HTML

Introduction to HTML

- HTML is a language for describing Web pages.
- HTML stands for HyperTextMarkup Language.
- HTML is not a programming language, it is a markup language.
- A markup language is a collection of markup tags.
- HTML uses markup tags to describe Web pages.

Tags:

- HTML markup tags are usually called HTML tags or just tags.
- HTML tags are keywords surrounded by angle brackets like <html>.
- HTML tags normally come in pairs, like and .
- The first tag in a pair is the start tag; the second tag is the end tag.
- Start and end tags are also called opening tags and closing tags.
- HTML documents contain HTML tags and plain text.
- HTML documents are also called Web pages

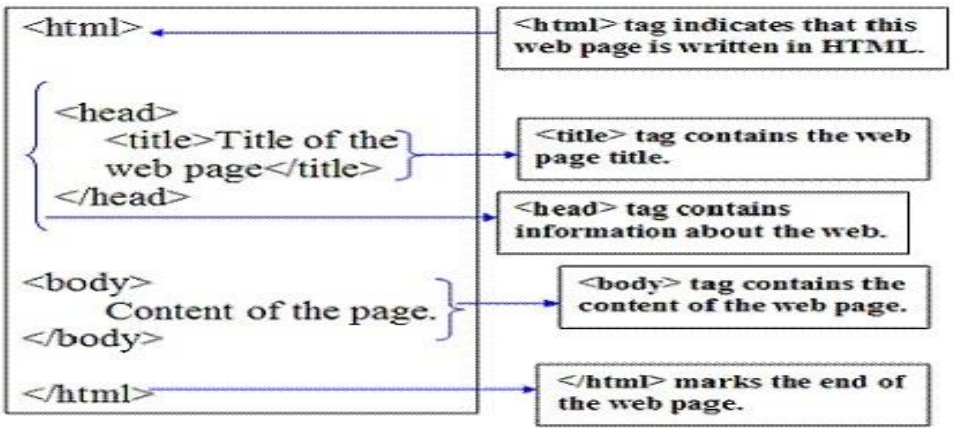
Example :

```
<!DOCTYPE html>
<html>
    <body>
        <h1>My First Heading</h1>
        <p>My first paragraph</p>
    </body>
</html>
```

In the previous code example,

- The DOCTYPE declaration defines the document type
- The text between <html> and </html> describes the Web page.
- The text between <body> and </body> is the visible page content.
- The text between <h1> and </h1> is displayed as a heading.
- The text between and is displayed as a paragraph.

The HTML Document Structure:



HTML Editors:

In this tutorial , we use a plan text editor (like Notepad) to edit HTML. Instead of writing plain text, however, professional web developers often prefer using HTML editors like FrontPage or Dreamweaver because they offer code writing shortcuts and helpful features.

.HTM or .HTML Extension

When you save an HTML file, you can use either the .html or the .html extension. It is a habit from the past, when the software only allow three letters in file extensions.

HTML Elements

HTML documents are defined by HTML elements. An HTML element is every thing between the start tag and the end tag. The start tag is often called the opening tag. The end tag is often called the closing tag.

Opening Tag	Element Content	Closing Tag
	This is a paragraph	
	This is a link	

HTML element syntax:

HTML elements followa certain format regardless of how the element isused.

- An HTML element starts with a start tag/opening tag.
- An HTML element ends with an end tag/closing tag. The element content is everything between the start and the end tag.
- Some HTML elements have emptycontent. Emptyelements are closed in the start tag. Most HTML elements canhave attributes.

Nested Elements:

HTML documents consistof nested HTML elements.

The following example contains three HTML elements. Notice that the elementis nested inthe <body> element, which in turnis nested in the <html>element.

```
<html>
    <body>
        <p>This is my first paragraph</p>
    </body>
</html>
```

Standard HTML Attributes:

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

Defining Attribute Values:

Attribute values should always be enclosed within quotation marks. While "double quotes" are the most common, single-style quotes are also allowed.

For example: name="rama"

As another example,

HTML links are defined with the <a> tag.The Web address, surroundedby quotation marks, is the value ofthe attribute of the link element.

This is a link to the bvrice Web site.

HTML HEADINGS:

The heading elements are used to create "headlines" in documents. Six different levels of headings are supported: <H1>, <H2>, <H3>, <H4>, <H5>, and <H6>. These range in importance from <H1>, the most important, to <H6>, the least important. Most browsers display headings in larger and/or bolder font than normal text.

<h1>This is a Heading 1</h1>

<h2>This is a Heading 2</h2>

<h3>This is a Heading 3</h3>

<h4>This is a Heading 4</h4>

<h5>This is a Heading 5</h5>

<h6>This is a Heading 6</h6>

An attribute that aligns the text left, right, or centre can be added to the heading elements. By default, headings are usually left-aligned, but by setting the ALIGN attribute of the various heading elements, the text may be aligned to the right, left, or centre of the screen. The following example shows ALIGN attribute for headings

```
<html>
<head>
<title>Heading Alignment Example</title>
</head>
<body>
<h1 ALIGN="left">Aligned Left</h1>
<h1 ALIGN="center">Aligned Center</h1>
<h1 ALIGN="right">Aligned Right</h1>
</body>
</html>
```

HTML PARAGRAPHS:

One of the most important structuring elements is the paragraph element. Surrounding text with the <P> and </P> tags indicates that the text is a logical paragraph unit. Normally, the browser places a blank line or two before the paragraph, but the exact rendering of the text depends on the browser.

Text within the <P> is normally aligns toleft . The ALIGN attribute is used to specify a left, right, or centre alignment.

```
<html>
<head>
<title>Paragraph Example</title>
</head>
<body>
<P>This is the first paragraph aligned to left </P>
<P ALIGN="centre">This is the second paragraphaligned the centre.</P>
<P ALIGN="right">This is the paragraph is aligned to the right.</P>
<P ALIGN="justify">This is  the paragraph is aligned to the justify. </P>
</body>
</html>
```

BREAKS:

HTML Line Breaks Use the
 tag if you want a line break (a new line) withoutstarting a new paragraph. The
 elementis an empty HTML element. It has noend tag.

```
<html>
<body>
<p>This is<br />a para-<br />graph with line breaks</p>
</body>
</html>
```

HTML Comments

Comments can be inserted inthe HTML code tomakeit more readable and under- standable. Comments areignored by the browser and are not displayed, Comments are written like this:

```
<html>
    <body>
        <!--This comment will not be displayed-->
        <p>This is a regular paragraph</p>
    </body>
</html>
```

RULER:

The <HR> element is an empty element, because it has no close tag and encloses no data. Adding an <HR> element between two paragraphs provides a simple way to put a horizontal rule between two sections.

Several attributes to the <HR> element. **SIZE** sets the bar's thickness (height). **WIDTH** sets the bar's width. **ALIGN** sets its vertical alignment. **NOSHADE** renders the bar without a surrounding shadow.

```
<html>
	<head>
		<title>horizontal rule example</title>
	</head>
	<body>
		<p>size of 10</p>
		<hr size="10">
		<p>width of 50% and no shading</p>
		<hr width="50%" noshade>
		<p>width of 200 pixels, size of 3 pixels, and no shading</p>
		<hr width="200" size="3" noshade>
		<p>width of 100, aligned right</p>
		<hr align="right" width="100">
		<p>width of 100, aligned left</p>
		<hr align="left" width="100">
		<p>width of 100, aligned center</p>
		<hr align="center" width="100">
</body>
</html>
```

PREFORMED TEXT

By using the preformatted text control we cancontrol the line breaks, spaces, andcharacterwidths withthe <pre> tag.

```
<html>
	body>
		<pre> this is preformatted text. it preserves both spaces and line
		breaks and shows the text in a monospace font. </pre>
		<p>the pre tag is good for displaying computer code:</p>
		<pre> for i = 1 to 10
		printi
		nexti
		</pre>
	</body>
</html>
```

QUOTATIONS

The following example shows how to tohandle longand shortquotations.

```
<html>
<body>
	A blockquote quotation:
		<blockquote> This is a long quotation. This is a long quotation. This
		is a long quotation. This is a long quotation. This is a
		long quotation.
		</blockquote>
		<p><b>The browser inserts line breaks and margins for a
```

```
        blockquote element.</b>
        </p>
    A short quotation:
    <q>This is a short quotation</q>
<p><b>The q element does not render as anything special.</ b></p>
</body>
</html>
```

ABBREVIATIONS AND ACRONYMS

The following example demonstrates how to handle an abbreviationor anacronym.

```
<html>
    <body>
        <abbr title="United Nations">UN</abbr><br /><acronym
        title="World Wide Web">WWW</acronym>
        <p>The title attribute is used to show the spelled-out version
        when holding the mouse pointer over the acronym or
        abbreviation
        .</p>
</body>
</html>
```

ADDRESS

The <ADDRESS> element is used to surround information, such as the signature of the person who created the page, or the address of the organization the page is about.

For example,

```
<ADDRESS>Padmasri Dr
BVRICE<BR>VISHNUPUR<BR>BHIMAVARAM<BR>534201<BR>www.bvrice.edu
.in

</ADDRESS>might be inserted toward the bottom of every page throughout a
Web site.
```

TEXT-LEVEL ELEMENTS

Text elements in HTML come in two basic flavors: physical and logical.

Physical elements, such as <B> for bold and <I> for italic, are used to specify how text should be rendered.

Logical elements, such as <STRONG> and <EM>, indicate what text is, but not necessarily how it should look.

PHYSICAL ELEMENTS

Element	**Element Type**
…	italics
…	bold

<tt> … </tt>	typewriter (monospaced)
<u> … </u>	underline
<strike> … </strike>	strikethrough
<s>… </s>	alternative element form of strikethrough
_…	subscript
[…]	superscript
<big> … </big>	bigger font (one font size bigger)
…	smaller font (one font size smaller)

The following example code shows the basic use of the physical text-formatting elements:

```
<HTML>
<HEAD>
<TITLE>Physical Text Elements</TITLE>
</HEAD>
<BODY>
<H1 ALIGN="center">Physical Text Elements</H1>
<HR>
This is <B>Bold</B><BR>
This is <I>Italic</I><BR>
This is <TT>Monospaced</TT><BR>
This is <U>Underlined</U><BR>
This is <STRIKE>Strike-through</STRIKE><BR>
This is also <S>Strike-through</S><BR>
This is <BIG>Big</BIG><BR>
This is even <BIG><BIG>Bigger</BIG></BIG><BR>
This is <SMALL>Small</SMALL><BR>
This is even <SMALL><SMALL>Smaller</SMALL></SMALL><BR>
This is <SUP>Superscript</SUP><BR>
This is <SUB>Subscript</SUB><BR>
</BODY>
</HTML>
```

Deleted and inserted text

This example demonstrates how to marka text that is deleted (strikethrough) or inserted (underscore) to a document.

```
<html>
<body>
<p> a dozen is <del>twenty</del><ins>twelve</ins> pieces </p>
<p>Most browsers will <del>overstrike</del> deleted text and
<ins>underscore</ins> inserted text. </p>
<p>Some older browsers will display deleted or inserted text as plain text.
</p>
</body>
```

</html>

LOGICAL ELEMENTS

Logical elements indicate the type of content that they enclose. The browser is relatively free to determine the presentation of that content, although expected renderings for these elements exist that are followed by nearly all browsers.

Element	**Element Type**
<ABBR> … </ABBR>	Abbreviation
<CITE> … </CITE>	Citation
<CODE> … </CODE>	Source code
<DFN> … </DFN>	Definition
<EM> … <EM>	Emphasis
<KBD> … </KBD>	Keystrokes
<SAMP> … </SAMP>	Sample (example information)
<STRONG> … </STRONG>	Strong emphasis
<VAR> … </VAR>	Programming variable

```
<HTML>
<HEAD>
<TITLE>Logical Text Elements</TITLE>
</HEAD>
<BODY>
<H1 ALIGN="center">Logical Text Elements</H1>
<HR>
This is <EM>Emphasis</EM><BR>
This is <STRONG>Strong</STRONG><BR>
This is <CITE>Citation</CITE><BR>
This is <CODE>Code</CODE><BR>
This is <DFN>Definition</DFN><BR>
This is <KBD>Keyboard</KBD><BR>
This is <SAMP>Sample</SAMP><BR>
This is <VAR>Variable</VAR><BR>
</BODY>
</HTML>
```

Sometimes, you need to put special characters within a document, such as accented letters, copyright symbols, or even the angle brackets used to enclose HTML elements.

To use such characters in an HTML document, they must be "escaped" by using a special code. All character codes take the form &code;, in which code is a word or numeric code indicating the actual character that you want to put onscreen.

Numeric Value	Named Value Symbol	Description
"	"	Quotation mark
&	&	& Ampersand
<	<	< Less than
>	>	> Greater than
		Nonbreaking space
©	©	© Copyright symbol

```
<html>
<head><title>Character Entities Example</title>
<body>
<p>Relational Operators &amp; Logical Operators</p>
<p>  5&lt; 6 </p>
<p>  6&gt; 5 </p>
<p>BVRICE   &quot;BHIMAVARAM&quot; </P>
</body>
</html>
```

TEXT ALIGNMENT

The <CENTER> Element

The <CENTER> element can enclose any form of content, which is then centred in the browser window. In early HTML, text could be centred using the following code:

<CENTER>Welcome to Big Company</CENTER>

<CENTER> can be used around an arbitrary amount of content in many different forms, including images and text

ALIGNattribute.

The<DIV>element,whichisusedto create divisioninadocument, may have the ALIGN attribute set to LEFT; CENTER; RIGHT; or, under 4, JUSTIFY. If the ALIGN attribute is not set, text generally is aligned to the left when language direction is set to LTR (left to right) and to the right when the language direction is set to RTL (right to left)

The <MULTICOL> Element:

This element allows page designers to specify text in multiple columns, The most important attribute of the <MULTICOL> element is COLS, which is set to the number of text columns to display.

The browser should attempt to flow the text evenly across columns and make the columns the same height, except for the last column, which may be shorter depending on the amount of text in the columns.

The element also supports the attribute GUTTER, which is used to specify the gutter space between columns in pixels. By default, the gutter width (if unspecified) is 10 pixels. The last attribute supported by <MULTICOL> is WIDTH, which specifies the width of each column in pixels

<HTML>
<HEAD>
<TITLE>MULTICOL Example</TITLE>
</HEAD>
<BODY>
<MULTICOL COLS="2" GUTTER="50" WIDTH="80%">
The rain in Spain falls mainly on the plain. Now is the time for all good men tocome to the aid of the country. There's no business like show business. The rain in Spain falls mainly on the plain. Now is the time for all good men to come to the aid of the country. There's no business like show business. The rain in Spain falls mainly on the plain. Now is the time for all good men to come to the aid of the country. There's no business like show business. The rain in Spain falls mainly on the plain. Now is the time for all good men to come to the aid of the country. There's no business like show business. The rain in Spain falls mainly on the plain. The rain in Spain falls mainly on the plain.
</MULTICOL>
</BODY>
</HTML>

FONTS:

The<FONT>element, which was used to specify the size and , starting with Navigator2,color of text using the SIZE and COLOR attributes. Microsoft later added an attribute called FACE to indicate which font type should be used.

It is possible to make a certain portion of text a particular color by enclosing it within the <FONT> element and setting the COLOR attribute equal to a valid color name such as red or an equivalent value such as #FF0000.

<FONT COLOR="red">This is important.</FONT>

It is also possible to set the relative size of type by setting the SIZE attribute of the <FONT> element. In a Web page, there are seven relative sizes for text numbered from 1 to 7, where 1 is the smallest text in a document and 7 is the largest. To set some text into the largest size, use <FONT SIZE="7">This is big</FONT>. By default, the size of text is 3.

<FONT COLOR="red" SIZE=”5”>This is important.</FONT>

The FACE attribute can be set to the name of the font to render the text.

<FONT FACE="Britannic Bold">This is important.</FONT>

Using the FACE attribute, it is possible to specify a comma-delimited list of fonts to try one by one before defaulting to the normal proportional or fixed-width font. The fragment shown here would try first Arial, then Helvetica, and finally a generic sans-serif font before giving up and using whatever the current browser font is.

<FONT FACE="Arial, Helvetica, Sans-Serif">This should be in a different font.</FONT>

Document-Wide Font Settings:

In some cases, it may be appropriate to change the font size, color, or face, document-wide.

To do this, use the <BASEFONT> element in the <HEAD> of the document. The <BASEFONT> should only occur once in the document and has major attributes COLOR, FACE, and SIZE.

Like the <FONT> element, COLOR should be set to an RGB hexadecimal equivalent value or color name.

FACE should be set to a font name or comma-delimited list of fonts. SIZE should be set to a size value between 1 and 7.

Document-Wide Color Attributes for <BODY>:

The <BODY> element has numerous attribute state can be used to affect the display of content in the body of the document, including setting the background color, the color of text, and the color of links. One of the most commonly used <BODY>element attributes, BGCOLOR defines the document's background color.

To create a white background, the attribute could beset to <BODY BGCOLOR="#FFFFFF"> (hexadecimal) or simply <BODYBGCOLOR="white">

The TEXT attribute of the <BODY> element defines the color of text in the entire document. The attribute takesacolor in the form of either ahex code or color name. So

<BODYBGCOLOR="white"TEXT="green">wouldcreateawhitepagewithgreentext.

Besides the body text, it is also possible to define the colors of links by setting the <BODY> element attributes: LINK, ALINK, and VLINK.

