SECTION - II: JavaScript

8. INTRODUCTION TO JAVASCRIPT

JAVASCRIPT IN WEB PAGES

Today's web sites need to go much beyond HTML. There is a definite need to allow users, browsing through a web site, to actually interact with the web site. The web site must be intelligent enough to accept users input and dynamically structure web page content, tailor made, to a user's requirements.

This may be as simple as ensuring a web page delivered to a user, having a background color that the user is comfortable with or as complex as delivering a web page with special textual formatting for a user with visual disabilities.

Users, who browse through a web site today, prefer to choose to view what interests them. Hence even the **content** of a web page needs to be dynamic, based on what a user wishes to see.

This requires a web site development environment that will allow the creation of **Interactive** web pages. These web pages will accept input from a user. Based on the input received return customized web pages, both in content and presentation, to the user.

In the absence of any user input the web site must be intelligent enough to return a default web page containing predetermined information and text formatting.

This calls for a web site development environment with coding techniques capable of accepting a client's requests and processing these requests. The result of the processing being passed back to the client via standard HTML pages.

The need to return standard HTML pages, that map to a user's input, is due to the fact that browsers use HTTP to communicate with a web server and are designed to interpret and render HTML on a client's machine.

Capturing user requests is traditionally done via a **Form**. Hence the web site development environment needs to have the facilities to create forms.

After a form captures user input, the form must have a *built in* technique for sending the information captured back to the web server for processing. Processing user requests is generally done via scripts (*small programs*) that are based on the server.

The web site development environment should also provide the facility for **Validating** user input. Invalid user input, will either cause data to be sent back from the web server to the browser, which is not what the user wants or give rise to an error message being sent back to the browser from the web server. Neither of which would really attract repeat visits to the web site.

Hence, the web site development environment must also facilitate coding which runs in a browser at client side for data validation. Most development environments offer standard constructs for data validation. Standard programming constructs are:

- Condition checking constructs
- Case checking constructs
- Super controlled Loop constructs

Additionally, all development environments provide syntax to create and use memory variables, constants, and functions.

If the development environment is object oriented it will provide an object hierarchy to work with. An Object Oriented Programming (OOP) environment always offers event driven programming. This means that the programming environment will recognize object based events and allow the connection of code snippets to these events. When an event occurs, the code snippets will execute.

All these facilities and more are available in JavaScript.

Netscape and JavaScript

JavaScript is a scripting language (*web site development environment*) created by Netscape hence JavaScript works best with the Netscape suite of Client and Server products.

The Netscape client **browser** product is called Netscape Communicator. The default scripting language that Netscape Communicator understands is JavaScript.

One Netscape server product, from its suite of server products, is Netscape Commerce Server. The default scripting language that Netscape Commerce Server understands is JavaScript.

Database Connectivity

Netscape has a product called **Live Wire**, which permits server side, JavaScript code, to connect to Relational DataBase Management Systems (RDBMS). RDBMS such as Oracle, MS SQL Server, MySQL, mSQL and a host of other widely used relational databases, which generally use ANSI SQL as their natural language. **Live Wire** database drivers also support a number of non-relational databases.

Client side JavaScript

Client-side JavaScript is traditionally embedded into a standard HTML program. JavaScript is embedded between the **<SCRIPT> . . . </SCRIPT>** HTML tags. These tags are embedded within the **<HEAD> . . . </HEAD>** or **<BODY> . . . </BODY>** tags of the HTML program.

JavaScript is embedded into an HTML program because JavaScript uses the *filename*.html and the HTTP protocol to transport itself from the web server to the client's browser where the JavaScript executes and processes client information.

Only a browser that is JavaScript enabled will be able to interpret JavaScript code. Netscape Communicator does this best as JavaScript is the natural language of Netscape Communicator.

Microsoft's Internet Explorer also interprets JavaScript. However, Internet Explorer latest releases support an old version of JavaScript. Hence, the total functionality of the latest release of JavaScript as is available in Netscape Communicator is not available in Internet Explorer.

Note

The default scripting language of Internet Explorer is VB Script. Netscape Communieator does not support VB Script.

Capturing User Input

Web site interactivity starts from being able to capture user input. The **<FORM> . . . </FORM>** HTML tags can be used to create a user **Request-form**. Between these HTML tags the HTML **<INPUT> </INPUT>** tags can be used to instantiate HTML objects in the HTML form to facilitate the capture of user data.

The HTML objects used in HTML form creation are **Text, TextArea, Radio Buttons, Push Buttons, Check Boxes** and so on. These will be passed as **values** to the TYPE attribute of the **<INPUT> . . . </INPUT>** tags as will be seen in later chapters. Once the HTML form has been coded in *filename*.html JavaScript can be embedded in the same HTML file to make it interactive and facilitate client side data validation and/or processing.

The concept being that standard HTML form objects are used to *capture* user input while client side, JavaScript (*embedded in the HTML file*) is used to *validate* and/or *process* such user input. The application of *validation rules* to the data captured by HTML form objects and *error handling* is conveniently done using JavaScript which executes in the client's browser.

Once user input passes the validation tests applied, and/or has been processed appropriately by client side, JavaScript, the form data captured will have to be passed backward to the web server from where the HTML file originated.

At the web server an appropriate program (*written in JavaScript, VB Script, CGI-PERL, Java and so on*) will accept this user input. Based on user input information will be assembled on the web server, converted to an HTML file (*page*) and returned to the user. Such an HTML page is created on **Demand**.

An HTML file (*page*) created on demand, based on user input, is a dynamically created HTML page. This then is a clear indication of a dynamic and interactive web site.

HTML itself is static. HTML allows a very minimum interaction with users by providing hyperlinks. Truly interactive pages as described above, cannot be created using standard HTML **Tags** alone. Embedding JavaScript in an HTML program does this

For instance, a Web Site can be created and hosted on a web server to take orders for products. Typically, an HTML form is used to capture '*User orders for products*'. At the same time, form data (*the user order's*) validation must be done. For example:

- Orders should be accepted only for products which are available
- Any quantity ordered should not exceed the quantity currently available
- The order date should be set to the current date
- The quantity required cannot be left blank
- The place of delivery (*address*) cannot be left blank

HTML alone cannot do any of this. At the maximum, HTML can provide an elegant interface for capturing **Order** information. HTML (as mentioned earlier) does not provide any techniques for validating user entries. This makes it necessary to introduce client side JavaScript in the HTML (*page*) program, which extends the functionality of the web page by introducing client side data processing.

JAVASCRIPT

JavaScript is an object-oriented language that allows creation of interactive Web Pages. JavaScript allows user entries, which are loaded into an HTML form to be processed as required. This empowers a web site to return site information according to a user's requests.

JavaScript offers several advantages to a Web developer such as a short development cycle, ease of learning, small size scripts and so on. The strengths of JavaScript can be easily and quickly used to extend the functionality of HTML pages, already on a web site.

The Advantages Of JavaScript

An Interpreted Language: JavaScript is an interpreted language, which requires no compilation steps. This provides an easy development process. The syntax is completely interpreted by the browser just as it interprets HTML tags.

Embedded Within HTML: JavaScript does not require any special or separate editor for programs to be written, edited or compiled. It can be written in any text editor like Notepad, along with appropriate HTML tags, and saved as *filename.html*. HTML files with embedded JavaScript commands can then be read and interpreted by any browser that is JavaScript enabled.

Minimal Syntax - Easy to Learn: By learning just a few commands and simple rules of syntax, complete applications can be built using JavaScript.

Quick Development: Because JavaScript does not require time-consuming compilations, scripts can be developed in a short period of time. This is enhanced by the fact that many GUI interface features, such as alerts, prompts, confirm boxes, and other GUI elements, are handled by client side JavaScript, the browser and HTML code.

Designed for Simple, Small Programs: It is well suited to implement simple, small programs (*for example, a unit conversion calculator between miles and kilometers, or pounds and kilograms*). Such programs can be easily written and executed at an acceptable speed using JavaScript. In addition, they can be easily integrated into a web page.

Performance: JavaScript can be written such that the HTML files are fairly compact and quite small. This minimizes storage requirements on the web server and download time for the client.

Additionally, because JavaScript programs are usually included in the same file as the HTML code for a web page, they require fewer separate network accesses.

Procedural Capabilities: Every programming language needs to support facilities such as Condition checking, Looping and Branching. JavaScript provides syntax, which can be used to add such procedural capabilities to web page (*filename*.html) coding.

Designed for Programming User Events: JavaScript supports Object/Event based programming. JavaScript recognizes when a form **Button** is pressed. This event can have suitable JavaScript code attached, which will execute when the **Button Pressed** event occurs.

JavaScript can be used to implement context sensitive help. Whenever an HTML form's **Mouse** cursor **Moves Over** a button or a link on the page a helpful and informative message can be displayed in the status bar at the bottom of the browser window.

Easy Debugging and Testing: Being an interpreted language, scripts in JavaScript are tested line by line, and the errors are also listed as they are encountered, i.e. an appropriate error message along with the line number is listed for every error that is encountered. It is thus easy to locate errors, make changes, and test it again without the overhead and delay of compiling.

Platform Independence / Architecture Neutral: JavaScript is a programming language that is completely independent of the hardware on which it works. It is a language that is understood by any JavaScript enabled browser. Thus, JavaScript applications work on any machine that has an appropriate JavaScript enabled browser installed. This machine can be anywhere on the network.

Since each browser is for a specific platform, JavaScript interpretation will be with respect to the specific platform. The browser will add whatever platform specific information is required to the JavaScript while it interprets the code. Thus, JavaScript is truly platform independent. A JavaScript program developed on a Unix machine will work perfectly well on a Windows machine.

The fact that a platform specific browser, maintained at the client end, does the interpretation of JavaScript, relieves the developer of the responsibility of maintaining multiple source code files for multiple platforms.

WRITING JAVASCRIPT INTO HTML

JavaScript syntax is embedded into an HTML file. A browser reads HTML files and interprets HTML tags. Since all JavaScripts need to be included as an integral part of an HTML document when required, the browser needs to be informed that specific sections of HTML code is JavaScript. The browser will then use its built-in JavaScript engine to interpret this code.

The browser is given this information using the HTML tags **<SCRIPT> . . . </SCRIPT>**. The <SCRIPT> tag marks the beginning of a snippet of scripting code. The paired </SCRIPT> marks the end of the snippet of scripting code.

Like most other HTML tags, the <SCRIPT> tag takes in an optional attribute, as listed below:

Attributes	Description
Language	Indicates the scripting language used for writing the snippet of scripting code. If left undefined Netscape Communicator will assume JavaScript. If left undefined Internet Explorer will assume VB Script

Table 8.1

Syntax:

```
<SCRIPT LANGUAGE= "JavaScript">
   // Javascript code snippet written here
</SCRIPT>
```

BASIC PROGRAMMING TECHNIQUES

JavaScript offers the very same programming capabilities found in most programming languages. Creating variables, constants, programming constructs, user defined functions and so on.

All these programming techniques can be used in JavaScript embedded in any HTML program. These are the techniques that make JavaScript an exciting programming language that extends the functionality of HTML and makes web pages interactive.

The **<HEAD> . . . </HEAD>** tags make an ideal place to create JavaScript variables and constants and so on. This is because the head of an HTML document is always processed before the body. Placing JavaScript memory variables, constants and user defined JavaScript functions in this section of an HTML document will cause them to be defined (*by the JavaScript interpreter*) before being used. This is important because any attempt to use a variable (*or any other JavaScript object*) before it is defined, results in an error.

Variables are used to store values that can be used in other parts of a program. Variables always have a name associated with them via which they can be referenced later. When naming variables, it is good programming practice to structure a descriptive name.

Variable names can begin with an uppercase letter (A through Z), lowercase letter (a through z), and underscore character (_) or dollar sign character ($). The remaining characters can consist of letters, the underscore character, the dollar sign character or digits 0 to 9. Variable names are **case sensitive.**

Caution

The dollar sign ($) character is reserved for machine generated code and should not be used in scripts. In particular it should not be used in scripts that will be run by earlier browsers that are not fully ECMAScript-compatible.

If two words are used to name a variable then it is a programming convention to start the first letter of the first word in lower case and the first letter of the second word in uppercase for example, *variableName*.

JavaScript does not allow the data type of the variable to be declared when a variable is created. The same variable may be used to hold different types of data at different times when the JavaScript code snippet runs.

Data Types And Literal

JavaScript supports four primitive types of values and supports complex types such as arrays and objects. Primitive types are types that can be assigned a single literal value such as number, string or boolean value. Literals are fixed values, which *literally* provide a value in a program.

The primitive data types that JavaScript supports are:

Number

Consists of integer and floating point numbers and the special **NaN** (*not a number*) value.

Integer literals can be represented in JavaScript in decimal, hexadecimal, and octal form.

Note

Hexadecimal and octal integers are converted to decimal before they are displayed.

Floating-point literals are used to represent numbers that require the use of a decimal point, or very large or very small numbers that must be written using exponential notation. A floating-point number must consist of either a number containing a decimal point or an integer followed by an exponent.

For example, 33, 12.10, -35.8, 2E3, 0x5F

Boolean

Consists of the logical value *true* and *false*.

JavaScript supports a pure Boolean type that consists of the two values *true* and *false*. Logical operators can be used in Boolean expressions.

JavaScript automatically converts the Boolean values *true* and *false* into 1 and 0 when they are used in numerical expressions.

Note

Values 1 and 0 are not considered Boolean values in JavaScript.

String

Consists of string values that are enclosed in single or double quotes.

JavaScript provides built-in support for strings. A string is a sequence of zero or more characters that are enclosed by double (“) or single (‘) quotes. If a string begins with a double quote it must end with a double quote. If a string begins with a single quote it must end with a single quote.

For example, "Rahul", '24, Sanjay Nagar, Bangalore'

Note

If a string has to include quote character in the string the quote character must be preceded by the backslash (\) escape character.

Null

Consists of a single value, null, which identifies a null, empty or nonexistent reference.

The null value is common to all JavaScript types. It is used to set a variable to an initial value that is different from other valid values. Use of the null value prevents the sort of errors that result from using un-initialized variables. The null value is automatically converted to default values of other types when used in an expression.

Type Casting

In JavaScript variables are loosely cast. The type of a JavaScript variable is **implicitly defined** based on the **literal** values that are assigned to it from time to time.

For instance, combining the string literal "Total amount is " with the integer literal 1000 results in a string with the value "Total amount is 1000". By contrast, adding together the numeric literal 10.5 and the string "20" results in the floating point integer literal 30.5. This process is known as **type casting**.

Creating Variables

In order to make working with data types convenient, variables are created. In JavaScript variables can be created that can hold any type of data.

In order to use a variable, it is good programming style to declare it. Declaring a variable tells JavaScript that a variable of a given name exists so that the JavaScript interpreter can understand references to that variable name throughout the rest of the script.

Although it is possible to declare variables by simply using them, declaring variables helps to ensure that programs are well organized and helps keep track of the scope of the variables. Variables can be declared using **var** command.

Syntax:

var <variable name> = *value*;

Examples:

```
var first_name;
var last_name = "Shah";
var phone = 6128879;
```

The equal sign (=) used in assigning a value to a variable is known as an *assignment* operator.

Note

Like properties and method names in JavaScript, variable names are case sensitive.

Incorporating Variables In A Script

Example 1:

The following illustrates incorporating variables in a script.

```
<HTML>
   <HEAD>
      <SCRIPT Language = "JavaScript">
         var name = prompt("Enter your name", "Name");
      </SCRIPT>
   </HEAD>
```



```
   <BODY>
      <SCRIPT Language = "JavaScript">
         document.write("<H2> Hello " + name + "</H2>");
      </SCRIPT>
   </BODY>
</HTML>
```

The JavaScript **prompt()** method picks up a string (i.e *User Name*) from the user which is then assigned to the variable *name*. The JavaScript code **document.write()** embedded in the **<BODY> . . . </BODY>** tags writes the contents of the variable *name* to the client browser.

Since the **<HEAD> . . . </HEAD>** section of the HTML program is interpreted first a 'User Name' is picked up first before anything is displayed in the client browser.

The JavaScript Array

Arrays are JavaScript objects that are capable of storing a sequence of values. These values are stored in indexed locations within the array. The length of an array is the number of elements that an array contains. The individual elements of an array are accessed by using the name of the array followed by the *index value* of the array element enclosed in square brackets.

Note

The array element index starts with **0**. Hence the last array element index number is **one less** that the length of the array.

An array must be declared before it is used. An array can be declared using any one of the following techniques.

```
arrayName = new Array(Array length)
arrayName = new Array( )
```

In the first example the array size is explicitly specified. Hence this array will hold a pre-determined set of values. The second example creates an array of the size 0.

Note

JavaScript automatically extends the length of any array when new array elements are initialized.

Example:

```
cust_Orders = new Array( )
cust_Orders[50] = "Lion Pencils"
cust_Orders[100] = "Steadler eraser"
```

When JavaScript encounters the reference to order [50], in the above example, it will extend the size of the array cust_Orders to 51 and initialize order[50]. When JavaScript encounters the reference to order [100], in the above example, it will extend the size of the array cust_Orders to 101 and initializes order[100].

Even if an array is initially created of a fixed length it may still be extended by referencing elements that are outside the current size of the array. This is done in the same manner as with zero-length arrays.

Dense Arrays

A dense array is an array that has been created with each of its elements being assigned a specific value. Dense arrays are used exactly in the same manner as other arrays. They are declared and initialized at the same time.

Listing the values of the array elements in the array declaration creates dense arrays. For example a dense array can be created as:

arrayName = new Array(value0, value1, ..., valuen)

In this array, since the element count starts from 0 to *n*, the array length is *n*+1.

Since array is a JavaScript object, arrays have several methods associated with them via which the array and its element content can be manipulated.

Join() returns all elements of the array joined together as a single string. This takes one argument; a string to be used as a separator between each element of the array in the final string. If the argument is omitted, join() uses a comma-space as the separator.

Reverse() reverses the order of the elements in the array.

Example 2:

An array is used with hard coded values. Displaying the values of the array elements in the browser makes use of an array's **join()** method to print the array elements in a single line.

```
<HTML>
   <HEAD><TITLE>Viewing the array elements of a JavaScript Array</TITLE></HEAD>
   <BODY>
      <SCRIPT LANGUAGE = "JavaScript">
         <!-- Begin hiding JavaScript
            friends = new Array(5);
            friends[0] = "Ananth";
            friends[1] = "Cedric";
            friends[2] = "Ketan";
            friends[3] = "Rohan";
            friends[4] = "Leela";
            document.write(friends[0] + "<BR/>");
            document.write(friends[1] + "<BR/>");
            document.write(friends[2] + "<BR/>");
            document.write(friends[3] + "<BR/>");
            document.write(friends[4] + "<BR/>");
            join_crit = friends.join();
            document.write(join_crit);
         // End hiding JavaScript -->
      </SCRIPT>
   </BODY>
</HTML>
```

The Elements Of An Array

JavaScript does not place any restrictions on the values assigned to the elements of an array. These values could be of different types or could refer to other arrays or objects.

Example:

multiTypeArray = new Array("Val1", "Val2", 1, 2, true, false, null, new array(3, 4))

The array named multiTypeArray has a length of eight and its elements are:

multiTypeArray[0] = "Val1"
multiTypeArray[1] = "Val2"
multiTypeArray[2] = 1

multiTypeArray[3] = 2
multiTypeArray[4] = true
multiTypeArray[5] = false
multiTypeArray[6] = null
multiTypeArray[7] = *a new dense array consisting of the values of 3,4*

The last element of the array, multiTypeArray, contains a dense array as its value. The two elements of this array can be accessed using a *second set of subscripts*:

num1 = multiTypeArray[7][0];
num2 = multiTypeArray[7][1];

The JavaScript Array and its length Property

JavaScript arrays are implemented as objects. *Objects* are named collections of data that have **properties** and whose values may be accessed via **methods**. A *property* returns a value that identifies some aspect of the state of an object. *Methods* are used to read or modify the data contained in an object's property.

The length of an array is a property of an array. Access to any JavaScript object's property is done by using **objectname.propertyname**.

For example, to find out the length of the multiTypeArray:

myvar = multiTypeArray.length;

The length of the multiTypeArray will be assigned to the variable *myvar*. By accessing the contents of myvar the length of (*no. of elements in multiTypeArray*) the multiTypeArray can be determined.

OPERATORS AND EXPRESSIONS IN JAVASCRIPT

An *operator* is used to transform one or more values into a single resultant value. The values to which the operator is applied is referred to as *operands*. A combination of an operator and its operands is referred to as an *expression*.

Expressions are evaluated to determine the value of the expression. This is the value that results when an operator is applied to its operands.

Note

For some operators, such as multiplication (*) the *order* of the operands do not matter. For example, A*Y = Y*A is *true* for all integers and floating point numbers.

Other operators such as string concatenation (+), the order of the operands matter. For example, "ab" + "cd" is not the same as "cd" + "ab".

Arithmetic Operators

Arithmetic operators are the most familiar operators because they are used every day to solve common math calculations. The arithmetic operators that JavaScript supports are;

Operator	Description
+	Addition
*	Multiplication
%	Modulus
--	Return the value then Decrement

Operator	Description
-	Subtraction or Unary negation
/	Division
++	Return the value then Increment

Table 8.2

Note

The % (modulus) operator calculates the remainder by dividing two integers. For example, 17 % 3 = 2 because 17/3 = 5 with a remainder of 2.

An operator requiring a single operand is referred to as a **Unary** operator and one that requires two operands is a *binary* operator.

The above standard arithmetic operators are binary operators. In addition to these binary operators, there are unary arithmetic operators. They are (++) and (--).

Both these increment and decrement operators can be used in two different ways. Before the operand or after the operand. For example, ++x increments x by **one** and returns the result, while x++ returns x and then increments the value of x by one. Similarly, --x decreases the value of x by **one** before returning a result, while x-- returns the value of x before decreasing its value by one.

Example:

```
X = 3;
Y = ++X;
Z = X++;
```

In these lines of code, X is first assigned the value of 3, which is then increased to 4 and assigned to Y. The new value of 4 is assigned to Z, and then the value of X is increased to 5. Finally, X is 5, Y is 4 and Z is 4.

Logical Operators

Logical operators are used to perform Boolean operators on Boolean operands **AND, OR, NOT.** The logical operators supported by JavaScript are:

Operator	&&	||	!
Description	Logical AND	Logical OR	Logical NOT

Table 8.3

Comparison Operators

Comparison operators are used to compare two values. The comparison operators supported by JavaScript are:

Operator	Description
==	Equal
!=	Not Equal
<	Less than
>	Greater than

Operator	Description
===	Strictly Equal
!==	Strictly Not Equal
<=	Less than or equal to
>=	Greater than or equal to

Table 8.4

Note

The equal (==) and not equal (!=) operators perform type conversions before testing for equality. For example, "5" == 5 evaluate to **true.**

The strictly equal (===) and the strictly not equal (!===) do not perform type conversions before testing for equality. For example, "5" === 5 will return the value **false.**

String Operators

String operators are those operators that are used to perform operations on strings. Currently JavaScript supports only the **string concatenation** (+) operator.

