if(!e) var e = window.event;
    if(e.keyCode) code = e.keyCode;
    else if(e.which ) code = e.which;

    // <use the keycode here>
}
```

It is also possible to convert the key code to a character as shown by the code snippet below.

```
function myKeyPressEventHandler(e)
{
    // Get the keycode
    var code;
    if(!e) var e = window.event;
    if(e.keyCode) code = e.keyCode;
    else if(e.which ) code = e.which;

    // Convert to character and show to user
    var char = String.fromCharCode(code);
    alert('Character was \'' + char + '\'.');
}

document.onkeypress = myKeyPressEventHandler;
```

The next code snippet shows how to get the absolute coordinate of the mouse pointer in an HTML page.

```
function myEventHandler(e)
{
    // Get the mouse pointer coordinate
    if(!e) e = event;
    var x = e.clientX ? e.clientX : e.pageX;
    var y = e.clientY ? e.clientY : e.pageY;

    // <use the coordinate here>
}
```

The code snippet below displays a convenience function that can be used to get the origin (top-left position) of an HTML element on an HTML page. The returned coordinate can be used to convert an absolute mouse pointer coordinate into a coordinate relative to the origin of the element.

```
// Get the origin (left, top) position of an element
function getElementOriginXY(obj)
{
    // Get the pixel value of a particular attribute of a particular object
    function _getAttrPixelValue(obj, atr) {
        var px = 0;
        if(window.getComputedStyle) {
            var css = null;
            var sty = window.getComputedStyle(obj, '');
            if(sty && sty.getPropertyCSSValue) {
                try {
                    css = sty.getPropertyCSSValue(atr);
                    if(css && css.primitiveType <= 18) px = css.getFloatValue(5) | 0;
                }
                catch(e) {}
            }
        }
        return px;
    }

    // Get the scroll position of the 'body' element
    var bodyScrollTop = document.body.scrollTop;
    if(bodyScrollTop == 0) {
        if(window.pageYOffset) bodyScrollTop = window.pageYOffset;
        else
            bodyScrollTop = (document.body.parentElement) ?
                document.body.parentElement.scrollTop : 0;
    }
    var bodyScrollLeft = document.body.scrollLeft;
    if(bodyScrollLeft == 0) {
        if(window.pageXOffset) bodyScrollLeft = window.pageXOffset;
        else
            bodyScrollLeft = (document.body.parentElement) ?
                document.body.parentElement.scrollLeft : 0;
    }

    // Save the offset of the current element
    var top = obj.offsetTop;
    var left = obj.offsetLeft;

    // Select the parent offset and loop until no more offset
    obj = obj.offsetParent;
    while(obj) {
        // IE
        if(document.all) {
            // Adjust the offset with the current scrollbar data
            if(obj.offsetParent) {
                if(obj.scrollTop) top -= obj.scrollTop;
                if(obj.scrollLeft) left -= obj.scrollLeft;
            }
        }
        // Non IE
        else {
            // Adjust the offset with the current scrollbar data
            if(obj.scrollTop) top -= obj.scrollTop;
            if(obj.scrollLeft) left -= obj.scrollLeft;
            // Handle Mozilla bug
            if((obj.tagName == 'DIV') ||
                (obj.tagName == 'TABLE' && navigator.vendor == 'Netscape')) {
                top += _getAttrPixelValue(obj, 'border-top-width') | 0;
                left += _getAttrPixelValue(obj, 'border-left-width') | 0;
            }
        }
        // Accumulate the offset
        top += obj.offsetTop;
    }
}
```

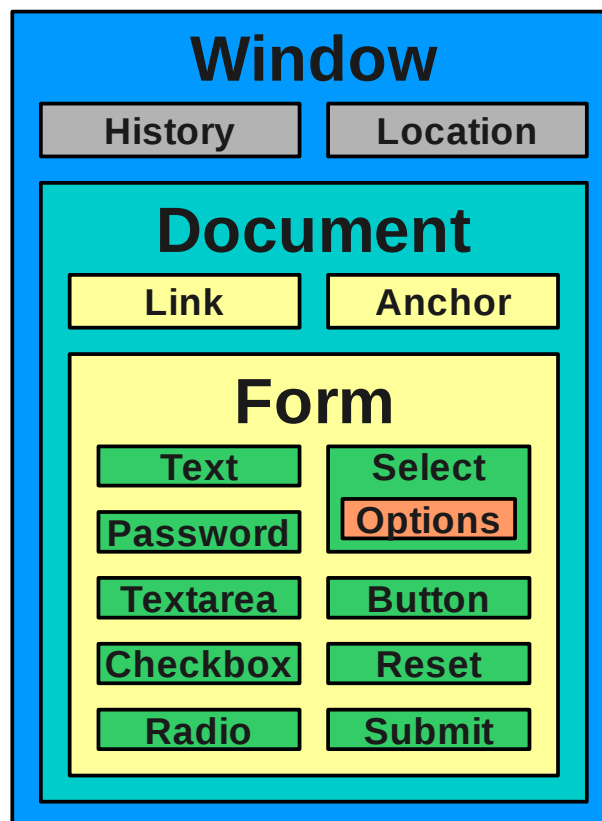
```
    left += obj.offsetLeft;
    // Select the parent offset
    obj = obj.offsetParent;
}

