Infosys®
POWERED BY INTELLECT
DRIVEN BY VALUES

HTML, CSS, JavaScript

Unit 1 - HTML – Hyper Text Markup Language

Education and Research

Course Objective

- To introduce the participants to
  - Different HTML tags
  - Cascading Style Sheets
  - Client side Scripting with JavaScript
  - Document Object Model

Session Plan

Day 1

HTML

Frames, Images and Forms

CSS

Day 2

Introduction to JavaScript

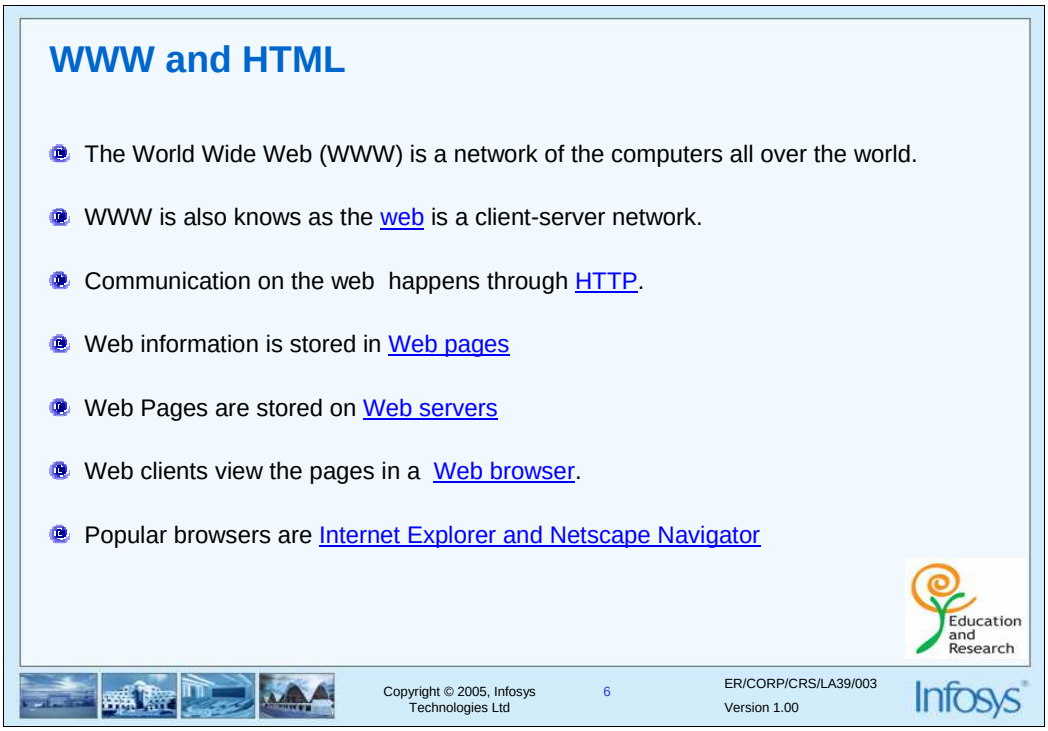
Document Object Model

References

- Ivan Byross, " *Web enabled Commercial Application development HTML,DHTML,JavaScript, Perl ,CGI* ", BPB publications
- David Flanagan, *JavaScript The Definite Guide* , Shroff Publishers.
- Thomas Powell ,*The Complete Reference – HTML,* TATA McGRAW-HILL
- www.w3schools.com
- www.htmlgoodies.com

Unit Plan

- What is HTML?
- Where does it fit in?
- HTML tags and attributes
- Text formatting tags
- Linking pages
- Working with lists and tables.



Overview of www and HTML

- Internet is a communication network linking computers world wide.
- The World Wide Web is a way of accessing information over the medium of Internet. It is an information sharing model that is built on top of the Internet.
- WWW is also known as the web.
- WWW is not a single entity it is a client-server network that includes web servers that are designed to deliver files to the client.
- The web uses the http protocol to transmit data.
- A web site is a collection of files, linked together and saved on the web server. These files are known as web pages.
- A Browser is an application that allows a user to find, view, hear, and interact with World Wide Web
- Client utilizes browsers such as Internet Explorer, Netscape Navigator, Mozilla, Opera etc to access web documents called web pages.
- First page which is displayed in the web browser when it connects to the web site is called the home page.(The file name of the home page is normally index.html or default.html)
- A Web server is the computer program (housed in a computer) that serves requested HTML pages or files.
- Hypertext Transfer Protocol (HTTP) - The standard language that computers connected to the World Wide Web use to communicate with each other.

Web Pages are coded in HTML. HTML documents are stored on the web server and are downloaded as part of a response to a request from the client.

HTML stands for Hyper Text Markup Language and is used to create hypertext documents that are platform independent. An HTML document is a text file that contains the elements a browser uses to display text, multimedia objects, and hyperlinks. A hyper link connects one document to another document. Hyper links in text are easy to spot because they are usually underlined and a different color from the rest of the text.

A markup language is a language that has codes for indicating layout and styling (such as boldface, italics, paragraphs, placement of graphics, etc.) within a text file. Markup language is entirely different from a programming language.

HTML is derived from Standard Generalized Markup Language (SGML). SGML is a language used to define other languages.

Tim Berners-Lee invented the World Wide Web, HTML, HTTP and Universal Resource Locators (URLs). He is currently the director of the World Wide Web Consortium(W3C) the group that sets technical standards for the Web.

HTML tags are not case sensitive, <body> means the same as <BODY>

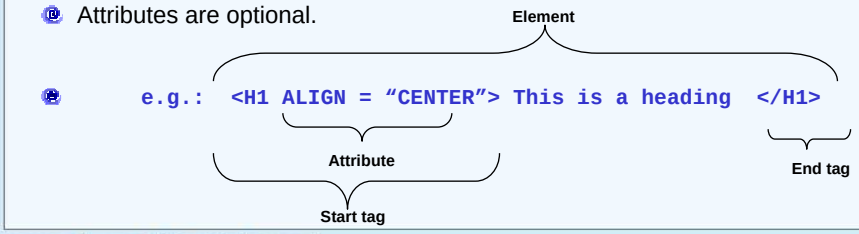
HTML using ASCII (plain text) format , hence platform independent

HTML is interpreted by the browser, not like other programming languages.

HTML form elements are used to create a GUI.

HTML tags and attributes

- The HTML instructions are called tags, and look like
  - `<TAG> ….. Text here…….. </TAG>`
- Container tags :Tags that have starting as well as ending part.
  - `e.g.: <TITLE>Title of the Web Page </TITLE>`
- Empty tags : Tags that do not have the closing part.
  - `e.g. <BR> , <HR>`
- (HTML instructions + text to which the instructions apply)= HTML elements
- An attribute is an additional feature you can use to configure the element
- Attributes are optional.



An HTML file contains markup tags that tell the browser how to display the page. HTML tags are used to mark the elements of a document for your browser. These elements could be headings, tables, lists etc. and can contain plain text, other elements, or both. The tags consist of a left angle bracket (<), a tag name, and a right angle bracket (>). Tags could be paired (e.g., <H1> and </H1>) to start and end the tag instruction.

Some tags don't need to have the closing part - these are called empty tags e.g.
 whereas some tags necessarily need to have both the starting and ending part - these are called container tags e.g. <H3> and </H3>. HTML tags are case insensitive.

Many HTML tags also have attributes assigned to them. An attribute is an additional feature you can use to configure the element. For example, <H1> tag can also have an attribute "ALIGN" assigned to it, so that <H1 ALIGN = "CENTER"> will generate a center aligned heading. Attributes are optional, can be used depending on how you want to modify the tag. Attributes always come in name/value pairs like this: name="value" and are added to the start tag of an HTML element.

Structure of HTML Document

```
<HTML>
 <HEAD> <!-- Head Section -->
       <TITLE>Title of the Web Page </TITLE>
 </HEAD>
<BODY> <!-- Body Section -->
       <!-- Contents on Web Page  -->
       <H1> Contents </H1>
</BODY>
</HTML>
```

first.html

- An HTML file can be created by using a simple text editor viz notepad, textpad, editplus.
- HTML file must have an extension htm or html.

Every HTML program or document has a rigid structure. The entire web page is enclosed within the <HTML> </HTML> tag. Within this tag, two distinct sections, head and body are created. These sections are as described below.

Document Head

The text between the <HEAD> tag and the </HEAD> tag is header information. Header information is not displayed in the browser window, it is necessary for the internal working of the document. An exception to this is the <TITLE> </TITLE> tag, which displays the document title in the browser's title bar.

Document Body

The <BODY></BODY> tag encloses the body of the HTML document.

Comments

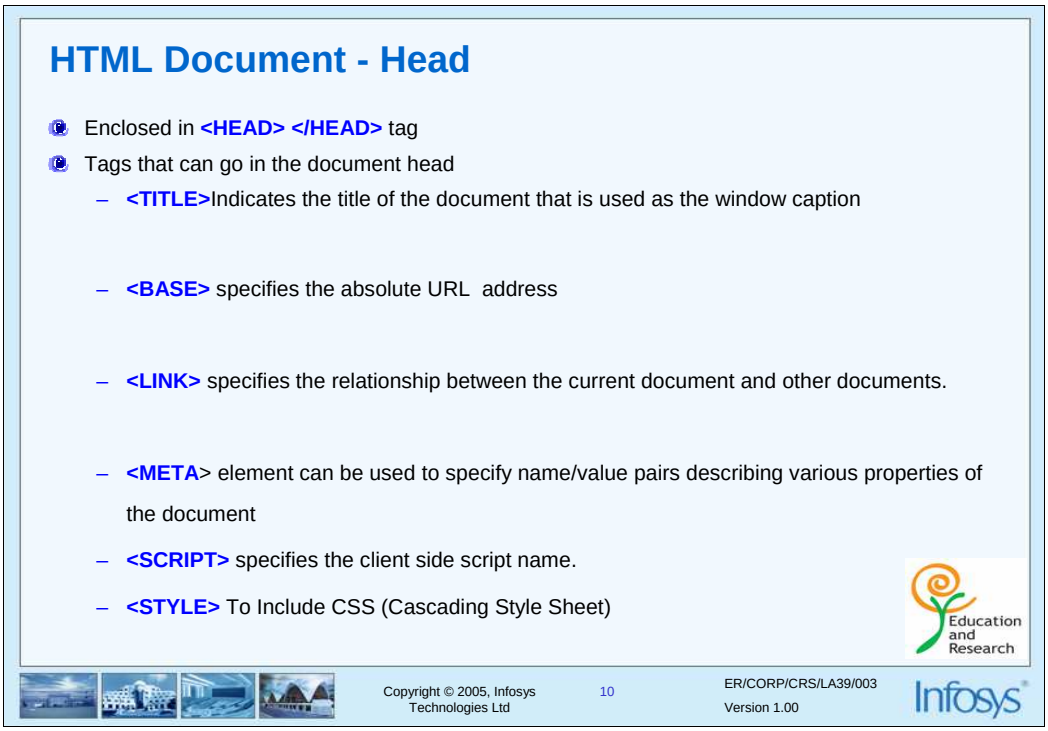
The <!-- --> tag is used to insert a comment in the HTML source code. A comment will be ignored by the browser. Comments can be used to explain your code or earmark the areas that need improvement, which can help you when you edit the source code at a later date.

e.g.: <!-- This is a comment -->

Note that you need an exclamation point after the opening bracket, but not before the closing bracket.

You can easily edit HTML files using a WYSIWYG (what you see is what you get) editor like FrontPage, Visual Interdev etc. However,

these tools do not generate very optimized HTML code because it may contain Unnecessary tags.



<BASE> element specifies an absolute URL address that is used to provide server and directory information for partially specified URL address called relative address.

e.g: <BASE HREF ="http://www.inf.com/file.html“>

Specifies the base URL of the document. This is used when dereferencing relative URLs in the page.

<META> element uses name value pairs to provide meta information about the document. It often provides descriptive information that is targeted by search engines.

Eg: .

1. To have your page automatically reloaded every X seconds
   <META HTTP-EQUIV="REFRESH" CONTENT=X >
2. To have a different page automatically loaded after X seconds
   <META HTTP-EQUIV="REFRESH" CONTENT="X; URL= http://address/file.html">
3. To specify an expiration date for the page so that it will be reloaded after a certain date.
   <META HTTP-EQUIV="Expires" CONTENT="Mon, 23 Sep 2001 01:21:00 GMT">
4. To specify keywords for certain search services to use.
   <META HTTP-EQUIV="Keywords" CONTENT="keyword1, keyword2, ...">
5. To specify a description of your page for certain search services to use
   <META HTTP-EQUIV="Description" CONTENT="Describe your site here....“

<LINK> element is used in linking external CSS which will be discussed in later chapters.

HTML Document – Body

- Enclosed in **<BODY> </BODY>** tag.
- Some important attributes of the BODY tag
  - **BGCOLOR = "color" / "#rrggbb"**
  - **BGPROPERTIES=FIXED**
  - **BACKGROUND = "url of the image"**
  - **TEXT = "color" / "#rrggbb"**
  - **LEFTMARGIN = n**
  - **LINK = "color" / "#rrggbb"**
  - **ALINK = "color" / "#rrggbb"**
  - **VLINK = "color" / "#rrggbb"**
  - **TOPMARGIN= n**
  - Colors are defined using a hexadecimal notation for the combination of Red, Green, and Blue color values (RGB).

body.html

<BODY > tag attributes

BACKGROUND=x.jpg	- Specifies an image to be tiled as background.
BGCOLOR=color	- Specifies the background color
BGPROPERTIES=FIXED	-Fixes the background image so that it doesn't scroll.
LEFTMARGIN=n	-Specifies the left margin for the entire page
TEXT=color	-Specifies the color of text in the page
TOPMARGIN=n	-Specifies the top margin for the entire page.
LINK	-Specifies the link color.
ALINK	-Specifies the active link color.
VLINK	-Specifies the active link color.

RGB Color Components.

Color attribute values give a color definition. Colors are defined using a hexadecimal notation for the combination of Red, Green, and Blue color values (RGB). The lowest value that can be given to one light source is 0 (hex #00). The highest value is 255 (hex #FF). A collection of color names like blue, green, cyan etc. are also supported by most of the browsers.

```
<html><head>
    <title>Background color</title>
</head><body bgcolor="cyan">
     <h1> Contents </h1>
</body></html>
```

Formatting the web page

- **<FONT>** tag
  - Allows you to specify the font face and font size.
  - Some common attributes are
    - FACE specifies the font type.

      Defaults fonts like “Arial”, “Times New Roman”, and “Courier” are available in all Systems.
    - SIZE specifies the font size. Value can range from 1 to 7. The default is 3.

      SIZE can be set as a relative value using + or – .
    - COLOR- The color of a font can be specified using a hexadecimal number value six characters long.
  - E.g.: `<FONT FACE=“Helvetica, Arial” SIZE=“7” COLOR=“#FF0000”>`

    `The Written Word</FONT>`

Education and Research

Infosys

Formatting tags are used inside the <BODY> of the document to format your text

This is the largest category of tags and you will be using these tags most frequently.

<FONT> tag is a physical tag that describes the display attributes of the text inside.

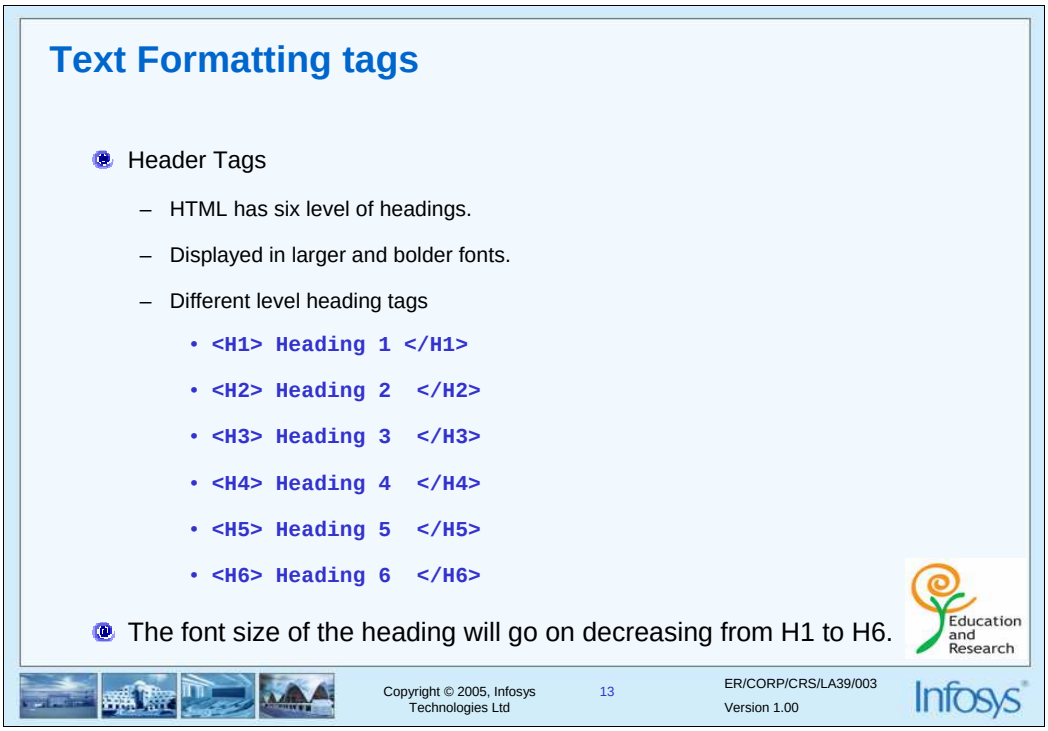
E.g:

```
<FONT FACE="Comic Sans MS" SIZE=“2“ COLOR="Red">
This is comical and red and small</FONT><BR>
    <!-- <br> tag used for line breaks                    -->
<FONT FACE="Comic Sans" SIZE="+2" COLOR="Red">
    This is red and big. Is it comical?</FONT><BR>
```

If the font face is not specified correctly or not available in the system , the text will be displayed in the default font.

In HTML the font size is limited to 1 to 7. In future you can use CSS (Cascading Style Sheets) for changing the properties.

ie you can have a font with size 10 cm .



Headings

HTML has six levels of headings, numbered 1 through 6, with 1 being the largest.

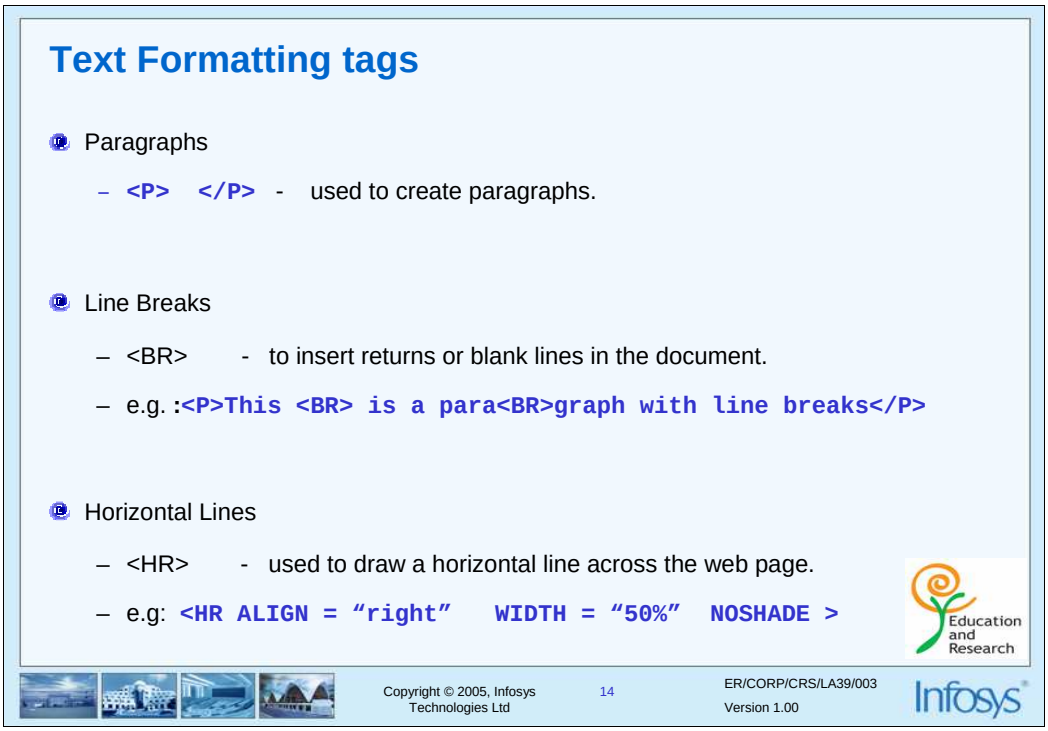
Headings are typically displayed in larger and/or bolder fonts than normal body text.

The first heading in each document should be tagged <H1>. The syntax of the heading element is: <Hy>Text of heading </Hy> where y is a number between 1 and 6.

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

HTML elements permitted within the BODY are classified as either block-level elements (or) inline elements.

Block-level elements typically contain inline elements and other block-level elements. Block-level elements usually begin on a new line.



Paragraphs

Unlike documents in most word processors, HTML document ignore multiple spaces, tabs and carriage returns. Word wrapping can happen any time in your source file and multiple spaces are collated into a single space. To preserve some text formatting we use the <p> tag which created paragraphs. Normally the browser places a blank line before the paragraph

Some elements do not have closing tags (known as empty tag).

 is an empty tag used when you want to end a line, but don't want to start a new paragraph. The
 tag forces a line break wherever you place it.

e.g. : <P>This
 is a para
graph with line breaks</P>

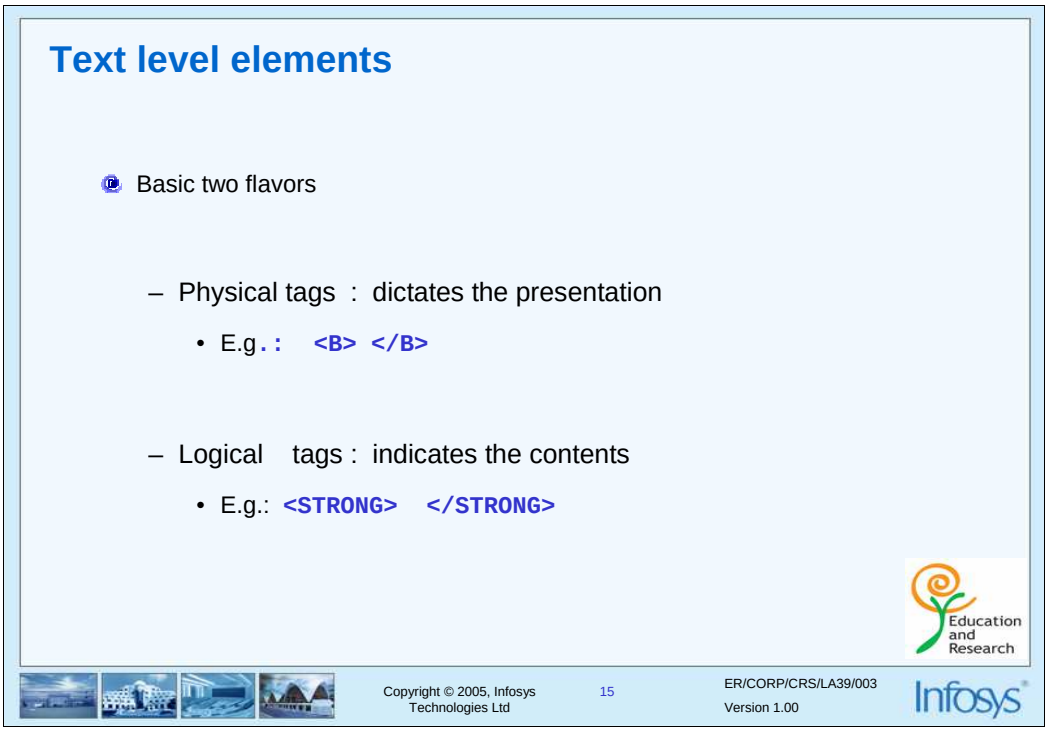
<P> tag has a ALIGN attribute which aligns the paragraph to LEFT, RIGHT, CENTER or JUSTIFY (justified)

Horizontal rules

The <HR> tag draws a horizontal line across the page. It is useful to separate different sections of a single page.

