1. INTRODUCATION OF HTML

Hyper Text Markup Language (HTML) is the main markup language for web pages. HTML elements are the basic building-blocks of webpages.HTML is written in the form of HTML elements consisting of *tags* enclosed in angle brackets (like <html>), within the web page content. HTML tags most commonly come in pairs like <h1> and </h1>, although some tags, known as *empty elements*, are unpaired, for example <img>.The first tag in a pair is the *start tag*, the second tag is the *end tag* (they are also called *opening tags* and *closing tags*). In between these tags web designers can add text, tags, comments and other types of text-based content.
The purpose of a web browser is to read HTML documents and compose them into visible or audible web pages. The browser does not display the HTML tags, but uses the tags to interpret the content of the page.

HTML elements form the building blocks of all websites. HTML allows images and objects to be embedded and can be used to create interactive forms. It provides a means to create structured documents by denoting structural semantics for text such as headings, paragraphs, lists, links, quotes and other items. It can embed scripts in languages such as JavaScript which affect the behavior of HTML webpages. Web browsers can also refer to Cascading Style Sheets (CSS) to define the appearance and layout of text and other material. The W3C, maintainer of both the HTML and the CSS standards, encourages the use of CSS over explicitly presentational HTML markup.

HTML :-

- HTML stands for Hyper Text Markup Language
- HTML is not a programming language, it is a markup language
- A markup language is a set of markup tags
- HTML uses markup tags to describe web pages

HTML Tags : =

- HTML tags are keywords surrounded by angle brackets like <html>
- HTML tags normally come in pairs like and
- 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

HTML Documents = Web Pages

- HTML documents describe web pages
- HTML documents contain HTML tags and plain text
- HTML documents are also called web pages

Example: <html><head><title>
My 1st website</title></head>
<body>
<h1>My First Heading</h1>
My first paragraph.
</body>
</html>

Example Explained: The text between <html> and </html> describes the web page.The text between <body> and </body> is the visible page content. The text between <h1> and </h1> is displayed ....as a heading . The text between and is displayed as a paragraph.

The HTML head Element: The head element is a container for all the head elements. Elements inside <head> can include scripts, instruct the browser where to find style sheets, provide meta information, and more.
The following tags can be added to the head section: <title>, <base>, <link>, <meta>, <script>, and <style>.

The HTML title Element : The <title> tag defines the title of the document. The title element is required in all HTML/XHTML documents.

The title element:

- defines a title in the browser toolbar
- provides a title for the page when it is added to favorites
- displays a title for the page in search-engine results

The HTML base Element : The <base> tag specifies a default address or a default target for all links on a page:

```
<head>
<base href="http://www.csdt.co.in/" />
<base target="_blank" />
</head>
```

The HTML link Element : The <link> tag defines the relationship between a document and an external resource.The <link> tag is most used to link to style sheets:

```
<head>
<link rel="stylesheet" type="text/css" href="mystyle.css" />
</head>
```

The HTML style Element : The <style> tag is used to define style information for an HTML document. Inside the style element you specify how HTML elements should render in a browser:

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

The HTML meta Element : The <meta> tag provides metadata about the HTML document. Metadata is information about data. Metadata will not be displayed on the page, but will be machine parsable.

Meta elements are typically used to specify page description, keywords, author of the document, last modified, and other metadata. The <meta> tag always goes inside the head element. The metadata can be used by browsers (how to display content or reload page), search engines (keywords), or other web services.

Keywords for Search Engines: Some search engines will use the name and content attributes of the meta element to index your pages.
The following meta element defines a description of a page:
<meta name="description" content=" Web tutorials on HTML, CSS, XML" />
The following meta element defines keywords for a page:
<meta name="keywords" content="HTML, CSS, XML" />
The intention of the name and content attributes is to describe the content of a page.

Note: A lot of webmasters have used <meta> tags for spamming, like repeating keywords (or using wrong keywords) for higher ranking. Therefore, most search engines have stopped using <meta> tags to index/rank pages.

The HTML script Element : The <script> tag is used to define a client-side script, such as a JavaScript.

HTML head Elements

Tag	Description
<head>	Defines information about the document
<title>	Defines the title of a document
<base />	Defines a default address or a default target for all links on a page
<link />	Defines the relationship between a document and an external resource
<meta />	Defines metadata about an HTML document
<script>	Defines a client-side script
<style>	Defines style information for a document

HTML Headings : HTML headings are defined with the <h1> to <h6> tags. Use HTML headings for headings only. Don't use headings to make text BIG or bold. Search engines use your headings to index the structure and content of your web pages. Since users may skim your pages by its headings, it is important to use headings to show the document structure. Example: <h1>This is a heading</h1>
<h2>This is a heading</h2>
<h3>This is a heading</h3>
Note: Browsers automatically add some empty space (a margin) before and after each heading. H1 headings should be used as main headings, followed by H2 headings, then the less important H3 headings, and so on.

HTML Paragraphs:HTML paragraphs are defined with the tag.
Example: This is a paragraph.
This is another paragraph.

HTML Elements: An HTML element is everything from the start tag to the end tag:

Start tag *	Element content	End tag *
	This is a paragraph	
	This is a link	

* The start tag is often called the opening tag. The end tag is often called the closing tag.

HTML Element Syntax : HTML element starts with a start tag / opening tag. An HTML element ends with an end tag / closing tag. The element content is everything between the start and the end tag. Some HTML elements have empty content. Empty elements are closed in the start tag.Most HTML elements can have attributes.

Nested HTML Elements :Most HTML elements can be nested (can contain other HTML elements).HTML documents consist of nested HTML elements.

HTML Document Example: <html><body> This is my first paragraph. </body></html>
The example above contains 3 HTML elements.

HTML Example Explained :
The element: This is my first paragraph.
The element defines a paragraph in the HTML document. The element has a start tag and an end tag .The element content is: This is my first paragraph.

The <body> element: <body> This is my first paragraph. </body> . The <body> element defines the body of the HTML document. The element has a start tag <body> and an end tag </body>.The element content is another HTML element (a p element).

The <html> element: <html><body> This is my first paragraph. </body></html>
The <html> element defines the whole HTML document. The element has a start tag <html> and an end tag </html>.The element content is another HTML element (the body element).

Don't Forget the End Tag: Some HTML elements might display correctly even if you forget the end tag: This is a paragraph .The example above works in most browsers, because the closing tag is considered optional.Never rely on this. Many HTML elements will produce unexpected results and/or errors if you forget the end tag .

Empty HTML Elements: HTML elements with no content are called empty elements.
 is an empty element without a closing tag (the
 tag defines a line break).

Tip: In XHTML, all elements must be closed. Adding a slash inside the start tag, like
, is the proper way of closing empty elements in XHTML (and XML).

NOTE : HTML tags are not case sensitive: <P> means the same as . Many web sites use uppercase HTML tags. CSDT use lowercase tags because the World Wide Web Consortium (W3C) recommends lowercase in HTML 4, and demands lowercase tags in XHTML.

HTML Attributes:
HTML elements can have attributes. Attributes provide additional information about an element. Attributes are always specified in the start tag. Attributes come in name/value pairs like: name="value".

Attribute Example:
HTML links are defined with the <a> tag. The link address is specified in the href attribute:

This is a link

NOTE : Always Quote Attribute Values. Attribute values should always be enclosed in quotes. Double style quotes are the most common, but single style quotes are also allowed.

HTML Attributes Reference:
Below is a list of some attributes that are standard for most HTML elements:

Attribute	Value	Description
Class	*Classname*	Specifies a classname for an element
Id	*Id*	Specifies a unique id for an element
Style	*style_definition*	Specifies an inline style for an element
Title	*tooltip_text*	Specifies extra information about an element (displayed as a tool tip)

HTML Lines : The <hr /> tag creates a horizontal line in an HTML page.The hr element can be used to separate content: Example : This is a paragraph <hr /> This is a paragraph <hr /> This is a paragraph

HTML Comments : Comments can be inserted into the HTML code to make it more readable and understandable. Comments are ignored by the browser and are not displayed . Comments are written like this: Example :
<!-- This is a comment -->
Note: There is an exclamation point after the opening bracket, but not before the closing bracket.

HTML Hyperlinks (Links) : A hyperlink (or link) is a word, group of words, or image that you can click on to jump to a new document or a new section within the current document. When you move the cursor over a link in a Web page, the arrow will turn into a little hand. Links are specified in HTML using the <a> tag.
The <a> tag can be used in two ways:

1. To create a link to another document, by using the href attribute.
2. To create a bookmark inside a document, by using the name attribute.

HTML Link Syntax : The HTML code for a link is simple. It looks like this:
Link text .
The href attribute specifies the destination of a link. Example:

Visit csdt

Tip: The "*Link text*" doesn't have to be text. It can be an image or any other HTML element.