// Adjust the offset
if(navigator.userAgent.indexOf('Mac') != -1 &&
    typeof(document.body.leftMargin) != 'undefined') {
    left += document.body.leftMargin;
    top  += document.body.topMargin;
}

// Return the origin point of the element
return [left - bodyScrollLeft, top - bodyScrollTop];
}
```

4.3. The Document Object Model (DOM)

The Document Object Model (DOM) is a W3C standard for representing and interacting with HTML elements (objects). It is designed to be cross-platform and language-independent. DOM Level 1 (the first DOM standard) was published in 1998. DOM Level 2 was published in 2000. DOM Level 3 was published in 2004. Currently, all the important features of DOM Level 1 are virtually supported by all major browsers. However, features from DOM Level 2 are only partially supported by those major browsers. Support for DOM Level 3 is even lesser. The picture below shows hierarchy of objects specified by DOM.



From http://en.wikipedia.org/wiki/Document_Object_Model

4.3.1. The window Object

The window object is the highest level object in the DOM hierarchy. Basically, all methods from the window object can be called as if they have global scope. The table below shows a partial list of methods supported by the window object.

Method	Description	Example Usage
alert	Display a message dialog.	<code>window.alert('welcome to my village!');</code>
confirm	Display a confirmation dialog.	<code>if(window.confirm('Sure to leave the shop?')) window.alert('You leave the shop.');</code> <code>else window.alert('You do not leave the shop.');</code>
prompt	Display an input dialog.	<code>var ret = window.prompt('How many potions to buy?', '5');</code> <code>if(ret) window.alert('You bought ' + ret + ' potion(s).');</code> <code>else window.alert('You bought no potion.');</code>
open	Open a new window/tab. Some browsers may block/confirm this operation before actually opening the new window/tab.	<code>window.open('http://www.google.com', 'My Google Window');</code>

4.3.2. The document Object

The document object basically contains all the HTML elements in the current page. The table below shows a partial list of properties supported by the document object.

Property	Description
domain	The domain name of the server that servers the document.
location or URL	The location (URL) of the current document.
referrer	The location (URL) of the document that refers to this document.
lastModified	The date the original file was last modified.
title	The title of the current document as specified by the <title></title> tag.

The most important method from the document object would be the `getElementById()`. It allows a JavaScript developer to directly access an HTML element with the given ID. The code snippet below demonstrate this feature.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" lang="en" xml:lang="en">
  <head></head>
  <body>
    <div id="divOne"></div>
  </body>
```

```
</html>

<script type='text/javascript'>
  document.getElementById('divOne').innerHTML = 'Hello world from JavaScript!';
</script>
```

There is a problem with the implementation of `getElementById()` in Internet Explorer before version 8 and in some other browsers. The buggy method will correctly return the element with that ID, but it will also return all elements of which name attributes are set to the same string. Therefore, it is a good idea to use a JavaScript framework/library that can overcome this problem. An example of such library is the Prototype.JS JavaScript framework (please refer to the URL in the reference section for more details). However, if one want a simpler solution, the code snippet below can be used to fix the method in those browsers.

```
// Save the original method
document.origGetElementById = document.getElementById;

// Replace the method
document.getElementById = function(id) {
  // Get any element of which ID and name match
  var elem = document.origGetElementById(id);
  // Just return if there is no match
  if(!elem) return null;
  // Verify if it is a valid match
  if(elem.attributes['id'] && elem.attributes['id'].value == id)
    return elem;
  // No valid match
  else {
    for(var i = 1; i < document.all[id].length; ++i) {
      if(document.all[id][i].attributes['id'] &&
        document.all[id][i].attributes['id'].value == id) {
        return document.all[id][i];
      }
    }
  }
}
```

From <http://webbugtrack.blogspot.com/2007/08/bug-152-getelementbyid-returns.html>

Prior executing the above code, one must ensure that the implementation of the `getElementById()` method does buggy in the currently used browser by checking the browser identification and version. Please note that the methods to check browser's identification and version are beyond the scope of this tutorial.

