Chapter 1

1.1 Review of Web Technology

Web technologies that relate to the interface between web servers and their clients called Web Technology.It includes markup languages, programming interfaces and languages, and standards for document identification and display. Web Technologies are playing the leading role in the World Wide Web includes many latest evolutions in it like Web Services, Web 2.0, Tableless Design, HTML, XHTML, XML, CSS 2.0 etc. Web technology aims to enhance creativity, secure information sharing, collaboration and functionality of the web. Web 2.0,Web 3.0 are the main revolutionary Technologies of it.

Wide Spread of Web 2.0

Web 2.0 is the business revolution in the computer industry caused by the move to the Internet as a platform, and an attempt to understand the rules for success on that new platform. Web 2.0 concepts have led to the development and evolution of web culture communities and hosted services, such as social-networking sites, video sharing sites, wikis, blogs, and folksonomies. Web 2.0 sites often feature a rich, user friendly interface based on Ajax, Open Laszlo, Flex or similar rich media.

Web 2.0’s latest technology amendments

- **Social Networking Websites:**

Social networks connect people at low cost. This can be beneficial for entrepreneurs and small businesses looking to expand their contact base. The social network sites focuses on building online communities of people who share interests and/or activities.

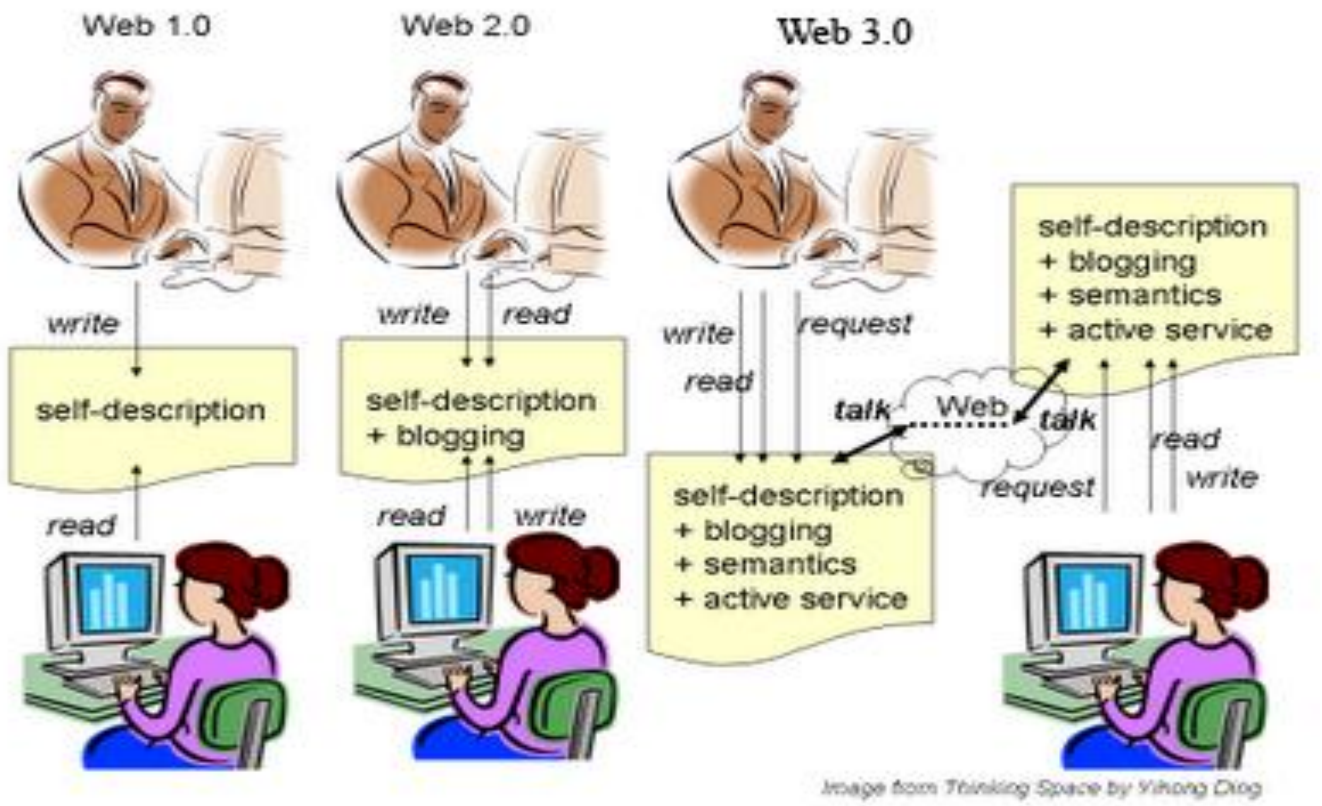
- **Video Sharing Websites:**

Video sharing refers to websites or software where users can distribute their video clips. Some services may charge, but the bulk of them offer free services.

- **Wiki’s:**
  A wiki is a collection of web pages designed to enable anyone who accesses it to contribute or modify content, using a simplified markup language.

Web 2.0 websites typically include some of the following features/techniques.

1. Searching.
2. Linking.
3. Authoring.
4. Taging.
5. Signals.
6. Higher Education.
7. Also for Government Officials.
   Web 3.0 is one of the terms used to describe the evolutionary stage of the Web that follows Web 2.0. Web 3.0 is highly speculative. Web 1.0 was dial-up 50K average bandwidth.Web 2.0 is an average 1 megabit of bandwidth ,Web 3.0 will be 10 megabits of bandwidth all the time, which will be the full video Web, and that will feel like Web 3.0 As the Technology is growing day by day, it is not surprising to see the evolution of new technologies by overriding the old ones.



Why Internet?

- Email
- Social Networking, Chat
- Information sharing
- Getting updates – News around the world
- Entertainment – Games, Videos and Music
- Virtual classrooms
- Remote Access
- Online Jobs

List of Technologies

Client Side Technologies

- HTML, CSS, JavaScript, VBScript
- XHTML, DHTML, WML, AJAX
- FLASH

Server Side Technologies

- ASP, PHP, Perl, JSP
- ASP.NET, Java

MySQL, SQL Server, Access

Some More Advanced Technologies

- XML, XSLT, RSS, Atom
- X-Path, XQuery, WSDL
- XML-DOM, RDF
- Ruby on Rails, GRAIL Framework
- REST, SOAP

How to choose a Technology?

Depends on:

- What is the type of content?
- Who is your audience?
- Who will modify your content?
- What are your Future Plans?
- Availability of technology?
- Your previous experience?
- Portability and Data sharing

1.2 Review of HTML (Hyper Text Markup Language):

HTML is a layout langugage .It contains commands that like a word processor,tell the computer in a very loose sense –what the content of the doucment is.HTML is a small subset of a much more full featured markup Langugage called Standard Generalized Markup Language(SGML).

