HTML, CSS AND JAVASCRIPT NOTES

HP

[COMPANY NAME] [Company address]

HTML

Introduction to HTML

HTML stands for Hypertext Mark-up Language, and it is the most widely used language to write Web Pages.

- Hypertext refers to the way in which Web pages (HTML documents) are linked together. Thus, the link available on a webpage is called Hypertext.
- As its name suggests, HTML is a Mark-up Language which means you use HTML to simply "mark-up" a text document with tags that tell a Web browser how to structure it to display.

Originally, HTML was developed with the intent of defining the structure of documents like headings, paragraphs, lists, and so forth to facilitate the sharing of scientific information between researchers. Now, HTML is being widely used to format web pages with the help of different tags available in HTML language.

Web Application Architecture

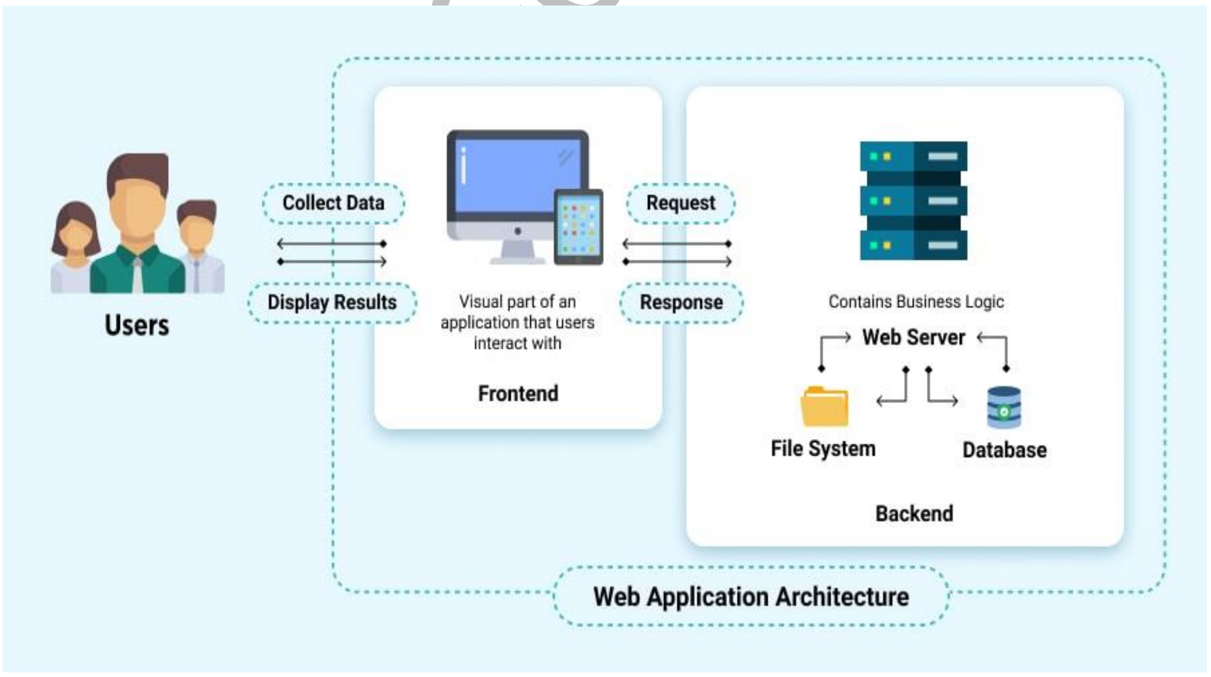


FIGURE - 1 : WEB APPLICATION ARCHITECTURE

WHAT IS WEB APPLICATION ARCHITECTURE?

"The web application architecture is a "skeleton" or layout that displays the interactions between application components, middleware systems, user interfaces, and databases. This kind of interaction allows a number of applications to work together simultaneously".

Once a user opens a webpage, the server sends specific data to the browser as a response to the user's request. To be precise, a web client (or user agent) may request web resources or more commonly-known web documents (HTML, JSON, PDF, and so on) through a web server. Then, voila — with these minimal manipulations, the requested information appears. After that, the interaction between a user and a website starts.

Basic HTML Document

Type in the following text in "NOTEPAD":

```
<!DOCTYPE html>
<html>
<head>
<title>Title of the page</title>
</head>
<body>
<h1>This is a heading</h1>
<p>Document content goes here.....</p>
</body>
</html>
```

Save the file as "mypage.html". Start your Internet browser. Select "Open" (or "Open Page") in the File menu of your browser. A dialog box will appear. Select "Browse" (or "Choose File") and locate the HTML file you just created - "mypage.html" - select it and click "Open". Now you should see an address in the dialog box, for example "C:\MyDocuments\webdesign\mypage.html". Click OK, and the browser will display the page.

HTML Tags

HTML mark-up tags are usually called HTML tags.

- HTML tags are keywords (tag names) surrounded by **angle brackets** like <html>
- HTML tags normally **come in pairs like** <p> and </p>
- The first tag in a pair is the **start tag**, the second tag is the **end tag**
- The end tag is written like the start tag, with a **slash** before the tag name
- Start and end tags are also called **opening tags** and **closing tags** content

```
<tagname>content</tagname>
```

Some of the commonly used tags are :

TAG	DISCRIPTION
<!DOCTYPE>	This tag defines the document type and HTML version.
<html>	This tag encloses the complete HTML document and mainly comprises of document header which is represented by <head>...</head> and document body which is represented by <body>...</body> tags.
<head>	This tag represents the document's header which can keep other HTML tags like <title>, <link> etc.
<title>	The <title> tag is used inside the <head> tag to mention the document title
<body>	This tag represents the document's body which keeps other HTML tags like <h1>, <div>, <p> etc.
<h1>	This tag represents the heading
<p>	This tag represents a paragraph.

Tag Attributes

- Tags can have attributes. Attributes can provide additional information about the HTML elements on your page.
- This tag defines the body element of your HTML page: **<body>.** With an added **bgcolor** attribute, you can tell the browser that the background color of your page should be red, like this: **<body bgcolor = "red">**
- This tag defines an HTML table: **<table>**. With an added border attribute, you can tell the browser that the table should have no borders: **<table border = "0">**

- Attributes always come in name/value pairs like this: **name="value"**. Attributes are always added to the start tag of an HTML element

HTML Versions

Since the early days of the web, there have been many versions of HTML:

HTML (1991), HTML+ (1993), HTML 2.0 (1995), HTML 3.2 (1997), HTML 4.01 (1999) XHTML (2000), HTML5 (2012)

The <!DOCTYPE> Declaration

- The <! DOCTYPE> declaration tag is used by the web browser to understand the version of
- the HTML used in the document.
- Current version of HTML is 5 and it makes use of the following
- declaration:

```
<!DOCTYPE html>
```

Basic HTML Tags

1) Heading Tags

Headings are defined with the <h1> to <h6> tags. <h1> defines the largest heading. <h6> defines the smallest heading.

```
<!DOCTYPE html>

<html>

<head>

<title>Heading Example</title>

</head>

<body>

<h1>This is heading 1</h1>

<h2>This is heading 2</h2>

<h3>This is heading 3</h3>

<h4>This is heading 4</h4>
```

```
<h5>This is heading 5</h5>
<h6>This is heading 6</h6>
</body>
</html>
```

This will produce the following result:

This is a heading

This is a heading

This is a heading

This is a heading

This is a heading

This is a heading

HTML automatically adds an extra blank line before and after a heading.

2) Paragraphs

The <p> tag offers a way to structure your text into different paragraphs. Each paragraph of text should go in between an opening <p> and a closing </p> tag as shown below in the example:

```
<!DOCTYPE html>
<html>
<head>
<title>Paragraph Example</title>
</head>
<body>
<p>Here is a first paragraph of text.</p>
<p>Here is a second paragraph of text.</p>
<p>Here is a third paragraph of text.</p>
</body></html>
```

This will produce the following result:

```
Here is a first paragraph of text.
Here is a second paragraph of text.
Here is a third paragraph of text.
```

3) Line Break Tag

- Whenever you use the <br /> element, anything following it starts from the next line. This tag is an example of an empty element, where you do not need opening and closing tags, as there is nothing to go in between them.
- The <br /> tag has a space between the characters br and the forward slash. If you omit this space, older browsers will have trouble rendering the line break, while if you miss the forward slash character and just use <br> it is not valid in XHTML.

```
<!DOCTYPE html>
<html>
<head>
<title>Line Break Example</title>
</head>
<body>
<p>Hello<br />
You delivered your assignment on time.<br />
Thanks<br />
Mahnaz</p>
</body>
</html>
```

This will produce the following result:

```
Hello
You delivered your assignment on time.
Thanks
Mahnaz
```

4) Horizontal Lines

Horizontal lines are used to visually break-up sections of a document. The tag creates a line from the current position in the document to the right margin and breaks the line accordingly. For example, you may want to give a line between two paragraphs as in the given example below:

```
<!DOCTYPE html>
<html>
<head>
<title>Horizontal Line Example</title>
</head>
<body>
<p>This is paragraph one and should be on top</p>
<hr />
<p>This is paragraph two and should be at bottom</p>
</body>
</html>
```

This will produce the following result:

```
This is paragraph one and should be on top
______________________________________________________________
This is paragraph two and should be at bottom
```

- Again <hr /> tag is an example of the empty element, where you do not need opening and closing tags, as there is nothing to go in between them.
- The <hr /> element has a space between the characters hr and the forward slash. If you omit this space, older browsers will have trouble rendering the horizontal line, while if you miss the forward slash character and just use <hr> it is not valid in XHTML

1) Comments in Html

The comment tag is used to insert a comment in the HTML source code. A comment will be ignored by the browser. You can use comments to explain your code, which can help you when you edit the source code at a later date.

```
<!-- This is a comment -->
```

HTML – Elements

An HTML element is defined by a starting tag. If the element contains other content, it ends with a closing tag, where the element name is preceded by a forward slash as shown below with few tags:

START TAG	CONTENT	END TAG
`<p>`	`This is paragraph content`	`</p>`
`<h1>`	`This is heading content`	`</h1>`
`<div>`	`This is division content`	`</div>`
`<br/>`		

- So here <p>....</p> is an HTML element, <h1>...</h1> is another HTML element. There are some HTML elements which don't need to be closed, such as <img.../>, <hr /> and <br /> elements. These are known as **void elements**.
- HTML documents consists of a tree of these elements and they specify how HTML documents should be built, and what kind of content should be placed in what part of an HTML document.