LINK defines the color of unvisited links in a document. A LINK defines the color of the link as it is being clicked. VLINK defines the color of alink after it has been visited.

```
<HTML>
<HEAD>
<TITLE>Colors</TITLE>
</HEAD>
<BODY BGCOLOR="#FFFFFF" TEXT="#008000"
LINK="#FF0000"VLINK="#FF00FF" ALINK="#FF0000">
...Content to color...
```

</BODY>
</HTML>

IMAGE:

In HTML, imagesare definedwith the <img> tag. The imgtag is empty, which means that it contains attributesonly and it has no closing tag. To display an image on a page, you need to use the src attribute. src stands for "source". The value of the src attribute is the URL of the image.

The syntaxofdefiningan image:

<imgsrc=" url " />

The URLpoints to the location or address where the image is stored. An image file named "boat.gif" located in the directory "images" on www.bvrice.edu.inhas the URL:http://www.bvrice.edu.in/images/boat.gif

The browser putsthe image wherethe image tag occurs inthedocument. If youput an image tagbetween two paragraphs, the browser shows the firstparagraph, then the image, and the the second paragraph.

The following example demonstrates how to insert images to your Web page.

```
<html>
<body>
    <p>An image: <imgsrc="constr4.gif" width="144" height="50" /></p>
</body>
</html>
```

Insert Images from Different Locations

```
<html>
<body>
<p>An image from another folder:</p>
<imgsrc="/images/chrome.gif" width="33" height="32" />
<p>An image from w3schools:</p>
<imgsrc="http://www.bvrice.edu.in/images/w3schools_green. jpg"
width="104" height="142" />
</body>
</html>
```

Background Images

The next example demonstrates how to add a backgroundimage to an HTML page.

```
<html>
<body background="background.jpg">
<h3>Look: A background image!</h3>
```

Both gif and jpg files can be used as HTML backgrounds.</ p>
If the image is smaller than the page, the image will re- peat itself.
</body>
</html>

Aligning ImagesAdjusting Image sizes:

<html>
<body>
The text is aligned with the image <imgsrc="hackanm.gif" align="bottom" width="48" height="48" /> at the bottom.
The text is aligned with the image <imgsrc="hackanm.gif" align="middle" width="48" height="48" /> in the middle.
The text is aligned with the image <imgsrc="hackanm.gif" align="top" width="48" height="48" /> at the top.
Note: The bottom alignment is the default!
</body>
</html>

Floating Images:

In this example, howto letan image float to the left or right of a paragraph.

<html>
<body>
<imgsrc="hackanm.gif" align="left" width="48" height="48" /> A paragraph with an image. The align attribute of the image is set to "left". The image will float to the left of this text.

<imgsrc="hackanm.gif" align="right" width="48" height="48" /> A paragraph with an image. The align attribute of the im- age is set to "right". The image will float to the right of this text.
</body>
</html>

Alt attribute:

The alt attributeis used to define an alternate text for an image. Thealtattribute tells the reader what he or she is missing on a pageif thebrowser can’t load images. The browser will then display the alternate text instead of theimage. The value ofthe alt attribute is an author-definedtext:

<imgsrc="boat.gif" alt="Big Boat" />

The HSPACE attribute is used to insert a buffer of horizontal space on the left and right of an image, while the VSPACE attribute is used to insert a buffer

of vertical space in between the top and bottom of the image and other objects. The value of both attributes should be a positive number of pixels. While under some browsers it may be possible to set the attribute values to percentage values.

Creating an Image Map:

The following example demonstrate howto create an image map with clickable regions. Each of the regions is a hyperlink.

```
<html>
<body>
<p>Click on the sun or on one of the planets to watch it closer:</p>
<imgsrc="planets.gif" width="145" height="126" alt="Planets"
usemap="#planetmap" />
<map name="planetmap">
<area shape="rect" coords="0,0,82,126" alt="Sun" href="sun. htm" />
<area shape="circle" coords="90,58,3" alt="Mercury" href="mercur.htm" />
<area shape="circle" coords="124,58,8" alt="Venus" href="venus.htm" />
</map>
</body>
</html>
```

HTML LINKS:

A link is the "address" to a document (or a resource) located on the WorldWide Web or elsewhere within yourown Web server. Both typesof links areshown in the following code example.

href attribute:

The hrefattribute defines the link"address".

This <a>element defines a link to BVRICE web site:

```
<a href="http://www.bvrice.edu.in/">Visit BVRICE</a>
```

```
<html>
<body>
<p><a href="lastpage.htm">This text</a> is a link to a page on this Web
site. </p>
<p><a href="http://www.microsoft.com/">This text</a> is a link to a
page on the World Wide Web. </p>
</body>
</html>
```

The target attribute:

The targetattribute defineswhere the linked document will be opened. The following code example opens thedocument in a new browser window:

```
<html>
```

```
<body>
<a href=http://www.bvrice.edu.in/ target="_blank">Visit BVRICE</a>
<a href="lastpage.htm" target="_blank">Last Page</a>
<p>If you set the target attribute of a link to "_blank", the link will open in
a new window. </p>
</body>
</html>
```

Creating an image Link:

The following exampledemonstrates how to use an image as a link. Click on the image togo tothe linked page.

```
<html>
<body>
<p>Create a link attached to an image:
<a href="default.htm"><imgsrc="smiley.gif" alt="HTML tutorial"
width="32" height="32" /></a></p>
<p>No border around the image, but still a link:
<a href="default.htm"><img border="0" src="smiley.gif" alt="HTML
tutorial" width="32" height="32" /></a></p>
</body>
</html>
```

Links on the same page :

The following code example demonstrates howto use a link to jump to another partof a document.

```
<html>
<body>
<p><a href="#C4">See also Chapter 4.</a></p>
<p><a href="#C17">See also Chapter 17.</a></p>
<h2>Chapter 1</h2><p>This chapter explains HTML</p>
<h2>Chapter 2</h2><p>This chapter explains HTML</p>
<h2>Chapter 3</h2><p>This chapter explains bablabla</p>
<h2><a name="C4">Chapter 4</a></h2><p>This chapter explains
HTML</p>
<h2>Chapter 5</h2><p>This chapter explains HTML</p>
<h2>Chapter 6</h2><p>This chapter explains HTML</p>
<h2>Chapter 7</h2><p>This chapter explains HTML</p>
<h2>Chapter 8</h2><p>This chapter explains HTML</p>
<h2>Chapter 9</h2><p>This chapter explains HTML</p>
<h2>Chapter 10</h2><p>This chapter explains HTML</p>
<h2>Chapter 11</h2><p>This chapter explains HTML</p>
<h2>Chapter 12</h2><p>This chapter explains HTML</p>
<h2>Chapter 13</h2><p>This chapter explains HTML</p>
```

```
<h2>Chapter 14</h2><p>This chapter explains HTML</p>
<h2>Chapter 15</h2><p>This chapter explains HTML</p>
<h2>Chapter 16</h2><p>This chapter explains HTML</p>
<h2><a name="C17"></a>Chapter 17</h2><p>This chapter explains
HTML</p>
</body>
</html>
```

Creating a mailto:

Link Thefollowing example demonstrateshow to linkto ane-mail address and generatea newe-mail message inyour default e-mail application (this works only if you have mail installed).

```
<html>
<body>
<p>This is a mail link:
<a href="mailto:someone@microsoft.com?subject=Hello%20 again"> Send
Mail</a>
</p>
<p><b>Note:</b> Spaces between words should be replaced by %20 to
<b>ensure</b> that the browser will display your text properly. </p>
</body>
</html>
```

UNORDERED LISTS HTML:

It supports ordered, unordered and definition lists. You can also nest one list within another.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 startswith the <li> tag.

```
<html>
<body>
<h4>An Unordered List:</h4>
<ul>
<li>Coffee</li>
<li>Tea</li>
<li>Milk</li>
</ul>
</body>
</html>
```

Inside a list item, you can put paragraphs, line breaks, images, links,otherlists, and so on. You can display differentkindsof bullets in an unorderedlist by using the type attribute. The following code shows listsmarked with discs, circles, and squares.

```
<html>
<body>
<h4>Disc bullets list:</h4>
<ul type="disc">
<li>Apples</li>
<li>Bananas</li>
<li>Lemons</li>
</ul>
<h4>Circle bullets list:</h4>
<ul type="circle">
<li>Apples</li>
<li>Bananas</li>
<li>Lemons</li>
</ul>
<h4>Square bullets list:</h4>
<ul type="square">
<li>Apples</li>
<li>Bananas</li>
<li>Lemons</li>
</ul>
</body>
</html>
```

Ordered Lists:

An ordered list is also a list of items;thelist items are numbered sequentially rather than bulleted. An ordered list starts withthe <ol> tag.

Each list itemstartswith the <li> tag. The following program shows howthe ordered list appears in the browser.

```
<html>
<body>
<h4>An Ordered List:</h4>
<ol>
<li>Coffee</li>
<li>Tea</li>
<li>Milk</li>
</ol>
</body>
</html>
```

Different types of Ordering :

You can display different kinds of orderedlists by usingthe type attribute. The following code shows listswith uppercase and lowercase letters and Roman numerals.

```
<html>
<body>
<h4>Letters list:</h4>
<ol type="A">
<li>Apples</li>
<li>Bananas</li>
<li>Lemons</li>
</ol>
<h4>Lowercase letters list:</h4>
<ol type="a">
<li>Apples</li>
<li>Bananas</li>
<li>Lemons</li>
</ol>
</body>
</html>

<html>
<body>
<h4>Roman numbers list:</h4>
<ol type="I">
<li>Apples</li>
<li>Bananas</li>
<li>Lemons</li>
</ol>
<h4>Lowercase Roman numbers list:</h4>
<ol type="i">
<li>Apples</li>
<li>Bananas</li>
<li>Lemons</li>
</ol>
</body>
</html>
```

Definition Lists:

A definition list is not a list of single items. It is a list of items(terms), together with a description of each item(term).

A definitionlist starts with a <dl> tag (definition list).

Each term startswith a <dt> tag (definition term).

Each description startswith a <dd> tag (definition description).

The following code shows the definition list in in a browser.

```
<html>
<body>
<h4>A Definition List:</h4>
```

```
<dl>
<dt>Coffee</dt>
<dd>Black hot drink</dd>
<dt>Milk</dt>
<dd>White cold drink</dd>
</dl>
</body>
</html>
```

Inside the <dd> tag you canput paragraphs, line breaks, images, links,otherlists, and so on.

Nested Lists:

Anested list is a list within another list. Usually the second list is indented another level and the item markerswill appear differently thantheoriginallist,

```
<html>
<body>
<h4>A nested List:</h4>
<ul>
<li>Coffee</li>
<li>Tea
<ul>
<li>Black tea</li>
<li>Green tea</li>
</ul>
</li>
<li>Milk</li>
</ul>
</body>
</html>
```

TABLES:

Creating HTML tables:

Tables are an excellent way to organize and display informationon a page. Tables are defined using the <table> tag. A table isdivided into rows with the <tr>tag, and each row is divided into data cells usingthe <td> tag.

The letters td stand for "table data," which is the content of a data cell. A data cell can contain text, images, lists, paragraphs, forms, horizontal rules, tables, and so on.

A basic table includes the followingtags:

Each table starts with a **table** tag.

Each tablerowstartswitha **tr** tag.

Each table data (cell) startswith a **td**tag.

```
<html>
<body>
<h4>One column:</h4>
<table border="1">
<tr>
<td>100</td>
</tr>
</table>
</body>
</html>
```

The followingcodecreates a table with one row and three columns.

```
<html>
<body>
<table border="1">
<tr>
<td>100</td>
<td>200</td>
<td>300</td>
</tr>
</table>
</body>
</html>
```

The following code creates a table with two rows andthreecolumns.

```
<html>
<body>
<table border="1">
<tr>
<td>100</td>
<td>200</td>
<td>300</td>
</tr>
<tr>
<td>400</td>
<td>500</td>
```

```
<td>600</td>
</tr>
</table>
</body>
</html>
```

The border attribute controlsthe appearance of the table’sborders or lines. The default border is 0, soif youdo not specify a border attribute, the table is displayed withoutany borders.

```
<html>
<body>
<h4>With a normal border:</h4>
<table border="1">
<tr>
<td>First</td>
<td>Row</td>
</tr>
<tr>
<td>Second</td>
<td>Row</td>
</tr>
</table>
<h4>With a thick border:</h4>
<table border="8">
<tr>
<td>First</td>
<td>Row</td>
</tr>
<tr>
<td>Second</td>
<td>Row</td>
</tr>
</table>
<h4>With a very thick border:</h4>
<table border="15">
<tr>
<td>First</td>
<td>Row</td>
</tr>
<tr>
<td>Second</td>
<td>Row</td>
</tr>
</table>
</body>
```

```
</html>
```

Headings in a table Table:Headings are defined with the <th> tag. The following code showsheading with tablein thebrowser.

```
<html>
<body>
<table border="1">
<tr>
<th>Heading</th>
<th>Another Heading</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>

<h4>Vertical headers:</h4>
<table border="1">
<tr>
<th>First Name:</th>
<td>Bill Gates</td>
</tr>
<tr>
<th>Telephone:</th>
<td>555 777 1854</td>
</tr>
<tr>
<th>Telephone:</th>
<td>555 777 1855</td>
</tr>
</table>
</body>
</html>
```

Cells spanning Multiple Columns& Rows:

Cell spanning means merging of cells. There are two types of spanning column spanning and row spanning . Column Span means merging of two or more horizontal cells in to a single cell. Row span means merging of two or more vertical cells in to a single cell . The example demonstrates Column span and row span .

```
<html>
<body>
<h4>Cell that spans two columns:</h4>
<table border="1">
<tr>
<th>Name</th>
<thcolspan="2">Telephone</th>
</tr>
<tr>
<td>Bill Gates</td>
<td>555 77 854</td>
<td>555 77 855</td>
</tr>
</table>
<h4>Cell that spans two rows:</h4>
<table border="1">
<tr>
<th>First Name:</th>
<td>Bill Gates</td>
</tr>
<tr>
<throwspan="2">Telephone:</th>
<td>555 77 854</td>
</tr>
<tr>
<td>555 77 855</td>
</tr>
</table>
</body>
</html>
```

Cell padding: This example demonstrates how to use cell padding to create more white space between the cell content and its borders.

```
<html>
<body>
<h4>Without cellpadding:</h4>
<table border="1">
<tr>
<td>First</td>
<td>Row</td>
</tr>
<tr>
<td>Second</td>
<td>Row</td>
</tr>
```

```
</table>
<h4>With cellpadding:</h4>
<table border="1" cellpadding="10">
<tr>
<td>First</td>
<td>Row</td>
</tr>
<tr>
<td>Second</td>
<td>Row</td>
</tr>
</table>
</body>
</html>
```

Cell spacing: This example demonstrates how to use cellspacing to increasethe distance between the cells

```
<html>
<body>
<h4>Without cellspacing:</h4>
<table border="1">
<tr>
<td>First</td>
<td>Row</td>
</tr>
<tr>
<td>Second</td>
<td>Row</td>
</tr>
</table>
<h4>With cellspacing:</h4>
<table border="1" cellspacing="10">
<tr>
<td>First</td>
<td>Row</td>
</tr>
<tr>
<td>Second</td>
<td>Row</td>
</tr>
</table>
</body>
</html>
```

FRAMES:

With frames, we can displaymorethan one HTML document in the same browser window. Each HTML document iscalled frame, and each frameisindependent of the others.

Frameset tag the<frameset> tag defines howto divide the window into frames. Eachframe- set defines a set of rows or columns.

Vertical Frameset

The following example demonstrates howto make a vertical frameset with three different documents.

```
<html>
<frameset cols="25%,50%,25%">
<frame src="frame_a.htm">
<frame src="frame_b.htm">
<frame src="frame_c.htm">
</frameset>
</html>
```

Horizontal Frameset

The following exampledemonstrates how to make a horizontal frameset withthreedifferent documents.

```
<html>
<frameset rows="25%,50%,25%">
<frame src="frame_a.htm">
<frame src="frame_b.htm">
<frame src="frame_c.htm">
</frameset>
</html>
```

Designing with Frames:

Ifa frame has visible borders, the user can resize it by dragging the border. To prevent a user from doing this, we can addnoresize="no resize" to the <frame> tag.

noframes tag:

Although they are less common these days, it’s a good ideato add the<noframes> tag for older or text-based browsersthat do notsupport frames.

You cannot use the <body> tags together with the <frameset> tags. However, ifyou add a <noframes>tag containing some text for browsers thatdo not support frames, you will have to enclose the text in <body> tags.

```
<html>
<frameset cols="25%,50%,25%">
<frame src="frame_a.htm">
<frame src="frame_b.htm">
```

```
<frame src="frame_c.htm">
<noframes>
<body>Your browser does not handle frames!</body>
</noframes>
</frameset>
</html>
```

Mixed Frameset:

Thefollowing example demonstrates how to make a framesetwith three documents and how to mixthemin rows and columns

```
<html>
<frameset rows="50%,50%">
<frame src="frame_a.htm">
<frameset cols="25%,75%">
<frame src="frame_b.htm">
<frame src="frame_c.htm">
</frameset>
</frameset>
</html>
```

noresize attribute:

This exampledemonstrates thenoresize attribute. The frames are notresizable. Unlike other frames, if you movethe mouse over the borders between theframes, you will notice that you cannot drag or move theframeborders.

```
<html>
<frameset rows="50%,50%">
<frame noresize="noresize" src="frame_a.htm">
<frameset cols="25%,75%">
<frame noresize="noresize" src="frame_b.htm">
<frame noresize="noresize" src="frame_c.htm">
</frameset>
</frameset>
</html>
```

Navigation Frame:

Thisexample demonstrateshow to make a navigation frame. A navigation frame contains a list of links with the second frame as the target. Thesecond frame willshowthe linked document.

```
<html>
<frameset cols="120,*">
<frame src="tryhtml_contents.htm">
<frame src="frame_a.htm" name="showframe">
</frameset>
```

</html

In the first column, thefile called tryhtml_contents.Htm contains links to three documents on the bvrice.edu.in Website. The source code for thelinks:

Frame a
Frame b
Frame c

Inline Frame:

Framescan also be used within a single HTML page. Theseare known as inline frames. The following example demonstrates how to create an inline frame

<html>
<body>
<iframesrc="default.asp"></iframe>
Some older browsers don't support iframes.
If they don't, the iframe will not be visible.
</body>
</html>

FORMS:

There are many uses for forms on the Web. The most common ones include comment response forms, order entry forms, subscription forms, registration forms, and customization forms.