HTML Contains only two kinds of information-*Markup*,Which consists of all the text contained between angle brackets(<>),and *content*,which is all the text not contained between angle breackets.The difference is that the browser doesn't display markup;instead,Markup contains the information that tells the browser how to display the content, for example, The HTML

```
<html>
<head>
<title> Introduction to HTML </title>
</head>
<body>Hello</body>
</html>
```

HTML is a language for describing web pages.It stands for **H**yper **T**ext **M**arkup **L**anguage .HTML is not a programming language, it is a **markup language ,**A markup language is a set of **markup tags which** uses **markup tags** to describe web pages

HTML Tags

HTML markup tags are usually called HTML tags.HTML tags are keywords surrounded by **angle brackets** like <html>.HTML tags normally **come in pairs** like <b> and </b>.The first tag in a pair is the **start tag,** the second tag is the **end tag.**Start and end tags are also called **opening tags** and **closing tags.**

Attributes and Values

- Properties, traits, or characteristics that modify the way a tag looks or acts
  - Usually in pairs: <body bgcolor="teal">
  - Sometimes not: <dl compact>
- Most HTML tags can take attributes
  - Format of value depends on attribute
  - width="150" ... href="page3.htm" *not* width="page3.htm" ... href="150"

  Anchor Tag
- The tag that puts the HT in HTML
  - <a> ... </a> (useless by itself)
  - Must have attributes to be useful
- HREF (Hypertext REFerence) attribute
  - Makes a jump to someplace (URL)
    <a href="mypage.htm">My Page</a>
    <a href="www.google.com">Google</a>
- Link text is underscored by default
- *Whatever* is between <a> and </a> is hot (clickable)
  - Clicking makes the link go somewhere or do something

HTML Documents = Web Pages

HTML documents describe web pages,HTML documents contain HTML tags and plain text.HTML documents are also called web pages.The purpose of a web browser (like Internet Explorer or Firefox) is to read HTML documents and display them as web pages. The browser does not display the HTML tags, but uses the tags to interpret the content of the page:

.HTM or .HTML File Extension?

When you save an HTML file, the file extension is .htm or the .html. There is no difference.

List of HTML Tags

Tag	**Description**
Basic	
<!DOCTYPE>	Defines the document type
<html>	Defines an HTML document
<body>	Defines the document's body
<h1> to <h6>	Defines HTML headings
<p>	Defines a paragraph
<br />	Inserts a single line break
<hr />	Defines a horizontal line
<!--...-->	Defines a comment
Formatting	
<acronym>	Defines an acronym
<abbr>	Defines an abbreviation
<address>	Defines contact information for the author/owner of a document
<b>	Defines bold text
<bdo>	Defines the text direction
<big>	Defines big text
<blockquote>	Defines a long quotation
<center>	Deprecated. Defines centered text
<cite>	Defines a citation

<code>	Defines computer code text
<del>	Defines deleted text
<dfn>	Defines a definition term
<em>	Defines emphasized text
<font>	Deprecated. Defines font, color, and size for text
<i>	Defines italic text
<ins>	Defines inserted text
<kbd>	Defines keyboard text
<pre>	Defines preformatted text
<q>	Defines a short quotation
<s>	Deprecated. Defines strikethrough text
<samp>	Defines sample computer code
<small>	Defines small text
<strike>	Deprecated. Defines strikethrough text
<strong>	Defines strong text
<sub>	Defines subscripted text
<sup>	Defines superscripted text
<tt>	Defines teletype text
<u>	Deprecated. Defines underlined text
<var>	Defines a variable part of a text
<xmp>	Deprecated. Defines preformatted text
Forms	
<form>	Defines an HTML form for user input

<input />	Defines an input control
<textarea>	Defines a multi-line text input control
<button>	Defines a push button
<select>	Defines a select list (drop-down list)
<optgroup>	Defines a group of related options in a select list
<option>	Defines an option in a select list
<label>	Defines a label for an input element
<fieldset>	Defines a border around elements in a form
<legend>	Defines a caption for a fieldset element
<isindex>	Deprecated. Defines a searchable index related to a document
Frames	
<frame />	Defines a window (a frame) in a frameset
<frameset>	Defines a set of frames
<noframes>	Defines an alternate content for users that do not support frames
<iframe>	Defines an inline frame
Images	
<img />	Defines an image
<map>	Defines an image-map
<area />	Defines an area inside an image-map
Links	
<a>	Defines an anchor
<link />	Defines the relationship between a document and an external resource
Lists	

<ul>	Defines an unordered list
<ol>	Defines an ordered list
<li>	Defines a list item
<dir>	Deprecated. Defines a directory list
<dl>	Defines a definition list
<dt>	Defines a term (an item) in a definition list
<dd>	Defines a description of a term in a definition list
<menu>	Deprecated. Defines a menu list
Tables	
<table>	Defines a table
<caption>	Defines a table caption
<th>	Defines a header cell in a table
<tr>	Defines a row in a table
<td>	Defines a cell in a table
<thead>	Groups the header content in a table
<tbody>	Groups the body content in a table
<tfoot>	Groups the footer content in a table
<col />	Defines attribute values for one or more columns in a table
<colgroup>	Defines a group of columns in a table for formatting
Styles	
<style>	Defines style information for a document
<div>	Defines a section in a document
<span>	Defines a section in a document

Meta Info	
<head>	Defines information about the document
<title>	Defines the document title
<meta>	Defines metadata about an HTML document
<base />	Defines a default address or a default target for all links on a page
<basefont />	Deprecated. Defines a default font, color, or size for the text in a page
Programming	
<script>	Defines a client-side script
<noscript>	Defines an alternate content for users that do not support client-side scripts
<applet>	Deprecated. Defines an embedded applet
<object>	Defines an embedded object
<param />	Defines a parameter for an object

HTML Tables

Tables are defined with the <table> tag.A table is divided into rows (with the <tr> tag), and each row is divided into data cells (with the <td> tag). td stands for "table data," and holds the content of a data cell. A <td> tag can contain text, links, images, lists, forms, other tables, etc.

```
<html>
<body>
<table border="1">
<tr>
<td>row 1, cell 1</td>
<td>row 1, cell 2</td>
</tr>
<tr>
<td>row 2, cell 1</td>
<td>row 2, cell 2</td>
</tr>
</table>
</body>
</html>
```

HTML Tables and the Border Attribute

If you do not specify a border attribute, the table will be displayed without borders. Sometimes this can be useful, but most of the time, we want the borders to show.

```
<table border="1">
<tr>
<td>Row 1, cell 1</td>
```

```
<td>Row 1, cell 2</td>
</tr>
</table>
```

HTML Table Headers

Header information in a table are defined with the <th> tag.
All major browsers will display the text in the <th> element as bold and centered.

```
<table border="1">
<tr>
<th>Header 1</th>
<th>Header 2</th>
</tr>
<tr>
<td>row 1, cell 1</td>
<td>row 1, cell 2</td>
</tr>
<tr>
<td>row 2, cell 1</td>
<td>row 2, cell 2</td>
</tr>
</table>
```

HTML Forms

HTML forms are used to pass data to a server.A form can contain input elements like text fields, checkboxes, radio-buttons, submit buttons and more. A form can also contain select lists, textarea, fieldset, legend, and label elements.The <form> tag is used to create an HTML form:

```
<form>
.
input elements
.
</form>
```

HTML Forms - The Input Element

The most important form element is the input element. The input element is used to select user information. An input element can vary in many ways, depending on the type attribute. An input element can be of type text field, checkbox, password, radio button, submit button, and more.

Text Fields

<input type="text" /> defines a one-line input field that a user can enter text into:

```
<form>
First name: <input type="text" name="firstname" /><br />
Last name: <input type="text" name="lastname" />
</form>
```

Password Field

<input type="password" /> defines a password field:

```
<form>
Password: <input type="password" name="pwd" />
</form>
```

Radio Buttons

<input type="radio" /> defines a radio button. Radio buttons let a user select ONLY ONE one of a limited number of choices:

```
<form>
<input type="radio" name="sex" value="male" /> Male<br />
<input type="radio" name="sex" value="female" /> Female
</form>
```

Checkboxes

<input type="checkbox" /> defines a checkbox. Checkboxes let a user select ONE or MORE options of a limited number of choices.

```
<form>
<input type="checkbox" name="vehicle" value="Bike" /> I have a bike<br />
<input type="checkbox" name="vehicle" value="Car" /> I have a car
</form>
```

Submit Button

<input type="submit" /> defines a submit button.
A submit button is used to send form data to a server. The data is sent to the page specified in the form's action attribute. The file defined in the action attribute usually does something with the received input:

```
<form name="input" action="html_form_action.asp" method="get">
Username: <input type="text" name="user" />
<input type="submit" value="Submit" />
</form>
```

