2 Server Side Scripting Through JavaScript

2.0 INTRODUCTION

JavaScript is a scripting language which provides HTML designers a programming tool that can be used for easy management of user interface. The user can put dynamic text into a HTML page, can make the page react to events or it can create and easily manipulate cookies. A JavaScript inserted in the HTML document allows a local recognition and processing (at the client level) of the events generated by the user such as those generated when the user scans the document or for management of fill in forms. For instance, we must recuperate the information referencing the client (name, address, payment etc). By inserting a JavaScript in the HTML page we can validate the data filled by the client. For example, we can validate the Credit Card Account, transaction history, etc. before it is submitted to the server.

JavaScript allows restructuring an entire HTML document for which we can add, remove, change, or reorder items on a page. In order to change anything on a page, JavaScript needs access to all elements in the HTML document. This access, along with methods and properties to add, move, change, or remove HTML elements, is given through the Document Object Model (DOM).

2.1 LEARNING OBJECTIVES

When you have finished this unit you will be able to understand the following:

- What is JavaScript?
- What is the difference between Java and JavaScript?
- Adding JavaScript to Your Document
- Embedding JavaScript
- Adding a JavaScript Block in the <HEAD>
- Linking JavaScript
- Future of JavaScript

2.2 INTRODUCTION TO JAVASCRIPT

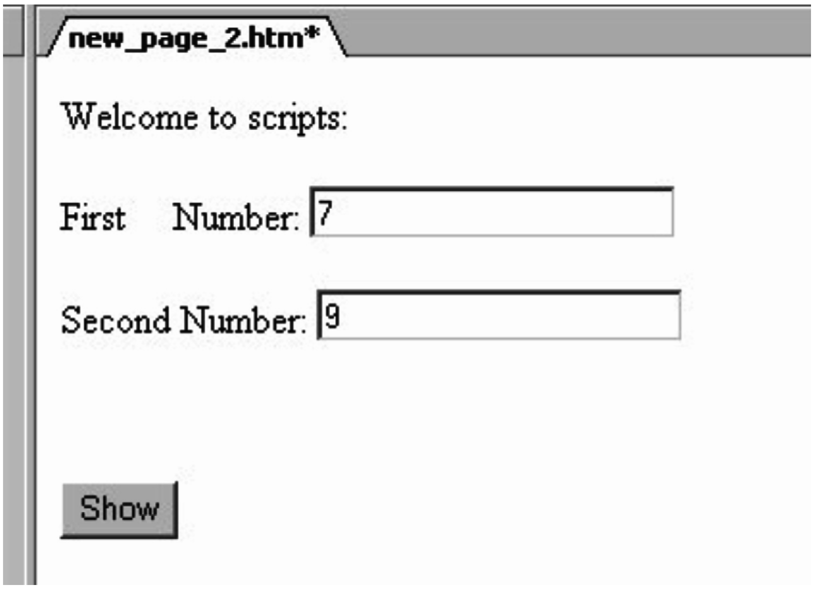
JavaScript can be considered as an extension to HTML (Hyper Text Markup Language), which allows programmer to incorporate some functionality in their web pages. The moment programmer presses the SUBMIT button, he may not necessarily have to invoke a CGI (Common Gateway Interface) script to do the processing. If it is something simple, he can do the processing locally using JavaScript and give back the results. JavaScript can also be used

document.getElementById('sampleparagraph').innerHTML – this is interpreted as HTML
document.getElementById('sampleparagraph').innerText – this is interpreted as text

If the content of sampleparagraph is "<b> inner text</>" then:

- innerText would display as <b>inner text</b>
- innerHTML would display as inner text.

Figure 2.1 shows a HTML form containing a VBScript and a JavaScript.


new_page_2.htm*

Welcome to scripts:

First Number: 7

Second Number: 9

Fig. 2.1 A JavaScript Example

2.2.1 The Client-Side Features of JavaScript

- Simple to use
- Dynamic (responds to events)
- Object-based

Example

```
<html>
<head>
<meta name="GENERATOR" content="Microsoft FrontPage 5.0">
<meta name="ProgId" content="FrontPage.Editor.Document">
<meta http-equiv="Content-Type" content="text/html; charset=windows-1252">
<title>New Page 2</title>
<script type="text/javaScript" language="javaScript">
```