HTML Tag (vs) HTML Elements

- An HTML element is defined by a starting tag. If the element contains other content, it ends with a closing tag.
- For example, **<p>** is starting tag of a paragraph and **</p>** is closing tag of the same paragraph but **<p>This is paragraph</p>** is a paragraph element.

HTML - Attributes

An attribute is used to define the characteristics of an HTML element and is placed inside the element's opening tag. All attributes are made up of two parts: a name and a value:

- The name is the property you want to set. For example, the paragraph element in the example carries an attribute whose name is align, which you can use to indicate the alignment of paragraph on the page.
- The value is what you want the value of the property to be set and always put within quotations. The below example shows three possible values of align attribute: left, centre and right.

Attribute names and attribute values are case-insensitive. However, the World Wide Web Consortium (W3C) recommends lowercase attributes/attribute values in their HTML 4 recommendation.

```
<!DOCTYPE html>
<html>
<head>
<title>Align Attribute Example</title>
</head>
<body>
<p align="left">This is left aligned</p>
<p align="center">This is center aligned</p>
<p align="right">This is right aligned</p>
</body>
</html>
```

This will display the following result:

```
This is left aligned
                              This is center aligned
                                                              This is right aligned
```

Core Attributes

The four core attributes that can be used on the majority of HTML elements (although not all) are:

- Id
- Title
- Class
- Style

1) Id Attributes

The id attribute of an HTML tag can be used to uniquely identify any element within an HTML page. There are two primary reasons that you might want to use an id attribute on an element:

- If an element carries an id attribute as a unique identifier, it is possible to identify just that element and its content.
- If you have two elements of the same name within a Web page (or style sheet), you can use the id attribute to distinguish between elements that have the same name.

Example :

```
<p id="html">This para explains what is HTML</p>
<p id="css">This para explains what is Cascading Style Sheet</p>
```

2) The title Attribute

- The title attribute gives a suggested title for the element. They syntax for the **title** attribute is similar as explained for **id** attribute:
- The behaviour of this attribute will depend upon the element that carries it, although it is often displayed as a tooltip when cursor comes over the element or while the element is loading.

Example :

```
<!DOCTYPE html>
<html>
<head>
<title>The title Attribute Example</title>
</head>
<body>
<h3 title="Hello HTML!">Titled Heading Tag Example</h3>
</body>
</html>
```

This will produce the following result:

```
Titled Heading Tag Example
```

3) The class Attribute

The class attribute is used to associate an element with a style sheet, and specifies the class of element. You will learn more about the use of the class attribute when you

will learn Cascading Style Sheet (CSS). So for now you can avoid it.

The value of the attribute may also be a space-separated list of class names.

For example:

```
class="className1 className2 className3"
```

4) Style Attribute

The style attribute allows you to specify Cascading Style Sheet (CSS) rules within the element.

```
<!DOCTYPE html>
<html>
<head>
<title>The style Attribute</title>
</head>
<body>
<p style="font-family:arial; color:#FF0000;">Some text...</p>
</body>
</html>
```

This will produce the following result:

```
Some text..
```

At this point of time, we are not learning CSS, so just let's proceed without bothering much about CSS. Here, you need to understand what are HTML attributes and how they can be used while formatting content.

HTML Text Formatting

HTML defines a lot of elements for formatting output, like bold or italic text. How to View HTML Source? To find out, click the **VIEW** option in your browser's toolbar and select **SOURCE** or **PAGE SOURCE**. This will open a window that shows you the HTML code of the page

1) Text Formatting Tags

TAG	DESCRIPTION
<i>	Defines italic text. Recommend using <em>
<b>	Defines bold text. Recommend using <strong>
<em>	Defines emphasized text. Renders as italic text.
<strong>	Defines strong text. Renders as bold text.

Example :

SOURCE	OUTPUT
`<i>Italic text</i><br />`	*Italic text*
`<b>Bold text</b><br />`	**Bold text**
`<em>Emphasized text</em><br>`	*Emphasized text*
`<strong>Strong text</strong><br>`	**Strong text**

What is the difference between <i> & <em> and <b> & <strong>, <b> and <i> will both become deprecated tags. Using <strong> and <em> are the tags that will be cross-browser compatible as browsers move forward to embrace the new standards in HTML (e.g., XHTML)

2) HTML Links

HTML uses a hyperlink to link to another document on the Web. HTML uses the <a> (anchor) tag to create a link to another document. An anchor can point to any resource on the Web: an HTML page, an image, a sound file, a movie, etc.

The syntax of creating an anchor:

```
<a href="url">Text to be displayed</a>
```

The <a> tag is used to create an anchor to link from, the href attribute is used to address the document to link to, and the words between the open and close of the anchor tag will be displayed as a hyperlink.

This anchor defines a link to W3Schools:

```
<a href="http://www.google.co.in/">Google</a>
```

The line above will look like this in a browser:

Google

3) The Target Attribute

With the target attribute, you can define where the linked document will be opened. The line below will open the document in a new browser window:

```
<a href="http://www.google.co.in/" target="_blank">Google</a>
```

The Anchor Tag and the Name Attribute -

The name attribute is used to create a named anchor. When using named anchors, we can create links that can jump directly into a specific section on a page, instead of letting the user scroll around to find what he/she is looking for.

Below is the syntax of a named anchor:

```
<a name="label">Text to be displayed</a>
```

The name attribute is used to create a named anchor. The name of the anchor can be any text you care to use.

The line below defines a named anchor:

```
<a name="tips">Useful Tips Section</a>
```

You should notice that a named anchor is not displayed in a special way. To link directly to the "tips" section, add a # sign and the name of the anchor to the end of a URL, like this:

```
<a href="http://www.amyschan.com/mypage.html#tips">

Jump to the Useful Tips Section</a>
```

HTML Lists

HTML supports ordered, unordered and definition lists.

1) Unordered Lists

- An unordered list is a list of items. The list items are marked with bullets (typically small black circles).
- An unordered list starts with the <ul> tag. Each list item starts with the <li> tag.

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

Here is how it looks in a browser:

```
• Coffee
• Milk
```

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

2) Ordered lists

- An ordered list is also a list of items. The list items are marked with numbers.
- An ordered list starts with the <ol> tag. Each list item starts with the <li> tag.

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

Here is how it looks in a browser:

```
1. Coffee
2. Milk
```

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

3) Description lists

- HTML also supports description lists.
- A description list is a list of terms, with a description of each term.
- The <dl> tag defines the description list, the <dt> tag defines the term (name), and the <dd> tag describes each term:

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

Here is how it looks in a browser:

```
Coffee
       - black hot drink
Milk
       - white cold drink
```

HTML Table

- 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

HTML <table> Tag

Definition and Usage

TAG	DESCRIPTION
`<table>`	Defines a table
`<th>`	Defines a header cell in a table
`<tr>`	Defines a row in a table
`<td>`	Defines a cell in a table
`<caption>`	Defines a table caption
`<colgroup>`	Specifies a group of one or more columns in a table for formatting
`<col>`	Specifies column properties for each column within a element
`<thread>`	Groups the header content in a table
`<tbody>`	Groups the body content in a table
`<tfoot>`	Groups the footer content in a table

HTML Images

With HTML you can display images in a document.

The Image Tag and the src Attribute

In HTML, images are defined with the <img> 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 on your page.

The syntax of defining an image: **`<img src="url">`**

The URL points to the location where the image is stored. An image named "boat.gif" located in the directory "images" on "www.w3schools.com" has the URL: **http://www.w3schools.com/images/boat.gif**. The browser puts the image where the image 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.

The Alt Attribute

The alt attribute is used to define an "alternate text" for an image. The value of the alt attribute is an author-defined text:

```
<img src="images/logo.jpg" alt="Google logo">
```

The "alt" attribute tells the reader what he or she is missing on a page if the browser can't load images. The browser will then display the alternate text instead of the image. It is a good practice to include the "alt" attribute for each image on a page, to improve the display and usefulness of your document for people who have text-only browsers.

HTML Backgrounds

A good background can make a Web site look really great.

Backgrounds

The <body> tag has two attributes where you can specify backgrounds. The background can be a color or an image.

- **Bgcolor**

  The bgcolor attribute specifies a background-color for an HTML page. The value of this attribute can be a hexadecimal number, an RGB value, or a color name:

```
<body bgcolor="#000000">
<body bgcolor="rgb(0,0,0)">
<body bgcolor="black">
```

The lines above all set the background-color to black.