HTML Links - The target Attribute :The target attribute specifies where to open the linked document. The example below will open the linked document in a new browser window or a new tab:
Example: Visit CSDT

HTML Links - The name Attribute :The name attribute specifies the name of an anchor. The name attribute is used to create a bookmark inside an HTML document.

Note: The upcoming HTML5 standard suggests using the id attribute instead of the name attribute for specifying the name of an anchor. Using the id attribute actually works also for HTML4 in all modern browsers. Bookmarks are not displayed in any special way. They are invisible to the reader.
Example: A named anchor inside an HTML document: Useful Tips Section

Create a link to the "Useful Tips Section" inside the same document: Visit the Useful Tips Section Or, create a link to the "Useful Tips Section" from another page: Visit the Useful Tips Section

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.
Syntax for defining an image :
The URL points to the location where the image is stored. An image named "boat.gif", located in the "images" directory on "www.csdt.co.in" has the URL: http://www.csdt.com/images/boat.gif.The browser displays the image where the <img> tag occurs in the document. If you put an image tag between two paragraphs, the browser shows the first paragraph, then the image, and then the second paragraph.

HTML Images - The Alt Attribute : The required alt attribute specifies an alternate text for an image, if the image cannot be displayed.The value of the alt attribute is an author-defined text:

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).

HTML Images - Set Height and Width of an Image: The height and width attributes are used to specify the height and width of an image.The attribute values are specified in pixels by default:

Tip: It is a good practice to specify both the height and width attributes for an image. If these attributes are set, the space required for the image is reserved when the page is loaded. However, without these attributes, the browser does not know the size of the image. The effect will be that the page layout will change during loading (while the images load).

Basic Notes - Useful Tips
Note: If an HTML file contains ten images - eleven files are required to display the page right. Loading images takes time, so my best advice is: Use images carefully.
Note: When a web page is loaded, it is the browser, at that moment, that actually gets the image from a web server and inserts it into the page. Therefore, make sure that the images actually stay in the same spot in relation to the web page, otherwise your visitors will get a broken link icon. The broken link icon is shown if the browser cannot find the image.

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.

Table Example:

```
<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>
```

How the HTML code above looks in a browser:

row 1, cell 1	row 1, cell 2
row 2, cell 1	row 2, cell 2

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. To display a table with borders, specify the border attribute:

```
<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 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>
```

How the HTML code above looks in your browser:

Header 1	Header 2
row 1, cell 1	row 1, cell 2
row 2, cell 1	row 2, cell 2

<caption> Tag: The <caption> tag defines a table caption. The <caption> tag must be inserted immediately after the <table> tag.You can specify only one caption per table.
Notes: By default, the table caption will be center-aligned above a table. However, the CSS properties"text-align" and "caption-side" can be used to align and place the caption.
Example :A table with a caption:

```
<table border="1">
 <caption>Monthly savings</caption>
 <tr>
  <th>Month</th>
  <th>Savings</th>
 </tr>
 <tr>
  <td>January</td>
  <td>$100</td>
```

```
 </tr>
</table>
```

HTML Layouts: Web page layout is very important to make your website look good. Design your webpage layout very carefully.

Website Layouts: Most websites have put their content in multiple columns (formatted like a magazine or newspaper). Multiple columns are created by using <table> or tags. Some CSS are normally also added to position elements, or to create backgrounds or colorful look for the pages.

HTML Layouts - Using Tables : The simplest way of creating layouts is by using the HTML <table> tag.The following example uses a table with 3 rows and 2 columns - the first and last row spans both columns using the colspan attribute:

Example

```
<html>
<body>
<table width="500" border="0">
<tr>
<td colspan="2" style="background-color:#FFA500;">
<h1>Main Title of Web Page</h1>
</td>
</tr>
<tr valign="top">
<td style="background-color:#FFD700;width:100px;text-align:top;">
<b>Menu</b><br />
HTML<br />
CSS<br />
JavaScript
</td>
<td style="background-color:#EEEEEE;height:200px;width:400px;text-align:top;">
Content goes here</td>
</tr>
<tr>
<td colspan="2" style="background-color:#FFA500;text-align:center;">
Copyright © 2011 csdt.co.in </td>
</tr>
</table>
</body>
</html>
```

The HTML code above will produce the following result:

Main Title of Web Page

Menu
HTML
CSS
JavaScript

Content goes here

Copyright © 2011 csdt.co.in

Note: Even though it is possible to create nice layouts with HTML tables, tables were designed for presenting tabular data - NOT as a layout tool!

HTML Layouts - Using Div Elements : The div element is a block level element used for grouping HTML elements.The following example uses five div elements to create a multiple column layout, creating the same result as in the previous example:

Example

```
<html>
<body>
<div id="container" style="width:500px">
<div id="header" style="background-color:#FFA500;">
<h1 style="margin-bottom:0;">Main Title of Web Page</h1></div>
<div id="menu" style="background-color:#FFD700;height:200px;width:100px;float:left;">
<b>Menu</b><br />HTML<br />CSS<br /> JavaScript</div>
<div id="content" style="background-color:#EEEEEE;height:200px;width:400px;float:left;">
Content goes here</div>
<div id="footer" style="background-color:#FFA500;clear:both;text-align:center;">
Copyright © 2011 csdt.co.in</div>
</div>
</body></html>
```

The HTML code above will produce the following result:

Main Title of Web Page

Menu
HTML
CSS
JavaScript

Content goes here

Copyright © 2011 csdt.co.in

The most common HTML lists are ordered and unordered lists:

An ordered list:

1. The first list item
2. The second list item
3. The third list item

An unordered list:

- List item
- List item
- List item

HTML Unordered Lists : An unordered list starts with the <ul> tag. Each list item starts with the <li> tag.The list items are marked with bullets (typically small black circles).

```
<ul><li>Coffee</li><li>Milk</li></ul>
```

How the HTML code above looks in a browser:

- Coffee
- Milk

HTML Ordered Lists : An ordered list starts with the <ol> tag. Each list item starts with the <li> tag.The list items are marked with numbers.

```
<ol> <li>Coffee</li><li>Milk</li></ol>
```

How the HTML code above looks in a browser:

1. Coffee
2. Milk

HTML Definition Lists:A definition list is a list of items, with a description of each item.The <dl> tag defines a definition list.The <dl> tag is used in conjunction with <dt> (defines the item in the list) and <dd> (describes the item in the list):

```
<dl>    <dt>Coffee</dt><dd>- black hot drink</dd>
        <dt>Milk</dt><dd>- white cold drink</dd>
</dl>
```

How the HTML code above looks in a browser:

Coffee

- black hot drink

Milk

- white cold drink

Basic Notes - Useful Tips

Tip: Inside a list item you can put text, line breaks, images, links, other lists, etc.

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.The most used input types are described below.

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>
```

How the HTML code above looks in a browser:

First name: Last name:

Note: The form itself is not visible. Also note that the default width of a text field is 20 characters.

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

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

How the HTML code above looks in a browser:

Password:

Note: The characters in a password field are masked (shown as asterisks or circles).

Radio Buttons : <input type="radio" /> defines a radio button. Radio buttons let a user select ONLY 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>
```

How the HTML code above looks in a browser:

Male Female

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>
```

How the HTML code above looks in a browser:

I have a bike I have a car

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>
```

How the HTML code above looks in a browser:

Username: Submit

If you type some characters in the text field above, and click the "Submit" button, the browser will send your input to a page called "html_form_action.asp". The page will show you the received input.

HTML Form Tags

Tag	Description
<form>	Defines an HTML form for user input
<input />	Defines an input control
<textarea>	Defines a multi-line text input control
<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
<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
<button>	Defines a push button

<fieldset> Tag: The <fieldset> tag is used to group related elements in a form.The <fieldset> tag draws a box around the related elements. The <fieldset> tag is supported in all major browsers. The <legend> tag defines a caption for the <fieldset> element.

Example :Group related elements in a form:

```
        <form>
 <fieldset>
  <legend>Personalia:</legend>
  Name: <input type="text" /><br />
  Email: <input type="text" /><br />
  Date of birth: <input type="text" />
 </fieldset>
</form>
```

<option> Tag: The <option> tag defines an option in a select list.The <option> element goes inside a <select> or <datalist> element.
Note: The <option> tag can be used without any attributes, but you usually need the value attribute, which indicates what is sent to the server.
Note: Use this tag in conjunction with <select> or <datalist> elements, elsewhere it is meaningless.
Example :A drop-down list with four options:

```
<select>
  <option value="volvo">Volvo</option>
  <option value="saab">Saab</option>
  <option value="opel">Opel</option>
  <option value="audi">Audi</option>
</select>
```

Attributes

Attribute	Value	Description
disabled	disabled	Specifies that an option should be disabled
label	*text*	Specifies a shorter label for an option
selected	selected	Specifies that an option should be pre-selected when the page loads
value	*text*	Specifies the value to be sent to a server

<optgroup> tag: The <optgroup> is used to group related options in a drop-down list.If you have a long list of options, groups of related options are easier to handle for a user.
Example :Group related options with <optgroup> tags:

```
<select>
  <optgroup label="Swedish Cars">
    <option value="volvo">Volvo</option>
    <option value="saab">Saab</option>
  </optgroup>
  <optgroup label="German Cars">
    <option value="vw">VW</option>
    <option value="audi">Audi</option>
  </optgroup>
</select>
```

Attributes

Attribute	Value	Description
label	*text*	Specifies a label for an option-group

disabled	disabled	Specifies that an option-group should be disabled

---

HTML Doctypes : A doctype declaration refers to the rules for the markup language, so that the browsers render the content correctly. 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.

