UNIT-II

Overview

Client-side scripting generally refers to the class of computer programs on the web that are executed client-side, by the user's web browser, instead of server-side (on the web server) This type of computer programming is an important part of the Dynamic HTML (DHTML) concept, enabling web pages to be scripted; that is, to have different and changing content depending on user input, environmental conditions (such as the time of day), or other variables.

Web authors write client-side scripts in languages such as JavaScript (Client-side JavaScript) and VBScript.

- JavaScript is considered to be one of the most famous scripting languages of all time. Please don't confuse JavaScript with Java; we shall discuss the difference amongst them later. JavaScript, by definition, is a Scripting Language of the World Wide Web. The main usage of JavaScript is to add various Web functionalities, Web form validations, browser detections, creation of cookies and so on. JavaScript, along with VBScript, is one of the most popular scripting languages and that is why it is supported by almost all web browsers available today like Firefox, Opera or the most famous Internet Explorer.

- JavaScript was originally designed in order to add interactivity to the Web Pages or HTML pages. Previously, the interactivity was achieved through the use of forms made in JavaScript but later JavaScript was used to add more dynamic and interactive features. JavaScript, as mentioned before, is a scripting language which falls under the category of light-weight programming languages. Using JavaScript is quite easy as the web programmer can easily embed the JavaScript code into the HTML pages. Another interesting fact about JavaScript is that it doesn't require the user to purchase any license. Another advantage of using JavaScript is that it is already interpreted, which is the reason it is also called as an interpreted language. Interpreted language means that the JavaScript execute simply without any preliminary compilation; which is another reason why it is called as light-weight programming language.

Objectives

- **Explain about client side scripting**
- **What is java script**
- **Why scripting languages are used**
- **Differentiate between javascript and vbscript**

Introduction

Client-side scripts are often embedded within an HTML document (hence known as an "embedded script"), but they may also be contained in a separate file, which is referenced by the document (or documents) that use it (hence known as an "external script"). Upon request, the necessary files are sent to the user's computer by the web server (or servers) on which they reside. The user's web browser executes the script, then displays the document, including any visible output from the script. Client-side scripts may also contain instructions for the browser to follow if the user interacts with the document in a certain way, e.g., clicks a certain button. These instructions can be followed without further communication with the server, though they may require such communication.

By viewing the file that contains the script, users may be able to see its source code. Many web authors learn how to write client-side scripts partly by examining the source code for other authors' scripts.

In contrast, server-side scripts, written in languages such as Perl, PHP, and server-side VBScript, are executed by the web server when the user requests a document. They produce output in a format understandable by web browsers (usually HTML), which is then sent to the user's computer. The user cannot see the script's source code (unless the author publishes the code separately), and may not even be aware that a script was executed. The documents produced by server-side scripts may, of course, contain client-side scripts.

Client-side scripts have greater access to the information and functions available on the user's browser, whereas server-side scripts have greater access to the information and functions available on the server. Server-side scripts require that their language's interpreter be installed on the server, and produce the same output regardless of the client's browser, operating system, or other system details. Client-side scripts do not require additional software on the server (making them popular with authors who lack administrative access to their servers); however, they do require that the user's web browser understands the scripting language in which they are written.

It is therefore impractical for an author to write scripts in a language that is not supported by the web browsers used by a majority of his or her audience.

Due to security restrictions, client-side scripts may not be allowed to access the user's computer beyond the web browser application. Techniques like ActiveX controls can be used to sidestep this restriction.

Unfortunately, even languages that are supported by a wide variety of browsers may not be implemented in precisely the same way across all browsers and operating systems. Authors are well-advised to review the behavior of their client-side scripts on a variety of platforms before they put them into use.

Client Side Image Maps

How to add an image map to your page
Image maps aren't as bad as they seem, at least if you use a client side image map using HTML rather than a CGI program. I may add a section on using a CGI-driven image map later on, but for now, lets look at one that just uses HTML and an image!

To begin, let's look at a simple image map I created. If you click the left half of the image, you will be taken to a tables tutorial. If you click the right side, you'll get a tutorial on frames. Both the links are on one single image, so we just call it an image map. Try it out:

So, how do we create one of these? First, you need to have an image you want to use. Just pick one you like, or make your own. Next, you need to know the width and height of the image you are using. If you are not sure, open the image in your image editing program. You should have some option that will tell you the width and height of the image.