This operator is used to join two strings. For example, "pq" + "ab" produces "pqab".

Assignment Operators

The assignment operator is used to update the value of a variable. Some assignment operators are combined with other operators to perform a computation on the value contained in a variable and then update the variable with the new value. Some of the assignment operators supported by JavaScript are:

Operator	Description
=	Sets the variable on the left of the = operator to the value of the expression on its right
+=	Increments the variable on the left of the += operator by the value of the expression on its right. When used with strings, the value to the right of the += operator is appended to the value of the variable on the left of the += operator
-=	Decrements the variable on the left of the - = operator by the value of the expression on the right
*=	Multiplies the variable on the left of the * = operator by the value of the expression on its right
/=	Divides the variable on the left of the /= operator by the value of the expression on its right
%=	Takes the modulus of the variable on the left of the % = operator using the value of the expression on its right

Table 8.5

The Conditional Expression Ternary Operator

JavaScript supports the conditional expression operator. They are ? and : The conditional expression operator is a ternary operator since it takes three operands, a condition to be evaluated and two alternative values to be returned based on the truth or falsity of the condition.

Syntax:

```
condition ? value1 : value2
```

The condition expressed must return a value true or false. If the condition is true, value1 is the result of the expression, otherwise value2 is the result of the expression.

Special Operators

JavaScript supports a number of special operators that do not fit into the operator categories covered above.

The delete Operator

The delete operator is used to delete a property of an object or an element at an array index.

Example:

To delete the **sixth** element of myArray.

```
delete myArray[5]
```

The new Operator

The new operator is used to create an instance of an object type.

Example:
To create a new JavaScript object of the type array and assign this array to a context area in memory called myArray.

```
myArray = new Array( )
```

The void Operator

The void operator does not return a value. It is typically used in JavaScript to return a URL with no value.

JAVASCRIPT PROGRAMMING CONSTRUCTS

Most programming languages support a common (**core**) set of constructs. Languages only differ in the syntax used for structuring these constructs.

Languages may also differ in the degree to which they support programming features such as **Object Oriented Programming**, abstract data definition, inference rules and list processing.

JavaScript provides a complete range of basic programming constructs. While it is not an object oriented programming environment JavaScript is an **object-based** language.

The constructs provided by JavaScript are as follows:

Construct / Statement	Purpose	Example
Assignment	Assigns the value of an expression to a variable.	x = y + z
data declaration	Declares a variable and optionally assigns a value to it.	var myVar = 10

if	Program execution depends upon the value of returned by the condition. If the value returned is True the program executes else the program does not execute.	if(x > y) {
		z = X;
		}

while	Repeatedly executes a set of statements until a condition becomes false	while(x != 7) {
		x %= n - n
		}

switch	Selects from a number of alternatives	switch(val) {
		case 1:
		// First alternative
		break;
		case 2:
		// Second alternative
		break;
		default
		// Default action
		}

for	Repeatedly executes a set of statements until a condition becomes false	for(i = 0; i < 7; ++i) {
		document.write(x[i]);
		}

do while	Repeatedly executes a set of statements while a condition is true	do {
		// Statements
		} while(i > 0)

label	Associates a label with a statement	LabelName: Statement

Table 8.6

Construct / Statement	Purpose	Example
break	Immediately terminates a do while or for loop construct	if(x > y) break
continue	Immediately terminates the current iteration of a do, while or for construct	if(x > y) continue
function call	Invokes a function	x = abs(y)
return	Returns a value from a function call	return x*y

with	Identifies the default object	with(Math) {
		d = PI * 2 * r;
		}

delete	Deletes an object property or an array element	delete a[5]
Method invocation	Invokes a method of an object	document.write("Hello")

Table 8.6 (continued)

CONDITIONAL CHECKING

The if - then - else Statement

The conditional construct in JavaScript offers a simple way to make a decision in a JavaScript program. The conditional construct in JavaScript will either return a **True** or a **False** depending upon how the **condition** evaluated.

Using the **if-else** construct the flow of the JavaScript program can be altered i.e. the **if** condition determines which section of the program code will be executed based on whether the condition evaluates to **TRUE** or **FALSE.**

Syntax:

```
if(condition) {
// JavaScript code
}
```

If the **condition** evaluation returns **True** the JavaScript code is *executed* if the evaluation returns **False** this section of JavaScript code will be *skipped.*

Example:

```
var day = "Saturday"
if (day == "Saturday") {
    document.writeln("It's the weekend");
    alert("It's the weekend");
}
```

Immediate if (Conditional expression)

A conditional expression can evaluate to either **True** or **False** based on the **evaluation** of the condition. The structure of a conditional expression is:

Syntax:

```
(condition) ? value1: value2
```

where, **condition** is an expression that can be evaluated to a boolean value. Based on the result, the whole expression evaluates to either **value1** (true condition) or **value2** (false condition).

Example:

```
var day = "Saturday"
    (day == "Saturday") ? "Weekend!" : "Not Saturday"
```

This expression will evaluate to:

"Weekend!"	where **day** holds Saturday!	(Condition TRUE).
"Not Saturday!"	if day holds any other string	(Condition FALSE)

SUPER CONTROLLED - ENDLESS LOOPS

Looping refers to the ability of a block of code to repeat itself. This repetition can be for a predefined number of times or it can go until certain conditions are met. For instance, a block of code needs to be executed till the value of a variable becomes 20 (Conditional Looping), or a block of code needs to be repeated 7 times.

For this purpose, JavaScript offers 2 types of loop structures:

- **for Loops** - These loops iterate a specific number of times as specified.
- **while Loops** - These are Conditional Loops, which continue until a condition is met.

For Loop

The **for** loop is the most basic type of loop and resembles a for loop in most other programming languages, including ANSI 'C'.

Syntax:

```
for(expression1; condition; expression2) {
// JavaScript commands
}
```

where, **expression1** sets up a counter variable and assigns the initial value. The declaration of the counter variable can also be done here itself, **condition** specifies the final value for the loop to fire (i.e. the loop fires till **condition** evaluates to true), **expression2** specifies how the initial value in **expression1** is **incremented**.

Example:
The following block prints numbers from 10 to 1 on the VDU screen (Reverse Order).

```
for (var num = 10; num >=1; num--)
{
    document.writeln(num);
}
```

While Loop

The **while** loop provides a similar functionality. The basic structure of a **while** loop is:

Syntax:

```
while (condition) {
// JavaScript commands
}
```

Where, the *condition* is a valid JavaScript expression that evaluates to a Boolean value. The JavaScript commands execute as long as the condition is **true.**

Example:
The following block prints numbers from 1 to 10 on the screen.

```
var num = 1;
while (num <= 10)
{
    document.writeln(num);
    num++;
}
```

FUNCTIONS IN JAVASCRIPT

Functions are blocks of JavaScript code that perform a specific task and often return a value. A JavaScript function may take zero or more parameters. Parameters are a standard technique via which control data can be passed to a function. Control data passed to a function, offers a means of controlling what a function returns.

Built-in Functions

JavaScript provides several built-in functions that can be used to perform explicit type conversions. Some of them are **eval()**, **parseInt()** and **parseFloat()**.

The **eval()** function can be used to convert a string expression to a numeric value.

Example:
The following results in the value 105 being assigned to the variable grand_Total.

```
var grand_Total = eval("10 * 10 + 5");
```

Note

Even if the string value passed as a parameter to **eval()** does represent a numeric value the use of **eval()** results in an error being generated.

The **parseInt()** function is used to convert a string value to an integer. The **parseInt()** function returns the first integer contained in a string or 0 if the string does not begin with an integer.

Example:
The following results in the integer 123 being assigned to the variable string2Num.

```
var string2Num = parseInt("123xyz");
```

The following results in "NaN" (Not a Number) being assigned to the variable string2Num.

```
var string2Num = parseInt("xyz");
```

The **parseFloat()** function returns the first floating point number contained in a string or 0 if the string does not begin with a valid floating point number.

Example:
The following results in the float 1.2 being assigned to the variable string2Num.

```
var string2Num = parseFloat("1.2xyz");
```

The following results in "NaN" (Not a Number) being assigned to the variable string2Num.

```
string2Num = parseInt("xyz");
```

USER DEFINED FUNCTIONS

Functions offer the ability to group together JavaScript program code that performs a specific task into a single unit that can be used repeatedly whenever required in a JavaScript program.

A user defined function first needs to be declared and coded. Once this is done the function can be invoked by calling it using the name given to the function when it was declared.

Functions can accept information in the form of arguments and can return results.

Appropriate syntax needs to be followed for declaring functions, invoking them, passing them values and accepting their return values.

Declaring Functions

Functions are declared and created using the **function** keyword. A function can comprise of the following:

- A name for the function
- A list of parameters (arguments) that will accept values passed to the function when called
- A block of JavaScript code that defines what the function does

Syntax:

```
function function_name(parameter1, parameter2, . . . ) {
// Block of JavaScript code
}
```

A **function_name** is case sensitive, can include underscores (_), and has to start with a letter. The list of **parameters** passed to the function appears in parentheses and commas separate members of the list.

Note

Defining a function does not execute the JavaScript code that makes up the function.

Place Of Declaration

Functions can be declared anywhere within an HTML file., Preferably, functions are created within the **<HEAD>...</HEAD>** tags of the **HTML** file. This ensures that all functions will be *parsed* before they are invoked or called. If the function is called before it is declared and parsed, it will lead to an error condition, as the function has not been evaluated and the 'Browser' does not know that it exists.

Note

The term **parsed** refers to the process by which the JavaScript interpreter evaluates each line of script code and converts it into a pseudo-compiled bytecode, before attempting to execute it. At this time, syntax errors and other programming mistakes that would prevent the script from running are trapped and reported.

Passing Parameters

Values can be passed to function parameters when a 'parameterized' function is called. Values are passed to the function by listing them in the parentheses following the function name. Multiple values can be passed, separated by commas provided that the function has been coded to accept multiple parameters.

Both JavaScript built-in functions and user-defined functions can accept parameters, process them and return values. During declaration, a function needs to be informed about the number of values that will be passed.

Example:

Function Declaration:

```
function printName(user) {
    document.write("<HR/>Your Name is <B><I>");
    document.write(user);
    document.write("</B></I><HR/>");
}
```

where, **printName** is a function, which has a parameter called **user**. The parameter user can be passed a value at the time of invoking the function. Within the function, reference to **user** will then refer to the **value** passed to the function.

Function Call:

A static value passed:

```
printName("Bob");
```

(will cause the string "Bob" to be assigned to the parameter **user**.)

A Variable Passed:

```
var user = "John";
printName(user);
```

(will cause the value "John" to be assigned to the parameter **user**.)

Note

- Both variables and literals can be passed as arguments when calling a function.
- If a variable is passed to the function, changing the value of the parameter within the function does not change the value of the variable passed to the function.
- Parameters exist only for the life of the function.

Example 3:

Once an HTML page completes loading it greets a user with a message 'Welcome'. When a user leaves this page, the 'Good-bye' alert dialog box is displayed.

```
<HTML>
    <HEAD>
        <TITLE>Creating and Using User Defined Functions</TITLE>
        <SCRIPT Language="JavaScript">
            var name = "";
            function hello()
            {
                name = prompt('Enter Your Name:', 'Name');
                alert('Greetings ' + name + ', Welcome to my page!');
            }
            function goodbye()
            {
                alert('Goodbye ' + name + ', Sorry to see you go!');
            }
        </SCRIPT>
    </HEAD>
    <BODY onLoad="hello();" onUnload="goodbye();">
        <IMG SRC="images/Pinkwhit.gif" />
    </BODY>
</HTML>
```

Variable Scope

The parameters of a function are local to the function. They come into existence when the function is called and cease to exist when the function ends. For instance, in the example **printName()**, **user** exists only within the function **printName()** - it cannot be referred to or manipulated outside the function.

Also, any variable declared using **var** variable-name *within* the function would have a scope limited to the function.

If a variable is declared outside the body of the function, it is available to all statements within the JavaScript.

If a *local variable* is declared within a function has the same name as an existing *global variable*, then within the function code, that variable name will refer to the *local variable* and **not** the global variable. It is as though the global variable does not exist with respect to the JavaScript code within the function.

Return Values

As with some JavaScript built-in functions, user defined functions can return values. Such values can be returned using the **return** statement. The return statement can be used to return any valid expression that evaluates to a single value.

Example:

```
function cube(number) {
    var cube = number * number * number;
    return cube;
}
```

where, **cube** is a function, which accepts a parameter *number*, calculates its cube, assigns this calculation to a variable **cube** and returns the value of cube.

This function can also be written as follows:

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

Recursive Functions

Recursion refers to a situation, wherein *functions call themselves*. In other words, there is a call to a specific function from within the same function. Such functions are known as Recursive Functions.

Example:

The following JavaScript function is an example of a Recursive Function that calculates the factorial of a number. (A factorial is a mathematical function. For example, factorial 5 is equal to 5*4*3*2*1)

```
function factorial(number) {
    if(number>1) {
        return number * factorial(number-1);
    }
    else {
        return number;
    }
}
```

This function receives a number as an argument and relies on the fact that the factorial of a number is the number multiplied by the factorial of one less than the number.

The function applies the formula and returns the number multiplied by the factorial of one less than the number.

Note

Recursive functions are powerful, but can lead to infinite recursions. Infinite recursions occur when the function is designed in such a way as to call itself without being able to stop.

It is important to note that the function **factorial()** prevents infinite recursions because the **if-else** construct ensures that eventually the function will stop calling itself once the number passed to the function is equal to one. Additionally, if the function is *initially* called with a value *less than two*, no recursion will take place at all.

PLACING TEXT IN A BROWSER

Using JavaScript a string can be written to the browser from within an HTML file. The **document** object in JavaScript has a **method** for placing text in a browser. This method is called **write()**. Methods are called by combining the object name with the method name:

Object-name.Method-Name

The **write()** method accepts a string value passed to it within its parentheses. The string value can then be written to the browser. The **write()** method accepts this string and places it in the current browser window. For example:

```
document.write("Test");
```

The string "Test" will be placed in the browser.

Example 4:

The following example illustrates how JavaScript code places text in the browser window using document.write().

```
<HTML>
    <HEAD><TITLE>Outputting Text </TITLE></HEAD>
    <BODY><CENTER><BR/><BR/>
        <IMG HEIGHT="100" SRC="sony-logo.gif" WIDTH="100"/>Silicon Chip Technologies.<BR/>
        <SCRIPT Language = "Javascript">
            document.write("<BR/><BR/>");
            document.write('<IMG HEIGHT="100" SRC="sony-logo.gif" WIDTH="100"/>');
            document.write("<B>Silicon Chip Technologies.</B> <BR/>");
        </SCRIPT>
    </CENTER></BODY>
</HTML>
```

Output For Example 4:
(Refer to diagram 8.1)

DIALOG BOXES

JavaScript provides the ability to pickup user input or display small amounts of text to the user by using dialog boxes. These dialog boxes appear as separate windows and their content depends on the information provided by the user. This content is independent of the text in the HTML page containing the JavaScript script and does not affect the content of the page in any way.

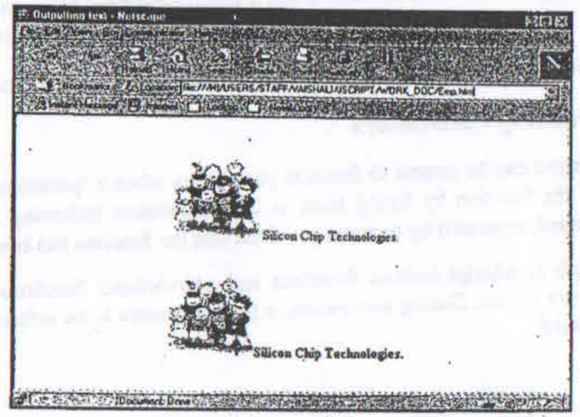

Diagram 8.1

There are three types of dialog boxes provided by JavaScript:

The Alert Dialog Box

The simplest way to direct small amounts of textual output to a browser's window is to use an alert dialog box. The JavaScript **alert()** method takes a string as an argument and displays an alert dialog box in the browser window when invoked by appropriate JavaScript.

The alert dialog box displays the string passed to the **alert()** method, as well as an `OK` button. The JavaScript and the HTML program, in which this code snippet is held, will not continue processing until the `OK` button is clicked.

The alert dialog box can be used to display a cautionary message or display some information. For instance:

- A message is displayed to the user when incorrect information is keyed in a form
- An invalid result is the output of a calculation
- A warning that a service is not available on a given date/time

Syntax:

alert("<Message>");

Example:

alert("Click OK to continue");

Example 5:

The following example shows an alert dialog box, welcoming a user. As soon as the `OK` button is clicked, an image is displayed in the browser. This illustrates that all background processing **stops** until an **alert** has been responded to.

```
<HTML>
        <HEAD><TITLE>Example</TITLE></HEAD>
        <BODY><SCRIPT Language="Javascript">
                alert("Welcome To My Web Site!");
                document.write('<IMG SRC="Images/welcome.gif"/>');
        </SCRIPT></BODY>
</HTML>
```

Output For Example 5:
(Refer to diagram 8.2)

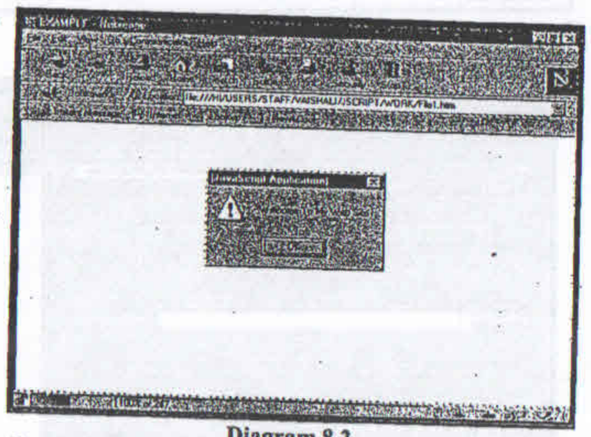

Diagram 8.2

The Prompt Dialog Box

As seen, the alert dialog box simply displays information in a browser and does not allow any interaction. The addition of the `OK` button provides some very minimal control over form events *i.e.* program execution halts completely until some user action takes place (*clicking on the* `OK` *button*).

An alert dialog box, cannot be used to customize any web page output based on user input, which is what user interaction requires.

JavaScript provides a prompt dialog box for this. The **prompt()** method instantiates the prompt dialog box which displays a specified message. In addition, the prompt dialog box also provides a single data entry field, which accepts user input. Thus, a prompt dialog box:

- Displays a predefined Message
- Displays a textbox and accepts user input
  - Can pass what the user keyed into the textbox back to the JavaScript
- Displays the `OK` and the `Cancel` buttons

The prompt dialog box also causes program execution to halt until user action takes place. This could be the `OK` button being clicked, or the `Cancel` button being clicked, which causes the following action to take place.

- Clicking on the `OK` button causes the text typed inside the textbox to be passed to the program environment (i.e. *JavaScript*)
- Clicking on the `Cancel` button causes a NULL value to be passed to the environment.

When the **prompt()** method is used to instantiate and use a dialog box the method requires two blocks of information:

- A message to be displayed as a prompt to the user
- Any message to be displayed in the textbox (*this is optional*)

Syntax:

prompt("<Message>", "<Default value>");

Example:

prompt("Enter your favorite color:", "Blue");

The value that the user keys into the textbox on the prompt dialog box is accepted and can be stored in a variable.

Example 6:

The following example shows a welcoming image on the screen. Asks the user for a name. Then displays the name keyed into the prompt dialog box along with a Greeting message.

```
<HTML>
    <HEAD><TITLE>Example </TITLE></HEAD>
    <BODY>
        <SCRIPT LANGUAGE="JavaScript">
            document.write('<IMG Src="Images/welcome.gif"/>');
            document.write("<H1>Greetings</H1>");
            document.write(prompt("Enter Your Name: ", "Name"));
            document.write(" Welcome to My HomePage!</H1>");
        </SCRIPT>
    </BODY>
</HTML>
```

Output For Example 6: (Refer to diagram 8.3)

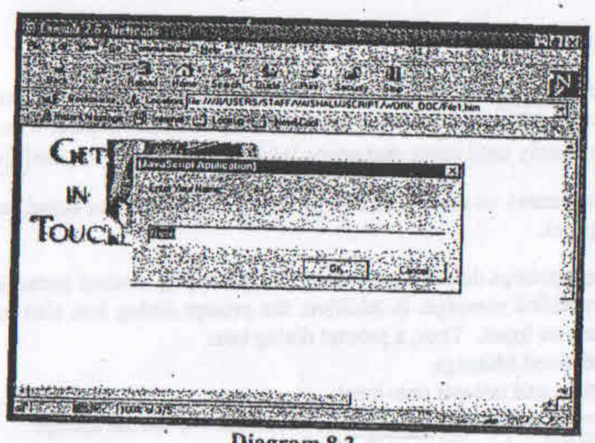

Diagram 8.3

The Confirm Dialog Box

JavaScript provides a third type of a dialog box, called the confirm dialog box. As the name suggests, this dialog box serves as a technique for confirming user action. The confirm dialog box displays the following information:

- A pre-defined message
- `Ok` and `Cancel` button

The confirm dialog box, causes program execution to halt until user action takes place. User action can be either the `OK` button being clicked, or the `Cancel` button being clicked, which causes the following action to take place.

- Clicking on the `OK` button causes TRUE to be passed to the program which called the confirm dialog box.
- Clicking on the `Cancel` button causes FALSE to be passed to the program which called the confirm dialog box.