Table 2.3 Style properties

Property	*Description*
style.background	Sets or retrieves the background picture tiled behind the text and graphics in the object.
style.backgroundAttachment	Sets or retrieves how the background image is attached to the object within the document.
style.backgroundColor	Sets or retrieves the color behind the content of the object.
style.backgroundImage	Sets or retrieves the background image of the object.
style.border	Sets or retrieves the width of the border to draw around the object.
style.borderBottom	Sets or retrieves the properties of the bottom border of the object
style.borderBottomColor	Sets or retrieves the color of the bottom border of the object.
style.borderBottomStyle	Sets or retrieves the style of the bottom border of the object.
style.borderBottomWidth	Sets or retrieves the width of the bottom border of the object.
style.borderCollapse	Sets or retrieves a value that indicates whether the row and cell borders of a table are joined in a single border or detached as in standard HTML.
style.borderColor	Sets or retrieves the border color of the object.
style.borderLeft	Sets or retrieves the properties of the left border of the object
style.borderLeftColor	Sets or retrieves the color of the left border of the object.
style.borderLeftStyle	Sets or retrieves the style of the left border of the object
style.borderLeftWidth	Sets or retrieves the width of the left border of the object.
style.borderRight	Sets or retrieves the properties of the right border of the object.
style.borderRightColor	Sets or retrieves the color of the right border of the object.
style.borderRightStyle	Sets or retrieves the style of the right border of the object.
style.borderRightWidth	Sets or retrieves the width of the right border of the object.
style.borderStyle	Sets or retrieves the style of the left, right, top, and bottom borders of the object
style.borderTop	Sets or retrieves the properties of the top border of the object.
style.borderTopColor	Sets or retrieves the color of the top border of the object.
style.borderTopStyle	Sets or retrieves the style of the top border of the object.
style. borderTopWidth	Sets or retrieves the width of the top border of the object.
style.borderWidth	Sets or retrieves the width of the left, right, top, and bottom borders of the object.
style.bottomMargin	Sets or retrieves the bottom margin of the entire body of the page.
style.color	Sets or retrieves the color of the text of the object.
style.font	Sets or retrieves a combination of separate font properties of the object. Alternatively, sets or retrieves one or more of six user-preference fonts.
style.fontFamily	Sets or retrieves the name of the font used for text in the object.
style.fontSize	Sets or retrieves a value that indicates the font size used for text in the object.
style.fontStyle	Sets or retrieves the font style of the object as italic, normal, or oblique.

Contd...

Contd...

style.fontVariant	Sets or retrieves whether the text of the object is in small capital letters.
style.fontWeight	Sets or retrieves the weight of the font of the object.
style.margin	Sets or retrieves the width of the top, right, bottom, and left margins of the object.
style.marginBottom	Sets or retrieves the height of the bottom margin of the object.
style.marginHeight	Sets or retrieves the top and bottom margin heights before displaying the text in a frame.

The JavaScript sentences involving text strings can be split up within the text string by using the\(backslash) character.

The multiline comments can be defined between /* and */; the one line or the inline comments can be defined by using // (two slashes). The extra space is ignored and the sentences are case sensitive. The; (semicolon) ending sentence character is optional for sentences defined alone on a line and compulsory for separating the commands defined in the same line (generally the inline scripts).

2.2.2 New Features of JavaScript in Netscape Navigator 3.0

- Can change GIF and JPEG images automatically, at specified time intervals, by clicking a button or icon, or moving the mouse over an object
- Can detect the presence of plug-ins on a webpage and tailor the user interface accordingly
- Can communicate with plug-ins on the same page
- Java applets can communicate with JavaScripts
- The server side of JavaScript (requires LiveWire for SSI (server-side includes) or IIS/PWS (Internet information server/ personal web server) for ASPs (application server pages).

2.3 WHEN TO USE JAVASCRIPT?

JavaScript can be used in any of the following :

- To move action from the server to the client
- To locally validate forms field's before submitting it to the server
- HTML documents to respond to local events
- Webpage developer can communicate information to and from applets and plug-ins
- Server-side JavaScript supports unique personalized user profiles
- Server-side JavaScript enables users to access databases

2.4 WHAT IS JAVASCRIPT? IS IT ADVANCED HTML? JAVA? SOME FORM OF ACTIVE X?

Actually, none of the above. JavaScript is JavaScript, and that's that. One of the most common preconceptions about JavaScript is that it's somewhat related to Java (since they have very similar names). Ironically, JavaScript and Java, besides the fact that they are all programming languages, are completely different. We won't go into detail what Java is, but we will say this: it's something we definitely won't be taking on in the near future.

An example of JavaScript

- The university arms at the top of this page are a link to the university home page. Just point at the image, without clicking.
- The gray coat of arms becomes colored and the message on the status bar at the foot of the window changes
- The same happens when you let the mouse move over the large ITS
- Having two image files for each image on the page, and using a scrap of JavaScript to tell Netscape to swap one for the other whenever the mouse is over the link and revert to the original image when the mouse leaves the link, achieve this.

2.4.1 How is it Done?

The HTML code shown below is not very different from what you have seen before. There is an <A ... > tag, followed by an <img ...> tag rounded off with a </A> tag, which means that the image is being used as the "link" to another page. This much should be familiar to you if you have created HTML files in the past.

There are two main differences:

- There is a fragment of JavaScript shown in bold in the **<A...>** tag
- there is a **NAME** attribute given to the image in the **<IMG...>** tag

The image is called "home" in the **<IMG...>** tag and this name is then used by the scripting inside the **<A...>** tag.