Some attributes are

SIZE	-	Sets the line thickness
WIDTH	-	Sets the width of the of the line
ALIGN	-	sets the alignment to LEFT,RIGHT or CENTER.
NOSHADE	-	renders the bar without surrounding shadow.



Logical tags

A logical tag gives some kind of indication about the content but does not dictate the presentation.

For e.g. Level 1 heading i.e. <H1> - this indicates the highest level heading in the page. However it does not specify whether this should be Arial or Times New Roman, 20 pt or 24 pt.

Advantage of logical tags is that they help enforce consistency in your documents.

For example, consider the <STRONG> tag. Most browsers render it in bold text.

Logical styles offer the flexibility of changing the display attributes while making minimum changes to code

Physical tags

Physical tags are used to specify how text should be rendered.

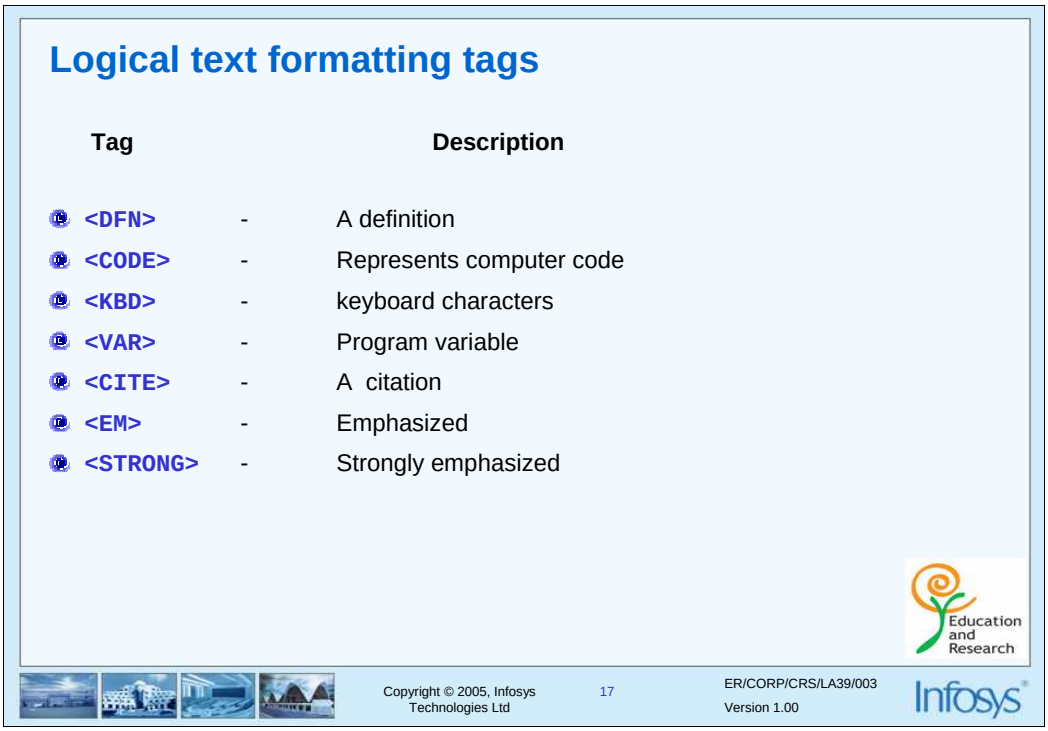
Physical text formatting tags

Tag		Description
<B>….</B>	-	Bold
<I>……</I>	-	Italic
<U>….</U>	-	Underline
<STRIKE>…</STRIKE>	-	Strikethrough
<TT>….</TT>	-	Typewriter (monospaced)
<CENTER></CENTER>	-	Centers the text on the screen.
_{….}	-	Subscript
^{….}	-	Superscript
<BIG>….</BIG>	-	Bigger font (one font size bigger)
<SMALL>….</SMALL>	-	Small font (one font size smaller)

phy.html

The following example shows the basic use of the physical text formatting tags:

```
<HTML>
<HEAD>
<TITLE> Physical tags </TITLE>
</HEAD>
<BODY BGCOLOR="#FFFFFF“ TEXT =“red”>
<H1 ALIGN ="center"> Physical tags </H1>
<HR>
This is <B> bold </B>   <BR>
This is <I> Italic </I> <BR>
This is <U> underline</U>  <BR>
This is <TT> Monospaced</TT> <BR>
This is <STRIKE> Strike-through</STRIKE>  <BR>
This is <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>
```



Logical Formatting Tags

The formatting of this logical unit may in some cases be the same as produced by other formatting tags. Remember, the tags specify logical units of the document, software other than the web browser may need this information.

Linking Pages

- Used to link text with **other** documents
- <A></A>
  - HREF
  - NAME (bookmarks inside the page)
  - TITLE (balloon help in IE)
  - TARGET (Define where the linked document will be opened)
- e.g.: `<A href="next.html"> Click here </A>`

link.html

- Used to link text with **same** documents
- e.g.:

```
<BODY link="blue" alink="green" vlink="red">
    <A name="top"> Top of the Page </A>
    ……
    <A href="#top">Top</A> </BODY>
```

Education and Research

Linking the pages

A link is a unidirectional pointer from a source document that contains the link to the some destination. Links help the user to navigate across pages as well as within a page. The text or an image that provides such link(s) is called hypertext or hyperlink.

Hyperlinks can be created by using a <A> tag, which stands for anchor and has the following attributes.

- HREF - Hypertext Reference: This attribute points the link to a bookmark, another file, either within the same webs site or elsewhere on the internet.
- NAME - Name: The name of the bookmark. This attribute lets you "bookmark" a location on the web page. An HREF anchor can point a link to that area on the page.

•TITLE - Displays balloon help in IE

•TARGET - With the target attribute, you can define where the linked document will be opened.

You can use HREF to point to a URL and allow the reader to view the page from the beginning. Or, you can use HREF to point to a specific area of that page, indicated by a NAME bookmark, so that the user goes straight to that section of the document.

Formatting the Link

The following attributes of <BODY> tag is used to provide color for the link.

LINK	-	Specifies the link color.
VLINK	-	Specifies the visited link color
ALINK	-	Specifies the active link color.

Lists

- **UnOrdered Lists - Bullets appear**
  - <UL> </UL> tag
  - <LI> tag
  - TYPE attributes specifies the type of bullet
  - TYPE = “disc” | “circle” | ”square”

```
<UL TYPE = “disc”>
    <LI> Item 1
    <LI> Item 2
    <LI> Item 3
</UL>
```

unord_list.html

O/P :

- Item 1
- Item 2
- Item 3

HTML has many ways to format lists information. Lists actually require two tags: The list tag itself, and the tag or tags used to define individual list items. Different types of list formatting available are discussed below.

Unordered Lists

- Unordered lists, are enclosed between the <UL > and </UL > tag.
- Each item starts with the <LI> tag .
- The attribute that can be specified with <UL> tag is

  TYPE= DISC will give a solid round black

  TYPE= SQUARE will give a solid square black bullet

  TYPE= CIRCLE will give a circle black bullet

Lists

- **Ordered Lists - serial numbers appear**
  - <OL> </OL> tag
  - <LI> tag
  - TYPE attribute controls the numbering scheme
    - TYPE = 1 | A | a | I | i

```
<OL TYPE = "1">
   <LI> Item 1
   <LI> Item 2
   <LI> Item 3
<OL>
```

ord_list.html

O/P : 1. Item 1
2. Item 2
3. Item 3

Ordered Lists

An Ordered list starts with a <OL> tag and ends with a </OL> tag.

Each list item starts with the tag <LI>

The attribute that can be set with <OL> are:

TYPE - Controls the numbering scheme to be used

TYPE= "1" will give counting numbers (1, 2, 3….)

TYPE= "A" will give uppercase letters (A, B, C….)

TYPE= "a" will give lowercase letters (a, b, c….)

TYPE= "I" will give uppercase Roman numerals (I, II, III…..)

TYPE= "i" will give lowercase Roman numerals (i, ii, iii,……..)

START - Alters the numbering sequence. Can be set to any value

VALUE - Can be set with the <LI >tag to changes the numbering sequence in the middle of an ordered list.

Lists

- **Definition List - defines a definition list**

  <DL> </DL> tag

  <DT> - Definition Term

  <DD> - Definition Data

```
<dl>
    <dt>Java</dt>
    <dd>An Object Oriented Language</dd>
    <dt>HTML</dt>
    <dd>A Markup Language</dd>
</dl>
```

def_list.html

O/P: Java

An Object Oriented Language

HTML

A Markup Language

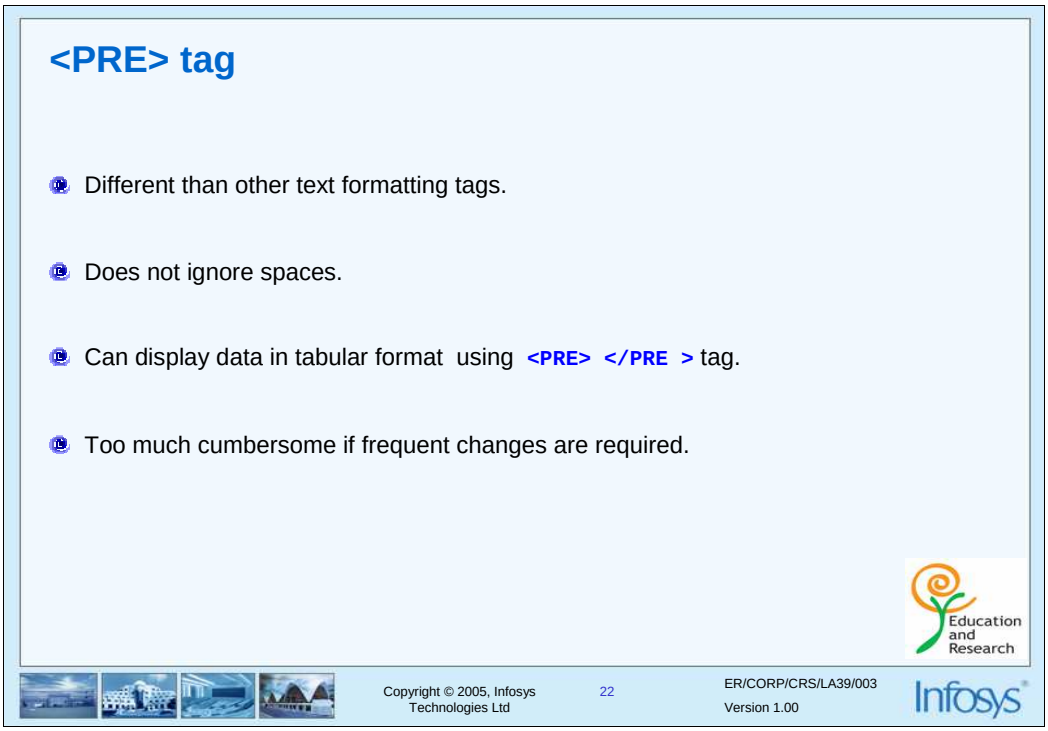
Education and Research

A definition list is not a list of items. This is a list of terms and explanation of the terms.

A definition list is a list of terms paired with associated definitions- in other words, a glossary.

Definition list are enclosed with in <dl> and </dl>

<dt>	-	Definition term.
<dd>	-	Data definition.



PRE –table era

Imagine creating a bus time-table or a movie ticket booking chart in a HTML page with whatever we know until now.

How can we place the elements correctly? When you code text into HTML format, browsers will ignore excess blank spaces and indents.

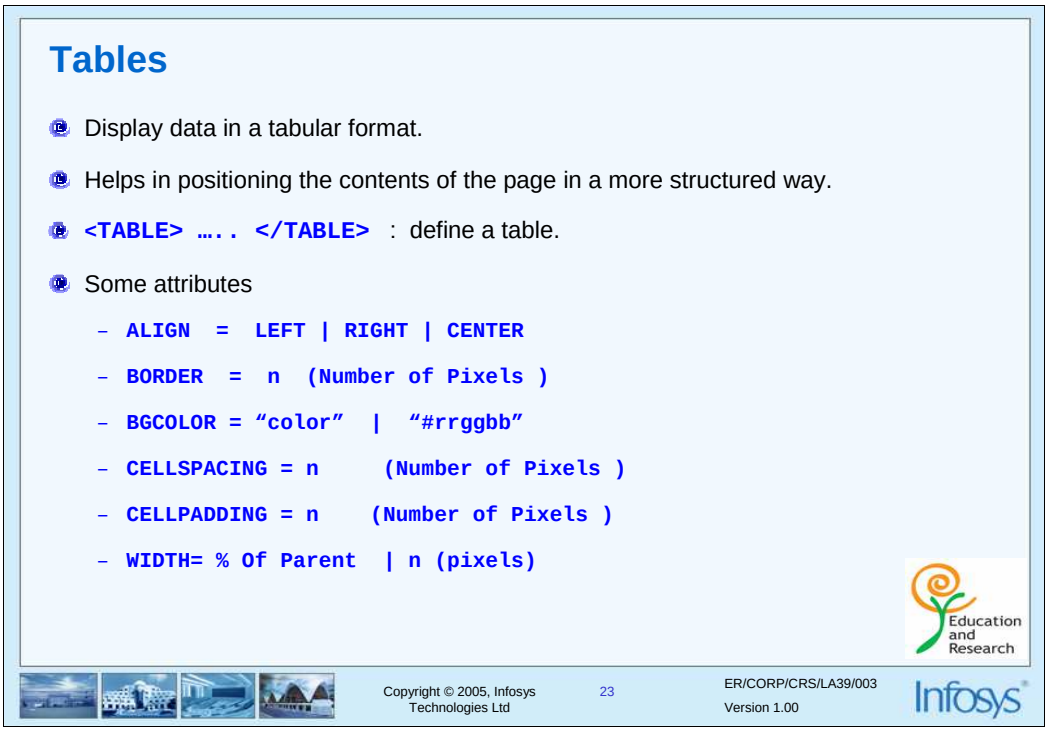
One exception to this is the PRE tag, which indicates the enclosed text should be displayed in the fashion it is entered.

To see how PRE works? Just try out the following code snippet.

```
<HTML>
<BODY>
<PRE>
    Let's make a marksheet
    Sr.   Student ID   CHSSC   PF    RDBMS
    1        1234          A      B     C
    2        5645          B      C     A
    3        6837          C      B     B
    4        9874          D      C     C
</PRE>
</BODY>
</HTML>
```

Did you see that the output looks identical to the HTML source?? However, it is not advisable to use the PRE tag for various reasons.

The solution to this is using tables



Tables can be used not only to display tabular data but also to position the contents of the page in a more structured manner. The page layout can be controlled very effectively with tables. Tables are defined with the <table> tag.

Note: Tables are used on websites for two major purposes.

1) The primary purpose of arranging information in a table
2) The more widely used purpose of creating a page layout with the use of hidden tables. (border attribute values set to “0”).

The Attributes and it values

ALIGN	- Align the table in a web page. (LEFT | RIGHT | CENTER).
BORDER	- Specifies the thickness of the border
BGCOLOR	- The background color for the table.
CELLPADDING	- Specifies the space between the cell wall and contents .
CELLSPACING	- Specifies the space between cell.
WIDTH	- Specifies the width of the table

The WIDTH attribute value can be in percent (or) pixels. Pixels can be thought of as the smallest logical unit for display. Pixel resolution can vary from PC to PC. Tables built with percents will occupy that percentage of the browser’s visible area or the container area.

Table structure

```
<TABLE BORDER=1>       <!-- start of table definition -->
<CAPTION> caption contents </CAPTION>   <!--caption definition -->

 <TR>            <!-- start of header row definition -->

    <TH> first header cell contents </TH>

    <TH> last header cell contents </TH>

 </TR>           <!-- end of header row definition -->
<TR>            <!-- start of first row definition -->

   <TD> first row, first cell contents </TD>

   <TD> first row, last cell contents </TD>

</TR>           <!-- end of first row definition -->
 <TR>           <!-- start of last row definition -->

    <TD> last row, first cell contents </TD>

    <TD> last row, last cell contents </TD>

 </TR>          <!-- end of last row definition –>
 </TABLE>       <!-- end of table definition -->
```

tab1.html

Infosys

Table structure

Tables are defined with the <TABLE> tag.

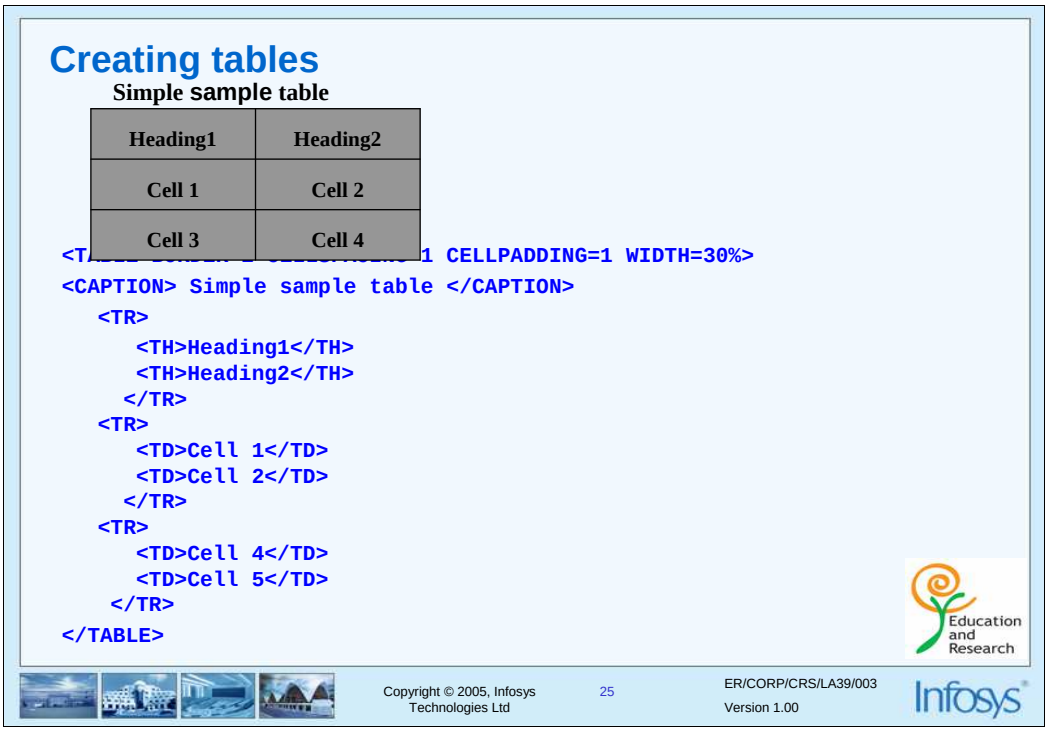
A table is divided into rows (with the <TR> tag), and each row is divided into data cells (with the <TD> tag).

The letters TD stands for “Table Data", which is the content of a data cell. A data cell can contain text, images, lists, paragraphs, forms, horizontal rules, tables, etc

<TH> specifies the table header cell. It should occur within a<TR> tag. Some browsers will emphasize the text enclosed by this tag.

<CAPTION> tag is used to give heading to the table. By default caption is placed above the table.

<CAPTION ALIGN =“bottom” > will place the caption below the table.



The <TD> Attributes.

ALIGN	- Defines the horizontal alignment in cells and values can be left, right, center, justify.
BGCOLOR	- Specifies the background color of the table cell.
VALIGN	- Specifies the vertical alignment of cell content and values can be top, iddle, bottom, baseline
WIDTH	- Specifies the width of the table cell.(% (or) pixels)
COLSPAN	- Indicates the number of columns this cell should span
ROWSPAN	- Indicates the number of rows this cell should span

Creating tables

This cell spans 2 columns!		Cell
This cell spans 3 rows!!	Cell	Cell
	Cell	Cell
	Cell	Cell

```
<TABLE>
    <TR ALIGN=CENTER >
        <TD COLSPAN=2>This cell spans 2 columns!</TD>
        <TD> Cell </TD></TR>
    <TR ALIGN=CENTER >
        <TD ROWSPAN=3>This cell spans 3 rows!!</TD>
        <TD> Cell </TD>
        <TD> Cell </TD></TR>
    <TR ALIGN=CENTER >
        <TD> Cell </TD>
        <TD> Cell </TD></TR>
    <TR ALIGN=CENTER >
        <TD> Cell </TD>
        <TD> Cell </TD>
    </TR>
</TABLE>
```

tab2.html

Education and Research

The rowspan and colspan attributes.

By adding the rowspan and colspan attributes to the table elements, it is possible to create data cells

that span a given number of rows or columns.

To set a cell to span three rows use <td rowspan="3"> and to set a heading to span two columns, use <th colspan="2">.

Setting the value of colspan and rowspan to more than the number of columns or rows in the table Should not extend the size of the table.

Summary

- What is HTML ? Where does it fit in?
- HTML tags and attributes
- Different text formatting tags like
  - Paragraph
  - Headings
  - <FONT> tag
- Linking pages : <A> tag
- Working with Lists
- <PRE> tag
- Tables : Different table tags and attributes.

Infosys®
POWERED BY INTELLECT
DRIVEN BY VALUES

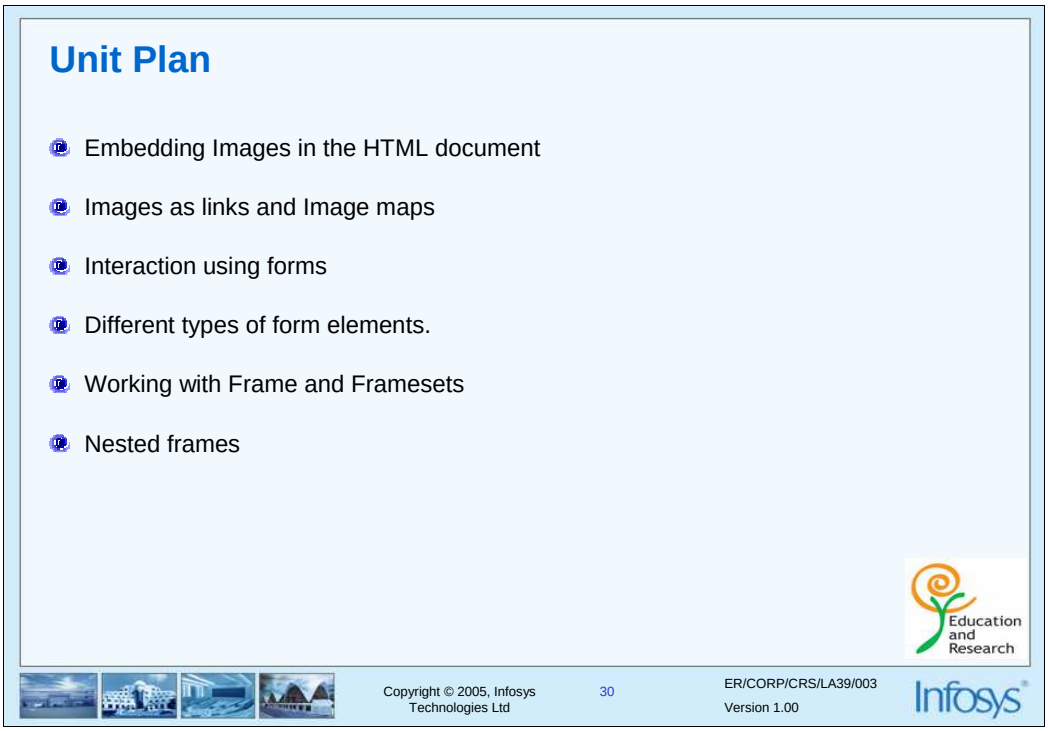
HTML, CSS, JavaScript

Unit2 – Images, Forms and Frames

Education and Research

Unit Objectives

- To explain how to embed images in a HTML document and using images as links and Image map.
- To explain how to divide the page into frames.
- To understand form designing and interaction using forms.