Display of a confirm dialog box thus requires only one block of information:
A pre-defined message to be displayed

Syntax:

confirm("<Message>");

Example:

confirm("Are you sure you want to exit out of the system");

Example 7:

The following JavaScript example asks a question and accepts an answer. The user is given three chances. The second and third chance to provide an answer can be accepted or rejected, if accepted the program prompts for an answer again.


```
<HTML>
   <HEAD>
      <TITLE> Confirm Method </TITLE>
      <SCRIPT LANGUAGE="JavaScript">
            var question = "What is 10+10 ?";
            var answer = 20;
            var correct = '<IMG SRC="images/man2.gif"/>';

            var incorrect ='<IMG Src="images/man1.gif"/>';
            var Response = prompt(question,"0");
            for(count = 0; count < 3; count++) {
                if(Response != answer) {
                    confirm("Wrong, Press OK For Another Chance");
                    Response = prompt(question,"0");
                    if(count == 1)
                    {
                        alert("Better Luck Next Time");
                    }
                }

                else {
                    alert("Great!! You Are Right");
                    count = 3;
                }
            }
            var output = (Response == answer) ? correct :  incorrect;
            document.write("<BR/>");
            document.write(output);
      </SCRIPT>
   </HEAD>
   <BODY></BODY>
</HTML>
```

Output:

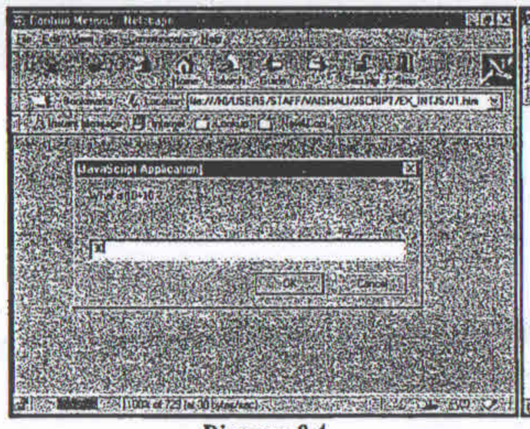

Diagram 8.4

Diagram 8.5

SELF REVIEW QUESTIONS

FILL IN THE BLANKS

1. Capturing user requests is traditionally done via a _____.
2. JavaScript is a scripting language created by _____.
3. JavaScript is embedded between the _____ HTML tags.
4. A user request form can be created with the _____ HTML tags.
5. _____ are used to store values that can be used in other parts of a program.
6. The _____ is reserved for machine generated code and should not be used in scripts.
7. In JavaScript, variable is _____ _____ based on the _____ value that is assigned to it.
8. A _____ is a sequence of zero or more characters that are enclosed by double (") or single (') quotes.
9. _____ are block of JavaScript code that perform a specific task and return a value.
10. _____ are JavaScript objects that are capable of storing a sequence of values.
11. A _____ array is an array that has been created with each of its elements being assigned a specific value.
12. _____ are named collections of data that have properties and may be accessed via methods.
13. The _____ operator calculates the remainder by dividing two integers.

TRUE OR FALSE

14. JavaScript is not an interpreted language.
15. A JavaScript program developed on a Unix machine will work perfectly well on a Windows machine.
16. The <BODY></BODY>HTML tags make an ideal place to create JavaScript variables and constants and so on.
17. JavaScript does not allow the data type of the variable to be declared when a variable is created.
18. If the quote character is to be included in the string, the quote character must be preceded by the frontslash (/) escape character.
19. eval(), parseInt() and parseFloat() are three functions provided by JavaScript to perform explicit type conversions.
20. *Methods* are used to read or modify the data contained in an object.
21. The combination of an operator and its operands is referred to as an *expression*.
22. If a variable is declared outside the body of the function, it is available throughout a script inside all functions and elsewhere in the program.

HANDS ON EXERCISES

1. Write a JavaScript code block using arrays and generate the current date in words, this should include the day, the month and the year. The output should be as follows, **Saturday, January 01, 2000.**

2. Write a JavaScript code block, which checks the contents entered in a form's Text element. If the text entered is in the lower case, convert to upper case. Make use of function *toUppercase()*.
3. Write a JavaScript code block, which validates a username and password against hard coded values.

   If either the name or password field is not entered display an error message showing:
   "You forgot one of the required fields. Please try again."

   In case, the fields entered do not match the hard coded values, display an error message showing:
   "Please enter a valid username and password"

   If the fields entered match, display the following message: "Welcome (username)"

9. THE JAVASCRIPT DOCUMENT OBJECT MODEL

INTRODUCTION

An HTML page is rendered (*painted*) in a browser. The browser assembles all the elements (*objects*) contained in the HTML page, downloaded from the web server, in its memory. Once done the browser then renders (*paints*) these objects in the browser window. Once the HTML page is rendered (*painted*) in the browser window, the browser can no longer recognize individual HTML elements (*objects*).

To create an interactive web page it is imperative that the browser continues to recognize individual HTML objects even after they are rendered in the browser window. This allows the browser to access the properties of these objects using the built-in methods of the object. Once the properties of an object are accessible then the functionality of the object can be controlled at will.

JavaScript enabled browsers are capable of recognizing individual objects in an HTML page, after the page has been rendered in the browser, because the JavaScript enabled browser recognizes and uses the Document Object Model (i.e *the DOM*).

Using the Document Object Model (DOM) JavaScript enabled browsers identify the collection of web page objects (*web page elements*) that have to be dealt with while rendering an HTML based, web page in the browser window.

The HTML objects (i.e. *the collection of web page elements*), which belong to the DOM, have a *descending* relationship with each other.

The topmost object in the DOM is the **Navigator** (i.e. *the browser*) itself. The next level in the DOM is the browser's **Window**. The next level in the DOM is the **Document** displayed in the browser's window.

Should the document displayed in the browser's window have an HTML **Form** coded in it, then the next level in the DOM is the **Form** itself.

The DOM hierarchy continues downward to encompass individual elements on a **FORM**, such as Text boxes, Labels, Radio buttons, Check boxes, Push buttons and so on, which belong to the form.

JavaScript's object hierarchy is mapped to the DOM, which in turn is mapped to the web page elements in the browser window. Hence, when a web page is rendered in a JavaScript enabled browser window, JavaScript is capable of uniquely identifying each element in the web page, because major elements of a web page are bound to the DOM.

The DOM that JavaScript recognizes is described in diagram 9.1.

JavaScript's DOM is referred to as an ***instance hierarchy***.

The Navigator – i.e. Netscape Navigator, Internet Explorer, Opera, Mosaic and so on..

```
Window
|-> Document
      |-> Anchor
      |-> Link
      |-> Form
            |-> textbox
            |-> textarea
            |-> radiabutton
            |-> checkbox
            |-> select
            |-> button
```

Diagram 9.1: JavaScript's DOM.

Instance

No HTML object is registered in the DOM by a JavaScript enabled browser unless they are assembled in memory prior being rendered in the browser window.

What this means is, if a document does not have any **Anchors** described in it the **Anchors** object will exist but it will be empty. If the document does not have any **Links** described in it the **Links** object will exist but it will be empty.

Hierarchy

All objects on a web page are not created equal. Each exists in a set relationship with other objects on the web page. From diagram 9.1 the navigator occupies the topmost slot in the DOM followed by the Window object and so on.

Below the Window is the **Document** object. Below the document object three other objects exist. They are the **Anchor**, **Link** and **Form** objects. Individual form elements are found under the **Form** object.

In addition to the DOM, other objects currently recognized by a JavaScript enabled browser are *Plug-ins*, *Applets* and *Images*. Hence using a JavaScript enabled browser and JavaScript most of the major web page objects are accessible.

However, every single element of a web page rendered in the browser window, is **not** part of the DOM.

For example, HTML tags such as **<HEAD> . . . </HEAD>** or **<BODY> . . . </BODY>** are not part of the DOM. Presentation styles, headings, body text, H1 to H6 and so on are **not** part of the DOM hence **not** recognized by JavaScript.

Diagram 9.1 shows web page objects that are part of the DOM.

JavaScript however, recognizes presentation styles, headings, body text, H1 to H6 and so on, when JavaScript assisted Style Sheets [JSSS] are in a web page. JSSS is usually between the **<HEAD> . . . </HEAD>** HTML tags in a web page.

THE JAVASCRIPT ASSISTED STYLE SHEETS DOM [JSSS DOM]

JSSS use JavaScript syntax to control a document's presentation style. When a JSSS is embedded in an HTML page within the **<HEAD> . . . </HEAD>** tags, then the JavaScript DOM picks up a whole new set of objects, which **add** to the standard DOM objects already recognized by JavaScript. The additional objects brought into the DOM by JSSS are shown in diagram 9.2.

By extending the DOM recognized by JavaScript by embedding JSSS in a web page, developers of web pages can access every element of a web page whether this element appears on the page when it is rendered in a client browser or not.

By accessing appropriate properties of the **Navigator** object, (i.e. *the Browser*), the topmost object in the DOM, JavaScript can recognize the browser type (i.e. *Netscape Navigator, Internet Explorer, Opera, Mosaic and so on*) and subsequently dispatch all HTML pages to the browser from the web server, with a **style** based on this knowledge. This is where the power of JavaScript really becomes visible in providing finely tuned web page content to a client's browser.

Objects added to the DOM by the use of JSSS are:

```
|-> Document
      |-> Tags
      |     |-> P
      |     |-> DIV
      |     |-> SPAN
      |     |-> H1 through H6
      |->classes
      |     |-> Tag Names
      |-> IDS
```

Diagram 9.2: JSSS's DOM.

Since JavaScript understands the DOM and can extend the DOM with the use of JSSS in a web page JavaScript understands **Objects**.

All **objects** have:

- **Properties** that determine the functionality of the object
- **Methods** that allow access to these properties
- **Events** that allow JavaScript code snippets to be connected to the object by being mapped to appropriate JavaScript event handlers.

Hence when a pre-determined event occurs the code snippet will execute. This is the traditional Object, Event driven, Code execution model of any object based programming environment.

Using appropriate JavaScript code snippets, which reference the properties of an object via its built-in methods, developers of web pages can actually control the functionality of any HTML object in the DOM (*or extended DOM*) while the HTML program executes (i.e. *at run time*).

JavaScript can access the methods of all objects belonging to the DOM and JSSS DOM. Hence using JavaScript, truly interactive web pages can be created.

Note

JavaScript is an **object-based** programming language it is **not** fully object oriented. It does not fully support basic object oriented programming capabilities such as classification, inheritance, encapsulation and information hiding.

JavaScript features are geared towards providing developers the capability to quickly generate scripts that will execute in the context of a web page within a JavaScript enabled browser **or** on a web server that understands JavaScript.

Although JavaScript does not provide all of the features of a full object oriented programming language, it does provide a suite of object-based features that are especially tailored to Browser or Server side scripting.

These features include the recognition of a number of predefined browser and server objects. JavaScript has the ability to control the behavior of these objects through their properties and methods.

In the following chapters the focus will be only on **browser** side JavaScripting. The browser in which JavaScript code snippets will always run correctly will be Netscape Communicator as JavaScript is a Netscape product. JavaScript is the natural language of Netscape Communicator.

UNDERSTANDING OBJECTS IN HTML

HTML can be used to create a Graphical User Interface (GUI) in a web page. HTML is capable of accessing and using a number of objects that actually belong to the operating system (i.e. Windows). One such object is a *textbox*. A 'textbox' is used in an HTML form to accept user input.

The text box is an object, which belongs to the DOM. JavaScript recognizes a text box. JavaScript facilitates access to all the *methods* of a *textbox*. One of the methods of the textbox permits access to the contents of the text box. Hence, JavaScript can process the contents of any textbox in an HTML form.

Properties Of HTML Objects

Just like real world objects, HTML objects have a number of **properties** that determine the behavior of that object. An object's properties can be referenced as:

ObjectName.PropertyName

For example, a *textbox* can have properties like *name, size* and so on.

Methods Of HTML Objects

As seen, properties determine the state of an object. While building interactive web pages an object's properties need to be set dynamically, *i.e. when the object is being used.* Thus all object based programming environments **must** have facilities to *set* or *get* the value of object properties. This allows program code to control the state of the object at run time.

Methods of an object are used to *set* or *get* a value of an object's property. Thus determining how an object behaves. An object's methods can be referenced as:

ObjectName.MethodName

Using JavaScript it is possible to use an object's built-in methods and manipulate the object's properties at run time. This gives a great deal of creative control to web page developers when web pages have to be created on demand.

Once JavaScript code accepts client side input and / or reads a client's browser parameters, on demand web pages can be structured and sent to the browser from a web server.

BROWSER OBJECTS

When any JavaScript enabled browser loads a web page, the browser automatically creates a number of JavaScript objects that map to the DOM (*or JSSS DOM*). It is the DOM, which provides JavaScript access to the HTML objects that are contained in the web page.

The JavaScript code snippets imbedded as part of the web page itself (i.e. embedded within the <SCRIPT> </SCRIPT> tags of *filename*.html) makes use of these objects to interact with the HTML objects in the web page.

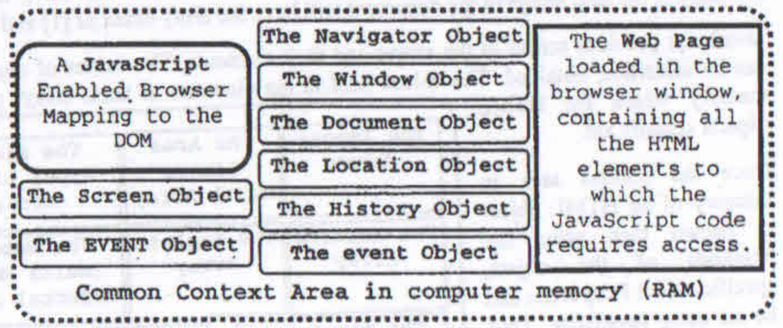

Diagram 9.3

The JavaScript objects created by [Netscape Communicator] are listed below:

Object Name	Its Use
navigator	To access information about the browser that is executing the current script.
window	To access a browser *window* or a *frame* within the window. The window object is assumed to exist and does not require the **window** prefix when referring to its properties and methods.
document	To access the document currently loaded into a window. The document object refers to an HTML document that provides content, that is, one that has HEAD and BODY tags.
location	To represent a URL. It can be used to create a URL object, access parts of a URL, or modify an existing URL.
history	To maintain a history of the URL's accessed within a window.
event	To access information about the occurrence of an event.
EVENT	The EVENT (*capitalized*) object provides constants that are used to identify events.
screen	To access information about the size and color depth of a client computer's screen in which the current browser is running.

Table 9.1

How A JavaScript Enabled Browser Handles The Document Object

Any document (i.e. *the HTML page*) can contain various HTML objects such as:

- Images
- Image Maps
- Hyperlinks
- Frames
- Anchors
- Applets
- Multimedia objects such a audio files, streaming video files
- A Form with various form elements

The browser creates one array in memory per HTML object in the document, thus registering each of these HTML objects.

If these HTML objects are actually contained in the HTML page then these arrays will hold indexed elements, which will point to the context area in memory where the HTML object are. Otherwise the array will exist, but will be empty (i.e. *have no elements in it*).

If there are multiple, similar HTML objects in the document (i.e *multiple images*) each array will have multiple (*indexed*) elements.

The array index value mapped to an HTML object will correspond to where the HTML object was described in the document. The first image in the document will have the array index as [0] (i.e. Images[0]), the next image in the document will have the array index of [1] and so on.

JavaScript provides access to the arrays and their elements. The values of *any/all* elements of each array can be identified, obtained. The values held in the elements of these arrays point to the context area in memory where the HTML objects actually are.

Once the context area in memory of an HTML object is known then using the **Methods** of the object, specific object **Properties** can be set using JavaScript. Thus the functionality of an HTML object can be controlled while the HTML page is running in the browser.

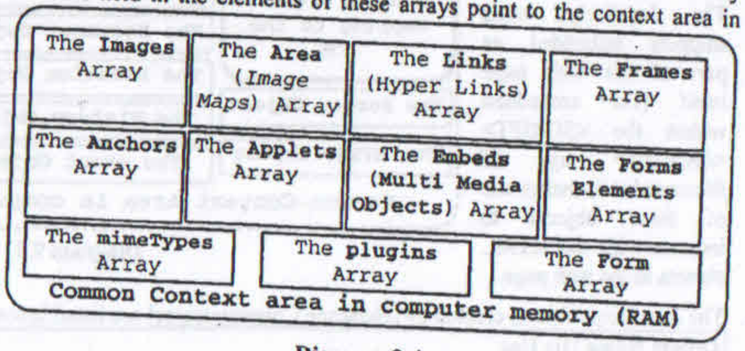

Diagram 9.4

The JavaScript arrays [created by Netscape Communicator] are listed below:

Image/Images Array	To access an image that is embedded in an HTML document. The **images array** is used to access all image objects in a document.
Link / Links Array	To access a text or image-based source anchor of a hypertext link. The **links array** is used to access all link objects within a document.
Area	To access an **area** within a client-side image map.
Frame / Frames Array	To access an HTML frame. The **frames array** is used to access all frames within a window.

Table 9.2

The JavaScript arrays [created by Netscape Communicator] are listed below: (Continued)

Anchor/Anchors array	To access the target of a hyperlink. The **anchors array** is used to access all anchor objects within a document.
Applet/Applets array	To Access a Java applet. The **applets array** being used to access all the applets in a document.
Embed / Embeds array	To access an embedded object. The **embeds array** provides access to all the embedded objects in a document.
MimeType / MimeTypes array	To access information about a particular MIME type supported by a browser. The **mimeTypes array** is an array of all the mimeType objects supported by a browser.
Plugin / Plugins array	To access information about a particular browser plug-in. The **plugins array** is an array of all plug-ins supported by a browser.
Form / Forms array	To access an HTML form. The **forms array** is used to access all forms within a document.

Table 9.2 (Continued)

The JavaScript **form elements** array [created by Netscape Communicator]:

elements	Access to all the form elements in the form.
text	To access a text field of a form.
textarea	To access a text area of a form.
radio	To access a set of radio buttons on the form or to access an individual radio button within the set.
checkbox	To access a checkbox on a form.
button	To access a form button that is not a reset or submit button.
submit	To access a submit button on a form.
reset	To access a reset button on a form.
select	To access a select list of a form.
option	The option object is used to access the elements of a select list.
password	To access a password field on a form.
hidden	To access a hidden object on a form.
fileupload	To access a file upload element of a form.

Table 9.3

THE WEB PAGE HTML OBJECT HIERARCHY

This is an *instance hierarchy*. This means if a web page does not have a specific HTML object defined in it the array associated with that specific HTML object will exist, but will have no elements.

Access To Elements Of A Web Page

Conceptually once a web page is rendered (*painted*) in a browser window it is completely static.

For any program code to be able to interact with the web page, each element of the web page would have to be held in memory, with a unique name. The unique name translates to a context area in memory where the web page element resides.

Hence, while a web page is being assembled in memory (RAM) prior being rendered (*made visible*) in the browser's window, a JavaScript enabled browser creates several arrays as described earlier. These arrays hold references to individual web page objects in their (*indexed*) elements.

Referencing an appropriate element in its associated array provides access to each element of a web page. Hence, using JavaScript web page elements can be updated or processed. Once processed, the web page can be re-rendered in the browser. When re-rendered in the browser changes made to the web page elements will be visible.

If updation or processing of any web page element is done, *based on client input*, the web page is then an *interactive* web page.

How A Web Page Element Is Manipulated

HTML tags are used to create (*instantiate*) objects in a web page. For example, <INPUT Type="TEXT"> will instantiates a text box on the web page when encountered in HTML code.

Each HTML object instantiated in a web page has **properties** and **methods** that allow access to the object's properties. Each HTML object has an **event** or several **events** bound to the object when the object was created. Thus an HTML object can recognize a specific event when it occurs.

Once the HTML object recognizes that an event has occurred this knowledge has to be passed to JavaScript so that JavaScript also recognizes that the 'event' occurred. To facilitate this JavaScript provides a number of named JavaScript 'Event Handlers'. The names of these event handlers are descriptively bound to an HTML object's event name.

Example:

A **Change** event is recognized by a text box when its contents change. JavaScript provides an event handler called **onChange** that is internally bound to the **Change** event of the text box. The **Change** event of the text box talks to the **onChange** event handler of JavaScript. The **onChange** event handler of JavaScript can then execute an appropriate JavaScript code snippet. For example:

```
<INPUT Type="TEXT" onChange="<myFunction>">
```

Here the JavaScript function **myFunction** is bound to the JavaScript event handler onChange. The assignment operator (=) does this binding.

The JavaScript, onChange event handler, is bound to the HTML object **TEXT** by being passed as one of its attributes.

Hence, as soon as the **Change** event of the text box occurs the JavaScript event handler onChange is invoked.

Since the JavaScript function myFunction is bound to the JavaScript event handler onChange, as soon as the onChange event handler is invoked the JavaScript code in myFunction executes.

HANDLING (WEB PAGE) EVENTS USING JAVASCRIPT

A web page event could be associated with the action of the mouse cursor on the web page. Such as:

- A mouse-click on an object in a web page
- The movement of the mouse cursor across a web page
- The mouse cursor hovering at a specific place on a web page and so on

These will be events recognized by the Window object of the DOM.

Other web page events could be the opening or closing of a Window, the loading of an image in a web page and so on.

JavaScript's approach to working with web page elements is a multi step process:

- Identify a web page object
- Choose an appropriate event associated with the object
- Have a standard method of connecting an object's event and JavaScript code snippets. JavaScript **event handlers** mapped to an object's **events** do this.

JavaScript has several named event handlers that are mapped to an HTML object's events. To work with JavaScript it is necessary to understand how to use JavaScript 'Event Handlers' correctly.

JavaScript, event handlers, can be divided into two types **Interactive** and **Non Interactive**.

An interactive event handler depends on user interaction with an HTML page. For example, the JavaScript **onMouseOver** event handler is an interactive event handler. This requires the user to move the mouse cursor over a web page.

A JavaScript, non-interactive, event handler, does not need user interaction to be invoked. For example the JavaScript 'onLoad' event handler is a non-interactive event handler as it automatically executes whenever a form is loaded into a web page.

Table 9.4 details JavaScript event handlers that descriptively bound to HTML object events. As long as the HTML object has an associated event, JavaScript provides an associated, named, event handler.

Named JavaScript Event Handlers

JavaScript Event Handler	Will Be Called When
onAbort	The loading of an image is aborted as a result of user action
onBlur	A document, window, frame set, or form element loses current input focus.
onChange	A text field, text area, file-uploaded field or selection is modified and loses the current input focus.
onClick	A link, client-side image map area or document is clicked
onDblClick	A link, client-side image map area or document is double clicked
onDragDrop	A dragged object is dropped in a window or frame
onError	An error occurs during loading of an image, window or frame
onFocus	A document, window, frame set, or form element receives the current input focus
onKeyDown	The user presses a Key
onKeyPress	The user presses and releases a Key
onKeyUp	The user releases a Key
onLoad	An image, document or frame set is loaded
onMouseDown	The user presses a mouse button
onMouseMove	The user moves the mouse
onMouseOut	The mouse is moved out of a link or an area of a client side image map
onMouseOver	The mouse is moved over a link or an area of a client side image map
onMouseUp	The user releases a mouse button
onReset	The user resets a form by clicking on the form's reset button
onResize	The user resizes a window or frame
onSelect	Text is selected in a text field or a text area
onSubmit	The user presses a form's submit button
onUnload	The user exits a document or frame set

Table 9.4

The naming convention followed by Java Script makes it easy to identify a JavaScript event handler. The JavaScript, event handler's name, simply has the string '*on*' added to the HTML object's event name (e.g. *onMouseOver*).

