WEB TECHNOLOGIES NOTES

HTTP Protocol

HTTP stands for **Hypertext Transfer Protocol**. It is a TCP/IP based communication protocol which is used to deliver virtually all files and other data, collectively called resources, on the World Wide Web. These resources could be HTML files, image files, query results, or anything else.

A browser works as an HTTP client because it sends requests to an HTTP server which is called Web server. The Web Server then sends responses back to the client. The standard and default port for HTTP servers to listen on is 80 but it can be changed to any other port like 8080 etc.

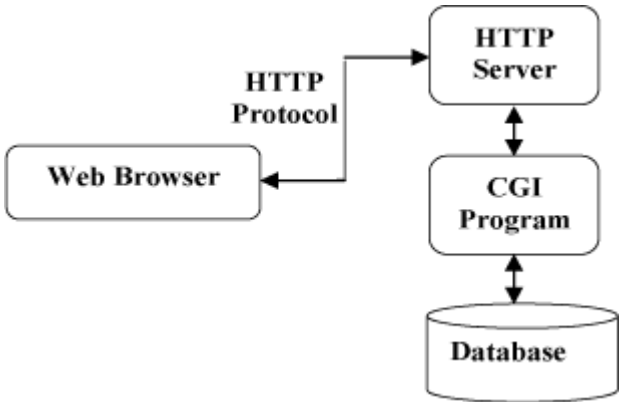
There are three important properties about HTTP

HTTP is connectionless: After a request is made, the client disconnects from the server and waits for a response. The server must re-establish the connection after it processes the request.

HTTP is media independent: Any type of data can be sent by HTTP as long as both the client and server know how to handle the data content.

HTTP is stateless: The server and client are aware of each other only during a request. Afterwards, each forgets the other. For this reason neither the client nor the browser can retain information between different requests across the web pages.

Following diagram shows where HTTP Protocol fits in communication:



The set of common methods for HTTP/1.0 is defined below

1) GET Method: The Get method is used to getting the data from the server. Get method appends the parameters passed as query string to a URL, in the form of key- value pairs. for example, if a parameter is name = Williams, then this string will be appended in the URL. By default the method is Get.

2) POST Method: The post method is used for sending data to the server. In post method the query string is appended along the request object, they do not get appended in the URL, so parameters transfer in hidden form.

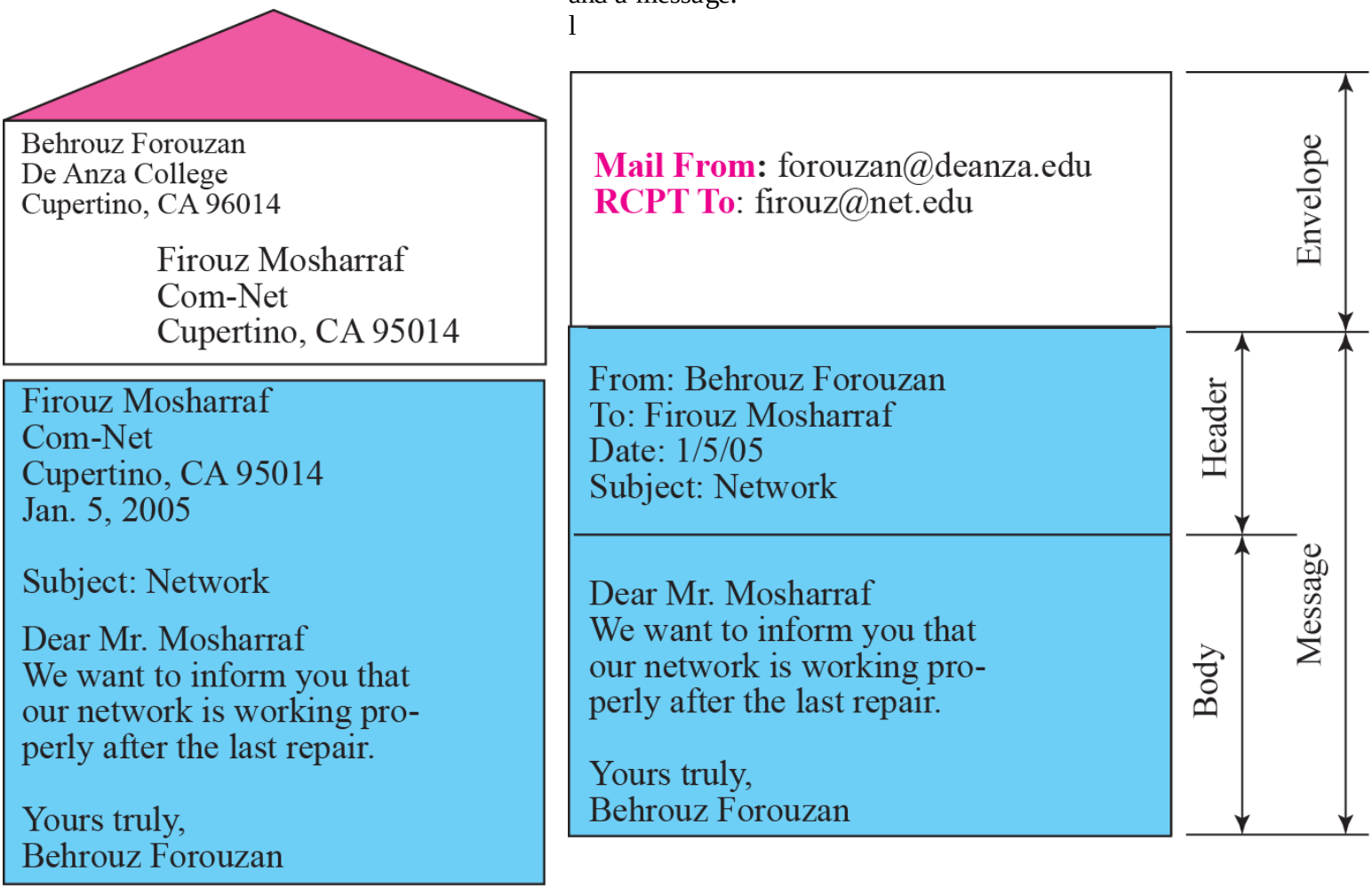
3) HEAD Method: When a user wants to know about the headers, like MIME types, charset, Content- Length then we use Head method. With this body content is not returned.