The <FORM> Element A form in HTML is enclosed between the <FORM> and </FORM> tags. The form itself contains regular text; other HTML elements such as tables; and form elements such as check boxes, pull-down menus, and text fields.

ACTION Attribute

The ACTION attribute is usually set to a URL of the program that will handle the form data

<FORM ACTION="http://www.bvrice.edu.in/sample.php" METHOD="POST">

METHOD Attribute:

How data will be submitted is handled by the METHOD attribute. There are two acceptable values for the METHOD attribute: GET and POST

GET Method:

The GET method is generally the default method for browsers to submit information. In fact, HTML documents are generally retrieved by requesting a single URL from a Web server using the GET method, which is part of the HTTP protocol.

Post Method:

The POST method transmits all form input information immediately after the requested URL. In other words, once the server has received a request from a form using POST, it knows to continue "listening" for the rest of the information.

NAME Attribute:

The NAME attribute can be set to an alphanumeric value for the form.

```
<html>
<head><title>form stub</title>
</head>
<body>
<form action="" method="post">
form field controls and standard html markup
</form>
</body>
</html>
```

Form Controls:

Form controls include text fields, password fields, multiple-line text fields, pop-up menus, scrolled lists, radio buttons, check boxes, and buttons.

Text Controls

Text controls are form fields, generally one line long, that take text input like a person's name and address, and other information. These fields are specified with the <INPUT> element.

To set a text entry control, use the <INPUT> element and set the TYPE attribute equal to "TEXT":

<INPUTTYPE="TEXT" NAME="customername">

All form elements should be named. NAME="CustomerName" is used to create a text field to collect a customer's name on an order form.

To set the size of the field in characters, use the **SIZE** attribute. For example,

<INPUT TYPE="TEXT" NAME="CustomerName" SIZE="40">

The value of the SIZE field for an <INPUT> element is the number of characters to be displayed

If you want to limit the size of the field, you need to set the value of the MAXLENGTH attribute to the maximum number of characters allowed in the field.

<INPUT TYPE="TEXT" NAME="CustomerName" SIZE="30" MAXLENGTH="60">

```
<html>
<head><title>text field example</title></head>
<body>
<h1 align="center">gadget order form</h1><hr>
```

```
<FORM ACTION=""METHOD="POST">
<B>Customer Name:</B>
<INPUT TYPE="text" NAME="CustomerName" SIZE="25"
MAXLENGTH="35">
</FORM><hr>
</body>
</html>
```

PASSWORD FIELDS:

The password style of form control is the same as the simple text entry field, except that the input to the field is not revealed. In many cases, the browser may render each character as an asterisk or dot to avoid people seeing the password being entered.

To set a password form control, use the <INPUT> element, but set the TYPE attribute equal to PASSWORD. As with the text entry field, it is possible to specify the size of the field in characters with SIZE, the maximum entry with MAXLENGTH in characters.

```
<html>
<head><title>password field example</title></head>
<body>
<h1 align="center">gadget order form</h1>
<hr>
<FORM ACTION=""METHOD="POST">
<B>Customer Name:</B>
<INPUT TYPE="text" NAME="CustomerName" SIZE="25"
MAXLENGTH="35">
<BR>
<B>Customer ID:</B>
<INPUT TYPE="PASSWORD"NAME="CustomerID" SIZE="10"
MAXLENGTH="10">
</FORM><hr>
</body>
</html>
```

Multiple-Line Text Input

The <TEXTAREA> element is used to enter more than one line of text in a form field.

For example, to set the number of rows in the text entry area, set the ROWS attribute equal to the number of rows desired. To set the number of characters per line, set the COLS attribute. So, to define a text area of five rows of 80 characters each, use the following:

<TEXTAREA ROWS="5" COLS="80" NAME="CommentBox"></TEXTAREA>

PULL-DOWN MENUS:

HTML form controls include pull-down menus. A pull-down menu lets the user select one choice out of many possible choices

To create a pull-down menu, use the <SELECT> element. This element must include both a start and an end tag. It should only contain zero or more occurrences of the <OPTION> element. The <OPTION> elements specify the actual choices on the menu, and generally do not use a close tag

```
<SELECT NAME="GadgetType">
<OPTION>Super Gadget <OPTION>Mega Gadget
<OPTION>Mongo Gadget <OPTION>Plain Gadget
</SELECT>
```

The <SELECT> element has a NAME attribute that is used to set a unique name for the control for purposes of decoding the user selection. It is also possible to set attributes for the <OPTION> element. An occurrence of the attribute SELECTED in the <OPTION> element sets the form control to select this item by default. . However, it is possible to set the VALUE attribute for the element that will be returned instead.

```
<html>
<head><title>Pull-down Menu Example</title></head>
<body>
<H1 ALIGN="center">Gadget Order Form</H1>
<FORM ACTION="" METHOD="POST">
<B>Gadget Type:</B>
<SELECT NAME="GadgetType">
<OPTION VALUE="SG-01">Super Gadget
<OPTION VALUE="MEG-G5">Mega Gadget
<OPTION VALUE="MO-45">Mongo Gadget
<OPTION SELECTED>Gadget
</SELECT>
</FORM>
</body>
</html>
```

SCROLLED LISTS

The <SELECT> element also may contain the SIZE attribute, which is used to specify the number of items showing on the screen at once. The default value for this attribute is 1, which specifies a normal pull-down menu. Setting a positive number creates a list in a window of the specified number of rows

In many cases, scrolled lists act just like pull-down menus. However, if the <SELECT> element contains the attribute MULTIPLE, it becomes possible to select more than one entry.

```
<html>
<head><title>Scrolled List Example</title></head>
<body><H1 ALIGN="center">Gadget Order Form</H1>
```

```
<FORM ACTION="" METHOD="POST">
<B>Gadget Options:</B>
<SELECT NAME="GadgetOptions" MULTIPLE SIZE="3">
<OPTION VALUE="Hit with hammer" SELECTED>Bumps
<OPTION VALUE="Add glitter">Sparkles <OPTION VALUE="Buff
it">Polished
<OPTION SELECTED>Scratches
<OPTION>Shrink wrapped
</SELECT>
</FORM>
<hr>
</body>
</html>
```

CHECK BOXES:

If there are a few options to select from that are not mutually exclusive, it is probably better to use a group of check boxes that the user can check off. Check boxes are best used to toggle choices on and off. While it is possible to have multiple numbers of check boxes

To create a check box, use the <INPUT> element and set the TYPE attribute equal to CHECKBOX. The check box should also be named by setting the NAME attribute. For example, to create a check box asking if a user wants cheese, use some markup like

Cheese: <INPUT TYPE="CHECKBOX" NAME="Cheese">

If the check box is selected, a value of Cheese=on will be transmitted to the server. Setting a value for the check box might make more sense. Values to be transmitted instead of the default value can be set with the VALUE attribute. The code

Cheese: <INPUT TYPE="Checkbox" NAME="Extras" VALUE="Cheese">

Would send a response like ExtrasCheese to the server. It is also possible to have multiple check box controls with the same name. The code

Cheese: <INPUT TYPE="CHECKBOX" NAME="Extras" VALUE="Cheese">
Pickles: <INPUT TYPE="CHECKBOX" NAME="Extras" VALUE="Pickles">

would send multiple entries like the following to the server when both extras were selected:Extras=Cheese&Extras=Pickels

```
<html>
<head><title>Check Box Example</title></head>
<body>
<H1 ALIGN="center">Gadget Order Form</H1>
<HR>
<FORM ACTION="" METHOD="POST">
<B>Gadget Bonus Options:</B><BR>
```

```
Super-magneto: <INPUT TYPE="CHECKBOX" NAME="BONUS"
VALUE="Magnetize"><BR>
Kryptonite Coating:<INPUT TYPE="CHECKBOX" NAME="BONUS"
VALUE="Anti-Superman" CHECKED><BR>
Anti-gravity:<INPUT TYPE="CHECKBOX" NAME="BONUS"
VALUE="Anti-gravity"><BR>
</FORM>
<hr>
</body>
</html>
```

RADIO BUTTONS:

Radio buttons use a similar notation to check boxes, but only one option may be chosen among many.

Like check boxes, this form control uses the standard <INPUT TYPE=""> format. In this case, set TYPE equal to RADIO. Setting the NAME field is very importantin the case of radio buttons because it groups together controls that share the radio functionality

Another important attribute is VALUE. It is important to set each individual radio button to a different value entry

Like check boxes, the occurrence of the SELECTED attribute in the <INPUT> element will preselect the item. Only one item may be selected as a default out of a radio group

```
<html>
<head><title>Radio Button Example</title></head>
<BODY><H1 ALIGN="center">Gadget Order Form</H1><HR>
<FORM ACTION="" METHOD="POST">
<B>Gadget Color:</B><BR>
Groovy Green: <INPUT TYPE="RADIO" NAME="Color"
VALUE="Green">
Rocket Red: <INPUT TYPE="RADIO" NAME="Color" VALUE="Red"
CHECKED>Yipee! Yellow: <INPUT TYPE="RADIO" NAME="Color"
VALUE="Yellow"></FORM>
<hr>
</body>
</html>
```

RESET AND SUBMIT BUTTONS:

The <INPUT> element has two values, RESET and SUBMIT, for the TYPE attribute; these can create common buttons that are useful for just about any form. Setting the TYPE attribute for the <INPUT> element to RESET creates a button that allows the user to clear or set to default all the form controls at once. Setting the TYPE attribute for <INPUT> to SUBMIT creates a button that triggers the browser to send the contents of the form to the address specified in the ACTION attribute of the <FORM> element.

The VALUE attribute sets both the value of the button when pressed and the wording of the button. The NAME value associates an identifier with the form control.

```
<HTML>
<HEAD><TITLE>Complete Form Example</TITLE></HEAD>
<BODY>
<H1 ALIGN="center">Gadget Order Form</H1><HR>
<FORM ACTION=""METHOD="POST">
<B>Customer Name:</B>
<INPUT TYPE="TEXT"
NAME="CustomerName" SIZE="25"
MAXLENGTH="35"><BR><BR>
<B>Customer ID:</B>
<INPUT TYPE="PASSWORD"NAME="CustomerID" SIZE="10"
MAXLENGTH="10">
<BR><BR>
<B>Gadget Type:</B>
<SELECT NAME="GadgetType">
<OPTION VALUE="SG-01">Super Gadget
<OPTIONVALUE="MEG-G5">Mega Gadget
<OPTION VALUE="MO-45">Mongo Gadget
<OPTION SELECTED>Gadget
</SELECT>
<BR><BR>
<INPUT TYPE="SUBMIT" VALUE="Order Gadget"
NAME="SubmitButton">
<INPUT TYPE="RESET" VALUE="Reset Form"
 NAME="ResetButton">
</form><hr>
</body>
</html>
```

FILE FORM CONTROL

This form control is used for file uploading. The field generally consists of a text entry box for a filename that can be manipulated with the SIZE and MAXLENGTH attributes, as well as a button immediately to the right of the field, which is usually labelled Browse. Pressing the Browse button allows the user to browse the local system to find a file to specify for upload.

```
<html><head>
<title>File Upload Test</title></head>
<body>
<H1 ALIGN="CENTER">File Upload System</H1><HR>
      <FORM ACTION=""METHOD="POST" >
<B>File Description:</B><BR>
```

```
<INPUT TYPE="TEXT" NAME="Description" SIZE="50"
MAXLENGTH="100"><BR>
<B>Specify File to Post:</B><BR>
<INPUT TYPE="FILE" NAME="FileName">
<BR><BR><HR>
<DIV ALIGN="center">
<INPUT TYPE="SUBMIT" NAME="Submit" VALUE="Send it"></DIV>
</FORM>
</body>
</html>
```

GENERALIZED BUTTONS

By using <INPUT TYPE="BUTTON">, it is possible to create a button in the style of the submit or reset buttons that, unlike the submit or reset buttons, has no predetermined actions.

```
<INPUT TYPE="BUTTON" VALUE="Press Me!" NAME="mybutton">
```

If you click the rendering of this button, no action is triggered, and no value will be submitted.

LABEL

<LABEL> element is to associate the labeling text of form controls with the actual controls

The <LABEL> element can be associated with a form control by enclosing it as shown here:

```
<LABEL>First Name: <INPUT TYPE="TEXT" NAME="FirstName"
SIZE="20" MAXLENGTH="30"></LABEL>
```

A <LABEL> element can also be associated with a control by referring to the control's ID with the FOR attribute.

The following code fragment illustrates how the <LABEL> element with the FOR attribute would be used.

```
<TABLE>
<TR>
<TD ALIGN="RIGHT">
<LABEL FOR="CustName">Customer Name : </LABEL>
<TD ALIGN="LEFT">
<INPUT TYPE="TEXT" ID="CustName" SIZE="25" MAXLENGTH="35">
</TR>
</TABLE>
```

FIELDSET

The <FIELDSET> element can be nested; it can also have an associated <LEGEND> element to describe the enclosed items. , the <LEGEND> element does support the ALIGN attribute, which can be used to specify where the description will be rendered in relation to the group of form items; its values are TOP (the default value), BOTTOM, LEFT, or RIGHT.

```
<html>
<head><title>Fieldset and Legend Example</title></head>
<body>
<FORM ACTION="index.php" METHOD="POST">
<FIELDSET>
<LEGEND>Customer Identification</LEGEND><BR>
<LABEL>Customer Name:
<INPUT TYPE="TEXT" ID="CustomerName" SIZE="25">
</LABEL>
<BR><BR>
<LABEL>Customer ID:
<INPUT TYPE="PASSWORD" ID="CustomerID" SIZE="8"
MAXLENGTH="8">
</LABEL><BR>
</fieldset>
</form>
</body>
</html>
```

The HTML <head> Element

The <head> element is a container for all the head elements. Elements inside <head> can include scripts, instruct the browser where to find style sheets, provide meta information, and more.

The following tags can be added to the head section:**<title>, <style>, <meta>, <link>, <script>, <noscript>, and <base>**.

1.The HTML <title> Element

The <title> tag defines the title of the document.

The <title> element:

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

A simplified HTML document:

```
<!DOCTYPE html>
<html>
<head>
<title>Title of the document</title>
```

```
</head>
<body>
The content of the document.....
</body>
</html>
```

2. The HTML <base> Element

The <base> tag specifies the base URL/target for all relative URLs in a page:

```
<!DOCTYPE html>
<html>
<head>
<base href="http://bvrice.edu.in/images/bvr/" target="_blank">
</head>
<body>
<imgsrc="fc1.jpg" width="24" height="39"> - Notice that we have only
specified a relative address for the image. Since we have specified a
base URL in the head section, the browser will look for the image at
"http://www.bvrice.edu.in/images/bvr/fc1.jpg"
<br><br>
<ahref="http://bvrice.edu.in/">BVRICE</a> - Notice that the link opens
in a new window, even if it has no target="_blank" attribute. This
is because the target attribute of the base element is set to
"_blank".
</body>
</html>
```

3. The HTML <link> Element

The <link> tag defines the relationship between a document and an external resource.

The <link> tag is most used to link to style sheets:

styles.css
```
body {background-color:yellow;}
p {color:blue;}
```

```
<!DOCTYPE html>
<html>
<head>
<link rel="stylesheet" type="text/css" href="styles.css">
</head>
<body>
<h1>I am formatted with a linked style sheet</h1>
<p>Me too!</p>
</body>
</html>
```

4.The HTML <style> Element

The <style> tag is used to define style information for an HTML document.

Inside the <style> element you specify how HTML elements should render in a browser:

```
<html>
<head>
<style type="text/css">
body {background-color:yellow;}
p {color:blue;}
</style>
</head>
<body>
This is body of the document
<p>This is a paragraph </p>
</body>
</html>
```

5.The HTML <meta> Element

Metadata is data (information) about data.

The <meta> tag provides metadata about the HTML document. Metadata will not be displayed on the page

Meta elements are typically used to specify page description, keywords, author of the document, last modified, and other metadata.

The metadata can be used by browsers (how to display content or reload page), search engines (keywords), or other web services.

<meta> tags always go inside the <head> element.

```
<!DOCTYPE html>
<html>
<head>
<meta name="description" content="Free Web tutorials">
<meta name="keywords" content="HTML,CSS,XML,JavaScript">
<meta name="author" content="HegeRefsnes">
</head>
<body>
<p>All meta information goes in the head section...</p>
</body>
</html>
```

6.The HTML <script> Element

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

Unit 2 - Style Sheet Basics

A style sheet is used **to formatting a particular element or set of elements** .Apply a style specification of an element is very simple, it consists of **selector**.

The selector contains the **element name** followed by **style information** within curly braces.

The **rules** are composed of property names and property values separated by colons with each rule in turn being separated by a semicolon.