Now you need to put the image on the page. To do this, you use the image tag, but with a new attribute: usemap.

The **usemap="#mymap"** command tells the browser to use a map on the page, which is named "mymap". Notice how it uses the "#" symbol in front of the map name. Also notice that we defined the width and height of the image. This needs to be done so we can use coordinates later on when we define the map. Speaking of that, lets see how to define

the map. For this map, we would place the following code somewhere on the page. I just put it right after the <img> tag to make it easy to find.

```
<map name="mymap" id="mymap">
<area shape="rect" coords="0,0,99,40" href="table1.htm" alt="Tables" />
<area shape="rect" coords="100,0,200,40" href="frame1.htm" alt="Frames" />
<area shape="default" href="http://www.pageresource.com" alt="Home" />
</map>
```

Now you can see where the usemap="#mymap" from the <img> tag comes from. The name of the map is "mymap". Now, let's look at what all of this means:

- **<map name="mymap" id="mymap">**
  This defines your image map section, and gives the map a name. This map is named "mymap" In XHTML, the id attribute is required rather than name. If you are using XHTML transitional, both the name and id can be used.
- **<area shape="rect" coords="0,0,99,40" href="table1.htm" alt="Tables" />**
  The area tag defines an area of the image that will be used as a link. The shape attribute tells the browser what shape the area will be. To keep it simple, I only used "rect", which stands for rectangle. The coords attribute is where we define the edges of each area. Since it is a rectangle, we will use two sets of coordinates. The first set defines where to start the rectangle, where the top-left edge of the rectangle will be. Since this rectangle starts at the top-left edge of the image, the coordinates are (0 pixels, 0 pixels). The second two numbers define where to end the rectangle. This will be the lower-right edge of the rectangle. Remember that the total image size was 200x40. We want the lower-right edge of this rectangle to be halfway accross the image and at the bottom of the image. Going accross, half of 200 is 100, but we use 99 here because 100 can only be used once. We will use it in the second rectangle here. Of course, 40 pixels takes us to the bottom of the image. So the lower-right corner of this rectangle will be 99 pixels accross the image, and 40 pixels (all the way) down the image. And now the easy part: The href attribute is used to tell the browser where to go when someone clicks someplace on that rectangle. Put the URL of the page you want to go to in there, and the first rectangle is set up! The alt attribute allows you to define alternate text for that area.
- **<area shape="rect" coords="100,0,200,40" href="frame1.htm" alt="Frames" />**
  Basically the same as the previous area tag, but it is for our second rectangle. We start where the other one left off, but back at the top of the image. Since the right edge of the last rectangle was at 99 pixels accross, we start this one at 100 pixels accross. And since this will be the upper-left of the second rectangle, we start it at 0 pixels down the image (the top!). We end this rectangle where the image ends, so the lower-right coordinate here is pretty nice- (200, 40), the size of the image!
- **<area shape="default" href="http://www.pageresource.com" alt="Home">**
  The default is not really a new shape, it just covers anything that may have been

left out. We didn't leave out anything in this map, but if we had, this would be the URL someone would go to if they clicked on any area we did not define earlier.

- **</map>**
  This ends the map section!

Now, you can use other shapes besides rectangles, but those are alot tougher to code by hand. I would suggest using a shareware image map creation program if you use other shapes in your map.

So, if you need a simple image map separated into rectangular sections, just do what we did above. You can have as many rectangular areas as you want, just add the extra area tags with coordiantes and urls. Sounds like fun, don't you think?

JavaScript is the most popular scripting language on the internet, and works in all major browsers, such as Internet Explorer, Firefox, Chrome, Opera, and Safari.

What You Should Already Know

Before you continue you should have a basic understanding of the following:

- HTML / XHTML

If you want to study these subjects first, find the tutorials on our Home page.

What is JavaScript?

- JavaScript was designed to add interactivity to HTML pages
- JavaScript is a scripting language
- A scripting language is a lightweight programming language
- JavaScript is usually embedded directly into HTML pages
- JavaScript is an interpreted language (means that scripts execute without preliminary compilation)
- Everyone can use JavaScript without purchasing a license

Are Java and JavaScript the same?