Example:

<A HRef=http://www.sctindia.com onMouseOver="*<JavaScript code snippet itself>* or *<A call to a* JavaScript *function>*">Text associated with the Link</A>

Tip

Here, the **onMouseOver** JavaScript event handler is bound to the hyperlink <A> . . . </A> HTML tag. When the mouse cursor moves over the hyperlink its **MouseOver** event occurs. When **MouseOver** event occurs a call is made to the JavaScript event handler **onMouseOver**. When the JavaScript **onMouseOver** event handler is invoked a JavaScript code snippet can execute directly or a JavaScript function can be called.

Note

If a JavaScript code snippet is directly passed as a value to an HTML attribute and is enclosed in double quotes ("), then double quotes (") cannot be used within the enclosed JavaScript code. Replacing these double quotes with single quotes (') where necessary.

To use the JavaScript DOM and manipulate its objects properties will help in raising this basic comfort level to a confidence level required for coding in JavaScript.

To achieve this several examples which use the HTML <FORM> . . . </FORM> tags along with their attributes mapped to JavaScript code will be used.

SELF REVIEW QUESTIONS

FILL IN THE BLANKS

1. Using the _____ _____ _____ JavaScript enabled browsers identify the collection of web page objects that have to be dealt with while rendering an HTML based web page in the browser window.
2. _____ _____ _____ _____ use JavaScript to control a document's presentation style
3. _____ of an object are used to set or get a value of an object's property.
4. JavaScript event handlers can be divided into two types _____ and _____.
5. _____ object is used to access information about the browser that is executing the current script

TRUE OR FALSE

6. If the document does not have any links described in it, the Link object does not exist.
7. JavaScript 'onLoad' event handler is an interactive event handler as it automatically executes whenever a form is loaded into a web page.
8. The browser creates one array in memory per HTML object in the document.

HANDS ON EXERCISES

1. Create a Web page using two image files, which switch between one another as the mouse pointer moves over the images. Use the onMouseOver and onMouseOut event handlers. The output is as shown in the diagrams 9.5 and 9.6.

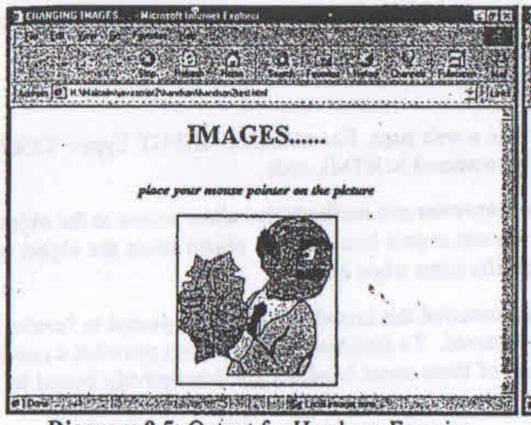

Diagram 9.5: Output for Hands on Exercise.

Diagram 9.6: Output for Hands on Exercise.

10. FORMS USED BY A WEB SITE

An HTML form provides data gathering functionality to a web page. This is very useful if the web site is used to advertise and sell products. HTML forms provide a full range of GUI controls. Additionally, HTML forms can automatically submit data collected in its controls to a web server.

The data submitted can be processed at the web server by CGI programs, server side JavaScripts, Java Servlets and so on.

JavaScript allows the validation of data entered into a form at the client side. JavaScript can be used to ensure that only valid data is returned to a web server for further processing.

This chapter focuses on:

- The JavaScript FORM object created when the HTML <FORM> </FORM> tags are encountered in an HTML program.
- Describes how JavaScript can be associated with an HTML form's GUI controls.

After working through chapter examples the following will be understood:

- The properties and methods of the JavaScript FORM object and it's associated GUI controls.
- How JavaScript is used to handle form related events and perform (*client side - in the browser window*) local processing of form data.

THE FORM OBJECT

When creating an interactive web site for the Internet it is necessary to capture user input and process this input. Based on the result of this processing, appropriate information from a web site can be dispatched to be viewed. Both the capturing of user input and the rendering of appropriate web pages takes place in the client side, browser's window.

Traditionally, user input is captured in a **Form**. HTML provides the **<FORM> ... </FORM>** tags with which an HTML form can be created to capture user input.

As soon as the **<FORM> ... </FORM>** tags are encountered in an HTML program by a JavaScript enabled browser, the browser creates a '*forms array*' in memory. This array tracks the number of form objects described in the HTML program.

Each form object in the HTML page will be described between its own <FORM> ... </FORM> HTML tags. Should there be multiple forms (i.e. multiple occurrences of the <FORM> ... </FORM> tags) described in the HTML page then the forms array will have multiple (*indexed*) elements, each holding a reference to an HTML form object.

The first form object described in the HTML file being held as array index[0], the second form object described in the HTML file being held in the array index[1] and so on. By referencing a specific index number of the forms array a specific form object can be accessed. The JavaScript **forms array** also holds information about each object used within the **<FORM> ... </FORM>** tags.

Common HTML objects used between the **<FORM> ... </FORM> tags are Text, TextArea, Radio Buttons, Buttons, Check Boxes** and so on. An HTML form is used extensively in creating interactive web pages. Using the associated arrays created by a JavaScript enabled browser and JavaScript code, all form elements (*objects contained in the form*) are accessible. Once accessible their properties can be manipulated so as to control the functionality of the form at run time.

There are two forms in the HTML file; **Form1** and **Form2**. Refer to diagram 10.1: These will be held in the first two elements of the Forms array. **Form1** will be an address, which points to where the Form elements array is located.

Form1 in the Forms array, will be a reference to the context area where the elements of Form1 are located. Form1 has three form elements.

Form2 in the Forms array, is a reference to the context area where the elements of Form2 are located. Form2 has five form elements.

To understand this model, let us write a JavaScript procedure that will read the elements of the Form object's array, and return the number of actual form objects held.

Once the reference to the form's elements are known let us write a JavaScript procedure that will read the each Form's Element array and return the names of the form elements held in the array. The elements held in each of the arrays must be exactly the same as the elements described between the **<FORM>...</FORM>** tags in the HTML file running in the browser.

Forms Object's (Array)

Index	Value
0	Form1
1	Form2

Form1.Elements

Index	Value
0	Form1.Element1
1	Form1.Element2
2	Form1.Element3

Form2.Elements

Index	Value
0	Form2.Element1
1	Form2.Element2
2	Form2.Element3
3	Form2.Element4
4	Form2.Element5

Diagram 10.1: The Form Element's Array

Exercise 1:

Focus: Count the number of elements in a Form's, elements array. Check the number returned against the number of form elements described between the **<FORM>...</FORM>** tags in the HTML page that is running in the Browser. Recognize that the number of elements in the elements array match the number of elements described between the <FORM>...</FORM> tags in the HTML page exactly.

The diagram 10.2.1 shows an Alert box that displays a message indicating the number of form elements of implemented by the First Form in the HTML file. Refer to the HTML and JavaScript code described in Exercise 1.

Pop up an Alert that displays the individual form elements of the array and recognize that these are the same as are specified between the <FORM>...</FORM> tags in the HTML program

The diagram 10.2.2 shows an Alert box displaying a message that the Textbox named **Text1** of the First Form is at position **0** in the Elements array.

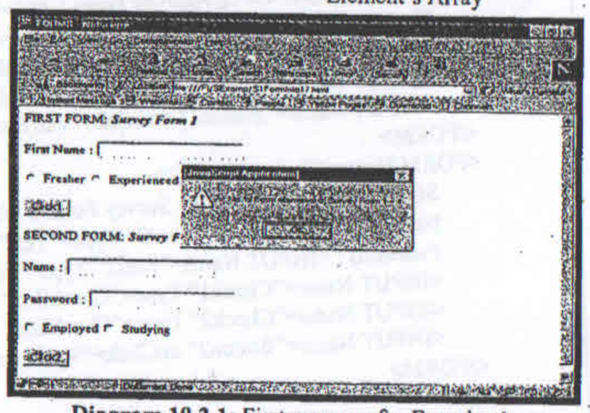

Diagram 10.2.1: First message for Exercise 1.

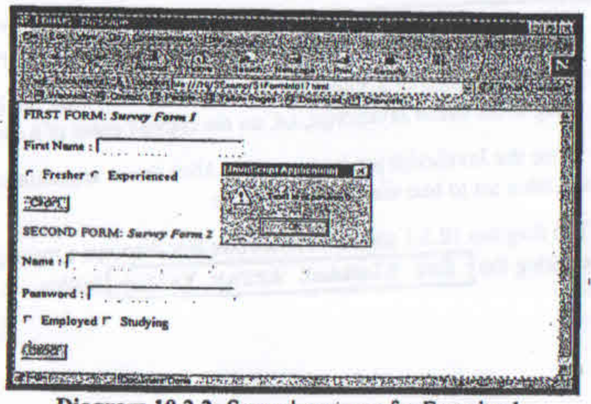

Diagram 10.2.2: Second message for Exercise 1.

The Code Listing For Exercise 1:

```
<HMTL>
   <HEAD><TITLE>FORMS</TITLE>
<!-- The code allows to access the Form objects Elements Array //-->
   <SCRIPT Language="JavaScript">
      function Ver(form1) {
         v = form1.elements.length;
         if (form1.elements[3].name == "Button1") {
            alert('First form name : ' + document.forms[0].name);
            alert('No. of Form elements of ' + document.forms[0].name + ' = ' + v);
         }
         else if (form1.elements[4].name == "Button2") {
            alert('Second form name : ' + document.forms[1].name);
            alert('No. of Form elements of ' + document.forms[1].name + ' = ' + v);
         }
         for(i=0; i < v; i++)
            alert(form1.elements[i].name + ' is at position ' + i);
      }
</SCRIPT></HEAD>
<BODY>
   <FORM Name="Survey Form 1">
      FIRST FORM: <I><B>Survey Form 1 </B></I><BR/><BR/>
      First Name : <INPUT Name="Text1" Type="Text" Value="" /><BR/><BR/>
      <INPUT Name="Radio1" Type="Radio" Value="" /> Fresher
      <INPUT Name="Radio2" Type="Radio" Value="" /> Experienced<BR/><BR/>
      <INPUT Name="Button1" onClick="Ver(form)" Type="Button" Value="Click1" />
   </FORM>
   <FORM Name="Survey Form 2">
      SECOND FORM: <I><B> Survey Form 2 </B></I><BR/><BR/>
      Name : <INPUT Name="Text2" Type="Text" Value="" /> <BR/><BR/>
      Password : <INPUT Name="Pass2" Type="Password" Value="" /> <BR/><BR/>
      <INPUT Name="Check1" Type="CheckBox" Value="" /> Employed
      <INPUT Name="Check2" Type="CheckBox" Value="" /> Studying <BR/><BR/>
      <INPUT Name="Button2" onClick="Ver(form)" Type="Button" Value="Click2" />
   </FORM>
</BODY>
</HTML>
```

Exercise 2: Illustrates the use of a form object's Elements array

Focus: The state of a Radio button and a Checkbox on the HTML form can be programmatically changed using event based JavaScript, *i.e. on the clicked event of a command button.*

When the JavaScript program runs an Alert draws attention to the fact that the Checkbox, **checked** property has been set to true via JavaScript code.

The diagram 10.3.1 shows an Alert box that displays a message indicating that the Checkbox is checked, on clicking the `Set Element Array Value` button.

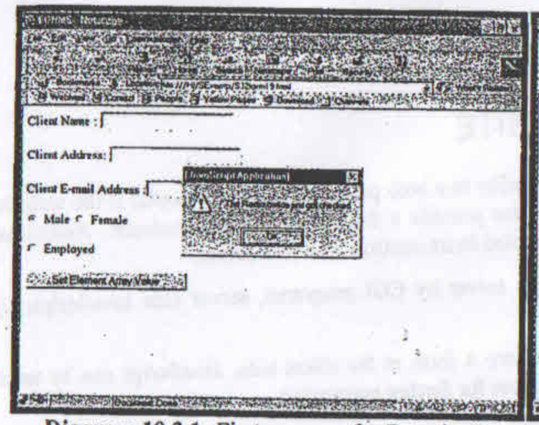

Diagram 10.3.1: First message for Exercise 2.

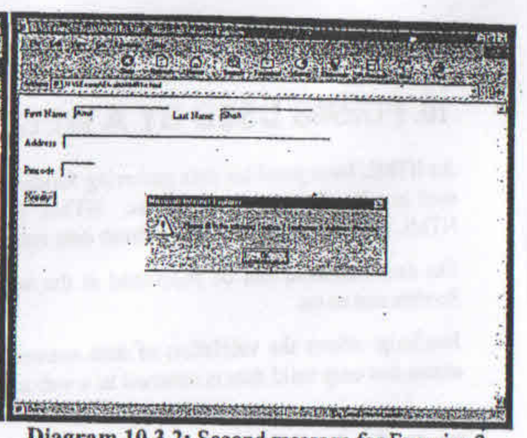

Diagram 10.3.2: Second message for Exercise 2.

On clicking the first Alert's `OK` button, another Alert pops up that displays a message indicating that the Radio button is checked, when the same 'Set Element Array Value' button was clicked.

Conceptually, without clicking on the HTML form objects their 'Checked' properties have been set. This illustrates how JavaScript code can access a form's element via the form elements array and manipulate an element's properties.

As soon as the elements properties change the Browser will display the object with its changed property in the HTML page. Refer to the HTML and JavaScript code described in Exercise 2.

The Code Listing For Exercise 2:

```
<HMTL>
   <HEAD><TITLE>FORMS</TITLE>
<!-- The code allows to access the Form objects Elements Array //-->
   <SCRIPT Language='JavaScript'>
      function Chk(f1) {
         f1.Check.checked=true;
         alert("The Checkbox just got checked");
         f1.Check.checked=false;
         f1.Radio[0].checked=true;
         f1.Radio[1].checked=false;
         alert("The Radio button just got checked");
      }
   </SCRIPT></HEAD>
   <BODY><FORM>
      Client Name : <INPUT Name="Text" Type="Text" Value="" /><BR/><BR/>
      Client Address : <INPUT Name="Text1" Type="Text" Value="" /> <BR/><BR/>
      Client E-mail Address : <INPUT Name="Text2" Type="Text" Value="" /><BR/><BR/>
      <INPUT Name="Radio" Type="radio" Value="" /> Male
      <INPUT Name="Radio" Type="radio" Value="" /> Female<BR/><BR/>
      <INPUT Name="Check" Type="CheckBox" Value="" /> Employed <BR/><BR/>
      <INPUT Name="Bt" onClick="Chk(this.form)" Type="Button"
             Value="Set Element Array Value"
   </FORM></BODY>
</HTML>
```

Exercise 3:

Focus: Create a HTML form that has a number of Textboxes. When the form runs in the Browser fill the Textboxes with data. Write JavaScript code that verifies that all Textboxes have been filled. If a Textbox has been left empty, popup an Alert indicating which Textbox has been left empty. When the Alert's OK button is clicked on, Set focus to that specific Textbox. If all the Textboxes are filled, display a Thank You alert.

The diagram 10.4.1 shows an Alert box that displays a message indicating that the **Address** and the **Pincode** Textboxes were not filled in.

The diagram 10.4.2 shows an Alert box that displays a message indicating that all the Textboxes have been filled.

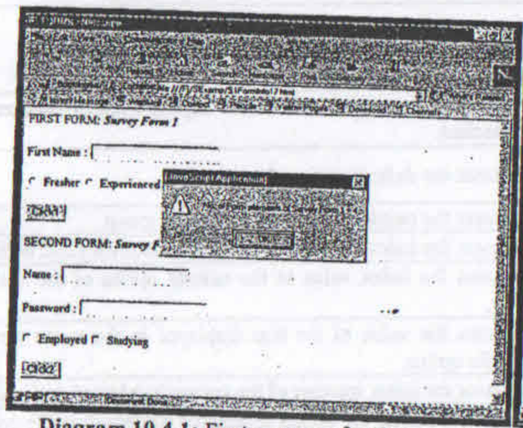

Diagram 10.4.1: First message for Exercise 1.

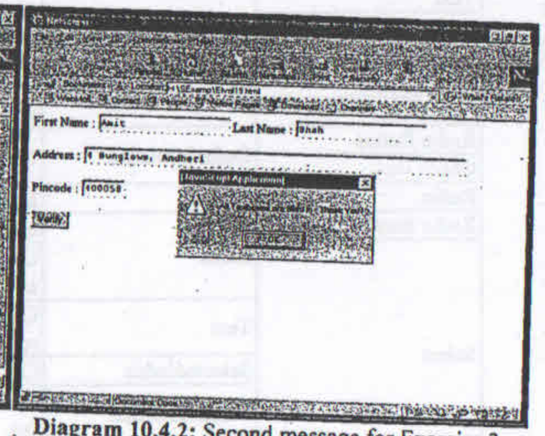

Diagram 10.4.2: Second message for Exercise 3.

This is a simple exercise, which illustrates how JavaScript code can be used to validate data that is entered in a Form.

Based on this concept, business rules can be implemented in any HTML form using JavaScript.

Thus data being entered into an HTML form can be validated on the client's machine before being dispatched to a web server for further processing. Refer to the HTML and JavaScript code described in Exercise 3.

The code listing for Exercise 3:

```
<HTML>
   <HEAD><SCRIPT LANGUAGE="JAVASCRIPT">
      function verifyData() {
         a=0;     r="";
         for (i=0; i<=4; i++) {
            if (document.forms[0].elements[i].value == "") {
               a=1;
               r = r + " " + document.forms[0].elements[i].name + ";" ;
            }
            else if ((i > 3)&&(a==0)) {
               alert("All Textboxes are filled in - Thank You !");
            }
```



```
         }
         for (i=0; i<=4; i++) {
            if (document.forms[0].elements[i].value == "") {
               alert("Please fill in the following Textbox / Textboxes :- " + r);
               document.forms[0].elements[i].focus( );
               break;
            }
         }
      }
   </SCRIPT></HEAD>
   <BODY><FORM>
      First Name : <INPUT Name="Firstname" Type="Text" Size=20 />
      Last Name : <INPUT Name="Lastname" Type="Text" Size=20 />
      <P>Address : <INPUT Name="Address" Type="Text" Size=60 /></P>
      <P>Pincode : <INPUT Name="Pincode" Type="Text" Size=6 /></P>
      <INPUT Name="act" onClick="verifyData( )" Type="button"Value="Verify" />
   </FORM></BODY>
   <SCRIPT Language="JavaScript">
      document.forms[0].Firstname.focus( );
   </SCRIPT>
</HTML>
```

The Form Object's Methods

HTML forms can be made up of a variety of HTML elements that accept user input. The <FORM> </FORM> HTML tags enclose the HTML elements that make up the form. Once a JavaScript enabled browser encounters these tags in an HTML file the JavaScript enabled browser creates a **form object** in memory, which is held as an element of the forms array. The form object has properties like **Name, Method** and **Action.**

Method

The **Method** property of a form is used to specify the method used to send data captured by various form elements back to the web server. The method used can be either **Get** or **Post.**

The **Get** method sends the data captured by form elements to the web server encoded into a URL, which points to a web server. The data captured in form elements is appended to the URL.

This technique is used when there is a small amount of data being sent back to the web server. The maximum amount of data that can be sent back to the web server using this method is 1024 bytes.

The **Post** method sends the data captured by form elements back to the web server as a separate bit-stream of data. When there is a large amount of data to be sent back to the web server, this is the method used.

If the method attribute is not specified within the <FORM> </FORM> tags, the default method used by the browser to send data back to the web server is the *Get* method, *i.e. as an encoded URL.*

Action

The **Action** attribute of the **<FORM>...</FORM>** tags points to the URL (*address*) of a program on the web server that will process the form data captured and being sent back. The server side program that processes this data can be written in any scripting language that the web server understands.

Exercise 3:

Focus: Create a HTML form that has a number of Textboxes. When the form runs in the Browser fill the Textboxes with data. Write JavaScript code that verifies that all Textboxes have been filled. If a Textbox has been left empty, popup an Alert indicating which Textbox has been left empty. When the Alert's OK button is clicked on, Set focus to that specific Textbox. If all the Textboxes are filled, display a Thank You alert.

The diagram 10.4.1 shows an Alert box that displays a message indicating that the **Address** and the **Pincode** Textboxes were not filled in.

The diagram 10.4.2 shows an Alert box that displays a message indicating that all the Textboxes have been filled.

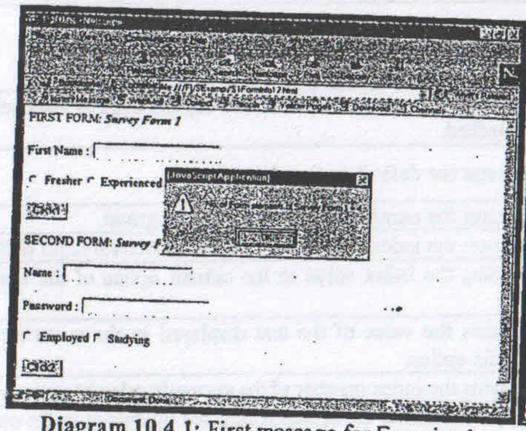

Diagram 10.4.1: First message for Exercise 1.

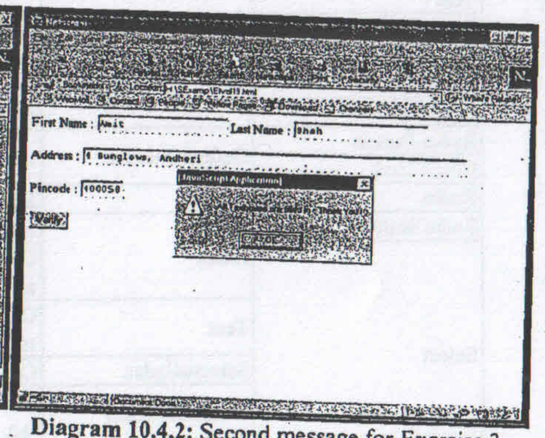

Diagram 10.4.2: Second message for Exercise 3.

This is a simple exercise, which illustrates how JavaScript code can be used to validate data that is entered in a Form.

Based on this concept, business rules can be implemented in any HTML form using JavaScript.

Thus data being entered into an HTML form can be validated on the client's machine before being dispatched to a web server for further processing. Refer to the HTML and JavaScript code described in Exercise 3.