The basic syntax is as follows

Selector {propery1:value1 ; ……property : valueN;}

Suppose we want to apply style rule 28 point to all h1 elements write like this

h1 { font-size : 28pt}

Suppose we want to apply style rules 28 point ,red colour and impact font to all h1 elements write like this

h1 {font-size:28pt ; color: red ; font-family:Impact ;}

The above can be write like this

```
h1 {font-size:28pt ;
    color: red ;
    font-family:Impact ;}
```

Most of the browsers property names and selectors are not case sensitive , so we can write like this

h1 {FONT-SIZE:28pt ; COLOR: red ; FONT-FAMILY:Impact ;}

There are three ways to add style to a document

1. External style sheet
2. Internal style sheet
3. Inline style sheet

1.External Style Sheet

An external style sheet is simply a plain text file containing the style specification for HTML tags. The extension of style sheet is .css (cascading style sheet)

sitestyle.css

```
body  { font-size : 10pt ;
        font-family : Serif ;
        color : black;
         background-color : white;
}
h1{   font-size : 24 pt ;
        font-family : Sans-Serif ;
        color : black;
        text-align : center ;
}

P{ text-indent : 0.5in ;
  margin-left : 50px;
```

```
margin-right : 50px;
}
a: link { color: blue; text-decoration : none;}
a: visited { color: red; text-decoration : none;}
a: active { color: red; text-decoration : none;}
a: hover { color: red; text-decoration : underline;}
```

HTML file uses this style sheet by using the **<link>** tag with in the **head** element of the document .

```
<html>
 <head><title> Style sheet linking example </title>
   <link rel="stylesheet" href="sitestyle.css" type="text/css">
 </head>
 <body>
    <h1> HTML with Style </h1>
    <p> Cascading style sheet defined by the <a
        href="http://www.w3.org">W3C </a> provides powerful page
layout
        facilities
    </p>
</body>
</html>
```

The rel attribute of link element is used to specify the type of the document to be link .

The href attribute is used to indicate the URL of the style sheet to use.

The value of type attribute is "text/css" indicate linked style sheet is cascading style sheet.

2.Internal Style Sheet

In this method we can write the style rules in the <style> tag with in the head element of HTML document. The syntax of <style> tag is as follows

```
<style type="text/css" media="all | print | screen ">
    Style rules here
</style>
```

Here , the type attribute is used to indicate the MIME type of the enclosed style sheet.

The media attribute indicates the media for which the style sheet applies. By default , the style sheet is applied to all media.

It is possible to define style sheets that are applied only to a particular output medium. The most common values are print and screen, which indicate rules applied only to page when it is printed or shown screen.

```
<html>
 <head><title> Internal Style Sheet Example </title>
         <style type="text/css">

body  { font-size : 10pt ;
        font-family : Serif ;
        color : black;
         background-color : white;
}
h1{   font-size : 24 pt ;
        font-family : Sans-Serif ;
        color : black;
        text-align : center ;
}

P{ text-indent : 0.5in ;
  margin-left : 50px;
margin-right : 50px;
}
a: link { color: blue; text-decoration : none;}
a: visited { color: red; text-decoration : none;}
a: active { color: red; text-decoration : none;}
a: hover { color: red; text-decoration : underline;}
</style>
</head>
<body>
<h1> HTML with Style </h1>
     <p> Cascading style sheet defined by the <a
          href="http://www.w3.org">W3C </a> provides powerful page
          layout  facilities
      </p>
</body>
</html>
```

Using inline Style

In this method we can add style information directly in a single element. Suppose we want to set one particular <h1> tag to 48 point, green , Arial font. This can be applied through the common style attribute of element .

Example :

```
<html>
 <head><title> Style sheet linking example </title>
```

```
  </head>
  <body>
<h1 style =”font-size: 48pt;font-family:Arial ; color:green;”> CSS INLINE</h1>
</body>
</html>
```

<u> and </u>

The can be used to apply a style to a certain section or division of a document

Eg

```
  <div style=”background-color:yellow; font-weight:bold; color:black ;”>
```

<p>In this method we can add style information directly in a single element. Suppose we want to set one particular <h1> tag to 48 point, green , Arial font. This can be applied through the common style attribute of element .</p.

<u>SELECTORS</u>

For example , consider setting the line spacing for all paragraphs using a rule such as the following

p { line-height : 150% }

For example , to set the background color of the entire document to black . we can write like this

body { background-color : black ;}

For example , to set the same back ground color and color of these tags <h1>,<h2> and <h3> , we can write like this

h1,h2,h3 {background-color : yellow ; color : black ;}

For example , each selector heading have a different size, we can write like this

h1 {font-size : 200%;}

h2 {font-size : 150%;}

h3 {font-size : 125%;}

<u>id Rules</u>

Without inline style , we can also set style of one particular element by using the class and id attributes .

<h1 id=”FirstHeading”> Welcome to demo company </h1>

Assigns a name of “FirstHeading” to this <h1> tag.

The css rule such as

#FirstHeading {background-color:green;}

would apply a green background to the element with its id attribute set to FirstHeading

Example :

```
<html>
  <head><title> ID rule example </title>
  <style type=”text/css”>
     #FirstParagraph {background-color:green;}
  </style>
 </head>
 <body>
   <p id=”FirstParagraph”>This is the first paragraph </p>
   <p> This is the second paragraph </p>
   <p> This is the third paragraph </p>
  </body>
</html>
```

Class Rules

The class attribute is used to define the name of the class to which a particular tag belongs . Many elements belongs to same class .

Writing a rule for class is very easy , simply specify the class name with a period before it.

Example

.nature {color : green ; }

Example :

```
<html>
  <head><title> class example </title>
  <style type=”text/css”>
  .veryimportant {background-color: yellow;}
</style>
  </head>
  <body>
<h1 class=”veryimportant”> Example </h1>
<p id=”veryimportant”>This is the first paragraph </p>
<p> This is the second paragraph </p>
<p class=”veryimportant”> This is the third paragraph </p>
</body>
```

```
</html>
```

For example , setting all h1 elements of the class veryimportant to have a background color of orange could be written like this

h1.veryimportant {background-color : orange ;}

example

```
<html>
  <head><title> class example </title>
  <style type=”text/css”>
  h1.veryimportant {background-color: yellow;}
</style>
  </head>
  <body>
<h1> Example </h1>
<h1> Another Example </h1>
</body>
</html>
```

Pseudoclasses

A special predefined class grouping , called pseudoclasses , is used in CSS to deal with special situations such as the first letter or line of information , the focus of the cursor , the state of links and so on.

In CSS , the presentation of link states is controlled through the pseudoclass selector a:link , a:visisted , and a: active.

```
<html>
  <head><title> Line Pseudoclass Example </title>
          <style type=”text/css”>

a: link { color: blue; text-decoration : none;}
a: visited { color: red; text-decoration : none;}
a: active { color: red; text-decoration : none;}
a: hover { color: red; text-decoration : underline;}
</style>
</head>
<body>
<h1> HTML with Style </h1>
     <p> Cascading style sheet defined by the <a
href=”http://www.w3.org”>W3C </a> provides powerful page layout facilities
</p>
</body>
</html>
```

Pseudo Elements

There are two pseudo-elements exist : fist-letter and :first-line. These selectors are used with text element such as p to affect the presentation of the first letter or first line of text .

```
<html>
  <head><title> First line and letter </title>
  <style type=”text/css”>
    p:first-line {background-color : yellow ;}
    p:first-letter {color :red ; font-size : 150% }
</style>
</head>
<body>
<p>CSS selectors can be used to select elements in a variety of interesting ways
</p>
<p>CSS selectors can be used to select elements in a variety of interesting ways
</p>
</body>
</html>
```

Font Properties

font-family

font-size

font-style

font-weight

font

Text Properties

text-transform

text-decoration

word-spacing

letter-spacing

vertical-align

text-align

text-indent

line-height

white-space

Color and Background Properties

color

background-color

background-image

background-repeat

background-position

background

Box Properties

Margin Property

Border Property

Padding Property

Unit3 & 4

DHTML?

Dynamic HTML, or DHTML, is an umbrella term for a collection of technologies used together to create interactive and animated web sites by using a combination of a static markup language (such as HTML), a client-side scripting language (such as JavaScript), a presentation definition language (such as CSS Cascading StyleSheets), and the Document Object Model (DOM).

"JavaScript is the premier client-side lightweight interpreted scripting computer programming language initially called LiveScript which is used today on the Web."

Benefits of JavaScript:

JavaScript has a number of benefits to make the web site dynamic:

- It is widely support in Web browsers
- It gives easy access to the document object and can manipulate most of them
- JavaScript can give interesting animations without taking long download time which are associated many multimedia data types
- Java Script is relatively secure

Problems with JavaScript:

There are number of benefits with JavaScript, it also consists the following problems:

- If our script does not work, then our page is useless
- Due to problems of broken scripts, Web surfers disable JavaScript support in their browser
- Scripts can run slowly and complex scripts can take a long time to start up

STRUCTURE OF JAVASCRIPT PROGRAM

JavaScript code looks like C Language. The key points that we need to apply in all scripts are listed below:

a) Each of the code is terminated by a semicolon

b) Blocks of code must be surrounded by a pair of curly braces

c) A block of code is a set of instructions that are to be executed together as a unit

d) Functions have parameters, which are passed inside parenthesis

e) Variables are declared using a keyword var

f) Scripts require neither a main function nor an exit condition

g) Execution of a script starts from the first line of code and runs to the last line of code

Including JavaScript into on HTML Document:

- Design the script as per need
- Include the script to the HTML page
- When the HTML page is opened in the browser, the script will be executed with an interpreter which is a part of the browser.
- The browser can able to debug the script and display the errors if any
- To include a script into HTML page, there two ways:

Way 1:

I f script size is very small, we can directly include into HTML code between <html>

and </html> tags as the following:

```
<html>
<head>
<script language="javascript">
Code related to JavaScript
</script>
</head>
```

<body>

……

……

</body>

</html>

Example :

ExWay 2:

If script is complex, then we can design the script in a separate file and save it as

"filename.js" and include it in HTML code as the following:

<html>

<head>

<script language="javascript" src="filename.js">

</script>

</head>

<body>

……

……

</body>

</html>

Ex:

Open a new file in Notepad and type the following code:

function myFunction() {

alert("My First JavaScript Function");

}

Save the above file with .js extension such as "myScript.js"

Open a new file in Notepad and type the following code:

<!DOCTYPE html>

<html>

<head>

<title>First External JS</title>

<body>

<h1>My Web Page</h1>

A Paragraph.

<button type="button" onclick="myFunction()">Try it</button>

```
<p><strong>Note:</strong> myFunction is stored in an external file called
                        "myScript.js".</p>
<script src="myScript.js"></script>
</body>
</html>
```

VARIABLES AND DATA TYPES

"A variable is a name assigned to a location in the computer's memory to store data." Before we can use a variable in the Javascript program, we must declare its name. We can create a variable with the var statement:

var <variablename> = <some value>;

Ex:

```
var x;
var y;
var sum;
```

We can initialize a value to a variable like this:

```
var fName = "Jack"
```

RULES FOR VARIABLE NAMES:

- Variable names are case sensitive
- They must begin with a letter or the underscore character
- We can't use spaces in names
- We can't use a reserved word as a variable name

```
<html>
<head>
<title>Variables Demo</title>
</head>
<body>
<script language="JavaScript" type="text/javascript">
        var str;
        str = "Hello";
        alert(str);
        str = 54321;
        alert(str);
</script>
</body>
</html>
```

Data Types:

• JavaScript supports five primitive data types:
– Number
– String
– Boolean
– Undefined
– Null.
• These types are referred to as primitive types because they are the basic building blocks from which more complex types can be built.
• Of the five, only number, string, and Boolean are real data types in the sense of
actually storing data.
• Undefined and null are types that arise under special circumstances.

NUMERIC DATA TYPE:

• The number type in JavaScript includes both integer and floating-point values.
• This representation permits exact representation of integers in the range -2^{53} to 2^{53} and floating-point magnitudes as large as $\pm 1.7976 \times 10^{308}$ and as small as $\pm 2.2250 \times 10^{-308}$. Valid ways to specify numbers in JavaScript

Example:

10	.333333e77	128e+100
177.5	-1.7E12	
-2.71	3.E-5	

STRINGS:

"A string is simply text. In JavaScript, a string is a sequence of characters surrounded by single or double quotes."

Ex:

```
var string1 = "This is a string";
var string2 = 'So am I';
var oneChar = "s";      ->defines a string of length one.
```

It defines a string values to be stored in string1 and string2 respectively. Strings are associated with a String object, which provides methods for manipulation and examination.

For example, we can extract characters from strings using the charAt() method.

```
var myName = "Thomas";
var thirdLetter = myName.charAt(2);
```

Booleans:

Boolean values are the simplest data type. They can contain one of two values: true orfalse. Boolean values are most commonly used as flags; variables that indicate whether a condition is true or not.

Ex:

```
var flag = true;
if (flag)   // if flag is true then increment x
{
    x = x + 1;
 }
```

Null:

The null value indicates an empty value. JavaScript provides the null keyword to
enable comparison and assignment of null values.

Ex:

```
var x;
var y = null;  // the following comparisons are true
if (x == null)    { // do something }
if (x == y)       { // do something }
```

OPERATORS

The operators in JavaScript are classified into seven types:

a) Arithmetic Operators
b) Assignment Operators
c) Relational Operators
d) Logical Operators
e) String (+) Operator
f) Conditional Operator
g) The 'typeof' operator

a) Arithmetic Operators: The Arithmetic operators are +,-,*,/,%,++,--

Ex:

```
<html>
<head>
<title>Operator Example</title>
</head>
<body>
<script language="JavaScript">
```

```
        var a = 5;
        ++a;
        alert("The value of a = " + a );
</script>
</body>
```

b) Assignment Operators: The Assignment operators are =,+=,-=,*=,/=,%=

Ex:

```
<html>
<head>
<title>Operator Example</title>
</head>
<body>
<script language="JavaScript">
        var a = 10;
        var b = 2;
        a += b;
        alert("a += b is " + a );
</script>
</body>
</html>
```

c) Relational Operators: The Relational Operators are ==,!=,<,>,<=,>=

d) Logical Operators: The Logical Operators are &&, ||, !

Ex:

```
<html>
<head>
<title>Operator Example</title>
</head>
<body>
<script language="JavaScript">
        var userID;
        var password;
        userID = prompt("Enter User ID"," ");
        password = prompt("Enter Password"," ");
if(userID == "user" && password == "secure"){
alert("Valid login")
}
```

```
else
{
alert("Invalid login")
}
</script>
</body>
</html>
```

Output

e) String (+) Operator:

In JavaScript, the '+' operator is also used to concatenate two strings.

Ex:

```
txt1="Welcome"
txt2="to L&T Infotech Ltd.!"
txt3=txt1 + " " + txt2
          ( or )
txt1="Welcome "
txt2="to L&T Infotech Ltd.!"
txt3=txt1 + txt2
```

f) Conditional Operator:

JavaScript also contains a conditional operator that assigns a value to a variable based on

some condition.

Syntax :

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

Ex:

```
Greeting = (visitor=="PRES")?"Dear President ":"Dear "
```

g) The 'typeof' operator:

To verify the type of the variable or value.

The typeof operator takes one parameter: the variable or value to check

Calling typeof on a variable or value returns one of the following values:

"undefined" if the variable is of the Undefined type.

"boolean" if the variable is of the Boolean type.

"number" if the variable is of the Number type.

"string" if the variable is of the String type.

"object" if the variable is of a reference type or of the Null type

Ex:

```
var sTemp = “test string”;
alert(typeof sTemp); //outputs “string”
alert(typeof 95); //outputs “number”
```

STATEMENTS IN JAVASCRIPT.

Java script Statements are divided into the categories like

a) Conditional Statements

b) Looping Statements

c) Jumping Statements

a) Conditional Statements:

“A conditional statement is a set of commands that executes if a specified condition is

true.”

JavaScript supports two conditional statements:

i. if...else

ii. switch

i. if...else Statement:

There are FOUR types if statements. They are

- Simple if
- if-else Statement
- Nested if
- Else-if ladder

The general formats are:

Simple if

```
if (expression)
statement;
```

if-else

```
    if (expression)
statement or block
else
statement or block
```

Nested if

```
 if (expression)
{
            if (expression)
  statement or block
            else
statement or block
}
else
statement or block
```

ELSE-IF LADDER

```
if (expression1)
 statement or block
else if (expression2)
statement or block
else if (expression3)
statement or block
...
else
statement or block
```

Ex:

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

```
 Otherwise you will get a “Good day” greeting.  </p>
</body>
</html>
```

ii switch

A switch statement allows a program to evaluate an expression and attempt tomatch the expression's value to a case label. If a match is found, the program executes the associated statement.

The general format is:

```
switch (expression)
{
 case label_1:
        statements_1
        break;
 case label_2:
        statements_2
        break;
    ….
 default:
        statements_def
}
```

Ex:

```
<!DOCTYPE html>
<html>
<body>
<script language=“javascript”>
<!--
  var d = new Date();
  theDay=d.getDay();
  switch (theDay);
  {
  case 5:
document.write(“<b>Finally Friday</b>“);
   break;
  case 6:
document.write(“<b>Super Saturday</b>“);
   break;
```

```
  case 0:
document.write("<b>Sleepy Sunday</b>");
   break;
  default:
document.write("<b>I'm really looking forward to this weekend!</b>")
  }
</script>
<p>This JavaScript will generate a different greeting based on what day it is.
              Note that Sunday=0, Monday=1, Tuesday=2, etc.</p>
</body>
</html>
```

b) Looping Statements:

A loop is a set of commands that executes repeatedly until a specified condition is
met. JavaScript supports

i. for loop

ii. do... while loop

iii. while loop

i. for loop:

The general format is

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

When a 'for loop' executes, the following occurs:

1. The initializing expression initial-expression, if any, is executed. This expression usually initializes one or more loop counters.
2. The condition expression is evaluated. If the value of condition is true, the loop statements execute. If the value of condition is false, the for loop terminates.
3. The statements execute.
4. The update expression increment Expression executes, and control returns to Step 2.

