Introduction to JavaScript and jQuery

Student Guide

Revision 4.6

Object Innovations Course 3201

Introduction to JavaScript and jQuery
Rev. 4.6

Student Guide

Authors: Robert J. Oberg and Marc Bueno

Object Innovations
877-558-7246
www.objectinnovations.com

Published in the United States of America.

Table of Contents (Overview)

Directory Structure

- **Install the course software by running the self-extractor *Install_IntroJs_46.exe*.**
- **The course software installs to the root directory *C:\OIC\IntroJs*.**
  - Example programs for each chapter are in named subdirectories of chapter directories **Chap01** and **Chap02**.
  - The **Labs** directory contains one subdirectory for each lab, named after the lab number. Starter code is frequently supplied, and answers are provided in the chapter directories.
  - The **Demos** directory is provided for performing in-class demonstrations led by the instructor.

Table of Contents (Detailed)

Chapter 1

Using JavaScript

Using JavaScript

Objectives

After completing this unit you will be able to:

- **Summarize the history of JavaScript and outline the principal standards.**
- **Describe the uses of JavaScript.**
- **Use JavaScript for client-side scripting in Web applications that run in a browser.**
- **Explain the interpretative nature of JavaScript.**
- **List the JavaScript built-in data types.**
- **Describe the use of variables and literals in JavaScript programs.**
- **Use control structures in JavaScript.**
- **Use functions in JavaScript.**
- **Use strings and arrays in JavaScript.**
- **Describe the object-based features of JavaScript.**

History and Standards

- **JavaScript was invented by Brendan Eich and introduced by Netscape in Navigator 2.0, under the original name LiveScript.**
  - Because "Java" was hot, the language's name was changed to JavaScript, although it has nothing to do with Java, other than syntax similarities.
- **Microsoft introduced its own version of the language in Internet Explorer 3.0, calling it *JScript*.**
- **JavaScript was standardized by ECMA in 1997, under the name *EcmaScript*.**
  - The name of the standard is ECMA-262.
  - With small changes the standard was approved as an international ISO standard in 1998.
  - ECMA-262 second edition reflects minor editorial changes to keep it aligned with the ISO standard.
- **For a long time ECMA-262 third edition, approved in 1999, was the standard. It added features such as regular expressions, better string handling, try/catch exception handling and new control structures.**
  - All the JavaScript covered in this course is consistent with the third edition standard.
- **The current version is the fifth edition, with changes to make EcmaScript more fully object-oriented.**

Uses of JavaScript

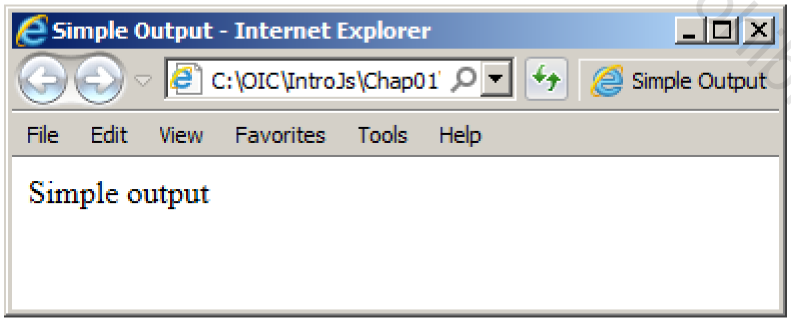
- **JavaScript can be used in Web applications on either the client or server side.**
  - On the client, it appears in a <script> block in an HTML/XHTML page.
  - On the server it can be invoked by a mechanism such as CGI, in a manner similar to invoking scripts in other languages such as Perl.
- **JavaScript can also be used for general scripting purposes, such as system administration.**
  - On Windows platforms, JavaScript is one of the scripting languages supported by Windows Scripting Host (WSH).

JavaScript in the Browser

- **JavaScript is supported in all the important modern browsers, such as Internet Explorer (IE), FireFox, Chrome and Safari.**
- **The interaction of JavaScript with the browser is specified by a Document Object Model (DOM).**
  - We discuss the DOM in the next chapter, but will make use of simple features in this chapter.
- **Here is some simple JavaScript that will use a method of the *document* object to display a string in the current page.**

```
<script type="text/javascript">
    document.write("Simple output");
</script>
```

  - See **SimpleOutput\Embedded** in the chapter folder.
- **Here is what is displayed in the browser (IE):**
  - Double-click on the file **Default.html** to bring up the default browser (or open Web site, open the file, and run from Visual Studio). If prompted, click to Allow Blocked Content.