The HTML frame Element

The <frame> tag defines one particular window (frame) within a frameset.In the example below a frameset with two columns.The first column is set to 25% of the width of the browser window. The second column is set to 75% of the width of the browser window. The document "frame_a.htm" is put into the first column, and the document "frame_b.htm" is put into the second column:

```
<frameset cols="25%,75%">
  <frame src="frame_a.htm" />
  <frame src="frame_b.htm" />
</frameset>
```

Iframe - Set Height and Width

The height and width attributes are used to specify the height and width of the iframe.The attribute values are specified in pixels by default, but they can also be in percent (like "80%").<iframe src="demo_iframe.htm" width="200" height="200"></iframe>

Iframe - Remove the Border

The frameborder attribute specifies whether or not to display a border around the iframe.Set the attribute value to "0" to remove the border:

```
<iframe src="demo_iframe.htm" frameborder="0"></iframe>
```

Color Values

HTML colors are defined using a hexadecimal notation (HEX) for the combination of Red, Green, and Blue color values (RGB). The lowest value that can be given to one of the light sources is 0 (in HEX: 00). The highest value is 255 (in HEX: FF).HEX values are specified as 3 pairs of two-digit numbers, starting with a # sign.

HTML Link Syntax

The HTML code for a link is simple. It looks like this:

<a href="*url*">*Link text*</a>

<a href="http://www.epfnepal.com/">KSK</a>

HTML Images - The <img> Tag and the Src Attribute

In HTML, images are defined with the <img> tag.

The <img> tag is empty, which means that it contains attributes only, and has no closing tag. To display an image on a page, you need to use the src attribute. Src stands for "source". The value of the src attribute is the URL of the image you want to display.

<img src="*url*" alt="*some_text*"/>

HTML Images - The Alt Attribute

The required alt attribute specifies an alternate text for an image, if the image cannot be displayed

```
<img src="boat.gif" alt="Big Boat" />
```

The alt attribute provides alternative information for an image if a user for some reason cannot view it (because of slow connection, an error in the src attribute, or if the user uses a screen reader).

```
<img src="pulpit.jpg" alt="Pulpit rock" width="304" height="228" />
```
Links
```
Ordinary link: <a href="http://www.example.com/">Link-text goes here</a>
Image-link: <a href="http://www.example.com/"><img src="URL" alt="Alternate Text" /></a>
Mailto link: <a href="mailto:webmaster@example.com">Send e-mail</a>
A named anchor:
<a name="tips">Tips Section</a>
<a href="#tips">Jump to the Tips Section</a>
```
Unordered list
```
<ul>
  <li>Item</li>
  <li>Item</li>
</ul>
```
Ordered list
```
<ol>
  <li>First item</li>
  <li>Second item</li>
</ol>
```
Definition list
```
<dl>
  <dt>First term</dt>
    <dd>Definition</dd>
  <dt>Next term</dt>
    <dd>Definition</dd>
</dl>
```
Tables
```
<table border="1">
  <tr>
    <th>Tableheader</th>
    <th>Tableheader</th>
  </tr>
  <tr>
    <td>sometext</td>
    <td>sometext</td>
  </tr>
</table>
```
Frames
```
<frameset cols="25%,75%">
  <frame src="page1.htm" />
  <frame src="page2.htm" />
</frameset>
```
Forms
```
<form action="http://www.example.com/test.asp" method="post/get">
<input type="text" name="email" size="40" maxlength="50" />
<input type="password" />
<input type="checkbox" checked="checked" />
<input type="radio" checked="checked" />
<input type="submit" value="Send" />
<input type="reset" />
<input type="hidden" />
<select>
<option>Apples</option>
<option selected="selected">Bananas</option>
```

```
<option>Cherries</option>
</select><textarea name="comment" rows="60" cols="20"></textarea>
</form>
```

Other Elements

```
<!-- This is a comment -->
<blockquote>
Text quoted from a source.
</blockquote>
<address>
Written by W3Schools.com<br />
<a href="mailto:us@example.org">Email us</a><br />
Address: Box 564, Disneyland<br />
Phone: +12 34 56 78
</address>
```

HTML Different Doctypes

The doctype declaration is not an HTML tag; it is an instruction to the web browser about what version of the markup language the page is written in.

The doctype declaration refers to a Document Type Definition (DTD). The DTD specifies the rules for the markup language, so that the browsers render the content correctly.

The doctype declaration should be the very first thing in an HTML document, before the <html> tag.

HTML 4.01 Strict

This DTD contains all HTML elements and attributes, but does NOT INCLUDE presentational or deprecated elements (like font and center). Framesets are not allowed:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
```

Internal Style Sheet

An internal style sheet can be used if one single document has a unique style. Internal styles are defined in the <head> section of an HTML page, by using the <style> tag, like this

```
<head>
<style type="text/css">
body {background-color:yellow}
p {color:blue}
</style>
</head>
```

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

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

```
Example1
<html>
<head>
<script type="text/javascript">
function displayDate()
{
document.getElementById("demo").innerHTML=Date();
}
</script>
</head>
<body>

<h1>My First Web Page</h1>
<p id="demo">This is a paragraph.</p>

<button type="button" onclick="displayDate()">Display Date</button>

</body>
</html>
```

Changing HTML Elements

The example below writes the current date into an existing <p> element:

```
<html>
<body>
<h1>My First Web Page</h1>
<p id="demo"></p>
<script type="text/javascript">
document.getElementById("demo").innerHTML=Date();
</script>
</body>
</html>
```

To manipulate HTML elements JavaScript uses the DOM method **getElementById()**. This method access the element with the specified id.

Scripts in <head> and <body>

You can place an unlimited number of scripts in your document, and you can have scripts in both the body and the head section at the same time.It is a common practice to put all functions in the head section, or at the bottom of the page. This way they are all in one place and do not interfere with page content.

JavaScript is Case Sensitive

Unlike HTML, JavaScript is case sensitive - therefore watch your capitalization closely when you write JavaScript statements, create or call variables, objects and functions.

JavaScript Statements

A JavaScript statement is a command to a browser. The purpose of the command is to tell the browser what to do.This JavaScript statement tells the browser to write "Hello Dolly" to the web page:

```
document.write("Hello Dolly");
```

It is normal to add a semicolon at the end of each executable statement. Most people think this is a good programming practice, and most often you will see this in JavaScript examples on the web.
The semicolon is optional (according to the JavaScript standard), and the browser is supposed to interpret the end of the line as the end of the statement. Because of this you will often see examples without the semicolon at the end.Using semicolons makes it possible to write multiple statements on one line.

JavaScript Blocks

JavaScript statements can be grouped together in blocks.
Blocks start with a left curly bracket {, and ends with a right curly bracket }.
The purpose of a block is to make the sequence of statements execute together.
This example will write a heading and two paragraphs to a web page:

```
<script type="text/javascript">
{
document.write("<h1>This is a heading</h1>");
document.write("<p>This is a paragraph.</p>");
document.write("<p>This is another paragraph.</p>");
}
</script>
```

The example above is not very useful. It just demonstrates the use of a block. Normally a block is used to group statements together in a function or in a condition (where a group of statements should be executed if a condition is met).

JavaScript Comments

Comments can be added to explain the JavaScript, or to make the code more readable.
Single line comments start with //.