“A picture is worth a thousand words”. Images probably are the most obvious part of a a web site. Carefully used imagery can add to both the appeal and usability of a web site.

An image-map is a large image that contains numerous hot spots that can be selected , sending the user to a different anchor destination.

Forms enable users to submit information that can be used to create an interactive web application ranging from an order entry system to an email application.

A frame divides a browser window into multiple panes, or smaller window frames. Each frame can contain a different document.

Embedding Images

- <IMG> tag
  - SRC = “url”
  - ALIGN = “TOP | MIDDLE | BOTTOM ”
  - BORDER = n
  - WIDTH=n (in pixels)
  - HEIGHT=n (in pixels)
  - ALT=“Alternate Text”
- Supports JPG, GIF and PNG image formats.

ALT attribute was set to provide alternative text for browsers that did not display images. ALT attribute can be used as a tool tip or placeholder information in image-based browsers.

Most graphical browsers can display inline images. An Image can be embedded using the <IMG> tag, which takes the name of the image file as an attribute. Lists below are some attributes of the image tag.

Web pages require JPG (or) GIF (or) PNG image types. On the web, JPG is the best choice (smallest file) for photo images, and GIF is common for graphic images because GIF supports transparent color feature. PNG was designed recently, but it's appeal is growing as people discover what it can do. Most of the Browsers can not show BMP images.

JPEG: Joint Photographic Experts Group.

GIF: Graphic Interchange Format.

PNG: Portable Network Graphics Format.

Images as link

- Images when put in the anchor tag act as hyperlinks

```
<A HREF="Mypage.html">
  <IMG SRC="Littlejoe.jpg" >
</A>
```

linkimg

- Another form of clickable images is the idea of an image map.

  Client Side Image Maps.

  <A> element is used to enclose a specially marked **<img>** element.

  Server Side Image Maps.

  <map> element that defines the image map's active areas.

Images can be put inside the anchor tag (<A> </A>) to make them clickable images or hyperlinks which the user can follow to link to different URL's. The image will be displayed with a border. If you don't like the border, add BORDER=0 to the IMG tag.

An image map is an image that contains many hyper links that might result in a different URL being loaded, depending on where the user clicks.

Image Maps

- An image containing numerous hotspots.
- Each hot spot can load different URL's.
- Create an image map using <MAP> tag and <AREA> tag
  eg:
  - `<MAP NAME="forMap">`
    `<AREA SHAPE="circle" COORDS="20,20,40" HREF="Page1.htm">`
    `<AREA SHAPE ="rect" COORDS="100,100,50,50" HREF="Page2.htm">`
    `</MAP>`
- Apply the map created to an image.
  - `<IMG SRC="in.jpg" USEMAP ="#forMap">`
    `forMap` is the name of the map

AREA Attribute	Description/Value
SHAPE	Shape of the hot spot can be set to RECT, CIRCLE, POLYGON, DEFAULT
COORDS	Sets the points that define a shape.
HREF	Defines the destination of the link.

The name attribute of the <map> element is used to specify the identifier associated with the map.

The map name is referenced with in the <img> element using the usemap attribute.

The <area> tag is used to define area's inside the image so that the user can use that area as a hyperlink.

The attributes of <area> element are href, shape, and coords.

Shape	Coordinate Format.
rect	left-x, top-y, right-x, bottom-y
circle	center-x, center-y, radius
poly	x1,y1,x2,y2,x3,y3,……

Eg:

```
<img src="clouds.jpg" usemap="#formap" height=200 width=200>
<map name="formap">          <!-- define the sections -->
<area shape="circle" coords="20,20,40" href="hdj_page1.htm">
<area shape="rect" coords="100,100,50,50“ href="hdj_page2.htm">
<area shape=“polygon" coords=“225,112,309,35,336,53,445,78,556,89"
                href="hdj_page2.htm"> </map>
```

HTML Character Entities

Some characters like the < character, have a special meaning in HTML, and therefore cannot be used in the text. The most common character entities:

Result	Description	Entity Name
	non-breaking space	
<	less than	<
>	greater than	>
&	ampersand	&
"	quotation mark	"
'	apostrophe	'

Some Other Commonly Used Character Entities

©	copyright	©
®	registered trademark	®
£	pound	£
¥	yen	¥

Education and Research

Infosys

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;

code is a word or numeric code indicating the actual character that you want to put onscreen.

Eg:

<h1> All html tags should be inside the <HTML> Tag</h1>

Forms

- Set of fields that can record information
- To interact with the client
- FORM by itself really cannot do anything
- Forms become powerful when connected to a server application
- A single HTML page can have multiple forms.

A form in HTML is enclosed between the <form> and <form> tags.

A form is an area that can contain form elements.

The form itself contains regular text, other HTML elements such as tables, and form elements.

Form elements allow the user to enter information (like text fields, textarea fields, drop-down menus, radio buttons, checkboxes, etc.) in a form.

Forms become a powerful tool when connected to a server application. When the user clicks on the submit button the web browser transmits all the information in the fields to the server and server side program is invoked.

JSP, ASP, and CGI are some examples of server side programs.

Form data can be manipulated by client side scripts like Javascript and VBScript.

Forms

- Can be designed using <FORM></FORM> tag

```
<FORM NAME="form1" ACTION="abc.asp" METHOD=GET>

 <!- NAME is used for future manipulation of data by scripting
 language

 ACTION indicates a program on the server that will be executed
 when this form is submitted. Mostly it will be an ASP or a CGI
 script.

 METHOD indicates the way the form is submitted to the server -
 popular options are GET/POST -->


(form elements go here)

  </FORM>
```

The differences between the two methods of submitting data GET/POST are summarized below:

GET method.

1) The GET method is the default method for browsers to submit information.
2) GET is easy to deal and fast.
3) All the information form the form is appended onto the end of the URL.
4) It is not secure because the data input appears in the URL.
5) Most browsers limit a URL to several thousand characters.

POST method.

1) Used to pass large amount of information to the server.
2) Contents are sent with HTTP request body not with URL.

Form elements

- <INPUT> tag is used to add elements to the form.
  - NAME = “controlname”
  - TYPE = text / password / checkbox / radio/ submit / reset / button / hidden / file
  - VALUE
  - MAXLENGTH
  - SIZE

form.html

All elements should be named by setting a unique value to the name attribute.

The value attribute is used to set a default value for the control.

Education and Research

Elements on a form

A form is made up of fields or controls. Form controls include text fields, password fields, multiline text fields, scroll lists etc.

All form elements should be enclosed in <form> and </form> tag.

```
<html>
<body>
    <form name=form1>
                Name <input type="text" name="nam"><br>
                DOB <input type="text" name="dob"><br>
    </form>
</body>
</html>
```

Text Box/Password

- A text field can be added to the form by typing
  - `<INPUT TYPE="TEXT" NAME="txtcompany" VALUE="INFOSYS" SIZE="10" MAXLENGTH="15">`
- A password field can be added to the form by typing
  - `<INPUT TYPE=PASSWORD NAME=pwdLogin SIZE=50 MAXLENGTH=12>`
  - Input to the field will not be revealed.
- Attributes are
  - VALUE is the default value loaded
  - SIZE sets the size of the field in no. of characters.
  - MAXLENGTH specifies max number of characters that can be entered to the control.

Text field.

Text is the default attribute for the <input> element type attribute.

<input name="t1" value="ok"> will create a a textbox.

Examples

- <INPUT TYPE="TEXT" NAME="txtCustomerName">

  Will create a text box. Since the above statement does not specify the size of the field or the maximum number of characters the field will generally be 20 characters.
- <INPUT TYPE="TEXT" NAME="txtCustomerName" SIZE ="40" >

  Will create a text box of size 40 .The user can type any number of characters.

  To restrict the number of characters use attribute MAXLENGTH
- <INPUT TYPE="TEXT" NAME="txtCustomerName" SIZE ="40" MAXLENGTH ="30">

  Will create a text box of size 40 .The user can type 30 characters.

Password field

The password field are same as the text field, except that the input to the field is not revealed. All the attributes of the text filed are applicable to the password field.

Text Area

- Multiline text input
  - `<TEXTAREA NAME="feedback" ROWS=3 COLS=40>` `Default text goes here`
    `</TEXTAREA>`
- ROWS is the number of rows desired.
- COLS is the no of characters per line.
- Default text is optional
- It is not possible to set the default text using the VALUE attribute
- The default text is to be put into

<TEXTAREA> </TEXTAREA> tags

When it is necessary to enter more than one line of text in a form field we can use <textarea>

The information enclosed within the <textarea> element must be plain text and should not include any HTML markup.

Internet explorer will wrap the text by default.

Some old versions of Netscape Browser will not support word wrap.

List Box

```
<SELECT NAME="Hobbies" MULTIPLE SIZE="3">
   <OPTION VALUE="TR">Travel
   <OPTION SELECTED>Reading
   <OPTION>Sleeping
   <OPTION>Walking
</SELECT>
```

- SIZE number of lines to display
- VALUE indicates what will be sent to the server
- SELECTED sets the default selected item

Lists let the user select one choice out of many possible choices. The SELECT element contains one or more OPTION elements to provide a menu of choices for the user. Its attributes are

•NAME attribute provides the key sent to the server with the value of the selected option. By default, the user can only select one option. The MULTIPLE attribute allows the user to select multiple options, which are submitted as separate name/value pairs.

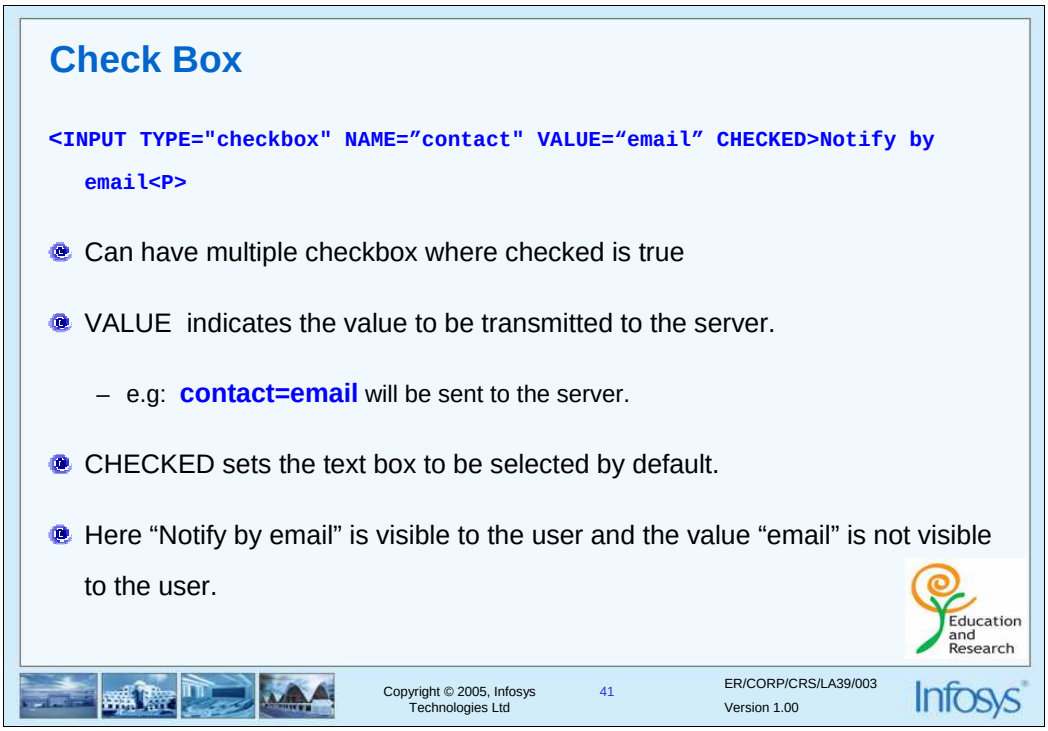
•The DISABLED attribute makes the SELECT element unavailable. The user is unable to edit the disabled selection, no value is submitted with the form, the SELECT element cannot receive focus, and the element is skipped when navigating the document by tabbing.

•The SIZE attribute of SELECT hints that visual browsers should display the element as a list box with the specified number of options visible at any time. A scroll bar would allow access to any non-visible options.

•The value enclosed by the OPTION element is submitted to the server.

•The attribute SELECTED in the OPTION element sets the form control to select this item by default.

• With the scroll list, it is possible to select multiple items out of a large group of items. But all items are not presented to the user at once.



•If there are a few options to choose from that are not mutually exclusive, it is probably better to use a group of check boxes that the user can check off.

• For example, to display a check box whether the user wants the facility of Internet banking use

<INPUT TYPE="checkbox" NAME="InternetBanking" > Internet Banking

In this example, if the checkbox is selected, value InternetBanking =no will be transmitted to the server.

• Setting a value for the checkbox might make more sense.

<INPUT TYPE="checkbox" NAME="Facilities" VALUE= "InternetBanking">

If the above check box is selected Facilites = InternetBanking will sent to the server

• It is also possible to have multiple check boxes with the same name.

The code

<INPUT TYPE="checkbox" NAME="Facilities" VALUE= "InternetBanking">

•<INPUT TYPE="checkbox" NAME="Facilities" VALUE= "DebitCard">

•Would send multiple entries like the following to the server when both the facilities were selected

•Facilities= InternetBanking&Facilities=DEbitCard

Radio Buttons

```
<INPUT TYPE="radio" NAME="output" VALUE="screen“ checked> Screen
<INPUT TYPE="radio" NAME="output" VALUE="printer">Printer
```

- Radio buttons with the same NAME are grouped together.
- Only one button can be selected in a group.
- VALUE data to be sent to the server.
- CHECKED will preselect the button.

The second mechanism for selection is a set of radio buttons. In a set of radio buttons only one can be selected at a time - to do this all the buttons should be given the same value for the NAME attribute. Which is known as a radio group.

e.g.

<INPUT TYPE=RADIO NAME=rdbAgeGroup VALUE=1>

<INPUT TYPE=RADIO NAME=rdbAgeGroup VALUE=2>.

You can select only one of the two radio buttons. Since they form a group.

• It is important to set the VALUE attribute for each radio button.

• Value will not be displayed to the user.

•If you are used “Checked” attribute the button will be selected by default.

Hidden text field

```
<INPUT TYPE="hidden"  NAME="useinformation"  VALUE ="form1">
```

- Can be used to transmit default or previously specified data.
- Can be used to pass data from one form to another.
- Cannot be modified by the user
- So it must have a VALUE attribute set
- VALUE data to be sent to the server.

Hidden Text field

<INPUT TYPE="hidden" NAME="userinformation" VALUE ="form1">

•creates a hidden text field.

•With a hidden text it is possible to transmit default or previously specified text that is hidden from the user.

•Since the hidden field cannot be modified by the user, its VALUE attribute must be set.

•This hidden text can be utilized for handling programs.

Buttons

The Submit button

- Sends the form contents to the server when clicked
- `<INPUT TYPE=submit NAME=cmdsubmit VALUE =“Submit”>`

The Reset button

- Resets all the form controls to the default state.
- `<INPUT TYPE=Reset NAME=cmdReset VALUE="Reset">.`

A button

- No predetermined action like submit or reset.
- Script should be written to make it work. (this will be covered in later chapters)
- `<INPUT TYPE=Button NAME=cmdAdd VALUE=“Click Me">.`

Types buttons

- The Submit button sends the form contents to the server when clicked.

  <INPUT TYPE=Submit NAME=cmdSubmit VALUE="Submit">
- The Reset button : Contrary to popular expectations, it does not clear the form but resets it to its original state. For e.g. if a textbox started off with a default value of "Me" and it was subsequently changed to "You" by the user, clicking on the Reset button will change it back to "Me".

  <INPUT TYPE=Reset NAME=cmdReset VALUE="Reset">.

- A button <INPUT TYPE=Button NAME=cmdSubmit VALUE="Submit"> doesn't do anything when clicked - you have to write some script (coming up in the next unit) to make it work. Then why use it at all? There may be times when some action other than submitting a form needs to be done on a button.

  You will learn more about client-side validations in later part of this course.

File and Image

The File Control

- Available from HTML 4.0.
- This form control is used to upload a file to the server.
- `<INPUT TYPE="file" NAME="load">`
- It is possible to set maxlength and size values to file control.
- Not suggested because the path name might be larger than the size specified.
- The file form control is not supported by all browsers.

The Image Control.

- The image control creates a graphical version of submit button.
- `<INPUT TYPE="IMAGE" SRC="sub.gif" alt="submit to server" NAME="flname">`

The file control generally consists of a text entry box and a button immediately to the right of the field.

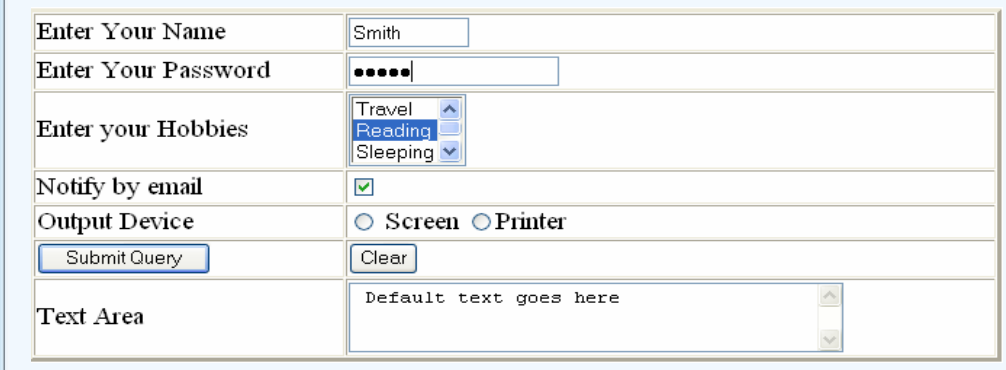
By clicking the "Browse" button enables the user to browse the local system to find a file to specify for upload.

To upload a file to the server set the <form> element attribute enctype value as "multipart/form-data".

The image control creates a graphical version of submit button, which not only submits the form but transmits coordinate information about where the user clicked in the image.

The image is specified by the src attribute.

Form example



- To display the form elements in a visually appealing way, put them into table cells as shown in the above form.

formdemo.html


```
<html><body>
<form  action="" method=post  name=form1>
<table  border=3 cellpadding=1 cellspacing=1 width="75%">
<tr> <td>enter your name</td>
      <td><input name="first_name" value="smith" size="10" maxlength="15" ></td>
</tr>  <tr>  <td>enter your password</td>
      <td><input name=password1 type=password></td>
</tr>  <tr>  <td>enter your hobbies</td><td><select name="hobbies" multiple size="2">
              <option value="tr">travel
              <option selected>reading
            <option>sleeping </select>  </td>
</tr>  <tr>    <td>notify by email</td>
       <td><input type="checkbox" name="email" checked></td>
</tr>  <tr>    <td>output device </td>
         <td><input type="radio" name="output" value="screen"> screen
             <input type="radio" name="output" value="printer">printer </td>
</tr>  <tr>    <td><input type="submit" value="submit query" name=submit1></td>
           <td><input type="reset" value="clear" name=reset1></td>
</tr>  <tr>    <td>text area</td>
        <td><textarea  name="feedback" rows =3 cols= 40> default text goes here
            </textarea></td>
 </tr>
 </table></form></body></html>
```

Frames

- Divides a browser window into number of panes.
- Each frame may contain a different document.
- <FRAMESET> element defines a set of frames.
  - Should preclude the <BODY> element.
  - <FRAMESET> </FRAMESET>
    - ROWS
    - COLS
- <FRAME > element defines an individual frame
  - NAME = “frame name”
  - SRC=“url”
  - SCROLLING = auto / yes / no
  - NORESIZE

Frames

With frames, the browser window can be split into separate sections, showing different but related information.

You can display more than one HTML document in the same browser window.

Each HTML document is called a frame, and each frame is independent of the others.

Each frame has its own URL, thus one frame can stay visible while another changes.

The Frameset and Frame Tag:

The <frameset> tag defines how to divide the window into frames.

Each Frameset can have number of frames defined by the frame tag.

The <frame> tag defines what HTML document to put into each frame.

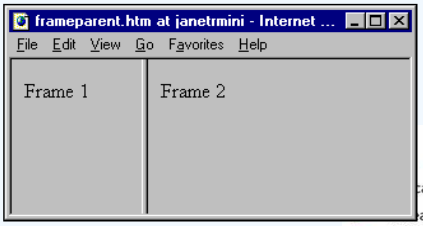
The <frameset> requires a mandatory attribute of either COLS or ROWS that decides the number and size of columns or rows in a browser window.

The disadvantages of using frames are:

- The web developer must keep track of more HTML documents
- It is difficult to print the entire page

Adding Frames

```
<HTML>
<!--comment: Two cols with 30% and 70% each -->
<FRAMESET COLS="30%,*">
<!-- Comment: * is indicating the remaining space  (here 70%) -->
   <FRAME SRC="Frame1.htm">
   <FRAME SRC="Frame2.htm">
</FRAMESET>
<HTML>
<!-- Comment:  no need to use the <BODY> tag at all  -->
```



Attributes of the FRAMESET/FRAME tag are as given below.

Frameset.

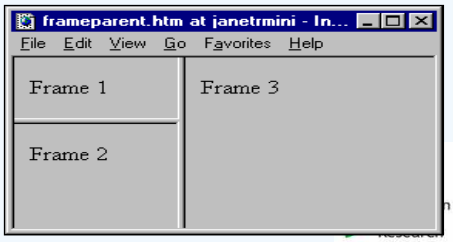
COLS	: Defines the number of frame columns and their widths.
ROWS	: Defines the number of frame rows and their heights.
FRAMEBORDER	: Defines borders around each frame, A value of "0" leaves no visible borders and a value of "1" display the border.
FRAMESPACING	: Defines the space in pixels between frames.
SCROLLING	: Determines weather or not scroll bars are displayed on all the frames, value are YES, NO and AUTO.

Frame.

NAME	: Name of the frame.
NORESIZE	: Prevents the user from resizing the frame.
SCROLLING	: Control the display of scrollbars. The value are AUTO (or) NO.
SRC	: Specifies the full or partial URL of the document displayed in a frame.

Nesting Frames

```
<HTML>
<FRAMESET COLS="40%,*">
      <FRAMESET ROWS="35%,*">
            <FRAME SRC="Cell1.htm">
            <FRAME SRC="Cell2.htm">
      </FRAMESET>
      <FRAME SRC="Cell3.htm">
</FRAMESET>
</HTML>
```



In the example above we have two framesets.