Tip: Always add a doctype to your pages. This helps the browsers to render the page correctly!

Example

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Title of the document</title>
</head>
<body>
The content of the document……
</body>
</html>
```

==========

=== Some html tags examples:

```
<MENU>
<LI type="circle">The first item in the list.
<LI type="square">The second item.
<LI type="disc">And the last item.
</MENU>
```

In Use Example.

Here's a simple list :

- The first item in the list.
- The second item.
- And the last item.

<S></S>
Code.
This is the new craze where <s>you can strike out</s> text *smiles*
In Use Example.
This is the new craze where ~~you can strike out~~ text *smiles*

<STRONG></STRONG>
Code.
This tag gives your text a strong look.
In Use Example.
This tag gives **your text** a strong look.

<SPAN></SPAN>
Code.
 we will highlight this text.
In Use Example.
We will highlight this text.

<SUB>
Code.
The <SUB> element lets you specify elements as B₂H₁T₃.
In Use Example. The <SUB> element lets you specify elements as $B_2H_1T_3$.

<SUP>
Code. This <SUP> tag lets you specify a trademark ^(TM) correctly.
In Use Example.
This tag lets you specify a trademark $^{(TM)}$ correctly.

HTML Frames : With frames, you can display more than one HTML document in the same browser window. Each HTML document is called a frame, and each frame is independent of the others.
The disadvantages of using frames are:

- Frames are not expected to be supported in future versions of HTML
- Frames are difficult to use. (Printing the entire page is difficult).
- The web developer must keep track of more HTML documents.

The HTML frameset Element : The frameset element holds one or more frame elements. Each frame element can hold a separate document. The frameset element states HOW MANY columns or rows there will be in the frameset, and HOW MUCH percentage/pixels of space will occupy each of them.

The HTML frame Element : The <frame> tag defines one particular window (frame) within a frameset. In the example below we have 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>
```

HTML Frame Tags

Tag	Description
<frameset>	Defines a set of frames
<frame />	Defines a sub window (a frame)
<noframes>	Defines a no frame section for browsers that do not handle frames

HTML Iframes : An iframe is used to display a web page within a web page.
Syntax for adding an iframe: <iframe src="*URL*"></iframe>
The URL points to the location of the separate page. 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%").
Example: <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:

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

Example : <iframe src="demo_iframe.htm" name="iframe_a"></iframe>
csdt.com

HTML Colors : Colors are displayed combining RED, GREEN, and BLUE light.

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.

Color Values

Color	Color HEX	Color RGB
	#000000	rgb(0,0,0)
	#FF0000	rgb(255,0,0)
	#00FF00	rgb(0,255,0)
	#0000FF	rgb(0,0,255)
	#FFFF00	rgb(255,255,0)
	#00FFFF	rgb(0,255,255)
	#FF00FF	rgb(255,0,255)

	#C0C0C0	rgb(192,192,192)
	#FFFFFF	rgb(255,255,255)

16 Million Different Colors. The combination of Red, Green, and Blue values from 0 to 255, gives more than 16 million different colors (256 x 256 x 256). Shades of Gray colors are created by using an equal amount of power to all of the light sources

==========

HTML Basic Document :

```
<html><head><title>Title of document goes here</title></head>
<body>Visible text goes here...</body> </html>
```

Heading Elements:

```
<h1>Largest Heading</h1> <h2> . . . </h2><h3> . . . </h3><h4> . . . </h4> <h5> . . .
</h5><h6>Smallest Heading</h6>
```

Text Elements

```
<p>This is a paragraph</p>
<br /> (line break)
<hr /> (horizontal rule)
<pre>This text is preformatted</pre>
```

Logical Styles

```
<em>This text is emphasized</em>
<strong>This text is strong</strong>
<code>This is some computer code</code>
```

Physical Styles

```
<b>This text is bold</b>
<i>This text is italic</i>
```

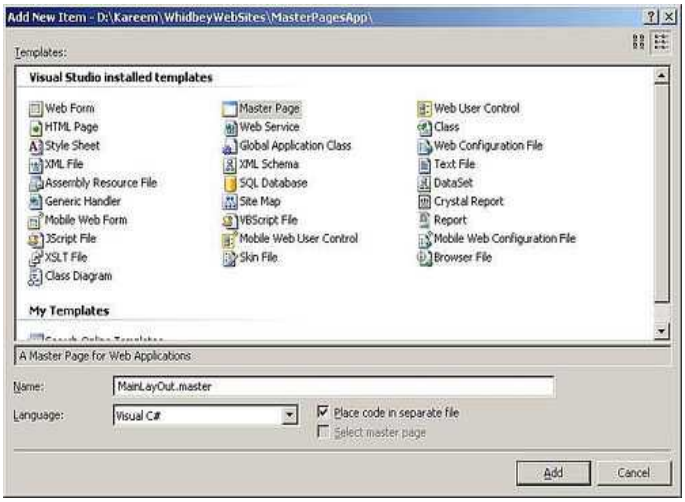
Master Pages : Master pages allow you to create a consistent look and behavior for all the pages (or group of pages) in your web application.

A master page provides a template for other pages, with shared layout and functionality. The master page defines placeholders for the content, which can be overridden by content pages. The output result is a combination of the master page and the content page. The content pages contains the content you want to display.

When users request the content page, ASP.NET merges the pages to produce output that combines the layout of the master page with the content of the content page.

Creating Master Page

- Open Visual Studio 2008.
- Create new ASP.NET Web site, enter name as MasterPagesApp.
- Select Location as File System, and enter the path as *D:\WhidbeyWebSites\MasterPagesApp*(or any other path).
- Select the Language you are more comfortable with, for this we will use C#.
- After creating the new Web Site, you will find a page created for you named *DefaulIt.aspx*.
- Delete this page and right click the application from solution explorer. Select Add New Item, select Master Page from installed Visual Studio templates and name it MainLayout, and check the Place code in separate file checkbox; this will create you a code-behind in the language you select, we work here using C#.



- The *MainLayOut.master* will be open in Source view. You will find this master page directive at the beginning:

```
<%@ Master AutoEventWireup="true" CodeFile="MainLayOut.master.cs"
  Inherits="MainLayOut" Language="C#" >
```

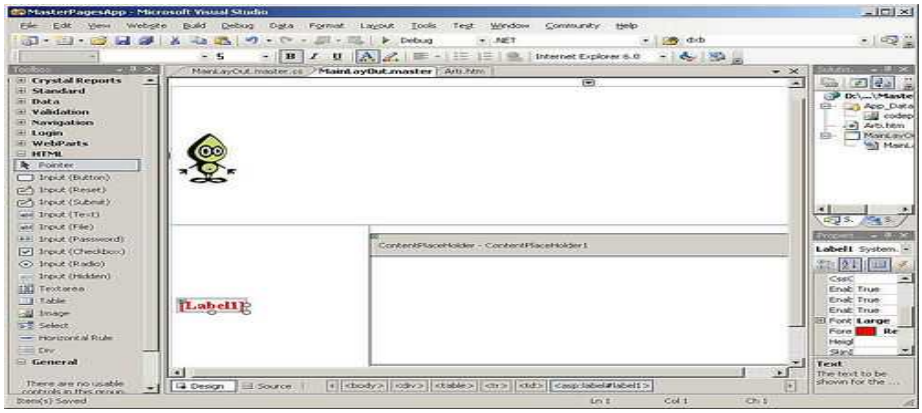
As you see, the attributes of the Master directive is common to the ones of the Page directive, and they are self-descriptive; CodeFile is the path of the code behind file. It can be VB or C#, note that in one Web Site you can mix between both languages. Inherits decides the class within the code file to be used. Language is the language of the code behind file. AutoEventWireup is so important; for any page there is an automatic way to bind the events to methods in the same aspx file or in code behind, if this attribute is true (default value is true so if it's not mentioned, it will be true). Page events are automatically bound to methods that are using naming convention as Page_event, for example Page_Load, Page_Init, and Page_PreInit (this is the event fired before creating the controls of the page). This has one disadvantage that the standard events of the page should adhere to this naming convention, but if you set it to false, it will give you more flexibility to use any names for the event handlers, here we should remember the importance of Handles keyword in VB.NET.

You should also examine the following section:

```
<asp:contentplaceholder id="ContentPlaceHolder1"
    runat="server"></asp:contentplaceholder>
```

This server control is the most important for the master pages as this is the zone in which the content pages will be rendered.