```
<script type="text/javascript">
// Write a heading
document.write("<h1>This is a heading</h1>");
// Write two paragraphs:
document.write("<p>This is a paragraph.</p>");
document.write("<p>This is another paragraph.</p>");
</script>
```

JavaScript Multi-Line Comments
Multi line comments start with /* and end with */.
The following example uses a multi line comment to explain the code:

```
<script type="text/javascript">
/*
The code below will write
one heading and two paragraphs
*/
document.write("<h1>This is a heading</h1>");
document.write("<p>This is a paragraph.</p>");
document.write("<p>This is another paragraph.</p>");
</script>
```

JavaScript Variables

As with algebra, JavaScript variables are used to hold values or expressions.
A variable can have a short name, like x, or a more descriptive name, like carname.
Rules for JavaScript variable names:

- Variable names are case sensitive (y and Y are two different variables)
- Variable names must begin with a letter or the underscore character

Note: Because JavaScript is case-sensitive, variable names are case-sensitive.

Declaring (Creating) JavaScript Variables

Creating variables in JavaScript is most often referred to as "declaring" variables.
You declare JavaScript variables with the **var** keyword:

```
var x;
var carname;
```

After the declaration shown above, the variables are empty (they have no values yet).
However, you can also assign values to the variables when you declare them:

```
var x=5;
var carname="Volvo";
```

After the execution of the statements above, the variable **x** will hold the value **5**, and **carname** will hold the value **Volvo**.
When you assign a text value to a variable, use quotes around the value. If you redeclare a JavaScript variable, it will not lose its value.

Local JavaScript Variables

A variable declared within a JavaScript function becomes **LOCAL** and can only be accessed within that function. (the variable has local scope).You can have local variables with the same name in different functions, because local variables are only recognized by the function in which they are declared.Local variables are destroyed when you exit the function.

Global JavaScript Variables

Variables declared outside a function becomes **GLOBAL**, and all scripts and functions on the web page can access it.
Global variables are destroyed when you close the page.
If you declare a variable, without using "var", the variable always becomes **GLOBAL**

Assigning Values to Undeclared JavaScript Variables

If you assign values to variables that have not yet been declared, the variables will automatically be declared as global variables.

```
x=5;
carname="Volvo";
```

will declare the variables x and carname as global variables (if they don't already exist).

JavaScript Arithmetic

As with algebra, you can do arithmetic operations with JavaScript variables:

```
y=x-5;
z=y+5;
```

JavaScript Arithmetic Operators
Arithmetic operators are used to perform arithmetic between variables and/or values.
Given that **y=5**, the table below explains the arithmetic operators:

Operator	**Description**	**Example**	**Result**	
+	Addition	x=y+2	x=7	y=5
-	Subtraction	x=y-2	x=3	y=5
*	Multiplication	x=y*2	x=10	y=5
/	Division	x=y/2	x=2.5	y=5
%	Modulus (division remainder)	x=y%2	x=1	y=5

++	Increment	x=++y	x=6	y=6
		x=y++	x=5	y=6
--	Decrement	x=--y	x=4	y=4
		x=y--	x=5	y=4

JavaScript Assignment Operators

Assignment operators are used to assign values to JavaScript variables.
Given that **x=10** and **y=5**, the table below explains the assignment operators:

Operator	**Example**	**Same As**	**Result**
=	x=y		x=5
+=	x+=y	x=x+y	x=15
-=	x-=y	x=x-y	x=5
*=	x*=y	x=x*y	x=50
/=	x/=y	x=x/y	x=2
%=	x%=y	x=x%y	x=0

The + Operator Used on Strings

The + operator can also be used to add string variables or text values together.
To add two or more string variables together, use the + operator.

```
txt1="What a very";
txt2="nice day";
txt3=txt1+txt2;
```

After the execution of the statements above, the variable txt3 contains "What a verynice day".
To add a space between the two strings, insert a space into one of the strings:

```
txt1="What a very ";
txt2="nice day";
txt3=txt1+txt2;
```

or insert a space into the expression:

```
xt1="What a very";
txt2="nice day";
txt3=txt1+" "+txt2;
```

After the execution of the statements above, the variable txt3 contains:
"What a very nice day"

Adding Strings and Numbers

The rule is: If you add a number and a string, the result will be a string!

```
x=5+5;
document.write(x);
```

x="5"+"5";
document.write(x);
x=5+"5";
document.write(x);
x="5"+5;
document.write(x);

Comparison Operators

Comparison operators are used in logical statements to determine equality or difference between variables or values. Given that **x=5**, the table below explains the comparison operators:

Operator	**Description**	**Example**
==	is equal to	x==8 is false
===	is exactly equal to (value and type)	x===5 is true x==="5" is false
!=	is not equal	x!=8 is true
>	is greater than	x>8 is false
<	is less than	x<8 is true
>=	is greater than or equal to	x>=8 is false
<=	is less than or equal to	x<=8 is true

How Can it be Used

Comparison operators can be used in conditional statements to compare values and take action depending on the result:if (age<18) document.write("Too young");

Logical Operators

Logical operators are used to determine the logic between variables or values.Given that **x=6 and y=3**, the table below explains the logical operators:

Operator	**Description**	**Example**
&&	and	(x < 10 && y > 1) is true
||	or	(x==5 || y==5) is false
!	not	!(x==y) is true

Conditional Operator

JavaScript also contains a conditional operator that assigns a value to a variable based on some condition.
variablename=(condition)?value1:value2
greeting=(visitor=="PRES")?"Dear President ":"Dear ";

If the variable **visitor** has the value of "PRES", then the variable **greeting** will be assigned the value "Dear President " else it will be assigned "Dear".

JavaScript If...Else Statements

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.

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

Note that if is written in lowercase letters. Using uppercase letters (IF) will generate a JavaScript error!

```
<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>
```

Notice that there is no ..else.. in this syntax. You tell the browser to execute some code **only if the specified condition is true**.

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.

```
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.

```
if (condition1)
  {
  code to be executed if condition1 is true
  }
else if (condition2)
  {
<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>
```

Example1

```
<html>
<body>
<script type="text/javascript">
var d = new Date();
var time = d.getHours();
if (time < 10)
  {
  document.write("<b>Good morning</b>");
  }
</script>
<p>This example demonstrates the If statement.</p>
<p>If the time on your browser is less than 10, you will get a "Good morning" greeting.</p>
</body>
</html>
```

Example 2

```
<html>
<body>
<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>
<p>This example demonstrates the if..else if...else statement.</p>
</body>
</html>
```

The JavaScript Switch Statement

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

```
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.

```
<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();
var 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>
```

JavaScript Popup Boxes

JavaScript has three kind of popup boxes: Alert box, Confirm box, and Prompt box.

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.
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.
confirm("*sometext*");

```
<html>
<head>
<script type="text/javascript">
function show_confirm()
{
var r=confirm("Press a button");
if (r==true)
  {
  alert("You pressed OK!");
  }
else
  {
  alert("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.
prompt("*sometext*","*defaultvalue*");

```
<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

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

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

JavaScript Function 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 page 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):

```
<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>
```

The Lifetime of JavaScript Variables

If you declare a variable, using "var", within a function, the variable can only be accessed within that function. When you exit the function, the variable is destroyed. These variables are called local variables. You can have local variables with the same name in different functions, because each is recognized only by the function in which it is declared.
If you declare a variable outside a function, all the functions on your page can access it. The lifetime of these variables starts when they are declared, and ends when the page is closed

JavaScript For Loop

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 kind 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.

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

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.
The increment parameter could also be negative, and the <= could be any comparing statement.

```
<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>
```

Example1

```
<html>
<body>
<script type="text/javascript">
for (i = 0; i <= 5; i++)
{
document.write("The number is " + i);
document.write("<br />");
}
</script>
<p>Explanation:</p>
<p>This for loop starts with i=0.</p>
<p>As long as <b>i</b> is less than, or equal to 5, the loop will continue to run.</p>
<p><b>i</b> will increase by 1 each time the loop runs.</p></body>
</html>
```

JavaScript While Loop

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

The while Loop

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

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

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.

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

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>
```

Example1:

```
<html>
<body>
<script type="text/javascript">
i = 0;
do
{
document.write("The number is " + i);
document.write("<br />");
i++;
}
while (i <= 5)
</script>
<p>Explanation:</p>
<p><b>i</b>  equal to 0.</p>
<p>The loop will run</p>
<p><b>i</b> will increase by 1 each time the loop runs.</p>
<p>While <b>i</b> is less than , or equal to, 5, the loop will continue to run.</p>
</body>
</html>
```

JavaScript Break and Continue Statements

The break Statement

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

```
<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.