Note the two "event-handlers" in the **<A...>** tag:

- **onMouseover="..."** -- tells the browser what to do when the mouse is over the link:
  - o use the file "armscolour.gif" as the source for the image called "home"
  - o make the Window's status line say "Got to University Home Page"
- **onMouseout="..."** -- tells the browser what to do when the mouse leaves the link:
  - o use the file "armsgrey.gif" as the source of the image called "home"
  - o change the Window's status bar to use the title of the current document

2.4.2 Using and Placing JavaScripts in a HTML Page

In the following paragraphs are introduced some examples of using and placing JavaScripts in a HTML page

<script>

</script>

The actual JavaScript codes will fall inside this tag

Example

```
<html>
<head>
<title>
Page containing JavaScript
```

```
</title>
</head>
<body>
<script type=”text/javaScript”>
document.write(“This text is displayed by a JavaScript!”)
</script>
</body>
</html>
```

Comments

The tag <script> have the attribute „type”, that specifies the script type (JavaScript in our case). This script composed by a single command that displays inside the page the text: “This text is displayed by a JavaScript!”. If you want include many commands on the same line this must be separated by the ”;” (semi colon) character.

The concatenation of text string is realized by using the + character, for example the expression “This text is “ + “concatenated.” will produce the string “This text is concatenated.”

The „/” haves a special meaning for the HTML language and consequently when we want display the slash character itself we must precede (prefix) this by a „\” (backslash), as illustrated in this example:

```
document.write(“<i>”+”The Operator + is Concatention!”+”<\/i>”)
```

If the script is included in a comment tag (<!--) then the browsers that do not „know” (interpret) JavaScript will not display in the page the script text (script source), as shown in the following sample:

```
<!--
document.write(“<i>”+”This text is displayed by a JavaScript!”+”<\/i>”)
//-->
```

The browsers that know JavaScript will process the script, even this included in a comment line.

The string „//”, comment in JavaScript, tells the browser do not process the line „-->”. We can not use the sintax „//<!--”, because a browser that does not interpret JavaScript will display that string „//”.

2.4.3 JavaScript in Heading

If we want to be sure that the script executes before displaying any element in the page, then we can include this in the heading part of the HTML page (file). The JavaScript in the head section will be executed when called, or when an event is triggered.

Example

```
<html>
<head>
```

```
<title>
Page with JavaScript
</title>
<script type=”text/javaScript”>
document.write(“This text is displayed by a JavaScript!”)
</script>
</head>
<body>
<P> This text must appear in the page after the execution of the JavaScript.
</body>
</html>
```

The number of scripts placed in the head section and in the body of a HTML page are unlimited.

2.4.4 External JavaScripts

A JavaScript can be stored into an external script file from where we can use it in many Web pages. In that way the script is written only once and in every HTML file we want use is enough to invoke the file containing the script. The stored script cannot contain the tag <script> or his pair </script>.

The steps followed when using externally stored scripts are:

1. The creation of the external file containing the script lines, for example the line:
   document.write(“Text from an external stored script.”)
2. The file is saved with the wanted name and the extension js (Javascript), for example we name the file scriptex.js
3. In the HTML pages we will include the stored script file by adding the following script:

   ```
   <script type=”text/javaScript” src=”scriptex.js”>
   </script>
   ```

   The „src” attribute of the tag <script> allows specifying the file containing the script we want execute.var

2.5 DEFINING AND USING VARIABLES IN JAVASCRIPT

JavaScript can contain variable definitions and references to the variables. The variables can be used to store values and the references to the values can be done by referencing the name of the variable. The lifetime of variables can be:

- **for variables declared within a function** - can only be accessed within the function; they created when encountered their declaration as the function progress and destroyed when exiting; they called local variables and you can use the same name in different functions;

- **for variables declared outside a function -** can be accessed anywhere in the page; the name must be unique at that level; the lifetime of these variables starts when they are declared, and ends when the page is closed.

The variable declaration can include an assignment and can be done using one of the sentences:

```
var variableName=somevalue
or
variableName=somevalue
```

In the following example, one can define a variable called „name” that is initialized with the value „This text contained by the variable called name”, and later on referenced in a write sentence:

```
<html>
<head>
<title>
Page with JavaScript
</title>
</head>
<body>
<script type=”text/javaScript”>
<!--
var name= “ This text contained by the variable called name”
document.write(“<i>”+name+”<\/i>”)
//-->
</script>
</body>
</html>
```

The variables, functions and, object names are case-sensitive and must begin with a letter or an _ (underscore) character.

2.6 DYNAMICALLY CHANGING THE DOCUMENT’S BACKGROUND COLOR

Now in this section you will see how we can change the background color of the document using JavaScript. See the following code:

```
<script>
document.bgColor=”blue”
</script>
```

OUTPUT: (Refer Fig. 2.2)

Fig. 2.2 Output of the Above Code

You can change blue to any color name, or the color's hex representation (i.e.: #000000). This is a very simple example of JavaScript at work; changing the background colour isn't exactly something good-old HTML can't do easily all by itself.