The code listing for Exercise 3:

```
<HTML>
    <HEAD><SCRIPT LANGUAGE="JAVASCRIPT">
        function verifyData() {
            a=0;     r="";
            for (i=0; i<=4; i++) {
                if (document.forms[0].elements[i].value == "") {
                    a=1;
                    r = r + " " + document.forms[0].elements[i].name + ";" ;
                }
                else if ((i > 3)&&(a==0)) {
                    alert("All Textboxes are filled in - Thank You !");
                }
```



```
            }
            for (i=0; i<=4; i++) {
                if (document.forms[0].elements[i].value == "") {
                    alert("Please fill in the following Textbox / Textboxes :- " + r);
                    document.forms[0].elements[i].focus( );
                    break;
                }
            }
        }
    </SCRIPT></HEAD>
    <BODY><FORM>
        First Name : <INPUT Name="Firstname" Type="Text" Size=20 />
        Last Name : <INPUT Name="Lastname" Type="Text" Size=20 />
        <P>Address : <INPUT Name="Address" Type="Text" Size=60 /></P>
        <P>Pincode : <INPUT Name="Pincode" Type="Text" Size=6 /></P>
        <INPUT Name="act" onClick="verifyData( )" Type="button"Value="Verify" />
    </FORM></BODY>
    <SCRIPT Language="JavaScript">
        document.forms[0].Firstname.focus( );
    </SCRIPT>
</HTML>
```

The Form Object's Methods

HTML forms can be made up of a variety of HTML elements that accept user input. The <FORM> </FORM> HTML tags enclose the HTML elements that make up the form. Once a JavaScript enabled browser encounters these tags in an HTML file the JavaScript enabled browser creates a **form object** in memory, which is held as an element of the forms array. The form object has properties like **Name, Method** and **Action.**

Method

The **Method** property of a form is used to specify the method used to send data captured by various form elements back to the web server. The method used can be either **Get** or **Post.**

The **Get** method sends the data captured by form elements to the web server encoded into a URL, which points to a web server. The data captured in form elements is appended to the URL.

This technique is used when there is a small amount of data being sent back to the web server. The maximum amount of data that can be sent back to the web server using this method is 1024 bytes.

The **Post** method sends the data captured by form elements back to the web server as a separate bit-stream of data. When there is a large amount of data to be sent back to the web server, this is the method used.

If the method attribute is not specified within the <FORM> </FORM> tags, the default method used by the browser to send data back to the web server is the *Get* method, *i.e. as an encoded URL.*

Action

The **Action** attribute of the **<FORM>...</FORM>** tags points to the URL (*address*) of a program on the web server that will process the form data captured and being sent back. The server side program that processes this data can be written in any scripting language that the web server understands.

The Text Element

Text elements are data entry fields used in HTML forms. Text fields accept a single line of text entry.

Properties

The text object has the following properties:

- name
- value

Methods

The text object has the following methods:

- focus()
- blur()
- select()
  (Selects the text in the data entry field, i.e. causes the text to be highlighted).

Events

- Focus()
- Blur()
- Select()
- Change()

JavaScript provides the following event handlers for the text object's events:

- onFocus()
- onBlur()
- onSelect()
- onChange()

Syntax:

<INPUT Name="`<NameOfTheObject>`**" Type="Text" Value="**`<DefaultValue>`**">**

Example:

```
<INPUT Name="txt_age" Type="Text" Value="18">
```

This places a **Text** field (i.e. a single line text edit area) within an HTML form, which can be referenced by the name **txt_age**. The text field will immediately display the value **18**.

The Password Element

The password element is a unique type of text entry field. All keystrokes for this field are displayed as an asterisk [*]. This makes the password element ideal for accepting input of confidential information, such as a password, bank account number or a personal identification number.

Properties

The password object has the following properties:

- defaultValue
- name
- value

Methods

The password object has the following methods:

- focus()
- blur()
- select()
  (Selects text in the password element, i.e. causes selected text to be highlighted).

Events

The password object has the following methods:

- Focus()
- Blur()
- Select()
- Change()

JavaScript provides the following event handlers for the password object's events:

- onFocus()
- onBlur()
- onSelect()
- onChange()

Syntax:

<INPUT Name="`<NameOfTheObject>`**" Type="Password" Value="**`<DefaultValue>`**">**

Example:

```
<INPUT Name="txt_usr_pswd" Type="Password" Value="">
```

This places a **Password** field within an HTML form, which can be referenced by the name **txt_user_name**.

The Button Element

An HTML button element is a commonly used form object. It is generally used to trigger appropriate form level processing.

Properties

- name
- value

Method

- click()

Event

- click()

JavaScript provides the following event handler for the button object's event:

- onClick().

Syntax:

<INPUT Name="`<NameOfTheObject>`**" Type="Button" Value="**`<ButtonLabel>`**">**

Example:

```
<INPUT Name="btn_check" Type="Button" Value="Verify...">
```

This places a **Button** on the HTML form named **btn_check**. The button will display the text **Verify...** on its face as a label.

Exercise 4:

Focus: Develop a HTML Page, which accepts:

- Any mathematical expression
- Evaluates the expression
- Displays the result of the evaluation

To achieve this a form is created which has two **Text** objects and one **Button** object as shown in diagram 10.5. The first **Text** object is used to accept a mathematical expression for evaluation. When the **Button** object (Calculate) is clicked on, the second **Text** object will display the output of the evaluation of the expression entered into the first **Text** object. Refer to the HTML and JavaScript code described in Exercise 4.

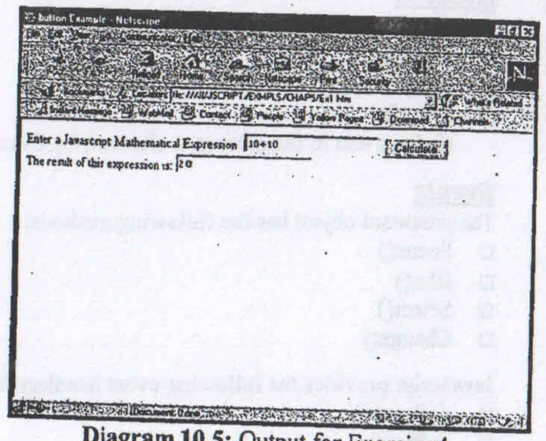

Diagram 10.5: Output for Exercise 4.

The Code listing for Exercise 4:

```
<HTML>
    <HEAD><TITLE>Using Text and Button objects in an HTML Form</TITLE>
    <SCRIPT Language="JavaScript">
        function calculate(form) {
            form.results.value = eval(form.entry.value);
        }
    </SCRIPT></HEAD>
    <BODY><FORM>Enter a Javascript Mathematical Expression:
        <INPUT Type="Text" Name="entry" Value="" />
        <INPUT Type="button" Value="Calculate" onClick="calculate(this.form);" /><BR/>
        The result of this expression is:
        <INPUT Type="Text" Name="results" onFocus="this.blur();" />
    </FORM></BODY>
</HTML>
```

When this program is executed in a JavaScript enabled browser, the used defined JavaScript function **calculate()** is registered by the browser first as it is written between the **<HEAD>... </HEAD>** tags.

The lines of code between the <FORM> </FORM> tags creates a form as seen in diagram 10.5. Enter an expression in the first 'Text' object on the form for evaluation and with the mouse cursor click on the button 'Calculate'.

When the button Calculate is clicked on its **click()** event fires. This in turn will activate the JavaScript event handler **onClick**. This invokes the function **calculate()**.

The function calculate() evaluates the expression entered in the first **Text** object and displays its output in the second **Text** object on the form.

The Submit (Button) Element

The submit button is a special purpose button. The submit button submits the **current** data held in **each** data aware, form element to a Web Sever for further processing.

Properties

The submit button is associated with 2 properties:

- name
- value

Method

The submit button's method is:

- click()

Event

The submit button's event is:

- click()

JavaScript provides the following event handler for the Submit button's event:

- onClick().

Example:

```
<INPUT Name="btn_submit" Type="Submit" Value="SUBMIT DATA">
```

This will place a **Submit** button on an HTML form, named **btn_submit**. The Submit button will display **SUBMIT DATA**. When this button is pressed the contents of each data aware, form element will be sent back to a web server for further processing.

Note

There is no example to the functionality of the HTML **submit** button at this stage since this material is only focused on client side JavaScript exclusively.

In subsequent chapters where PERL CGI programming is covered the **GET** and **POST** methods of the **Form** object will be covered with appropriate examples.

The Reset (Button) Element

Properties

The reset button has 2 properties, (*the same as the button*):

- name
- value

Methods

The reset button has only one method associated with it:

- click()

Events

The reset button has only one event associated with it:

- click()

The reset button's JavaScript event handler is:

- onClick( )

Example:

```
<INPUT Name="btn_reset" Type="Reset" Value=" RESET FORM ">
```

This places a Reset button on the HTML form, named **btn_reset**. When this button is pressed each data aware form object will be reset to their default values. All user-input values will be initialized.

The Submit and Reset buttons are usually used together on an HTML form. The Submit button will cause the contents of all the data aware form elements to be dispatched to the web server for further processing. The Reset button will empty the contents of the form objects so that the form is ready for reuse.

Note

It is good programming practice to simulate the reset button in a form's unLoad() event.

This will ensure that whenever a form is closed its fields will be emptied so that the next time the form is opened, its form elements will be empty, ready for reuse.

Exercise 5: To illustrate how the **Reset** button on a form functions.

Focus: Create a form having Textboxes, Radio buttons, a Checkbox and a Reset button as shown in diagram 10.6.1. On clicking the Reset button, the entire form is reset (i.e. cleared). Refer to the HTML and JavaScript code described in Exercise 5.

When the Reset button is clicked the form will appear as in diagram 10.6.2.

The diagram 10.6.2 shows an Alert box that displays a message indicating that all the form elements have been cleared.

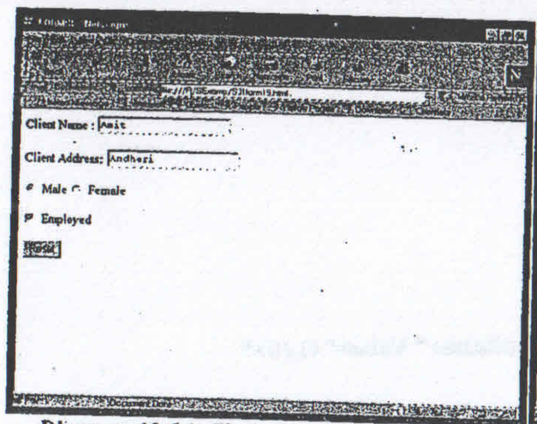

Diagram 10.6.1: First message for Exercise 5.

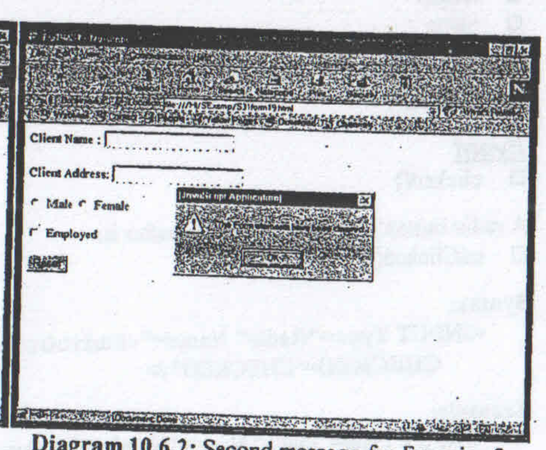

Diagram 10.6.2: Second message for Exercise 5.

The code listing for Exercise 5:

```
<HTML>
   <HEAD><TITLE>FORMS1</TITLE>
<!-- Using Reset Button //-->
   <SCRIPT>
      function func(f1) {   alert("The Form Elements have been cleared");   }
   </SCRIPT></HEAD>
   <BODY><FORM onReset="func(this.form)">
      Client Name : <INPUT Name="Text1" Type="Text" Value="" /><BR/><BR/>
      Client Address : <INPUT Name="Text2" Type="Text" Value="" /><BR/><BR/>
      <INPUT Type="Radio" Name="Radio" Value="" />Male
      <INPUT Type="Radio" Name="Radio" Value="" />Female<BR/><BR/>
```



```
      <INPUT Type="CheckBox" Name="Check" Value="" />Employed<BR/><BR/>
      <INPUT Name="Rst" Type="Reset" Value="Reset" />
   </FORM></BODY>
</HTML>
```

The Checkbox Element

A checkbox is an HTML form object that behaves as a toggle switch. This means a checkbox can be in either one of two states, either **checked** or **unchecked**. A checkbox is used to return a single specific value to a web server. Either **T** or **F** or **1** or **0** can be returned depending upon whether the checkbox is checked or unchecked. Based on the value returned from the HTML form, a web server scrip can decide what further processing the web server should do.

Properties

- name
- value
- checked

Method

- click()

Event

- click()

A checkbox's JavaScript event handler is:

- onClick()

Syntax:

<INPUT Name="<NameOfTheObject>**" Type="checkbox" Value="**<Yes/No>**" CHECKED>**

Example:

<INPUT Name="Employed" Type="Checkbox" VALUE="Yes" CHECKED>

This places a **Checkbox** on an HTML form, which can be referenced by the name **Employed**. The Value attribute assigns a meaning to the checkbox. This is the value that is returned if the checkbox is **Checked**.

The checkbox is checked on (*i.e. marked checked*) by default. Hence, if the check box is not unchecked and the form is submitted to a web server this check box will automatically return the value **Yes**.

Exercise 6:

Focus: Develop a HTML Page, which uses two text fields and a checkbox. The **first** text object accepts a numeric value. Depending on the **checked** or **unchecked** state of the **checkbox**, the **second** text object displays:

Checkbox is **not** checked	**Double** the numeric value entered
Checkbox **is** checked	The **square** of the numeric value entered

If the **second** text object is loaded with a numeric value depending on the **checked** or **unchecked** state of the **checkbox**, the **first** text object displays:

Checkbox is **not** checked	**Half** the numeric value entered
Checkbox **is** checked	The **square root** of the numeric value entered

Whenever the state of the checkbox is changed, recalculation and display of values takes place appropriately. The diagram 10.7 shows a form created to perform mathematical calculations as specified above. Refer to the HTML and JavaScript code described in Exercise 6.

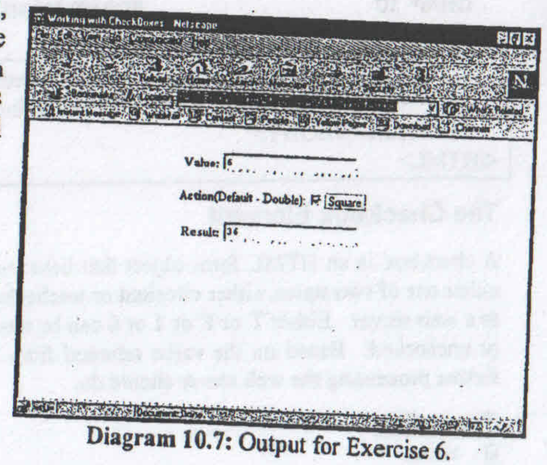

Diagram 10.7: Output for Exercise 6.

The Code Listing for Exercise 6:

```
<HTML>
  <HEAD><TITLE>Working with Check
        Boxes</TITLE>
  <SCRIPT>
    function calculate(form, callingField) {
      if (callingField == "result") {
        if(form.square.checked) {
          form.entry.value = Math.sqrt(form.result.value);
        }
        else {   form.entry.value = form.result.value/2;     }
      }
      else {
        if(form.square.checked) {
          form.result.value = form.entry.value * form.entry.value;
        }
        else {   form.result.value = form.entry.value * 2;    }
      }
    }
  </SCRIPT></HEAD>
  <BODY><FORM><CENTER><BR/>
    <B>Value:</B>
    <INPUT Name="entry" onChange="calculate(this.form, this.name);" Type="Text" Value="0" />
    <BR/><BR/><B>Action</B>(Default - Double):
    <INPUT Name="square" onClick="calculate(this.form, this.name);" Type="Checkbox" />Square
    <BR/><BR/><B>Result:</B>
    <INPUT onChange="calculate(this.form, this.name);" Name="result" Type="Text" Value=0 />
  </CENTER></FORM></BODY>
</HTML>
```

Analysis

This example illustrates the use of the **onClick** event handler of a check box. If the check box is *checked* or *not* passes an appropriate Boolean value back to the JavaScript, user defined, function **calculate()**. Depending upon this value a decision is made.

A checkbox named **square** is placed on the HTML form it can be **checked** or **unchecked**.

If a value is placed in the **first** text field on the form and the checkbox is *checked*, the *square* of the value in the first text field will be displayed in the **second** text field. If the checkbox is unchecked *double* the numeric value of the first text field will be displayed in the second text field.

If a value is placed in the **second** text field on the form and the checkbox is checked, the *half* the value held in the second text field will be displayed in the **first** text field. If the checkbox it is unchecked then the *square root* of the value in the second text field will be displayed in the first text field.

The *onClick* event handler of the *checkbox* ensures that when checkbox is clicked, to change (*toggle*) its state, all values are recalculated.

The onChange event handler of the *text boxes* ensures that when the values in the text fields change, the form is recalculated.

The Radio Element

The radio button element has two states and can toggle between them. The one special exception is that when several radio buttons are combined into a radio (button) group only a single radio button can be selected at any given time. Giving the **same name** to all the radio buttons places them in the same radio group.

Properties

- checked
- index
- length
- name

Method

- clicked( )

Event

- clicked()

A radio button's JavaScript event handler is:

- onClicked( )

Syntax:

```
<INPUT Type="Radio" Name="<RadioGroupName>" Value="<1/0>"
    CHECKED="CHECKED" />
```

Example:

```
<INPUT Type="Radio" Name="Numbers" Value="1" Checked="Checked" >1<BR/>
<INPUT Type="Radio" Name="Numbers" Value="2" >2<BR/>
<INPUT Type="Radio" Name="Numbers" Value="3" >3<BR/>.
```

This places three *radio buttons* on the HTML form, belonging to the same radio group called *'Numbers'*. The first radio button will be set as *active* as soon as the HTML page is visible.

Exercise 7:

Focus: An HTML form displays two text boxes and two radio buttons. The first text box accepts a numeric value. If the **first** radio button if active, **double** the number entered in the first text field will be displayed in the second text field. If the **second** radio button is active the **square** of the number entered in the first field will be displayed in the second text field.

Whenever the radio buttons are toggled, recalculation takes place and the output is displayed appropriately.

The diagram 10.8 show a form created to perform mathematical calculations as specified above. Refer to the HTML and JavaScript code described in Exercise 7.

The Code Listing for Exercise 7:

```
<HTML>
    <HEAD><TITLE> Working with Radio
            Buttons </TITLE>
    <SCRIPT>
        function calculate(form) {
            if(form.elements[1].checked) {
                form.result.value =
                                form.entry.value
                                * 2;
            }
          else {
                form.result.value =
form.entry.value * form.entry.value;
            }
        }
    </SCRIPT></HEAD>
    <BODY><FORM><CENTER><BR/>
        <B>Value:</B>
        <INPUT Name="entry" Type="Text" Value="0" /><BR/><BR/><SPACER Size="190" />
        <B>Action:<B><BR/><SPACER Size="225">
        <INPUT Name="action1" onClick="calculate(this.form);" Type="Radio" Value="twice" />
        Double
        <BR/><SPACER Size = "225" />
        <INPUT Name="action1" onClick="calculate(this.form);" Type="Radio" Value="square" />
        Square<BR/><BR/>
        <B>Result:</B>
        <INPUT Name="result" onFocus="this.blur();" Type="Text" />
    </CENTER></FORM></BODY>
</HTML>
```

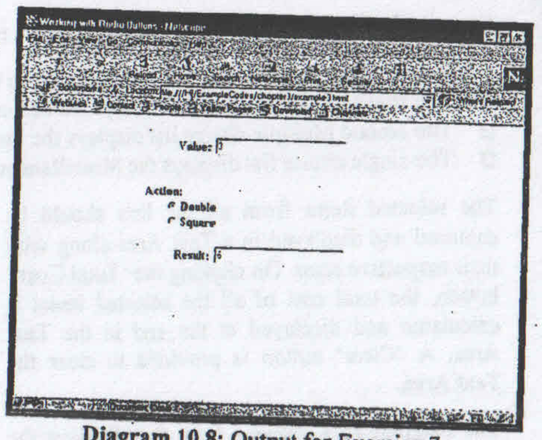

Diagram 10.8: Output for Exercise 7.

Analysis

This example illustrates the use of the *onClick* and *onChange* event handlers.

Two radio buttons are displayed. These radio buttons belong to the same radio group because each of the radio buttons has been given the same **name** as specified in the <INPUT> tag. Each radio button is named *square*.

In the above example, if the first radio button is checked, the program will double the numeric value held in the first text field. On checking the second radio button, the program will square the value held in the first text box. In either case the results of this processing will be displayed in the second text box.

The *onClick* event handler in the radio button ensures that when radio buttons are toggled, the recalculation takes place.

The *onChange* event handler in the text boxes ensures that when changes to values in the text boxes takes place, recalculation is done.

The TextArea Element

The textArea form element provides a way to create a custom-sized, multiple line, text entry object, which can be placed on an HTML form.

Properties

- defaultChecked
- name
- value

Methods:

- Focus()
- Blur()
- Select()

Events:

- Focus()
- Blur()
- Select()

The Javacript eventhandlers of a TextArea are:

- onFocus()
- onBlur()
- onSelect()

Syntax:

```
<INPUT TYPE="TextArea" Name="<MyTextArea>" Row="10" Cols="25">
    <H2>Enter Data Here</H2></TEXTAREA>
```

This will place a textArea object on a form. It can be referenced by the name *MyTextArea*. The textarea can accommodate 25 characters per line and 10 such lines inside its user defined boundaries. The boundaries are set by the values passed to the ROW and COLS attributes of the TextArea object.

The Select And Option Elements

A Select object on an HTML form appears as drop-down list or a scrollable list of selectable items. To conserve form space, the scrollable list of selectable items is used.

The list of items to choose from is described between the **<SELECT>...</SELECT>** tags using the **<OPTION>** tag. The <OPTION> tag is not a paired tag.

Example:

```
<SELECT Name="Items">
    <OPTION SELECTED> French Fries
    <OPTION>Hamburgers
    <OPTION>Hot Dogs
</SELECT>
```

This will place a Select object on an HTML form. The select object can be referenced by the name "Items". There will be three choices French Fries, Hamburgers, and Hot Dogs available in the select object. Using the <SIZE> attributes the number of items visible in a select list can be controlled.

If the <SIZE> attribute is set to a value less than the actual choices available in the select list a scrollable list will be created.

The following code snippet will display a drop down list on an HTML form, which displays 2 items and has a vertical scrollbar.

Example:

```
<SELECT Name = "Items" Size="2">
    <OPTION SELECTED>French Fries</ OPTION>
    <OPTION>Hamburgers</ OPTION>
    <OPTION>Hot Dogs</ OPTION>
    <OPTION>Ice Cream Cones</ OPTION>
    <OPTION>Salads</ OPTION>
</SELECT>
```

Using a Select object, only **one** item can be chosen, from the list of items.