Simple output

Embedded JavaScript

- **The simplest way to write JavaScript is to simply embed the JavaScript code in a <script> block.**

```
<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Simple Output</title>
<script type="text/javascript">
    document.write("Simple output");
</script>
</head>
<body>
    <form id="form1" runat="server">
    <div>

    </div>
    </form>
</body>
</html>
```

  – The attribute **type=”test/javascript”** specifies that the script is written in JavaScript.

  – Note that the **language** attribute is obsolete.

.js Files

- **Another way to implement JavaScript code in a Web page is to have the JavaScript in its own file with the extension *.js*.**
  - See **SimpleOutput\Js** in the chapter folder for an example.
- **In the <*script*> tag specify the *.js* file with the *src* attribute.**

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

- **Now there is no content between the start and end <*script*> tags.**
- **The *.js* file has only JavaScript statements.**

```
// simple.js
document.write("Simple output");
```

JavaScript Language

- **JavaScript is an *interpreted* language.**
  - There is no separate compilation step. JavaScript code is interpreted by its runtime host, such as a Web browser or Windows Scripting Host.
- **JavaScript is like the C family of languages in its general syntax.**
  - C, C++, C# or Java programmers should be quite comfortable with JavaScript.
- **There is no *main()* entry point.**
  - In browser-hosted JavaScript, execution begins with the first statement in a **<script>** tag and in functions that handle events.
- **JavaScript is weakly typed, and the same variable can hold objects of different types at different times during program execution.**
- **JavaScript is object-based.**
  - JavaScript programs comfortably use properties, methods and events.
  - JavaScript does not have class-based inheritance, but it does support a more complex inheritance model using *prototypes*.

Variables

- **In JavaScript you must declare a variable before using it, with the keyword *var*.**

```
var x;
```

- **Optionally, you may initialize a variable when it is declared.**

```
var x = "12abc";
```

  - A variable that has not been assigned a value has the special value **undefined**.

- **In JavaScript a variable does not have a type, but the current value of a variable does have a type.**
  - In the example above, the type of x is a **String**.

- **The same variable may have values of different types during different times.**

```
x = parseInt(x);    // x is now a number
```

  - In this example the string is parsed until a non-numeric character is encountered. (If the string does not begin with a numeric character, the parse will fail.)

Variables Example

- **The program *Variables* illustrates working with variables.**

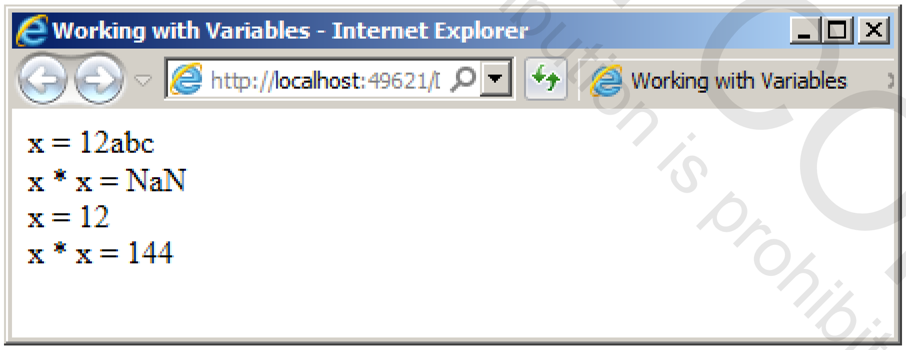
```
<script type="text/javascript">
    var x = "12abc";        // x is a string
    document.write("x = " + x);
    document.write("<br>");

    var y = x * x;
    document.write("x * x = " + y);
    document.write("<br>");

    x = parseInt(x);        // x is now a number
    document.write("x = " + x);
    document.write("<br>");

    var z = x * x;
    document.write("x * x = " + z);
</script>
```

- **Here is the output:**



```
x = 12abc
x * x = NaN
x = 12
x * x = 144
```

   – NaN means “Not A Number.”