2.7 STATUS BAR MESSAGE

Using JavaScript, you can display messages in the status bar of your browser below. This is accomplished by setting a string value to the "window.status" property. For example:

```
<script>
window.status="Welcome to my homepage"
</script>
```

By doing the above, the message "Welcome to my homepage" is shown in the status bar. One trick you may have seen on the web is a status bar message that is initiated only when the user moves her mouse over a link:

2.8 JAVASCRIPT DIALOG BOX

JavaScript dialog boxes are interesting little "pop-up" boxes that can be used to display a message, ask for confirmation, user input etc. They're very easy to create, not to mention cool!

Three types of dialog boxes exist in JavaScript- alert, confirm, and prompt. Examples of each of them are given below.

1. **Alert:**

```
<script>
alert("Welcome, my friend!")
</script>
```

Fig. 2.3 Output of the Image

In the following is a “random” quote example, where a quote out of three is randomly displayed each time this page is loaded (Reload this web page to see another quote):

“You can take away my life, but you can never take away my freedom!”

The source code for the above quote is:

```
<script>
var whichquote=Math.round(Math.random())*3
if (whichquote<=1)
document.write('"You can take away my life, but you can never take away my freedom!"')
else if (whichquote<=2)
document.write('"I\'ll be back"')
else if (whichquote<=3)
document.write('"You can count on it"')
</script>
```

The key here is the code:

```
var whichquote=Math.round(Math.random())*3
```

OUTPUT: This text will move randomly.--”You can take away my life, but you can never take away my freedom!”

2.11 JAVASCRIPT METHODS

JavaScript is an object based programming language and uses objects (as shown in Tables 2.4 and 2.5). It has many built-in objects such as Area, Image, Date, Window, and Document, and also allows user define his own objects. In the following table some methods for document and window are explained:

Contd...

	{
	var buttonpressed=confirm("Press a button")
	if (buttonpressed ==true)
	{
	document.write("You pressed the OK button!")
	}
	else
	{
	document.write("You pressed the Cancel button!")
	}
	}
	window.open("URL",
	"name_of_new_window",
"specifications")	Opens a new browser window for the page indicated by the URL argument. The window can be referenced by the name "name_of_new_window" and can be customized by the values supplied by the "specifications" argument.
	Example:
	<html>
	<head>
	<script type="text/javaScript">
	function open_win_ase()
	{
	window.open("http://www.ase.ro","_blank","toolbar=yes,
	location=yes, directories=no, status=no, menubar=yes,
	scrollbars=yes, resizable=no, copyhistory=yes, width=400,
	height=300")
	}
	function open_win_avrams()
	{
	window.open("http://www.aavrams.ro","_blank","toolbar=yes,
	location=yes, directories=no, status=no, menubar=yes,
	scrollbars=yes, resizable=no, copyhistory=yes, width=400,
	height=300")
	}
	</script>
	</head>

Contd...

Contd...

	<body>
	<form>
	<input type=”button” value=”Faculty” onclick=”open_win_ase()”>
	<input type=”button” value=”Course Notes”
	onclick=”open_win_avrams()”>
	</form>
	</body>
	</html>

Table 2.5 Common Escape Sequences for text display formatting are represented by:

Ampersand	\&
Double quote	\”
Single quote	\’
Newline	\n
Form feed	\f
Carriage return	\r
Backslash	\\
Backspace	\b
Tab	\

2.12 DOCUMENT OBJECT MODEL (DOM)

The Document Object Model defines HTML documents as a group of objects and provides access to every element, identified uniquely by intermediate of an id attribute, in a document.

Any element may be accessed (by using the method getElementById) and modified by a snippet of JavaScript. The Window object is the top level object in the JavaScript hierarchy (it represents a browser window). A Window object is created automatically with every instance of a <body> or <frameset> tag.

Examples

(a) This sample displays the message „This is first paragraph! Click, and see”. If you click somewhere in the displayed text it displays:

“The background is: the current color name will be changed in Yellow!”:

```
<html>
<head>
<title>Using DOM</title>
<script language=”javaScript”>
<!--
```

```
function xalert()
{
var x=document.getElementById("par1");
x.style.background="red";
alert("The background is:" + x.style.background+"\n Will be changed in Yellow!")
if(x.style.background=="red")
{
x.style.background="yellow";
}
else
{
x.style.background="red";
}
}
-->
</script>
</head>
<body>
<p id="par1" onclick="xalert()" >This is first paragraph! Click, and see</p>
</body>
</html>
```

(b) This sample uses innerHTML to change dynamically the header identified by „chgheader":

```
<html>
<head>
<script type="text/javaScript">
function getValue()
{
var x=document.getElementById("myHeader")
alert(x.innerHTML)
}
function chgval()
{
```

```
document.getElementById("chgheader").innerHTML="My Header (Changed)"
}
</script>
</head>
<body>
<h1 id="myHeader" onclick="getValue()">This is first header</h1>
<p>Click on the header to alert its value</p>
<h2 id="chgheader" onclick="chgval()">This is the second header</h2>
<p>Click on the header to change its value</p>
</body>
</html>
```

2.13 USING AND DEFINING FUNCTION

A function contains a set of statements which is executed when triggered by an event or a call to that function. In JavaScript it is possible to use the Java language intrinsic functions or user defined functions (need to be defined before any usage). In the case of user defined functions it is preferable that the definition is made in the head section of the HTML page to be sure (or to ensure) that they are loaded before calling. This is required because the browser starts processing the HTML page before completely downloading this from server. You may call a function from anywhere within a page (or even from other pages if the function is embedded in an external script).