The first frameset have two columns one of 40% width and the other of the remaining width i.e 60% note that it is indicated by a *.

The second frameset set divides the first column into two rows. First row of width 35% and the second row remaining width i.e 65% note that it is indicated by a *.

The HTML document “cell1.htm" is put into the first row and first column The HTML document “cell2.htm" is put into the second row and first column.

The HTML document “cell3.htm" is put into the second column.

Frame targeting

- Ensure frame naming
-
```
<FRAMESET COLS="*, 20%">
    <FRAME NAME="Frame1" SRC="Cell_1.htm">
    <FRAME NAME="Frame2" SRC="Cell_2.htm">
</FRAMESET>
```
- Set the TARGET attribute for one hyperlink. Here the target value is case sensitive.

```
<A HREF="file.htm" TARGET="Frame1">Link Text</A>
<A HREF="file.htm" TARGET="_top">Link Text</A>
```

- By specifying a BASE target, you can automatically set a default target for all links in a Frame.
- If the TARGET attribute is specified in the <A> tag then that target will be used.
  - `<BASE TARGET="Frame2">`

Education and Research

Having made the frame structure, one would also be interested in doing things like clicking on something in one frame and having some action in some other frame. So here's how to do it. This involves two steps .

1. Ensure frame naming by setting the NAME attribute of the <FRAME> tag
   e.g: <FRAME SRC="HDJ_Page1.htm" NAME="frame1">.
2. Use the TARGET attribute in the <A> tag to specify the target frame where the page is to be displayed.

Target can be a name of the frame that you specified in the FRAME tag or one of the following predefined values.

_blank : Loads the link into a new blank window
_parent : Loads the link into the immediate parent of the document the link is in.
_self : Loads the link into the same window. (default)
_top : Loads the link into the full body of the window.

Value	Description
_blank	Loads the page into a new window
_self	Loads the page over the current frame
_parent	Loads the link over the parent frame
_top	Loads the link over all the frames in the window.

Floating Frames (Inline Frames)

- Floating Frames
- Floating frames are scrollable areas that appear in a HTML document. Unlike regular frames they cannot be resized.
- Not attached to the sides of the browser.
  - Acts similar to an embedded object.
  - Occurs within the <BODY> .
  - <IFRAME> </IFRAME> tag.

```
<IFRAME SRC="bot.html" WIDTH="450" HEIGHT="400"></IFRAME>
```

Floating Frames

Up to this point all frames shown have been attached to the sides of the browser. Another form of frame, called a floating frame also called as inline frame. These are used to create an inline framed region, or window, that acts as any other embedded object. An inline frame can be defined by a <IFRAME> tag and may occur anywhere within the <BODY> tag. Unlike the <FRAME> element which should occur only within the <FRAMESET> element.

The major attributes that can be set here are

- SRC is set to the URL of the file to load,
- HEIGHT and WIDTH are set in pixel or percentage value of the screen that the floating –frame region should consume.

```
<html>
<head>
<title>floating frame example</title>
</head>
<body>
<h1 align="center"> floating frame </h1>
<iframe name="floatframe1 src="images.htm" height=100 width=100 align="left">
there will  be a floating frame if your browser supports it
</iframe>
<p> this is a simple example of how floating frames are used.
</body>
</html>
```

<NOFRAMES> tags

- <NOFRAMES> </NOFRAMES> tag
  - Encloses text to be displayed when the browser does not support frames.
  - There are some of the browser versions that do not support frames at all! They will either get an error message or a blank screen.

```
<NOFRAMES>
  <BODY>
        <H1>Your Browser do not support Frames</H1>
  </BODY>
<NOFRAMES>
```

  - Some of the browsers do not support <IFRAME>

```
<IFRAME SRC="bot.html" WIDTH="450" HEIGHT="400">
  <H1> Your Browser do not support Inline Frame
</IFRAME>
```

For the browsers who don't support frames

There are some of the browser brethren that do not support frames at all. They will either get an error message or a blank screen. So how do we deal with them? Simple!

After the first <FRAMESET> tag, put a message for such browsers between <NOFRAMES> and </NOFRAMES> tags. You could also include a link to a non-frames version of the page.

Usage:

```
<html>
<frameset cols="*, 2*">
    <frame src="frame1.htm">
    <frame src="frame2.htm">
    <noframes> <p>this document uses frames. please follow this link for the <a href="noframes.htm"></a> version
    <noframes>
</frameset>
<html>
```

Summary

- <IMG> tag and its attributes.
- Images as links and Image maps
- Interaction with the client using forms.
- Working with Frame and Framesets
- Working with Inline Frames.
- Different type of elements.

Infosys®
POWERED BY INTELLECT
DRIVEN BY VALUES

HTML, CSS and JavaScript

Unit3 - Cascading Style Sheet (CSS)

Education and Research

Unit Plan

- What are Style Sheets, Its syntax and working
- Cascading Style sheets (CSS) with HTML doc
- Same related tags - <SPAN> and <DIV>
- Style Sheet properties

Cascading Style sheets (CSS)

- In **1996, W3C** announced style sheets.
  Two cascading style sheet standards exist:
  CSS1 was a set of rules to format and enhance text, paragraphs and documents.
  CSS2 the current standard adds to the CSS1 base a set of styles for visual browsers, aural devices, printers and so on.

- Features
  - Separates the presentation and contents of the HTML document.
  - Provide numerous attributes to create dynamic effects.
  - Simple.
  - Reusable.

- Style Sheet
  - A set of statements that specify presentation of a document.
  - A powerful mechanism for adding styles.
  - Styles can be assigned by the <STYLE> </STYLE> tag.

Style Sheets is a breakthrough in Web design because it allows developers to control the style and layout of multiple Web pages all at once. A web developer can define a style for each HTML element and apply it to as many Web pages as you want. To make a global change, simply change the style, and all elements in the web are updated automatically.

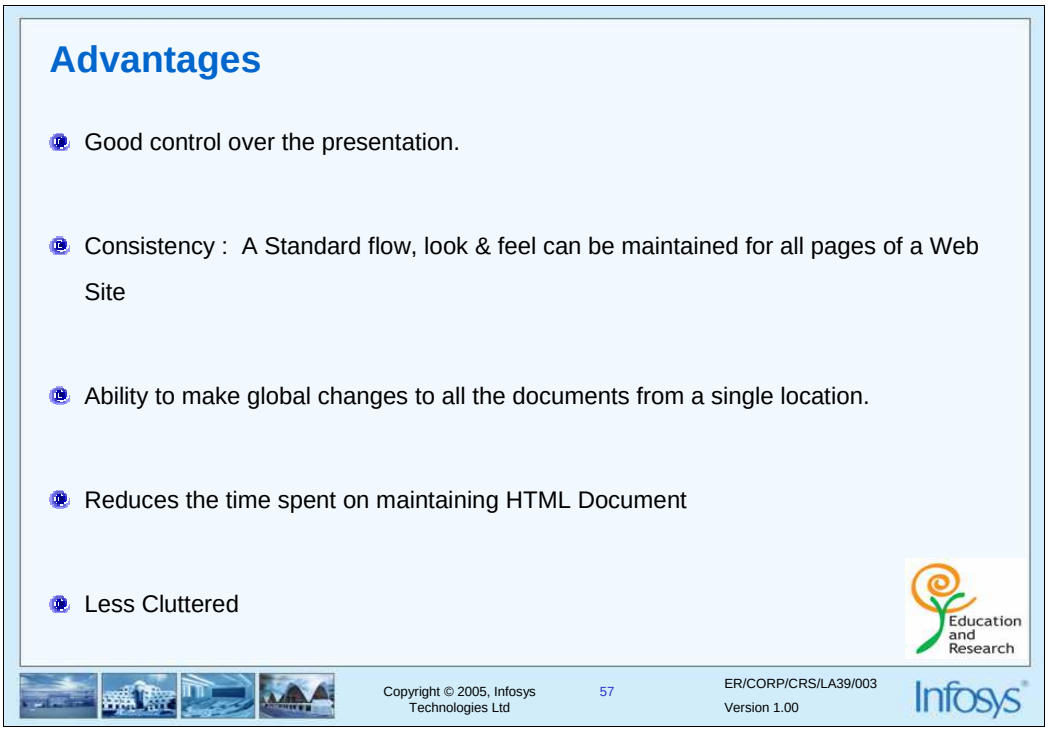
Style sheets are said to cascade when they combine to specify the web page.

A style sheet is made up of style rules that tell a browser how to present a document.

The simplest way of using style sheet with HTML document is embed styles in <STYLE> element.

This element is placed in the document HEAD, and it contains the style rules for the page.

Each rule is made up of a selector--usually an HTML element such as BODY, P, or H1.



Advantages

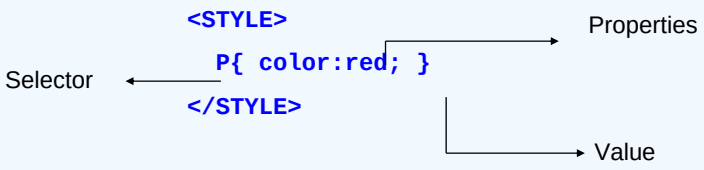
•Since style sheets support a wide variety of properties. It has a good control over the presentation.

•With the external style sheets the contents and the presentation can be separated. The same style sheet can be linked to one or more web pages. This maintains a standard look and feel for all the pages of a web site.

•Changes made to the external style sheet reflect globally in all the connected web pages. This helps in reducing the effort and time spent for web site maintenance.

How do Style Sheets Work?

- Separate Section is defined to place the Style Properties of the Document
  - Section consists of two parts
    - Selectors
    - Declarations
- Defined sections in the document are attached with their respective properties
- <STYLE > tag is used to define styles.



The selector is normally the HTML element/tag you wish to define, the property is the attribute you wish to change, and each property can take a value. The property and value are separated by a colon and surrounded by curly braces.

The selector is simply the element that is linked to a particular style .

The example above is a simple CSS rule. A rule consists of two main parts: selector (‘P') and declaration ('color: red'). The declaration has two parts: property ('color') and value (‘red'). While the example above tries to influence only one of the properties needed for rendering an HTML document .

Selectors

A selector identifies elements on an HTML page

- **Element Selector (Type Selector)**

  - An Element selector matches the name of a document language element.

  Eg. <H1> , <P>

- **Inheritance**

  - Style properties are inherited from the parent element to the child el  nt .

  cssdemo1.html

```
<BODY>
      <H1> H1 inherits to BODY style <H1>
</BODY>
```


The selector is the link between the HTML document and the style sheet, and all HTML element types are possible selectors.

Inheritance

Virtually all selectors which are nested within selectors will inherit the property values assigned to the outer selector unless otherwise modified. For example, a color defined for the BODY will also be applied to text in a paragraph.

```
<html>
<head>
        <style>
                h2{
                        color:blue;
                        font-size:80pt;
                }
        </style>
</head>
<body>

        <h2> This is CSS style</h2>
</body>
</html>
```

Selectors

- **Class selectors**

  - With the class selector you can define different styles for the same type of HTML element.

  Eg: `<H1 class="head1">Hello</h1>`

- **ID selectors**

  - The ID attribute of a document language allows authors to assign an  fier to one element .

  cssdemo2.html

  - An ID attribute must be unique within the document.

  Eg: `<P id="para1">First para</P>`

Education and Research

Class Selector.

A simple selector can have different classes, thus allowing the same element to have different styles. For example, an author may wish to display code in a different color depending on its language:

```
code.html { color: #191970 }
code.css { color: #4b0082 }
```

Classes may also be declared without an associated element:

```
.css { color: red}
```

ID Selector.

ID selectors are individually assigned for the purpose of defining on a per-element basis. This selector type should only be used sparingly due to its inherent limitations. An ID selector is assigned by using the indicator "#" to precede a name.

```
#top1{  color: red}
<P ID="top1">The Top Paragraph</P>
```

```
<html><head>
<style>
H1          {          color:red; font_size:20pt;}
h1.c        {          color:blue;       }
</style><head>
<body>
            <h1> First H1 Tag</h1>
            <h1 class="c"> Second H1 Tag with class attribute</h1>
</body>
</html>
```

Selectors and Comments

- **Contextual Selectors**

  - Contextual selectors are merely strings of two or more simple selectors separated by white space.

  Eg. `P EM { background: yellow }`

- **Grouping**

  - In order to decrease repetitious statements within style sheets, grouping of selectors and declarations is allowed.

  Eg: `H1, P, TD { color:red;}`

Comments

- Comments are denoted within style sheets with the same conventions that are used in C programming.

```
/* COMMENTS CANNOT BE NESTED */
```

Contextual Selectors

For example, the contextual selector

P EM { background: yellow }

This rule says that emphasized text within a paragraph should have a yellow background; emphasized text in a heading would be unaffected.

Grouping.

For example, all of the headings in a document could be given identical declarations through a grouping:

```
H1, H2, H3, H4, H5, H6 {
        color: red; font-family: sans-serif
}
```

Pseudo-classes

- Pseudo-classes are special "classes" that are automatically recognized by CSS-supporting browsers. Pseudo-classes distinguish among different element types (e.g., visited links and active links represent two types of anchors).

- **Anchor Pseudo-classes**

  Pseudo-classes can be assigned to the **A** element to display links, visited links and active links differently.

csslinks.html

```
A:link          { color: blue; }
A:active     { color: green; }
A:visited    { color: red; }
A:hover         { color: cyan; }
```

Education and Research

Rules with pseudo-classes take the form

selector:pseudo-class { property: value }

Pseudo-classes and pseudo-elements should not be specified with HTML's CLASS attribute.

Normal classes may be used with pseudo-classes as follows.

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

The anchor element can give the pseudo-classes link, visited, or active. A visited link could be defined to render in a different color and even a different font size and style.

```
<html>
<head><style>
a:link{          color:blue;       }
a:hover{         color:red;        }
</style>
<head><body>
                 <h2> Open sparsh <a href="http://sparsh">Sparsh</a></h2>
</body></html>
```

Ways of specifying styles

1) Inline

2) Embedded (Internal styles sheet)

3) External Style sheets (Linking)

4) Importing

Ways of specifying styles

1) Inline

```
<P style="color:blue; margin-right: 10px;">
Styled paragraph
</P>
```

- Can be applied to a single occurrence of an element
- Mixes content with presentation
- Should be used sparingly

Style Sheets allow style information to be specified in many ways. Styles can be specified inside a single HTML element, inside the <HEAD> element of an HTML page, or in an external CSS file. Even multiple external Style Sheets can be referenced inside a single HTML document.

Ways of specifying styles

2) Embedded (Internal styles sheet)

- Can be used by single document.
- Enclosed within the HEAD tag.

```
<HEAD>
<!–
<STYLE TYPE=“text/css”>
    HR{color:blue}
    P{margin-right:10px}
</STYLE>
 -->
</HEAD>
```

cssemb.html

Embedded Style : The style definition is part of the HTML doc - typically within the HEAD tags. This is much better than inline. If a change has to be made to a certain tag e.g. H1 in the page, you need to make it only in one place i.e. in the style definition (unlike inline styles where the change has to be made at every occurrence of H1). However embedded styles still have the drawback of not providing automatic consistency over a set of HTML pages.

```
<html><head>
<style>
                /* Using Embedded style   */
                        h2{
                                color:blue;
                                font-size:40pt;
                                letter-spacing :10pt;
                        }
</style></head><body>
                <!-- Comments  : Using inline style                -->
                <h2 style="color:red;">Using Inline and Embedded</h2>
</body></html>
```

Ways of specifying styles

3) External Style sheets (Linking)

- Style Properties are defined and placed in external files and is saved with extension .css
- These files are then Cascaded with the HTML Documents and properties are suitably applied.

```
<HEAD>
   <LINK REL="stylesheet" TYPE="text/css" HREF="mystyle.css">
</HEAD>
```

An external style sheet is ideal when the style is applied to many pages. With an external style sheet, you can change the look of an entire Web site by changing one file.

- All the style definitions are stored in a separate css file and referenced from the HTML document using a LINK tag..

  <LINK REL=stylesheet HREF="myStyle.css" TYPE=text/css>
- REL is the relationship specifier

  HREF specifies the URL of the style sheet to use.

  TYPE specifies the content type. e.g: "text/html", "image/png", "image/gif", "video/mpeg", "text/css",
- The above line can be kept in the HEAD tag.
- The browser will read the style definitions from the file mystyle.css, and format the document according to it.
- This is best way to use CSS because the same .css file can be used for the entire web site.

Ways to cascade style sheets

4) Importing

- By importing the Style sheet
  - @import url("<filename>.css ")
- Multiple websites can use the same style sheet.

```
<style type="text/css">
@import url(http://www.xyz.com/style2.css)
</style>
```

Apart from the <LINK> tag ,importing a style to a document is another way to use a document –wide style. An external style sheet is referenced ; but, in this case, the reference is similar to a macro expansion inline.

The syntax is : @import followed by the URL of the style sheet to import.

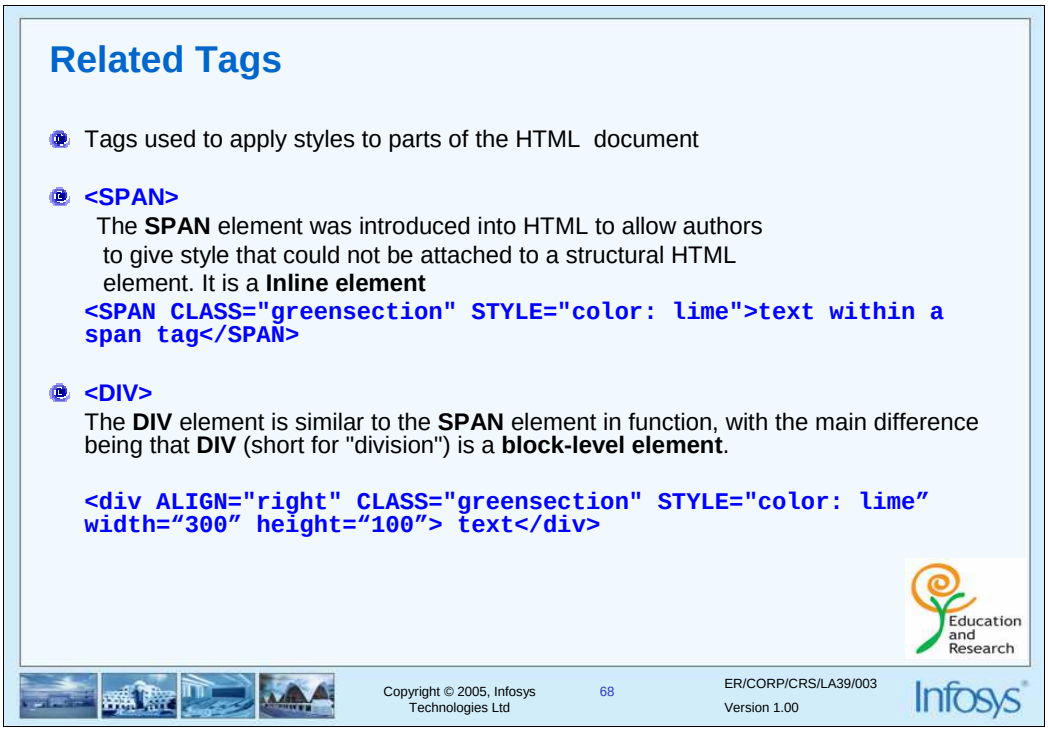
e.g . @import "http://www.mywebsite.com/styles/mystyle.css"

What style will be used when there is more than one style specified for an HTML element?

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

- Inline Style (inside HTML element)
- Internal Style Sheet (inside the <head> tag)
- External Style Sheet
- Browser default

So, an inline style (inside an HTML element) has the highest priority, which means that it will override every style declared inside the <head> tag, in an external style sheet, and in a browser (a default value).



The DIV and SPAN elements, in conjunction with the id and class attributes, offer a generic mechanism for adding structure to documents. These elements define content to be inline (SPAN) or block-level (DIV) but impose no other presentational idioms on the content. Thus, web page authors may use these elements in conjunction with style sheets, ., to tailor HTML to their own needs and tastes

DIV element:

•The DIV element is an all-purpose, generalized HTML block structure. Use this element when you wish to define a block or section of Styled text.

•The DIV Formatting element is with an implied line break before and after the enclosed contents.

•The DIV element is nestable to allow hierarchies of sections, subsections or chapters to be defined.

DIV element Attributes:

•ALIGN :This indicates the horizontal alignment of the division block text in the browser window. These values can be over-ridden by style sheets values

•COLS :It indicates the number of evenly distributed columns the contained content will be split into.

•HEIGHT and WIDTH :This attribute explicitly specifies the height and width of this block element in pixels.

•NOWRAP :If False, normal line breaking behavior is used. If set to True, the element will not wrap to the rendering viewport unless explicit line breaking elements are added.

All of the above are the style properties that are supported. In this unit we will be looking at all these properties and their corresponding attributes in detail.

Most of the above properties have attributes that require length values to be specified. Lets look at the length values in detail.

Length values

• It can have positive or negative values, have a numerical value followed by a unit of measurement.

•Two kind of length unit, relative and absolute

Unit Name	Abbreviation	Explanation
Em	Em	The height of the font
Ex	Ex	Height of the letter x in the font.
Pica	Pc	1 pica is 12 points
Point	Pt	1/72 of an inch
Pixel	Px	One dot on the screen
Millimeter	Mm	Printing unit
Centimeter	Cm	Printing unit
Inch	In	Printing unit

In the table above Em, Ex and pixel are relative units, rest are absolute.

NOTE : Absolute units should be used only when the physical characteristics of the output device is known.

Background and Color Properties

- **background**
  - background : “color” / “#rrggbb” / url(“*.gif”)
- **color**
  - color : “color name” / “#rrggbb”

Eg. `BODY{ Background:”red”;}`

Properties	Values
background-attachment	scroll ,fixed
background-image	URL, none
background-repeat	repeat, repeat-x, repeat-y, no-repeat
background-color	color-rgb, color-hex, color-name, transparent

Color properties

CSS supports a variety of properties that can be used to control the color and backgrounds.

It supports three basic forms of color specifications:

• Color names : 16 colors keywords are supported by browsers. These are the same predefined colors from the HTML specifications.

• Hexadecimal : Supports standard 6-digit color form #RRGGBB same as used within the <FONT> and <BODY> tag.

• RGB values : The RGB format is also specified in the form rgb (R,G,B), where R,G and B values range from 0 to 255.

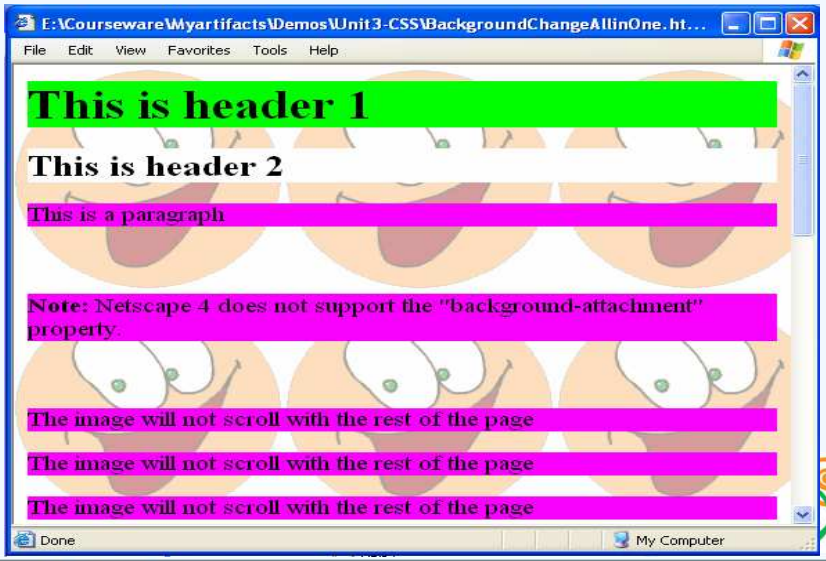
CSS supports the color property, which is used to set the text color.

e.g. BODY {color:blue }

H1 { color: #FF0088}

The Background properties allow you to control the background color of an element, set an image as the background, repeat a background image vertically or horizontally, and position an image on a page.