Ex:

```
<!DOCTYPE html>
<html>
<body>
<script language="javascript">
            for (i = 1; i <= 6; i++){
          document.write("<h" + i + ">This is header " + i)
              document.write("</h" + i + ">")
           }
</script>
</body>
</html>
```

Output

for...in Statement:

There is one more loop supported by JavaScript is called "for...in" loop. The JavaScript "for…in" statement loops through the properties of an object. The general format is:

```
for (variablename in object)
{
                statement or block to execute
           }
```

Where variablename specifies a different property name is assigned to variable on each iteration object specifies whose enumerable properties are iterated

Ex:

```
<html>
<body>
<script language="javascript">
          var x;
var mycars = new Array();
mycars[0] = "Saab";
mycars[1] = "Volvo";
          mycars[2] = "BMW";
    for (x in mycars){
  document.write(mycars[x] + "<br />");
    }
</script>
```

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

ii. do... while loop:

The "do...while" loop repeats until a specified condition evaluates to false. The general format is:

```
do {
        statement;
  } while (condition);
```

Ex:

```
<!Doctype html>
<html>
<body>
<script language="javascript">
          i=0;
          do {
 i++;
document.write(" "+i);
} while (i<5);
</script>
</body>
</html>
```

Output:

iii. while loop:

A while loop executes its statements until the condition true. The general format is

```
while (condition)
 {
   statement;
}
```

Eg.

```
<!Doctype html>
<html>
<body>
```

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

c) Jumping Statements:

A jump statement forces the flow of execution to jump unconditionally to another location in the script. Jump statements in JavaScript are used to terminate iteration statements. JavaScript is providing two types of jump statements. They are

i. break statement
ii. continue statement

i. break statement:

The break statement can be used to jump out of a loop. The break statement breaks the loop and continues executing the code after the loop. The breakstatement, without a label reference, can only be used inside a loop or a switch. With a labelreference, it can be used to "jump out of" any JavaScript code block. The general format is

break;

break label;

Ex: 1

```
<!DOCTYPE html>
<html>
<body>
<p>Click the button to do a loop with a break.</p>
<button onclick="myFunction()">Try it</button>
<p id="demo"></p>
<script>
        function myFunction(){
            var x="",i=0;
```

```
            for (i=0;i<10;i++){
                 if (i==3){
        break;
}
x=x + "The number is " + i + "<br>";
}
document.getElementById("demo").innerHTML=x;
          }
</script>
</body>
</html>
```

ii. continue statement:

The continue statement breaks one iteration (in the loop), if a specified conditionoccurs, and continues with the next iteration in the loop. When we use continue with a label, it applies to the looping statement identified with that label. The general format is

```
continue;
continue label;
```

Ex:1

```
<!DOCTYPE html>
<html>
<body>
<p>Click the button to do a loop which will skip the step where i=3.</p>
<button onclick="myFunction()">Try it</button>
<p id="demo"></p>
<script>
            function myFunction(){
               var x="",i=0;
               for (i=0;i<10;i++){
                  if (i==3){
                       continue;
                  }
                  x=x + "The number is " + i + "<br>";
                }
                document.getElementById("demo").innerHTML=x;
              }
```

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

Ex:2

```
<!Doctype html>
<html>
<body>
<script type="text/javascript">
            document.write("Entering the loop!<br /> ");
 outerloop:    // This is the label name
for (var i = 0; i < 3; i++){
      document.write("Outerloop: " + i + "<br />");       f
                  for (var j = 0; j < 5; j++){
if (j == 3){
continue outerloop;
}
document.write("Innerloop: " + j + "<br />");
}
}
document.write("Exiting the loop!<br /> ");
</script>
</body>
</html>
```

FUNCTIONS IN JAVASCRIPT.

"A function is a piece of code that performs a specific task."

Defining Functions:

The general format is

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

JavaScript functions can be used to create script fragments that can be used over and over again.

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

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

A function definition consists of the function keyword, followed by the name of the function.

A list of arguments to the function enclosed in parentheses and separated by commas.

The JavaScript statements that define the function, enclosed in curly braces, { }. The statements in a function can include calls to other functions defined in the current application.

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

Ex:

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

Parameter Passing: In and Out:

Primitive data types are passed by value in JavaScript. This means that a copy is made of a variable when it is passed to a function, so any manipulation of a parameterholding primitive data in the body of the function leaves the value of the originalvariable untouched.

Unlike primitive data types, composite types such as arrays and objects arepassed by reference rather than value.

For this reason, non-primitive types are often called "reference types." When afunction is passed a reference type, script in the function's body modifying theparameter will modify the original value in the calling context as well. Instead of acopy of the original data, the function receives a reference to the original data

Passing Variables as parameters:

function fiddle(arg1)

```
{
 arg1 = 10;
document.write("In function fiddle arg1 = "+arg1+"<br>");
 }
----
----
var x = 5;
document.write("Before function call x = "+x+"<br >");
fiddle(x);
document.write("After function call x ="+x+"<br>");
```

Examining the Function Call:

In JavaScript parameters are passed as arrays. Every function has two properties that can be used to find information about the parameters.

The general format is

```
 function.arguments            // This is the array of parameters that have been passed
 function.arguments.length   // This is the number of parameters that have been
 //passed into the  function
```

COMMENTS:

Comments are pieces of text which can be used by programmers as they readthrough the source code. The comment text is ignored by the JavaScript. JavaScript is providing Single Line (//) and Multi-Line comments (/* */).

Ex:

```
 // Write to a heading
 /* The code below will write
                to a heading and to a paragraph,
                and will represent the start of
                my homepage:  */
```

PARSING:

Any JavaScript in the head section is parsed by the browser. If it finds any errors in the coding (such as missing semicolons, inverted commas, mistyped built-in functionnames), then we will get an error. However the parser does not check the logic of code. Itchecks only the code could run correctly or not.

Scoping Rules:

The variables can be either local or global.

global:

Global scoping means that a variable is available to all parts of the program.

local:

Local variables are declared inside of a function. They can use only inside a function. If we want to use such a variable in other functions, it should be passed as a parameter to that function.

Ex:

```
<html>
<head>
<title>Variables, Functions and Scope</title>
</head>
<body>
<script language="javascript">
var the_var=32;
        var tmp=the_var;
        var tmp2=setLocal(17);
        document.writeln("<h1>Scope</h1>");
        document.writeln("<p>The global is " + the_var);
        document.writeln("<br>tmp is " + tmp);
        document.writeln("<br>tmp2 is " + tmp2);
        document.writeln("</p>");
        document.close();
function setLocal(num){
the_var=num;
alert("tmp is: " + tmp);
return(the_var);
}
</script>
</body>
</html>
```

ARRAYS

"An array is an ordered set of data elements which can be accessed through a single variable name."

In JavaScript, an array is a special type of object. The structure of an array is as the following:

Item One	Item Two	Item Three	Item Four	…….	Item N

Basic Array Functions:

The basic operations that are performed on arrays are creation, addition of elements, accessing individual elements and removing elements.

Creating Arrays:

There are three ways of creating an array. They are

a. Simply declare a variable and pass it some elements in array format.

The general format is

var arrayObjectName = [element0, element1, ..., element N];

Ex:

var colors = ["Red", "Green", "Blue"];

b. Create an array object using the keyword new and a set of elements to store. The general format is

var arrayObjectName = new Array(element0, element1, ..., element N);

Ex:

var colors = new Array("Red", "Green", "Blue");

c. Create an empty array object which has space for a number of elements can be created.

The general format is

var arrayObjectName = new Array(arrayLength);

Ex:

var colors=new array(4);

var colors = Array("Red", "Green", "Blue");

Adding elements into Array:

Array elements are accessed by their index. Index denotes the position of the element in the array, generally start from 0 (zero). Adding element uses square bracket.

Ex:

var days[3]=”MON”;

MON is added at position 3

data[23]=48;

48 is added to position 23

Accessing Array Members:

The elements in the array are accessed through their index.

Searching an Array:

Read each element in turn and compare it with the value that we are for.

Ex:

```
 for( var count=0;count<len;count++){
 if(data[count]==true){
  document.write(date[count]+",");
  break;
}
```

Removing Array Members:

The following steps are required to remove an element from an array:

a. Read each element in the array

b.If element is not the one we want to delete, copy it into a temporary array

c. If we want to delete the element then do nothing

d.Increment the loop counter

e. Repeat the process

LENGTH PROPERTY:

The length property sets or returns the number of elements in an array. The general format is:

array.length=number

Set the length of an array

array.length

Return the length of an array

Ex:1

```
<!Doctype html>
<html>
<head>
<title>Looping through an Array</title>
</head>
<body>
<script language="javascript">
        document.writeln("<h1>Looping through an Array</h1>");
        document.write("<p>");
        var data=["Monday","Tuesday",34,76,34,"Wednesday"];
        var len=data.length;
```

```
          for(var count=0;count<len;count++){
           document.write(data[count]+",");
          }
         document.writeln("</p>");
         document.close();
</script>
</body>
</html>
```

Ex:2

```
<!Doctype html>
<html>
<head>
<title>Removing an Array Elelment</title>
</head>
<body>
<script language="javascript">
          document.writeln("<h1>Removing an Array Element</h1>");
          var data=["Monday","Tuesday",34,76,34,"Wednesday"];
          document.write("<p>");
          var len=data.length;
          for(var count=0;count<len;count++){
             document.write(data[count]+",");
          }
           document.writeln("</p>");
           var rem=prompt("Which item shall I remove?", " ");
           var tmp=new Array(data.length - 1);
           var count2=0;;
           for(var count=0;count<len;count++){
             if(data[count]!=rem)
           {
              tmp[count2]=data[count];
              count2++;
            }
          }
           data=tmp;
         document.write("<p>");
```

```
        var len=data.length;
        for(var count=0;count<len-1;count++){
         document.write(data[count]+",");
        }
        document.writeln("<p>");
        document.close();
</script>
</body>
</html>
```

OBJECT BASED ARRAY FUNCTIONS

In JavaScript, array is treated as an object. The Array object is used to store multiple values in a single variable. The general format is

arrayname.function (parameter 1, parameter 2);

Array Functions or Array Object methods:

The following are the methods of the Array object.

concat() - Joins two or more arrays, and returns a copy of the joined arrays

join() - Joins all elements of an array into a string

pop() - Removes the last element of an array, and returns that element

push() - Adds new elements to the end of an array, and returns the new length

reverse() - Reverses the order of the elements in an array

shift() - Removes the first element of an array, and returns that element

slice() - Selects a part of an array, and returns the new array

sort() - Sorts the elements of an array

splice() - Adds/Removes elements from an array

unshift() - Adds new elements to the beginning of an array, and returns the new length

concat() Method:

The concat() method is used to join two or more arrays. This method does not change the existing arrays, but returns a new array, containing the values of the joined arrays.

The general format is

array1.concat(array2 [,array3…arrayN]);

Ex:

```
<!Doctype html>
```

```
<html>
<body>
<script language="javascript">
            var arr = new Array(3);
            arr[0] = "Jani";
  arr[1] = "Tove";
  arr[2] = "Hege";
            var arr2 = new Array(3);
            arr2[0] = "John";
            arr2[1] = "Andy";
            arr2[2] = "Wendy";
            document.write(arr.concat(arr2));
</script>
</body>
</html>
```

join() Method:

The join() method joins the elements of an array into a string, and returns the string. The elements will be separated by a specified separator. The default separator is comma (,). The general format is

arrayname.join(seperator);

Ex:

```
<!Doctype html>
<html>
<body>
<script language="javascript">
         var arr = new Array(3);
        arr[0] = "Jani";
        arr[1] = "Hege";
        arr[2] = "Stale";
        document.write (arr.join() + "<br />");
        document.write (arr.join("."));
</script>
</body>
</html>
```

pop() Method:

The pop() method removes the last element of an array, and returns that element. This method changes the length of an array. To remove the first element of an array, use the shift() method. The general format is

arrayname.pop();

Ex:

```
<!Doctype html>
<html>
<body>
<script language="javascript">
        var todolist=new Array();
        todolist[0]="maruthi 800 ";
         todolist[1]="Shift";
         todolist[2]="Shift Dzire ";
         todolist.pop();
         document.write("length of array after pop "+todolist.length+"<br>");
         document.write("<b>Array after pop :</b> "+"<br>");
        for(i=0;i<todolist.length;i++){
        document.write(todolist[i]+"<br>");
}
</script>
</body>
</html>
```

push() Method:

The push() method adds new items to the end of an array, and returns the new length. The new item(s) will be added at the end of the array. This method changes the length of the array. To add items at the beginning of an array, use the unshift() method. The general format is

arrayname.push(element1[,element2..elementN]);

Ex:

```
<!Doctype html>
<html>
<body>
<script language="javascript">
         var todolist=new Array();
         todolist[0]="Book the Waldorf for your birthday party";
```

```
        todolist[1]="Give Donald a call and invite him to your party";
        document.write("<b>Original array is : </b>"+"<br>");
        for(i=0;i<=1;i++){
         document.write(todolist[i]+"<br>");
        }
        todolist.push("Wake up from your dream");
        document.write("<b>Array after push :</b> "+"<br>");
        for(i=0;i<=2;i++){
         document.write(todolist[i]+"<br>");
        }
</script>
</body>
</html>
```

reverse() Method:

The reverse() method reverses the order of the elements in an array. The general format is

arrayname.reverse();

Ex:

```
<!Doctype html>
<html>
<body>
<script language="javascript">
         var todolist=new Array();
        todolist[0]="Book the Waldorf for yourbirthday party";
       todolist[1]="Give Donald a call and invite him to your party";
       todolist[2]="wake up from your dream";
       todolist.reverse();
       document.write("<b>Array after reverse method :</b> "+"<br>");
        for(i=0;i<todolist.length;i++){
         document.write(todolist[i]+"<br>");
        }
       </script>
</body>
</html>
```

shift() Method:

The shift() method removes the first item of an array, and returns that item. This method changes the length of an array. The general format is

arrayname.shift();

Ex:

```
<!Doctype html>
<html>
<body>
<script language="javascript">
          var a = [1, 2, 3];
          var first = a.shift();
          alert(a);
         alert(first);
</script>
</body>
</html>
```

sort() Method:

The sort() method sorts the items of an array. The sort order can be either alphabetic or numeric, and either ascending or descending. Default sort order is alphabetic and ascending. When numbers are sorted alphabetically, "40" comes before "5". To perform a numeric sort, we must pass a function as an argument when calling the sort method. The function specifies whether the numbers should be sorted ascending or descending. This method changes the original array. The general format is

arrayname.sort(sortfunction);

Ex:

```
<!DOCTYPE html>
<html>
      <body>
      <p id="demo">Click the button to sort the array.</p>
      <button onclick="myFunction()">Try it</button>
<script>
      function myFunction()
      {
```

```
        var fruits = ["Banana", "Orange", "Apple", "Mango"];
        fruits.sort();
        var x=document.getElementById("demo");
        x.innerHTML=fruits;
      }
</script>
</body>
</html>
```

unshift() Method:

The unshift() method adds new items to the beginning of an array, and returns the new length. This method changes the length of an array. To add new items at the end of an array, use the push() method. The general format is

arrayname.unshift(element1[,element2…elementN]);

The element1[,element2…elementN] is/are required and the element(s) to add to the beginning of the array.

```
<!Doctype html>
<html>
<body>
<script language="javascript">
        var a1 = [1, 2, 3];
        a1.unshift(4);
        alert(a1);
</script>
</body>
</html>
```

STRING MANIPULATION FUNCTIONS

'length' property:

The length property holds the number of characters in the string.

Ex:

```
var l1="This is the string".length;
alert(l1);
```

charAt() and charCodeAt():

The charAt() method takes one parameter: the index position of the character we want in the string. It then returns that character. charAt() treats the positions of the string characters as starting at 0, so the first character is at index 0, the second at index 1, and so on.

The charCodeAt() method is similar in use to the charAt() method, but it returns a number that represents the decimal character code for that character in the Unicode character set.

The general format is:

<StringObject>.charAt(index);

<StringObject>.charCodeAt(index);

Ex:

```
<!DOCTYPE html>
<html>
<head>
<title>Demo on String Functions</title>
</head>
<body>
<h1>Demo on charAt() and charCodeAt() Functions</h1>
<hr>
<script>
var str="Welcome to BVRICE";
document.write("<br>Character at 5th position = "+ str.charAt(5));
document.write("<br>Character Code at 5th position = "+ str.charCodeAt(5));
</script>
</body>
</html>
```

indexOf() and lastIndexOf():

The methods indexOf() and lastIndexOf() are used for searching for the occurrence of

one string inside another

Both indexOf() and lastIndexOf() take two parameters:

- The string we want to find
- The character position we want to start searching from (optional)

The return value of indexOf() and lastIndexOf() is the character position in the string at which the substring was found. If there is no match, then the value -1 is returned.

```
<!DOCTYPE html>
<html>
```

```
<head>
<title>Demo on String Functions</title>
</head>
<body>
<h1>Demo on indexOf() Function</h1>
<hr>
<script>
  var s="Hai Hello Hai Hello Hai";
          var pos=0;
var count=0;
                while(pos != -1){
    pos=s.indexOf("Hai",pos);
 if(pos != -1){
                    pos++;
                    count++;
                }
 }
 document.write("Hai occured "+count+" times");
</script>
</body>
</html>
```

substring() Method:

If we wanted to cut out part of a string and assign that cut-out part to another variable or use it in an expression, we would use the substring() method.