- Switch to the Design of the master page, you will get the ContentPlaceHolder1 shown, now you can design the master page just as any aspx page, so for this practice we'll do a simple master page.
- From Layout menu select Insert Table, Select Template option, then select Header , footer and side, then drag the ContentPlaceHolder1 control into the right middle cell of the table you have just added.
- Add one label into the left middle cell, and the logo of CodeProject to the upper cell of the table, you should have the same as in the following figure



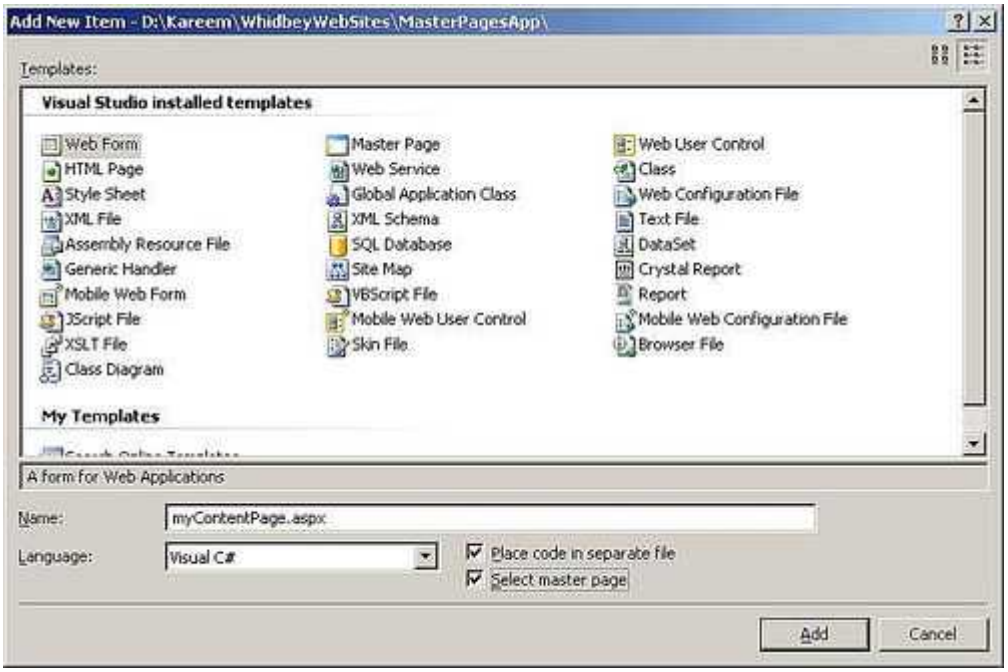
- Note: You should make all the paths in the master page relative to one folder because the image path will be relative to the rendered content page not for the master page. It's advisable to create one images folder to hold all your images, for example the logo of CodeProject image path is : *./images/codeprojectlogo.JPG*. When the content page renders, the path will be evaluated to *http://localhost:4843/MasterPagesApp/images/codeprojectlogo.JPG*.
- Double click anywhere at the master page. This will open the code behind and will add the Page_Load event handler. You can write the following code:

```
Label1.Text = "The Time of Server is: " + DateTime.Now.TimeOfDay;
```

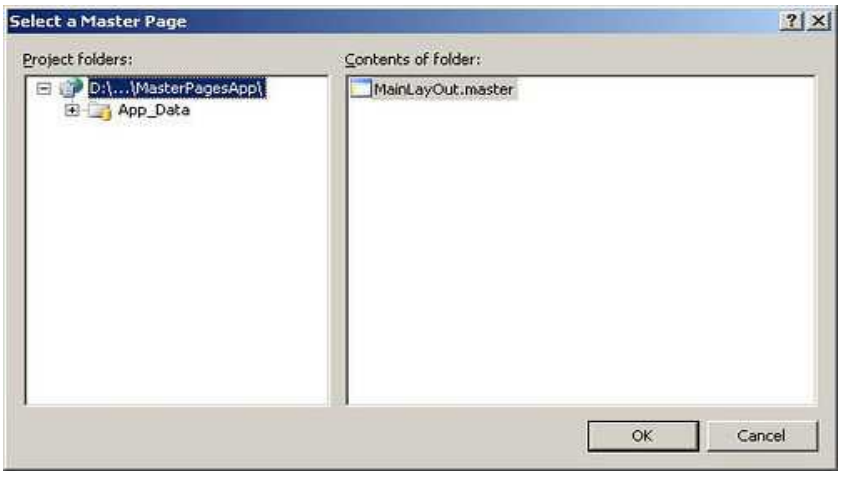
This will show the time of the server at the label.

Creating Content Page

You should know that the content page is just one aspx page but you bind this page to one master page while you are creating it. To do this, right click your web application from solution explorer and select Add New Item, select Web Form from the installed templates and check Select Maser Page, enter the name of the new page as *myContentPage.aspx* as in the following figure:



Because you have checked the Select Master Page box, you will get the following dialog to select one master page, this will bind the new aspx page to the selected master page as shown in the following figure:



The page you have just added will open in source view and you will find one entry. Add the MasterPageFile attribute set to the value of the master page file you have selected before. This should be assigned the value ~/MainLayOut.master. You will also find one Content server control added by

default and the attribute ContentPlaceHolderID assigned to ContentPlaceHolder1. This is the default value which refers to the ContentPlaceHolder control at the master page and if you have renamed this control at the master page, you should change it now to the correct ID. This sets the zone in which this content page will be shown as we said before.

Switch to the design view of *myContentPage.aspx*. You will get the following figure which shows all the master page contents added by all the contents of the master page is dim because they are not editable, only the Content1 will be enabled if you click the white area, and you can now add any controls just as you do for any aspx page.

Double click the white area of Content1, you will get the event handler of Page_Load, add the following line: Label1.Text = this.MasterPageFile;

This will show the path of the Master Page file into the label. Actually you can get a reference to the Master Page from a content page by using the Master property which refers to the page's master page, furthermore you can refer to any control on the master page by using this.Master.FindControl(string id), this method takes the Id of the control on the master page and returns Control object, so you should cast to the data type of the control.

Saving Master Page path in web.config

There is one technique by which you can make all the pages comprising your web application as content pages and you can bind all the pages to one master page, this master page that will be a template to your entire application. It's advisable to save the path of this master page into *web.config*, you can do that using the following:

```
<configuration>

 <pages masterpagefile="~/sitetemplate.master">

</configuration>
```

If you specify a MasterPageFile for your page, it will override your *web.config* value but the importance of the value of *web.config* is that it guarantees that all the pages added to your application are bound to this master page file, so if you have more than one master page, you can just change the *web.config* value, and this will update the whole application pages with no need to recompile. Indeed this technique was a trend at ASP.NET 1.x, as some developers were making all the pages as user controls and pass the id of the user control at the required page URL.

Complete list of HTML tags : Below is a complete list of HTML tags from the HTML 4.01 specification. The HTML tags are listed alphabetically to help you quickly find the tag you're looking for (or to find out whether it exists or not!).

DTD: indicates in which HTML 4.01 / XHTML 1.0 DTD the tag is allowed. S=Strict, T=Transitional, and F=Frameset

Tag	Description	DTD
Basic		
<!DOCTYPE>	Defines the document type	STF
<html>	Defines an HTML document	STF
<body>	Defines the document's body	STF
<h1> to <h6>	Defines HTML headings	STF
<p>	Defines a paragraph	STF
 	Inserts a single line break	STF
<hr />	Defines a horizontal line	STF
<!--...-->	Defines a comment	STF
Formatting		
<acronym>	Defines an acronym	STF
<abbr>	Defines an abbreviation	STF
<address>	Defines contact information for the author/owner of a document	STF
<b>	Defines bold text	STF
<bdo>	Overrides the current text direction	STF
<big>	Defines big text	STF
<blockquote>	Defines a long quotation	STF
	Deprecated. Defines centered text	TF
<cite>	Defines a citation	STF
<code>	Defines a piece of computer code	STF
<del>	Defines text that has been deleted from a document	STF
<dfn>	Defines a definition term	STF
	Defines emphasized text	STF
<font>	Deprecated. Defines font, color, and size for text	TF
<i>	Defines italic text	STF
<ins>	Defines text that has been inserted into a document	STF
<kbd>	Defines keyboard input	STF
<pre>	Defines preformatted text	STF
<q>	Defines a short quotation	STF
<s>	Deprecated. Defines strikethrough text	TF
<samp>	Defines sample output from a computer program	STF
<small>	Defines smaller text	STF