The general syntax of a function is:

```
function function_name([argument1,argument2,etc])
{
some_statements
[return expression]
}
```

where:

function_name is the name of the function you want to have; argument1,argument2,... the name for the function parameters, if it has, is allowed not pass any parameter to the function; some_statements generally variable declarations and executables statements that describe the steps of the algorithm you model. They define together with the return statement (if present) the body of the function; expression is the expression whose evaluation will represent the returned value. A function can return (the sentence return must be present in the body) a value or not (the return statement is not appearing between those of the function's body); A function can be invoked in one of the ways:

- without arguments:

  function_name()

```
}
//-->
</script>
</head>
<body>
<script type=”text/javaScript”>
<!--
var name= “ This text contained by the variable called name”
document.write(“<i>”+name+”<\/i>”)
alert(mess)
alert(suma(“This text is a ”,”<String Concatenation>”))
document.write(“This digit is the result of adding 2 to 7 by using the user defined function
suma:”+suma(2,7))
//-->
</script>
</body>
</html>
```

C. Example 3: In this example the JavaScript defined in the head part of the page and in the body part in a href tag.

```
<html>
<head>
<title>A Page with scripts</title>
<script type=”text/javaScript”>
function chgbgcolor()
{
document.bgColor=”green”
}
function chcolor()
{
document.bgColor=”orange”
}
</script>
<script id=clientEventHandlersJS language=javaScript>
<!—
function window_onload() {
document.bgcolor=”orange”
}
//-->
</script>
<script language=javaScript for=window event=onload>
<!—
return window_onload()
//-->
</script>
</head>
<body>
<p>The page not empty </>
<a href="javaScript:chcolor(); alert('The background was changed!')">This is a script in
href</a></p>
<h1 id="header1" onclick="chgbgcolor()"> IT4B: </h1>
<h2 id="header2">Errata</h2>
</body>
</html>
```

2.14 ASSIGNMENTS AND EXPRESSIONS

The general syntax for assignment is:

variable = expression

The interpretation of this is: the variable before the assignment operator is assigned a value of the expression after it, and in the process, the previous value of variable is destroyed. An expression can be an arithmetic expression, a logical expression or expression on character string. It can be a variable, a constant, a literal, or a combination of these connected by appropriate operators.

An assignment statement stores a literal value or a computational result in a variable and is used to perform most arithmetic operation in a program.

2.14.1 Arithmetic Expression

An arithmetic expression is used to perform arithmetic operations. The syntax of arithmetic expression is:

<operand1><arithmetic_operator><operand2>

where:

- <operand1>,<operand2> can be numeric variables, constants or arithmetic expressions (or calls to functions that return numeric values)

– arithmetic_operator (binary operators) is one of those shown in Table 2.6

Table 2.6 Arithmetic Operators

Operator	*Description*	*Example Suppose x=2*
+	Addition	x+2 = 4
-	Subtraction	5-x = 3
*	Multiplication	x*5 = 10
/	Division	9/3 = 3; 5/2 = 2.5
%	Modulus (division remainder)	5%2 = 1; x%2 = 0; 10%5 = 0
++	Increment By One*)	x++ = 3
--	Decrement By One*)	x-- = 1

The increment and decrement operators can either be used as a pre- or a post-operator:

Post-Increment: the line of code is	Pre-Increment: the increment or decrement is
executed as then the increment/decrement	performed before whatever other operations
is performed.	are present within the given line of code.
y = x++ is equivalent to the sequence:	y = ++x is equivalent to the sequence:
y=x	x=x+1
x=x+1	y=x

For arithmetic expressions the assignment operator can be combined with the arithmetic ones to define compact expressions as shown in Table 2.7.

Table 2.7 Assignment Operators

Operator	*Example*	*Is The Same As*
=	x=y	x=y

+=	x += y
	x+=y-12
		x=x+y
		x = x - (y + 12)

-=	x-=y x
	-= y +12
		x=x-y
		x = x - (y + 12)

*=	x*=y
	x *= 3 + y
		x=x*y
		x = x * (3 + y)

/=	x /= y
	x/=y+2
		x=x/y
		x=x/(y+2)

%=	x %= y
	x%=y+2
		x=x%y
		x=x%(y+2)

2.14.2 Logical Expression

A logical expression can be simple. It is used to check the logic and will return Boolean value (refer to Tables 2.8 and 2.9).

