**Content**

**Synopsis**

**Introduction**

**Modules**

**Part I**

* **Understanding JavaScript**
* JavaScript and the Browser Wars
* Setting up a Web Test System
* **Performing Operations**
* The <script> tag
* Rules and Syntaxes
* The Message box & The Console Log
* Keywords
* Variables
* Data Types and Escape Sequence
* String manipulations
* Functions
* Event Handlers & Event Delegation
* Closure
* Global Pollution & Hoisting
* Unobtrusive JavaScript and the DOM
* Breakpoints
* Constructors

* **Conditional Statements & The Window Object**
* The WHILE loop
* The FOR loop
* The CONTINUE and BREAK statements
* The IF, ELSE IF and ELSE clauses
* Window properties
* The CALL statement
* The APPLY statement
* **Best Practices**
* **Frameworks & Libraries**
* **Introduction to Object Oriented JavaScript and ECMA2015**

**Learn To Write JavaScript**

**JavaScript for Web development**

In order to provide **meaningful information** to corporate intranet users and to **compete for business** on the World Wide Web, organisations must build websites that are **effective and responsive**. Java Script is a **programming language** that provides a **powerful tool** for **rapidly developing highly** **interactive, dynamic web sites**.

This course provides in-depth hands on **experience writing client-side JavaScript**. You will learn how to **integrate** JavaScript **into your web pages** and create an **interactive** and **dynamic web site**.

**You will learn to:**

**Development Skills**

* Write and debug client – side JavaScript that works across multiple browsers
* Create dynamic and interactive web pages
* Add interactivity forms that capture and validate user input
* Personalise pages
* Use Event Handlers (triggers), Event Delegation and Calls To Action (CTA) on JavaScript functions
* Write Unobtrusive JavaScript, manipulate DOM and AJAX operations
* Build and work with JavaScript Libraries

**What will I achieve?**

**Mastery Skills**

* Write and debug JavaScript functions
* Create interactive, self validating forms
* Implement interactive user interface using modal and dialogue boxes.
* Understand unobtrusive JavaScript and Event Delegation techniques.
* Manipulate DOM and AJAX operations

# Synopsis

JavaScript is **Client side interpreted programming language**. A language, that **runs** within the **web browser.**

Its interpreteris **embedded** inside web browser software, such as **Microsoft Internet Explorer** and **Mozilla** browser family, **Chrome**, **Safari** and **Opera** to mention a few.

The language **grants access** to the current page and lets the script determine **properties** of the client, **redirect** the user to another page, **access** cookies and do much more.

The main principle behind **embedded** scripting within an HTML document is to **control the web page from the client-side rather than from the server** in contrast to CGI scripting. In this way the web documents can be read by the browser’s “**JavaScript engine**” whenever the document is loaded into the browsers window.

The JavaScript language contains many impressive features but for **security** reasons it cannot **read or write** (**copy**) files, with the exception of “**cookie**” files that store a small amount of data (maximum size: 4 kilobytes). Referred to as **Cross Domain Policy.**

Combining JavaScript with knowledge of DOM enables powerful creation of dynamic html effects.

1. **Understanding JavaScript**

**1a. JavaScript and the Browser Wars**

The birth of JavaScript was in **September 1995**, originally by Brendan Eich with Netscape version 2 named **Mocha** and later re-named **LiveScript** after marketing deal with Java. Microsoft began to promote and integrate JavaScript and later into their Internet Explorer (IE) browsers. For legal reasons called it **JScript**. The worse was to follow as Microsoft begun computing specific **incompatible library features** and **characteristics** for their JScript engine without **legal commercial licensing.**

In **1997** the European Computer Manufacturers Association( **ECMA** 262) the standard was published, and JavaScript was the **first** implementation. The standard only **specified** the **language** and **not features**. This escalated more inconsistencies and the browser war continued. Netscape **scrapped** the idea of releasing browser **version 5** and decided to start it all over again with **Netscape 6** (the Mozilla project). All this helped Internet Explorer to expand its market share reaching over **90%** by **early 2000**. The Mozilla project gave birth to **Firefox** and other flavours of the **Mozilla** browser derivatives (Epiphany, Camino, Galen etc).

Until IE7, Internet Explorer was far away from supporting the latest JavaScript 1.5 released in 2005. With IE7+ and FF 2.0 + released, developers are now much more comfortable developing for **cross browser compatibility**.