<strike>	Deprecated. Defines strikethrough text	TF
	Defines strong text	STF
<sub>	Defines subscripted text	STF
<sup>	Defines superscripted text	STF
<tt>	Defines teletype text	STF
<u>	Deprecated. Defines underlined text	TF
<var>	Defines a variable	STF
<xmp>	Deprecated. Defines preformatted text	
Forms		
<form>	Defines an HTML form for user input	STF
<input />	Defines an input control	STF
<textarea>	Defines a multiline input control (text area)	STF
<button>	Defines a clickable button	STF
<select>	Defines a drop-down list	STF
<optgroup>	Defines a group of related options in a drop-down list	STF
<option>	Defines an option in a drop-down list	STF
<label>	Defines a label for an <input> element	STF
<fieldset>	Groups related elements in a form	STF
<legend>	Defines a caption for a <fieldset> element	STF
Frames		
<frame />	Defines a window (a frame) in a frameset	F
<frameset>	Defines a set of frames	F
<noframes>	Defines an alternate content for users that do not support frames	TF
<iframe>	Defines an inline frame	TF
Images		
<img />	Defines an image	STF
<map>	Defines an image-map	STF
<area />	Defines an area inside an image-map	STF
Links		
<a>	Defines an anchor	STF
<link />	Defines the relationship between a document and an external resource	STF
Lists		
	Defines an unordered list	STF
	Defines an ordered list	STF
	Defines a list item	STF
<dir>	Deprecated. Defines a directory list	TF

<dl>	Defines a definition list	STF
<dt>	Defines an item in a definition list	STF
<dd>	Defines a description of an item in a definition list	STF
<menu>	Deprecated. Defines a menu list	TF
Tables		
<table>	Defines a table	STF
<caption>	Defines a table caption	STF
<th>	Defines a header cell in a table	STF
<tr>	Defines a row in a table	STF
<td>	Defines a cell in a table	STF
<thead>	Groups the header content in a table	STF
<tbody>	Groups the body content in a table	STF
<tfoot>	Groups the footer content in a table	STF
<col />	Defines attribute values for one or more columns in a table	STF
<colgroup>	Defines a group of columns in a table for formatting	STF
Styles		
<style>	Defines style information for a document	STF
	Defines a section in a document	STF
	Defines a section in a document	STF
Meta Info		
<head>	Defines information about the document	STF
<title>	Defines the document title	STF
<meta>	Defines metadata about an HTML document	STF
<base />	Specifies the base URL/target for all relative URLs in a document	STF
<basefont />	Deprecated. Specifies a default color, size, or font for all the text in a document	TF
Programming		
<script>	Defines a client-side script	STF
<noscript>	Defines an alternate content for users that do not support client-side scripts	STF
<applet>	Deprecated. Defines an embedded applet	TF
<object>	Defines an embedded object	STF
<param />	Defines a parameter for an object	STF

2. CSS (Cascading Style Sheets)

The biggest advantage of using CSS is that, if you place the CSS code in an external style sheet, your site becomes MUCH EASIER to maintain. You can change the layout of all your pages by editing one file.

CSS :CSS stands for Cascading Style Sheets. Styles define how to display HTML elements. Styles were added to HTML 4.0 to solve a problem. External Style Sheets can save a lot of work. External Style Sheets are stored in CSS files. Styles Solved a Big Problem because HTML was never intended to contain tags for formatting a document.HTML was intended to define the content of a document, like: <h1>This is a heading</h1> This is a paragraph.

When tags like <font>, and color attributes were added to the HTML 3.2 specification, it started a nightmare for web developers. Development of large web sites, where fonts and color information were added to every single page, became a long and expensive process. To solve this problem, the World Wide Web Consortium (W3C) created CSS.

In HTML 4.0, all formatting could be removed from the HTML document, and stored in a separate CSS file. All browsers support CSS today.

. Styles are normally saved in external .css files. External style sheets enable you to change the appearance and layout of all the pages in a Web site, just by editing one single file!

CSS Syntax :A CSS rule has two main parts: a selector, and one or more declarations:

The selector is normally the HTML element you want to style.Each declaration consists of a property and a value.The property is the style attribute you want to change. Each property has a value. CSS Example: A CSS declaration always ends with a semicolon, and declaration groups are surrounded by curly brackets: p {color:red;text-align:center;}

CSS Comments:Comments are used to explain your code, and may help you when you edit the source code at a later date. Comments are ignored by browsers.A CSS comment begins with "/*", and ends with "*/", like this:

```
/*This is a comment*/
p{
text-align:center;
/*This is another comment*/
color:black;
font-family:arial;
}
```

CSS The id and class Selectors :In addition to setting a style for a HTML element, CSS allows you to specify your own selectors called "id" and "class".

The id Selector:The id selector is used to specify a style for a single, unique element.The id selector uses the id attribute of the HTML element, and is defined with a "#".The style rule below will be applied to the element with id="para1":

Example:

```
#para1{
text-align:center;
color:red;
}
```

The class Selector:The class selector is used to specify a style for a group of elements. Unlike the id selector, the class selector is most often used on several elements.This allows you to set a particular style for many HTML elements with the same class.The class selector uses the HTML class attribute,

and is defined with a "." . In the example below, all HTML elements with class="center" will be center-aligned:

Example: .center {text-align:center;}

Three Ways to Insert CSS: There are three ways of inserting a style sheet:

- External style sheet
- Internal style sheet
- Inline style

External Style Sheet:An external style sheet is ideal when the style is applied to many pages. With an external style sheet, you can change the look of an entire Web site by changing one file. Each page must link to the style sheet using the <link> tag. The <link> tag goes inside the head section:

```
<head>
<link rel="stylesheet" type="text/css" href="mystyle.css" />
</head>
```

An external style sheet can be written in any text editor. The file should not contain any html tags. Your style sheet should be saved with a .css extension. An example of a style sheet file is shown below: .hr {color:sienna;}

```
.p {margin-left:20px;}
.body {background-image:url("images/back40.gif");}
```

Internal Style Sheet:An internal style sheet should be used when a single document has a unique style. You define internal styles in the head section of an HTML page, by using the <style> tag, like this:

```
<head>
<style type="text/css">
hr {color:sienna;} p {margin-left:20px;} body {background-
image:url("images/back40.gif");}
</style>
</head>
```

Inline Styles : An inline style loses many of the advantages of style sheets by mixing content with presentation. Use this method sparingly!To use inline styles you use the style attribute in the relevant tag. The style attribute can contain any CSS property. The example shows how to change the color and the left margin of a paragraph:

```
<p style="color:sienna;margin-left:20px">This is a paragraph.</p>
```

Multiple Style Sheets :If some properties have been set for the same selector in different style sheets, the values will be inherited from the more specific style sheet.
For example, an external style sheet has these properties for the h3 selector:

```
.h3{
color:red;
```

```
text-align:left;
font-size:8pt;
}
```

And an internal style sheet has these properties for the h3 selector:

```
.h3{
text-align:right;
font-size:20pt;
}
```

If the page with the internal style sheet also links to the external style sheet the properties for h3 will be: color:red; text-align:right; font-size:20pt;

The color is inherited from the external style sheet and the text-alignment and the font-size is replaced by the internal style sheet.

Multiple Styles Will Cascade into One Styles can be specified:

- inside an HTML element
- inside the head section of an HTML page
- in an external CSS file

Tip: Even multiple external style sheets can be referenced inside a single HTML document.
Cascading order :What style will be used when there is more than one style specified for an HTML element?

Generally speaking we can say that all the styles will "cascade" into a new "virtual" style sheet by the following rules, where number four has the highest priority:

1. Browser default
2. External style sheet
3. Internal style sheet (in the head section)
4. Inline style (inside an HTML element)

So, an inline style (inside an HTML element) has the highest priority, which means that it will override a style defined inside the <head> tag, or in an external style sheet, or in a browser (a default value). Note: If the link to the external style sheet is placed after the internal style sheet in HTML <head>, the external style sheet will override the internal style sheet!

CSS Styles: Styling Backgrounds ,Styling Text ,Styling Fonts ,Styling Links ,Styling Lists ,Styling Tables,CSS Box Model ,CSS Border ,CSS Outline ,CSS Margin ,CSS Padding.

CSS Advance:

Grouping Selectors ,In style sheets there are often elements with the same style.

```
.h1{ color:green; }
```

```
.h2{ color:green;}
.p{ color:green;}
```

To minimize the code, you can group selectors.Separate each selector with a comma.In the example below we have grouped the selectors from the code above:

Example: . h1,.h2,.p{ color:green; }

Nesting Selectors :It is possible to apply a style for a selector within a selector.In the example below, one style is specified for all p elements, one style is specified for all elements with class="marked", and a third style is specified only for p elements within elements with class="marked":

```
Example: . p{
                color:blue;
                  text-align:center;
                    }
            .marked{
                      background-color:red;
                         }
                .marked p{
                 color:white;
                     }
```

CSS pseudo-classes: CSS pseudo-classes are used to add special effects to some selectors.
The syntax of pseudo-classes:

```
selector : pseudo-class
{ property :value;
}
```

CSS classes can also be used with pseudo-classes :

```
selector. class : pseudo-class { property: value ; }
```

Anchor Pseudo-classes: Links can be displayed in different ways in a CSS-supporting browser:
Example:

```
a:link {color:#FF0000;}     /* unvisited link */
a:visited {color:#00FF00;}  /* visited link */
a:hover {color:#FF00FF;}  /* mouse over link */
a:active {color:#0000FF;}  /* selected link */
```

Note: a: hover MUST come after a:link and a:visited in the CSS definition in order to be effective!!
Note: a: active MUST come after a: hover in the CSS definition in order to be effective!!
Note: Pseudo-class names are not case-sensitive.
Pseudo-classes and CSS Classes : Pseudo-classes can be combined with CSS classes :

```
a . red : visited { color:#FF0000; }
```

CSS Syntax
If the link in the example above has been visited, it will be displayed in red.