Background Properties -Example



NOTE: Try out the code given below in the browser.

```
<HTML><HEAD>
<STYLE>
body {
                    background: url (“smiley.jpg");
}
h1 {background-color: #00ff00;}
h2 {background-color: white;}
p {background-color: rgb(250,0,255);}
}
</STYLE></HEAD>
<BODY>
<H1>This is header 1</H1>
<H2>This is header 2</H2>
<P>This is a paragraph</P> <BR><BR> <BR>
<p><b>Note:</b> Netscape 4 does not support the "background-attachment" property.</p>
<BR><BR>
<P>The image will not scroll with the rest of the page</P><BR><BR>
<P>The image will not scroll with the rest of the page</P><BR><BR>
<P>The image will not scroll with the rest of the page</P><BR><BR>
<P>The image will not scroll with the rest of the page</P><BR><BR>
</BODY>
</HTML>
```

Font Properties

Properties	Values
Font-family	Arial, Monospace, ….
Font-style	Normal, italic, oblique
Font-variant	normal, small-caps
Font-size	x-small, small, medium, large
Font-weight	normal, bold, bolder, light, x-large

cssfont.html

CSS measurements

When you manipulate text and other objects with a style sheet, you often must specify a length or size. CSS supports measurements such as

1) inches (in) 2) centimeters (cm) 3) millimeters (mm)

4) point size (pt) 5) pixels (px)

Education and Research

CSS provides numerous font oriented properties to set the family, style, size and variations of the font used.

- font-family : Is used to set the font-family that is used to render text. This property may be set to specific font like Arial , to a generic family, such as sans serif or a coma delimited sequence of font family names.
- font-size : Is used to set the relative or physical size of the font used.
- font-style : is used to specify normal,italic or oblique font style for the font being used
- font-weight: Selects the weight or darkness of the font. Value ranges from 100 to 900, in increments of 100. Keywords are also supported

```
<html>
<head>
<style type="text/css">
                p.fontstyle {font-style:italic;}
                p.fontfamily {font-family:"comic sans ms", arial;}
                p.fontweight {font-weight:bold;}
                p.fontsize {font-size:xx-small;}
</style>
</head>
<body>
<p class="fontfamily">what type of font is this - comic sans or arial???</p>
<p class="fontstyle">this is now italicized</p>
<p class="fontweight">the bold and the beautiful</p>
<p class="fontsize">small is beautiful</p>
</body>
</html>
```

Text Properties

Properties	Values
word-spacing	measurement (px/cm)
letter-spacing	measurement (px/cm)
text-decoration	None, underline, overline, line-through
vertical-align	top,text-bottom,super,sub
text-transform	none, capitalize,uppercase, lowercase
text-align	left, right, center, justify
text-indent	measurement

cssfont1.html

Text properties allow you to control the appearance of text. It is possible to change the color of a text, increase or decrease the space between characters in a text, align a text, decorate a text, indent the first line in a text, and more.

The word-spacing property defines an additional amount of space between words

The letter-spacing property defines an additional amount of space between characters.

The text-decoration property allows text to be decorated.

The vertical-align property may be used to alter the vertical positioning of an inline element.

The text-transform property allows text to be transformed by one of four properties.

The text-align property can be applied to block-level elements (P, H1, etc.) to give the alignment of the element's text.

The text-indent property can be applied to block-level elements, to define the amount of indentation that the first line of the element should receive.

```
<html><head>
<style>
h2{
                letter-spacing:25pt;
                font-size:100pt;
                text-align:center;                }
</style>
<head><body>
                <h2>Infosys</h2>
</body></html>
```

Margin and Box Properties

Properties	Values
margin-top	measurements (in terms px,cm, mm, in)
margin-right	
margin-bottom	
margin-left	
margin	measurement
border-width	thick, medium, thin
border-color	#rrggbb
Border-style	dotted, dashed, solid, groove, ridge, inset, outset
width	measurement
height	measurement
Float	right, left, none
border	border-width, border-style, border-color

cssborder.html

The Margin properties define the space around elements. It is possible to use negative values to overlap content. The top, right, bottom, and left margin can be changed independently using separate properties. A shorthand margin property can also be used to change all of the margins at once.

Eg. DIV { margin: 1em 2em 3em 4em }
/* top margin 1em, right margin 2em, bottom margin 3em, left margin 4em */

The Border properties allow you to specify the style, color and width of an block level element's border. In HTML we use tables to create borders around a text, but with the CSS Border properties we can create borders with nice effects, and it can be applied to any element.

The border property is a shorthand for setting the width, style, and color of an element's border.

Eg. A:active { border: thick double red }

```
<html>
<head><style>
h2{         text-align:center;
            border-width:thick;
            border-color:red;
            Border-style:solid;                    }
</style>
<head><body>
            <h2>Border Demo</h2>
</body></html>
```

Classification Properties

Properties	Values
cursor	auto, crosshair ,pointer
display	inline, block, list-item
float	left , right , none
list-style-type	disc, circle, square, decimal, lower-roman, upper-roman, lower-alpha, upper-alpha
list-style-image	url("*.gif")
list-style-position	Inside, outside

The Classification properties allow you to control how to display an element, set where an image will appear in another element, position an element relative to its normal position, position an element using an absolute value, and how to control the visibility of an element. Most often used properties are briefly described in the table below.

Property	Description	Values
clear	Sets the sides of an element where other floating elements are not allowed	left, right, both, none
cursor	Specifies the type of cursor to be displayed	url, auto, crosshair default, pointer, move
display	Sets how/if an element is displayed	none, inline, block list-item, run-in compact, marker
float	Sets where an image or a text will appear in another element	left, right, none
list-style-type	Sets the type of the list-item marker	none, disc, circle, square decimal, lower-roman upper-roman, lower-alpha, upper-alpha
list-style-image	Sets an image as the list-item marker	none, url
visibility	Sets if an element should be visible or invisible	visible, hidden, collapse

Position Properties

Properties	Values
Position	absolute, relative, static, fixed
top	measurement
bottom	measurement
left	Measurement
right	measurement
visibility	visible, hidden
Z-index	-1,1

csspos.html

Education and Research

CSS Positioning properties define the position of an element.

The Positioning properties allow you to specify the left, right, top, and bottom position of an element. It also allows you to set the shape of an element, place an element behind another, and to specify what should happen when an element's content is too big to fit in a specified area.

Position: Places an element in a static, relative, absolute or fixed position .

Z-index: Sets the stack order of an element.

visibility: The visibility property sets if an element should be visible or invisible.

Invisible elements takes up space on the page. Use the "display" property to create invisible elements that do not take up space.

This property is used with scripts to create Dynamic HTML.

Property	**Description**	**Values**
bottom	Sets how far the bottom edge of an element is above/below the bottom edge of the parent element Similarly we have left , right and top.	auto, *%, length*
Position	Places an element in a static, relative, absolute or fixed position	static, relative, absolute fixed
overflow	Sets what happens if the content of an element overflow its area	visible, hidden, scroll auto
vertical-align	Sets the vertical alignment of an element	baseline, sub, super, top, text-top, middle, bottom text-bottom, *length, %*
z-index	Sets the stack order of an element. Default z-index is 0. Element with smaller Z-index will be placed "behind" another element having larger z-index,	Auto, *number*

Summary

- What are Style Sheets, Its syntax and working
- Cascading Style sheets with HTML doc
- Some related tags - <SPAN> and <DIV>
- Various style sheet properties


```
<html>
<head><style>
h1{
                color:blue;
                position: absolute;
                top:0px;        }
h2{
                color:red;
                position: absolute;
                top:20px;       }
</style>
<head><body>
                <h1>Infy</h1><h2>Infy</h2>
</body></html>
```

Infosys®
POWERED BY INTELLECT
DRIVEN BY VALUES

HTML, CSS and JavaScript

Unit 4 – JavaScript

Education and Research

Unit Objectives

- To explain the necessity of Scripting
- To explain writing client side scripting using JavaScript.
- To discuss about the in-built and user defined objects.
- To explain event handling using JavaScript.

Unit Plan

- Why Scripting?
- Client side Scripting Vs Sever side scripting.
- Introduction to JavaScript .
- JavaScript variable, operators and programming constructs
- Functions and Objects.
- Standard Objects.
- Event Handling.

Introduction to Scripting

- Scripting Languages are mainly used to build the programming environment in HTML document
- Make Web pages dynamic and interactive.
- Some languages : VBScript, JavaScript, Jscript and ECMA Script
- Browser Includes Scripting Interpreter
- Choosing a Scripting Language
  - Browser compatibility
  - Programmer familiarity
- Scripts can be executed on client or the server.

HTML is used to create client-side interfaces, but cannot handle client-side activities to respond to the user.

HTML does not provide event-handling facilities.

Scripting provides static HTML pages the ability to make decisions and to perform operations.

The client side processing includes validation, animation and interactive features with input devices.

A script is a series of commands or instructions. Script commands can store a user's name in a variable, display the user's name in a page returned to the browser, or store the user's name in a database.

The browser interprets the script while parsing the HTML tags. The script could do very simple things like validations or some really complex things like changing the content of the HTML page. One of the most obvious examples could be - if a user leaves some mandatory fields blank, it does not make sense for the form to be submitted to the server. This could be trapped at the client itself.

Netscape Communications Corporation developed JavaScript.

Similar to JavaScript , Microsoft introduced JScript and VBScript.

Based on JavaScript and JScript, ECMAScript was introduced.

JavaScript is supported by all major browsers, like Netscape and Internet Explorer.

Script which is executing on a client (browser) known as client side script.

Script which is executing on a Server known as server side script.

Most of the scripts can be used as both client side and server side.

Client Vs. Server Scripting

Client Side Scripting	Server Side Scripting
Runs on the user's computer i.e. Browser interprets the script	Runs on the Web server
Visible to users if they view the HTML source	Not visible to user
Script is client side by default `<script language="scriptname">`	Need to include RUNAT parameter RUNAT ="server" `<script language="scriptname" RUNAT="server">`

Client Side scripting means the script executes on the client side, i.e., the browser interprets the script embedded in HTML and executes it. This means some of the application's functionality executes in the browser on the user's workstation. Using client-side Script yields two main benefits:

· First, some application processing tasks are moved off the Web server, allowing the application, as a whole, to run faster and support more users.

· Second, client-side events can provide immediate feedback to the user without constantly posting data to the server and waiting for a reply. The result is a more responsive application.

The Client Side Scripting has its own disadvantages too.

Since the browser interprets the script, the scripting language should be selected with care. Some browsers understand a specific scripting language while others do not. Hence, your HTML page with client side scripting might work well on one browser but not on another. For example, if we use VB Script for client side scripting, it is fully supported by Internet explorer, but not on Netscape Navigator.

Client Vs. Server Scripting

Client Side Scripting	Server Side Scripting
Reasons to use: Create control event procedures	Reasons to use: Access server resources such as databases and hide business logic implementation
Dynamic related functions.	Dynamic creation of pages using databases, files etc.
Data is already available at client-side	Functions processed on server completely
Subsequent processing has to be dynamically performed. e.g: Form Validation.	Results to the client. e.g. Sorted data in combo

Server side scripting, means including script as part of an HTML file that executes on the server. When a user accesses a page containing server side script, the script executes on the server and dynamically generates HTML content or page, which is sent to the browser.

By default any script in an HTML file is considered to be client side script. To explicitly state that the script has to execute on the server side, we need to include the RUNAT parameter in the script tag as follows:

<SCRIPT LANGUAGE=”VBScript” RUNAT=”server”>

History of JavaScript

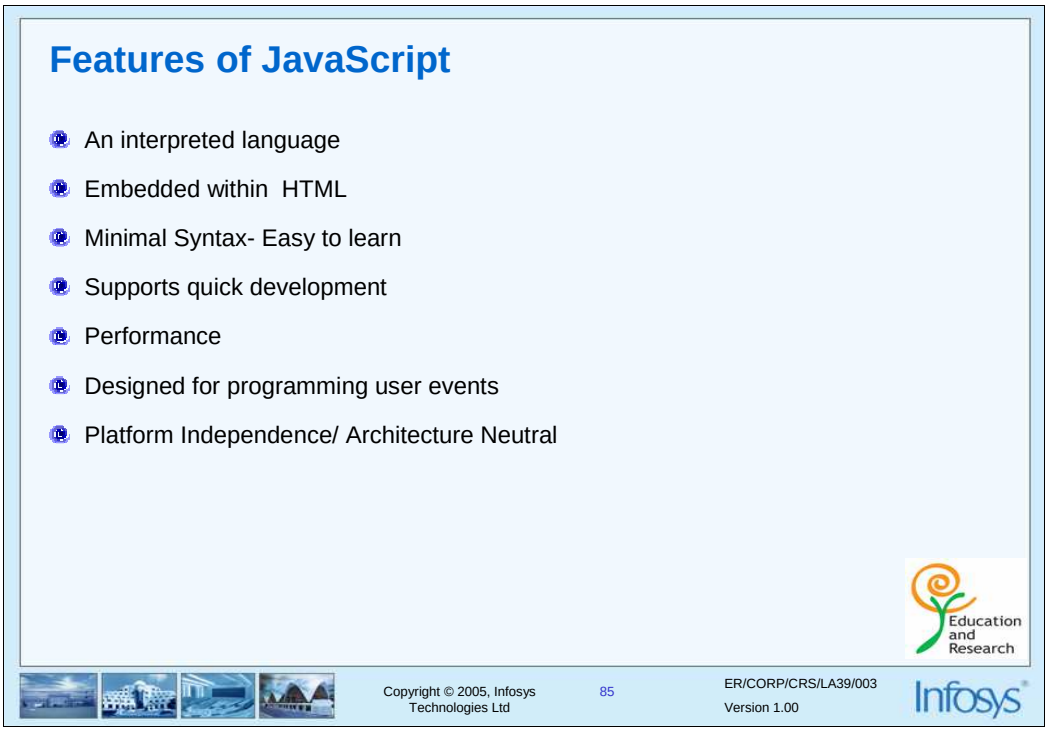
- Netscape integrated Java into its Web Browser called Netscape Navigator.
- In 1995, Netscape Communications introduced 'LiveScript', a Web Scripting Language, to simplify Java Programming on Web pages.
- Support for LiveScript began in June, 1995, with the release of Beta Version 2.0b1 of Netscape Navigator.
- Later in 1995, after an agreement with Sun, LiveScript was re-named JavaScript, to leverage the popularity of Java.
- To this date, JavaScript continues to evolve...

JavaScript was born at Netscape as "LiveScript" - a scripting language that could be used both on the client and server side. It was then rechristened "JavaScript" with the release of Netscape Navigator 2.0. Subsequent releases of the browser have resulted in more advanced versions of JavaScript.

JavaScript is Netscape's cross-platform, object-oriented scripting language. JavaScript is a small, lightweight language; it is not useful as a standalone language, but is designed for easy embedding in other products and applications, such as web browsers.

A JavaScript is usually embedded directly in HTML pages

Javascript is the most popular client side scripting language on the internet and supported by all major browsers .



An interpreted language

JavaScript code does not require compilation. The syntax is completely interpreted by the browser just as it interprets HTML tags.

Embedded within HTML

JavaScript does not require special editor for programs to be written,edited or compiled. It can be written in any text editor like notepad, textpad or editplus along with the HTML tag.

Minimal Syntax- Easy to learn

Just by learning few commands and simple syntax rules a complete JavaScript application can be developed.

Supports quick development

Since JavaScript does not require time-consuming compilation.

Performance

JavaScript can be written such that the HTML file are fairly compact and quite small. This minimizes storage requirements on the web server and download time for the client.

Designed for programming user events

Supports Object/Event based programming.

Platform Independence/ Architecture Neutral

JavaScript is a programming language that is completely independent of the hardware.

It is understood by any JavaScript enabled browser.

JavaScript program developed on a Unix machine will work perfectly well on a windows machine.

Java Vs JavaScript

JavaScript	Java
Interpreted(not Compiled) by Client Code Integrated with, and Embedded in HTML	Compiled Bytecodes downloaded from Server, Executed on Client
Source code visible to user	Applet code not visible
Variable Data Types not declared (Dynamic binding)	Uses strict data types Variable Data Types must be declared (Static binding)
Object-based	Object Oriented :Class-based

One of the most common misconceptions about JavaScript is that it is a simplified version of Java. Other than some what syntactical resemblance and the fact that both provide executable contents over the web, the two languages are entirely unrelated.

• JavaScript and Java make a good team. These two languages have disjoint set of capabilities. JavaScript can control browser behavior and content but cannot draw graphics or perform networking which java can do.

• JavaScript is Object based : No Distinction between types of Objects. Inheritance is through the Prototype mechanism, and properties and methods can be added to any object

•Java is Object Oriented - Class-based. Objects are divided into classes and instances; with all inheritance through the Class Hierarchy. Classes and Instances cannot have properties or methods added Dynamically.

Embedding JavaScript into HTML page

- <SCRIPT>.....</SCRIPT> tag

```
<SCRIPT LANGUAGE="JavaScript">
    -----
      (JavaScript code goes here)
    ----
</SCRIPT>
```

- LANGUAGE - the scripting language used for writing scripts

The <*SCRIPT*> tag is what allows JavaScript to be included in the HTML page. It looks something like this –

```
<SCRIPT LANGUAGE="JavaScript">
        (JavaScript code goes here)
</SCRIPT>
```

•LANUAGE attribute indicates the scripting language used for writing the snippet of the scripting code.

•If you see the LANGUAGE attribute, there is a version number attached to it - this is recommended to ensure that there are no unpleasant surprises with browsers not taken into account. The browser will interpret the script with whatever version it has and this might lead to unexpected results! This could be either a syntax error or a totally different output.

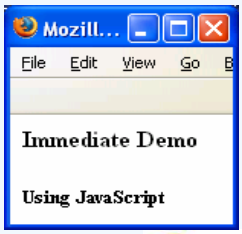
•The SCRIPT tag can be either in the HEAD tag or the BODY tag. The former is better in order to separate it from the actual HTML tags. Of course, you also need to take into account the nature of the code you want to execute.

•You can use more than one <script> tag in your document.

Deferred and Immediate Script

- SCRIPT tag can be placed in HEAD or BODY tag
- Mode of execution is decided by where is the SCRIPT tag placed.
- Immediate mode
  - SCRIPT tag is placed in the BODY tag or the HEAD tag
  - Scripts gets executed as the page loads.

```
<body>
<h4> Immediate Demo</h4>
<script language="JavaScript">
document.write("<h5> Using JavaScript</h5>");
</script>
</body>
```



Scripts in a page will be executed immediately while the page loads into the browser. This is not always what we want. Sometimes we want to execute a script when a page loads, other times when a user triggers an event.

Immediate mode implies that the code gets executed as the page loads.

This can be done by keeping the SCRIPT tags in the BODY or by trapping the onload event (events will be dealt with in later chapters).

Everything between the <SCRIPT> and the </SCRIPT> tag is interpreted as JavaScript code. document.write displays the message. Where document is an DOM object and write is a method. This will be covered in details in the next unit.

Deferred and Immediate Script

- **Deferred mode**
  - Script is executed based on some user action
  - SCRIPT tag is placed in the HEAD tag

```
<script language="JavaScript">
<!--
/*comments :  calling function when user    clicks on the
  button */
  function msg(){
   alert("Hi");
  }
// -->
</script>
<form name="f1">
  <input type="button" value=" ok " onClick="msg()">
</form>
```

jsdef.html

Education and Research

Deferred mode indicates that the script is executed based on some user action e.g. the clicking of the mouse or the pressing of a button. It is preferred that all the code that needs to be executed in this fashion be kept in the HEAD tag. Consider

```
<html><head>
<script language="JavaScript">
<!--
                    // Comments :  calling function when user clicks
                    function msg()
                    {
                                        alert("Hi");
                    }
//-->
</script>
</head><body><h3>Deferred  Script Demo</h3>
<form name="f1"><input type="button" value=" ok " onClick="msg()">   </form>
</body></html>
```

<!-- , // --> is used to “comment out” or hide the text of the script. Because the browsers which dose not support Javascript

will display the Javascript code as it is if you are not using HTML comments.

Remember that HTML interpreter will interpret HTML and Javascript interpreter interpret Javascript.

JavaScript – lexical structure

- JavaScript is object based and action-oriented.
- JavaScript is case sensitive.
- A semicolon ends a JavaScript statement
- C-based language developed by Netscape
- Comments
  - Supports single line comments using //
  - and multi line comments using /*…..*/

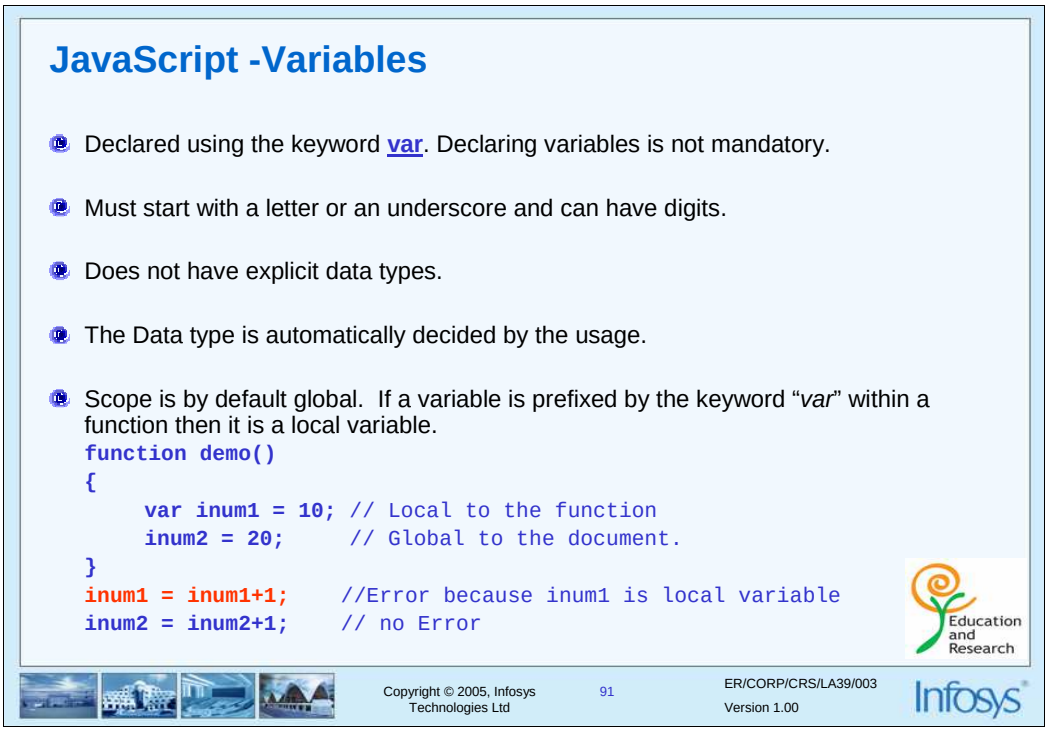
The lexical structure of the programming language is the set of elementary rules that specify how to write programs in the language.