- **Background**

  The background attribute specifies a background-image for an HTML page. The value of this attribute is the URL of the image you want to use. If the image is smaller than the browser window, the image will repeat itself until it fills the entire browser window.

```
<body background="images/cloudswirl.jpg">
<body background="http://www.amyschan.com/images/cloudswirl.jpg">
```

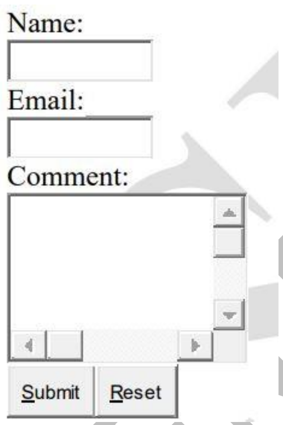
HTML FORMS

Forms can be used to collect information. Sophisticated forms may be linked to a database to store the information, but this is beyond the scope of the article. This article will show you how to create a form that emails you the information.

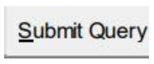
Here is a very basic email form:

```
<form action="mailto:support@pixelmill.com" method="post">
Name:<br>
<input type="text" name="name" size="20"><br>
Email:<br>
<input type="text" name="email" size="20"><br>
Comment:<br>
<textarea cols="20" rows="5"></textarea><br>
<input type="submit" value="Submit"> <input type="reset" value="Reset">
</form>
```

And here's what it looks like:



Notice that the "input" tag was used for the text fields as well as the buttons! The "input" tagis the tag you will use most often in forms. Here are different forms of the input tag:

- type="text"
- type="password" (asterisks when you type)
- type="checkbox"
- type="radio"
- type="submit"  (You can change the text on the button.)

- type="reset" (You can change the text on the button.)
- type="hidden" (You can't see this because it's hidden! This allows you to send additional variables that your user can't see.)
- type="button" (You can change the text on the button.

Below you will find a table of the various form elements and how you can use them.

FORM – attributes

action ="[url]"	This attribute usually has a link to a web page that contains code which processes the form. In our example above, we use "mailto" to tell the form to send the information as an email.
method ="get" ="post"	This attribute specifies how the information is sent. With the "get" method, all the information is sent in one long URL string. With the "post" method, all the information is sent but is "invisible" to the user.
name ="[name of form]"	When creating complex forms and using scripting, you may need to specify the name of the form. The name should be unique - that is, you shouldn't have other forms on the page with the same form name.

INPUT – attributes

type ="text" ="password" ="checkbox" ="radio" ="submit" ="reset" ="hidden" ="button"	The "type" attribute allows you to specify what type of form element you want to use. As you can see, the "input" element can have many different forms - see above for examples.
checked	This attribute does not have any values. When you put this attribute into the tag for "radio" or "checkbox," they will be selected.
src ="[url]"	Use this attribute when you have an "image" to specify the location of the image

TEXTAREA – attributes

This form element is the multi-line text box that you often seen for "comments." The opening and closing tag surround the text that will be initially displayed.

name ="[referring name]"	This attribute must be set to a unique name so you know how to refer to the form element.
rows ="[number]"	This attribute allows you to set the number of rows for the text area
cols ="[number]"	This attribute allows you to set the number of columns for the text area

SELECT – attributes

The SELECT tag allows you to create "select boxes" or "drop down menus." It is the "outer" element, used with the OPTION tag. (See the example link below)

name ="[referring name]"	This attribute must be set to a unique name so you know how to refer to the form element.
size ="[number]"	This attribute allows you to set how many rows are visible.
multiple	This attribute does not have any "values." When you set this attribute, it allows the user to select more than one option

OPTION – attributes

The OPTION tag is used with the SELECT tag to define the different options in the select box or dropdown menu. The opening and closing tag surround the text that will be displayed.

value ="[value]"	This attribute must be set to a unique name so you know how to refer to the form element. It will probably be an abbreviated version of the text that is displayed.
selected	This attribute does not have any "values." When you set this attribute, it marks this option as being "preselected."

Putting it all together...

Here's a sample page using this code below:

```
<form>
Enter your Name: <input type="text" width="20" size="20">
<p>Favorite Color:<br>
<input type="checkbox" checked value="red" name="colors"> Red<br>
<input type="checkbox" value="blue" name="colors"> Blue</p>
<p>Gender:<br>
<input type="radio" checked value="male" name="gender">Male<br>
<input type="radio" value="female" name="gender">Female</p>
<p>Address:<br>
<textarea rows="5" cols="20">Please enter your permanent address
here.</textarea></p>
Country:<br>
<select name="country">
<option value="usa" selected>USA</option>
<option value="uk">United Kingdom</option>
<option value="canada">Canada</option>
</select>
```

CSS

Introduction to CSS

CSS stands for Cascading Style Sheets. CSS describes how HTML elements are to be displayed on screen, paper, or in other media. CSS saves a lot of work. It can control the layout of multiple web pages all at once. External style sheets are stored in ".css" files. CSS is used to define styles for your web pages, including the design, layout and variations in display for different devices and screen sizes.

Types of CSS:

There are three types of CSS available. They are:

- Inline CSS
- Internal CSS
- External CSS

1. Inline CSS:

If the styles are mentioned along with the tag then this type of CSS is known as inline CSS.

Example:

```
<p style=”text-align: justify; background-color: red”>Hi this is the Inline CSS. </p>
```

2. Internal CSS:

For internal CSS, we write all the desired "styles" for the "selectors" along with the properties and values in the "head" section. And in the body section then newly defied selector tags are used with the actual contents.

Example:

```
<html>
<head>
<style type= “text/css”>
p
{
text-align: justify;
background-color: red;
}
```

```
</style>
</head>
<body>
<p>This is Internal CSS.</p>
</body>
</html>
```

3. External CSS:

Sometimes we need to apply particular style to more than one web page; in such cases external CSS can be used. The main idea in this type of CSS is that the desired styles can be written in one ".css" file. And this file can be called in our web pages to apply the styles.

Example:

```
<html>
<head>
<link rel = "stylesheet" type = "text/css" href = "external.css">
</head>
<body>
<p>This is Internal CSS.</p>
</body>
</html>
external.css:
p
{
 text-align: justify;
 background-color: red;
}
```

CSS Selectors:

CSS selectors are used to "find" (or select) HTML elements based on their element name, id, class, attribute, and more.

The element Selector

The element selector selects elements based on the element name. You can select all <p> elements on a page like this (in this case, all <p> elements will be center-aligned, with a red text color):

Example:

```
p {
text-align: center;
color: red;
}
```

The id Selector

The id selector uses the id attribute of an HTML element to select a specific element. The id of an element should be unique within a page, so the id selector is used to select one unique element! To select an element with a specific id, write a hash (#) character, followed by the id of the element. The style rule below will be applied to the HTML element with id="para1":

Example:

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

The class Selector

The class selector selects elements with a specific class attribute. To select elements with a specific class, write a period (.) character, followed by the name of the class. In the example below, all HTML elements with class="center" will be red and center-aligned:

Example:

```
.center {
 text-align: center;
 color: red;
}
```

You can also specify that only specific HTML elements should be affected by a class. In the example below, only <p> elements with class="center" will be center-aligned:

Example:

```
p.center {
 text-align: center;
 color: red;
}
```

HTML elements can also refer to more than one class. In the example below, the <p> element will be styled according to class="center" and to class="large":

Example:

```
<p class="center large">This paragraph refers to two classes.</p>
```

Controlling Page Layout

By default, HTML <div> sections will fill 100% of the web page width, and stack up one on top of the other. It is highly unlikely that this is how we want our page elements to be arranged; chances are we want some sections arranged side by side, and using various widths. For example, we might want to have a main content area with a menu down the left-hand side. Thankfully CSS makes this a simple task. The first step is to understand the float property.

The CSS float Property:

The float property tells HTML elements how to arrange themselves in relation to the other divs around them. We can specify one of two values for the float property - left or right. If we tell a <div> to float to the left, then it will shift itself as far left on the line as it can go before bumping into another <div> section. If we tell an element to float to the right it will shift as far right as it can? Let's take a look at a working example; this will give us a good visualization of how floating works. We'll begin by defining three areas for our web page:

```
<div id="header">This is the page header</div>
<div id="menu">Menu links goes in here</div>
<div id="content">And finally the page content in here</div>
```

Now, what we want to do is to have the header take up the whole of the top row, with the menu and content underneath, arranged side by side. To do this we would use the following CSS (the header section doesn't require any CSS):

```
#menu {
        float: left;
        }
        CSE - 16 -
        #content {
        float: left;
        }
```

Using CSS to set HTML element widths

One problem with arranging our HTML in this way is that the <div> sections will size themselves automatically to fit the text contained within them. This means that for a standard menu bar, where we have lots of text that we want to contain in a narrow area, we will end up with a very wide bar which will either squeeze the content section into a small space, or ever push it onto another line. We can fix this by specifying widths for our sections:

Example:

```
#menu {
float: left;
width: 10em;
}
```

Widths can be specified in pixels, ems or as a percentage of the total page width.

The CSS clear property

We have seen how we can position HTML elements next to each other using CSS, but sometimes we might want an element to appear beneath other elements. We can easily force an element to sit below another using the clear property. When using clear, we can tell an element to either clear those elements which are floated left, those which are floated right, or both:

Example:

```
#divname {
clear: left / right / both;
}
```

Changing Background

The CSS background properties are used to define the background effects for elements.