Naming Variables

- **Legal name for a variable:**
  - The first character must be a letter or underscore.
  - Subsequent characters must be letters, numbers or underscores.
  - The name must not be a **reserved word**.
- **A variable may either be initialized when it is defined, or subsequently assigned a value;**

```
var n = 10;  // value via initialization

var m;
m = 20;      // value via assignment
```

- **If you want to declare and initialize a variable without specifying a particular value, you may use the special value *null*.**

```
var p = null;
```

- **Names in JavaScript are case sensitive.**

Reserved Words

- **According to the ECMA-262 specification (v3), the following are the reserved words in JavaScript:**

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

- **There are also *future reserved words*, which are reserved to allow for possible language extensions.**

abstract	enum	int	short
boolean	export	interface	static
byte	extends	long	super
char	final	native	synchronized
class	float	package	throws
const	goto	private	transient
debugger	implements	protected	volatile
double	import	public	

Types

- **In JavaScript there are *primitive*, *composite*, and *special* data types.**
- **The primitive data types in JavaScript are:**
  - String
  - Number
  - Boolean
- **The special data types are:**
  - Null (only value is **null**)
  - Undefined (only value is **undefined**)
- **The composite (or *reference*) data types are:**
  - Array
  - Object

Strings

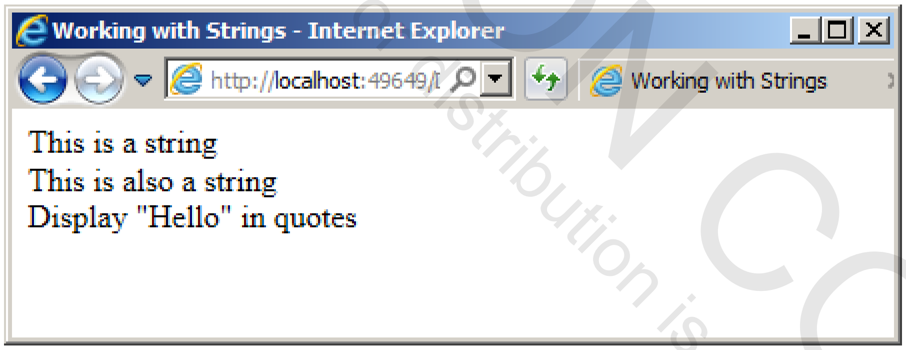
- **A string in JavaScript consists of a sequence of zero or more Unicode characters.**
- **A string literal is created by matching pairs of single or double quotation marks.**

```
"This is a string"
'This is also a string'
```

- **You may display a quoted string by using both forms of quotation mark.**

```
'Display "Hello" in quotes'
```

- **The example program *Strings* provides an illustration.**



```
This is a string
This is also a string
Display "Hello" in quotes
```

Numbers

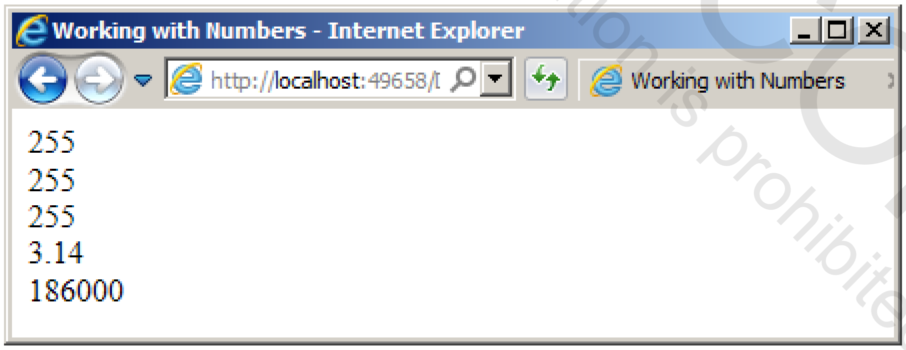
- **JavaScript does not distinguish between integers and floating point numbers.**
  - Internally JavaScript stores all numbers in floating point.
- **Integers are whole numbers.**
  - They may be represented in decimal (base 10, the default), hexadecimal (base 16), or more rarely in octal (base 8).
  - Hexadecimal literals begin with "0x" and octal literals with "0" (zero).
- **Floating point numbers may have a decimal part.**

```
3.14
```

  - They may also be represented using exponential notation.

```
1.86e5
```

- **See the sample program *Numbers* in chapter folder.**



```
255
255
255
3.14
186000
```