Multi Choice Select Lists

To use a Select object, from which multiple choices can be made from within the list the **MULTIPLE** attribute must be set in the select object.

```
<SELECT Name = "Items"  Size ="2" MULTIPLE>
    <OPTION SELECTED>French Fries</ OPTION>
    <OPTION>Hamburgers</ OPTION>
    <OPTION>Hot Dogs</ OPTION>
    <OPTION>Ice Cream Cones</ OPTION>
    <OPTION >Salads</ OPTION>
</SELECT>
```

Selection lists are accessible in JavaScript through the *Select* object. When the <SELECT> </SELECT> HTML tags is encountered in an HTML page a JavaScript enabled browser's creates an array in memory that holds the items available in the list.

In each of the sample code snippets an array of the name *Items* is created in memory. This array has an array index which starts with '0' (i.e. *zero*). The value of any element in the array can be obtained in JavaScript by using:

memvarname="arrayname.indexvalue"

Properties

- selectedIndex
- defaultSelected
- index
- selected
- text
- value

Methods

- Blur
- Focus
- Change

Events

- Blur()
- Focus()
- Change()

JavaScript Event handlers bound to these events:

- onBlur()
- onFocus()
- onChange()

Exercise 8: Illustrate the use of Select and Option elements on a HTML form.

Focus: The form consists of a two Multiple choice lists and one single choice list.

- The first Multiple choice list displays the Major dishes available.
- The second Multiple choice list displays the Starters available.
- The single choice list displays the Miscellaneous (Milkshakes, Soft drinks, Softy) available.

The selected items from all the lists should be captured and displayed in a Text Area along with their respective costs. On clicking the 'Total Cost ' button, the total cost of all the selected items is calculated and displayed at the end in the Text Area. A 'Clear' button is provided to clear the Text Area.

The diagram 10.9 shows a form that displays the basic form elements required to capture data in a commercial application as specified above.

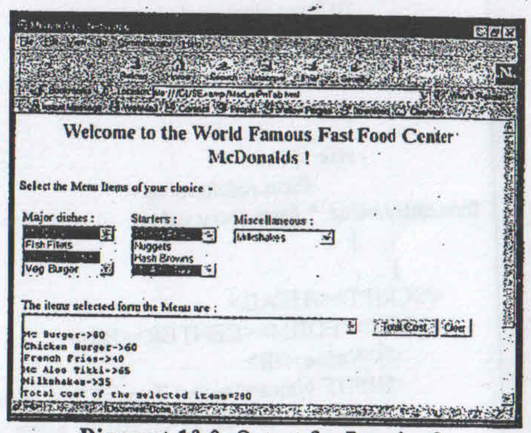

Diagram 10.9: Output for Exercise 8.

The code listing for Exercise 8:

```
<HTML>
    <HEAD><TITLE>McDonalds</TITLE>
    <SCRIPT Language="JavaScript">
        var m;
        function pick(F1) {
            var z=" ";
            for(j=0; j<3; j++) {
                for(i=0; i<F1.elements[j].length; i++) {
                    if (F1.elements[j][i].selected) {
                        var y=F1.elements[j].options[i].value;
                        z=z + "\n" + y;
                        F1.elements[3].value=z;
                    }
                }
            }
            m=z;
        }
        function cal(F1) {
            var d=0;
            for(j=0; j<3; j++) {
                for(i=0; i<F1.elements[j].length; i++) {
                    if (F1.elements[j][i].selected) {
                        var y=F1.elements[j].options[i].value;
                        s=new String(y);
                        var a=s.indexOf(">");
                        var b=s.substring(a+1,a+3);
                        c=parseInt(b);
```

```
                    d=d+c;
                }
            }
        }
        p="Total cost of the selected items=" + d;
        m=m + "\n" + p;
        F1.elements[3].value=m;
    }
    function clr(F1) {
        F1.elements[3].value=" ";
    }
</SCRIPT></HEAD>
<BODY>
    <H2><CENTER>
        <SPAN Style="color: blue; ">Welcome to the World Famous Fast Food Center </SPAN>
        <SPAN Style="color: red;">McDonalds !</SPAN>
    </CENTER></H2>
    <FORM Name="F1">Select the Menu Items of your choice - <BR/><BR/>
    <TABLE><TR valign="Top"><TD>Major dishes :<BR/>
        <SELECT Name="s1" MULTIPLE onBlur="pick(this.form)">
            <OPTION Value="Mc Burger->80" SELECTED> Mc Burger</OPTION>
            <OPTION Value="Fish Fillets->70"> Fish Fillets</OPTION>
            <OPTION Value="Chicken Burger->60"> Chicken Burger</OPTION>
            <OPTION Value="Veg. Burger->45"> Veg. Burger</OPTION>
        </SELECT><BR/><BR/>
    </TD><TD></TD><TD></TD><TD>
        Starters :<BR/>
        <SELECT Name="s2" MULTIPLE onBlur="pick(this.form)">
            <OPTION Value="French Fries->40"> French Fries</OPTION>
            <OPTION Value="Nuggets->50">Nuggets</OPTION>
            <OPTION Value="Hash Browns->55">Hash Browns</OPTION>
            <OPTION Value="Mc Aloo Tikki->65">Mc Aloo Tikki</OPTION>
        </SELECT><BR/><BR/>
    </TD><TD></TD><TD></TD><TD>
        Miscellaneous :<BR/>
        <SELECT Name="s3" onBlur="pick(this.form)">
            <OPTION Value=" ">'Check these out'
            <OPTION Value="Milkshakes->35">Milkshakes</OPTION>
            <OPTION Value="Soft drinks->20">Soft drinks</OPTION>
            <OPTION Value="Softy->25">Softy</OPTION>
        </SELECT><BR/><BR/>
    </TD><TD></TD><TD></TD></TR></TABLE><BR/>
    <TABLE><TR valign="Top"><TD>
        The items selected form the Menu are :
        <TEXTAREA Name="TA1" Rows="10" Cols="50"></TEXTAREA><BR/><BR/>
    </TD><TD></TD><TD></TD><TD>
        <BR/><Input Type="button" Value="Total Cost" onClick="cal(this.form)" />
        <Input Type="button" Value="Clear" onClick="clr(this.form)" />
```



```
        </TD></TR></TABLE>
    </FORM></BODY>
</HTML>
```

OTHER BUILT-IN OBJECTS IN JAVACRIPT

JavaScript provides a few other objects that are not related to the current window or the document loaded in the current window. These objects are used quite extensively for data processing in JavaScript.

The String Object

Every string in JavaScript is an object. The string object has a number of properties, methods, which helps perform a variety of manipulations on a given string. These include methods for searching for a string, extracting sub-strings from a string and applying various HTML tags to string contents and so on.

Properties

The string object has only one property:

Property	Description
Length	An integer value indicating the number of characters in the string.

Table 10.4

Methods

The flexibility and power of the string object rests in the wide variety of methods available to manipulate the contents of the string. Some of the methods available for string manipulation are as follows:

Method	Description
big( )	Surrounds the string with the HTML big tag
blink( )	Surrounds the string with the HTML blink tag
bold( )	Surrounds the string with the HTML bold tag
charat( )	Given an index as an argument, returns the character at the Specified index
italics( )	Surrounds the string with the HTML <I> tag.
tolowercase( )	Makes the entire string lowercase.
touppercase( )	Makes the entire string uppercase.
substring( )	Given two indexes, returns the substring starting at the first index and ending with the character before the last index. If the second index is greater, the substring with the second index and ends with the character before the first index; if the two indexes are equal, returns the empty string.

Table 10.5

Example:

If a variable named **sample contains** a value **"Hello"**, then:

sample.substring(0,3)	returns	Hel
sample.toLowercase()		hello
sample.CharAt(3)		l
sample.italics()		*Hello*

sample.substring(2,4)	returns	ll
sample.toUpperCase()		HELLO
sample.bold()		**Hello**

The Math Object

The **math** object provides methods and properties to move beyond simple arithmetic manipulations offered by arithmetic operators.

Among the features offered by the **math** object are several special values such as pi, natural logarithms, common square roots, trigonometric methods, rounding methods, an absolute value method, and more.

Properties

Property	Description
E	Euler's constant – the base of natural logarithms.
LN10	The natural logarithms of 10(roughly 2.302).
LN2	The natural logarithms of 2 (roughly 0.693).
PI	The ratio of the circumference of a circle to the diameter of the same circle (roughly 3.1415).

Table 10.6

Methods

Method	Description
abs( )	Calculates the absolute value of a number.
ceil( )	Returns the next integer greater than or equal to a number.
cos( )	Calculates the cosine of a number.
floor( )	Returns the next integer less than or equal to a number.
pow( )	Calculates the value of one number to the power of a second number – takes two arguments.
random( )	Returns a random number between zero and one.
sin( )	Calculates the sine of a number.
sqrt( )	Calculates the square root of a number.
tan( )	Calculates the tangent of a number.

Table 10.7

Example:

abs(-15)	returns	15
tan(45)		1

ceil(15.45)	returns	16
pow(2,2)		4

The Date Object

The Date object enables JavaScript programmers to create an object that contains information about a particular date and provides a set of methods to work with that information. To create an instance of the date object, use the keyword new as follows:

var my_date = new Date(<parameters>);

The parameter, if left empty, indicates today's date & time. The parameter could be any specific date format.

Methods

The date object provides the following methods:

Method	Description
getDate( )	Returns the day of the month as an integer from 1 to 31
setDate( )	Sets the day of the month based on an integer argument from 1 to 31
getHours( )	Returns the hours as an integer from 0 and 23
setHours( )	Sets the hours based on an argument from 0 to 23
getTime( )	Returns the number of milliseconds since 1 January 1970 at 00:00:00.
setTime( )	Sets the time based on an argument representing the number of milliseconds since 1 January 1970 at 00:00:00.

Table 10.8

Exercise 9:

Focus: Write JavaScript code to display the current date and time in a Browser.

The diagram 10.10 shows a form created to display the current date and time in a Browser as specified above. Refer to the HTML and JavaScript code described in Exercise Nine.

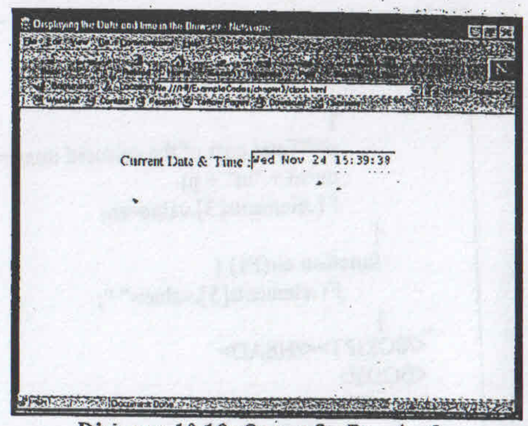

Diagram 10.10: Output for Exercise 9.

The code listing for Exercise 9:

```
<HTML>
    <HEAD>
        <TITLE>Displaying the Date and time in
                the Browser</TITLE>
    </HEAD>
    <BODY>
        <SCRIPT Language="JavaScript">
        function begin(form) {
            form_name = form;
            time_out=window.setTimeout("display_date()",500)
        }
        function display_date() {
            form_name.date.value=new Date();
            time_out=window.setTimeout("display_date()",1000)
        }
        function display_clock() {
            document.write('<CENTER><SPAN Style="color:red;font-size:14px;" >
            <FORM Name=time_form><BR/> <BR/> Current Date & Time :')
            document.write('<INPUT Name="date" Size="19" Value=""
            /></FORM></SPAN></CENTER>')
            begin(document.time_form);
        }
        display_clock();
        </SCRIPT>
    </BODY>
</HTML>
```

Other JavaScript objects can be summarized as follows:

Name	Description
String	The string object enables programs to work with and manipulate strings of text, including extracting substrings and converting text to upper or lowercase characters
Math	The math object provides methods to perform trigonometric functions (sine and tangent) as well as general mathematical functions, such as square roots.
Date	With the date object, programs can work with the current date or create instances for specific dates. The object includes methods for calculating the difference between two dates and working with times.

Table 10.9

USER DEFINED OBJECTS

In addition to the wide range of built-in objects, JavaScript permits the creation of user defined objects. As with every other object, a user-defined object will also be associated with properties and methods, which belong to it. After creation of such an object, any number of instances of this object can be created and used.

For example, if the name, age, and marks obtained by a student needs to be stored and there are fifty such students. It is completely possible in JavaScript to create and object named **Student**, which has three **properties** name, age and marks.

This user-defined object would also require **methods** that will allow the storage of name, age and the marks obtained by a student as properties of the object.

Creating A User Defined Object

A user-defined object called **student** is to be created with three properties:

- Name
- Age
- Marks

The object **student** also will include a method **insert()**.

Just as variables are named containers of data in traditional languages, similarly properties are named containers used to hold data, such as numbers or text in an object oriented approach.

Properties

The properties of the object student can be referenced as:

- Student.Name
- Student.Age
- Student.Marks

Methods

The methods of the object student can be referenced as:

- Student.insert()

Example:

```
function student(name, age, marks) {
    this.name = name;
    this.age = age;
    this.marks = marks;
}
```

This creates an object called student with three properties, name, age and marks.

Note

- **this:** refers to the current object in focus. For example, **this.name** will refer to the name of the current object.
- **Parent:** refers to the parent of the current object.

Instances

In object oriented programming, once an object is defined and created, any number of copies of the object can be created and used to store information. If there are fifty students whose data has to be maintained then 50 copies of the object **student** must be created to store the information of the fifty students.

Every new copy of the object is called an **Instance** of the object. The process of creating an instance of an object is termed as **Instantiation** of the object.

Example:

```
student1 = new student("Anil",10,75);
student2 = new student("Chhaya",9,82);
```

Objects Within Objects

Just as objects consist of properties and methods, they can also consist of other objects. For instance, the marks of a student will further depend on the marks of individual subjects Science, Math and English. Hence, another object can be created called as **Marks**, which has three properties:

- English
- Science
- Math

This object can now be included in the student object, where marks is no more a singular property, but an **object** called **Marks**, which consists of three individual properties English, Science and Math.

In such a case, a new instance of the object **Marks** must be created first, and this new instance must be included in the object **student**. The properties of this compound object can now be referenced as:

- Student.Name
- Student.Age
- Student.Marks.English
- Student.Marks.Science
- Student.Marks.Maths

Example:

Creating the Marks object:

```
function Marks(English, Science, Math) {
    this.Math = Math;
    this.English = English;
    this.Science = Science;
}
```

Creating the Student object:

```
function Student(name, age, marks) {
    this.name = name;
    this.age = age;
    this.marks = marks;
}
```

The technique of creating an object instance embedded in another object is as follows:

```
anilGrade = new marks(75, 80, 77);
chhayaGrade = new marks(82, 88, 75);
student1 = new student("Anil", 10, anilGrade);
student2 = new student("Chhaya", 9, chhayaGrade);
```

Under these circumstances the property marks of the object student actually holds another object 'anilGrade' or 'chhayaGrade' which in turn hold marks for Math, English or Science.

To access the Math marks of Anil the technique would be:

```
myvar = student1.anilGrade.Math
```

The variable **myvar** will now hold the Math marks obtained by Anil.

Example10:
This example illustrates opening a new window when a link on a web page is clicked. The new window opened, is closed by placing a button on the new window and writing JavaScript code in the **onClick** event of the button

```
<HTML>
   <HEAD><SCRIPT>
      function makeNewWindow() {
         var newwindow = prompt("Enter the document to go to in format http://www.url.com", "");
         if (!(newwindow == null || newwindow == "" )) {
            window.open(newwindow , "", "status, menubar=yes,resizable=no, toolbar=no")
         }
         else {
            var blankwindow=window.open("" ,"", "status, menubar=yes,resizable=no, toolbar=no")
            var windowcontent="<HTML><BODY Style='background-color:#FFFFFF'><H1>
            New window</H1>"
            windowcontent += "<FORM><INPUT Name=close onClick=window.close()"
            windowcontent += " Type=Button Value=Close /></FORM></BODY></HTML>"
         blankwindow.document.write(windowcontent)
         }
      }
   </SCRIPT></HEAD>
   <BODY><FORM>
      <INPUT Name="open" onClick="makeNewWindow()" Type="button" Value="Open URL">
   </FORM></BODY>
</HTML>
```

Example 11:
This example illustrates an application with two frames. The top frame contains entry fields for the background color, text color, link color, active link color and visited link color and a button to enable users to test their color combinations. When the user presses the button, the script loads a simple document using specified colors, into the lower frame.

The Parent Frameset for the Color Tester:

```
<HTML>
   <HEAD><TITLE>Example 11 On Document Object</TITLE></HEAD>
   <FRAMESET Rows="45%,*">
      <FRAME Src="pick.html">
      <FRAME Name="output" Src="blank.html">
   </FRAMESET>
</HTML>
```

Pick.html file

```
<HTML>
   <HEAD><SCRIPT Language="JavaScript">
   <!-- HIDE FROM THE BROWSERS
      function display(form) {
         doc = open(" ","output");
         doc.document.write('<BODY Style="background-color:'+ form.bg.value
         +';color:'+form.fg.value+'"');
         doc.document.write('" Link="' + form.link.value + '" ALink="'+ form.alink.value);
         doc.document.write('" VLink="'+ form.vlink.value + '">');
         doc.document.write("<H1>This is a Test</H1>You Have Selected These Colors<BR/>");
         doc.document.write('<A HRef="# ">This is a Test Link</A></BODY>');
         doc.document.close();
      }
   // STOP HIDING SCRIPT -->
   </SCRIPT></HEAD>
   <BODY><CENTER><SCRIPT Language="JavaScript">
   <!-- HIDE FROM OTHER BROWSERS
      document.write('<H1>The Color Picker</H1><FORM Method=POST>');
      document.write('Enter Colors:<BR/>');
      document.write('Background: <INPUT Name="bg" Type=Text Value="' + document.bgColor +
                     '" /><BR/>');
      document.write('Text: <INPUT Name="fg" Type=Text Value="' + document.fgColor + '"
                     /><BR/>');
      document.write('Link: <INPUT Name="link" Type=Text Value="' + document.linkColor + '"
                     /><BR/>');
      document.write('Active Link: <INPUT Name="alink" Type=Text Value="' +
                     document.alinkColor + '" /><BR/>');
      document.write('Followed Link: <INPUT Name="vlink" Type=Text Value="' +
                     document.vlinkColor + '" /><BR/>');
      document.write('<INPUT onClick="display(this.form);" Type=Button Value="Test" />');
      document.write('</FORM>');
      display(document.forms[0]);
   // STOP HIDING FROM OTHER BROWSERS -->
   </SCRIPT></CENTER></BODY>
</HTML>
```


Example 12:
This example illustrates the use of the History object. Using this object we can navigate through the previous or next pages opened in the browser.

```
<HTML>
   <BODY><FORM>
      <INPUT Name="back" onClick="history.back()" Type="Button" Value="Back" />
      <INPUT Name="forward" onClick="history.forward()" Type="Button" Value="Forward" />
   </FORM></BODY>
</HTML>
```

Example 13:
Capture and displays the properties of the browser using the navigator object.

```
<HTML>
   <HEAD><TITLE>Displaying A Browser's Attributes</TITLE>
   <SCRIPT language="JavaScript">

      function displayNavigatorProperties() {
      // to avoid "document.write" every where below
         with(document) {
            write("<B>appName:</B>")
            writeln(navigator.appName + "<BR/>")
```

```
                write("<B>appVersion:</B>")
                writeln(navigator.appVersion + "<BR/>")
                write("<B>appCodeName:</B>")
                writeln(navigator.appCodeName + "<BR/>")
                write("<B>platform:</B>")
                writeln(navigator.platform + "<BR/>")
                write("<B>useragent:</B>")
                writeln(navigator.useragent + "<BR/>")
                write("<B>language: </B>")
                writeln(navigator.language + "<BR/>")
                write("<B>Number of mimeTypes: </B>")
                writeln(navigator.mimeTypes.length + "<BR/>")
                write("<B>Number of plugins:</B>")
                writeln(navigator.plugins.length)
            }
        }
        function displayExplorerProperties() {
            with(document) {
                write("<B>appName:</B>")
                writeln(navigator.appName + "<BR/>")
                write("<B>appVersion:</B>")
                writeln(navigator.appVersion + "<BR/>")
                write("<B>appMinorVersion:</B>")
                writeln(navigator.appMinorVersion + "<BR/>")
                write("<B>appCodeName:</B>")
                writeln(navigator.appCodeName + "<BR/>")
                write("<B>platform:</B>")
                writeln(navigator.platform + "<BR/>")
                write("<B>cpuClass:</B>")
                writeln(navigator.cpuClass + "<BR/>")
                write("<B>useragent:</B>")
                writeln(navigator.useragent + "<BR/>")
                write("<B>cookieEnabled:</B>")
                writeln(navigator.cookieEnabled + "<BR/>")
                write("<B>browserLanguage:</B>")
                writeln(navigator.browserLanguage + "<BR/>")
                write("<B>userLanguage:</B>")
                writeln(navigator.userLanguage + "<BR/>")
                write("<B>systemLanguage:</B>")
                writeln(navigator.systemLanguage + "<BR/>")
                write("<B>onLine: </B>")
                writeln(navigator.onLine + "<BR/>")
                write("<B>Number of mimeTypes: </B>")
                writeln(navigator.mimeTypes.length + "<BR/>")
                write("<B>Number of plugins:</B>")
                writeln(navigator.plugins.length)
                write("<B>userProfile: </B>")
                writeln(navigator.userProfile)
            }
        }
```



```
        function displayBrowserProperties() {
            if(navigator.appName=="Netscape")
                displayNavigatorProperties()
            else
                if(navigator.appName=="Microsoft Internet Explorer")
                    displayExplorerProperties()
        }
        displayBrowserProperties()
    //-->
    </SCRIPT></HEAD>
</HTML>
```

SELF REVIEW QUESTIONS

FILL IN THE BLANKS

1. HTML provides _____ and _____ tags with which an HTML form can be created to capture user input.
2. The form tag has two properties namely _____ and _____.
3. The method property is used to specify the method used to send data, which can be _____ or _____.
4. The _____ attribute of the < FORM> tag points to the URL of a program on the web server that will process the form data.
5. Various form elements can be specified as attributes of the _____ tag.
6. The _____ button sends the current information held in each field of the form to the web server for further processing.
7. To have a select object from which multiple choices can be made from a list the _____ attribute must be set in the select object.
8. _____ is the only property of the string object.
9. The _____ object provides methods and properties to move beyond simple arithmetic manipulations offered by arithmetic operators.

TRUE OR FALSE

10. As soon as the <FORM></FORM> tags are encountered by the browser, the browser creates an array called form1 in the absence of a Form name being specified in the HTML code.
11. The Post method sends data captured by the form elements to the web server, encoded as part of the URL that points to the web server.
12. Text fields accept a single line of text entry.
13. The Select object allows multiple choices from a list of choices that are offered.
14. Date object enables the creation of an object that contains information about a particular date.
15. In addition to the wide range of built-in objects, JavaScript permits the creation of user defined objects.