```
<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>
```

Example1

```
<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>
<p>Explanation: The loop will break the current loop and continue with the next value when i=3.</p>
</body>
</html>
```

JavaScript For...In Statement

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

for (*variable* in *object*)
  {

```
  code to be executed
  }
```

The code in the body of the for...in loop is executed once for each property.

```
var person={fname:"John",lname:"Doe",age:25};

for (x in person)
{
document.write(person[x] + " ");
}
```

Example 1

```
<html>
<body>
<script type="text/javascript">
var person={fname:"John",lname:"Doe",age:25};
for (x in person){
document.write(person[x] + " ");
}
</script>
</body>
</html>
```

JavaScript Events

Events are actions that can be detected by JavaScript.

Acting to an Event

The example below displays the date when a button is clicked:

```
<html>
<head>
<script type="text/javascript">
function displayDate()
{
document.getElementById("demo").innerHTML=Date();
}
</script>
</head>
<body>
<h1>My First Web Page</h1>
<p id="demo"></p>
<button type="button" onclick="displayDate()">Display Date</button>
</body>
</html>
```

Events

By using JavaScript, we have the ability to create dynamic web pages. Events are actions that can be detected by JavaScript.Every element on a web page has certain events which can trigger a JavaScript. For example, we can use the onClick event of a button element to indicate that a function will run when a user clicks on the button. We define the events in the HTML tags.

Examples of events:

- A mouse click
- A web page or an image loading
- Mousing over a hot spot on the web page
- Selecting an input field in an HTML form

- Submitting an HTML form
- A keystroke

Events are normally used in combination with functions, and the function will not be executed before the event occurs

onLoad and onUnload

The onLoad and onUnload events are triggered when the user enters or leaves the page.
The onLoad event is often used to check the visitor's browser type and browser version, and load the proper version of the web page based on the information.
Both the onLoad and onUnload events are also often used to deal with cookies that should be set when a user enters or leaves a page. For example, you could have a popup asking for the user's name upon his first arrival to your page. The name is then stored in a cookie. Next time the visitor arrives at your page, you could have another popup saying something like: "Welcome John Doe!".

onFocus, onBlur and onChange

The onFocus, onBlur and onChange events are often used in combination with validation of form fields.
Below is an example of how to use the onChange event. The checkEmail() function will be called whenever the user changes the content of the field:

```
<input type="text" size="30" id="email" onchange="checkEmail()">
```

onSubmit

The onSubmit event is used to validate ALL form fields before submitting it.
Below is an example of how to use the onSubmit event. The checkForm() function will be called when the user clicks the submit button in the form. If the field values are not accepted, the submit should be cancelled. The function checkForm() returns either true or false. If it returns true the form will be submitted, otherwise the submit will be cancelled:

```
<form method="post" action="xxx.htm" onsubmit="return checkForm()">
```

onMouseOver

The onmouseover event can be used to trigger a function when the user mouse over an HTML element:

Example

```
<html>
<head>
<script type="text/javascript">
function writeText(txt)
{
document.getElementById("desc").innerHTML=txt;
}
</script>
</head>
<body>
<img src ="planets.gif" width ="145" height ="126" alt="Planets" usemap="#planetmap" />
<map name="planetmap">
<area shape ="rect" coords ="0,0,82,126"
onmouseover="writeText('The Sun and the gas giant planets like Jupiter are by far the largest objects in our
Solar System.')"
href ="sun.htm" target ="_blank" alt="Sun" />
<area shape ="circle" coords ="90,58,3"
onmouseover="writeText('The planet Mercury is very difficult to study from the Earth because it is always so
close to the Sun.')"
href ="mercur.htm" target ="_blank" alt="Mercury" />
<area shape ="circle" coords ="124,58,8"
onmouseover="writeText('Until the 1960s, Venus was often considered a twin sister to the Earth because
Venus is the nearest planet to us, and because the two planets seem to share many characteristics.')"
href ="venus.htm" target ="_blank" alt="Venus" />
</map>
<p id="desc">Mouse over the sun and the planets and see the different descriptions.</p>
```

```
</body>
</html>
```

JavaScript Try...Catch Statement

The try...catch statement allows you to test a block of code for errors.
JavaScript - Catching Errors
When browsing Web pages on the internet, we all have seen a JavaScript alert box telling us there is a runtime error and asking "Do you wish to debug?". Error message like this may be useful for developers but not for users. When users see errors, they often leave the Web page.This chapter will teach you how to catch and handle JavaScript error messages, so you don't lose your audience.
The try...catch Statement
The try...catch statement allows you to test a block of code for errors. The try block contains the code to be run, and the catch block contains the code to be executed if an error occurs.

```
try
  {
  //Run some code here
  }
catch(err)
  {
  //Handle errors here
  }
```

Note that try...catch is written in lowercase letters. Using uppercase letters will generate a JavaScript error!
Examples
The example below is supposed to alert "Welcome guest!" when the button is clicked. However, there's a typo in the message() function. alert() is misspelled as adddlert(). A JavaScript error occurs. The catch block catches the error and executes a custom code to handle it. The code displays a custom error message informing the user what happened:

```
<html>
<head>
<script type="text/javascript">
var txt="";
function message()
{
try
  {
  adddlert("Welcome guest!");
  }
catch(err)
  {
  txt="There was an error on this page.\n\n";
  txt+="Error description: " + err.description + "\n\n";
  txt+="Click OK to continue.\n\n";
  alert(txt);
  }
}
</script>
</head>
<body>
<input type="button" value="View message" onclick="message()" />
```

The next example uses a confirm box to display a custom message telling users they can click OK to continue viewing the page or click Cancel to go to the homepage. If the confirm method returns false, the user clicked Cancel, and the code redirects the user. If the confirm method returns true, the code does nothing:

```
<html>
<head>
<script type="text/javascript">
var txt="";
function message()
{
try
  {
  adddlert("Welcome guest!");
  }
catch(err)
  {
  txt="There was an error on this page.\n\n";
  txt+="Click OK to continue viewing this page,\n";
  txt+="or Cancel to return to the home page.\n\n";
  if(!confirm(txt))
    {
    document.location.href="http://www.w3schools.com/";
    }
  }
}
</script>
</head>
<body>
<input type="button" value="View message" onclick="message()" />
</body>
</html>
```

The throw Statement

The throw statement can be used together with the try...catch statement, to create an exception for the error. Learn about the throw statement in the next chapter.

Example:

```
<html>
<head>
<script type="text/javascript">
var txt="";
function message()
{
try
  {
  adddlert("Welcome guest!");
  }
catch(err)
  {
  txt="There was an error on this page.\n\n";
  txt+="Click OK to continue viewing this page,\n";
  txt+="or Cancel to return to the home page.\n\n";
  if(!confirm(txt))
    {
    document.location.href="http://www.epfnepal.com/";
    }
  }
}
</script>
</head>
<body>
<input type="button" value="View message" onclick="message()" />
</body>
</html>
```

JavaScript Throw Statement

The throw statement allows you to create an exception.The Throw Statement.The throw statement allows you to create an exception. If you use this statement together with the try...catch statement, you can control Program flow and generate accurate error messages.

throw *exception*

The exception can be a string, integer, Boolean or an object.

Note that *throw* is written in lowercase letters. Using uppercase letters will generate a JavaScript error!

Example

The example below determines the value of a variable called x. If the value of x is higher than 10, lower than 0, or not a number, we are going to throw an error. The error is then caught by the catch argument and the proper error message is displayed:

```
<html>
<body>
<script type="text/javascript">
var x=prompt("Enter a number between 0 and 10:","");
try
  {
  if(x>10)
    {
    throw "Err1";
    }
  else if(x<0)
    {
    throw "Err2";
    }
  else if(isNaN(x))
    {
    throw "Err3";
    }
  }
catch(er)
  {
  if(er=="Err1")
    {
    alert("Error! The value is too high");
    }
  if(er=="Err2")
    {
    alert("Error! The value is too low");
    }
  if(er=="Err3")
    {
    alert("Error! The value is not a number");
    }
  }
</script>
```

```
</body>
</html>
```

Example:

```
<html>
<body>
<script type="text/javascript">
var x=prompt("Enter a number between 0 and 10:","");
try
{
if(x>10)
  {
  throw "Err1";
  }
else if(x<0)
  {
  throw "Err2";
  }
else if(isNaN(x))
  {
  throw "Err3";
  }
}
catch(er)
{
if(er=="Err1")
  {
  alert("Error! The value is too high");
  }
if(er=="Err2")
  {
  alert("Error! The value is too low");
  }
if(er=="Err3")
  {
  alert("Error! The value is not a number");
  }
}
</script>
</body>
</html>
```

JavaScript Special Characters

Insert Special Characters

The backslash (\) is used to insert apostrophes, new lines, quotes, and other special characters into a text string.

Look at the following JavaScript code:

```
var txt="We are the so-called "Vikings" from the north.";
document.write(txt);
```

In JavaScript, a string is started and stopped with either single or double quotes. This means that the string above will be chopped to: We are the so-called.To solve this problem, you must place a backslash (\) before each double quote in "Viking". This turns each double quote into a string literal:

```
var txt="We are the so-called \"Vikings\" from the north.";
document.write(txt);
```

JavaScript will now output the proper text string: We are the so-called "Vikings" from the north.The table below lists other special characters that can be added to a text string with the backslash sign:

Code	**Outputs**
\'	single quote
\"	double quote
\\	backslash
\n	new line
\r	carriage return
\t	tab
\b	backspace
\f	form feed

JavaScript is Case Sensitive

A function named "myfunction" is not the same as "myFunction" and a variable named "myVar" is not the same as "myvar".JavaScript is case sensitive - therefore watch your capitalization closely when you create or call variables, objects and functions.

White Space

JavaScript ignores extra spaces. You can add white space to your script to make it more readable. The following lines are equivalent:

```
var name="Hege";
var name = "Hege";
```

Break up a Code Line

You can break up a code line **within a text string** with a backslash. The example below will be displayed properly:

```
document.write("Hello \
World!");
```

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

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.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>
```

JavaScript String Object

Example1

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

Example2:

```
<html>
<body>
<script type="text/javascript">
var txt = "Hello World!";
document.write("<p>Big: " + txt.big() + "</p>");
document.write("<p>Small: " + txt.small() + "</p>");
document.write("<p>Bold: " + txt.bold() + "</p>");
document.write("<p>Italic: " + txt.italics() + "</p>");
document.write("<p>Fixed: " + txt.fixed() + "</p>");
document.write("<p>Strike: " + txt.strike() + "</p>");
document.write("<p>Fontcolor: " + txt.fontcolor("green") + "</p>");
document.write("<p>Fontsize: " + txt.fontsize(6) + "</p>");
document.write("<p>Subscript: " + txt.sub() + "</p>");
document.write("<p>Superscript: " + txt.sup() + "</p>");
document.write("<p>Link: " + txt.link("http://www.w3schools.com") + "</p>");
document.write("<p>Blink: " + txt.blink() + " (does not work in IE, Chrome, or Safari)</p>");
</script>
</body>
</html>
```

Example3:

```
<html>
<body>
<script type="text/javascript">
var txt="Hello World!";
document.write(txt.toLowerCase() + "<br />");
document.write(txt.toUpperCase());
</script>
</body>
</html>
```

example4:

```
<html>
<body>
```

```
<script type="text/javascript">
var str="Hello world!";
document.write(str.match("world") + "<br />");
document.write(str.match("World") + "<br />");
document.write(str.match("worlld") + "<br />");
document.write(str.match("world!"));
</script>
</body>
</html>
```

Example5:

```
<html>
<body>
<script type="text/javascript">
var str="Visit Microsoft!";
document.write(str.replace("Microsoft","W3Schools"));
</script>
</body>
</html>
```

Example6:

```
<html>
<body>
<script type="text/javascript">
var str="Hello world!";
document.write(str.indexOf("d") + "<br />");
document.write(str.indexOf("WORLD") + "<br />");
document.write(str.indexOf("world"));
</script>
</body>
</html>
```

String object

The String object is used to manipulate a stored piece of text.

Examples of use:

The following example uses the length property of the String object to find the length of a string:

```
var txt="Hello world!";
document.write(txt.length);
var txt="Hello world!";
document.write(txt.toUpperCase());
```

Example1:

```
<html>
<body>
<script type="text/javascript">
var d=new Date();
document.write(d);
</script>
</body>
</html>
```

Example2:

```
<html>
<body>
<script type="text/javascript">
var d=new Date();
document.write(d.getFullYear());
</script>
```

```
</body>
</html>
```

Example3:

```
<html>
<body>
<script type="text/javascript">
var d=new Date();
document.write(d.getTime() + " milliseconds since 1970/01/01");
</script>
</body>
</html>
```

Example4

```
<html>
<body>
<script type="text/javascript">
var d = new Date();
d.setFullYear(1992,10,3);
document.write(d);
</script>
</body>
</html>
```

Example5:

```
<html>
<body>
<script type="text/javascript">
var d=new Date();
document.write("Original form: ");
document.write(d + "<br />");
document.write("To string (universal time): ");
document.write(d.toUTCString());
</script>
</body>
</html>
<html>
<body>
<script type="text/javascript">
var d=new Date();
var weekday=new Array(7);
weekday[0]="Sunday";
weekday[1]="Monday";
weekday[2]="Tuesday";
weekday[3]="Wednesday";
weekday[4]="Thursday";
weekday[5]="Friday";
weekday[6]="Saturday";
document.write("Today is " + weekday[d.getDay()]);
</script>
</body>
</html>
```

Create a Date Object

The Date object is used to work with dates and times.
Date objects are created with the Date() constructor.

There are four ways of instantiating a date:

```
new Date() // current date and time
new Date(milliseconds) //milliseconds since 1970/01/01
new Date(dateString)
new Date(year, month, day, hours, minutes, seconds, milliseconds)
```

Most parameters above are optional. Not specifying, causes 0 to be passed in.
Once a Date object is created, a number of methods allow you to operate on it. Most methods allow you to get and set the year, month, day, hour, minute, second, and milliseconds of the object, using either local time or UTC (universal, or GMT) time.
All dates are calculated in milliseconds from 01 January, 1970 00:00:00 Universal Time (UTC) with a day containing 86,400,000 milliseconds.
Some examples of instantiating a date:

```
var today = new Date()
var d1 = new Date("October 13, 1975 11:13:00")
var d2 = new Date(79,5,24)
var d3 = new Date(79,5,24,11,33,0)
```

Set Dates

We can easily manipulate the date by using the methods available for the Date object.
In the example below we set a Date object to a specific date (14th January 2010):

```
var myDate=new Date();
myDate.setFullYear(2010,0,14);
```

And in the following example we set a Date object to be 5 days into the future:

```
var myDate=new Date();
myDate.setDate(myDate.getDate()+5);
```

If adding five days to a date shifts the month or year, the changes are handled automatically by the Date object itself!