•JavaScript is based on an action-oriented model of the World Wide Web. Elements of a Web page, such as a button or checkbox, may trigger actions or *events*. When one of these events occurs, a corresponding piece of JavaScript code, usually a JavaScript function, is executed. That function, in turn, is composed of various statements which perform calculations, examine or modify the contents of the Web page, or perform other tasks in order to respond in some way to that event. For example, pressing the SUBMIT button on an online order form might invoke a JavaScript function that validates the contents of that form to ensure that the user entered all the required information.

•JavaScript is case sensitive .this means that the language keywords, variables , function names and other identifiers must always be typed with a consistent capitalization of letters.

•Optional semicolon : Just as in C,C++ or Java simple statements in JavaScript are followed by a semicolon(;). This servers as a statement separator, you are allowed to omit this semicolon if your statements are placed on separate line.

•Comments : Any text between // and end of the line is treated as a comment. Also, any text between /* and */ is treated as a comment.



All JavaScript variables are declared using the keyword var. For example,

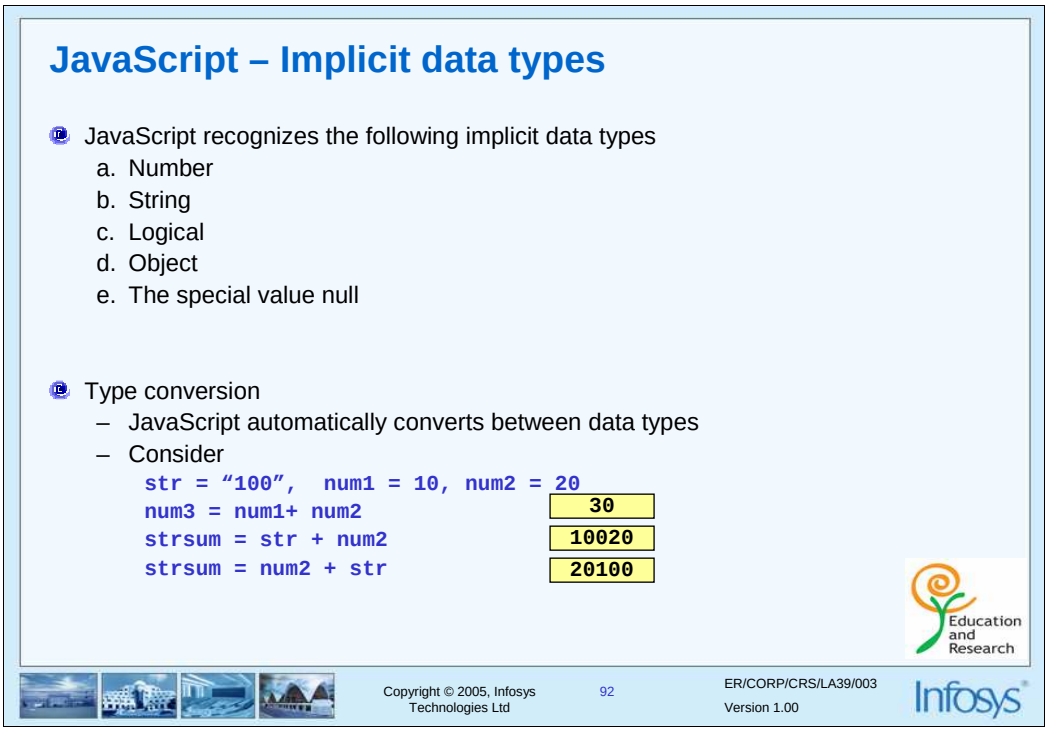
var i_Age = 27
var y,z = "20"
var s_Name = "lucky"
i_Num ="5" // variable can be declared just by assigning a value

Notice that there is no variable type that can be declared. The type is automatically decided by the usage.

One of the main differences between JavaScript and most other languages is that it does not have explicit data types. There is no way to specify that a particular variable represents an integer, a string, or a floating-point (real) number. Any JavaScript variable can be any of these-in fact, the same variable can be interpreted differently in different contexts.

One rule about variable names: a valid JavaScript variable name must begin with a letter or with the underscore character (_). Case is important, so that norl, NoRl, NORL, and _NORL are all valid JavaScript variable names that refer to different variables.

NOTE :It is advisable that variables be named appropriately with the type of variable as prefix e.g. *i_Interest*, *s_UserName* etc. One thing to remember is that JavaScript is case-sensitive.



There are five major implicit data types in JavaScript. A JavaScript value may be as follows:

- A number, such as -5, 0, or 3.3333
- One of the logical values true or false
- A string, such as "Click Here" or "JavaScript"
- The special value null
- A "non-atomic" JavaScript element, such as a function or object

The value of each variable depends on the context in which it is used. This context is related to the order in which the variables are seen. As you might guess, the expressions

e.g: *i_Age + z* // will result in 37

s_Name+ i_Age // will result in “lucky27”

Here a string is added to a number that results in a string “lucky27” .

- JavaScript often attempts to treat all variables within a statement as if they had the same type as the first variable in the statement.This is known as *Type casting* or *Type conversion.*

What would be the value of an expression such as

x = z + y where y is uninitialized

It may result in something seemingly innocent, such as x being assigned the value of z, as if y were zero. It may also result in something much more serious, such as the value of x becoming something strange, or, more likely, a JavaScript error occurring.

Initialize all JavaScript variables to meaningful default values. If a variable has no meaningful default, initialize it to null.

JavaScript – Operators

- Arithmetic Operators

  `+, ++, -, --, *, /, %`

- Relational Operators

  `==, !=, ===, !==, >, >=, < , <=`

- Logical Operators (and , or , not)

  `&&, ||, !`

- Assignment Operators

  `=, +=, -=, *=, /=, %=`

- Strict equal (===)

  Returns true if the operands are equal and of the same type.

- Strict not equal (!==)

  Returns true if the operands are not equal and/or not of the same type.

strict.html

The standard equality operators (== and !=) compare two operands without regard to their type. The strict equality operators (=== and !==) perform equality comparisons on operands of the same type. Use strict equality operators if the operands must be of a specific type as well as value or if the exact type of the operands is important.

The “+” operator can use for concatenation (String addition) of two strings.

```
Fullname=“Jack”+”Den”;
```

```
<script language="JavaScript">
        num1=10;        str1="10";
        document.write("<h3>num1 is "+num1+"  and str1 is "+ str1+"</h3>")
        document.write("<h3>Using Equals(==)</h3>")
        if(num1==str1)

        document.write("<h4>num1 equal to str1</h4>");
        else

        document.write("<h4>num1 not equal to str1</h4>");
        document.write("<h3>Using Equals(===)</h3>")
        if(num1===str1)

        document.write("<h4>num1 equal to str1</h4>");
        else

        document.write("<h4>num1 not equal to str1</h4>");
</script>
```

Special operators

- Conditional operator (?:)
  - Ternary operator.
  - Syntax :
    - `(expression1) ? (expression2) : ( expression3)`
  - Eg
    - `ans=((10%2)==0)?”even”:”odd”;`
- typeof operator
  - Unary operator
  - Indicates the data type of the operand.
  - Eg

```
x=123;
  alert(typeof(x));          // Number
x="Hello"
  alert(typeof(x));          //  String
```


Conditional Operator (? :)

This is a conditional operator that takes three operands and is used to replace simple if statements. The first operand is a condition that evaluates to true or false, the second is an expression of any type to be returned if the condition is true, and third is an expression of any type to be returned if the condition is false. The following code displays one of two messages depending on the value of the object property 'percent_proof':

Example:

document.write("This milk is " + ((percent_proof < 5) ? "Adulterated Milk" : "Pure Milk"));

typeof operator

The typeof operator returns the type of an unevaluated operand, which can be a number, string, variable, object or keyword. It can be used with or without brackets as in the following examples of a numeric literal and the variable 'age' being 60:

Example:

typeof(age) returns number
typeof 33 returns number

The following values are returned for various types of data:

a number returns 'number';
a string returns 'string';
the keyword true returns 'boolean';
the keyword null returns 'object';
methods, functions and predefined objects return 'function';
a variable returns the type of the data assigned to it;
a property returns the type of its value;

Special Operators

- new
  - Used for instantiation of objects.
  - Eg: `today = new Date( )`
- this
  - used to refer to the current object
- with
  - Used to group several entries that are associated with a single object.
  - Syntax

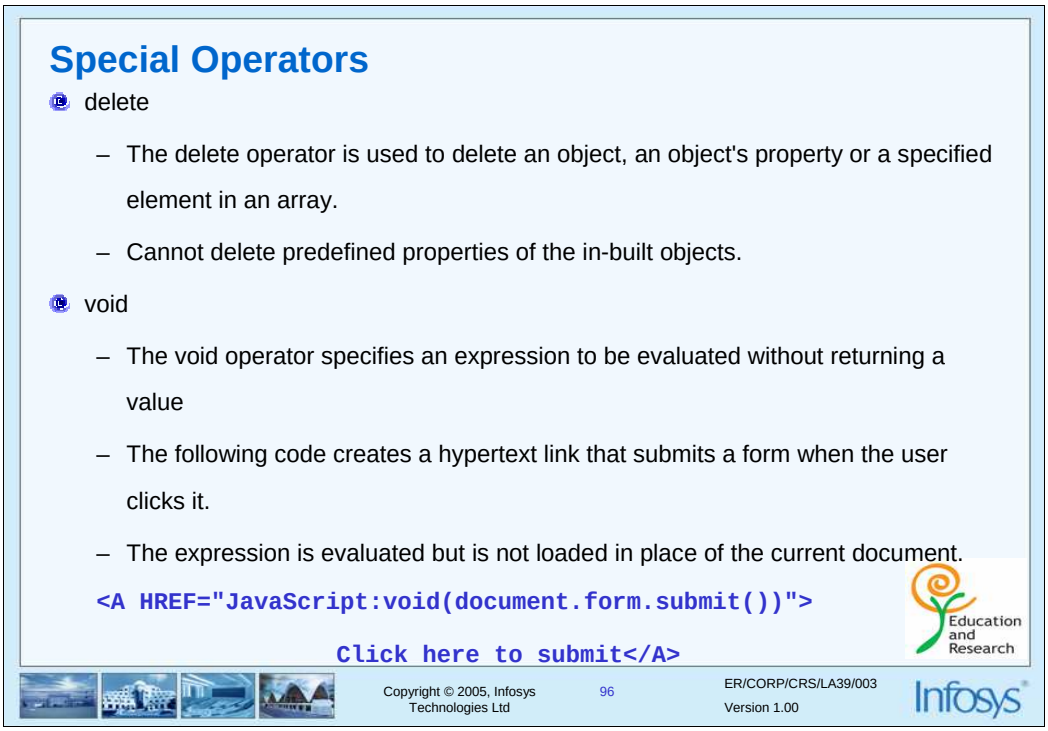
```
with(object)
{
script statements all referred to specified object
}
```

new : The new operator can be used to create an instance of a user-defined object type or of one of the built-in object types that has a constructor function. To create a user-defined object type you must first define it by writing a function that specifies its name, properties and methods. For example, the following function creates an object for book with properties for title, category and author:

```
mybook = new book ("The Thing", "horror", "John Lynch");
                                        // where book is a user defined object
mydate = new Date( ) ;                  // where Date is a built-in object
```

with: Convenient means for grouping several related JavaScript entries that are associated with a particular object.

```
with (mybook)
{
        Name=“ The Thing“;
        Type= "horror“;
        Author=  "John Lynch “
}
```



delete

The delete operator deletes an object, an object's property, or an element at a specified index in an array. Its syntax is:

delete objectName, delete objectName. property
delete objectName[index], delete property // legal only within a with statement

where objectName is the name of an object, property is an existing property, and index is an integer representing the location of an element in an array.

The fourth form is legal only within a with statement, to delete a property from an object.

You can use the delete operator to delete variables declared implicitly but not those declared with the var statement.

If the delete operator succeeds, it sets the property or element to undefined. The delete operator returns true if the operation is possible; it returns false if the operation is not possible.

x=42
var y= 43
myobj=new Number()
myobj.h=4 // create property h
delete x // returns true (can delete if declared implicitly)
delete y // returns false (cannot delete if declared with var)
delete Math.PI // returns false (cannot delete predefined properties)
delete myobj.h // returns true (can delete user-defined properties)
delete myobj // returns true (can delete user-defined object)

void

The void operator specifies an expression to be evaluated without returning a value.

You can use the void operator to specify an expression as a hypertext link.

JavaScript – Control structures

- Control structure in JavaScript, as follows:
  - if
    - Is used to conditionally execute a single block of code
  - if .. else
    - a block of code is executed if the test condition evaluates to a boolean true

      else another block of code is executed.
  - switch …. case
    - switch statement tests an expression against a number of case options
    - executes the statements associated with the first match.

control.html

The if...else statement executes one set of statements if a specified condition is true, and another if it is false, Consider

```
<script language="JavaScript">
        num=11;
        document.write("<h3> To find  "+num+" is even or odd</h3>");
        if(num%2==0){
                document.write("<h3>"+num+" is even</h3>");}
        else{
                document.write("</h3>"+num+" is odd</h3>");}
</script>
```

The switch statement tests an expression against a number of case options and executes the statements associated with the first one to match. If no match is found, the program looks for a set of default statements to execute, and if these aren't found either, it carries on with the statement immediately following switch. An optional break statement with each case ensures that once a set of statements has been executed, the program exits switch. If break were omitted, the statements associated with the following case would also be executed

Ex:
```
switch (i)
{ case "Bangalore" :
    document.write ("Flights to Bangalore: Saturdays.");
    break;
  case "Mumbai" :
    document.write ("Flights to Mumbai: Fridays.");
   break;
  default :
    document.write ("Sorry, there are no flights to " + i +          ".<BR>");
}
  document.write("Thank you for enquiring with Indian  Airlines.<BR>");
```

JavaScript – Loop

- while loop
  - The while statement is used to execute a block of code while a certain condition is true
  - Syntax :
```
while ( test condition)
{
  zero or more statements
}
```
- for loop
  - Iterate through a block of statements for some particular range of values
  - Syntax :
```
for(initstmt; condstmt; updstmt ){
        zero or more statements
}
```
- do while loop
  - block of statements is executed first and then condition is checked

Syntax :
```
do
{
  zero or more statements
}while ( test condition)
```

loop.html

Infosys

```
<script language="JavaScript">
        /*    Using for Loop    */
        for (i=10;i<=100;i=i+10)
        {
                document.write("<hr color=red size=3 width="+i+"%>");
        }
        for(i=100;i>=10;i=i-10)
        {
                document.write("<hr color=red size=3 width="+i+"%>");
        }

document.write("<form name='f1'><select name='s1'>");
year=2000;
        /*          Using While Loop          */
        while(year<2050)
        {
                document.write("<option  value="+year+">"+year+"</option>");
                year++;
        }
document.write("</select></form>");
</script>
```

JavaScript – Control structures

- break
  - Terminates the current loop, switch, or label statement and transfers program control to the statement following the terminated loop.
- continue
  - In contrast to the break statement, continue does not terminate the execution of the loop entirely.
  - In a for loop, it jumps to the update expression.
  - In a while loop, it jumps back to the condition.

```
for(i=0; i<5; i++)
{
    inner :     /*JS Comments : label */
    for(j=0;j<5;j++)
    {
           if(i==j){ break inner;}
           document.write(j+"  ");
    }
    document.write("<br>");
}
```

Education and Research

break: Aborts execution of the loop, drops out of loop to the next statement following the loop.

Ex:

```
var i = 0
while (i < 10)
{
   document.write(i);
   if (i==7)
   {
      document.write("the counter has reached " + i);
       break;
   }
   i++;
}
```

Will terminate the loop when the value of i is 7.

continue : Aborts this single iteration of the loop, returns execution to the loop control, meaning the condition specified by the loop statement. Loop may execute again if condition is still true.

A *continue* Statement Returns to the Top of a *while*

```
var x = 0; var xsum = 0;
var loopcount = 0;
while ( loopcount++ < 100 )
{                                   // 1: loop 100 times
x++;                                // 2: increment x
if ( ( x % 5 ) == 0 )               // 3: if x is divisible by 5
continue;                           // 4: skip it
xsum += x;                          //5:add x to xsum
}
```

Functions

- A function is a block of code that has a name.
- Way for bundling several commands together
- Tool to organize your code. User can write his own functions
- JavaScript functions is to link actions on a web page with the JavaScript code.
- JavaScript has some in-built functions.

To create a function you define its name, any values ("arguments"), and some statements:

```
function myfunction(argument1,argument2,etc){
    some statements;
}
```

function.html

Infosys

Functions serve two purposes. A function is an organizational tool, in the sense that it permits you to perform the same operation without simply copying the same code.

The second purpose of JavaScript functions is to link actions on a Web page with JavaScript code. Mouse clicks, button presses, text selections, and other user actions can call JavaScript functions by including suitable tags in the HTML source for the page.

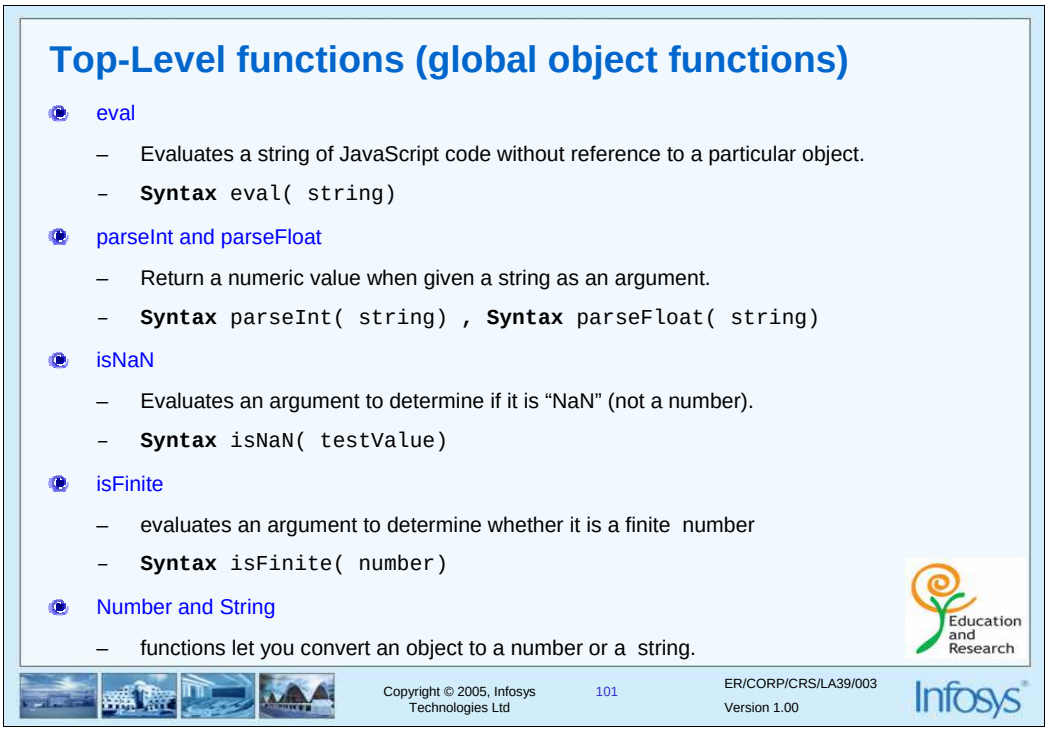
Functions may also be called with values, known as *parameters*, which may be used inside the body of the function.

This list may be empty, indicating that the function does not use any arguments (often called a *void function*)

The *return* statement returns a value to where it is called from

Eg:

```
<script language="JavaScript">
function addFun(num1,num2)
{
                                        alert("the sum of  "+num1+" "+num2+" =
"+(num1+num2)); // alert is a window function to display a message box
}
function subFun(num1,num2)
{
                    alert("the diffrence of  "+num1+" "+num2+" = "+(num1-num2));
}
</script>
```



JavaScript provides several built-in functions that can be used to perform explicit type conversion. Some of them are listed below

eval( ) - This function evaluates a string of JavaScript code without reference to a particular object

e.g.: *var ivalue = eval(“ 10*5+5”)*

ivalue will be assigned the value 55 after the execution of this statement.

parseInt() – Converts a string value to an integer. parseInt() returns the first integer contained in a string . If the string does not begin with an integer, NaN (not a number) is returned.

e.g.: *var inum=parseInt(“ 1abc”)* // returns 1
var inum= parseInt(“abc”) // returns NaN

parseFloat( ) - The parseFloat method returns the first floating point number contained in string passed. If the srint does not begin with a floating point number, NaN (not a number) is returned.

e.g: *var fnum=parseFloat(“3.14abc”)* // returns 3.14
var fnum= parseFloat(“abc”) // returns NaN

isNaN( ) - Returns true if the argument passed is not a number or false otherwise.

The *isFinite* function evaluates an argument to determine whether it is a finite number.

The syntax of isFinite is:

isFinite(number), where *number* is the number to evaluate.

If the argument is NaN, positive infinity or negative infinity, this method returns false, otherwise it returns true.

Number and String

The *Number and String* functions let you convert an object to a number or a string. The syntax of these functions is:

Number(objRef) OR String(objRef)

where objRef is an object reference.

The following example converts the Date object to a readable string.

D = new Date (430054663215) // The following returns

x = String(D) // "Thu Aug 18 04:37:43 GMT-0700 (Pacific Daylight Time) 1983"

In-built properties (properties of the global object)

- Infinity
  - Infinity is a numeric value representing infinity
- NaN
  - NaN is a value representing Not-ANumber.
- undefined
  - undefined is the value undefined.

Dialog boxes (Window Object methods)

- Alert dialog box - *alert(message)*
  - Takes in a string argument and displays an alert box.
- Prompt dialog box – *prompt(message[, inputDefault])*
  - Displays a message and a data entry field

alert.html

- Confirm dialog box *– confirm(message)*
  - Serves as a technique for confirming user actions

There are three different types of dialog boxes which are the methods of window object.

All these three dialog boxes are modal i.e. the user must close it before continuing.

alert ( ) - Displays a dialog box with the message string passed to the alert() method and an OK button.

Syntax: *alert (" message ")*

e.g.: *alert("Click OK to continue")*

prompt ( ) - Displays a Prompt dialog box with a message and an input field. Returns the value the user types in.

Syntax: *prompt("message", "default value")*

e.g.: *prompt ("Enter your age", 25)*

confirm() - The confirm dialog box lets you ask the user a "yes-or-no" question, and gives the user the option of clicking either

an OK button or a Cancel button. The confirm method returns either true or false.

Syntax: *confirm ("message")*

e.g.: *confirm ("Click OK to continue; Click cancel to exit")*

Objects

- JavaScript is based on object-based paradigm.
- Object is a construct with *properties* and *methods*.
- Any object should be instantiated before being used.
- Once instantiated, the properties and methods of the object can be used accordingly.
- HTML elements become objects in JavaScript and Attributes become properties in JavaScript.

An *object* is a construct with *properties* that are JavaScript variables or other objects.

An object also has functions associated with it that are known as the object's *methods*.

JavaScript provides support for both standard and user-defined objects. The following are the features provided by JavaScript for objects:

Creating object types

Instantiating objects of both standard and user-defined object types

You can define the composition of the object in terms of other information.

You can add and delete properties from an existing object.