CSS background properties:

- background-color
- background-image
- background-repeat
- background-attachment
- background-position

Background Color

The background-color property specifies the background color of an element. The background color of a page is set like this:

Example

```
body {
 background-color: lightblue;
}
```

With CSS, a color is most often specified by:

- A valid color name - like "red"
- A HEX value - like "#ff0000"
- An RGB value - like "rgb(255,0,0)"

Background Image

The background-image property specifies an image to use as the background of an element. By default, the image is repeated so it covers the entire element. The background image for a page can be set like this:

Example:

```
body {
 background-image: url("paper.gif");
}
```

Background Image - Repeat Horizontally or Vertically:

By default, the background-image property repeats an image both horizontally and vertically. Some images should be repeated only horizontally or vertically, or they will look strange, like this:

Example:

```
body {
 background-image: url("gradient_bg.png");
}
```

If the image above is repeated only horizontally (background-repeat: repeat-x;), the background will look better:

```
body {
 background-image: url("gradient_bg.png");
 background-repeat: repeat-x;
}
```

Tip: To repeat an image vertically, set background-repeat: repeat-y;

Background Image - Set position and no-repeat:

Showing the background image only once is also specified by the background-repeat property:

Example:

```
body {
 background-image: url("img_tree.png");
 background-repeat: no-repeat;
}
```

In the example above, the background image is shown in the same place as the text. We want to change the position of the image, so that it does not disturb the text too much. The position of the image is specified by the background-position property:

Example:

```
body {
 background-image: url("img_tree.png");
 background-repeat: no-repeat;
 background-position: right top;
}
```

Background Image - Fixed position

To specify that the background image should be fixed (will not scroll with the rest of the page), use the background-attachment property:

Example:

```
body {
 background-image: url("img_tree.png");
 background-repeat: no-repeat;
 background-position: right top;
 background-attachment: fixed;
}
```

Controlling borders of the HTML elements using CSS:

The CSS border properties allow you to specify the style, width, and color of an element's border. The border-style property specifies what kind of border to display.

The following values are allowed:

- `dotted` - Defines a dotted border
- `dashed` - Defines a dashed border
- `solid` - Defines a solid border
- `double` - Defines a double border
- `groove` - Defines a 3D grooved border. The effect depends on the border-color value
- `ridge` - Defines a 3D ridged border. The effect depends on the border-color value
- `inset` - Defines a 3D inset border. The effect depends on the border-color value
- `outset` - Defines a 3D outset border. The effect depends on the border-color value
- `none` - Defines no border
- `hidden` - Defines a hidden border

The border-style property can have from one to four values (for the top border, right border, bottom border, and the left border).

Example:

```
p.dotted {border-style: dotted;}
p.dashed {border-style: dashed;}
p.solid {border-style: solid;}
p.double {border-style: double;}
p.groove {border-style: groove;}
p.ridge {border-style: ridge;}
p.inset {border-style: inset;}
p.outset {border-style: outset;}
p.none {border-style: none;}
p.hidden {border-style: hidden;}
p.mix {border-style: dotted dashed solid double;}
```

The `border-width` property specifies the width of the four borders. The width can be set as a specific size (in px, pt, cm, em, etc) or by using one of the three pre-defined values: thin, medium, or thick. The `border-width` property can have from one to four values (for the top border, right border, bottom border, and the left border).

Example

```
p.one {
 border-style: solid;
 border-width: 5px;
}
p.two {
 border-style: solid;
 border-width: medium;
}
p.three {
 border-style: solid;
 border-width: 2px 10px 4px 20px;
}
```

The `border-color` property is used to set the color of the four borders. The color can be set by:

- name - specify a color name, like "red"
- Hex - specify a hex value, like "#ff0000"
- RGB - specify a RGB value, like "rgb(255,0,0)"

- transparent

The `border-color` property can have from one to four values (for the top border, right border, bottom border, and the left border). If border-color is not set, it inherits the color of the element.

Example

```
p.one {
 border-style: solid;
 border-color: red;
}
p.two {
 border-style: solid;
 border-color: green;
}
p.three {
 border-style: solid;
 border-color: red green blue yellow;
}
```

JAVASCRIPT

JavaScript is the scripting language of the Web. JavaScript is used in millions of Web pages to add functionality, validate forms, detect browsers, and much more.

Introduction to JavaScript

- JavaScript is used in millions of Web pages to improve the design, validate forms, detect browsers, create cookies, and much more.
- JavaScript is the most popular scripting language on the Internet, and works in all major browsers, such as Internet Explorer, Mozilla Firefox, and Opera.

What is JavaScript?

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

Java and JavaScript are two completely different languages in both concept and design! Java (developed by Sun Microsystems) is a powerful and much more complex programming language - in the same category as C and C++.

What can a JavaScript Do ?

- **JavaScript gives HTML designers a programming tool** - HTML authors are normally not programmers, but JavaScript is a scripting language with a very simple syntax! Almost anyone can put small "snippets" of code into their HTML pages
- **JavaScript can put dynamic text into an HTML page** - A JavaScript statement like this: document.write("<h1>" + name + "</h1>") can write a variable text into an HTML page

- **JavaScript can react to events -** A JavaScript can be set to execute when something happens, like when a page has finished loading or when a user clicks on an HTML element
- **JavaScript can read and write HTML elements** - A JavaScript can read and change the content of an HTML element
- **JavaScript can be used to validate data -** A JavaScript can be used to validate form data before it is submitted to a server. This saves the server from extra processing
- **JavaScript can be used to detect the visitor's browser -** A JavaScript can be used to detect the visitor's browser, and - depending on the browser - load another page specifically designed for that browser
- **JavaScript can be used to create cookies -** A JavaScript can be used to store and retrieve information on the visitor's computer.

JavaScript Variables

Variables are "containers" for storing information. JavaScript variables are used to hold values or expressions. A variable can have a short name, like x, or a more descriptive name, like carname.

Rules for JavaScript variable names:

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

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

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.

```
<html>
<body>
<script type="text/javascript">
var firstname;
firstname="Welcome";
document.write(firstname);
document.write("<br />");
firstname="XYZ";
document.write(firstname);
</script>
<p>The script above declares a variable,
assigns a value to it, displays the value, change the value,
and displays the value again.</p>
</body>
</html>
```

Output :

```
Welcome
XYZ
```

The script above declares a variable, assigns a value to it, displays the value, change the value, and displays the value again.

Declaring (Creating) JavaScript Variables

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

```
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="Scorpio";
```

After the execution of the statements above, the variable x will hold the value 5, and carname will hold the value Scorpio.

Note: When you assign a text value to a variable, use quotes around the value.

Assigning Values to Undeclared JavaScript Variables

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

These statements:

```
x=5;
carname="Scorpio";
have the same effect as:
var x=5;
var carname="Scorpio";
```

Redeclaring JavaScript Variables

If you redeclare a JavaScript variable, it will not lose its original value.

```
var x=5;
var x;
```

After the execution of the statements above, the variable x will still have the value of 5. The value of x is not reset (or cleared) when you redeclare it.

DataTypes

- **Numbers** - are values that can be processed and calculated. You don't enclose them in quotation marks. The numbers can be either positive or negative.
- **Strings** - are a series of letters and numbers enclosed in quotation marks. JavaScript uses the string literally; it doesn't process it. You'll use strings for text you want displayed or values you want passed along.
- **Boolean (true/false) -** lets you evaluate whether a condition meets or does not meet specified criteria.
- **Null -** is an empty value. null is not the same as 0 -- 0 is a real, calculable number, whereas null is the absence of any value.

TYPE	EXAMPLE
Numbers	Any number, such as 17, 21, or 54e7
Strings	"Greetings!" or "Fun"
Boolean	Either true or false
Null	A special keyword for exactly that – the null value (that is, nothing)

JavaScript Arithmetic

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

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

JavaScript Operators

- The operator = is used to assign values.
- The operator + 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 + x	X = 7
-	Subtraction	X = y – x	X = 3
*	Multiplication	X = y * x	X = 10

/	Division	X = y / x	X = 2.5
%	Modulus	X = y % x	X = 1
++	Increment	X = ++y	X = 6
--	Decrement	X = --y	X = 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 = 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

Look at these examples:

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

The rule is:

- If you add a number and a string, the result will be a string.
- 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
===	Is exactly equal to (value and type)	x===5 is true, x==="5" is false
!=	Is not equal to	x!=8 is true
>	Greater than	x>8 is false
<	Less than	x<8 is true
>=	Greater than equal to	x>=8 is false
<=	Less than 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");
```

You will learn more about the use of conditional statements in the next chapter of this tutorial.

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
```

Example

```
greeting=(visitor=="PRES")?"Dear President ":"Dear ";
```

If the variable **visitor** has the value of "PRES", then the variable **greeting** will be assigned the value

```
"Dear President " else it will be assigned "Dear".
```

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 if you want to execute some code only if a specified condition is true
- if...else statement - use this statement if you want 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 if you want to select one of many blocks of code to be executed
- switch statement - use this statement if you want to select one of many blocks of code to be executed

If Statement

You should use the if statement if you want 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 1

```
<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>
```

Example 2

```
<script type="text/javascript">
//Write "Lunch-time!" if the time is 11
var d=new Date();
var time=d.getHours();
if (time==11)
{
document.write("<b>Lunch-time!</b>");
}
</script>
```

Note: When comparing variables you must always use two equals signs next to each other (==)!

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

If...else Statement

If you want to execute some code if a condition is true and another code if the condition is not true, use the if....else statement.

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

You should use the if....else if...else statement if you want to select one of many sets of lines to execute.