NO!

Java and JavaScript are two completely different languages in both concept and design!

Java (developed by Sun Microsystems) is a powerful and much more complex programming language - in the same category as C and C++.

What can a JavaScript do?

- **JavaScript gives HTML designers a programming tool -** HTML authors are normally not programmers, but JavaScript is a scripting language with a very simple syntax! Almost anyone can put small "snippets" of code into their HTML pages
- **JavaScript can put dynamic text into an HTML page -** A JavaScript statement like this: document.write("<h1>" + name + "</h1>") can write a variable text into an HTML page
- **JavaScript can react to events -** A JavaScript can be set to execute when something happens, like when a page has finished loading or when a user clicks on an HTML element
- **JavaScript can read and write HTML elements -** A JavaScript can read and change the content of an HTML element
- **JavaScript can be used to validate data -** A JavaScript can be used to validate form data before it is submitted to a server. This saves the server from extra processing
- **JavaScript can be used to detect the visitor's browser** - A JavaScript can be used to detect the visitor's browser, and - depending on the browser - load another page specifically designed for that browser
- **JavaScript can be used to create cookies** - A JavaScript can be used to store and retrieve information on the visitor's computer

---

The Real Name is ECMAScript

JavaScript's official name is ECMAScript.

ECMAScript is developed and maintained by the ECMA organization.

ECMA-262 is the official JavaScript standard.

The language was invented by Brendan Eich at Netscape (with Navigator 2.0), and has appeared in all Netscape and Microsoft browsers since 1996.

The development of ECMA-262 started in 1996, and the first edition of was adopted by the ECMA General Assembly in June 1997.

The standard was approved as an international ISO (ISO/IEC 16262) standard in 1998.

The development of the standard is still in progress.

The HTML <script> tag is used to insert a JavaScript into an HTML page.

JavaScript

JavaScript is a scripting language used to enable programmatic access to objects within other applications. It is primarily used in the form of client-side JavaScript for the development of dynamic websites. JavaScript is a dialect of the ECMAScript standard and is characterized as a dynamic, weakly typed, prototype-based language with first-class functions. JavaScript was influenced by many languages and was designed to look like Java, but to be easier for non-programmers to work with.

JavaScript-specific

JavaScript is officially managed by Mozilla, and new language features are added periodically. However, only some non-Mozilla "JavaScript" engines support these new features:

- conditional `catch` clauses
- property getter and setter functions
- iterator protocol adopted from Python
- shallow generators/coroutines also adopted from Python
- array comprehensions and generator expressions also adopted from Python
- proper block scope via new `let` keyword
- array and object destructuring (limited form of pattern matching)
- concise function expressions (`function(args) expr`)
- E4X

Use in web pages

The primary use of JavaScript is to write functions that are embedded in or included from HTML pages and interact with the Document Object Model (DOM) of the page. Some simple examples of this usage are:

- Opening or popping up a new window with programmatic control over the size, position, and attributes of the new window (i.e. whether the menus, toolbars, etc. are visible).
- Validation of web form input values to make sure that they will be accepted before they are submitted to the server.
- Changing images as the mouse cursor moves over them: This effect is often used to draw the user's attention to important links displayed as graphical elements.

Because JavaScript code can run locally in a user's browser (rather than on a remote server) it can respond to user actions quickly, making an application feel more responsive. Furthermore, JavaScript code can detect user actions which HTML alone cannot, such as individual keystrokes. Applications such as Gmail take advantage of this: much of the user-interface logic is written in JavaScript, and JavaScript dispatches requests for information (such as the content of an e-mail message) to the server. The wider trend of Ajax programming similarly exploits this strength.

A JavaScript engine (also known as JavaScript interpreter or JavaScript implementation) is an interpreter that interprets JavaScript source code and executes the script accordingly. The first ever JavaScript engine was created by Brendan Eich at Netscape Communications Corporation, for the Netscape Navigator web browser. The engine, code-named SpiderMonkey, is implemented in C. It has since been updated (in JavaScript 1.5) to conform to ECMA-262 Edition 3. The Rhino engine, created primarily by Norris Boyd (formerly of Netscape; now at Google) is a JavaScript implementation in Java. Rhino, like SpiderMonkey, is ECMA-262 Edition 3 compliant.