The method substring() takes two parameters:

- The character start position
- The character end position of the part of the string we want.
- The second parameter is optional; if we don't include it, all characters from the start position to the end of the string are included.

```
var myString = "JavaScript";
var mySubString = myString.substring(0,4);
alert(mySubString);
```

toLowerCase() and toUpperCase():

If we want to change the case of a string, for example to remove case sensitivity when comparing strings, we need the toLowerCase() and toUpperCase() methods.

```
var myString = "I Don't Care About Case"
if (myString.toLowerCase() == "i don't care about case"){
 alert("Who cares about case?");
}
```

replace():

Replace characters in a string. The general format is

<StringObject>.replace()

Ex:

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

concat():

Used to join the strings together. Will take another string or comma separated strings as an argument. The general format is

<StringObject>.concat(string1[,string2,string3…]);

Ex:

```
var you=prompt("enter your name:");
var urage=prompt(Enter ur age");
var result=you.concat(urage).concat("Thanks");
```

split():

If we want to split the string into number of pieces this is separated by delimeter. split() breaks the string into individual elements whenever it encounters a specified character as the parameter. The pieces of string are stored in array.

Ex:

```
Var words = "The String is separated by comma" . split(" ");
for(var x in words)
```

document.writeln(“ “+ x + “ “);

NUMBERS FUNCTIONS

JavaScript numbers can be written with, or without decimals:

Example

```
Varx=3.14;//Anumberwithdecimals
vary = 34;  // A number without decimals
```

Number Properties and Methods

Primitive values (like 3.14 or 2014), cannot have **properties** and methods (because they are not objects).

But with JavaScript, methods and properties are also available to primitive values, because JavaScript treats primitive values as objects when executing methods and properties.

Number Properties

Property	Description
MAX_VALUE	Returns the largest number possible in JavaScript
MIN_VALUE	Returns the smallest number possible in JavaScript
NaN	Represents a "Not-a-Number" value
POSITIVE_INFINITY	Represents infinity (returned on overflow)

Example

```
var x = Number.MAX_VALUE;
```

Numbers methods help you to work with numbers.

Global Methods

JavaScript global functions can be used on all JavaScript data types.

These are the most relevant methods, when working with numbers:

Method	Description
Number()	Returns a number, converted from its argument.
parseFloat()	Parses its argument and returns a floating point number

parseInt()	Parses its argument and returns an integer

MATHEMATICAL FUNCTIONS

Mathematical functions and values are part of a built-in JavaScript object called Math. The following table consist Mathematical built-in functions in JavaScript:

Ex:

```
<!DOCTYPE html>
<html>
<body>
<p id="demo">Click the button to display the result of 4*4*4.</p>
<button onclick="myFunction()">Try it</button>
<script>
        function myFunction(){
          alert(Math.pow(4,3));
  }
</script>
</body>
</html>
```

(OR)

```
<!DOCTYPE html>
<html>
<body>
<h1>Demo on pow() function</h1>
<hr>
<script>
          document.write("<br><br>4 power 3 = " + Math.pow(4,3));
</script>
</body>
</html>
```

Output

Math Object Methods

Method	Description

abs(x)	Returns the **absolute value** of x
acos(x)	Returns the arccosine of x, in radians
asin(x)	Returns the arcsine of x, in radians
atan(x)	Returns the arctangent of x as a numeric value between -PI/2 and PI/2 r
atan2(y,x)	Returns the arctangent of the quotient of its arguments
ceil(x)	Returns x, rounded upwards to the nearest integer
cos(x)	Returns the cosine of x (x is in radians)
exp(x)	Returns the value of E^x
floor(x)	Returns x, rounded downwards to the nearest integer
log(x)	Returns the natural logarithm (base E) of x
max(x,y,z,...,n)	Returns the number with the highest value
min(x,y,z,...,n)	Returns the number with the lowest value
pow(x,y)	Returns the value of x to the power of y
random()	Returns a random number between 0 and 1
round(x)	Rounds x to the nearest integer
sin(x)	Returns the sine of x (x is in radians)
sqrt(x)	Returns the square root of x
tan(x)	Returns the tangent of an angle

The Date Object:

The Date object is used to work with dates and times. Date objects are created with new Date(). There are four ways of instantiating a date:

var d = new Date();

var d = new Date(milliseconds);

var d = new Date(dateString);

var d = new

Date(year, month, day, hours, minutes, seconds, milliseconds);

Date Object Methods:

Method Description getDate() Returns the day of the month (from 1-31) getDay() Returns the day of the week (from 0-6) getFullYear() Returns the year (four digits)

Method	Description
getDate()	Get the day as a number (1-31)
getDay()	Get the weekday a number (0-6)
getFullYear()	Get the four digit year (yyyy)
getHours()	Get the hour (0-23)
getMilliseconds()	Get the milliseconds (0-999)
getMinutes()	Get the minutes (0-59)
getMonth()	Get the month (0-11)
getSeconds()	Get the seconds (0-59)
getTime()	Get the time (milliseconds since January 1, 1970)

Date Set Methods

Set methods are used for setting a part of a date. Here are the most common (alphabitically):

Method	Description
setDate()	Set the day as a number (1-31)
setFullYear()	Set the year (optionally month and day yyyy.mm.dd)
setHours()	Set the hour (0-23)
setMilliseconds()	Set the milliseconds (0-999)
setMinutes()	Set the minutes (0-59)
setMonth()	Set the month (0-11)

setSeconds()	Set the seconds (0-59)
setTime()	Set the time (milliseconds since January 1, 1970)

```
<!DOCTYPE html>
<html>
<body>
<p id="demo"></p>
<script>
var d = new Date();
document.getElementById("demo").innerHTML = d.getFullYear();
</script>
</body>
</html>
```

EXCEPTION HANDLING

Runtime error handling is vitally important in all programs. An exception is an error which we have designed our program to handle with.

An exception in object-based programming is an object, created dynamically at runtime, which encapsulates an error and some information about it. Exceptions can define their own exception classes to include exactly what we need to handle the problem successfully.

JavaScript try and catch:

The try statement allows you to define a block of code to be tested for errors while it is being executed. The catch statement allows you to define a block of code to be executed, ifan error occurs in the try block. The JavaScript statements try and catch come in pairs. Thegeneral format is

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

Ex: The following program consist typographical error in the try block. The catch block catches the error in the try block, and executes code to handle it.

```
<!DOCTYPE html>
<html>
```

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

The Throw Statement:

When an error occurs, when something goes wrong, the JavaScript engine will normally stop, and generate an error message. The technical term for this is: JavaScript will throw an error. The throw statement allows us to create a custom error. If we use the throw statement together with try and catch, we can control program flow and generate custom error messages. The general format is

throw exception

The exception can be a JavaScript String, a Number, a Boolean or an Object.

Ex: This example examines the value of an input variable. If the value is wrong, an exception (error) is thrown. The error is caught by the catch statement and a custom error message is displayed:

```
<!DOCTYPE html>
<html>
<body>
<h1>Exception Handling</h1>
```

```
<script>
<!--
 try{
      var x=parseInt(prompt("Please input a number between 5 and 10"," "));
      if(x=="")    throw "empty";
      if(isNaN(x)) throw "not a number";
      if(x>10)     throw "too high";
      if(x<5)      throw "too low";
document.write("<br><h2>U Entered correct value<h2>");
}
 catch(err){
     document.write("<br>ERROR: "+ err);
 }
</script>
</body>
</html>
```

REGEXP OBJECT

A regular expression is an object that describes a pattern of characters.

Regular expressions are used to perform pattern-matching and "search-and-replace" functions on text.

Syntax

var patt = new RegExp(pattern,modifiers);

or more simply:

var patt = /pattern/modifiers;

pattern specifies the pattern of an expression

modifiers specify if a search should be global, case-sensitive, etc.

MODIFIERS

Modifiers are used to perform case-insensitive and global searches:

Modifier	Description
i	Perform case-insensitive matching
g	Perform a global match (find all matches rather than

stopping after the first match)

m Perform multiline matching

```
<!DOCTYPE html>
<html>
<body>
<p>Click the button to do a case-insensitive search for "bvrice" in a string.</p>
<button onclick="myFunction()">Try it</button>
<p id="demo"></p>
<script>
function myFunction() {
var str = "Visit BVRICE";
var patt1 = /bvrice/i;
var result = str.match(patt1);
document.getElementById("demo").innerHTML = result;
}
</script>
</body>
</html>
```

```
<!DOCTYPE html>
<html>
<body>
<p>Click the button to do a global search for "is" in a string.</p>
<button onclick="myFunction()">Try it</button>
<p id="demo"></p>
<script>
function myFunction() {
   var str = "Is this all there is?";
   var patt1 = /is/g;
   var result = str.match(patt1);
   document.getElementById("demo").innerHTML = result;
}
</script>
</body>
</html>
```

```
<!DOCTYPE html>
<html>
<body>
<p>Click the button to do a multiline search for "is" at the beginning of each line in a string.</p>
<button onclick="myFunction()">Try it</button>
```

```
<p id="demo"></p>
<script>
function myFunction() {
   var str = "\nIs th\nis it?";
   var patt1 = /^is/m;
   var result = str.match(patt1);
   document.getElementById("demo").innerHTML = result;
}
</script>
</body>
</html>
```

BRACKETS

Brackets are used to find a range of characters:

Expression Description

[abc] Find any character between the brackets

[^abc] Find any character NOT between the brackets

[0-9] Find any digit between the brackets

[^0-9] Find any digit NOT between the brackets

```
<!DOCTYPE html>
<html>
<body>
<p>Click the button to do a global search for the character "h" in a string.</p>
<button onclick="myFunction()">Try it</button>
<p id="demo"></p>
<script>
function myFunction() {
   var str = "Is this all there is?";
   var patt1 = /[h]/g;
   var result = str.match(patt1);
   document.getElementById("demo").innerHTML = result;
}
</script>
</body>
</html>
```

METACHARACTERS

Metacharacters are characters with a special meaning:

Metacharacter Description

.	Find a single character, except newline or line terminator
\w	Find a word character
\W	Find a non-word character
\d	Find a digit
\D	Find a non-digit character
\s	Find a whitespace character
\S	Find a non-whitespace character
\b	Find a match at the beginning/end of a word
\B	Find a match not at the beginning/end of a word
\n	Find a new line character

Eg-1

```
<!DOCTYPE html>
<html>
<body>
<p>Click the button to do a global search for word characters in a string.</p>
<button onclick="myFunction()">Try it</button>
<p id="demo"></p>
<script>
function myFunction() {
   var str = "Give 100%!";
   var patt1 = /\w/g;
   var result = str.match(patt1);
   document.getElementById("demo").innerHTML = result;
}
</script>
</body>
</html>
```

Eg-2

```
<!DOCTYPE html>
<html>
<body>
<p>Click the button to do a global search for digits in a string.</p>
<button onclick="myFunction()">Try it</button>
<p id="demo"></p>
<script>
function myFunction() {
```

```
   var str = "Give 100%!";
   var patt1 = /\d/g;
   var result = str.match(patt1);
   document.getElementById("demo").innerHTML = result;
}
</script>
</body>
</html>
```

Quantifiers

Quantifier	Description
n+	Matches any string that contains at least one n
n*	Matches any string that contains zero or more occurrences of n
n?	Matches any string that contains zero or one occurrences of n
n{X}	Matches any string that contains a sequence of X n's
n{X,Y}	Matches any string that contains a sequence of X to Y n's
n{X,}	Matches any string that contains a sequence of at least X n's
n$	Matches any string with n at the end of it
^n	Matches any string with n at the beginning of it
?=n	Matches any string that is followed by a specific string n
?!n	Matches any string that is not followed by a specific string n

```
<!DOCTYPE html>
<html>
<body>
<p>Click the button to do a global search for at least one "o" in a string.</p>
<button onclick="myFunction()">Try it</button>
<p id="demo"></p>
<script>
function myFunction() {
   var str = "Hellooo World! Hello W3Schools!";
   var patt1 = /o+/g;
   var result = str.match(patt1);
   document.getElementById("demo").innerHTML = result;
}
</script>
</body>
</html>
```

RegExp Object Properties

Property	Description
constructor	Returns the function that created the RegExp object's prototype
global	Checks whether the "g" modifier is set
ignoreCase	Checks whether the "i" modifier is set
lastIndex	Specifies the index at which to start the next match
multiline	Checks whether the "m" modifier is set

RegExp Object Methods

Method	Description
compile()	Compiles a regular expression
exec()	Tests for a match in a string. Returns the first match
test()	Tests for a match in a string. Returns true or false
toString()	Returns the string value of the regular expression

```
<!DOCTYPE html>
<html>
<body>
<script>
var str = "The rain in Spain stays mainly in the plain";
var patt1 = /ain/g;
while (patt1.test(str)==true)
  {
  document.write("'ain' found. Index now at: "+patt1.lastIndex);
  document.write("<br>");
  }
</script>
</body>
</html>
```

OBJECTS IN JAVASCRIPT

In JavaScript almost everything is an object.

Even primitive **data types** (except null and undefined) can be treated as objects.

- Booleans can be objects (or primitive data treated as objects)
- Numbers can be objects (or primitive data treated as objects)

- Strings are also objects (or primitive data treated as objects)
- Dates are always objects
- Maths and Regular Expressions are always objects
- Arrays are always objects
- Even functions are always objects

JavaScript Objects

A JavaScript object is a complex variable, with **properties** and **methods**.

Object Properties

Properties are the values associated with an object.

Each property has a name and a value.

The syntax for accessing the property of an object is:

```
objectName.property
```

or

```
objectName[expression]
```

Object Methods

Methods are the actions that can be performed on objects.

Syntax:

```
objectName.methodName()
```

Creating a JavaScript Object

There are **different ways to create** new objects:

- Define and create a single object, using an **object literal**.
- Define and create a single object, with the **keyword new**.
- Define an **object constructor**, and then create objects of the constructed type.

Using an Object Literal

An object literal is a list of name:value pairs (like age:50) inside curly **braces** {}.

The following example creates a new JavaScript object with four properties:

Example

```
<!DOCTYPE html>
<html>
<body>
<p id="demo"></p>
```

```
<script>
var person = {
firstName : "John",
lastName  : "Doe",
age       : 50,
eyeColor  : "blue"
};
document.getElementById("demo").innerHTML =
person.firstName + " is " + person.age + " years old.";
</script>
</body>
</html>
```

Output:

John is 50 years old.

Using the JavaScript Keyword new

The following example also creates a new JavaScript object with four properties:

Example

```
<!DOCTYPE html>
<html>
<body>
<p id="demo"></p>
<script>
var person = new Object();
person.firstName = "John";
person.lastName = "Doe";
person.age = 50;
person.eyeColor = "blue";
document.getElementById("demo").innerHTML =
person.firstName + " is " + person.age + " years old.";
</script>
</body>
</html>
```

Using an Object Constructor

The standard way to create an "object type" is to use an object constructor function:

Example

```
<!DOCTYPE html>
<html>
<body>
```

```
<p id="demo"></p>
<script>
functionperson(first, last, age, eye) {
   this.firstName = first;
   this.lastName = last;
   this.age = age;
   this.eyeColor = eye;
}
var myFather = new person("John", "Doe", 50, "blue");
var myMother = new person("Sally", "Rally", 48, "green");
document.getElementById("demo").innerHTML =
"My father is " + myFather.age + ". My mother is " + myMother.age;
</script>
</body>
</html>
```

Built-in JavaScript Constructors

JavaScript has built-in constructors for native objects:

Example

```
var x1 = new Object();    // A new Object object
var x2 = new String();    // A new String object
var x3 = new Number();    // A new Number object
var x4 = new Boolean()    // A new Boolean object
var x5 = new Array();     // A new Array object
var x6 = new RegExp();    // A new RegExp object
var x7 = new Function();  // A new Function object
var x8 = new Date();      // A new Date object
```

JavaScript has object versions of the primitive data types String, Number, and Boolean.

```
<!DOCTYPE html>
<html>
<body>
<p id="demo"></p>
<script>
var x1 = {};
var x2 = "";
var x3 = 0;
var x4 = false;
var x5 = [];
var x6 = /()/;
var x7 = function(){};
document.getElementById("demo").innerHTML =
       "x1: " + typeof x1 + "<br>" +
       "x2: " + typeof x2 + "<br>" +
```

```
        "x3: " + typeof x3 + "<br>" +
        "x4: " + typeof x4 + "<br>" +
        "x5: " + typeof x5 + "<br>" +
        "x6: " + typeof x6 + "<br>" +
        "x7: " + typeof x7 + "<br>";
</script>
</body>
</html>
```

Output:
x1:object
x2:string
x3:number
x4:boolean
x5:object
x6:object
x7:function

JavaScript Objects are Mutable

Object are mutable: They are addressed by reference, not by value.If y is an object, the following statement will not create a copy of y:

```
var x = y;  // This will not create a copy of y.
```

The object x is not a **copy** of y. It **is** y. Both x and y points to the same object.Any changes to y will also change x, because x and y are the same object.

Example

```
<!DOCTYPE html>
<html>
<body>
<p id="demo"></p>
<script>
var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"}
var x = person
x.age = 10;
document.getElementById("demo").innerHTML =
person.firstName + " is " + person.age + " years old.";
</script>
</body>
```

</html>

JavaScript Properties

Properties are the values associated with a JavaScript object.
A JavaScript object is a collection of unordered properties.
Properties can usually be changed, added, and deleted, but some are read only.