Other browsers, by the way support JavaScript as well. Differences are subtle, but still can be extremely annoying when one is developing a **browser – agnostic** web application. Among the browsers that currently support JavaScript are the following:

* Internet Explorer
* Mozilla and all derivatives (Firefox, Epiphany, Camino and Galeaon)
* Opera
* Konqueror
* Safari
* Chrome

The current modern war is both on **platforms** and **technology**. Some companies are:

* Abandoning platforms or slow to deploy their technology to platforms (delayed Flash on iPads, abandoned on new Android browsers, etc)
* The modern ecology of the web-space continues to be challenging, and there is evangelization of more broad support for the basic structures of the web, but for the bleeding-edge this is still an organic process.

Note, not all browsers may have JavaScript enabled. As such, test your website and applications built with JavaScript first **enabled**, and then **disabled**. **Core** **functionality should be maintained when JavaScript is** **disabled** (you may have to provide alternate solution or strategy for core functionality without JavaScript), and the rendering of your pages should **degrade gracefully**.

Generally speaking, JavaScript should be used for **progressive enhancement**. Of late **CTM** (**cut the mustard)**  tests (courtesy of the British Broadcasting Corporation - BBC) is used to provide enhanced user experience for modern mobile phones that have higher CPUs and bandwidth.

**The Future**

* JavaScript **performance** will continue to be an important part of each browser's competitive advantage.
* Google has a big interest in JavaScript speed and performance on different browsers due to their **cloud computing** strategies.
* Microsoft is catching up to their competitors' on how well JavaScript performs on IE.
* Popular new web standards such as HTML5 and CSS3 need JavaScript and its associated Frameworks to join them together.

.

#### 2. Performing Operations and Making Statements

### **2.1 The *<script>* tag**

The **<script>** tag enables us to **include** scripts within an HTML document. It contains the **script block** and usually shares the **HEAD** section of the document with the **TITLE** and the **META** tags. The **LANGUAGE** attribute defines the **type** of scripting language we are using. If omitted the **SCRIPT** tag will assume that **JavaScript** is used. Also, the type **attribute** must be placed in the script tag and must be set to “**text/javascript**”.

When a browser loads a document it **reads,** or **parses,** the code **sequentially**. Top, to bottom. So placing thescript blockin the **head** of the document other than in the **body** ensures that the JavaScript code is **parsed before the rest of the HTML content.** However, it should be noted that depending on the **timely requirement and intervention of resources needed by an application on the page, including DOM elements**, the script can equally be put at the **bottom** of the embedded html document.

A script block looks like this:

**<script language= “javascript” type= “text/javascript” >**

**<!- -**

**//- - >**

**</script>**

**nb :** The actual code will be placed in between the line beginning with <! -- and //- ->. **These lines are there to hide the script from old browsers.**  This syntax is hardly needed these days with the volley of of new devices in the market and enhanced browser.

Alternatively, the script can be written into an **external** JavaScript file and referenced back into the html document using the **src** attribute within the script tag. This is simply a text file saved with a “**.js**” file extension and containing **no** html tags**.** Wherethey **occur,** they should be **escaped** appropriately in order not tostop the page from rendering properly or throwing errors.

In the script tag which is placed in the header will now have a **src** attribute pointing the browser to the **url** (Universal Resource Locator) of the JavaScript file from its repository. As below, assuming this file is called librarycode.js. The closing **</script>** tag **must** persist.

**<script language= “javascript” type = “text/javascript”**

**src = “librarycode.js”></script>**

**Note: The space in between the opening and closed tags MUST be empty.**

### **Syntax rules. [ 5]**

* There is a **semi-colon** at the end of the JavaScript statement to comply with JavaScript syntax rules. This makes it a self contained statement.
* JavaScript is a **case sensitive** language and that “**ALERT**”, “**Alert**” and “**alert**” are all different words.
* All JavaScript **keywords** are ***identifier names*** and are specially, **reserved words**. They are in **lowercase** only.
* **Spaces, tabs and new lines** are collectively known as **white space**  and are completely **ignored** by JavaScript **parser**, so the code maybe **formatted** and **indented** and its appearance made more **human-readable**.
* It is often useful to add **comments** or **annotation** to JavaScript code as **explanation**, or when an **update** needs to happen. The parser sees any text between // and the **end** of that line as a **single-line** comment, which it ignores. Also any text, on **one** or **more lines** between **/\*** **\*/** is ignored.