Boolean

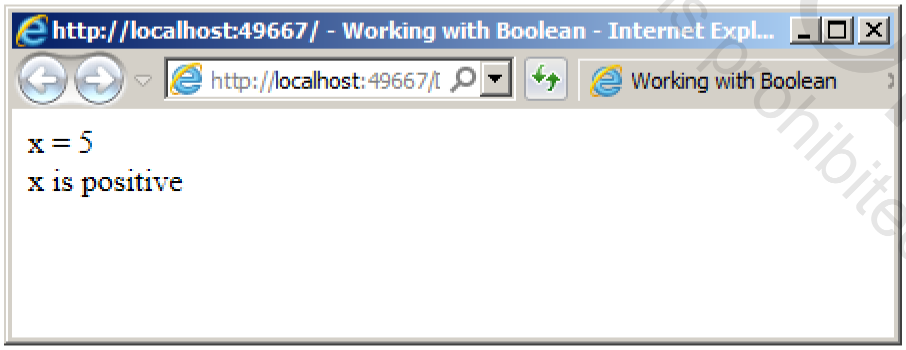
- **A Boolean data type has the value *true* or *false*.**
- **Booleans are useful in control structures such as *if* tests.**
  - You combine a comparison operator with a control structure that uses it.
  - Comparison operators:

```
==   !=   >    <    <=   >=
```

- **A simple example is provided by the *Boolean* program in the chapter folder.**

```
<script type="text/javascript">
    var x = 5;
    document.write("x = " + x);
    document.write("<br>");
    if (x > 0)
        document.write("x is positive");
</script>
```

- **Here is the output:**


http://localhost:49667/ - Working with Boolean - Internet Expl...
http://localhost:49667/
Working with Boolean

```
x = 5
x is positive
```

Operators in JavaScript

- **JavaScript has a comprehensive set of operators, similar to those in the C language.**
- **Here are the computational operators:**

Unary negation	`-`
Increment	`++`
Decrement	`--`
Multiplication	`*`
Division	`/`
Modulus arithmetic	`%`
Addition	`+`
Subtraction	`-`

- **The modulus operator divides two numbers and returns only the remainder as result.**
  - The sign of the result is the same as the sign of the first number.
  - The value of the result is between 0 and the absolute value of second number.

```
37 % 5  is  2
```

Logical Operators

Logical not	`!`
Less than	`<`
Greater than	`>`
Less than or equal to	`<=`
Greater than or equal to	`>=`
Equality	`==`
Inequality	`!=`
Logical AND	`&&`
Logical OR	`||`
Strict equality	`===`
Strict inequality	`!==`

- **Difference between equality and strict equality:**
  - The equality operator will “coerce” (convert type) values of different types before doing the comparison. (The string “7” and number 7 will compare as equal.)
  - The strict equality operator insists that the values being compared are of the same type. (The string “7” and number 7 will compare as unequal.)
- **The AND, OR operators perform “short circuit” evaluation, stopping when truth/falsity is known.**

Bitwise and Assignment Operators

- **The bitwise operators perform operations on the binary representations of their operands, bit by bit.**

Bitwise NOT	`~`
Bitwise left shift	`<<`
Bitwise right shift	`>>`
Unsigned right shift	`>>>`
Bitwise AND	`&`
Bitwise OR	`|`
Bitwise XOR	`^`

- **JavaScript has both simple and compound assignment. The operator may be any computational bitwise binary operator.**

Assignment	`=`
Compound assignment	`op=`

  – A compound assignment is simply shorthand for the operation followed by an assignment.

```
X += 5  is the same as  x = x + 5
```

Functions

- **Functions can be used to modularize programs.**
- **JavaScript has built-in functions, such as *parseInt()*.**
- **You can also write your own functions, consisting of:**
  - The reserved word **function** along with a parameter list
  - A block containing JavaScript statements.
  - An optional **return** statement if the function returns a value.

```
function cube(x)
{
   return x * x * x;
}
```

- **A function may just execute some statements and not return a value.**
  - This function contains repetitive code for displaying two numbers separated by a string, followed by a **<br>** to cause a new line in the browser.

```
function display(a, str, b)
{
   document.write(a + str + b);
   document.write("<br>");
}
```