JavaScript and Java

A common misconception is that JavaScript is similar or closely related to Java; this is not so. Both have a C-like syntax, are object-oriented, are typically sandboxed and are widely used in client-side Web applications, but the similarities end there. Java has static typing; JavaScript's typing is dynamic (meaning a variable can hold an object of any type and cannot be restricted). Java is loaded from compiled bytecode; JavaScript is loaded as human-readable code. C is their last common ancestor language.

Nonetheless, JavaScript was designed with Java's syntax and standard library in mind. In particular, all Java keywords are reserved in JavaScript, JavaScript's standard library follows Java's naming conventions, and JavaScript's Math and Date classes are based on those from Java 1.0.

Conditional statements are used to perform different actions based on different conditions.

Conditional Statements

Very often when you write code, you want to perform different actions for different decisions. You can use conditional statements in your code to do this.

In JavaScript we have the following conditional statements:

- **if statement** - use this statement to execute some code only if a specified condition is true
- **if...else statement** - use this statement to execute some code if the condition is true and another code if the condition is false

- **if...else if....else statement** - use this statement to select one of many blocks of code to be executed
- **switch statement** - use this statement to select one of many blocks of code to be executed

If Statement

Use the if statement to execute some code only if a specified condition is true.

Syntax

```
if (condition)
  {
  code to be executed if condition is true
  }
```

Example

```
<script type="text/javascript">
//Write a "Good morning" greeting if
//the time is less than 10

var d=new Date();
var time=d.getHours();

if (time<10)
  {
  document.write("<b>Good morning</b>");
  }
</script>
```

If...else Statement

Use the if....else statement to execute some code if a condition is true and another code if the condition is not true.

Syntax

```
if (condition)
  {
  code to be executed if condition is true
  }
else
```

```
  {
  code to be executed if condition is not true
  }
```

Example

```
<script type="text/javascript">
//If the time is less than 10, you will get a "Good morning" greeting.
//Otherwise you will get a "Good day" greeting.

var d = new Date();
var time = d.getHours();

if (time < 10)
  {
  document.write("Good morning!");
  }
else
  {
  document.write("Good day!");
  }
</script>
```

If...else if...else Statement

Use the if....else if...else statement to select one of several blocks of code to be executed.

Syntax

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

Example

```
<script type="text/javascript">
var d = new Date()
var time = d.getHours()
```

```
if (time<10)
  {
  document.write("<b>Good morning</b>");
  }
else if (time>10 && time<16)
  {
  document.write("<b>Good day</b>");
  }
else
  {
  document.write("<b>Hello World!</b>");
  }
</script>
```

The JavaScript Switch Statement

Use the switch statement to select one of many blocks of code to be executed.

Syntax

```
switch(n)
{
case 1:
  execute code block 1
  break;
case 2:
  execute code block 2
  break;
default:
  code to be executed if n is different from case 1 and 2
}
```

This is how it works: First we have a single expression n (most often a variable), that is evaluated once. The value of the expression is then compared with the values for each case in the structure. If there is a match, the block of code associated with that case is executed. Use **break** to prevent the code from running into the next case automatically.

Example

```
<script type="text/javascript">
//You will receive a different greeting based
//on what day it is. Note that Sunday=0,
//Monday=1, Tuesday=2, etc.

var d=new Date();
```

```
theDay=d.getDay();
switch (theDay)
{
case 5:
  document.write("Finally Friday");
  break;
case 6:
  document.write("Super Saturday");
  break;
case 0:
  document.write("Sleepy Sunday");
  break;
default:
  document.write("I'm looking forward to this weekend!");
}
</script>
```

Alert Box

An alert box is often used if you want to make sure information comes through to the user.

When an alert box pops up, the user will have to click "OK" to proceed.

Syntax

```
alert("sometext");
```

Example

```
<html>
<head>
<script type="text/javascript">
function show_alert()
{
alert("I am an alert box!");
}
</script>
</head>
<body>

<input type="button" onclick="show_alert()" value="Show alert box" />

</body>
</html>
```

Confirm Box

A confirm box is often used if you want the user to verify or accept something.