CSS Navigation Bars : Having easy-to-use navigation is important for any web site. With CSS you can transform boring HTML menus into good-looking navigation bars. Navigation Bar = List of Links
A navigation bar needs standard HTML as a base. In our examples we will build the navigation bar from a standard HTML list. A navigation bar is basically a list of links, so using the <ul> and <li> elements makes perfect sense:

Example:

```
<ul>
<li><a href="default.asp">Home</a></li>
<li><a href="news.asp">News</a></li>
<li><a href="contact.asp">Contact</a></li>
<li><a href="about.asp">About</a></li>
</ul>
```

Now let's remove the bullets and the margins and padding from the list:
Example:

```
ul{
list-style-type:none;
margin:0;
padding:0;
}
```

Example explained:

- list-style-type:none - Removes the bullets. A navigation bar does not need list markers.
- Setting margins and padding to 0 to remove browser default settings.

The code in the example above is the standard code used in both vertical, and horizontal navigation bars.
Vertical Navigation Bar:To build a vertical navigation bar we only need to style the <a> elements, in addition to the code above:
Example :

```
a{display:block;
width:60px;}
```

Example explained:

- display:block - Displaying the links as block elements makes the whole link area clickable (not just the text), and it allows us to specify the width
- width:60px - Block elements take up the full width available by default. We want to specify a 60 px width.

Note: Always specify the width for <a> elements in a vertical navigation bar. If you omit the width, IE6 can produce unexpected results.

Horizontal Navigation Bar :There are two ways to create a horizontal navigation bar.

Using inline or floating list items.Both methods work fine, but if you want the links to be the same size, you have to use the floating method.

Inline List Items : One way to build a horizontal navigation bar is to specify the <li> elements as inline, in addition to the "standard" code above:

Example :

```
li{
display: inline;
}
```

Example explained:

display: inline; - By default, elements are block elements. Here, we remove the line breaks before and after each list item, to display them on one line

Floating List Items:In the example above the links have different widths.For all the links to have an equal width, float the <li> elements and specify a width for the <a> elements:

Example:

```
li{float: left;
}
a{display: block;
width:60px;
}
```

Example explained:

float: left - use float to get block elements to slide next to each other.

display: block - Displaying the links as block elements makes the whole link area clickable (not just the text), and it allows us to specify the width. width:60px - Since block elements take up the full width available, they cannot float next to each other. We specify the width of the links to 60px.

CSS image gallery: CSS can be used to create an image gallery.

Add a description of the image here

Add a description of the image here

Add a description of the image here

Add a description of the image here

Image Gallery :The following image gallery is created with CSS:

Example:

```
<html><head>
<style type="text/css">
div.img {
 margin:2px;
 border:1px solid #0000ff;
 height:auto;
 width:auto;
 float:left;
 text-align:center;
 }
div.img img {
 display:inline;
 margin:3px;
 border:1px solid #ffffff;
 }
div.img a:hover img {
 border:1px solid #0000ff;
 }
div.desc{
 text-align:center;
 font-weight:normal;
 width:120px;
 margin:2px;
 }
</style>
</head><body>

<div class="img">
 <a target="_blank" href="klematis_big.htm">
 <img src="klematis_small.jpg" alt="Klematis" width="110" height="90" />
 </a>
 <div class="desc">Add a description of the image here</div>
</div>
<div class="img">
 <a target="_blank" href="klematis2_big.htm">
 <img src="klematis2_small.jpg" alt="Klematis" width="110" height="90" />
 </a>
 <div class="desc">Add a description of the image here</div>
</div>
<div class="img">
 <a target="_blank" href="klematis3_big.htm">
 <img src="klematis3_small.jpg" alt="Klematis" width="110" height="90" />
 </a>
 <div class="desc">Add a description of the image here</div>
</div>
<div class="img">
 <a target="_blank" href="klematis4_big.htm">
 <img src="klematis4_small.jpg" alt="Klematis" width="110" height="90" />
 </a>
```

```
<div class="desc">Add a description of the image here</div>
</div>
</body></html>
```

Creating transparent images with CSS is easy.:Example 1 - Creating a Transparent Image :The CSS3 property for transparency is opacity.First we will show you how to create a transparent image with CSS. Regular image:

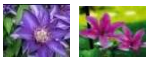
 The same image with transparency:

 Look at the following CSS:

```
img{
opacity:0.4;
filter:alpha(opacity=40); /* For IE8 and earlier */
}
```

IE9, Firefox, Chrome, Opera, and Safari use the property opacity for transparency. The opacity property can take a value from 0.0 - 1.0. A lower value makes the element more transparent.IE8 and earlier use filter :alpha(opacity=x). The x can take a value from 0 - 100. A lower value makes the element more transparent.

Example 2 - Image Transparency - Hover Effect.
Mouse over the images:



The CSS looks like this:

```
img{
opacity:0.4;
filter:alpha(opacity=40); /* For IE8 and earlier */
}
img:hover{
opacity:1.0;
filter:alpha(opacity=100); /* For IE8 and earlier */
}
```

The first CSS block is similar to the code in Example 1. In addition, we have added what should happen when a user hover over one of the images. In this case we want the image to NOT be transparent when the user hover over it. The CSS for this is: opacity=1. IE8 and earlier: filter:alpha(opacity=100). When the mouse pointer moves away from the image, the image will be transparent again.

Example 3 - Text in Transparent Box

(background image missing)This is some text that is placed in the transparent box. This is some text that is placed in the transparent box. This is some text that is placed in the transparent box. This is some text that is placed in the transparent box. This is some text that is placed in the transparent box.

The source code looks like this:

```
<html><head>
<style type="text/css">
div.background {
 width:500px;
 height:250px;
 background:url(klematis.jpg) repeat;
 border:2px solid black;
 }

div.transbox {
 width:400px;
 height:180px;
 margin:30px 50px;
 background-color:#ffffff;
 border:1px solid black;
 opacity:0.6;
 filter:alpha(opacity=60); /* For IE8 and earlier */
 }
div.transbox p {
 margin:30px 40px;
 font-weight:bold;
 color:#000000;
 }
</style>
</head><body>
<div class="background"><div class="transbox">
<p>This is some text that is placed in the transparent box.This is some text that is placed in the transparent box.This is some text that is placed in the transparent box.This is some text that is placed in the transparent box.This is some text that is placed in the transparent box.</p></div></div></body></html>
```

First, we create a div element (class="background") with a fixed height and width, a background image, and a border. Then we create a smaller div (class="transbox") inside the first div element. The "transbox" div have a fixed width, a background color, and a border - and it is transparent. Inside the transparent div, we add some text inside a p element.

3.JAVA SCRIPT

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. 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. JavaScript is valuable for adding client-side functionality to web pages. However ASP.NET programming models suggest that developers produce page layout while emitting client-side JavaScript from ASP.NET controls.

What You Should Already Know : Before you continue you should have a basic understanding of the following: HTML and CSS

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 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 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.

JavaScript = ECMAScript

JavaScript is an implementation of the ECMAScript language standard. ECMA-262 is the official JavaScript standard. JavaScript was invented by Brendan Eich at Netscape (with Navigator 2.0), and has appeared in all browsers since 1996. The official standardization was adopted by the ECMA organization (an industry standardization association) in 1997. The ECMA standard (called ECMAScript-262) was approved as an international ISO (ISO/IEC 16262) standard in 1998.The development is still in progress.

How To JavaScript

The HTML <script> tag is used to insert a JavaScript into an HTML page. Writing to The HTML Document :The example below writes a <p> element with current date information to the HTML document: Example:

```
<html> <body>
<h1>My First Web Page</h1>
```

```
<script type="text/javascript">
Var name=”shayam”
document.write("<p>" +name+ "</p>");
</script>
</body></html>
```

Note: Try to avoid using document.write() in real life JavaScript code. The entire HTML page will be overwritten if document.write() is used inside a function, or after the page is loaded. However, document.write() is an easy way to demonstrate JavaScript output in a tutorial.

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

```
Example :<html><body>
          <h1>My First Web Page</h1>
         <script type="text/javascript">
         document.write(“Welcome To JavaScript”)
        </script>
        </body> </html>
```

Examples Explained : To insert a JavaScript into an HTML page, use the <script> tag. Inside the <script> tag use the type attribute to define the scripting language. The <script> and </script> tells where the JavaScript starts and ends.