Accessing JavaScript Properties

The syntax for accessing the property of an object is:

```
objectName.property          // person.age
```

or

```
objectName["property"]       // person["age"]
```

or

```
objectName[expression]       // x = "age"; person[x]
```

Example 1

```
person.firstname + " is " + person.age + " years old.";
```

JavaScript for...in Loop

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

Syntax

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

The block of code inside of the for...in loop will be executed once for each property.

Looping through the properties of an object:

Example

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

for (x in person) {
   txt += person[x];
}
<!DOCTYPE html>
<html>
<body>
```

```
<p id="demo"></p>
<script>
var txt = "";
var person = {fname:"John", lname:"Doe", age:25};
var x;
for (x in person) {
txt += person[x] + " ";
}
document.getElementById("demo").innerHTML = txt;
</script>
</body>
</html>
```

Adding New Properties

You can add **new properties** to an existing object by simply giving it a value.Assume that the person object already exists - you can then give it new properties:

Example

```
person.nationality = "English";
```

```
<!DOCTYPE html>
<html>
<body>
<p id="demo"></p>
<script>
var person = {
firstname:"John",
lastname:"Doe",
age:50,
eyecolor:"blue"
};
person.nationality = "English";
document.getElementById("demo").innerHTML =
person.firstname + " is " + person.nationality + ".";
</script>
</body>
</html>
```

Deleting Properties

The delete keyword deletes a property from an object:

Example

```
delete person.age;   // or delete person["age"];
```

```
<!DOCTYPE html>
<html>
<body>

<p id="demo"></p>
<script>
var person = {
   firstname:"John",
lastname:"Doe",
          age:50,
          eyecolor:"blue"
};
        delete person.age;
        document.getElementById("demo").innerHTML =
        person.firstname + " is " + person.age + " years old.";
</script>
</body>
</html>
```

The delete keyword deletes both the value of the property and the property itself.After deletion, the property cannot be used before it is added back again.

JavaScript Methods

You can call a method with the following syntax:

```
objectName.methodName()
```

This example uses the toUpperCase() method of the String object, to convert a text to uppercase:

```
var message = "Hello world!";
var x = message.toUpperCase();
```

The value of x, after execution of the code above will be:

```
HELLO WORLD!
```

Adding New Methods

Defining methods to an object is done inside the constructor function:

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

   function changeName(name) {
```

```
      this.lastname = name;
   }
}
```

The changeName() function assigns the value of name to the person's lastname property.

Now You Can Try:

myMother.changeName("Doe");

BUILT IN OBJECTS IN JAVASCRIPT

1. **Document Object**

The JavaScript Document object essentially contains the **HTML elements** that are contained the **<head> and <body>** sections of a web page.

The Document object gets the **initial value** of its title property from the HTML <title> tag and a number of **color and event values** from the <body> element:

```
<body>
   [alink="activated link color"]
   [background="background image file"]
   [bgcolor="background color"]
   [link="unvisited link color"]
   [onload="page load functionName"]
   [onunload="page unload functionName"]
   [text="foreground color"]
   [vlink="visited link color"]>
</body>
```

It is important to note that these are only optional initial values. Regardless of whether these are declared in the <body> element they can still be **set, or modified** from JavaScript via the Document object.

JavaScript Document Object Methods and Properties

The JavaScript Document contains a wide range of methods and properties that are of considerable use to developers. The following table lists the key methods and properties provided by the Document object:

Document Object Methods

Method	**Description**
open()	Opens the output stream to the document for writing
close()	Closes the output stream to the document to prevent further writing
write()	Appends text to the document.
writeln()	Appends text to the document followed by a newline character

Document Object Properties

Property	**Description**
alinkcolor	The color to be used when displaying activated links in the document
anchors	An array of anchor objects present in the document
applets	An array of Java Applet objects embedded into the document
bgColor	The background color of the document
fgColor	The foreground color (i.e. the color of the text in the document)
cookie	The cookie, if any, associated with the document. See Understanding JavaScript Cookies
domain	The current domain of the document
images	An Array of Image objects contained the document
lastModified	The date that the document was last modified
linkColor	The color to be used when displaying links in the document
links	An array of links contained in the document
title	The title of the document (corresponds to the HTML <title> tag)
URL	The URL of the current document

Writing text to a document

The document object open() and write() methods can be used to dynamically write text to a web page. The following example contains both some text contained in the HTML file, together with some JavaScript to write additional text into the page:

<html>

```
<head>
<title>JavaScript Document Object Example</title>
</head>
<body>
<p>
This is a line of text specified in the HTML file
</p>
<script language="javascript" type="text/javascript">
document.open()
document.write("This is a line of text written to the document using
JavaScript");
</script>
</body>
</html>
```

The above example will display the following in the browser window:

This is a line of text specified in the HTML file.This is a line of text written to the document using JavaScript

Writing Text to a document in a different Window

The writing of text to a document is not restricted to writing to the document in the current window. It is also possible to write content to a different window simply by referencing the window in the JavaScript command

```
<html>
<head>
<title>JavaScript Document Object Example</title>
<script language="javascript" type="text/javascript">
function writeContent(content, windowObj)
{
windowObj.document.write(content);
}
</script>
</head>
<body>
<script language="javascript" type="text/javascript">
newWindowObj = window.open ("", "MyWindow");
</script>
<form action="">
<input type=button value="Write to New Window"
onclick="writeContent('This is new content in the new window',
newWindowObj)"/>
</form>
</body>
</html>
```

Changing the Document Title

Even when the title of a document has been specified using the HTML <title> tag it is possible to **dynamically change** the current setting using the title property of the document object. The following example initially sets the document title to JavaScript Document Object Example and provides a button which, when pressed, changes the title to a new value:

```
<html>
<head>
<title>JavaScript Document Object Example</title>
</head>
<body>
<form action="">
<input type=button value="Press to change title"
onclick="document.title='hello'"/>
</form>
</body>
</html>
```

Changing the Document Colors

A similar approach can be used to change the colors using in the document. The following example changes the foreground and background colors when the "Change colors" button is pressed:

```
<html>
<head>
<title>JavaScript Document Object Example</title>
<script language="javascript" type="text/javascript">
function changeColors()
{
document.fgColor="red";
document.bgColor="blue";
}
</script>
</head>
<body>
<p>Sample text to show color change</p>
<form action="">
<input type=button value="Change colors" onclick="changeColors()"/>
</form>
</body>
</html>
```

Getting a List of Objects in a Document

A typical web page can contain a number of other object types, such as links, anchors, images and Java applets.

For example, the links property of the document object can be used to obtain a list of Link objects in the current document.

```
<html>
<head>
<title>JavaScript Document Object Example</title>
</head>
<body>
<a href="http://www.amazon.com">Buy more books</a>
<br>
<a href="http://www.yahoo.com">Stop making Google rich</a>
<br>
<script language="javascript" type="text/javascript">
for (i in document.links)
{
document.write( document.links[i] + "<br>" );
}
</script>
</form>
</body>
</html>
```

The above HTML file outputs the following for the two links found in the document:

http://www.amazon.com/

http://www.yahoo.com/

2. The Window Object:

The window object represents an open window in a browser. If a document contain frames (<frame> or <iframe> tags), the browser creates one window object for the HTML document, and one additional window object for each frame.

The open() method opens a new browser window. The general format is

window.open(URL,name,specs,replace)

```
<!DOCTYPE html>
<html>
<head>
<script>

var myWindow;
```

```
      function openWin() {
            myWindow = window.open("", "myWindow", "width=400,
height=200");
      }

function closeWin() {
   if (myWindow) {
      myWindow.close();
   }
}

function checkWin() {
   if (!myWindow) {
      document.getElementById("msg").innerHTML = "'myWindow' has never
been opened!";
   } else {
      if (myWindow.closed) {
         document.getElementById("msg").innerHTML = "'myWindow' has been
closed!";
      } else {
         document.getElementById("msg").innerHTML = "'myWindow' has not
been closed!";
      }
   }
}

</script>
</head>
<body>

<button onclick="openWin()">Open "myWindow"</button>
<button onclick="closeWin()">Close "myWindow"</button>
<br><br>
<button onclick="checkWin()">Has "myWindow" been closed?</button>
<br><br>
<div id="msg"></div>

</body>
</html>
```

Window Object Properties

Property	Description
closed	Returns a Boolean value indicating whether a window has been closed or not

defaultStatus Sets or returns the default text in the statusbar of a window

document Returns the Document object for the window

innerHeight Returns the inner height of a window's content area

innerWidth Returns the inner width of a window's content area

length Returns the number of frames (including iframes) in a window

location Returns the Location object for the window

name Sets or returns the name of a window

outerHeight Returns the outer height of a window, including toolbars/scrollbars

outerWidth Returns the outer width of a window, including toolbars/scrollbars

parent Returns the parent window of the current window

screen Returns the Screen object for the window

Window Object Methods

Method Description

alert() Displays an alert box with a message and an OK button

close() Closes the current window

confirm() Displays a dialog box with a message and an OK and a Cancel button

createPopup() Creates a pop-up window

open() Opens a new browser window

print() Prints the content of the current window

stop() Stops the window from loading

3. The Form Object:

The Form object represents an HTML form. For each <form> tag in an HTML document, a Form object is created. Forms are used to collect user input, and contain input elements like text fields, checkboxes, radio-buttons, submit buttons and more.

A form can also contain select menus, textarea, fieldset, legend, and label elements. Forms are used to pass data to a server.

Form Object Methods:

Method Description

reset() Resets a form

submit() Submits a form

Form Object Events:

EventThe event occurs when...

onReset The reset button is clicked

onSubmit The submit button is clicked

onClick This occurs when the user clicks on an element.

The elements of the form are held in an array. It means that any of the properties of those elements that we can set using HTML code can be accessed through our JavaScript.

Ex:

```
<html>
<head>
<script language="javascript">
          function validate(){
var method=document.forms[0].method;
var action=document.forms[0].action;
var value=document.forms[0].elements[0].value;
if(value != "Mary"){
document.forms[0].reset();
}
 else{
alert("Hi Mary");
}
}
</script>
</head>
<body>
<form method="post">
<input type="text" name="user" size="32">
<input type="submit" value="Press Me" onClick="validate()">
</form>
</body>
</html>
```

4. **Navigator Object**

The navigator object contains information about the browser.

Navigator Object Properties

Property	Description
appName	Returns the name of the browser
appVersion	Returns the version information of the browser

```
<!DOCTYPE html>
<html>
<body>
<p>Click the button to return the name of your browser.</p>
<button onclick="myFunction()">Try it</button>
<p id="demo"></p>
<script>
function myFunction() {
   var x = "Browser Name: " + navigator.appName;
   document.getElementById("demo").innerHTML = x;
}
</script>
</body>
</html>
```

```
<!DOCTYPE html>
<html>
<body>
<p>Click the button to return the version of your browser.</p>
<button onclick="myFunction()">Try it</button>
<p id="demo"></p>
<script>
function myFunction() {
   var x = "Version info: " + navigator.appVersion;
   document.getElementById("demo").innerHTML = x;
}
</script>
</body>
</html>
```

5. Screen Object

The screen object contains information about the visitor's screen.

Screen Object Properties

Property	Description
availHeight-	Returns the height of the screen (excluding the Windows Taskbar)
availWidth	Returns the width of the screen (excluding the Windows Taskbar)

height Returns the total height of the screen

width Returns the total width of the screen

```
<!DOCTYPE html>
<html>
<body>
<p>Click the button to display the available height of your screen.</p>
<button onclick="myFunction()">Try it</button>
<p id="demo"></p>
<script>
function myFunction() {
   var x = "Available Height: " + screen.availHeight + "px";
   document.getElementById("demo").innerHTML = x;
}
</script>
</body>
</html>
```

Event Handling in JavaScript.

An event is defined as "something that takes place". An event handler is JavaScript code that is designed to run each time a particular event occurs.

The Syntax of handling the events is

<Tag Attributes event="handler">

Categories of Events:

Events fall into four major categories:

1. **User interface events**
2. **Mouse events**
3. **Key events**
4. **HTML events**

User interface eventsoccurs when the objects in web page **gain and lose** focus. These events are caused by the user actions such as a **tab key press**.

Mouse events occur when the user **moves the mouse or presses** one of the mouse buttons.

Key events occur when the user **presses and/or releases** one of the keyboard keys.

HTML events Finally, there are several events specific to certain HTML elements. They are related to the browser **window** itself or to **form controls** and **other objects embedded** in a web page.

Handle User Interface Events:

User interface events deal exclusively with the **transfer of focus from one object inside** the web page to another. There are three user interface events defined in most web browsers.

Event Name	**Event Handler Name**
focus	onfocus
blur	onblur
activate	onactivate

Examples

```
<HTML>
<TITLE>Example of onFocus Event Handler</TITLE>
<HEAD></HEAD>
<BODY>
<H3>Example of onFocus Event Handler</H3>
Click your mouse in the text box:<br>
<form name="myform">
<input type="text" name="data" value="" size=10 onFocus='alert("You focused
the textbox!!")'>
</form>
</BODY>
</HTML>
```

```
<HTML>
<HEAD><TITLE>Example of onBlur Event Handler</TITLE>
</HEAD>
<BODY>
<H3> Example of onBlur Event Handler</H3>
Try inputting a value less than zero:<br>
<form name="myform">
<input type="text" name="data" value="" size=10
onBlur="valid(this.form)">
</form>

<SCRIPT LANGUAGE="JavaScript">
function valid(form){
```

```
var input=0;
input=document.myform.data.value;
          if (input<0){
           alert("Please input a value that is less than 0");
          }
}
</SCRIPT>
</BODY>
</HTML>
```

Handle Mouse Events:

JavaScript programs have the ability to **track mouse movement and button clicks** as they relate to the web page and inside the control.

Event Name	Event Handler Name
mousedown	onmousedown
mouseup	onmouseup
mouseover	onmouseover
mousemove	onmousemove
mouseout	onmouseout
click	onclick
dbclick	ondbclick

1. The content of the <h1> element is changed when a user clicks on it.

```
<!DOCTYPE html>
<html>
<body>
<h1 onclick="this.innerHTML='Ooops!'">Click on this text!</h1>
</body>
</html>
```

2. A function is called from the event handler.

```
<!DOCTYPE html>
<html>
<head>
<script>
              function changetext()
{
  document.write("Ooops!");
```

```
    }
</script>
</head>
<body>
<h1 onclick="changetext()">Click on this text!</h1>
</body>
</html>
```

3. To assign events to HTML elements we can use event attributes.

```
<!DOCTYPE html>
<html>
<body>
<p>Click the button to execute the <em>displayDate()</em>
function.</p>
<button onclick="displayDate()">Try it</button>
<script>
        function displayDate()
{
  document.write(Date());
}
</script>

</body>
</html>
```

The onmouseover and onmouseout Events: The onmouseover and onmouseout events can be used to trigger a function when the user mouses over, or out of, an HTML element.

4. A simple **onmouseover-onmouseout** example:

```
<html>
<body>
<div onmouseover="mOver(this)" onmouseout="mOut(this)"
style="background-color:#D94A38;
width:120px;
height:20px;
padding:40px;">
Mouse Over Me
</div>
<script>
            function mOver(obj)
{
             obj.innerHTML="Thank You"
          }
          function mOut(obj)
```

```
{
          obj.innerHTML="Mouse Over Me"
        }
</script>
</body>
</html>
```

Handle Key Events:

Like the user interface events, **key events fire in a predictable sequence**. There are three main key events in HTML.

Event Name	Event Handler Name
keypress	onkeypress
keydown	onkeydown
keyup	onkeyup

Example :

onkeydown event handler executes some javascript code when user press the key in the keyboard.

```
<html>
<head><title> Example of Key Events </tile>

</head>
<body>
<p>A function is triggered when the user is pressing a key in the input
field.</p>
<input type="text" onkeydown="myFunction()">
<script>
function myFunction()
{
alert("You pressed a key inside the input field");
}
</script>

</body>
</html>
```

Handle HTML Events:

An HTML event means any events that do not belong in the user interface, mouse, or key event categories. Some HTML events are **triggered directly by a user action**, while **others are fired** only as an **indirect result of a user action**

Event Name	Event Handler Name	Fires When(Event)

load	onload	Browser finishes loading document
unload	onunload	Browser about to unload document
submit	onsubmit	Form about to be submitted
reset	onreset	Form about to be reset

Example:

The onload and onunload Events: The onload and onunload events are triggered when the user enters or leaves the page.