When a confirm box pops up, the user will have to click either "OK" or "Cancel" to proceed.

If the user clicks "OK", the box returns true. If the user clicks "Cancel", the box returns false.

Syntax

confirm("sometext");

Example

```
<html>
<head>
<script type="text/javascript">
function show_confirm()
{
var r=confirm("Press a button");
if (r==true)
  {
  document.write("You pressed OK!");
  }
else
  {
  document.write("You pressed Cancel!");
  }
}
</script>
</head>
<body>

<input type="button" onclick="show_confirm()" value="Show confirm box" />

</body>
</html>
```

Prompt Box

A prompt box is often used if you want the user to input a value before entering a page.

When a prompt box pops up, the user will have to click either "OK" or "Cancel" to proceed after entering an input value.

If the user clicks "OK" the box returns the input value. If the user clicks "Cancel" the box returns null.

Syntax

prompt("sometext","defaultvalue");

Example

```
<html>
<head>
<script type="text/javascript">
function show_prompt()
{
var name=prompt("Please enter your name","Harry Potter");
if (name!=null && name!="")
  {
  document.write("Hello " + name + "! How are you today?");
  }
}
</script>
</head>
<body>

<input type="button" onclick="show_prompt()" value="Show prompt box" />

</body>
</html>
```

JavaScript Functions

To keep the browser from executing a script when the page loads, you can put your script into a function.

A function contains code that will be executed by an event or by a call to the function.

You may call a function from anywhere within a page (or even from other pages if the function is embedded in an external .js file).

Functions can be defined both in the <head> and in the <body> section of a document. However, to assure that a function is read/loaded by the browser before it is called, it could be wise to put functions in the <head> section.

How to Define a Function

Syntax

```
function functionname(var1,var2,...,varX)
{
some code
}
```

The parameters var1, var2, etc. are variables or values passed into the function. The { and the } defines the start and end of the function.

Note: A function with no parameters must include the parentheses () after the function name.

Note: Do not forget about the importance of capitals in JavaScript! The word function must be written in lowercase letters, otherwise a JavaScript error occurs! Also note that you must call a function with the exact same capitals as in the function name.

Example

```
<html>
<head>
<script type="text/javascript">
function displaymessage()
{
alert("Hello World!");
}
</script>
</head>

<body>
<form>
<input type="button" value="Click me!" onclick="displaymessage()" />
</form>
</body>
</html>
```

If the line: alert("Hello world!!") in the example above had not been put within a function, it would have been executed as soon as the line was loaded. Now, the script is not executed before a user hits the input button. The function displaymessage() will be executed if the input button is clicked.

The return Statement

The return statement is used to specify the value that is returned from the function.

So, functions that are going to return a value must use the return statement.

The example below returns the product of two numbers (a and b):

Example

```
<html>
<head>
<script type="text/javascript">
function product (a,b)
{
return a*b;
}
</script>
</head>

<body>
<script type="text/javascript">
document.write(product(4,3));
</script>

</body>
</html>
```

Loops execute a block of code a specified number of times, or while a specified condition is true.

JavaScript Loops

Often when you write code, you want the same block of code to run over and over again in a row. Instead of adding several almost equal lines in a script we can use loops to perform a task like this.

In JavaScript, there are two different kinds of loops:

- **for** - loops through a block of code a specified number of times
- **while** - loops through a block of code while a specified condition is true

The for Loop

The for loop is used when you know in advance how many times the script should run.

Syntax

```
for (var=startvalue;var<=endvalue;var=var+increment)
{
code to be executed
}
```

Example

The example below defines a loop that starts with i=0. The loop will continue to run as long as **i** is less than, or equal to 5. **i** will increase by 1 each time the loop runs.

```
<html>
<body>
<script type="text/javascript">
var i=0;
for (i=0;i<=5;i++)
{
document.write("The number is " + i);
document.write("<br />");
}
</script>
</body>
</html>
```

The while Loop

The while loop loops through a block of code while a specified condition is true.

Syntax

```
while (var<=endvalue)
  {
  code to be executed
  }
```

Note: The <= could be any comparing statement.