The general syntax:

<variable>[<relation_operator><variable>]

or

<variable>[<relation_operator><constant>]

<relation_operator>::=<|<=|>|>=|==|!=

Table 2.8 Shows and explains the comparison operators.

• complex, with the general syntax:

e1 && e2	- logical and;
e1 || e2	- logical or;
! e1	- logical not.

<logical_expression1><logical_operator><logical_expression2>

where the logical_operator can be:

&& (and; two ampersand character), || (or; two vertical bar character) as binary operators (connectives);

! (not) as unary operator.

The precedence of evaluation of logical operators is Not, And, Or. The logical operator together with the truth table defining the way they operate and usage examples are shown in Table 2.10.

Table 2.9 Comparison Operators

Operator	*Description*	*Example Suppose x=2*
==	is equal to	x == 3 returns false
===	Is equal to (Is equal to (checks for both value and data type)	y="2" x==y returns true x===y return false (x integer; y string)
!=	is not equal	x != 3 returns true
>	is greater than	x > 3 returns false
<	is less than	x < 3 returns true
>=	is greater than or equal to	x >= 3 returns false
<=	is less than or equal to	x <= 3 returns true

Table 2.10 Logical Operators

Operator	*Description*			*Example Suppose x=2; y=3*
&&	x	y	and	(x < 9 && y > 1) returns true (x < 9 && y < 1) returns false (x > 9 && y > 1) returns false (x > 9 && y < 1) returns false
	T	T	T	
	T	F	F	
	F	T	F	
	F	F	F	

Contd...

Contd...

||	x	y	or	(x < 9 || y > 1) returns true
	T	T	T	(x < 9 || y < 1) returns true
	T	F	F	(x > 9 || y > 1) returns true
	F	T	F	(x > 9 || y < 1) returns false
!	x		not	!(x == y) returns true
	F		T	!(x > y) returns false
	T		F	

2.14.3 String Expression

An expression on character string can be built using following types of string operators (Refer to Table 2.11) to perform string type operations:

Table 2.11 String operators

- the concatenation operator: +	intrinsic functions for extracting substrings from a string variable or string constant
Right(string,number_of_characters) -	extracting substring from the end
Left(string,number_of_characters) -	extracting substring from the beginning
Functions that Manipulate Strings:	
Cstr(expression) –	convert the expression in a character string;
Lcase(string_expression)	convert the string in lower case;
Ltrim(string), Rtrim(string), Trim(string) –	eliminates the spaces (trailing) from left (leading blanks), right and, respectively left-right;
Str(number)	converts number in string;
Ucase(string)	converts string to uppercase.

The code sequence below is string concatenation by using the concatenation operator + example.

```
text1 = “Faculty of”
text2 = “Business Administration”
text3 = text1 +” “+ text2
```

The variable text3 now contains the string “Faculty of Business Administration”. The concatenation with the space character “ “ is realized to separate the strings (generally stored with no trailing blanks).

2.15 CONDITIONAL EXECUTION

A script can include branches (If...Then...Else...) allowing the definition of conditional executions similar to the example in the following sequence:

```
operations;
}
```

if...else if....else statement

This statement allows to select one of many blocks of code to be executed .

```
if (condition1)
{
code to be executed if condition1 is true
}
else if (condition2)
{
code to be executed if condition2 is true
}
else
{
code to be executed if condition1 and condition2 are not true
}
```

The logical condition <conditionx> is a logical expression that will be evaluated either to True or either to False. The logical conditions can be simple or complex logical conditions.

A simple logical condition has the general syntax:

<variable> [<relation_operator ><variable>]

or

<variable> [<relation_operator ><constant>]

The relation_operator can be one of (Refer to table 2.12):

Table 2.12 Relation Operator

Relation Operator	*Interpretation*
<	Less than. Example: delta < 0
<=	Less than or equal. Example: delta <= 0
>	Greater than. Example: delta > 0
>=	Greater than or equal. Example: delta >= 0
==	Equal to. Example: a == 0
!=	Not equal. Example: a!=0

The simple logical conditions will be connected by the AND, OR, and NOT logical operators to form complex conditions. The logical operators are evaluated in the order NOT, AND, and OR. The change of the natural order of evaluation can be done by using parenthesis in the same way for arithmetic expressions.

```
}
```

- expression_int is an expression that must produce an integral value (int);
- constant_expressioni must be a constant expression;
- the label default: can be used only once.

The expression_int is also called the selector of instruction case.

- if the value of the selector doesn't fit to a constant the operations specified on branch Default (otherwise) will be executed;
- the values of constants must be unique for a switch sentence.