Some amount of polymorphism - the arguments array can be used for function definition.

Creating objects

- Using keyword **new**
- Any Object should be instantiated before being used.
- Once instantiated, the properties and methods of the object can be used accordingly.
- Example

```
var newDateObj = new Date();
alert("Today Date is :" + newDateObj.getDate());
alert("And the Month is :" +  newDateObj.getMonth());
```

- Where Date is an inbuilt object template.

You can use the new operator to create an instance of a user-defined object type or of one of the predefined object types Array, Boolean, Date, Function, Image, Number, Object, Option, RegExp, or String.

Note : User-defined objects are not in scope of this course and will not be covered.

Use new as follows:

objectName = new objectType (param1 [, param2] ...[, paramN])

To instantiate a standard object type

var dtToday = new Date();

where *Date()* is an inbuilt object.

Date Object

- The Date object is used to work with dates and times.
- Constructors
  - new Date(); -returns the current system date
  - new Date(milliseconds) – 1 January 1970 00:00:00.
  - new Date(yr_num, mo_num, day_num[, hr_num, min_num, sec_num, ms_num])

getDate() / setDate()	toString()
getDay()	getFullYear()
getHours()	getMilliseconds()
getMinutes()	getUTCHours()
getMonth()	getUTCMilliseconds()
getYear()	
getSeconds()	
getTime()	
toLocaleString()	

dateobj.html

The standard objects include *Date*, *Math*, *String* etc. apart from those included in the DOM. The DOM objects we will talk about in successive slides

The Date object is needed because there is no corresponding data type in JavaScript. It provides a set of methods commonly needed for working with dates. These include

getMinutes() : to get the minutes from the Date object

getDay() : returns the day of the week i.e. value 0 (Sunday) to 6 (Saturday)

setTime() : sets the time of the date object

getTime() : returns the number of milliseconds since January 1, 1970,

Eg:

```
function JSClock() {
        var time = new Date()     // object will contain todays date
        var hour = time.getHours()
        var minute = time.getMinutes()
        var second = time.getSeconds()
         var temp = "" + ((hour > 12) ? hour - 12 : hour)
        temp += ((minute < 10) ? ":0" : ":") + minute
        temp += ((second < 10) ? ":0" : ":") + second
        temp += (hour >= 12) ? " P.M." : " A.M."
        return temp
        }
```

Math Object

- The built-in Math object includes mathematical constants and functions
- You do not need to create the Math object before using it.
- Methods/ properties
  - Math.abs(number)
  - Math.sin(number)
  - Math.exp(number)
  - Math.log(number)
  - Math.max(num1,num2)
  - Math.sqrt(number)
  - Math.random()
  - Math.round(x)
  - Math.pow(x,y)
  - Math.ceil(x)
  - Math.floor(x)
  - Math.PI - The ratio of the circumference of a circle to its diameter.
  - Math.E - Euler's constant and the base of natural logarithms.

The Math object has properties and methods for mathematical constants and functions.

For example, the Math object's PI property has the value of pi (3.141...), which you would use in an application as

Math.PI

Similarly, standard mathematical functions are methods of Math. These include trigonometric, logarithmic, exponential, and other functions. For example, if you want to use the trigonometric function sine, you would write

Math.sin(1.56)

Note that all trigonometric methods of Math take arguments in radians

To store a random number between 0 and 1 in a variable called "r_number":

r_number=Math.random().

String Object

- Wrapper around the String datatype.
- This object provides useful methods to manipulate strings
  - s1 = “infy" //creates a string literal value
  - s2 = new String(“infy") //creates a String object
  - The length property specifies the number of characters in the string.
- A String object has two types of methods:
  - Methods that return a variation on the string itself, such as substring and toUpperCase,
  - Those that return an HTML-formatted version of the string, such as bold and link.

string.html

The String object is a wrapper around the string primitive data type. Do not confuse a string literal with the String object. For example, the following code creates the string literal s1 and also the String object s2:

s1 = "foo" //creates a string literal value

s2 = new String("foo") //creates a String object

You can call any of the methods of the String object on a string literal value—JavaScript automatically converts the string literal to a temporary String object, calls the method, then discards the temporary String object.

You can also use the String.length property with a string literal.

You should use string literals unless you specifically need to use a String object, because String objects can have counterintuitive behavior. For Eg:

s1 = "2 + 2" //creates a string literal value

s2 = new String("2 + 2") //creates a String object

eval(s1) //returns the number 4

eval(s2) //returns the string "2 + 2"

A String object has one property, length, that indicates the number of characters in the string. For example, the following code

myString = "Hello, World!"

x = mystring.length //assigns x the value 13, because “Hello, World!” has 13 characters:

A String object has two types of methods: those that return a variation on the string itself, such as substring and toUpperCase, and those that return anHTML-formatted version of the string, such as bold and link.

e.g: *mystring.toUpperCase( )* //return the string “HELLO, WORLD!”.

mystring.link(“http://www.helloworld.com”) // create a hyperlink

String Object

Methods	Description
charAt(index)	Return the character or character code at the specified position in string.
indexOf(searchValue[, fromIndex])	Return the position of specified substring in the string or last position of specified substring respectively
lastIndexOf(searchValue[, fromIndex])	Returns the index within the calling String object of the last occurrence of the specified value, or -1 if not found.
concat(string2, string3[, ..., stringN])	Combines the text of two strings and returns a new string.
split([separator][, limit])	Splits a String object into an array of strings by separating the string into substrings.
substr(start[, length])	Returns the characters in a string beginning at the specified location through the specified number of characters.
substring(indexA, indexB)	Return the specified subset of the string
toLowerCase(), toUpperCase()	Return the string in all lowercase or all uppercase, respectively

e.g : It illustrates usage of some string function .Try this code in the browser to check the output.

```
<html><head><title>Chapter 13 - String examples</title></head>
<body>
<H3> Changing the formatting with the <I>bold</I> method</H3>
<script language="JavaScript">
var str=  "I am a happy string!"
document.write ("The string is \"" + str + "After bolding, it becomes "+ str.bold())
</script>
<H3>Getting the length of a string with the <I>length</I> property</H3>
<script language="JavaScript">
var str= "I am a string!"
document.write ("The string is \"" + str +"And it\'s length is " + str.length)
</script>
<H3>Looking for a string within a string <I>indexOf</I> property</H3>
<script language="JavaScript">
var str = "Barking dogs seldom buy ties"
document.write("looking for the string 'dog' in the string '" + str + "'<P>")
var pos=str.indexOf("dog")
if (pos>=0){
 document.write("Result : dog found at position “+pos + "<br>")
}
else{   document.write("Result : dog not found")
}</script>
<H3>Getting a specified part of a string with the <I>substr</I> method</H3>
<script language="JavaScript">
var str = "A bard in Avon is worth two of George Bush"
document.write ("The string is '" + str + "' and <I>substr(2,4)</I> returns “+str.substr(2,4) + "'")
</script></body></html>
```

Arrays

- Arrays are objects in JavaScript.
- The Array object is used to store a set of values in a single variable name.
- You create an instance of the Array object with the "new" keyword.
- The length property specifies the number of elements in an array.

```
var family_names=new Array(3)

var family_names=new Array("Tove","Jani","Stale")
```

DENSE ARRAYS

Data can be of any type, and there can be mixed data types within the same array if it makes sense for your data requirements

```
var darr = [10, "Chicago", "Houston", "Port"]
```

JavaScript does not have an explicit array data type. However, you can use the predefined Array object and its methods to work with arrays in your applications. The Array object has methods for manipulating arrays in various ways, such as joining, reversing, and sorting them. It has a property for determining the array length and other properties for use with regular expressions.

Array index starts with 0.

Creating and populating the array

```
var arrBook = new Array ();      <!-- Array Declaration -->
arrBook[0] = "Oliver Twist";    <!—Array Initialization -->
arrBook[1] = "And then there were none!";
```

Even if an array is initially created of a fixed length it may still be extended by referencing elements that are outside the current size.

A dense array is an array that has been created with each of its element being assigned specific value. They are exactly the same manner as other arrays.

Increasing the array length indirectly. An array's length increases if you assign a value to an element higher than the current length

of the array. The following code creates an array of length 0, then assigns a value to element 99. This changes the length of the array to 100.

```
colors = new Array()
colors[99] = "midnightblue"
```

Arrays

Methods	Description
concat(arrayName2, arrayName3, ..., arrayNameN)	joins two arrays and returns a new array.
join(separator)	joins all elements of an array into a string.
pop()	Removes the last element from an array and returns that element.
push(element1, ..., elementN)	Add / removes the last element to/from an array and returns that element.
reverse()	transposes the elements of an array: the first array element becomes the last and the last becomes the first.
shift()	removes the first element from an array and returns that element
unshift(element1,..., elementN)	adds one or more elements to the front of an array and returns the new length of the array.


Array.html

Eg: Try the following code in the browser.

```
<html>
<body>
<script type="text/JavaScript">
                var arrname = new Array(3)
                arrname[0] = “Smith"
                arrname[1] = “John"
                arrname[2] = "Hernry"
                document.write(arrname.length + "<br>")
                document.write(arrname.join(".") + "<br>")
                document.write(arrname.reverse() + "<br>")
                document.write(arrname.push(“Suzie","Jon") + "<br>")
                document.write(arrname.pop() + "<br>")
                document.write(arrname.shift() + "<br>")
</script>
</body>
</html>
```

JavaScript as a referenced file

- 3 Ways of Using JavaScript in HTML
  - *External file*
  - *Embedded*
  - *Inline*
- External file
  - Enables a JavaScript script stored in a separate file to be included as part of the current page.
- The important use of the referenced file is
  - The creation of Code Libraries .
  - Enables a JavaScript script stored in a separate file to be included as part of the current page
  - Multiple pages can use the same .js file.

Sometimes you might want to run the same script on several pages, without writing the script on each and every page.

To simplify this you can write a script in an external file, and save it with a .js file extension.

The external script cannot contain the <script> tag

Now you can call this script, using the "src" attribute, from any of your pages:

```
<html>
<head> </head>
<body> <script src="myscript.js">
</script>
</body>
</html>
```

Remember to place the script exactly where you normally would write the script.

External and Inline JavaScript

- Can be achieved by using the SRC attribute.
- External Javascript file should have an extension `.js`
- Should not use <script> tag inside the .JS file

  For example:

```
<script language="JavaScript" src="external.js">
</script>
```

- Inline JavaScript
  - Scripts are included inside the HTML tag.
  - For example

```
<html>
<body onLoad="alert('document loaded')";>
<h1>Inline JavaScript</h1>
<input type="button" name="but1" value=" click " onClick="window.close()">
</body></html>
```

inline.html

Eg: Consider the following .JS file (ext.js)

```
function f2()
{
        x=prompt("enter your name","name");
}
```

Include the .JS file inside the HTML file

```
<html><head>
        <script language="JavaScript" src="ext.js">
</script></head>
<body>
        <input type=Button name=b1 value="Input"  onClick="f2()">
</body></html>
```

<NOSCRIPT> tag

- Used to inform those browsers that can't or won't process JavaScript.
- Old browsers won't support JavaScript. (Eg : IE 2.0)
- Set of instructions to be processed by those browsers that are not capable of executing JavaScript.

```
<script language="JavaScript">
<!--
  document.write("<h1> Supports JavaScript</h1>");
//  -->
</script>
<noscript>
  <h1> Do not support JavaScript</h1>
</noscript>
```

The <noscript> element is used to define an alternate content (text) if a script is NOT executed.

This tag is used for browsers that recognizes the <script> tag, but does not support the script in it.

If a browser supports scripting, it will not display the text in the <noscript> element.

The Client can disable JavaScript using the Browser options.

HTML event Handlers

- Events are actions that occur usually as a result of some action by the user.
- With an event handler you can do something with an element when an event occurs.
- Most frequently event handlers are

Event	Occurs when	Event Handler
Click	User Clicks	onClick
Blur	Removes Input Focus	onBlur
Change	Changes value	onChange
KeyDown	Depresses a Key	onKeyDown
KeyPress	Holds down a Key	onKeyPress
KeyUp	Releases a Key	onKeyUp
Load	Loads the page	onLoad
UnLoad	Unloads the page	onUnload
MouseDown	Depresses a Mouse Button	onMouseDown
MouseMove	Moves the mouse	onMouseMove
MouseOut	Mouse out	onMouseOut
MouseOver	Mouse Over	onMouseOver
MouseUp	Releases a mouse Button	onMouseUp
Submit	Submit a form	onSubmit

Events are actions that usually occur as a result of some action by a user.

For example, clicking a button is an event, as is changing a text field or moving the mouse over a link. Now what is the purpose of defining such events? These can be trapped and event handlers that will respond to the event can be written.

Programming Languages these days are often quoted as being Object-Orientated Event-Driven Languages. This means that the data and methods for the data are stored and realized in objects, and human-computer interaction is provided by events.

Event Object

The DOM contains many objects that store a variety of information about the browser, the computer running the browser, visited URLs, and so on. The Event Object stores data about events. Event handlers are small JavaScript code placed inside an HTML tag.

Summary

- Need of Scripting
- Client side Scripting Vs Sever side scripting.
- Introduction to JavaScript .
- JavaScript Basics
- Functions and Objects.
- Standard Objects such as Arrays, String, Math, Date.

Infosys®
POWERED BY INTELLECT
DRIVEN BY VALUES

HTML, CSS and JavaScript

Unit5 - Document Object Model

Education and Research

Course Objective

- To introduce the participants to the Document Object Model and various objects in it.
- To illustrate the usage of objects in the DOM.
- Event Handling.
- Form validation.

Unit Plan

- What is DOM and its use?
- Hierarchy of DOM objects.
- Reflection of these objects in an HTML page.
- Some of the important objects, their methods and properties
- Illustrate through examples the usage of the objects, methods and properties in JavaScript code.
- Form validation and advantages

Some of the important DOM objects, their methods and properties that will be discussed are.

- window object
- location object
- document object
- form object
- history object

What is DOM?

- DOM stands for the Document Object Model.
- DOM is a document content model for HTML documents.
- It is a hierarchy of pre-existing Objects
- A programmer can reference the object and their contents.
- The World Wide Web Consortium (W3C) is the body which sets standards for the web.
- W3C has defined a standard Document Object Model, known as the W3C DOM.
- DOM stands apart from JavaScript because other scripting languages could access it.

What is its use?

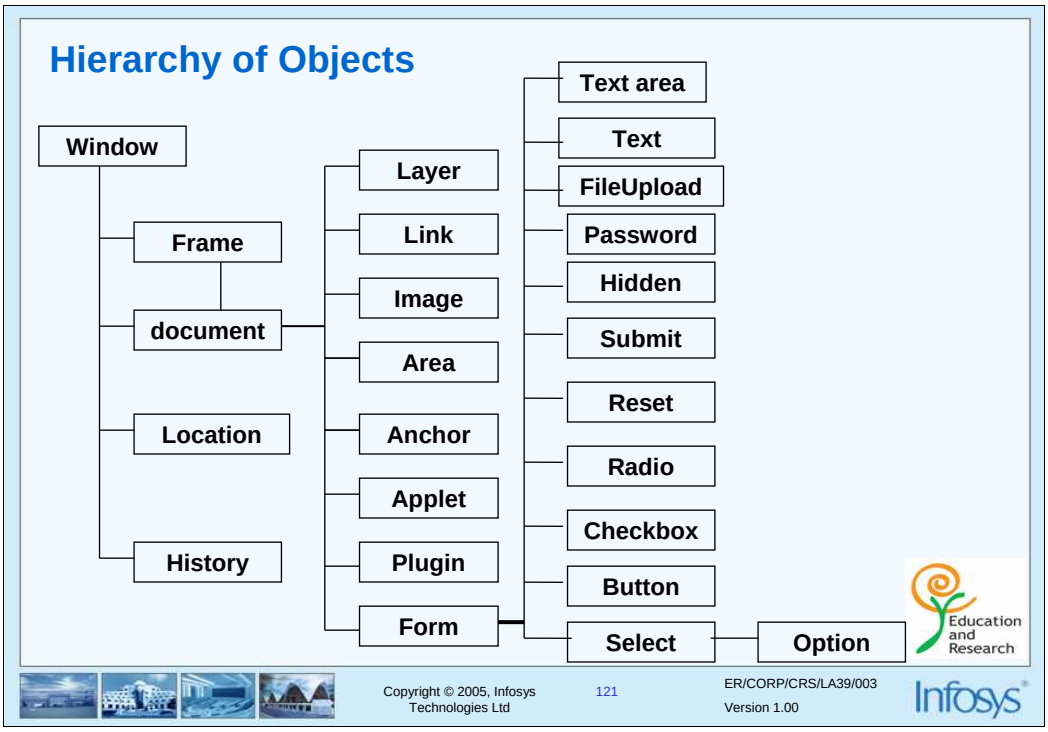
DOM means the Document Object Model this is a way of describing the HTML document to a scripting language, like JavaScript or VBScript, as objects that can be accessed and manipulated. Objects can be added or deleted from the page and the events triggered by these objects can be handled.

DOM has been around in different forms for quite some time now, but in its present version enables us to make web pages more dynamic.

Various object models

Both Netscape Navigator and Internet Explorer have their own DOMs. Of course, it is not a major show-stopper, but yes, as responsible web programmers, we need to keep this in mind especially when dealing with events and layers. Now that Netscape and Microsoft are both working within the W3C DOM working group, it is likely that these will converge in their features and usage.

The most important thing about objects is that you can reference them and their contents. For instance, JavaScript allows you to reference the title of a document as follows: document. title



The HTML objects, which belong to the DOM have a descending relationship with each other.

The window object corresponds to the browser window. A Window object is created automatically with every instance of a <body> or <frameset> tag.

The DOM hierarchy continues downwards to encompass individual elements on a 'FORM' such as text boxes, labels, radio buttons and so on,which belong to the form.

To refer to specific properties, you must specify the object name and all its ancestors. Generally, an object gets its name from the NAME attribute of the corresponding HTML tag.

For example, the following refers to the value property of a text field named

text1 in a form named myform in the current document:

document.myform.text1.value

When you load a document in Navigator, it creates a number of JavaScript objects with property values based on the HTML in the document and other pertinent information.

JavaScript's object hierarchy is mapped to the DOM, which in turn is mapped to the web page elements in the browser widow.

DOM Top Level objects

Object	Has	Description
window	methods properties	The window object is the top level object in the DOM. It contains information about the window and the frames
document	methods properties collections	The document object represents the HTML document, and is used to access the HTML elements inside the document
event	properties collections	The event object contains information about events that occurs
navigator	methods properties	Contains information about the version of Navigator in use.

Every page has the following objects:

- navigator: has properties for the name and version of the Navigator being used, for the MIME types supported by the client, and for the plug-ins

installed on the client.

- window: the top-level object; has properties that apply to the entire window. There is also a window object for each “child window” in a frames

document.

- document: contains properties based on the content of the document, such as title, background color, links, and forms.
- location: has properties based on the current URL.
- history: contains properties representing URLs the client has previously requested.

Depending on its content, the document may contain other objects. For instance, each form (defined by a FORM tag) in the document has a

corresponding Form object.

Navigator object

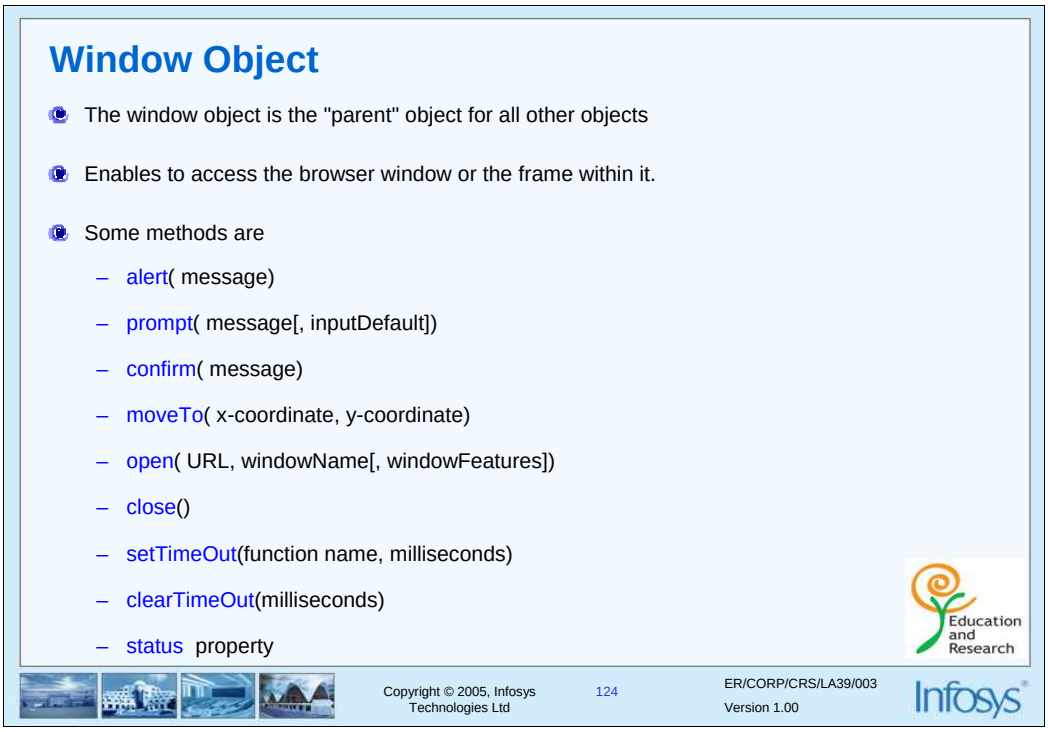
- The navigator object refers to the browser itself.
- Has properties only.You cannot modify properties .
- All of the properties of the navigator object are read-only.

appCodeName	The code name of the browser
appName	The name of the browser.
appVersion	A string that represents version information about the browser.
userAgent	The value of the user-agent header sent in the HTTP protocol from client to server.
platform	indicates the platform (Windows, UNIX, and so on)

The navigator object refers to the browser that is executing the current script. It has properties that you can use to tailor your JavaScript to the browser. The navigator object has no methods and no event handlers.

It has properties only which gives you information about the user's browser. Example that illustrates different properties of navigator object

```
<HTML>
<HEAD>
<TITLE>Browser Information</TITLE>
</HEAD>
<BODY>
<SCRIPT LANGUAGE="Javascript">
document.write("<LI><B>Code Name: </b> " +navigator.appCodeName);
document.write("<LI><B>App Name: </b> " +navigator.appName);
document.write("<LI><B>App Version: </b> " +navigator.appVersion);
document.write("<LI><B>User Agent: </b> " +navigator.userAgent);
document.write ("<LI><B>Platform: </b> " +navigator.platform);
</SCRIPT>
</UL>
<HR>
</BODY>
</HTML>
```



Windows object is used to access the browsers window or a frame within the window.

The term "window" refers to the entire browser display on the user's screen. Each FRAMESET and FRAME element also defines a window.

The browser display window has several components to which the window object's properties and methods refer.

defaultStatus	The message that appears in the window status bar when nothing else is in the status bar. If you change this property within an onMouseOver event handler, the event handler must return true for defaultStatus to be changed.
length	The number of frames. You cannot change this property.
name	The window's name.
parent	A synonym for the window or frame object that contains this window object. You cannot change this property.
self	A synonym for this window object. You cannot change this property.
status	The transitory message that appears in the window status bar-for example, when a mouseOver event occurs. If you change this property within an onMouseOver event handler, the event handler must return true for status to be changed.
top	A synonym for the topmost window that contains frames or nested framesets. You cannot change this property.
opener	Sets or returns the object that opened the current window
window	A synonym for the this window object. You cannot change this property.