HANDS ON EXERCISES

1. Create a web page, which accepts user information and user comments on the web site. Design the web page using form elements and check if all the Text fields have being entered with data else display an alert. Obtain an output as shown in the diagrams 10.11.1, 10.11.2 and 10.11.3.

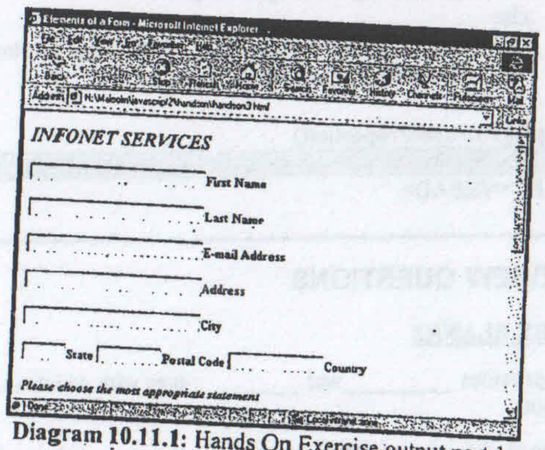

Diagram 10.11.1: Hands On Exercise output part 1.

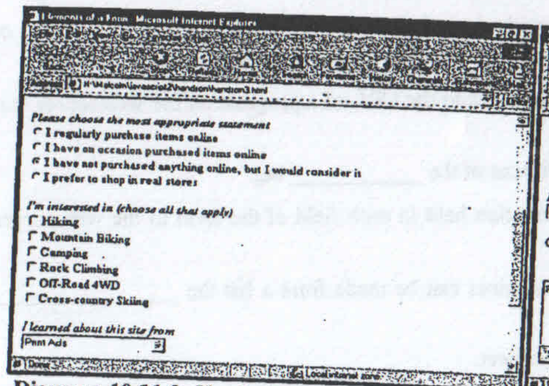

Diagram 10.11.2: Hands On Exercise output part 2.

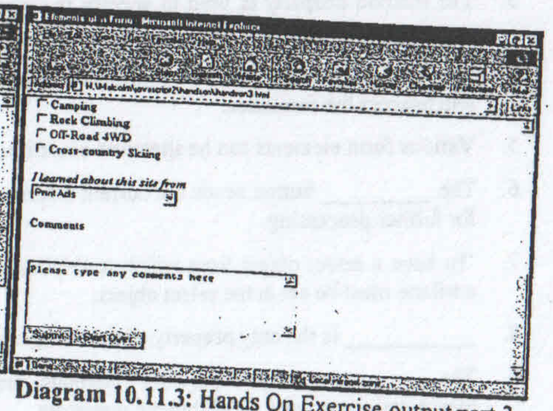

Diagram 10.11.3: Hands On Exercise output part 3.

11. COOKIES

WHAT ARE COOKIES

One of the challenges of writing applications for the World Wide Web has been inability ofthe web to maintain state. That is, after a user sends a request to the server and a web page is returned the server forgets all about the user and the page that has been downloaded. If the user clicks on a link the server doesn't have background information about what page the user is coming from, and more importantly, if the user returns to the page at a later date, there is no information available to the server abou the user's previous actions on the page.

Maintaining state can be important to developing complex interactive applications. Howeve, browsers address this problem with **cookies**, which is a method of storing information locally in the browser and sending it to the server whenever the appropriate pages are requested by the user.

The term *cookies* has no special significance. It is just a name in the same way Java is just a name for Sun's object-oriented programming language.

When a user requests a page, an HTTP request is sent to the server. The request includes a header that defines several pieces of information, including the page being requested.

The server returns an HTTP response that also includes a header. The header contains information about the document being returned, including its MIME type. These headers all contain one or moe fields of information in a basic format.

FieldName: Information

Cookie information is shared between the client browser and a server using fields in the HTTP headers. When the user requests a page for the first time, a cookie (or more than one cookie) can be stored in the browser by a **set-cookie** entry in the header of the response from the server. The set-cookie fied includes the information to be stored in the cookie along with several optional pieces of information including an expiry date, path, and server information and if the cookie requires security.

Then, when the user requests a page in the future, if a matching cookie is found among all the stored cookies, the browser sends a cookie field to the server in a request header. The header will contain the information stored in that cookie.

The set-cookie and cookie fields use a syntax to transfer significant information between client and server.

SETTING A COOKIE

Syntax:

Set-cookie: NAME=value ; EXPIRES=date; PATH=path; DOMAIN=domain; SECURE

The **NAME=value** is the only required piece of information that must be included in the set-cookie field. All other entries are optional.

Name	Description
NAME = value	Specifies the name of the cookie.
PATH = path	Specifies the path portion of the URLs for which the cookie is valid. If the URL maches both the PATH and the DOMAIN, then the cookie is sent to the server in the request header. (If left unset, the value of the PATH is the same as the document that set the cookie)

Table 11.1

Name	Description
EXPIRES = date	Specifies the expiry date of the cookie. After this date the cookie will no longer be stored by the client or sent to the server (DATE takes the form Weekday, DD-MON-YY HH:MM:SS GMT- dates are only stored in Greenwich Mean Time). By default the value of expires is set to the end of current Navigator session.
DOMAIN= domain	Specifies the domain portion of the URLs for which the cookie is valid. The default value for this attribute is the domain of the current document setting the cookie.
SECURE	Specifies that the cookie should only be transmitted over a secure link (i.e. to HTTP servers using the SSL protocol-known as HTTPS servers)

Table 11.1 (Continued)

Example:

This example illustrates the use of cookies. While the user traverses from page to page, the information entered by the user in any page is stored in cookies at the client side and made available to the user whenever the user returns to any page previously visited.

```
<HTML>
  <HEAD><SCRIPT>
    function newcookie(cookiename, value) {
      document.cookie = cookiename + "=" + value;  }
    function getCookie(cookiename) {
      var cookiefound = false;
      var start = 0;
      var end = 0;
      var cstring = document.cookie;
      var clength = 0;
      while (clength <= cstring.length) {
        start = clength;
        end = start + cookiename.length;
        if (cstring.substring(start, end) == cookiename) {
          cookiefound = true;
          break;
        }
        clength++;
      }
      if (cookiefound) {
        start = end + 1;
        end = document.cookie.indexOf(";", start);
        if (end < start) {
          end = document.cookie.length;
          var contents = document.cookie.substring(start, end);
          return contents;
        }
        else {
          var contents = document.cookie.substring(start, end);
          return contents;
        }
      }
    }
  </SCRIPT>
  </HEAD>
```



```
<BODY ><SCRIPT>
    var thisCookie = ((document.cookie !== "") && (document.cookie != null));
    var cfirst_name = (thisCookie) ? getCookie("first_name") : "";
    var cmiddle_name = (thisCookie) ? getCookie("middle_name") : "";
    var clast_name = (thisCookie) ? getCookie("last_name") : "";
    document.write('<FORM>');
    document.write('<TABLE Border="0" CellPadding="4" CellSpacing="4" Width = 100%>');
    document.write('<TR><TD ColSpan="2"><B> Personal Details (<SPAN Style="color: Red"> *
                    </SPAN> indicates Required)</B> </TD></TR>');
    document.write('<TR>');
    document.write('<TD Width="25%"> First Name <SPAN Style="color: Red">
                     * </SPAN></TD>');
    document.write('<TD Width="75%"><INPUT Name="first_name"
                    onBlur="newcookie(this.name, this.value);" Size="20" Type="Text" Value="' +
                    cfirst_name + '" /></TD></TR>');
    document.write('<TR><TD Width="25%"> Middle Name </TD>');
    document.write('<TD Width="75%"><INPUT Name="middle_name"
                    onBlur="newcookie(this.name, this.value);" Size="20" Type="Text" Value="' +
                    cmiddle_name + '" /></TD></TR>');
    document.write('<TR>');
    document.write('<TD Width="25%"> Last Name <SPAN Style="color: Red">
                     * </SPAN></TD>');
    document.write('<TD Width="75%"><INPUT Name="last_name"
                    onBlur="newcookie(this.name, this.value);" Size="20" Type="Text" Value="' +
                    clast_name + '" /></TD>');
    document.write('</TR></TABLE><BR />');
    document.write('<CENTER>');
    document.write('<INPUT Type="Submit" Value="Done" />');
    document.write('<INPUT Type="Reset" Value="Reset" />');
    document.write('</CENTER>');
    document.write('</FORM>');
  </SCRIPT></BODY>
</HTML>
```

SELF REVIEW QUESTIONS

FILL IN THE BLANKS

1. Information can be stored locally in the browser which can be sent to the server whenever required by using _____.
2. Cookie information is shared between the client browser and the server using fields in the _____ headers.
3. When the user requests a page for the first time a cookie can be stored in the browser by a _____ entry in the header of the response from the server.
4. If a matching cookie is found among all the stored cookies, the browser sends a cookie field to the server in a _____.
5. The _____ attribute specifies that the cookie should be transmitted only over a secure link.

TRUE OR FALSE

6. The name Cookie has a special significance with respect to object oriented programming language.
7. The set-cookie field includes the information to be stored in the cookie along with several optional piece of information.

B - PROJECTS IN JAVASCRIPT

Project Specifications For The First Project In JavaScript – Guest Book

Since the learning of Java Script has now been completed, it is time to consolidate this learning by building a small guest book.

A Guest Book is used to store any comments that a visitor to the site may have regarding the site. The comments (if applicable) help improve the functionality of the site.

The structure of the guest book is given in the following pages along with html source code. This is a description of how the guest book will be used by the person signing-in. The java script code working with Html which provides for various client side data capture validations.

The guest book is a html document with:
- Simple visuals in the form of .gif or .jpeg files.
- Java Script embedded HTML code.

The files required to construct these pages is available on the accompanying CD-ROM for immediate use.

If required run the HTML files first in a web browser and get a look and feel of what the project could be like. Once this is done, code the web pages according to what you believe is appropriate.

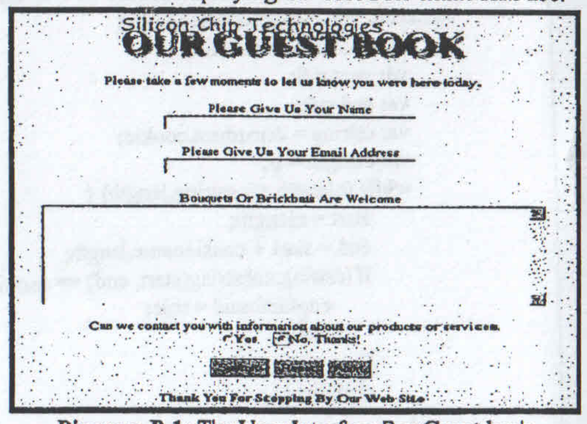

Diagram B.1: The User Interface For Guest book

For a visitor to leave comments about the site, or comments in general, the HTML form used in diagram B.1 is used. This is the guest book page.

This form captures the following:
- Visitors Name (Not compulsory)
- Emailid (Mandatory)
- A visitors Comments (Mandatory)

Feel free to improvise and get a look and feel that satisfies you.

Each HTML file is really just a simple guideline to what the web page could look like. Java script is added to perform simple client side validations.

For your guidance:
- The source code is provided in the document file.

Client Side Validations

The Client side validations to be coded in Java script are as follows
- Emailid, General Comments, cannot be left empty
- The Emailid needs to be scanned for the presence of an '@' and '.' symbol.
- The length of the Name and Emailid fields cannot exceed 30 characters.

```
<HTML>
    <HEAD><TITLE>SCT'S GUEST BOOK</TITLE>
    <SCRIPT Language = "Javascript">
    <!-- The function verify() checks whether appropriate information is filled in all the elements. If any
element is left empty, an alert() box is displayed informing the user to fill in the empty element The code
also scans the Emailid for the presence of an '@' and a '.' symbol. -->
        function verify(form) {
            for (i=1; i<=2; i++) {
                if (document.forms[0].elements[i].value == "") {
                    alert("Please fill in the " + document.forms[0].elements[i].name + " field");
                    document.forms[0].elements[i].focus( );
                    return (false);
                }
                if(document.forms[0].elements[1].value != "") {
                    pass = document.forms[0].elements[1].value.indexOf('@',0);
                    pass1 = document.forms[0].elements[1].value.indexOf('.',0);
                    if((pass==-1) || (pass1==-1)) {
                        alert("Not a valid Email address");
                        document.forms[0].elements[1].focus();
                        return (false);
                    }
                }
            }
            return(true);
        }
    <!-- This function takes the user back to the 'Home Page' from the current page -->
        function abort(form) {
            history.back();
        }
    <!-- Sets the focus on the first element when the form is loaded -->
        function set(form) {
            document.forms[0].elements[0].focus( );
        }
    <!-- The function checklen() checks that the length of name and email addresses does not exceed
30 characters -->
        function checklen(form) {
            for (i=0; i<=1; i++) {
                val=document.forms[0].elements[i].value;
                len=val.length;
                if (len > 30) {
                    alert ("Value exceeds 30 characters");
                    document.forms[0].elements[i].value="";
                    document.forms[0].elements[i].focus();
                }
            }
        }
    </SCRIPT></HEAD>
    <BODY Style="background: url(images/grid1.gif);color:green;" onLoad="set(this.form)">
        <CENTER><IMG Src="images/Gestbk.gif" Alt="gestbk.gif" /></CENTER>
        <P><CENTER>Please take a few moments to let us know you were here today.</P>
```



```
        <P><FORM Action = "" Method="POST" onSubmit=" return verify(this.form)"></P>
            <P>Please Give Us Your Name<BR/>
            <INPUT Type="text" Name="name" Size="40" onBlur="checklen(this.form)" /><BR/>
            <P>Please Give Us Your Email Address<BR/>
            <INPUT Type="text" Name="emailid" Size="40" onBlur="checklen(this.form)"
            /><BR/></P>
            <P>Bouquets Or Brickbats Are Welcome<BR/>
            <TEXTAREA Name="request" Rows="8" Cols="65"></TEXTAREA></P>
            <P>Can we contact you with information about our products or services.<BR/><BR/>
            <INPUT TYPE="radio" NAME="moreinfo" VALUE="y" />
            <SPAN Style="color: Blue">Yes</SPAN><IMG Src="images/shim.gif" Width="20"
            Height="10" />
            <INPUT TYPE="radio" NAME="moreinfo" VALUE="n" Checked="True" />
            <SPAN Style="color: Red">No, Thanks!</SPAN></P>
            <P><INPUT Type="Submit" Value="Submit"><INPUT Type="Reset" Value="Reset" />
            <INPUT Type="Button" Value="Abort" onClick = "abort(this.form)" /></P>
    </FORM>
    <P><B>Thank You For Stopping By Our Web Site</B></P></CENTER>
    </BODY>
</HTML>
```

Project Specifications For The First Project In JavaScript – Pen Pals

Since the learning of Java Script has now been completed, it is time to consolidate this learning by building a page, which offers free hosting of pen pal information. Java Script will then be used to validate data captured by this page.

Using this service, visitors can post their Name, Emailid, Sex, Date of birth, Interests and Hobbies on the Web Site. Other visitors who find compatible information in the Pen pals list would email the registrant. This could become the start of a lasting friendship.

The structure of the pen pal page is given in the following pages along with the html source code. This is a description of how the Pen Pal page is to be used by the person visiting the page. The java code working within html provides for data capture validations.

Other visitors can then refer to these pages and email of individuals who's Age, Sex, Interests and other criteria are interesting. To post Pen Pal Information on a Web Site a simple form needs to be filled in with details as shown in diagram B.2.

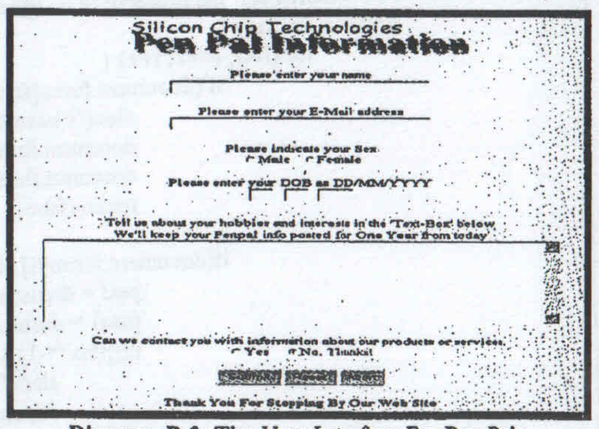

Diagram B.2: The User Interface For Pen Pals

The Pen Pal page is a html document with:

- Simple visuals in the form of .gif or .jpeg files.
- Java script embedded HTML code.

The files required to construct these pages is available on the accompanying CD-ROM for immediate use.

If required run the HTML files first in a web browser and get a look and feel of what the project could be like. Once this is done, code the web pages according to what you believe is appropriate.

Pen Pal information is hosted for free on our website to visitors to the website.

For a visitor to leave his/her information on a site. The HTML form used in diagram 1 is used. This is the Pen Pal page.

The form captures the following:

- Visitors Name (Mandatory)
- Emailid (Mandatory)
- The visitors D.O.B in DD/MM/YYYY (Mandatory)
- Textbox to retrieve users hobbies and interests (Not Compulsory)

Feel free to improvise and get a look and feel that satisfies you.

Each HTML file is really just a simple guideline to what the web page could look like. Java script is added to perform simple client side validations.

For your guidance:

- The source code is provided in the document file.

Client Side Validations:

The Client Side validations to be coded in Java script are as follows:

- Name, Email id, Sex, DOB entries, cannot be left empty
- The Email id needs to be scanned for the presence of an '@' and '.' sign
- The length of the Name and Email id fields cannot exceed 30 characters
- Ensure that the date is keyed in as "DD/MM/YYYY"

```
<HTML>
        <HEAD><TITLE>SCT's PEN PAL</TITLE>
        <SCRIPT Language = "Javascript">
<!-- The function verify() checks whether appropriate intervention is filled in all the form elements. If any
element is left empty an alert() box is displayed informing the user to fill in all the empty elements. The
code also searches the emailed for the presence of an '@' and a '.' symbol -->
                function verify() {
                        for (i=0; i<=7; i++) {
                                if (document.forms[0].elements[i].value == "") {
                                        alert("Please fill in the " +
                                        document.forms[0].elements[i].name + " field");
                                        document.forms[0].elements[i].focus( );
                                        return (false);
                                }
                                if(document.forms[0].elements[1].value != "") {
                                        pass = document.forms[0].elements[1].value.indexOf('@',0);
                                        pass1 = document.forms[0].elements[1].value.indexOf('.',0);
                                        if((pass==-1) || (pass1==-1)) {
                                                alert("not a valid email address");
                                                document.forms[0].elements[1].focus();
                                                return (false);
                                        }
                                }
                        }
```



```
                return(true);
        }
        <!-- The function checkdate() checks whether the date is between (1-31) whether the month is
entered as a number and is between (1-12) whether the year entered is within 1995 -->
                function checkdate() {
                        year=document.forms[0].elements[6].value;
                        if (year>1995) {
                                alert("Enter proper year (till 1995)");
                                document.forms[0].elements[6].focus();
                                return(false);
                        }
                        monval=document.forms[0].elements[5].value;
                        dtval=document.forms[0].elements[4].value;
                        if ((dtval > 31) || (dtval < 1)) {
                                alert("Enter proper date");
                                document.forms[0].elements[4].focus();
                                return(false);
                        }
                        if (isNaN(monval) != true) {
                                if ((monval > 12) || (monval < 1)) {
                                        alert("Enter proper month");
                                        document.forms[0].elements[5].focus();
                                        return(false);
                                }
                        }
                        else {
                                alert("Enter the month number");
                                document.forms[0].elements[5].focus();
                                return(false);
                        }
                        return (true);
                }
        <!-- The function checklen() checks whether the length of name and email address does not
exceed 30 characters -->
                function checklen() {
                        for (i=0; i<=1; i++) {
                        val=document.forms[0].elements[i].value;
                        len=val.length;
                        if (len > 30) {
                                alert ("Value exceeds 30 characters");
                                document.forms[0].elements[i].value="";
                                document.forms[0].elements[i].focus();
                        }
                }
        }
        <!-- The function takes the user to the previous page -->
                function abort(form) {
                        history.back();
                }
        <!-- Sets the focus on the first field when the form is loaded -->
```

```
        function set(form) {
                document.forms[0].elements[0].focus();
        }
</SCRIPT></HEAD>
<BODY Background="images\Grid1.gif" onLoad="set(this.form)">
        <CENTER><IMG SRC="images\Penpals.gif" /></CENTER>
        <CENTER>
        <FORM Action="" Method="POST" onSubmit="return verify()">
                <P>Please enter your name<BR/>
                <INPUT Type="text" Name="name" Size="40" onBlur="checklen()"
                /><BR/></P>
                <P>Please enter your E-Mail address<BR/>
                <INPUT Type="Text" Name="emailid" Size="40" onBlur="checklen()"
                /><BR/></P>
                <P>Please indicate your Sex<BR/>
                <INPUT Type="Radio" Name="sex" Value="m" checked />Male
                <IMG Src="images/Shim.gif" Height="1" Width="5" />
                <INPUT Type="Radio" Name="sex" Value="f" />Female<BR/></P>
                <P>Please enter your DOB as DD/MM/YYYY<BR/>
                <INPUT Type="Text" Name="day" Size="2" />
                <IMG Src="images/Shim.gif" Height="1" Width="2" />
                <INPUT Type="Text" Name="month" Size="2" />
                <IMG Src="images/Shim.gif" Height="1" Width="2" />
                <INPUT Type="Text" Name="year" Size="4" /></P>
                <P>Tell us about your hobbies and interests in the 'Text-Box' below<BR/>
                We'll keep your Penpal info posted for <B>One Year</B> from today</P>
                <P><TEXTAREA NAME="request" ROWS="8" COLS="65"
                onFocus="checkdate()">
                </TEXTAREA></P>
                <P>Can we contact you with information about our products or services.<BR/>
                <INPUT Type="Radio" Name="moreinfo" Value="y" />
                <SPAN Style="color: blue;">Yes</SPAN><IMG Src="images\shim.gif"
                Width="20" Height="10" />
                <INPUT Type="Radio" Name="moreinfo" Value="n" Checked="True" />
                <SPAN Style="color: Red;">No, Thanks!</SPAN></P>
                <P><CENTER><INPUT Type="Submit" Value="Submit" Name="Submit" />
                <INPUT Type="Reset" Value="Reset" Name="Reset" />
                <INPUT Type="Button" Value="Abort" Name="At" onClick =
                "abort(this.form)" />
        </CENTER></P></FORM>
        <P><B>Thank You For Stopping By Our Web Site</B></P></CENTER>
</BODY>
</HTML>
```


Project Specifications For The First Project In JavaScript – Registration Form

Since the learning of Java Script has now been completed, it is time to consolidate this learning by building a Registration form page.