Example

The sequence below uses the switch statement to find out the Romanian name for the day of the week of a date.

```
<HTML>
<HEAD>
<meta name=vs_defaultClientScript content="JavaScript">
<TITLE></TITLE>
<META NAME="GENERATOR" Content="Microsoft Visual Studio">
<META HTTP-EQUIV="Content-Type" content="text/html; charset=UTF-8">
<script type="text/javaScript">
/* The sequence will write the name of the day in romanian
(Sunday=0, Monday=1, Tuesday=2, etc)
*/
function RODay(aDayNumber)
{
switch (aDayNumber)
{
case 0:
return "Duminica"
case 1:
return "Luni"
case 2:
return "Marti"
case 3:
return "Miercuri"
case 4:
return "Joi"
case 5:
return "Vineri"
case 6:
return “Sambata”
default:
alert(“What day is it? \n The computer is virused or hardware damaged !”)
return “What day is it? \n The computer is virused or hardware damaged !”
}
}
</script>
<script id=clientEventHandlersJS language=javaScript>
<!--
function Button1_onclick() {
var i=0
var datadeazi=new Date()
var ziua=datadeazi.getDay()
for (i=ziua;ziua<=6;ziua++){
document.write(ziua+”: “ + RODay(ziua)+”<br />”)
}
}
//-->
</script>
<script language=javaScript for=Button1 event=onclick>
<!--
return Button1_onclick()
//-->
</script>
</HEAD>
<BODY>
<p>This page contains a JavaScript exploiting the switch sentence.</p>
<p>
<input id=Button1 type=button value=”Press This”></p>
</BODY>
</HTML>
```

2.15.3.1 Conditional Operator (?)

The conditional operator has the syntax:

(conditional_expression) ? true_case_expression: false_case_expression where: <conditional_ expression> is a logical expression that will be evaluated either to True or either to False.

Is a very good idea to include the expression in parenthesis (to enforce his evaluation).

<true_case_expression> is the expression whose evaluation will be returned if the conditional expression evaluates to True

<false_case_expression> is the expression whose evaluation will be returned if the conditional expression evaluates to False.

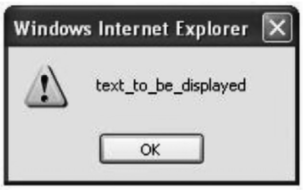
Example

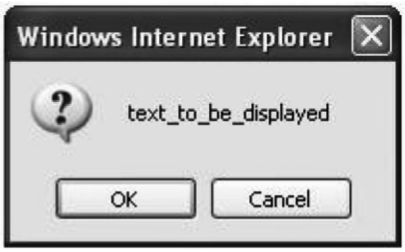
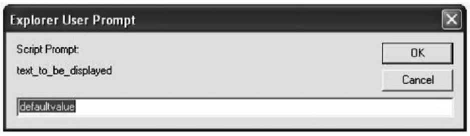
```
<HTML>
<HEAD>
<meta name=vs_defaultClientScript content=”JavaScript”>
<TITLE></TITLE>
<META NAME=”GENERATOR” Content=”Microsoft Visual Studio”>
<META HTTP-EQUIV=”Content-Type” content=”text/html; charset=UTF-8”>
</HEAD>
<BODY>
<script language=javaScript>
var TotalBalance, savings=300
TotalBalance =(savings==0) ? 0:(savings*1.03)
// TotalBalance is now 309
document.write(“Total Balance is now: “ + TotalBalance)
</script>
</BODY>
</HTML>
```

2.16 POPUP BOXES

In JavaScript we can create three kinds of popup boxes by invoking the associated intrinsic function (refer Table 2.13). Their functions are:

Table 2.13 Popup functions

Function	*Description*
alert(”text_to_be_displayed”)	Displays an alert box containing the message passed in argument and an OK button. This call produces the box:  Windows Internet Explorer — text_to_be_displayed — OK

confirm(”text_to_be_displayed”)	Displays a box containing the message passed in argument provided with an OK (confirm) and Cancel (denny) button. This call produces the box:  Windows Internet Explorer text_to_be_displayed OK Cancel
prompt(”text_to_be_displayed”, ”deaultvalue”)	Displays an input dialog box provided with a text box to fill data and an OK (return the value typed to the caller) and Cancel (return a Null value to caller). This call produces the box:  Explorer User Prompt Script Prompt: text_to_be_displayed OK Cancel defaultvalue

2.17 CYCLES

The repeating structure repeats a block of statements till a condition is True or Until a condition becomes True. The repetition of steps in a program is known as Loop. The repeated execution of a sequence of instructions (loop) can be done by using the looping statements while ,do... while and for.

Example

```
<html>
<head>
<title>
Page containing loop
</title>
</head>
<body>
<script type=”text/javaScript”>
<!--
for (i=0; i<5; i++)
{
document.write(“Step i: “+i+”<br>”)
}
```