Window Methods

```
<HTML>
<BODY>
<FORM name="Winform">
<INPUT TYPE="button" value="Open new window"
   onClick="NewWin=window.open('','NewWin','toolbar=no,status=no,width=700
   ,height=500');"><BR>
<!-- HTML Comments : NewWin is the reference  of the new window -->
<INPUT TYPE="button" value="Close new window"
   onClick="NewWin.close();"><BR>
<INPUT type=button value="Close main window" onClick="window.close();">
 </form>
<!-- HTML Comments : window  is the reference  of the current window -->
</BODY>
</HTML>
```

winopen.html

Opening a Window object

Syntax : *object*=open(URL, windowName[, windowFeatures])

open() returns a window object reference.

URL is the URL to open in the new window.

windowName is a name that a FORM or A element can use in its TARGET attribute.

features is a column-delimited list of window options. Do not use spaces.

height	:	Specifies the height of the window in pixels.
location	:	If yes, creates a Location entry field.
menubar	:	If yes, creates the menu at the top of the window.
resizable	:	If yes, allows a user to resize the window.
scrollbars	:	If yes, creates horizontal and vertical scrollbars
status	:	If yes, creates the status bar at the bottom of the window.
titlebar	:	If yes, creates a window with a title bar.
toolbar	:	If yes, creates the standard browser toolbar.
width	:	Specifies the width of the window in pixels.

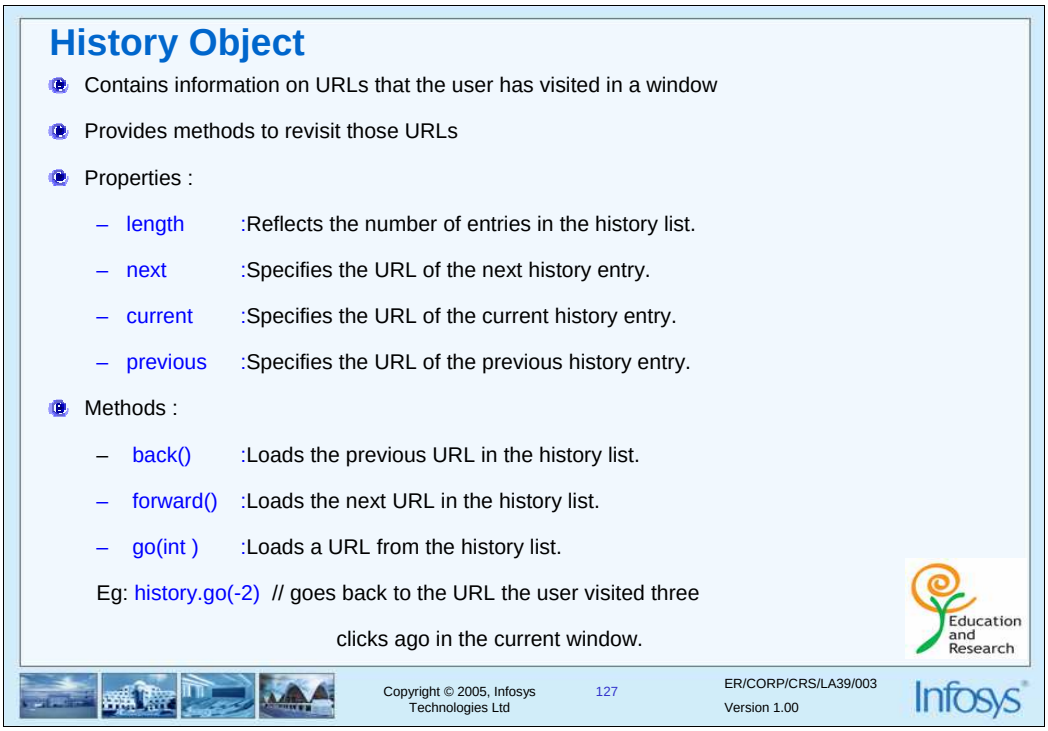
Eg: window.open(“new.html", "messageWindow", "toolbar=yes,directories=yes")

Window Object – Method

- setTimeout – calls a function or evaluates an expression after a specifed interval
- Syntax: **setTimeout("Update();",2000);**
- setTimeout returns a timer ID that can be used in a call to clearTimeout().
- clearTimeout- cancels the timeout specified by timeout id.
- Syntax: **clearTimeout(id);**

settime.html


```
<HTML><HEAD>
<SCRIPT LANGUAGE=Javascript>
var counter=0;   // global variable
ID=window.setTimeout("Update();",2000);
function Update() {
                counter++;
                window.status="The counter is now at "+ counter;
                document.form1.input1.value=" The counter is now at " + counter;
                // referring the text box name using DOM
                ID=window.setTimeout("Update();",2000); // 2000 milliseconds
                // the function will be called recursively every 2 seconds
}
</SCRIPT></HEAD><BODY>
<H1>Using the <I>setTimeout</I> function</H1>
Watch as the number in the textbox changes. This can be changed by setting the second parameter in the <I>setTimeout</I> function. <P>
Also notice what is happening to the browser status bar. By using the <I>window.status</I> property<P>
<FORM  NAME="form1">
<INPUT TYPE="text" Name="input1" ><p>
<INPUT TYPE="button" value="Reset the counter" onClick="counter=0; Update();">
<INPUT TYPE=button value="Stop the counter" onClick="window.clearTimeout(ID);">
</FORM><P>
</BODY>
</HTML>
```



The history object contains a list of strings representing the URLs the client has visited. You can access the current, next, and previous history entries by using the history object's current, next, and previous properties. You can access the other history values using the history array. This array contains an entry for each history entry in source order; each array entry is a string containing a URL. Length property indicates number of entries in the history object. You cannot modify the length property.

```
<HTML>
<BODY>
<H1> Back and Forward Buttons</H1>
<A href="javascript:history.go(-1);">BACK</A><br>
<A href="javascript:history.go(1);">Forward</A><br>
<A href="javascript:location.reload();"> Reload</A><br>
<A href="javascript:location.replace('backandforward.html');"> Replace</A>
<BR>
<SCRIPT LANGUAGE="Javascript1.2">
document.write ("document.URL=  " + document.URL)
document.write ("<br> history.current=  " + history.current)
document.write ("<br> window.location.href=  " + window.location.href )
</SCRIPT>
</BODY>
</HTML>
```

Location Object

- Contains information on the current URL.
- properties

host	A string specifying the server name, subdomain, and domain name.
hostname	A string containing the full hostname of the server
href	A string specifying the entire URL.
protocol	A string specifying the beginning of the URL, up to and including the first colon.

location.html

- Methods

reload()	Forces a reload of the window's current document
replace(URL)	Loads the specified URL over the current history entry.

The location object represents the complete URL associated with a given window object. Each property

of the location object represents a different portion of the URL.

For example to Change the page you can use

location.href="http://sparsh"

Example illustrating the usage of location objects methods:

```
<HTML>
<HEAD>
</HEAD>
<BODY>
<H1>Using the <I>location</I> object</H1>
<FORM  NAME="form1">Click on this button to go to
Sparsh    
<INPUT  TYPE="button" VALUE="Go to Sparsh"
onClick="location.replace('http://Sparsh/')"><P>
Click on this button to refresh page    
<input type="button" value="Refresh page" onClick="location. reload()">
<P>
</FORM>
</BODY>
</HTML>
```

Document Object

- Each page has only one document object.
- Contains information about the current document, and provides methods for displaying HTML output to the user.
- Methods

write(string)	Writes one or more HTML expressions to a document in the specified window.
writeln(string)	Writes one or more HTML expressions to a document in the specified window and follows them with a new line character.
open()	Opens a stream to collect the output of write method.
close()	Closes an output stream and forces data to display.

An HTML document consists of HEAD and BODY tags. The HEAD tag includes information on the document's title and base

(the absolute URL base to be used for relative URL links in the document). The BODY tag encloses the body of a

document, which is defined by the current URL. The entire body of the document (all other HTML elements for the document) goes

within the BODY tag.

The HTML BODY tag. The JavaScript runtime engine creates a document object for each HTML page. Each window object has a document property whose value is a document object.

Property	Description
anchors	Returns a collection of all anchor elements in a document
applets	Returns a collection of all object elements included in an applet
body	Returns the body or frameset element
cookie	Returns the cookies for the document
domain	Returns the domain name of the document's server
forms	Returns a collection of all the form elements in a document
images	Returns a collection of all the image elements in a document
links	Returns a collection of all the anchor elements in the document with the href attribute specified
referrer	Returns the URL of the previous page
title	Sets or returns the title of this document
URL	Returns the URL of the document

Document Object

- Properties

URL	A string that specifies the complete URL of a document.
title	A string that specifies the contents of the TITLE tag.
lastModified	A string that specifies the date the document was last modified.
alinkColor linkColor vlinkColor	A string that specifies the ALINK attribute. A string that specifies the LINK attribute. A string that specifies the VLINK attribute.
fgColor bgColor	A string that specifies the TEXT attribute. A string that specifies the BGCOLOR attribute.
forms	An array a containing an entry for each form in the document.
height, width	Height and width of the browser

document.html

ER/CORP/CRS/LA39/003
Version 1.00

Infosys

Do not use location as a property of the document object; use the document.URL property instead.

The document.location property, which is a synonym for document.URL.

This example will accept the color from the user and set it as background color of the document and display a message on the document

```
<HTML>
<HEAD>
<TITLE>Document Object Demo</TITLE>
</HEAD>
<BODY>
<SCRIPT LANGUAGE=Javascript>
var sAnswer = window.prompt("Please enter your favourite color : choose from the following Yellow, Green, Red, Blue, Pink, Orange, Black, Gray")
document.bgColor = sAnswer
document. write ("<H1>Welcome " + sAnswer + ":-)!!</H1><HR>We are very happy to demonstrate the usage of <I> document. write</I>. Do remember to view the source!<P>")
</SCRIPT>
</BODY>
</HTML>
```

Form Object

- Each form in a document creates a form object.
- A document can have more than one form
- Form objects in a document are stored in a *forms[ ]* collection.
- The elements on a form are stored in an *elements[ ]* array.
- Each form element has a form property that is a reference to the element's parent form.

```
<form name="form1">
  <input type="radio" name="rad1" value="1">Yes<br>
  <input type="radio" name="rad1" value="0">No<br>
</form>
```

To access the first radio button value you can use

```
document.forms[0].elements[0].value
                (or)
document.form1.rad1[0].value
```

form1.html

Each form in a document creates a Form object. Because a document can contain more than one form, Form objects are stored in an array called forms.

The first form (topmost in the page) is forms[0], the second forms[1], and so on.

In addition to referring to each form by name, you can refer to the first form in a document as

document.forms[0]

Likewise, the elements in a form, such as text fields, radio buttons, and so on, are stored in an elements array. You could refer to the first element (regardless of what it is) in the first form as

document.forms[0].elements[0]

Each form element has a *form* property that is a reference to the element's parent form. This property is especially useful in event handlers, where you might need to refer to another element on the current form.

If multiple objects on the same form have the same NAME attribute, an array of the given name is created automatically. Each element in the array represents an individual Form object. Elements are indexed in source order starting at 0.

```
<html><head>
<script language="JavaScript">
function getval(){
        alert(document.forms[0].elements[0].value);
        alert(document.form1.rad1[0].value);
}</script></head><body>
<form name="form1"><!-- creating radio button group -->
<input type="radio" name="rad1" value="1" checked>Yes<br><input type="radio" name="rad1" value="0">No<br> <input type="button" onClick="getval()" value="click">
</form></body></html>
```

Form Object

- Properties

action	Reflects the ACTION attribute.
elements	An array reflecting all the elements in a form.
length	Reflects the number of elements on a form.
method	Reflects the METHOD attribute.
target	Reflects the TARGET attribute.

- Methods

reset()	Resets a form.
submit()	Submits a form.

- Event Handlers
  - onReset, onSubmit

form3.html

Infosys

You can refer to a form's elements in your code by using the elements array. This array contains an entry for each object (Button, Checkbox,File, Hidden, Password, Radio, Reset, Select, Submit, Text, or Textarea object) in a form in source order. Each radio button in a Radio

object appears as a separate element in the elements array.

The reset method restores a form element's default values. A reset button does not need to be defined for the form.

The submit method submits the specified form. It performs the same action as a submit button.

```
<html><head>
<script language="JavaScript">
                function sub()
                {
                                document.form1.action="http://sparsh";
                                document.form1.submit();
                }
</script>
</head><body>
<form name="form1">
                <input type="button"  value="don't submit" name="b1">
                <input type="button" value=" submit" name ="b2" onClick="sub()">
</form></body></html>
```

Text, Textarea, Password, hidden Objects

- Properties
  - defaultValue : Reflects the VALUE attribute.
  - name : NAME attribute.
  - type : Reflects the TYPE attribute.
  - value : Reflects the current value of the Text object's field.
- Methods
  - focus() : Gives focus to the object.
  - blur() : Removes focus from the object.
  - select() : Selects the input area of the object.

form4.html

- Event Handler
  - onBlur : when a form element loses focus
  - onChange : field loses focus and its value has been modified.
  - onFocus : when a form element receives input focus.
  - onSelect : when a user selects some of the text within a text field.

Infosys

```
<html>
<body>
<form name="form1">
                <!-- HTML Comment:  Using Inline JavaScript  -->
                <input type="text"  name="txt1" value="default"
onSelect="alert('selected')"><br>
                <input type="button" value=" focus" name="but1"
onClick="document.form1.txt1.focus()"><br>
                <input type="button" value="select" name="but2"
onClick="document.form1.txt1.select()"><br>
                <input type="button" value="clear" name="but3"
onClick="document.form1.txt1.value='';"><br>
</form>
</body>
</html>
```

Radio object

- Properties
  - name, type, value, defaultChecked, defaultvalue, checked
  - checked property will have a Boolean value specifying the selection state of a radio button. (true/false)
- Methods
  - click( ), focus( ), blur( )
- Event Handler
  - onClick, onBlur, onFocus()

```
if(document.forms[0].rad1[0].checked == true)
{
    alert("button is checked")
}
```

form5.html

An individual radio button in a set of radio buttons on an HTML form. The user can use a set of radio buttons to choose one item from a list.

```
<html><head>
<script language="JavaScript">
function fun1()
{
/*document.form1.radcol.length ,    will give the number of radio buttons in the group*/
for(i=0;i<document.form1.radcol.length;i++)
{
if(document.form1.radcol[i].checked == true)   //radcol is the name of the radio buttons
                    {
                    document.bgColor=document.form1.radcol[i].value;
                    }
}
}
</script></head><body>
<form name="form1"><!-- creating radio button group -->
                    <input type="radio" name="radcol" value="red"
onClick="fun1()">Red<br>
                    <input type="radio" name="radcol" value="green"
onClick="fun1()">Green<br>
                    <input type="radio" name="radcol" value="blue"
onClick="fun1()">Blue<br>
</form>
</body></html>
```

Checkbox object

- Properties
  - checked, defaultChecked, name, type, value
- Methods
  - click()
- Event Handler
  - onClick, onBlur, onFocus()

```
if(document.form1.chk1.checked==true)
      red=255;
else
      red=0;
```

form6.html

A checkbox is a toggle switch that lets the user set a value on or off. It's has same properties and methods as that for a radio button object.

The following example contains a form with three text boxes and one checkbox. The user can use the checkbox to choose whether the text fields are converted to uppercase. onChange event of the text field and onClick event of the checkbox converts the field value to uppercase if the checkbox is checked.

```
<html><head><script language="JavaScript">
red=0;          green=0; blue=0;
function fun1(){
                if(document.form1.chk1.checked==true)
                                red=255;
                                else            red=0;
                if(document.form1.chk2.checked==true)
                                green=255;
                                else green=0;
                if(document.form1.chk3.checked==true)
                                blue=255;
                                else blue=0;
                col="rgb("+red+","+green+","+blue+")"
                document.bgColor=col
}</script></head><body>
<form name="form1"><!-- creating radio button group -->
                <input type="checkbox" name="chk1" value="red" onClick="fun1()">Red<br>
                <input type="checkbox" name="chk2" value="green" onClick="fun1()">Green<br>
                <input type="checkbox" name="chk3" value="blue" onClick="fun1()">Blue<br>
</form></body></html>
```

Button, reset, submit Objects

- Properties
  - form, name, type, value
- Methods
  - click( ), blur( ), focus( )
- Event Handler
  - onClick, onBlur, onFocus, onMouseDown, onMouseUp
- The disabled property
  - If this attribute is set to true, the button is disabled
  - This attribute can use with all other form elements to disable the element

```
<input type="button" name="b1" value="ok" disabled>

document.forms[0].b1.disabled=true;
```

form7.html

The button object is created for each push button on an HTML form.
form property: An object reference specifying the form containing the button.

```
<html><head>
<script language="JavaScript">
                function fun1()  {
                                alert("clicked");
                                document.form1.chk1.disabled=true;
                }
                function m1()    {
                                                document.form1.chk1.width=60;
                                }
                function m2()    {
                                                document.form1.chk1.width=40;
                                }
</script>
</head><body>
<form name="form1"><!-- creating radio button group -->
                <input type="button" name="chk1" value="red" onClick="fun1()"
                onMouseOver="m1()" onMouseOut="m2()">
</form></body></html>
```

Select Object

- The user can choose one or more items from a selection list, depending on how the list was created.
- Properties

length	Reflects the number of options in the selection list.
options	Reflects the OPTION tags.
selectedIndex	Reflects the index of the selected option (or the first Selected option, if multiple options are selected).

- Methods
  - blur(), focus()
- Event Handler
  - onBlur, onChange, onFocus

The select object is created for a selection list on an HTML form. Option objects are created the options in the selection list.

The user can choose one or more items from a selection list, depending on how the list was created.

The runtime engine also creates Option objects for each OPTION tag inside the SELECT tag.

Options in a Select object are indexed in the order in which they are defined, starting with an index of 0. You can set the selectedIndex property at any time. The display of the Select object updates immediately when you set the selectedIndex property.

If no option is selected, selectedIndex has a value of -1.

In general, the selectedIndex property is more useful for Select objects that are created without the MULTIPLE attribute.

The Option.selected property is more useful in conjunction with Select objects that are created with the MULTIPLE attribute.

You can refer to the options of a Select object by using the options array. This array contains an entry for each option in a Select object (OPTION tag) in source order. For example, if a Select object named musicStyle contains three options, you can access these options as musicStyle.options[0], musicStyle.options[1], and musicStyle.options[2].

Option Object

- An option in a selection list.
- Properties

index	The zero-based index of an element in the Select.options array.
selected	Specifies the current selection state of the option
text	Specifies the text for the option
value	Specifies the value for the option
length	The number of elements in the Select.options array.

form9.html

Usually you work with Option objects in the context of a selection list (a Select object). When JavaScript creates a Select object for each SELECT tag in the document, it creates Option objects for the OPTION tags inside the SELECT tag and puts those objects in the options array of the Select object.

```
<html><head>
<script language="JavaScript">
function disp()
{
        ind = document.f1.music.selectedIndex;  //index of the selected Item
        alert("the value is  \n"+document.f1.music.options[ind].value);
        alert("the text is  \n"+document.f1.music.options[ind].text);
}
</script>
</head><body><form name="f1">
        <SELECT NAME="music" onChange="disp()">
                <OPTION  value="10"> R&B
                <OPTION  value="20"> Jazz
                <OPTION value="30"> Blues
                <OPTION value="40"> New Age
        </SELECT>
</form></body></html>
```

Image Object

- An image on an HTML form.

```
<img src="s1.jpg" name="img1">
```

Properties

- src
- width
- height

WinZip File

- You can change the image using the src property.

```
document.img1.src="s2.jpg"
```

Education and Research

The JavaScript runtime engine creates an Image object corresponding to each IMG tag in your document. It puts these

objects in an array in the document.images property. You access an Image object by indexing this array.

```
<html><head>
<script language="JavaScript">
function chng()
{
        document.img1.src="s2.jpg";
}
</script>
</head>
<body>
                <img src="s1.jpg" name="img1">
                <input type="button" name="b1" value=" change "
onClick="chng()">
</body>
</html>
```

Form Validation

- Client-side form validation is an important part of a website where data needs to be collected from the user.
- It is the job of the web programmer, to make sure his pages which use forms include client-side form validation using JavaScript.
- What to Validate?
  - required fields are entered
  - correct data type is entered
  - user's input conforms to a certain standard (such as telephone numbers, email, etc.).
  - Data is in a certain range. (age between 1 and 100)
- Tips
  - Put your common JavaScript validation functions into a common file (.js).
  - validate data on all the fields in your forms (if storing inside database)
  - JavaScript string object has a lot of functionality that many people spend time programming themselves
  - Limit the number of characters in a field and disable the fields which user need not enter the data. (use MAXLENGTH of form elements)

Advantages of Client Side Validation

- Client side validation is faster than server side validation
- Validating forms on the client side lightens the load on the server and allows for you to format data before it is sent for processing.
- Using server-side validation, there is noticeable lag time when data the user submits contains an error. Form data must travel over the Internet and be examined by the server.
- Developers will find it easier and more intuitive to do form validation with JavaScript than with server side techniques since the code has already been written and is easy to implement.

Form Validation

- Let's look at an example.
- You have a form with the following fields:
  - Name
  - Age
  - Email
- What to validate
  - All the fields are entered.
  - Name should be a string with minimum 3 Character.
  - Age should be a number between 1 and 100
  - Email should be a string with an “@” character.
- Make use of String functions and top level functions to validate the fields

val.html


```
<html><head>
<style>
                td{ font-size:20pt;}
</style>
<script language="JavaScript">
/* function for checking age is number between 1 and 100  */
function checkAge(age)
{
                if(!(isFinite(age))){  // not isFinite
                                                return false;}
                iage=parseInt(age);
                if((iage>0) && (iage <=100)){
                                                return true;}
                else{ return false;             }
}
/* function for checking name is a string with minimum 3 characters */
function checkName(username)
{
                if((isFinite(username))){  return false;}
                if( username.length>2){ return true;}
                else { return false; }
}
```

Summary

- What is DOM ?
- Hierarchy of DOM objects
- How does the HTML page refletcts these objects?
- Window Object
- Navigator Object
- History Object
- Location object
- Document Object
- Form Object
- Form validation


```
/*  ******          *Include function for validating email                **********/
function check(){
                nameflag=checkName(document.f1.txtname.value)
                if(nameflag==false){
                                alert("Please enter a proper name!..");
                                document.f1.txtname.select();
                                document.f1.txtname.focus();
                                return false;       }
                ageflag=checkAge(document.f1.txtage.value);
                if(ageflag==false){
                                alert("Please enter a proper age!..");
                                document.f1.txtage.select();
                                document.f1.txtage.focus();
                                return false;       }
return true;}
</script></head><body>
<center><h1> Enter Details</h1></center>
<form name="f1"><table width="50%" align="center" border="0" bgcolor="gray">
<tr><td>Name (*)</td><td>
<input type="text" name="txtname" maxlength="32" size="32"></td></tr>
<tr><td>Age (*)</td><td>
<input type="text" name="txtage" maxlength="3" size="10"></td></tr>
<tr><td>Email</td><td>
<input type="text" name="txtemail" maxlength="30" size="32"></td></tr>
<tr><td>  (*) mandatory fields</td><td><input type="submit" name="b1"
value="submit " onClick="return check()"></td></tr>
</table></form></body></html>
```

Thank You!

Education and Research