Some Browsers do Not Support JavaScript. Browsers that do not support JavaScript, will display JavaScript as page content. To prevent them from doing this, and as a part of the JavaScript standard, the HTML comment tag should be used to "hide" the JavaScript. Just add an HTML comment tag <!-- before the first JavaScript statement, and a --> (end of comment) after the last JavaScript statement, like this:

```
    <html><body>
<script type="text/javascript">
<!-- document.getElementById("demo").innerHTML=Date(); //-->
</script>
</body></html>
```

The two forward slashes at the end of comment line (//) is the JavaScript comment symbol. This prevents JavaScript from executing the --> tag.

JavaScript Where To : JavaScripts can be put in the <body> and in the <head> sections of an HTML page.

JavaScript Functions and Events : JavaScripts in an HTML page will be executed when the page loads. This is not always what we want. Sometimes we want to execute a JavaScript when an event occurs, such as when a user clicks a button. When this is the case we can put the script inside a function. Events are normally used in combination with functions (like calling a function when an event occurs).

JavaScript in <head> The example below calls a function when a button is clicked:

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

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.

Using an External JavaScript : JavaScript can also be placed in external files. External JavaScript files often contain code to be used on several different web pages. External JavaScript files have the file extension (.js).

Note: External script cannot contain the <script></script> tags!

To use an external script, point to the .js file in the "src" attribute of the <script> tag:

Example :<html><head><script type="text/javascript" src="xxx.js"></script></head>
<body>.....</body></html>

Note: Remember to place the script exactly where you normally would write the script !

JavaScript Statements : JavaScript is a sequence of statements to be executed by the browser. 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. 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.

Note: Using semicolons makes it possible to write multiple statements on one line.

JavaScript Code : JavaScript code (or just JavaScript) is a sequence of JavaScript statements. Each statement is executed by the browser in the sequence they are written. This example will write a heading and two paragraphs to a web page:

Example: <script type="text/javascript">
document.write("<h1>This is a heading</h1>");
document.write(" This is a paragraph. ");
document.write(" This is another paragraph. ");</script>

JavaScript Blocks: JavaScript statements can be grouped together in blocks.Blocks start with a left curly bracket {, and end 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:

```
Example:<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 //.The following example uses single line comments to explain the code:

```
Example:<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:

```
Example:<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>
```

Using Comments to Prevent Execution:In the following example the comment is used to prevent the execution of a single code line (can be suitable for debugging):

```
Example :<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>
```

In the following example the comment is used to prevent the execution of a code block (can be suitable for debugging):

```
Example: <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>
```

Using Comments at the End of a Line :In the following example the comment is placed at the end of a code line:

```
Example : <script type="text/javascript">
document.write("Hello"); / / Write "Hello"
document.write(" Dolly!"); / / Write " Dolly!"
</script>
```

JavaScript Variables: Variables are "containers" for storing information. 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 car name.
Rules for JavaScript variable names:

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

Note: Because JavaScript is case-sensitive, variable names are case-sensitive.
Example: A variable's value can change during the execution of a script. You can refer to a variable by its name to display or change its value.

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.

Note: When you assign a text value to a variable, put quotes around the value.
Note: 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 deleted as soon as the function is completed.

Global JavaScript Variables: Variables declared outside a function become GLOBAL, and all scripts and functions on the web page can access it.Global variables are deleted when you close the page.

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.

These statements: 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 Operators: = is used to assign values. + is used to add values.

The assignment operator = is used to assign values to JavaScript variables.The arithmetic operator + is used to add values together. y=5; z=2; x= y+z ; The value of x, after the execution of the statements above, is 7.

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:

```
txt1="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!
Example :-

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

JavaScript Comparison and Logical Operators : Comparison and Logical operators are used to test for true or false. 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 x==5 is true
===	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.
Syntax : variablename=(condition)?value1:value2

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":

```
<script type="text/javascript">
var visitor="PRES";
var greeting=(visitor=="PRES")?"Dear President ":"Dear ";
document.write(greeting);
</script>
```

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.

Syntax :
```
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! 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>
```

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.

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 neither condition1 nor condition2 is 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>
```

JavaScript Switch Statement :Conditional statements are used to perform different actions based on different conditions.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();
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.

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)
 {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.

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("<p>Hello " + name + "! How are you today?</p>");
 }
}
</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. 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.

JavaScript Function Example :
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):
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>
```

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. Syntax : for (*variable=startvalue;variable<=endvalue;variable=variable+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.
Note: The increment parameter could also be negative, and the <= could be any comparing statement. Example :<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>
```

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. Syntax: while (*variable*<=*endvalue*)

```
 {
 code to be executed
 }
```

Note: The <= could be any comparing operator.
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:
Example :<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 (variable<=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:

```
Example : <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>
```

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).

```
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 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 property.

Looping through the properties of an object :
Example :

```
var person={fname:"John",lname:"Doe",age:25};
var x;
for (x in person)
{
document.write(person[x] + " ");
}
```

JavaScript Events : Events are actions that can be detected by JavaScript.
The example below displays the date when a button is clicked:
Example:

```
<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

Note: 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 mouses over an HTML element:
Example

 Mouse over the sun and the planets and see the different descriptions.

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.

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. Syntax : 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 type 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:

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+="Error description: " + err.message + "\n\n";
 txt+="Click OK to continue.\n\n";
 alert(txt);
 }
}
</script>
</head>
<body><input type="button" value="View message" onclick="message()" /></body>
</html>
```

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:

```
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.csdt.co.in/";
  }
 }
}
</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.

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.
Syntax: 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 5, 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:
Example:

```
<html>
<body>
<script type="text/javascript">
var x=prompt("Enter a number between 5 and 10:","");
try
 {
 if(x>10)
```

```
  {
  throw "Err1";
  }
 else if(x<5)
  {
  throw "Err2";
  }
 else if(isNaN(x))
  {
  throw "Err3";
  }
 }
catch(err)
 {
 if(err=="Err1")
  {
  document.write("Error! The value is too high.");
  }
 if(err=="Err2")
  {
  document.write("Error! The value is too low.");
  }
 if(err=="Err3")
  { document.write("Error! The value is not a number.");
  }
}
</script></body>
</html>
```

Insert Special Characters: In JavaScript you can add special characters to a text string by using the backslash sign. 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 Guidelines : Some other important things to know when scripting with JavaScript.
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!");

However, you cannot break up a code line like this: document.write \ ("Hello World!");

JavaScript Objects:

JavaScript is an Object Based Programming language. An Object Based Programming language allows you to define your own objects and make your own variable types.

Object Based Programming : JavaScript is an Object Based Scripting language, and allows you to define your own objects and make your own variable types.

Note that an object is just a special kind of data. An object has properties and methods. Objects are useful to organize information. JavaScript has several built-in objects, like String, Date, Array, and more.
Let's illustrate with an example: A person is an object. Properties are the values associated with the object. The persons' properties include name, height, weight, age, skin tone, eye color, etc. All

persons have these properties, but the values of those properties will differ from person to person. Objects also have methods. Methods are the actions that can be performed on objects. The persons' methods could be eat(), sleep(), work(), play(), etc.

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>
```

The output of the code above will be: 12

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>
```

The output of the code above will be: HELLO WORLD!

JavaScript 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);
```

The code above will result in the following output: 12
The following example uses the toUpperCase() method of the String object to convert a string to uppercase letters:

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

The code above will result in the following output: HELLO WORLD!

JavaScript Date Object : The Date object is used to work with dates and times.
Return today's date and time : How to use the Date() method to get today's date.
getFullYear() : Use getFullYear() to get the year.
getTime() : getTime() returns the number of milliseconds since 01.01.1970.
setFullYear() : How to use setFullYear() to set a specific date.
toUTCString() : How to use toUTCString() to convert today's date (according to UTC) to a string.
getDay() : Use getDay() and an array to write a weekday, and not just a number.
Display a clock : How to display a clock on your web page.

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);
```

Note: 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 2100:

```
var x=new Date();
x.setFullYear(2100,0,14);
var today = new Date();
if (x>today)
 {
 alert("Today is before 14th January 2100");
 }
else
 {
 alert("Today is after 14th January 2100");
 }
```

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 a 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").

JavaScript Array Object

The Array object is used to store multiple values in a single variable.