Example

The example below defines a loop that starts with i=0. The loop will continue to run as long as **i** is less than, or equal to 5. **i** will increase by 1 each time the loop runs:

```
<html>
<body>
<script type="text/javascript">
var i=0;
while (i<=5)
  {
  document.write("The number is " + i);
  document.write("<br />");
  i++;
  }
</script>
</body>
</html>
```

The do...while Loop

The do...while loop is a variant of the while loop. This loop will execute the block of code ONCE, and then it will repeat the loop as long as the specified condition is true.

Syntax

```
do
  {
  code to be executed
  }
while (var<=endvalue);
```

Example

The example below uses a do...while loop. The do...while loop will always be executed at least once, even if the condition is false, because the statements are executed before the condition is tested:

```
<html>
<body>
<script type="text/javascript">
var i=0;
do
  {
```

```
 document.write("The number is " + i);
 document.write("<br />");
 i++;
 }
while (i<=5);
</script>
</body>
</html>
```

The break Statement

The break statement will break the loop and continue executing the code that follows after the loop (if any).

Example

```
<html>
<body>
<script type="text/javascript">
var i=0;
for (i=0;i<=10;i++)
 {
 if (i==3)
   {
   break;
   }
 document.write("The number is " + i);
 document.write("<br />");
 }
</script>
</body>
</html>
```

The continue Statement

The continue statement will break the current loop and continue with the next value.

Example

```
<html>
<body>
<script type="text/javascript">
var i=0
for (i=0;i<=10;i++)
 {
 if (i==3)
```

```
   {
   continue;
   }
  document.write("The number is " + i);
  document.write("<br />");
  }
</script>
</body>
</html>
```

JavaScript For...In Statement

The for...in statement loops through the elements of an array or through the properties of an object.

Syntax

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

Note: The code in the body of the for...in loop is executed once for each element/property.

Note: The variable argument can be a named variable, an array element, or a property of an object.

Example

Use the for...in statement to loop through an array:

```
<html>
<body>

<script type="text/javascript">
var x;
var mycars = new Array();
mycars[0] = "Saab";
mycars = "Volvo";
mycars = "BMW";

for (x in mycars)
  {
  document.write(mycars[x] + "<br />");
  }
```

```
</script>

</body>
</html>
```

JavaScript is an Object Oriented Programming (OOP) language.

Object Oriented Programming

JavaScript is an Object Oriented Programming (OOP) language. An OOP language allows you to define your own objects and make your own variable types.

However, creating your own objects will be explained later, in the Advanced JavaScript section. We will start by looking at the built-in JavaScript objects, and how they are used. The next pages will explain each built-in JavaScript object in detail.

Note that an object is just a special kind of data. An object has properties and methods.

Properties

Properties are the values associated with an object.

In the following example we are using the length property of the String object to return the number of characters in a string:

```
<script type="text/javascript">
var txt="Hello World!";
document.write(txt.length);
</script>
```

Methods

Methods are the actions that can be performed on objects.

In the following example we are using the toUpperCase() method of the String object to display a text in uppercase letters:

```
<script type="text/javascript">
var str="Hello world!";
document.write(str.toUpperCase());
</script>
```

What is an Array?

An array is a special variable, which can hold more than one value, at a time.

If you have a list of items (a list of car names, for example), storing the cars in single variables could look like this:

```
$cars1="Saab";
$cars2="Volvo";
$cars3="BMW";
```

However, what if you want to loop through the cars and find a specific one? And what if you had not 3 cars, but 300?

The best solution here is to use an array!

An array can hold all your variable values under a single name. And you can access the values by referring to the array name.

Each element in the array has its own ID so that it can be easily accessed.

Create an Array

The following code creates an Array object called myCars:

```
var myCars=new Array();
```

There are two ways of adding values to an array (you can add as many values as you need to define as many variables you require).

1:

```
var myCars=new Array();
myCars[0]="Saab";
myCars="Volvo";
myCars="BMW";
```

You could also pass an integer argument to control the array's size:

```
var myCars=new Array(3);
myCars[0]="Saab";
myCars="Volvo";
myCars="BMW";
```

2:

var myCars=new Array("Saab","Volvo","BMW");

Note: If you specify numbers or true/false values inside the array then the type of variables will be numeric or Boolean instead of string.