Syntax

```
if (condition1)
{
code to be executed if condition1 is true
}
```

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

Example

```
<script type="text/javascript">
var d = new Date()
var time = d.getHours()
if (time<10)
{
document.write("<b>Good morning</b>");
}
else if (time>10 && time<16)
{
document.write("<b>Good day</b>");
}
else
{
document.write("<b>Hello World!</b>");
}
</script>
```

The JavaScript Switch Statement

You should use the switch statement if you want to select one of many blocks of code to be executed.

Syntax

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

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

Example

```
<script type="text/javascript">
//You will receive a different greeting based
//on what day it is. Note that Sunday=0,
//Monday=1, Tuesday=2, etc.
var d=new Date();
theDay=d.getDay();
switch (theDay)
{
case 5:
 document.write("Finally Friday");
 break;
case 6:
 document.write("Super Saturday");
 break:
case 0:
 document.write("Sleepy Sunday");
break;
default:
 document.write("I'm looking forward to this weekend!");
}
</script>
```

JavaScript Controlling(Looping) Statements

Loops in JavaScript are used to execute the same block of code a specified number of times or while a specified condition is true.

JavaScript Loops

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

Example

Explanation: 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 10. **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.

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

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

Result

```
The number is 0
The number is 1
The number is 2
The number is 3
The number is 4
The number is 5
The number is 6
The number is 7
The number is 8
The number is 9
The number is 10
```

JavaScript While Loop

Loops in JavaScript are used to execute the same block of code a specified number of times or while a specified condition is true.

The while loop -

The while loop is used when you want the loop to execute and continue executing while the specified condition is true.

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

Note: The <= could be any comparing statement.

Example

Explanation: 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 10. i will increase by 1 each time the loop runs.

```
<html>
```

```
<body>
<script type="text/javascript">
var i=0;
while (i<=10)
{
document.write("The number is " + i);
document.write("<br />");
i=i+1;
}
</script>
</body>
</html>
```

Result

```
The number is 0
The number is 1
The number is 2
The number is 3
The number is 4
The number is 5
The number is 6
The number is 7
The number is 8
The number is 9
The number is 10
```

The do...while Loop -

The do...while loop is a variant of the while loop. This loop will always execute a block of code ONCE, and then it will repeat the loop as long as the specified condition is true. This loop will always be executed at least once, even if the condition is false, because the code is executed before the condition is tested.

```
do
{
 code to be executed
}

while (var<=endvalue);
```

Example

```
<html>
<body>
<script type="text/javascript">
var i=0;
do
{
document.write("The number is " + i);
document.write("<br />");
i=i+1;
}
while (i<0);
</script>
</body>
</html>
```

Result

```
The number is 0
```

JavaScript Break and Continue Statements

There are two special statements that can be used inside loops: **break** and **continue**.

Break

The break command 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>
```

Result

```
The number is 0
The number is 1
The number is 2
```

Continue

The continue command 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>
```

Result

```
The number is 0
The number is 1
The number is 2
The number is 4
The number is 5
The number is 6
The number is 7
The number is 8
The number is 9
The number is 10
```

JavaScript Functions

A function (also known as a method) is a self-contained piece of code that performs a particular "function". You can recognise a function by its format - it's a piece of descriptive text, followed by open and close brackets. A function is a reusable code-block that will be executed by an event, or when the function is called.

- 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 that function.
- You may call a function from anywhere within the 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 the function is read/loaded by the browser before it is called, it could be wise to put it in the **<head>** section.

Example

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

</script>
</head>
<body>
```

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

If the line: alert("Hello world!!") in the example above had not been put within a function, it would have been executed as soon as the line was loaded. Now, the script is not executed before the user hits the button. We have added an onClick event to the button that will execute the function displaymessage() when the button is clicked.

How to Define a Function?

The syntax for creating a function is:

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

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:

```
function functionname()
{
some code
}
```

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.

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.

Example

The function below should return the product of two numbers (a and b):

```
function prod(a,b)
{
x=a*b;
return x;
}
```

When you call the function above, you must pass along two parameters:

```
product=prod(2,3);
```

The returned value from the prod() function is 6, and it will be stored in the variable called product.

The Lifetime of JavaScript Variables

- When you declare a variable 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.

What is an Event?

Event Handlers

- **Event Handlers** are JavaScript methods, i.e. functions of objects, that allow us as JavaScript programmers to control what happens when events occur.
- Directly or indirectly, an **Event** is always the result of something a user does. For example, we've already seen Event Handlers like **onClick** and **onMouseOver** that respond to mouse actions. Another type of Event, an internal change-of-state to the page (**completion** of loading or **leaving** the page). An **onLoadEvent** can be considered an indirect result of a user action.

- Although we often refer to Events and Event Handlers interchangeably, it's important to keep in mind the distinction between them. An **Event** is merely something that happens - something that it is initiated by an **Event Handler** (**onClick**, **onMouseOver**, etc...).
- The elements on a page which can trigger events are known as "**targets**" or "**target elements**," and we can easily understand how a button which triggers a Click event is a target element for this event. Typically, events are defined through the use of Event Handlers, which are bits of script that tell the browser what to do when a particular event occurs at a particular target. These Event Handlers are commonly written as attributes of the target element's HTML tag.

The Event Handler for a **Click** event at a form field button element is quite simple to understand:

```
<INPUT TYPE="button" NAME="click1" VALUE="Click me for fun!"
onClick="event_handler_code">
```

The **event_handler_code** portion of this example is any valid JavaScript and it will be executed when the specified event is triggered at this target element. This particular topic will be continued in **Incorporating JavaScripts into your HTML pages.**

There are "three different ways" that Event Handlers can be used to trigger Events or Functions.

Method 1 (Link Events):

Places an Event Handler as an attribute within an **<A HREF= >** tag, like this:

```
<A HREF="foo.html" onMouseOver="doSomething()"> ... </A>
```

You can use an Event Handler located within an **<A HREF= >** tag to make either an image or a text link respond to a mouse over Event. Just enclose the image or text string between the **<A HREF= >** and the **</A>** tags.

Whenever a user clicks on a link, or moves her cursor over one, JavaScript is sent a **Link Event**. One Link Event is called **onClick**, and it gets sent whenever someone clicks on a link. Another link event is called **onMouseOver**. This one gets sent when someone moves the cursor over the link.

You can use these events to affect what the user sees on a page. Here's an example of how to use link events. Try it out, View Source, and we'll go over it.

```
<A HREF="javascript:void('')"
onClick="open('index.htm',  'links',  'height=200,width=200');">How  to  Use  Link
Events
</A>
```

The first interesting thing is that there are no **<SCRIPT>** tags. That's because anything that appears in the quotes of an onClick or an onMouseOver is automatically interpreted as JavaScript. In fact, because semicolons mark the end of statements allowing you to write entire JavaScripts in one line, you can fit an entire JavaScript program between the quotes of an onClick. It'd be ugly, but you could do it.

Here are the three lines of interest:

```
1. <A HREF="#" onClick="alert('Ooo, do it again!');">Click on me!</A>
2.  <A HREF="javascript:void('')" onClick="alert('Ooo, do it again!');">
   Click on me!
   </A>
3. <A HREF="javascript:alert('Ooo, do it again!')" >Click on me!</A>
```

In the first example we have a normal <A> tag, but it has the magic onClick="" element, which says, "When someone clicks on this link, run the little bit of JavaScript between my quotes." Notice, there's even a terminating semicolon at the end of the alert. Question: is this required? NO.

Let's go over each line:

- **HREF="#"** tells the browser to look for the anchor #, but there is no anchor "#", so the browser reloads the page and goes to top of the page since it couldn't find the anchor.
- **<A HREF="javascript:void('')"** tells the browser not to go anywhere - it "deadens" the link when you click on it. HREF="javascript: is the way to call a function when a link (hyperlinkor an HREFed image) is clicked.
- **HREF="javascript:alert('Ooo, do it again!')"** here we kill two birds with one stone. The default behaviour of a hyperlink is to click on it. By clicking on the link we call the window Method alert() and also at the same time "deaden" the link.

The next line is

```
<A HREF="javascript:void('')" onMouseOver="alert('Hee hee!');">
Mouse over me!
</A>
```

This is just like the first line, but it uses an **onMouseOver** instead of an **onClick**.

Method 2 (Actions within FORMs):

The second technique we've seen for triggering a Function in response to a mouse action is to place an **onClick** Event Handler inside a button type form element, like this:

```
<FORM>
 <INPUT TYPE="button" onClick="doSomething()">
</FORM>
```

While any JavaScript statement, methods, or functions can appear inside the quotation marks of an Event Handler, typically, the JavaScript script that makes up the Event Handler is actually a call to a function defined in the header of the document or a single JavaScript command. Essentially, though, anything that appears inside a command block (inside curly braces {}) can appear between the quotation marks.

For instance, if you have a form with a text field and want to call the function **checkField()** whenever the value of the text field changes, you can define your text field as follows:

```
<INPUT TYPE="text" onChange="checkField(this)">
```

Nonetheless, the entire code for the function could appear in quotation marks rather than a function call:

```
<INPUT TYPE="text" onChange="if (this.value <= 5) {
 alert("Please enter a number greater than 5");
}">
```

To separate multiple commands in an Event Handler, use semicolons

```
<INPUT TYPE="text" onChange="alert('Thanks for the entry.');
confirm('Do you want to continue?');">
```

The advantage of using functions as Event Handlers, however, is that you can use the same Event Handler code for multiple items in your document and, functions make your code easier to read and understand.

Method 3 (BODY onLoad & onUnLoad):

The third technique is to us an Event Handler to ensure that all required objects are defined involve the **onLoad** and **onUnLoad**. These Event Handlers are defined in the **<BODY>** or **<FRAMESET>** tag of an HTML file and are invoked when the document or frameset is fully loaded or unloaded. If you set a flag within the **onLoad** Event Handler, other Event Handlers can test this flags to see if they can safely run, with the knowledge that the document is fully loaded and all objects are defined.

For example:

```
<SCRIPT>
var loaded = false;
function doit() {
// alert("Everything is \"loaded\" and loaded = " + loaded);
alert('Everything is "loaded" and loaded = ' + loaded);
}
</SCRIPT>
<BODY onLoad="loaded = true;">
-- OR --
<BODY onLoad="window.loaded = true;">
<FORM>
<INPUT TYPE="button" VALUE="Press Me"
onClick="if (loaded == true) doit();">
-- OR --
<INPUT TYPE="button" VALUE="Press Me"
onClick="if (window.loaded == true) doit();">
-- OR --
<INPUT TYPE="button" VALUE="Press Me"
onClick="if (loaded) doit();">
</FORM></BODY>
```

The **onLoad** Event Handler is executed when the document or frameset is fully loaded, which means that all images have been downloaded and displayed, all subframes have loaded, any Java Applets and Plugins (Navigator) have started running, and so on. The **onUnLoad** Event Handler is executed just before the page is unloaded, which occurs when the browser is about to move on to a new page. Be aware that when you are working with multiple frames, there is no guarantee of the order in which the **onLoad** Event Handler is invoked for the various frames, except that the Event Handlers for the parent frame is invoked after the Event Handlers of all its children frames.

Setting the bgColor Property

The **first example** allows the user to change the color by clicking buttons, while the **second example** allows you to change colors by using drop down boxes.

Event Handlers

EVENT	DESCRIPTION
onAbort	the user cancels loading of an image
onBlur	input focus is removed from a form element (when the user clicks outside the field) or focus is removed from a window
onClick	the user clicks on a link or form element
onChange	the value of a form field is changed by the user
onError	an error happens during loading of a document or image
onFocus	input focus is given to a form element or a window
onLoad	once a page is loaded, NOT while loading
onMouseOut	the user moves the pointer off of a link or clickable area of an image map
onMouseOver	the user moves the pointer over a hypertext link
onReset	the user clears a form using the Reset button
onSelect	the user selects a form element’s field
onSubmit	a form is submitted (ie, when the users clicks on a submit button)
onUnLoad	the user leaves a page

Note: Input focus refers to the act of clicking on or in a form element or field. This can be done by clicking in a text field or by tabbing between text fields.

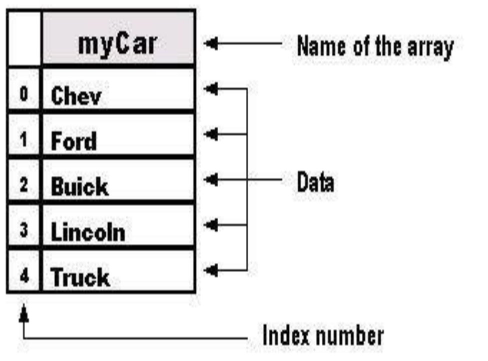
Which Event Handlers Can Be Used

OBJECT	EVENT HANDLERS AVAILABLE
Button element	onClick, onMouseOver
Checkbox	onClick
Clickable ImageMap area	onClick, onMouseOver, onMouseOut
Document	onLoad, onUnload, onError
Form	onSubmit, onReset
Framesets	onBlur, onFocus
Hypertext link	onClick, onMouseOver, onMouseOut
Image	onLoad, onError, onAbort
Radio button	onClick
Reset button	onClick
Selection list	onBlur, onChange, onFocus
Submit button	onClick
TextArea element	onBlur, onChange, onFocus, onSelect
Text element	onBlur, onChange, onFocus, onSelect
Window	onLoad, onUnload, onBlur, onFocus

JavaScript Arrays

An array object is used to create a database-like structure within a script. Grouping

data points (array elements) together makes it easier to access and use the data in a script. There are methods of accessing actual databases (which are beyond the scope of this series) but here we're talking about small amounts of data.



Comparison of an array to a column of data

- An array can be viewed like a column of data in a spreadsheet. The name of the array would be the same as the name of the column. Each piece of data (element) in the array is referred to by a number (index), just like a row number in a column.
- An array is an *object*. Earlier, I said that **an object** is a thing, a collection of properties (array elements, in this case) grouped together.
- You can name an array using the same format as a variable, a function or an object. Remember our basic rules: The first character cannot be a number, you cannot use a reserved word, and you cannot use spaces. Also, be sure to remember that the name of the array object is capitalized, e.g. Array.
- The JavaScript interpreter uses numbers to access the collection of elements (i.e. the data) in an array. Each index number (as it is the number of the data in the array's index) refers to a specific piece of data in the array, similar to an ID number. It's important to remember that the index numbering of the data starts at "0." So, if you have 8 elements, the first element will be numbered "0" and the last one will be "7."
- Elements can be of any type: character string, integer, Boolean, or even another array. An array can even have different types of elements within the same array. Each element in the array is accessed by placing its index number in brackets, i.e. myCar[4]. This would mean that we are looking for data located in the array myCar which has an index of "4." Since the numbering of an index starts at "0," this would actually be the fifth index. For instance, in the following array,

```
var myCar = new Array("Chev","Ford","Buick","Lincoln","Truck");
alert(myCar[4])
```

- The data point with an index of "4" would be Truck. In this example, the indexes are numbered as follows: 0=Chev, 1=Ford, 2=Buick, 3=Lincoln, and 4=Truck. When creating loops, it's much easier to refer to a number than to the actual data itself.

The Size of the Array

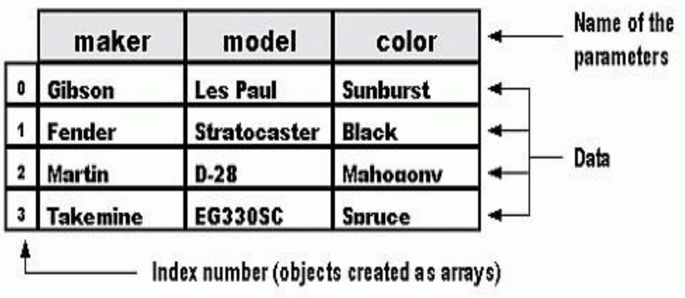
- The size of an array is determined by either the actual number of elements it contains or by actually specifying a given size. You don't need to specify the size of the array. Sometimes, though, you may want to pre-set the size, e.g.:

```
var myCar = new Array(20);
```

- That would pre-size the array with 20 elements. You might pre-size the array in order to set aside the space in memory.

Multidimensional Arrays

This type of an array is similar to parallel arrays. In a multidimensional array, instead of creating two or more arrays in tandem as we did with the parallel array, we create an array with several levels or "dimensions." Remember our example of a spreadsheet with rows and columns? This time, however, we have a couple more columns.



	maker	model	color
0	Gibson	Les Paul	Sunburst
1	Fender	Stratocaster	Black
2	Martin	D-28	Mahogany
3	Takemine	EG330SC	Spruce

Name of the parameters
Data
Index number (objects created as arrays)

Comparison of a multidimensional array to a column of data

Multidimensional arrays can be created in different ways. Let's look at one of these method. First, we create the main array, which is similar to what we did with previous arrays.

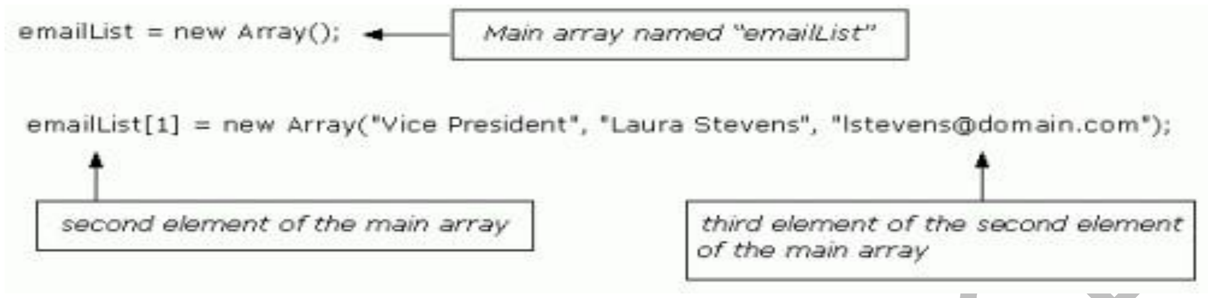
```
var emailList = new Array();
```

Next, we create arrays for elements of the main array :

```
emailList[0] = new Array("President", "Paul Smith", "psmith@domain.com");
emailList[1] = new Array("Vice President", "Laura Stevens", "lstevens@domain.com");
emailList[2] = new Array("General Manager", "Mary Larsen", "mlarsen@domain.com");
emailList[3] = new Array("Sales Manager", "Bob Lark", "blark@domain.com");
```

In this script we created "sub arrays" or arrays from another level or "dimension." We used the name of the main array and gave it an index number (e.g., emailList[0]). Then we created a new instance of an array and gave it a value with three elements.

In order to access a single element, we need to use a double reference. For example, to get the e-mail address for the Vice President in our example above, access the third element "[2]" of the array named emailList.


emailList = new Array();
Main array named "emailList"
emailList[1] = new Array("Vice President", "Laura Stevens", "lstevens@domain.com");
second element of the main array
third element of the second element of the main array

It would be written like this:

```
var vpEmail = emailList[1][2]
alert("The address is: "+ vpEmail)
```

- We declared a variable, named it emailList, and initialized it with a value of a new instance of an array.
- Next, we created an array for each of the elements within the original array. Each of the new arrays contained three elements.
- Then we declared a variable named vpEmail and initialized it with the value of the third element (lstevens@domain.com) of the second element "[1]" of the array named emailList.

You could also retrieve the information using something like:

```
var title = emailList[1][0]
var email = emailList[1][2]
alert("The e-mail address for the " + title +" is: " + email)
```

Array Properties

Length

The length property returns the number of elements in an array. The format is arrayName.length. The length property is particularly useful when using a loop to cycle through an array. One example would be an array used to cycle banners:

```
var bannerImg = new Array();
 bannerImg[0]="image-1.gif";
 bannerImg[1]="image-2.gif";
 bannerImg[2]="image-3.gif";
var newBanner = 0
var totalBan = bannerImg.length
function cycleBan() {
 newBanner++
 if (newBanner == totalBan) {
 newBanner = 0
 }
 document.banner.src=bannerImg[newBanner]
setTimeout("cycleBan()", 3*1000)
}
window.onload=cycleBan;
```

This portion is then placed in the body where the banner is to be displayed:

```
&lt;img src="image-1.gif" name="banner">
```

Let's take a look and see what happened here:

- On the first line, we created a new instance of the array bannerImg, and gave it three data elements. (Remember, we are only making a copy of the Array object here.)
- Next, we created two variables: newBanner, which has a beginning value of zero; and totalBan, which returns the length of the array (the total number of elements contained in the array).
- Then we created a function named cycleBan. This function will be used to create a loop to cycle the images.

  - We set the newBanner variable to be increased each time the function cycles. (Review: By placing the increment operator [" ++ "] after the variable [the "operand"], the variable is incremented only after it returns its current value to the script. For example, its beginning value is "0", so in the first cycle it will return a value of "0" to the script and then its value will be increased by "1".)

- When the value of the newBanner variable is equal to the variable totalBan (which is the length of the array), it is then reset to "0". This allows the images to start the cycle again, from the beginning.
- The next statement uses the Document Object Method (DOM - we'll be taking a look at that soon) to display the images on the Web page. Remember, we use the dot operator to access the properties of an object. We also read the statement backwards, i.e., "take the element from the array bannerImg, that is specified by the current value of the variable newBanner, and place it in the src attribute located in the element with the name attribute of banner, which is located in the document object."
- We then used the setTimeout function to tell the script how long to display each image. This is always measured in milliseconds so, in this case, the function cycleBan is called every 3,000 milliseconds (i.e., every 3 seconds).

- Finally, we used the window.onload statement to execute the function cycleBan as soon as the document is loaded.

There are a total of five properties for the Array object. In addition to the length property listed above, the others are:

- **constructor:** Specifies the function that creates an object's prototype.
- **index:** Only applies to JavaScript arrays created by a regular expression match.
- **input:** Only applies to JavaScript arrays created by a regular expression match.
- **prototype:** Used to add properties or methods.

The other properties listed here are either more advanced or seldom used. For now, we'll stick to the basics.

Javascript Object Hierarchy

HIERARCHY OBJECTS

OBJECTS	PROPERTIES	METHODS	EVENT HANDLERS
Window	defaultStatus frames opener parent scroll self-status top window	alert blur close confirm focus open prompt clearTimeout setTimeout	onLoad onUnload onBlur onFocus
History	length forward go	back	none
Navigator	Navigator appCodeName appName appVersion mimeTypes plugins userAgent	javaEnabled	none
document	alinkColor anchors applets area bgColor cookie fgColor forms images lastModified linkColor links location referrer title vlinkColor	clear close open write writeln	none (the onLoad and onUnload event handlers belong to the Window object.

image	border complete height hspace lowsrc name src vspace width	none	none
form	action elements encoding FileUpload method name target	submit reset	onSubmit onReset
text	defaultValue name type value	focus blur select	onBlur onCharge onFocus onSelect
BUILT – IN OBJECTS			
Array	length	join reverse sort xx	none

Date	none	getDate getDay getHours getMinutes getMonth getSeconds getTime getTimeZoneoffset getYear parse prototype setDate setHours setMinutes setMonth setSeconds setTime	none
		setYear toGMTString toLocaleString UTC	
String	length prototype	anchor big blink bold charAt fixed fontColor fontSize indexOf italics lastIndexOf link small split strike sub substring sup toLowerCase toUpperCase	Window

JavaScript Array Object

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

Create an Array

The following code creates an Array object called myCars:

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

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

```
1)    var myCars=new Array();
      myCars[0]="Saab";
      myCars[1]="Volvo";
      myCars[2]="BMW";
```

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

```
var myCars=new Array(3);
myCars[0]="Saab";
myCars[1]="Volvo";
myCars[2]="BMW"
```

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

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

Access an Array

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

The following code line:

```
document.write(myCars[0]);
```

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
```

JavaScript Date Object

Create a Date Object

- The Date object is used to work with dates and times.
- The following code create a Date object called myDate:

```
var myDate=new Date()
```

Note: The Date object will automatically hold the current date and time as its initial value!

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 2010:

```
var myDate=new Date();
myDate.setFullYear(2010,0,14);
var today = new Date();
if (myDate>today)
{
alert("Today is before 14th January 2010");
}
else
{
alert("Today is after 14th January 2010");
}
```

JavaScript Math Object

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 pi_value=Math.PI;
var sqrt_value=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:

```
0.4218824567728053
```