Compare Two Dates

The Date object is also used to compare two dates.
The following example compares today's date with the 14th January 2010:

```
var myDate=new Date();
myDate.setFullYear(2010,0,14);
var today = new Date();

if (myDate>today)
  {
  alert("Today is before 14th January 2010");
  }
else
  {
  alert("Today is after 14th January 2010");
  }
```

JavaScript Array Object

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:

```
var car1="Saab";
var car2="Volvo";
var car3="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

An array can be defined in three ways.
The following code creates an Array object called myCars:

```
1: var myCars=new Array(); // regular array (add an optional integer
myCars[0]="Saab";       // argument to control array's size)
myCars[1]="Volvo";
myCars[2]="BMW";
2: var myCars=new Array("Saab","Volvo","BMW"); // condensed array
3: var myCars=["Saab","Volvo","BMW"]; // literal array
```

If you specify numbers or true/false values inside the array then the variable type will be Number 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]);
<html>
<body>
<script type="text/javascript">
var parents = ["Jani", "Tove"];
var children = ["Cecilie", "Lone"];
var family = parents.concat(children);
document.write(family);
</script>
</body>
</html>
```

example1:

```
<html>
<body>
<script type="text/javascript">
var parents = ["Jani", "Tove"];
var brothers = ["Stale", "Kai Jim", "Borge"];
var children = ["Cecilie", "Lone"];
var family = parents.concat(brothers, children);
document.write(family);
</script>
</body>
</html>
```

JavaScript Math Object

round(),Random(),Max(),Min()

Example (round)

```
<html>
<body>
<script type="text/javascript">
document.write(Math.round(0.60) + "<br />");
document.write(Math.round(0.50) + "<br />");
document.write(Math.round(0.49) + "<br />");
document.write(Math.round(-4.40) + "<br />");
document.write(Math.round(-4.60));
</script>
</body>
</html>
```

Example(random())

```
<html>
<body>
<script type="text/javascript">
//return a random number between 0 and 1
document.write(Math.random() + "<br />");
//return a random integer between 0 and 10
document.write(Math.floor(Math.random()*11));
</script>
</body>
</html>
```

example:

```
<html>
<body>
<script type="text/javascript">
document.write(Math.max(5,10) + "<br />");
document.write(Math.max(0,150,30,20,38) + "<br />");
document.write(Math.max(-5,10) + "<br />");
document.write(Math.max(-5,-10) + "<br />");
document.write(Math.max(1.5,2.5));
</script>
</body>
</html>
```

What is RegExp?

A regular expression is an object that describes a pattern of characters.When you search in a text, you can use a pattern to describe what you are searching for.A simple pattern can be one single character.A more complicated pattern can consist of more characters, and can be used for parsing, format checking, substitution and more.Regular expressions are used to perform powerful pattern-matching and "search-and-replace" functions on text.

Syntax

```
var patt=new RegExp(pattern,modifiers);
```

or more simply:

```
var patt=/pattern/modifiers;
```

- pattern specifies the pattern of an expression
- modifiers specify if a search should be global, case-sensitive, etc.

RegExp Modifiers

Modifiers are used to perform case-insensitive and global searches.
The i modifier is used to perform case-insensitive matching.
The g modifier is used to perform a global match (find all matches rather than stopping after the first match).
Do a case-insensitive search for "KSK" in a string:

```
var str="Visit KSK";
var patt1=/KSK/i;
```

test()

The test() method searches a string for a specified value, and returns true or false, depending on the result.
The following example searches a string for the character "e":

```
var patt1=new RegExp("e");
document.write(patt1.test("The best things in life are free"));
```

exec()

The exec() method searches a string for a specified value, and returns the text of the found value. If no match is found, it returns *null.*The following example searches a string for the character "e":

```
var patt1=new RegExp("e");
document.write(patt1.exec("The best things in life are free"));
```

example:

```
<html>
```

```
<body>
<script type="text/javascript">
var patt1=new RegExp("e");
document.write(patt1.exec("The best things in life are free"));
</script>
</body>
</html>
```

Browser Detection

Almost everything in this tutorial works on all JavaScript-enabled browsers. However, there are some things that just don't work on certain browsers - especially on older browsers.Sometimes it can be useful to detect the visitor's browser, and then serve the appropriate information.The Navigator object contains information about the visitor's browser name, version, and more.

There is no public standard that applies to the navigator object, but all major browsers support it.

Example1:

```
<html>
<body>
<div id="example"></div>
<script type="text/javascript">
txt = "<p>Browser CodeName: " + navigator.appCodeName + "</p>";
txt+= "<p>Browser Name: " + navigator.appName + "</p>";
txt+= "<p>Browser Version: " + navigator.appVersion + "</p>";
txt+= "<p>Cookies Enabled: " + navigator.cookieEnabled + "</p>";
txt+= "<p>Platform: " + navigator.platform + "</p>";
txt+= "<p>User-agent header: " + navigator.userAgent + "</p>";
document.getElementById("example").innerHTML=txt;
</script>
</body>
</html>
```

What is a Cookie?

A cookie is a variable that is stored on the visitor's computer. Each time the same computer requests a page with a browser, it will send the cookie too. With JavaScript, you can both create and retrieve cookie values.

Examples of cookies:

- Name cookie - The first time a visitor arrives to your web page, he or she must fill in her/his name. The name is then stored in a cookie. Next time the visitor arrives at your page, he or she could get a welcome message like "Welcome John Doe!" The name is retrieved from the stored cookie
- Password cookie - The first time a visitor arrives to your web page, he or she must fill in a password. The password is then stored in a cookie. Next time the visitor arrives at your page, the password is retrieved from the cookie
- Date cookie - The first time a visitor arrives to your web page, the current date is stored in a cookie. Next time the visitor arrives at your page, he or she could get a message like "Your last visit was on Tuesday August 11, 2005!" The date is retrieved from the stored cookie

Create and Store a Cookie

In this example we will create a cookie that stores the name of a visitor. The first time a visitor arrives to the web page, he or she will be asked to fill in her/his name. The name is then stored in a cookie. The next time the visitor arrives at the same page, he or she will get welcome message.

First, we create a function that stores the name of the visitor in a cookie variable:

```
function setCookie(c_name,value,exdays)
{
var exdate=new Date();
exdate.setDate(exdate.getDate() + exdays);
var c_value=escape(value) + ((exdays==null) ? "" : "; expires="+exdate.toUTCString());
document.cookie=c_name + "=" + c_value;
}
```

The parameters of the function above hold the name of the cookie, the value of the cookie, and the number of days until the cookie expires.In the function above we first convert the number of days to a valid date, then we add the number of days until the cookie should expire. After that we store the cookie name, cookie value and the expiration date in the document.cookie object.Then, we create another function that returns a specified cookie:

```
function getCookie(c_name)
{
var i,x,y,ARRcookies=document.cookie.split(";");
for (i=0;i<ARRcookies.length;i++)
{
  x=ARRcookies[i].substr(0,ARRcookies[i].indexOf("="));
  y=ARRcookies[i].substr(ARRcookies[i].indexOf("=")+1);
  x=x.replace(/^\s+|\s+$/g,"");
  if (x==c_name)
    {
    return unescape(y);
    }
  }
}
```

The function above makes an array to retrieve cookie names and values, then it checks if the specified cookie exists, and returns the cookie value.
Last, we create the function that displays a welcome message if the cookie is set, and if the cookie is not set it will display a prompt box, asking for the name of the user, and stores the username cookie for 365 days, by calling the setCookie function:

```
function checkCookie()
{
var username=getCookie("username");
  if (username!=null && username!="")
  {
  alert("Welcome again " + username);
  }
else
  {
  username=prompt("Please enter your name:","");
  if (username!=null && username!="")
    {
    setCookie("username",username,365);
    }
  }
}
```