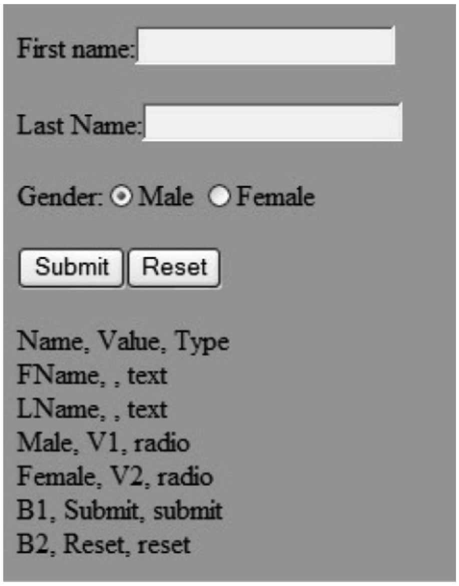
```
file=”C:\Documents and Settings\Vio\My
Documents\My Webs\_private\form_results.csv”
s-format=”TEXT/CSV” s-label-fields=”TRUE” -->
<!-- This is the description of the form -->
<p>First name:<input type=”text”
name=”FName” size=”20”></p>
<p>Last Name:<input type=”text”
name=”LName” size=”20”></p>
<p>Gender:<input type=”radio” value=”V1”
elements
name=”Male” checked>Male
<input type=”radio” name=”Female” value=”V2”>Female</p>
<p><input type=”submit” value=”Submit” name=”B1”>
<input type=”reset” value=”Reset” name=”B2”></p>
</form>
<script type=”text/vbscript”>
document.write(“Name, Value, Type “+”<br />”)
</script>
<script type=”text/javaScript”>
for (i=0; i < document.forms[0].elements.length;i++)
{
document.write(document.forms[0].elements[i].name + “, “);
//syntax below uses the name attribute of the form to access the form’s elements
document.write(document.forms[0].elements[i].value + “, “);
document.write(document.forms[0].elements[i].type + “<br />”);
}
</script>
</body>
</html>
```

2.18 FOR ... IN STATEMENT

```
for (variable in object)
{
code to be executed
 }
```

Example

In this example is defined an array object called divisions and the first three elements initialized. The for...in sentence will fill in the HTML document the lines initialized in the array.



```
Name, Value, Type
FName, , text
LName, , text
Male, V1, radio
Female, V2, radio
B1, Submit, submit
B2, Reset, reset
```

Fig. 2.4 Accessing HTML form

```
<html>
<body>
<script type=”text/javaScript”>
var x, nr
var divisions = new Array()
divisions[0] = “English”
divisions[1] = “French”
divisions[2] = “German”
for (x in divisions)
{
nr=x/1+1;
document.write(nr+”: “+divisions[x] + “<br />”)
}
</script>
</body>
</html>
```

that produces the output :

```
1: English
2: French
3: German
```

2.19 USING EVENTS TO TRIGGER SCRIPT EXECUTION

Some events that can be associated with HTML pages are represented by the following (Refer Table 2.14):

Table 2.14 Events associated with trigger script execution

Event	*Occurs when...*
onabort	a user aborts page loading
onblur	a user leaves an object
onchange	a user changes the value of an object
onclickt	a user clicks on an objec
ondblclick	a user double-clicks on an object
onerror	an error occurs
onfocus	a user makes an object active
onkeydown	a keyboard key is on its way down
onkeypress	a keyboard key is pressed
onkeyup	a keyboard key is released
onload	a page is finished loading (in Netscape, this event occurs during the loading of a page).
onmousedown	a user presses a mouse-button
onmousemove	a cursor moves on an object
onmouseover	a cursor moves over an object
onmouseout	a cursor moves off an object
onmouseup	a user releases a mouse-button
onreset	a user resets a form
onselect	a user selects content on a page
onsubmit	a user submits a form
Onunload	a user closes a page; a frequent usage is to deal with cookies.

Table 2.15 below shows common usage of events:

Table 2.15 Common Commands for usage of Events

Event	*Usage*
onload, onunload	The onload event is often used to check the visitor's browser type and browser version, and load the proper version of the web page based on the information. Both events frequently used to deal with cookies.
onfocus, onblur, onchange	Generally used in combination with validation of form fields.
onsubmit	Is used to validate all form fields before submission (is possible to deal with
logical validation involving more fields from the form).	
onmouseover, onmouseout	Generally used for creating "animated" buttons.

In the following example is shown an inline JavaScript code (without the tags <script> and </script>). The web page contains a button whose property “Caption” has the value “ASE”. When the event “onclick” occurs (when clicking the button) it is called the function “open” (member of “windows” functions group), having in arguments the arguments required to open the ASE site home page in a window called internally “ase_home”.

Example

```
<html>
<head>
<title>
Command button link
</title>
</head>
<body>
<form>
<input type=”button” value=”ASE” onclick=’window.open(“http://www.ase.ro”, “ase_
home”)’>
</form>
</body>
</html>
```

Example

```
<html>
<head>
<title>Pagina cu Cronometru </title>
<script type=”text/javaScript”>
function startTime()
{
var today=new Date()
var h=today.getHours()
var m=today.getMinutes()
var s=today.getSeconds()
// add a zero in front of numbers<10
m=checkTime(m)
s=checkTime(s)
document.getElementById(‘txt’).innerHTML=h+”:”+m+”:”+s
t=setTimeout(‘startTime()’,500)
}
function checkTime(i)
```