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:

```
4
```

JavaScript String Object

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!
```

Window Object

- The Window object is the top level object in the JavaScript hierarchy.
- The Window object represents a browser window.
- A Window object is created automatically with every instance of a <body> or <frameset> tag.

IE: Internet Explorer, F: Firefox, O: Opera.

Window Object Collections

Collection	Description	IE	F	O
frames[ ]	Returns all named frames in the window	4	1	9

Window Object Properties

Property	Description	IE	F	O
closed	Returns whether or not a window has been closed	4	1	9
defaultStatus	Sets or returns the default text in the statusbar of the window	4	No	9
document	See Document object	4	1	9
history	See History object	4	1	9
length	Sets or returns the number of frames in the window	4	1	9
location	See Location object	4	1	9
name	Sets or returns the name of the window	4	1	9
opener	Returns a reference to the window that created the window	4	1	9
outerHeight	Sets or returns the outer height of a window	No	1	No
outerWidth	Sets or returns the outer width of a window	No	1	No
pageXOffset	Sets or returns the X position of the current page in relation to the upper left corner of a window's display area	No	No	No
pageYOffset	Sets or returns the Y position of the current page in relation to the upper left corner of a window's display area	No	No	No
parent	Returns the parent window	4	1	9
personalbar	Sets whether or not the browser's personal bar (or directories bar) should be visible			

scrollbars	Sets whether or not the scrollbars should be visible			
self	Returns a reference to the current window	4	1	9
status	Sets the text in the statusbar of a window	4	No	9
statusbar	Sets whether or not the browser's statusbar should be visible			
toolbar	Sets whether or not the browser's tool bar is visible or not (can only be set before the window is opened and you must have UniversalBrowserWrite privilege)			
top	Returns the topmost ancestor window	4	1	9

Window Object Methods

Method	Description	IE	F	O
alert()	Displays an alert box with a message and an OK button	4	1	9
blur()	Removes focus from the current window	4	1	9
clearInterval()	Cancels a timeout set with setInterval()	4	1	9
clearTimeout()	Cancels a timeout set with setTimeout()	4	1	9
close()	Closes the current window	4	1	9
confirm()	Displays a dialog box with a message and an OK and a Cancel button	4	1	9
createPopup()	Creates a pop-up window	4	No	No
focus()	Sets focus to the current window	4	1	9
moveBy()	Moves a window relative to its current position	4	1	9
moveTo()	Moves a window to the specified position	4	1	9
open()	Opens a new browser window	4	1	9
print()	Prints the contents of the current window	5	1	9
prompt()	Displays a dialog box that prompts the user for input	4	1	9
resizeBy()	Resizes a window by the specified pixels	4	1	9
resizeTo()	Resizes a window to the specified width and height	4	1.5	9

scrollBy()	Scrolls the content by the specified number of pixels	4	1	9
scrollTo()	Scrolls the content to the specified coordinates	4	1	9
setInterval()	Evaluates an expression at specified intervals	4	1	9
setTimeout()	Evaluates an expression after a specified number of milliseconds	4	1	9

Document Object

- The Document object represents the entire HTML document and can be used to access all elements in a page.
- The Document object is part of the Window object and is accessed through the window.document property.

IE: Internet Explorer, F: Firefox, O: Opera, W3C: World Wide Web Consortium (Internet Standard).

Document Object Collections

Collection	Description	IE	F	O	W3C
anchors[ ]	Returns a reference to all Anchor objects in the document	4	1	9	Yes
forms[ ]	Returns a reference to all Form objects in the document	4	1	9	Yes
Images[ ]	Returns a reference to all Image objects in the document	4	1	9	Yes
links[ ]	Returns a reference to all Area and Link objects in the document	4	1	9	Yes

Document Object Properties

Property	Description	IE	F	O	W3C
body	Gives direct access to the <body> element				
cookie	Sets or returns all cookies associated with the current document	4	1	9	Yes
domain	Returns the domain name for the current document	4	1	9	Yes
lastModified	Returns the date and time a document was last modified	4	1	No	No
referrer	Returns the URL of the document that loaded the current document	4	1	9	Yes
title	Returns the title of the current document	4	1	9	Yes
URL	Returns the URL of the current document	4	1	9	Yes

Document Object Methods

Method	Description	IE	F	O	W3C
close()	Closes an output stream opened with the document.open() method, and displays the collected data	4	1	9	Yes
getElementById()	Returns a reference to the first object with the specified id	5	1	9	Yes
getElementsByName()	Returns a collection of objects with the specified name	5	1	9	Yes
getElementsByTagName()	Returns a collection of objects with the specified tagname	5	1	9	Yes
open()	Opens a stream to collect the output from any document.write() or document.writeln() methods	4	1	9	Yes
write()	Writes HTML expressions or JavaScript code to a document	4	1	9	Yes

writeln()	Identical to the write() method, with the addition of writing a new line character after each expression	4	1	9	Yes

History Object

- The History object is actually a JavaScript object, not an HTML DOM object.
- The History object is automatically created by the JavaScript runtime engine and consists of an array of URLs. These URLs are the URLs the user has visited within a browser window.
- The History object is part of the Window object and is accessed through the window.history property.

IE: Internet Explorer, **F:** Firefox, **O:** Opera.

History Object Properties

Property	Description	IE	F	O
length	Returns the number of elements in the history list	4	1	9

History Object Methods

Method	Description	IE	F	O
back()	Loads the previous URL in the history list	4	1	9
forward()	Loads the next URL in the history list	4	1	9
go()	Loads a specific page in the history list	4	1	9

Form Object

- The Form object represents an HTML form.
- For each instance of a <form> tag in an HTML document, a Form object is created.

IE: Internet Explorer, **F:** Firefox, **O:** Opera, **W3C:** World Wide Web Consortium (Internet Standard).

Form Object Collections

Collection	Description	IE	F	O	W3C
elements[ ]	Returns an array containing each element in the form	5	1	9	Yes

Form Object Properties

Property	Description	IE	F	O	W3C
acceptCharset	Sets or returns a list of possible character-sets for the form data	No	No	No	Yes
action	Sets or returns the action attribute of a form	5	1	9	Yes
enctype	Sets or returns the MIME type used to encode the content of a form	6	1	9	Yes
id	Sets or returns the id of a form	5	1	9	Yes
length	Returns the number of elements in a form	5	1	9	Yes
method	Sets or returns the HTTP method for sending data to the server	5	1	9	Yes
name	Sets or returns the name of a form	5	1	9	Yes
target	Sets or returns where to open the action-URL in a form	5	1	9	Yes

Standard Properties

Property	Description	IE	F	O	W3C
className	Sets or returns the class attribute of an element	5	1	9	Yes
dir	Sets or returns the direction of text	5	1	9	Yes
lang	Sets or returns the language code for an element	5	1	9	Yes
title	Sets or returns an element's advisory title	5	1	9	Yes

Form Object Methods

Method	Description	IE	F	O	W3C
reset()	Resets the values of all elements in a form	5	1	9	Yes
submit()	Submits a form	5	1	9	Yes

Image Object

- The Image object represents an embedded image.
- For each instance of an <img> tag in an HTML document, an Image object is created.

IE: Internet Explorer, **F:** Firefox, **O:** Opera, **W3C:** World Wide Web Consortium (Internet Standard).

Image Object Properties

Property	Description	IE	F	O	W3C
align	Sets or returns how to align an image according to the surrounding text	5	1	9	Yes
alt	Sets or returns an alternate text to be displayed, if a browser cannot show an image	5	1	9	Yes
border	Sets or returns the border around an image	4	1	9	Yes
complete	Returns whether or not the browser has finished loading the image	4	1	9	No
height	Sets or returns the height of an image	4	1	9	Yes
hspace	Sets or returns the white space on the left and right side of the image	4	1	9	Yes
id	Sets or returns the id of the image	4	1	9	Yes
isMap	Returns whether or not an image is a server-side image map	5	1	9	Yes
longDesc	Sets or returns a URL to a document containing a description of the image	6	1	9	Yes
lowsrc	Sets or returns a URL to a low-resolution version of an image	4	1	9	No
name	Sets or returns the name of an image	4	1	9	Yes
src	Sets or returns the URL of an image	4	1	9	Yes
useMap	Sets or returns the value of the usemap attribute of an clientside image map	5	1	9	Yes
vspace	Sets or returns the white space on the top and bottom of the image	4	1	9	Yes
width	Sets or returns the width of an image	4	1	9	Yes

Standard Properties

Property	Description	IE	F	O	W3C
className	Sets or returns the class attribute of an element	5	1	9	Yes
title	Sets or returns an element's advisory title	5	1	9	Yes

Area Object

- The Area object represents an area of an image-map (An image-map is an image with clickable regions).
- For each instance of an <area> tag in an HTML document, an Area object is created.

IE: Internet Explorer, **F:** Firefox, **O:** Opera, **W3C:** World Wide Web Consortium (Internet Standard).

Area Object Properties

Property	Description	IE	F	O	W3C
accessKey	Sets or returns the keyboard key to access an area	5	1	No	Yes
alt	Sets or returns an alternate text to be displayed, if a browser cannot show an area	5	1	9	Yes
coords	Sets or returns the coordinates of a clickable area in an imagemap	5	1	9	Yes
hash	Sets or returns the anchor part of the URL in an area	4	1	No	No
host	Sets or returns the hostname and port of the URL in an area	4	1	No	No
href	Sets or returns the URL of a link in an image-map	4	1	9	Yes
id	Sets or returns the id of an area	4	1	9	Yes
noHref	Sets or returns whether an area should be active or inactive	5	1	9	Yes
pathname	Sets or returns the pathname of the URL in an area	4	1	9	No
protocol	Sets or returns the protocol of the URL in an area	4	1	9	No
search	Sets or returns the query string part of the URL in an area	4	1	9	No
shape	Sets or returns the shape of an area in an image-map	5	1	9	Yes
tabIndex	Sets or returns the tab order for an area	5	1	9	Yes
target	Sets or returns where to open the link-URL in an area	4	1	9	Yes

Standard Properties

Property	Description	IE	F	O	W3C
className	Sets or returns the class attribute of an element	5	1	9	Yes
dir	Sets or returns the direction of text	5	1	9	Yes
lang	Sets or returns the language code for an element	5	1	9	Yes
title	Sets or returns an element's advisory title	5	1	9	Yes

Navigator Object

- The Navigator object is actually a JavaScript object, not an HTML DOM object.
- The Navigator object is automatically created by the JavaScript runtime engine and contains information about the client browser.

IE: Internet Explorer, **F:** Firefox, **O:** Opera.

Navigator Object Collections

Collection	Description	IE	F	O
plugins[]	Returns a reference to all embedded objects in the document	4	1	9

Navigator Object Properties

Property	Description	IE	F	O
appCodeName	Returns the code name of the browser	4	1	9
appMinorVersion	Returns the minor version of the browser	4	No	No
appName	Returns the name of the browser	4	1	9
appVersion	Returns the platform and version of the browser	4	1	9
browserLanguage	Returns the current browser language	4	No	9
cookieEnabled	Returns a Boolean value that specifies whether cookies are enabled in the browser	4	1	9
cpuClass	Returns the CPU class of the browser's system	4	No	No
onLine	Returns a Boolean value that specifies whether the system is in offline mode	4	No	No
platform	Returns the operating system platform	4	1	9
systemLanguage	Returns the default language used by the OS	4	No	No

userAgent	Returns the value of the user-agent header sent by the client to the server	4	1	9
userLanguage	Returns the OS' natural language setting	4	No	9

Navigator Object Methods

Method	Description	IE	F	O
javaEnabled()	Specifies whether or not the browser has Java enabled	4	1	9
taintEnabled()	Specifies whether or not the browser has data tainting enabled	4	1	9

Zip Code Validation

```
<!-- TWO STEPS TO INSTALL ZIP CODE VALIDATION:
1.     Copy the coding into the HEAD of your HTML document
2.     Add the last code into the BODY of your HTML document  -->
<!-- STEP ONE: Paste this code into the HEAD of your HTML document  -->
<HEAD>
<SCRIPT LANGUAGE="JavaScript">
<!-- Original:  Brian Swalwell -->
<!-- This script and many more are available free online at -->
<!-- The JavaScript Source!! http://javascript.internet.com -->
<!-- Begin function validateZIP(field) { var valid = "0123456789-"; var hyphencount
= 0; if (field.length!=5 && field.length!=10) { alert("Please enter your 5 digit or
5 digit+4 zip code."); return false;
} for (var i=0; i < field.length; i++) { temp = "" + field.substring(i, i+1); if
(temp == "-") hyphencount++; if (valid.indexOf(temp) == "-1") { alert("Invalid
characters in your zip code.  Please try again."); return false;
} if ((hyphencount > 1) || ((field.length==10) && ""+field.charAt(5)!="-")) {
alert("The hyphen character should be used with a properly formatted 5 digit+four
zip code, like '12345-
6789'.   Please try again."); return false;
   } } return true; }
//  End -->
</script>
</HEAD>
 <!-- STEP TWO: Copy this code into the BODY of your HTML document  -->
<BODY>
<center>
```

```
<form name=zip onSubmit="return validateZIP(this.zip.value)">
Zip: <input type=text size=30 name=zip>
<input type=submit value="Submit">
</form>
</center>
<p><center>
<font face="arial, helvetica" size="-2">Free JavaScripts provided<br> by <a
href="http://javascriptsource.com">The JavaScript Source</a></font>
</center><p>
```