All together now:

```
<html>
<head>
<script type="text/javascript">
function getCookie(c_name)
{
var i,x,y,ARRcookies=document.cookie.split(";");
for (i=0;i<ARRcookies.length;i++)
  {
  x=ARRcookies[i].substr(0,ARRcookies[i].indexOf("="));
  y=ARRcookies[i].substr(ARRcookies[i].indexOf("=")+1);
  x=x.replace(/^\s+|\s+$/g,"");
  if (x==c_name)
    {
```

```
    return unescape(y);
    }
  }
}

function setCookie(c_name,value,exdays)
{
var exdate=new Date();
exdate.setDate(exdate.getDate() + exdays);
var c_value=escape(value) + ((exdays==null) ? "" : "; expires="+exdate.toUTCString());
document.cookie=c_name + "=" + c_value;
}

function checkCookie()
{
var username=getCookie("username");
if (username!=null && username!="")
  {
  alert("Welcome again " + username);
  }
else
  {
  username=prompt("Please enter your name:","");
  if (username!=null && username!="")
    {
    setCookie("username",username,365);
    }
  }
}
</script>
</head>
<body onload="checkCookie()">
</body>
</html>
```

The example above runs the checkCookie() function when the page loads

JavaScript Form Validation

JavaScript can be used to validate data in HTML forms before sending off the content to a server.
Form data that typically are checked by a JavaScript could be:

- has the user left required fields empty?
- has the user entered a valid e-mail address?
- has the user entered a valid date?
- has the user entered text in a numeric field?

Required Fields

The function below checks if a field has been left empty. If the field is blank, an alert box alerts a message, the function returns false, and the form will not be submitted:

Example1:

```
<html>
<head>
<script type="text/javascript">
function validateForm()
{
```

```
var x=document.forms["myForm"]["fname"].value
if (x==null || x=="")
  {
  alert("First name must be filled out");
  return false;
  }
}
</script>
</head>
<body>
<form name="myForm" action="demo_form.asp" onsubmit="return validateForm()" method="post">
First name: <input type="text" name="fname">
<input type="submit" value="Submit">
</form>
</body>
</html>
```

E-mail Validation

The function below checks if the content has the general syntax of an email.
This means that the input data must contain an @ sign and at least one dot (.). Also, the @ must not be the first character of the email address, and the last dot must be present after the @ sign, and minimum 2 characters before the end:

example1:

```
<html>
<head>
<script type="text/javascript">
function validateForm()
{
var x=document.forms["myForm"]["email"].value
var atpos=x.indexOf("@");
var dotpos=x.lastIndexOf(".");
if (atpos<1 || dotpos<atpos+2 || dotpos+2>=x.length)
  {
  alert("Not a valid e-mail address");
  return false;
  }
}
</script>
</head>

<body>
<form name="myForm" action="demo_form.asp" onsubmit="return validateForm();" method="post">
Email: <input type="text" name="email">
<input type="submit" value="Submit">
</form>
</body>
</html>
```

JavaScript Timing Events

With JavaScript, it is possible to execute some code after a specified time-interval. This is called timing events.
It's very easy to time events in JavaScript. The two key methods that are used are:

- setTimeout() - executes a code some time in the future

- clearTimeout() - cancels the setTimeout()

The setTimeout() and clearTimeout() are both methods of the HTML DOM Window object.

The setTimeout() Method

var t=setTimeout("*javascript statement*",*milliseconds*);

The setTimeout() method returns a value. In the syntax defined above, the value is stored in a variable called t. If you want to cancel the setTimeout() function, you can refer to it using the variable name.
The first parameter of setTimeout() can be a string of executable code, or a call to a function.
The second parameter indicates how many milliseconds from now you want to execute the first parameter.
There are 1000 milliseconds in one second.
When the button is clicked in the example below, an alert box will be displayed after 3 seconds.

```
<html>
<head>
<script type="text/javascript">
function timeMsg()
{
var t=setTimeout("alertMsg()",3000);
}
function alertMsg()
{
alert("Hello");
}
</script>
</head>
<body>
<form>
<input type="button" value="Display alert box in 3 seconds"
onclick="timeMsg()" />
</form>
</body>
</html>
```

Example - Infinite Loop

To get a timer to work in an infinite loop, we must write a function that calls itself.
In the example below, when a button is clicked, the input field will start to count (for ever), starting at 0.
Notice that we also have a function that checks if the timer is already running, to avoid creating additional timers, if the button is pressed more than once:

The clearTimeout() Method

clearTimeout(*setTimeout_variable*)

Example

The example below is the same as the "Infinite Loop" example above. The only difference is that we have now added a "Stop Count!" button that stops the timer:

```
<html>
<head>
<script type="text/javascript">
var c=0;
var t;
var timer_is_on=0;
function timedCount()
{
document.getElementById('txt').value=c;
c=c+1;
t=setTimeout("timedCount()",1000);
}
function doTimer()
{
if (!timer_is_on)
  {
  timer_is_on=1;
  timedCount();
  }
}
function stopCount()
{
clearTimeout(t);
timer_is_on=0;
}
</script>
</head>
<body>
<form>
<input type="button" value="Start count!" onclick="doTimer()">
<input type="text" id="txt">
<input type="button" value="Stop count!" onclick="stopCount()">
</form>
</body>
</html>
```

Example2:

```
<html>
<head>
<script type="text/javascript">
var c=0;
var t;
var timer_is_on=0;

function timedCount()
{
document.getElementById('txt').value=c;
c=c+1;
t=setTimeout("timedCount()",1000);
}

function doTimer()
{
if (!timer_is_on)
  {
  timer_is_on=1;
  timedCount();
  }
}
```

```
function stopCount()
{
clearTimeout(t);
timer_is_on=0;
}
</script>
</head>
<body>
<form>
<input type="button" value="Start count!" onclick="doTimer()" />
<input type="text" id="txt" />
<input type="button" value="Stop count!" onclick="stopCount()" />
</form>
<p>
Click on the "Start count!" button above to start the timer. The input field will count forever, starting at 0. Click on the "Stop count!" button to stop the counting. Click on the "Start count!" button to start the timer again.
</p>
</body>
</html>
```

Creating Your Own Objects

There are different ways to create a new object:

1. Create a direct instance of an object

The following code creates an new instance of an object, and adds four properties to it:

```
personObj=new Object();
personObj.firstname="John";
personObj.lastname="Doe";
personObj.age=50;
personObj.eyecolor="blue";
```

alternative syntax (using object literals):

```
personObj={firstname:"John",lastname:"Doe",age:50,eyecolor:"blue"};
```

Adding a method to the personObj is also simple. The following code adds a method called eat() to the personObj:

```
personObj.eat=eat;
```

2. Create an object constructor

Create a function that construct objects:

```
function person(firstname,lastname,age,eyecolor)
{
this.firstname=firstname;
this.lastname=lastname;
this.age=age;
this.eyecolor=eyecolor;
}
```

Inside the function you need to assign things to this.propertyName. The reason for all the "this" stuff is that you're going to have more than one person at a time (which person you're dealing with must be clear). That's what "this" is: the instance of the object at hand.

Once you have the object constructor, you can create new instances of the object, like this:

```
var myFather=new person("John","Doe",50,"blue");
var myMother=new person("Sally","Rally",48,"green");
```

You can also add some methods to the person object. This is also done inside the function:

```
function person(firstname,lastname,age,eyecolor)
{
this.firstname=firstname;
this.lastname=lastname;
this.age=age;
this.eyecolor=eyecolor;

this.newlastname=newlastname;
}
```

Note that methods are just functions attached to objects. Then we will have to write the newlastname() function:

```
function newlastname(new_lastname)
{
this.lastname=new_lastname;
}
```

The newlastname() function defines the person's new last name and assigns that to the person. JavaScript knows which person you're talking about by using "this.". So, now you can write: myMother.newlastname("Doe").

Assignment

1. What is Web Technology?
2. What are the Uses of HTML ?Explain Different Tags uesd in HTML.
3. What is JavaScript?