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 : Create an array, assign values to it, and write the values to the output. 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
```

Note: 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]); will result in the following output: Saab

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]); will result in the following output: Opel

More Examples :
Join two arrays - concat()
Join three arrays - concat()
Join all elements of an array into a string - join()
Remove the last element of an array - pop()
Add new elements to the end of an array - push()
Reverse the order of the elements in an array - reverse()
Remove the first element of an array - shift()

Select elements from an array - slice()
Sort an array (alphabetically and ascending) - sort()
Sort numbers (numerically and ascending) - sort()
Sort numbers (numerically and descending) - sort()
Add an element to position 2 in an array - splice()
Convert an array to a string - toString()
Add new elements to the beginning of an array - unshift()

JavaScript Boolean Object

The Boolean object is used to convert a non-Boolean value to a Boolean value (true or false).
Examples : Check Boolean value :Check if a Boolean object is true or false.

Create a Boolean Object : The Boolean object represents two values: "true" or "false".The following code creates a Boolean object called myBoolean: var myBoolean=new Boolean();
If the Boolean object has no initial value, or if the passed value is one of the following:

- 0
- -0
- null
- ""
- false
- undefined
- NaN

the object is set to false. For any other value it is set to true (even with the string "false")!

JavaScript Math Object

The Math object allows you to perform mathematical tasks.
Examples:
round() :How to use round().
random() :How to use random() to return a random number between 0 and 1.
max() :How to use max() to return the number with the highest value of two specified numbers.
min() : How to use min() to return the number with the lowest value of two specified numbers.

Math Object :The Math object allows you to perform mathematical tasks. The Math object includes several mathematical constants and methods.
Syntax for using properties/methods of Math:
var x=Math.PI;
var y=Math.sqrt(16);
Note: Math is not a constructor. All properties and methods of Math can be called by using Math as an object without creating it.

Mathematical Constants: JavaScript provides eight mathematical constants that can be accessed from the Math object. These are: E, PI, square root of 2, square root of 1/2, natural log of 2, natural log of 10, base-2 log of E, and base-10 log of E.
You may reference these constants from your JavaScript like this:
Math.E

Math.PI
Math.SQRT2
Math.SQRT1_2
Math.LN2
Math.LN10
Math.LOG2E
Math.LOG10E

Mathematical Methods :In addition to the mathematical constants that can be accessed from the Math object there are also several methods available. The following example uses the round() method of the Math object to round a number to the nearest integer:
document.write(Math.round(4.7));

The code above will result in the following output: 5

The following example uses the random() method of the Math object to return a random number between 0 and 1:document.write(Math.random()); The code above can result in the following output: 0.6872347085736692

The following example uses the floor() and random() methods of the Math object to return a random number between 0 and 10: document.write(Math.floor(Math.random()*11));

The code above can result in the following output: 0

JavaScript RegExp Object

RegExp, is short for regular expression.
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).
Example 1: Do a case-insensitive search for "w3schools" in a string:
var str="Visit W3Schools";
var patt1=/w3schools/i;
Example 2. Do a global search for "is":
var str="Is this all there is?";
var patt1=/is/g;

Example 3. Do a global, case-insensitive search for "is":
var str="Is this all there is?";
var patt1=/is/gi;

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":
Example:
var patt1=new RegExp("e");
document.write(patt1.test("The best things in life are free"));
Since there is an "e" in the string, the output of the code above will be: true

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":
Example 1.
var patt1=new RegExp("e");
document.write(patt1.exec("The best things in life are free"));
Since there is an "e" in the string, the output of the code above will be: e

JS Advanced

JavaScript Browser Detection: The Navigator object contains information about the visitor's browser. 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.

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

The Navigator Object: The Navigator object contains all information about the visitor's browser:

Example:

```
<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>
```

JavaScript Cookies : A cookie is often used to identify a user. 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
- 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: Example: <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:

```
function validateForm()
{
var x=document.forms["myForm"]["fname"].value;
if (x==null || x=="")
 {
 alert("First name must be filled out");
 return false;
 }
}
```

The function above could be called when a form is submitted:

```
Example :<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>
```

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:

```
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;
 }
}
```

The function above could be called when a form is submitted:

Example :

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

JavaScript Timing Events:

JavaScript can be executed in time-intervals. This is called 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()

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

The setTimeout() Method

Syntax : 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.

Note: There are 1000 milliseconds in one second.

Example:When the button is clicked in the example below, an alert box will be displayed after 3 seconds.

Example:

```
<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:

Example:

```
<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();
 }
}
</script></head>

<body>
<form>
<input type="button" value="Start count!" onclick="doTimer()">
<input type="text" id="txt" />
</form>
</body>
</html>
```

The clearTimeout() Method :

Syntax :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:

Example :

```
<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>
```

Now You Know JavaScript, What's Next?
The next step is to learn about the HTML DOM, jQuery, and AJAX. If you want to learn about server-side scripting, the next step is to learn ASP or PHP.
HTML DOM:The HTML DOM defines a standard way for accessing and manipulating HTML documents.The HTML DOM is platform and language independent and can be used by any programming language like Java, JavaScript, and VBScript.

jQuery:jQuery is a JavaScript Library.jQuery greatly simplifies JavaScript programming.

AJAX: AJAX = Asynchronous JavaScript and XML.AJAX is not a new programming language, but a new way to use existing standards.AJAX is about exchanging data with a server, and update parts of a web page - without reloading the whole page.Examples of applications using AJAX: Google Maps, Gmail, Youtube, and Facebook tabs.
ASP / PHP :While scripts in an HTML file are executed on the client (in the browser), scripts in an ASP/PHP file are executed on the server. With ASP/PHP you can dynamically edit, change or add any content of a Web page, respond to data submitted from HTML forms, access any data or databases and return the results to a browser, customize a Web page to make it more useful for individual users. Since ASP/PHP files are returned as plain HTML, they can be viewed in any browser.

4. JQuary

jQuery is a fast and concise JavaScript Library created by John Resig in 2006 with a nice motto: Write less, do more. jQuery simplifies HTML document traversing, event handling, animating, and Ajax interactions for rapid web development.

jQuery is a JavaScript toolkit designed to simplify various tasks by writing less code. Here is the list of important core features supported by jQuery:

- DOM manipulation: The jQuery made it easy to select DOM elements, traverse them and modifying their content by using cross-browser open source selector engine called Sizzle.
- Event handling: The jQuery offers an elegant way to capture a wide variety of events, such as a user clicking on a link, without the need to clutter the HTML code itself with event handlers.
- AJAX Support: The jQuery helps you a lot to develop a responsive and feature-rich site using AJAX technology.
- Animations: The jQuery comes with plenty of built-in animation effects which you can use in your websites.
- Lightweight: The jQuery is very lightweight library - about 19KB in size (Minified and gzipped).
- Cross Browser Support: The jQuery has cross-browser support, and works well in IE 6.0+, FF 2.0+, Safari 3.0+, Chrome and Opera 9.0+
- Latest Technology: The jQuery supports CSS3 selectors and basic XPath syntax.

How to install jQuery ?

This is very simple to do require setup to use jQuery library. You have to carry two simple steps:

1. Go to the download page to grab the latest version available.
2. Now put downloaded jquery-1.3.2.min.js file in a directory of your website, e.g. /jquery.

The downloaded file name jquery-1.3.2.min.js may vary for your version. Your minified version would be kind of unreadable which would not have any new line or unnecessary words in it. The jQuery does not require any special installation and very similar to JavaScript, we do not need any compilation or build phase to use jQuery.

How to use jQuery library?

Now you can include *jquery* library in your HTML file as follows:

```
<html>
<head>
<title>The jQuery Example</title>
  <script type="text/javascript"
  src="/jquery/jquery-1.3.2.min.js"></script>
  <script type="text/javascript">
    // you can add our javascript code here
  </script>
</head>
<body>
```

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

How to call a jQuery library functions?

As almost everything we do when using jQuery reads or manipulates the document object model (DOM), we need to make sure that we start adding events etc. as soon as the DOM is ready.

If you want an event to work on your page, you should call it inside the $(document).ready() function. Everything inside it will load as soon as the DOM is loaded and before the page contents are loaded. To do this, we register a ready event for the document as follows:

```
$(document).ready(function() {
  // do stuff when DOM is ready
});
```

To call upon any jQuery library function, use HTML script tags as shown below:

```
<html>
<head>
<title>The jQuery Example</title>
  <script type="text/javascript"
  src="/jquery/jquery-1.3.2.min.js"></script>

  <script type="text/javascript" language="javascript">
  // <![CDATA[
  $(document).ready(function() {
    $("div").click(function() {
      alert("Hello world!");
    });
  });
  // ]]>
  </script>

</head>
<body>
<div id="newdiv">
Click on this to see a dialogue box.
</div>
</body>
</html>
```

How to use Custom Scripts?

It is better to write our custom code in the custom JavaScript file : custom.js, as follows:

```
/* Filename: custom.js */
$(document).ready(function() {
 $("div").click(function() {
            alert("Hello world!");
 });
});
```

Now we can include custom.js file in our HTML file as follows:

```
<html>
<head>
<title>The jQuery Example</title>
  <script type="text/javascript"
  src="/jquery/jquery-1.3.2.min.js"></script>
  <script type="text/javascript"
  src="/jquery/custom.js"></script>
</head>
<body>
<div id="newdiv">
Click on this to see a dialogue box.
</div>
</body>
</html>
```