Onload Example:

```
<HTML>
<HEAD><TITLE> Example of onload Event Handler </TITLE>
</HEAD>
<BODY onload=”myFunction()”>
<h3> Example of on load event handler </h3>
<script>
            function myFunction()
            {
               alert(“This is an example of onload event”);
            }
</script>
</BODY>
</HTML>
```

Onunload Example:

```
<HTML>
<HEAD><TITLE> Example of onunload Event Handler </TITLE>
</HEAD>
<BODY onunload=”myFunction()”>
<h3> Example of on load event handler </h3>
<script>
            function myFunction()
            {
               alert(“Thank you for visiting this web page”);
            }
</script>
</BODY>
</HTML>
```

The submit and reset Events :

```
<HTML>
<TITLE> Example of onSubmit Event Handler </TITLE>
<HEAD>
</HEAD>
<BODY>
<H3>Example of onSubmit Event Handler </H3>
Type your name and press the button<br>

<form name="myform" onSubmit="alert('Thank you ' + myform.data.value
+'!')">
<input type="text" name="data">
```

```
<input type="submit" name ="submit" value="Submit this form">
</form>
</BODY>


<HTML>
<TITLE>Example of onReset Event Handler</TITLE>
<HEAD></HEAD>
<BODY>
<H3> Example of onReset Event Handler </H3>
Please type something in the text box and press the reset button:<br>

<form name="myform" onReset="alert('This will reset the form!')">
<input type="text" name="data" value="" size="20">
<input type="reset" Value="Reset Form" name="myreset">
</form>
</BODY>
</HTML>
```

The onchange Event: The onchange event are often used in combination with validation of input fields. Below is an example of how to use the onchange. The upperCase() function will be called when a user changes the content of an input field.

```
<html>
<head><title> Change Event Example </title>


</head>
<body>
Enter your name: <input type="text" id="fname"
onchange="myFunction()">
<p>When you leave the input field, a function is triggered which
transforms the input text to upper case.</p>
<script>
            function myFunction(){
var x=document.getElementById("fname");
x.value=x.value.toUpperCase();
 }
</script>
</body>
</html>
```

Examples on Event Handlers:

4. The HTML DOM allows us to assign events to HTML elements using JavaScript.

```
<html>
<head>
</head>
<body>
<p>Click the button to execute the <em>displayDate()</em> function.</p>
<button id="myBtn">Try it</button>
<script>

document.getElementById("myBtn").onclick=function(){displayDate()};
  function displayDate(){
document.getElementById("demo").innerHTML=Date();
  }
</script>
<p id="demo"></p>
</body>
</html>
```

5. The onload and onunload Events: The onload and onunload events are triggered when the user enters or leaves the page. The onload event can be used to check the visitor's browser type and browser version, and load the

proper version of the web page based on the information. The onload and onunload events can be used to deal with cookies.

```
<html>
<body onload="checkCookies()">
<script>
        function checkCookies()
{
if (navigator.cookieEnabled==true)
{
alert("Cookies are enabled")
}
            else
{
alert("Cookies are not enabled")
 }
   }
</script>

<p>An alert box should tell you if your browser has enabled cookies or
not.</p>
</body>
</html>
```

The onchange Event: The onchange event are often used in combination with validation of input fields. Below is an example of how to use the onchange. The upperCase() function will be called when a user changes the content of an input field.

```
<!DOCTYPE html>

<html>

<head>

<script>

            function myFunction(){

var x=document.getElementById("fname");

x.value=x.value.toUpperCase();

     }

</script>

  /head>

<body>
```

```
Enter your name: <input type="text" id="fname" onchange="myFunction()">
<p>When you leave the input field, a function is triggered which transforms
the input text to upper case.</p>
</body>
</html>
```

HTML Events

An HTML event can be something the browser does, or something a user does.

Here are some examples of HTML events:

An HTML web page has finished loading

An HTML input field was changed

An HTML button was clicked

Often, when events happen, you may want to do something.

JavaScript lets you execute code when events are detected.

HTML allows event handler attributes, with JavaScript code, to be added to HTML elements.

With single quotes:

<some-HTML-element some-event='some JavaScript'>

With double quotes:

<some-HTML-element some-event="some JavaScript">

In the following example, an onclick attribute (with code), is added to a button element:

```
<!DOCTYPE html>
<html>
<body>
<button onclick="getElementById('demo').innerHTML=Date()">The time
is?</button>
<p id="demo"></p>
</body>
</html>
```

In the example above, the JavaScript code changes the content of the element with id="demo".

```
<!DOCTYPE html>
<html>
<body>
button onclick="this.innerHTML=Date()">The time is?</button>
</body>
</html>
```

Common HTML Events

Here is a list of some common HTML events:

Event	Description
onchange	An HTML element has been changed
onclick	The user clicks an HTML element
onmouseover	The user moves the mouse over an HTML element
onmouseout	The user moves the mouse away from an HTML element
onkeydown	The user pushes a keyboard key
onload	The browser has finished loading the page

```
<!DOCTYPE html>
<html>
<body>
<p>Click the button to trigger a function.</p>

<button onclick="myFunction()">Click me</button>
<p id="demo"></p>
<script>
function myFunction() {
   document.getElementById("demo").innerHTML = "Hello World";
}
```

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

```
<!DOCTYPE html>
<html>
<head>
<script>
function myFunction()
{
alert("Page is loaded");
}
</script>
</head>
<body onload="myFunction()">
<h1>Hello World!</h1>
</body>
</html>
```

What can JavaScript Do?

Event handlers can be used to handle, and verify, user input, user actions, and browser actions:

Things that should be done every time a page loads

Things that should be done when the page is closed

Action that should be performed when a user clicks a button

Content that should be verified when a user input data

Many different methods can be used to let JavaScript work with events:

HTML event attributes can execute JavaScript code directly

HTML event attributes can call JavaScript functions

You can assign your own event handler functions to HTML elements

You can prevent events from being sent or being handled

UNIT – 5

XML

What is XML?

- XML stands for EXtensible Markup Language
- XML is a markup language much like HTML
- XML was designed to describe data, not to display data

- XML tags are not predefined. You must define your own tags
- XML is designed to be self-descriptive
- XML is a W3C Recommendation

```
<?xml version="1.0" >
<note>
  <to> Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

How Can XML be Used?

1. XML Separates Data from HTML

If you need to **display dynamic data** in your HTML document, it will take a lot of work to edit the HTML each time the data changes.

With XML, **data can be stored in separate XML files**. This way you can concentrate on using **HTML/CSS for display and layout**, and be sure that changes in the underlying data will not require any changes to the HTML.

With a few lines of **JavaScript code**, you can read an **external XML file and update the data content** of your web page

2. XML Simplifies Data Sharing

In the real world, computer systems and **databases contain data in incompatible formats**.
XML data is stored in **plain text format**. This provides a software- and hardware-independent way of storing data.
This makes it much **easier to create data** that can be **shared by different applications**.

3. XML Simplifies Data Transport

One of the most time-consuming **challenges for developers** is to **exchange data between incompatible systems** over the Internet.
Exchanging data as XML greatly reduces this complexity, since the data can be **read by different incompatible applications**.

4. XML Simplifies Platform Changes

XML data is **stored in text format**. This makes it **easier to expand or upgrade to new operating systems**, new applications, or new browsers, without losing data.

5. XML Makes Your Data More Available

Different applicationscan access the data, **not only in HTML pages**, but also from **XML data sources**.

6. Internet Languages Written in XML

Several Internet languages are written in XML. Here are some examples:

- XHTML
- XML Schema
- SVG
- WSDL
- RSS

XML Syntax Rules

All XML Elements Must Have a Closing Tag

In HTML, some elements do not have to have a closing tag:
This is a paragraph.

In XML, it is illegal to omit the closing tag. All elements **must** have a closing tag:
This is a paragraph.

XML Tags are Case Sensitive

XML tags are case sensitive. The tag <Letter> is different from the tag <letter>.
Opening and closing tags must be written with the same case:
<Message>This is incorrect</message>
<message>This is correct</message>

XML Elements Must be Properly Nested

In HTML, you might see improperly nested elements:
This text is bold and italic

In XML, all elements **must** be properly nested within each other:

This text is bold and italic

XML Attribute Values Must be Quoted

XML elements can have attributes in name/value pairs just like in HTML.
In XML, the attribute values must always be quoted.

```
<note date="12/11/2007">
  <to>Tove</to>
  <from>Jani</from>
</note>
```

Comments in XML

The syntax for writing comments in XML is similar to that of HTML.
<!-- This is a comment -->

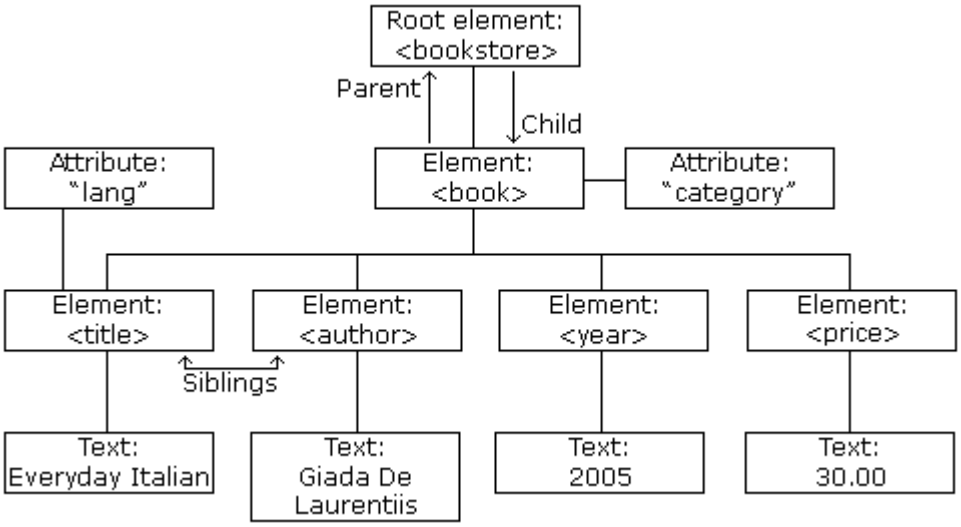
XML Documents Form a Tree Structure

XML documents must contain a **root element**. This element is "the parent" of all other elements.

The elements in an XML document form a document tree. The tree starts at the root and branches to the lowest level of the tree.
All elements can have sub elements (child elements):

```
<root>
  <child>
    <subchild>.....</subchild>
  </child>
</root>
```

The terms parent, child, and sibling are used to describe the relationships between elements. Parent elements have children. Children on the same level are called siblings (brothers or sisters).
All elements can have text content and attributes (just like in HTML).



The image above represents one book in the XML below:

```
<bookstore>
  <book category="COOKING">
    <title lang="en">Everyday Italian</title>
    <author>Giada De Laurentiis</author>
    <year>2005</year>
    <price>30.00</price>
  </book>
  <book category="CHILDREN">
    <title lang="en">Harry Potter</title>
    <author>J K. Rowling</author>
    <year>2005</year>
    <price>29.99</price>
  </book>
  <book category="WEB">
    <title lang="en">Learning XML</title>
    <author>Erik T. Ray</author>
    <year>2003</year>
    <price>39.95</price>
  </book>
</bookstore>
```

XML Document Types

An XML document with correct syntax is called "Well Formed".

Well Formed XML Documents

An XML document with correct syntax is "Well Formed".

- XML documents must have a root element
- XML elements must have a closing tag
- XML tags are case sensitive
- XML elements must be properly nested
- XML attribute values must be quoted

```
<?xml version="1.0" encoding="UTF-8"?>
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>
```

An XML Validator

To help you check the syntax of your XML files, we have created an XML validator to syntax-check our XML.

Valid XML Documents

A valid XML document is not the same as a well formed XML document.
The first rule, for a valid XML document, is that it must be well formed.
The second rule is that a valid XML document must conform to a document type.
Rules that defines legal elements and attributes for XML documents are often called document definitions

Document Definitions

There are different types of document definitions that can be used with XML:

- The original Document Type Definition (DTD)
- The newer, and XML based, XML Schema

XML DTD

An XML document with correct syntax is called "Well Formed".

An XML document validated against a DTD is "Well Formed" and "Valid".

Valid XML Documents

A "Valid" XML document is a "Well Formed" XML document, which also conforms to the rules of a DTD:

Internal DTD Declaration

If the DTD is declared inside the XML file, it should be wrapped in a DOCTYPE definition with the following syntax:

```
<!DOCTYPE root-element [element-declarations]>
```

Example XML document with an internal DTD:

```
<?xml version="1.0"?>
<!DOCTYPE note [
<!ELEMENT note (to,from,heading,body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
]>
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend</body>
</note>
```

External DTD Declaration

If the DTD is declared in an external file, it should be wrapped in a DOCTYPE definition with the following syntax:

```
<!DOCTYPE root-element SYSTEM "filename">
```

This is the same XML document as above, but with an external DTD (Open it, and select view source):

```
<?xml version="1.0"?>
<!DOCTYPE note SYSTEM "note.dtd">
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

And this is the file "note.dtd" which contains the DTD:

```
<!ELEMENT note (to,from,heading,body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
```

```
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
```

The DTD above is interpreted like this:

- !DOCTYPE note defines that the root element of the document is note
- !ELEMENT note defines that the note element contains four elements: "to, from, heading, body"
- !ELEMENT to defines the to element to be of type "#PCDATA"
- !ELEMENT from defines the from element to be of type "#PCDATA"
- !ELEMENT heading defines the heading element to be of type "#PCDATA"
- !ELEMENT body defines the body element to be of type "#PCDATA"

Data Types in DTD

PCDATA - Parsed Character Data

XML parsers normally parse all the text in an XML document.

CDATA - (Unparsed) Character Data

The term CDATA is used about text data that should not be parsed by the XML parser.

The XMLHttpRequest Object

The XMLHttpRequest object is used to exchange data with a server behind the scenes.

The XMLHttpRequest object is **a developer's dream**, because you can:

- Update a web page without reloading the page
- Request data from a server after the page has loaded
- Receive data from a server after the page has loaded
- Send data to a server in the background

Create an XMLHttpRequest Object

All modern browsers (IE7+, Firefox, Chrome, Safari, and Opera) have a built-in XMLHttpRequest object.

Syntax for creating an XMLHttpRequest object:

xmlhttp=new XMLHttpRequest();

Old versions of Internet Explorer (IE5 and IE6) use an ActiveX Object:

xmlhttp=new ActiveXObject("Microsoft.XMLHTTP"**);**

To extract the text from the element in the XML file use the following syntax

getElementsByTagName("tag")[0].childNodes[0].nodeValue

To extract the text "Tove" from the <to> element in the XML file above ("note.xml"), the syntax is:

```
getElementsByTagName("to")[0].childNodes[0].nodeValue
```

Notice that even if the XML file contains only ONE <to> element you still have to specify the array index [0]. This is because the getElementsByTagName() method returns an array.

XML DOM

A DOM (Document Object Model) defines a standard way for accessing and manipulating documents.

The XML DOM views an XML document as a tree-structure.

All elements can be accessed through the DOM tree. Their content (text and attributes) can be modified or deleted, and new elements can be created. The elements, their text, and their attributes are all known as nodes.

The following example parses an XML document (note.xml) into an XML DOM object and then extracts some info from it with a JavaScript:

```
<!DOCTYPE html>
<html>
<body>
<h1> Internal Note</h1>
<div>
<b>To:</b><span id="to"></span><br>
<b>From:</b><span id="from"></span><br>
<b>Message:</b><span id="message"></span>
</div>
<script>
if (window.XMLHttpRequest)
  {// code for IE7+, Firefox, Chrome, Opera, Safari
xmlhttp=new XMLHttpRequest();
  }
else
  {// code for IE6, IE5
xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
  }
```

```
xmlhttp.open("GET","note.xml",false);
xmlhttp.send();
xmlDoc=xmlhttp.responseXML;

document.getElementById("to").innerHTML=
xmlDoc.getElementsByTagName("to")[0].childNodes[0].nodeValue;
document.getElementById("from").innerHTML=
xmlDoc.getElementsByTagName("from")[0].childNodes[0].nodeValue;
document.getElementById("message").innerHTML=
xmlDoc.getElementsByTagName("body")[0].childNodes[0].nodeValue;
</script>
</body>
</html>
```

note.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>
```

Output:

Internal Note

To: Tove
From: Jani
Message: Don't forget me this weekend!

XML Schema

An XML Schema describes the structure of an XML document, just like a DTD.

An XML document with correct syntax is called "Well Formed".

An XML document validated against an XML Schema is both "Well Formed" and "Valid".

XML Schemas define the elements of your XML files.

A simple element is an XML element that contains only text. It cannot contain any other elements or attributes.

Defining a Simple Element

The syntax for defining a simple element is:

```
<xs:element name="xxx" type="yyy"/>
```

where xxx is the name of the element and yyy is the data type of the element.

XML Schema has a lot of built-in data types. The most common types are:

- xs:string
- xs:decimal
- xs:integer
- xs:boolean
- xs:date
- xs:time

Example

Here are some XML elements:

```
<lastname>Refsnes</lastname>
<age>36</age>
<dateborn>1970-03-27</dateborn>
```

And here are the corresponding simple element definitions:

```
<xs:element name="lastname" type="xs:string"/>
<xs:element name="age" type="xs:integer"/>
<xs:element name="dateborn" type="xs:date"/>
```

Default and Fixed Values for Simple Elements

Simple elements may have a default value OR a fixed value specified.

A default value is automatically assigned to the element when no other value is specified.

In the following example the default value is "red":

```
<xs:element name="color" type="xs:string" default="red"/>
```

A fixed value is also automatically assigned to the element, and you cannot specify another value.

In the following example the fixed value is "red":

```
<xs:element name="color" type="xs:string" fixed="red"/>
```

XML Schema

XML Schema is an XML-based alternative to DTD:

<xs:element name="note">

<xs:complexType>
  <xs:sequence>
    <xs:element name="to" type="xs:string"/>
    <xs:element name="from" type="xs:string"/>
    <xs:element name="heading" type="xs:string"/>
    <xs:element name="body" type="xs:string"/>
  </xs:sequence>
</xs:complexType>

</xs:element>

note.xml

<?xml version="1.0" encoding="UTF-8"?>
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>

The Schema above is interpreted like this:

- <xs:element name="note"> defines the element called "note"
- <xs:complexType> the "note" element is a complex type
- <xs:sequence> the complex type is a sequence of elements
- <xs:element name="to" type="xs:string"> the element "to" is of type string (text)
- <xs:element name="from" type="xs:string"> the element "from" is of type string
- <xs:element name="heading" type="xs:string"> the element "heading" is of type string
- <xs:element name="body" type="xs:string"> the element "body" is of type string