Access an Array

You can refer to a particular element in an array by referring to the name of the array and the index number. The index number starts at 0.

The following code line:

document.write(myCars[0]);

Modify Values in an Array

To modify a value in an existing array, just add a new value to the array with a specified index number:

myCars[0]="Opel";

Now, the following code line:

document.write(myCars[0]);

Example

```
<html>
<head>
<script language="JavaScript">

function temp(form)
{
var f = parseFloat(form.DegF.value, 10);
var T = 0;
T = (f - 62.0) * 8.0 / 7.0;
form.DegC.value = T;
}
// done hiding from old browsers -->
</script>
</head>
<body>
<FORM>
```

```
<h2>Fahrenheit to Celsius Converter</h2>
Enter a temperature in degrees F:
<INPUT NAME="DegF" VALUE="0" MAXLENGTH="25" SIZE=25>
<p>
Click button to calculate the temperature
in degrees T:
<INPUT NAME="calc" VALUE="Calculate" TYPE=BUTTON
onClick=temp(this.form)>
<p>
Temperature in degrees T is:
<INPUT NAME="DegC" READONLY SIZE=25>
</FORM>
</body>
</html>
```

Review

Summary

- JavaScript was originally designed in order to add interactivity to the Web Pages or HTML pages.
- JavaScript is quite easy as the web programmer can easily embed the JavaScript code into the HTML pages.
- Iinteresting fact about JavaScript is that it doesn't require the user to purchase any license.
- The advantage of using JavaScript is that it is already interpreted, which is the reason it is also called as an interpreted language. Interpreted language means that the JavaScript execute simply without any preliminary compilation; which is another reason why it is called as light-weight programming language.
    - conditional `catch` clauses
    - property getter and setter functions
    - iterator protocol adopted from Python
    - shallow generators/coroutines also adopted from Python
    - array comprehensions and generator expressions also adopted from Python

- proper block scope via new `let` keyword
- array and object destructuring (limited form of pattern matching)
- concise function expressions (`function(args) expr`)
- E4X

Key terms ::

- **ECMA**
- **Java Script**
- **href**
- **https,http**
- **PERL,PHP,CGI,VB Script**

Key Terms Quiz:

Complete each statement by writing one of the terms listed under key terms in each blank

1. Client-side scripts are often embedded within an ----------.
2. Server-side scripts are written in ----------languages.
3. JavaScript was designed to add interactivity to---------- pages.
4. A scripting language is a---------- programming language.
5. JavaScript is an ---------- language.
6. JavaScript can read and write ---------- elements
7. ---------- is the official JavaScript standard.
8. JavaScript can be used to create ----------.
9. ECMAScript is developed and maintained by ----------.
10. DOM stands for----------.

Multiple choices

1. A prompt box is often used if you want the user to input a value.

a.after entering a page b. before entering a page

c select the page d.none of the above

2. JavaScript is officially managed by

a. Hot java b.Internet Explore

c Mozilla d.Netscape

3. A JavaScript engine is ---------- JavaScript source code and executes the script accordingly

a. an interpreter that interprets b. a compiler that compiles

c both compiler and interpreter that compiles and interprets d. none of the above.

4. Internet Explorer has ----------available for java script

a. Three debuggers b. Four debuggers

c only one debugger d. none of the above

5. Properties are ----------associated with ---------

a. constant , a class b. the values ,a class

c the values, an object. d. constant, an object

Review questions

In your own words briefly answer the following questions

1. What is client side scripting?

2 Can you make ac omparision between Clinetside scripting and server side scripting?

3. What is java scitping?

4. What is client side image map?

5. What can java script do?

6. How many features are supported by java script engine? What are they?

7. How many conditional statements are there in java script?

8. Why is confirm box used in jscript?

9. How do you define a function?

10. What is the syntax for “for loop “in java script?

Lesson lab::

Complete the following exercise

1. Develop static pages (using only HTML) of an online Book store. The pages should resemble: www.amazon.com
The website should consist of the following pages.
Home page, Registration and user Login, User profile page, Books catalog, Shopping cart, Payment By credit card, order confirmation.

2. Validate the registration, user login, user profile and payment by credit card pages using JavaScript**.**