**2.1 The Message Box.**

The **alert** command - Generating the *message box;*

Ex. 1

<html>

<head>

<title>scripting 1</title>

<script language ="JavaScript" type=”text/javascript”>

**alert** ( "Hello ! this is an alert message, and my first JavaScript programme. ");

</script>

</head>

<body>

</body>

</html>

The message box is achieved by calling the ***alert*** command in the code. When the document is loaded in the browser; the “**JavaScript engine**” implements the instructions contained in the code. **The code causes the browser to open a *grey box* (an alert box) bearing the message contained within the quotation marks inside the brackets**. The quotation marks do not appear and are used as **delimiters** for the string of characters representing the message. The quotation marks **MUST** always be there.

Using the console tab and break points in modern browsers, you can debug your code

**Note**: Using escapes. Alternate single, and double quotes: Both are interchangeable and must be consistent.

Ex. 2

<html>

<head>

<title>Alternating with single and double quotes</title>

<script language ="JavaScript" type=”text/javascript”>

**alert** ( "Hello ! this is an alert message for the ‘cool’ guys ");

**// alert** ( "Hello ! this is an alert message for the \”cool\” guys ");

</script>

</head>

<body>

</body>

</html>

## 2.2 Keywords

These **should not be used when using identifier names** and are part of the JavaScript language syntax (reserved words). In a professional editor all keywords will come out **bolded.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
| break | do | function | null | typeof |
| case | else | if | return | var |
| continue | export | import | switch | void |
| default | false | in | this | while |
| delete | for | new | true | with |
| abstract | debugger | goto | package | synchronised |
| boolean | enum | implements | private | throws |
| byte | extends | instanceof | protected | transient |
| catch | final | Int | public | try |
| char | finally | interface | short | call |
| class | float | long | static | apply |
| const | double | native | super |  |

**2.3 Variables**

A **variable** is a **place** where **data** is **stored** and can be **manipulated** within a JavaScript **program**. Any **letter, digit** or the **underscore** (i.e. a[A], 1[n], \_ ) may be used to name a variable. The only exception is **not to begin with a digit**.

In the following example a new variable called ***message*** has been created using the JavaScript keyword “**var**”. The text string is stored inside the variable named *message*. This variable is used in the **call** to the JavaScript alert function that opens an **alert dialogue box** displaying the **string** that is stored in the variable. Note the call also known as ***initialisation***to the variable name inside alert has no **delimiters.**

Ex 4

<script language = “javascript” type=”text/javascript”>

<!- -

var message = “First JavaScript variable”;

alert (**message**);

//- - >

</script>

**2.4** **Data types & Escape Sequences**

Variables in JavaScript can be used to store **numbers, text strings** or **boolean** values **(true/false)**.This is unlike certain programming languages like C++ and Java where specific data types have to be declared and can store data of declared type.

The example below creates variables with initial values of the **three different** data types supported by JavaScript. These three values are *first* passed to the *alert ( )* function, then the JavaScript keyword **typeof**  is used to **return** their data types for display in the alert dialog box.

The syntax “**+\n+**” inside the brackets displays the output for each value on a **new line** in the alert dialog box.

Ex 5

<script language=”javascript” type=”text/javascript”>

var

a = 0.06,

b = “Is JavaScript easy?”,

c = false;

alert (**typeof** a + “\n” + **typeof** b + “\n” + **typeof** c );

</script>

**Escape sequence**:

When a **character** in a string is **preceded** by the **backslash** character **“\”** there is a **special effect** on the **character immediately following it**. This is referred to as **escape sequence** as it allows the *letter* **following it to escape recognition but present a meaning as part of the JavaScript syntax**. The table below lists more escape sequences:

|  |  |  |
| --- | --- | --- |
| \b | Backspace |  |
| \f | Form feed |  |
| \n | New line |  |
| \r | Carriage return |  |
| \t | Tab |  |
| \' | Single quote that will not terminate a string |  |
| \" | Double quote that will not terminate a string |  |
| \\ | Single backslash character |  |

The escape sequence “ **\”** “ (be careful) is useful to incorporate quotation marks within a string without the string terminating early as shown in the example below.

alert (“We all say **\”** Javascript is handy **\”** “);