This Registration page is used to register visitors and provide access to a number of services provided by the Web Site. To register a simple form as show in diagram B.3 is used.

The structure of the registration page is given in the following pages along with html source code. The Java script code working within html, which provides for various client side data capture validations.

This captures two different blocks of information. The first block of information is associated with the visitors Login ID and Password that will be used when access to the sites services will be required in future. The second block of information captured, will be the services that the visitor wishes to sign up for.

The Registration page is a html document with:

- Simple visuals in the form of .gif or .jpeg files.
- Java Script embedded HTML code.

The files required to construct these pages is available on the accompanying CD-ROM for immediate use.

If required run the HTML files first in a web browser and get a look and feel of what the project could be like. Once this is done, code the web pages according to what you believe is appropriate.

Feel free to improvise and get a look and feel that satisfies you.

Each HTML file is really just a simple guideline to what the web page could look like.
Java script is added to perform simple client side validations.

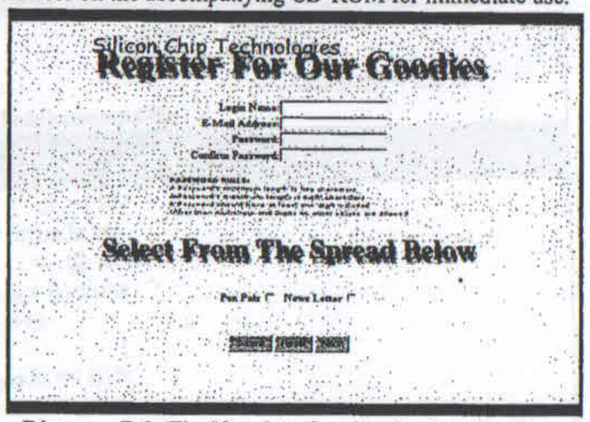

Diagram B.3: The User Interface For Registration Form

For your guidance:

- The source code is provided in the document file.

Client Side Validations

The Client side validations to be coded in Java script are as follows

- The Login name, Password, Confirm Password cannot be left empty.
- The Emailid needs to be scanned for the presence of an '@' and '.' symbol.
- The Password and the Confirm Password has to be the same.
- The length of the Name and Emailid fields cannot exceed 30 characters.

```
<HTML>
        <HEAD><TITLE>SCT'S GOODIES SIGN UP FORM</TITLE>
        <SCRIPT Language="JavaScript">
        <!-- Declaration of Variables  -->
                var valueofpass1="";
                var valueofpass2="";
                var whitespace = " \t\n\r";
```

```
<!-- Function to check whether the value in a Text Field is Null -->
        function isEmpty(s) {
                return ((s == null) || (s.length == 0))
        }
        <!-- Function to check whether the value in a Text Field is a WhiteSpace -->
        function isWhitespace (s) {
                var i;
        <!-- Is  empty? -->
                if (isEmpty(s)) return true;
<!-- Search through string's characters one by one until we find a non-whitespace character -->
                for (i=0; i < s.length; i++) {
                <!-- Check that current character isn't' whitespace.-->
                        var currchar = s.charAt(i);
                        if (whitespace.indexOf(currchar) == -1)
                                return false;
                        }
        <!-- All characters are whitespace. -->
                return true;
        }
<!-- Function to ensure that the email address is in proper format -->
        function isEmail (eadd) {
                if (isEmpty(eadd))
<!-- Is s whitespace? -->
                        if (isWhitespace(eadd)) return false;
<!-- There must be >= 1 character before @, so we start looking at character position 1 (i.e. second
character) -->
                        var i = 1;
                        var sLength = eadd.length;
<!-- look for @ -->
                        while ((i < sLength) && (eadd.charAt(i) != "@")) {
                                i++;
                        }
                        if ((i >= sLength) || (eadd.charAt(i) != "@"))
                                return false;
                        else
                                i += 2;
<!-- look for period -->
                        while ((i < sLength) && (eadd.charAt(i) != ".")) {
                                i++;
                        }
                        <!--There must be at least one character after the period-->
                        if ((i >= sLength - 1) || (eadd.charAt(i) != "."))
                                return false;
                        else
                                return true;
                        }
```



```
<!-- Function to check whether the value in Password contains Alphabets and
Characters -->
        function isCharsInBag (string, bag) {
                var i;
<!-- Search through string's characters one by one. If character is in bag, append to returnString -->
                for (i=0; i < string.length; i++) {
                        var charval = string.charAt(i);
                        if (bag.indexOf(charval) == -1)      return false;
                }
                return true;
        }
                <!-- Function to check whether the Password contains atleast one number -->
                function isNumberInPass (string, bag) {
                var i, flag;
                flag=0;
<!-- Search through string's characters one by one. If character is in bag, append to return String -->
                for (i=0; i < string.length; i++) {
                        var charval = string.charAt(i);
                        if (bag.indexOf(charval) == -1) {
                                continue;
                        }
                        else {
                                flag=1;
                                break;
                        }
                }
                if(flag == 1) {     return true;        }
                else {    return false;       }
                return false;
        }
<!-- Function to check the values entered in all the elements of the form called on the clicked event of the
submit button -->
        function verify() {
                var flag =0;
                <!-- Begining the check for Email address, Password and Confirm Password -->
                        var loginName = document.forms[0].elements[0].value;
                        var email = document.forms[0].elements[1].value;
                        var passwd = document.forms[0].elements[2].value;
                        var confPasswd = document.forms[0].elements[3].value;

                        <!-- Check to see if the Login name field is left blank -->
                        if (loginName == "") {
                                alert("Please fill in the login name");
                                document.forms[0].elements[0].focus( );
                                return false;
                        }
```

```
<!-- Check to see if the Email Address field is left blank -->
                else if (email == "") {
                        alert("Please fill in the email address");
                        document.forms[0].elements[1].focus( );
                        return false;
                }
<!-- Check to see if the Email Address filled is valid or not-->
                else if (!isEmail(email)) {
                        alert("Please enter the Email address in the proper Format");
                        document.forms[0].elements[1].focus( );
                        return false;
                }

<!-- Check to see if the password field is left blank -->
                else if ( passwd == "" ) {
                        alert("Please fill in the Password");
                        document.forms[0].elements[2].focus( );
                        return false;
                }
<!-- Check to see if the length of the password entered is less than 5 -->
                else if ( passwd.length < 5 ) {
                        alert( "Password must be 5 or more characters." );
                        document.forms[0].elements[2].focus( );
                        return false;
                }
<!-- Check to see if the length of the password entered is more than 8 -->
                else if (passwd.length > 8 ) {
                        alert( "Password must be 8 or less characters." );
                        document.forms[0].elements[2].focus( );
                        return false;
                }

<!-- Check to see if the password field is contain other than alphabate or number -->
                else if (!isCharsInBag(passwd,
                        "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRS
                        TUVWXYZ0123456789")) {
                        alert( "Password must only contain alphabets and number");
                        document.forms[0].elements[2].focus( );
                        return false;
                }

<!-- Check to see if the length of the password entered should have atleast one numeric character -->
                else if ( !isNumberInPass( passwd, "0123456789")) {
                        alert( "Password must have atleast one number.");
                        document.forms[0].elements[2].focus( );
                        return false;
                }
```



```
<!-- Check to see if the Confirm password is entered or not -->
                else if ( confPasswd == "" ) {
                        alert("Please fill in the Confirm Password");
                        document.forms[0].elements[3].focus( );
                        return false;
                }

<!-- Check to see if the password and Confirm password is not match or not -->
                else if (document.forms[0].elements[2].value
                != document.forms[0].elements[3].value) {
                  alert( "Your passwords do not match. Please retype and try again.");
                        return false;
                }

                <!-- Check to see if atleast one is checkbox checked or not-->
                for (j=4; j<=5; j++) {
                        if(document.forms[0].elements[j].checked) {
                                break;
                }
                else if (j>=5) {
                        alert("Atleast Check on One of Our Services");
                        document.forms[0].elements[j].focus();
                        return (false);
                        }
                }
                return(true);
        }
<!-- Function called on the Clicked event of the Abort button -->
        function Abort() {
                history.back();
        }
</SCRIPT></HEAD>
<BODY Style="background-image: url(images/Grid1.gif);text-align:center;">
        <IMG Src="images/RegGoodies.gif" />
        <FORM Action="" Method="post" onSubmit="return verify()">
                <TABLE Align="center" Border="0" CellPadding="0" CellSpacing="0"
                Size="800">
                <TBODY><TR>
                        <TD Align="right"><B>Login Name:</B></TD>
                        <TD><INPUT Name="LoginName" /></TD>
                </TR><TR>
                        <TD Align="right"><B>E-Mail Address:</B></TD>
                        <TD><INPUT Name="Email:" /></TD>
                </TR><TR>
                        <TD Align="right"><B>Password:</B></TD>
                        <TD><INPUT Name="Passwd1" Type="password" /></TD>
                </TR><TR>
                        <TD Align="right"><B>Confirm Password:</B></TD>
```

```
<TD><INPUT Name="Passwd2" Type="password" /></TD>
</TR></TBODY></TABLE>
<P>
<TABLE Align="center" Border="0" CellPadding="0" CellSpacing="0"
        Size="800" Style="color:red;font-family:verdana;font-size:10px;">
<TBODY><TR><TD>
        <B>PASSWORD RULES:</B></FONT>
</TD></TR><TR><TD>
        A Password's minimum length is five characters
</TD></TR><TR><TD>
        A Password's maximum length is eight characters
</TD></TR><TR> <TD>
        A Password should have at least one digit included
</TD></TR><TR> <TD>
        Other than Alphabets and Digits no other values are allowed
</TD></TR></TBODY></TABLE>
<P>
<IMG Height="40" Src="images\Shim.gif" Width="1" />
        <IMG Src="images/SeleGoodies.gif" />
<IMG Height="30" Src="images\Shim.gif" Width="1" />
</P>
<TABLE Align="center" Border="0" CellPadding="0"
        CellSpacing="0" Size="800">
<TBODY><TR>
        <TD><IMG Height=5 Src="images/Shim.gif" width="10" /></TD>
        <TD><B>Pen Pals</B>
        <IMG Height="1" src="images/Shim.gif" Width="3" />
                <INPUT type="checkbox" Name="PenPals"
                value="y" /></TD>
        <TD><IMG Height=5 Src="images/Shim.gif" width="10" /></TD>
        <TD><B>News Letter</B>
                <IMG Height="1" src="images/Shim.gif" Width="3" />
                <INPUT type="checkbox" Name="NewsLtr" value="y" />
        </TD>
        <TD><IMG Height="5" src="images/Shim.gif" Width="10" /></TD>
</TR></TBODY></TABLE>
<IMG Height="30" src="images/Shim.gif" Width="1" />
<P>
        <INPUT Type="submit" Value="Submit" />
        <INPUT Type="reset" Value="Reset" />
        <INPUT onClick="Abort()" Type="button" Value="Abort" />
</P>
</FORM>
</BODY>
</HTML>
```

ANSWERS TO SELF REVIEW QUESTIONS

8. INTRODUCTION TO JAVASCRIPT

FILL IN THE BLANKS

1. Form
2. Netscape
3. <SCRIPT></SCRIPT>
4. <FORM></FORM>
5. Variables
6. ($) character
7. implicitly defined, literal
8. string
9. Functions
10. Arrays
11. dense
12. Objects
13. %(modulus)

TRUE OR FALSE

14. False
15. True
16. False
17. True
18. False
19. True
20. True
21. True
22. True

9. OBJECTS IN JAVASCRIPT

FILL IN THE BLANKS

1. Document Object Model
2. JavaScript Assisted Style Sheets
3. Method
4. Interactive, Non Interactive
5. Navigator

TRUE OR FALSE

6. False
7. False
8. True

10. FORMS OF A WEB SITE

FILL IN THE BLANKS

1. <FORM> and </FORM>
2. Method and Action
3. Get or Post
4. Action
5. <INPUT>
6. Submit
7. <MULTIPLE>
8. Length
9. Math

TRUE OR FALSE

10. True
11. False
12. True
13. False
14. True
15. True

11. COOKIES

FILL IN THE BLANKS

1. Cookie
2. HTTP
3. set-cookie
4. Request header
5. SECURE

TRUE OR FALSE

6. False
7. True

SOLUTIONS TO HANDS ON EXERCISES

8. INTRODUCTION TO JAVASCRIPT

1. **Creating a JavaScript code block using arrays to generate the current date in words, (in the format: Saturday, January 01, 2000).**

```
<HTML>
    <HEAD><TITLE> Date validations </TITLE><SCRIPT>
        var monthNames = new Array(12);
        monthNames[0]= "January";
        monthNames[1]= "February";
        monthNames[2]= "March";
        monthNames[3]= "April";
        monthNames[4]= "May";
        monthNames[5]= "June";
        monthNames[6]= "July";
        monthNames[7]= "August";
        monthNames[8]= "September";
        monthNames[9]= "October";
        monthNames[10]= "November";
        monthNames[11]= "December";
        var dayNames = new Array(7) ;
        dayNames[0]= "Sunday";
        dayNames[1]= "Monday";
        dayNames[2]= "Tuesday";
        dayNames[3]= "Wednesday";
        dayNames[4]= "Thursday";
        dayNames[5]= "Friday";
        dayNames[6]= "Saturday";
        function customDateString (m_date) {
            var daywords = dayNames[m_date.getDay()];
            var theday = m_date.getDate();
            var themonth = monthNames[m_date.getMonth()];
            var theyear = m_date.getYear();
            return daywords + ",  " + themonth + " " + theday + ", " + theyear;
        }
    </SCRIPT></HEAD>
    <BODY><H1>WELCOME!</H1>
        <SCRIPT>document.write(customDateString( new Date()))</SCRIPT>
    </BODY>
</HTML>
```


2. **Creating a JavaScript code block, which checks the contents entered in a form's Text element. If the text entered is in the lower case, convert to upper case.**

```
<HTML>
    <HEAD><SCRIPT  Language="JavaScript">
        function checkData(column_data) {
            if ( column_data != "" &&  column_data.value != column_data.value.toUpperCase() ) {
                column_data.value = column_data.value.toUpperCase()
            }
        }
    </SCRIPT></HEAD>
    <BODY><FORM><B>Field 1:</B> 
        <INPUT TYPE="text" NAME="collector" SIZE=10 onChange="checkData(this)"><BR>
        <B>Field 2:</B> <INPUT TYPE="text" NAME="dummy" SIZE=10>
    </FORM></BODY>
</HTML>
```

3. **Creating a JavaScript code block, which validates a username and password against hard coded values.**

```
<HTML>
    <HEAD><TITLE>Password Validation</TITLE>
    <SCRIPT language="JavaScript">
    <!--
        var userpassword = new Array(4);
        userpassword[0]= "Allwyn";
        userpassword[1]= "allwyn";
        userpassword[2]= "Rohan";
        userpassword[3]= "rohan";
        function checkOut() {
            var flag =0;
            var flag1 =0;
            var i =0;
            var j =0;
            for (x=0; x<document.survey.elements.length; x++) {
                if (document.survey.elements[x].value == "") {
                    alert("You forgot one of the required fields. Please try again");
                    return;
                }
            }
            var user= document.survey.elements[0].value;
            var password = document.survey.elements[1].value;
            while(i<=3) {
                if(userpassword[i] == user) {
                    j=i;
                    j++;
                    if(userpassword[j] == password) {
                        flag=1;
                        break;
                    }
                }
```

```
            i+=2;
        }
        if(flag == 0) {
            alert("Please enter a valid user name and password");
            return;
        }
        else {
            alert("Welcome !!\n" + document.forms[0].Username.value);
        }
        return;
    }
//-->
</SCRIPT></HEAD>
<BODY><FORM Action="" Method="POST" onSubmit="return checkOut(this.form)"
                Name="survey" >
    <INPUT Type="TEXT" Name="Username" Size="15" MaxLength="15">User Name<BR>
    <INPUT Type="PASSWORD" Name="Pasword" size="15">Password<BR>
    <INPUT Type="SUBMIT" Value="Submit">
    <INPUT Type="RESET" Value="Start Over">
</FORM></BODY>
</HTML>
```

9. OBJECTS IN JAVASCRIPT

1. **Creating a Web page using two image files, which switch between one another as the mouse pointer moves over the images.**

```
<HTML>
    <HEAD><TITLE>CHANGING IMAGES....</TITLE><SCRIPT>
        if(document.images)  rollover="yes";
        else       rollover="no";
        if(rollover=="yes") {
            img1on = new Image();
            img1off = new Image();
            img1on.src="Images/man1.gif";
            img1off.src="Images/man2.gif";
        }
        function imageSwitchOn(imagename) {
            if(rollover=="yes") {
                imageOn=eval(imagename + "on.src");
                document [imagename].src = imageOn;
            }
        }
        function imageSwitchOff(imagename) {
            if (rollover=="yes") {
                imageOff=eval(imagename + "off.src");
                document [imagename].src = imageOff;
            }
        }
    </SCRIPT></HEAD>
```



```
    <BODY><CENTER>
        <H1>IMAGES.....</H1><BR>
        <H3><I>place your mouse pointer on the picture</I></H3>
        <A onMouseOver="imageSwitchOn('img1')" onMouseOut="imageSwitchOff('img1')">
            <IMG Alt="space" Height=200 Name="img1" Src="Images/man2.gif" Width=200></A>
    </CENTER></BODY>
</HTML>
```

10. FORMS OF A WEB SITE

1. **Creating a Web page, which accepts user information and user comments on the web site to check if all the Text fields have being entered with data else display an alert.**

```
<HTML>
    <HEAD><TITLE>Elements of a Form</TITLE>
    <SCRIPT language="JavaScript">
    <!--
        function checkOut() {
            for (x=0; x<document.survey.elements.length; x++) {
                if (document.survey.elements[x].value == "") {
                    alert("Sorry, you forgot one of the required fields. Please try again.");
                    break;
                }
            }
            return false;
        }
    //-->
    </SCRIPT></HEAD>
    <BODY bgcolor="lightyellow"><H2><I>INFONET SERVICES</I></H2>
        <FORM NAME="survey" onSubmit="return checkOut(this.form)">
            <INPUT Name="firstname" Size="30" Type="TEXT" MaxLength="30">
            <B>First Name</B><BR><BR>
            <INPUT Type="TEXT" Name="lastname" Size="30"><B>Last  Name</B><BR><BR>
            <INPUT Type="TEXT" Name="emailaddr" Size="30"><B>E-mail Address</B><BR><BR>
            <INPUT Type="TEXT" Name="address" Size="30"><B>Address</B><BR><BR>
            <INPUT TYPE="TEXT" NAME="city" SIZE="30"><B>City</B><BR><BR>
            <INPUT NAME="state" SIZE="6"><b>State</b>
            <INPUT NAME="zip" SIZE="10"><b>Postal Code</b>
            <INPUT NAME="country" SIZE="15"><b>Country</b>
            <P><B><I>Please choose the most appropriate statement</I></B><BR>
            <INPUT Type="RADIO" Name="buying" Value="regular">
                <B>I regularly purchase items online</B><BR>
            <INPUT Type="RADIO" Name="buying" Value="sometimes">
                <B>I have on occasion purchased items online</B><BR>
            <INPUT Type="RADIO" Name="buying" Value="might" CHECKED>
                <B>I have not purchased anything online, but I would consider it</B><BR>
            <INPUT Type="RADIO" Name="buying" Value="willnot">
                <B>I prefer to shop in real stores</B>
            <P><B><I>I'm interested in (choose all that apply)</I></B><BR>
            <INPUT TYPE="CHECKBOX" NAME="hiking" VALUE="hiking"><B>Hiking</B><BR>
```

```
            <INPUT Type="CHECKBOX" Name="mbiking" Value="mbiking">
                <B>Mountain Biking</B><BR>
            <INPUT Type="CHECKBOX" Name="camping" Value="camping">
                <B>Camping</B><BR>
            <INPUT Type="CHECKBOX" NAME="rock" Value="rock">
                <B>Rock Climbing</B><BR>
            <INPUT Type="CHECKBOX" Name="4wd" Value="4wd">
                <B>Off-Road 4WD</B><BR>
            <INPUT Type="CHECKBOX" Name="ccskiing" Value="ccskiing">
                <B>Cross-country Skiing</B>
            <P><B><I>I learned about this site from</I></B><BR>
            <SELECT NAME="referral">
                <OPTION VALUE="print" SELECTED>Print Ads
                <OPTION VALUE="visit"> In-Store Visit
                <OPTION VALUE="rec"> Friend's Recommendation
                <OPTION VALUE="internet"> Sources on the Internet
                <OPTION VALUE="other"> Other
            </SELECT>
            <P><H4>Comments</H4><BR>
            <TEXTAREA Name="comments" Cols="40" Rows="5">
                Please type any comments here</TEXTAREA><BR>
            <INPUT Type="SUBMIT" Value="Submit">
            <INPUT Type="RESET" Value="Start Over">
        </FORM>
    </BODY>
</HTML>
```

SECTION - III: Dynamic Hyper Text Markup Language

12. DYNAMIC HTML

First Impression	- Did the initial page grab attention?
Interface Design	- Is the menu interface interactive enough and visually interesting?
Corporate Mildew	- Is the site trapped in a web of corporate look, feel and canned marketing speak?
Coriolis Effect	- Does the site generate enough currents of interest based on design and content for the user to comeback?

The above points emphasize the requirements of a good web site.

DHTML is a new and emerging technology that has evolved to meet the increasing demand for eye-catching and mind-catching web sites.

DHTML combines HTML with Cascading Style Sheets (CSSs) and Scripting Languages. HTML specifies a web page's elements like table, frame, paragraph, bulleted list, etc. Cascading Style Sheets can be used to determine an element's size, color, position and a number of other features. Scripting Languages (JavaScript and VBScript) can be used to manipulate the web page's elements so that styles assigned to them can change in response to a user's input.

CASCADING STYLE SHEETS

Style Sheets are powerful mechanism for adding styles (e.g. fonts, colors, spacing) to Web documents. They enforce standards and uniformity throughout a web site and provide numerous attributes to create dynamic effects. With Style Sheets, text and image formatting properties can be predefined in a single list. HTML elements on a web page can then be bound to the style sheet. The advantages of a Style Sheet includes the ability to make global changes to all documents from a single location. Style Sheets are said to Cascade when they combine to specify the appearance of a page.

The Style assignment process is accomplished with the **<STYLE>...</STYLE>** tags. The syntax for making the assignment is simple. Between the <STYLE>...</STYLE> HTML tags, specific style attributes are listed. The <STYLE>...</STYLE> tags are written within the <HEAD>...</HEAD> tags.

Syntax:

```
<STYLE Type="text/css">
    tag {attribute:value; attribute:value ... }
    ...
</STYLE>
```

Note

In the <STYLE> tag, the expression "Type = text/css" indicates that the style sheet conforms to CSS syntax