5. More Code Examples

This section presents some code snippets that combine some of the techniques that have been discussed in the previous sections. For simplicity, only the content of the `<body></body>` and `<script></script>` tags that will be shown in all code snippets. Implementation details for functions that have been discussed in the previous sections are also excluded.

Try to modify and experiment with the code snippets in this section. You can also try to

write your own JavaScript program by combining techniques that have been discussed in the previous sections.

5.1.1. Generating Random Numbers with Range

```
...
...
<body>
  <div id='divRandom' style='margin-bottom:10px; width:300px; height:100px;
    background-color:#000000; color:#00FF00;
    text-align:center; font-size:500%'></div>
  <div><input id='btnGen' type='button'
    value='Generate a New Random Number'></input></div>
</body>
...
...
<script type='text/javascript'>
  // Generate a random number between the given range inclusively
  function genRandom(min, max)
  { return Math.floor(Math.random() * (max - min + 1)) + min; }

  // Assign event handler
  document.getElementById('btnGen').onclick = function() {
    document.getElementById('divRandom').innerHTML = genRandom(1, 10);
  }

  // Run the event handler once on startup
  document.getElementById('btnGen').onclick();
</script>
```

tut01.html

5.1.2. Displaying Absolute and Relative Mouse Pointer Coordinate

```
...
...
<body>
  <div id='divArea1' style='float:left; margin:50px; padding:10px; width:300px;
    height:200px; background-color:#0000FF;'></div>
  <div id='divArea2' style='float:left; margin:50px; padding:10px; width:300px;
    height:200px; background-color:#FF0000;'></div>
  <div id='divInfoA' style='clear:both; margin:50px; padding:10px; width:300px;
    height:100px; background-color:#000000;
    color:#00FF00; font-family:monospace;'></div>
</body>
...
...
<script type='text/javascript'>
  // Get the origin (left, top) position of an element
  function getElementOriginXY(obj)
  {
    ...
    ...
  }

  // Common event handler
  function divMMEventHandler(e)
  {
    // Get the mouse pointer coordinate
    if(!e) e = event;
    var x = e.clientX ? e.clientX : e.pageX;
    var y = e.clientY ? e.clientY : e.pageY;
  }
</script>
```

```
// Get the origin of this object
var o = getElementOriginXY(this);

// Generate information text
var info = '<b>' + this.id + '</b>'
          + '<p>Absolute position = ' + x + ', ' + y + '</p>'
          + '<p>Relative position = ' + (x - o[0]) + ', ' + (y - o[1]) + '</p>'
document.getElementById('divInfoA').innerHTML = info;
}

// Assign event handlers
document.getElementById('divArea1').onmousemove = divMMEventHandler;
document.getElementById('divArea2').onmousemove = divMMEventHandler;
</script>
```

tut02.html

References

http://en.wikipedia.org/wiki/Java_Community_Process, June 24, 2010

http://www.w3schools.com/js/js_intro.asp, June 01, 2010

<http://www.howtcreate.co.uk/tutorials/javascript/introduction>, June 01, 2010

<http://www.daaq.net/old/javascript>, June 24, 2010

<http://en.wikipedia.org/wiki/JavaScript>, June 24, 2010

<http://webdevelopersjournal.com/articles/jsevents1/jsevents1.html>, June 26, 2010

http://en.wikipedia.org/wiki/Document_Object_Model, June 27, 2010

<http://www.tizag.com/javascriptT/javascript-getelementbyid.php>, June 24, 2010

<http://webbugtrack.blogspot.com/2007/08/bug-152-getelementbyid-returns.htm>, June 26, 2010

http://www.w3schools.com/browsers/browsers_stats.asp, June 24, 2010

<http://www.sergiopereira.com/articles/prototype.js.html>, June 01, 2010