4) DELETE Method: It is used for delete the resources, files at the requested URL

5) OPTIONS Method: It lists the Http methods to which the thing at the requested URL can respond.
6) PUT Method: It put the enclosed information at the requested URL.
7) CONNECT Method: It connects for the purpose of tunneling.

Simple Mail Transfer Protocol (SMTP)

SMTP stands for **Simple Mail Transfer Protocol**. It's a set of communication guidelines that allow software to transmit email over the Internet. Most email software is designed to use SMTP for communication purposes when sending email, and it only works for outgoing messages. When people set up their email programs, they will typically have to give the address of their Internet service provider's SMTP server for outgoing mail.
To send an e-mail, the user creates mail that looks very similar to postal mail. It has an envelope and a message.
l



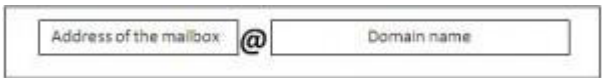
Envelope:

The envelope usually contains the sender address, the receiver address, and the other information.

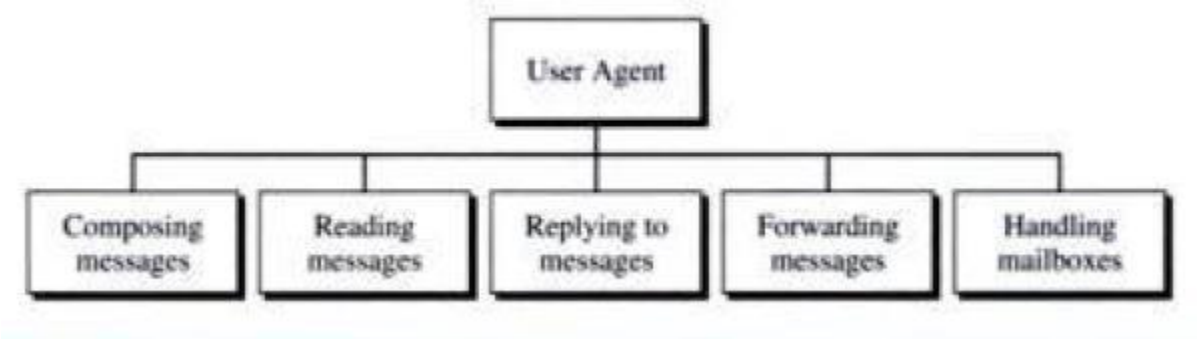
Message:
The message contains the headers and the body. The headers of the message define the sender, the receiver, the subject of the message and the other information. The body of the message contains the actual information to be read by the recipient.

For the process of sending an email to other computers it uses two components.
o **User Agent (UA)**
o **Message Transfer Agent (MTA)**

User Agent (UA): The user agent prepares the message which the user wants to send. It creates an envelope for the message and then it puts the message in the envelope. The user agent prepares the envelope using an unique addressing system.
In the address we have two parts: local port and the domain name.



A user agent is a software package that composes the messages, reads, replies to, and forwards messages. It also handles mail boxes.



Mail Transfer Agent (MTA):
The responsibility of MTA is to deliver the email message to recipient's mail server. This is the important agent in SMTP, because it performs the process of transferring mail across the internet. We have MTA's in both client and server side. Not only have we had MTA's at the client and server side, we have many other MTA's in the network which are all working as relays.

POP3 Protocol

The **POP (Post Office Protocol 3)** protocol provides a simple, standardized way for users to access mailboxes and download messages to their computers. When using the POP protocol all the eMail messages will be downloaded from the mail server to our local computer.

POP is the oldest and most recognizable Internet email protocol. Its current widespread implementation is POP3. POP is a simple protocol to configure, operate and maintain.

When POP was first designed, the cost of constantly staying online was very high. Because of this, POP was built around the offline mail delivery model. This means the end-user connects to an email server, downloads messages, disconnects from the server, then reads email while offline. In other words, POP was designed to collect mail for a single email client.

POP Messages are stored on the mail server until downloaded to the client. They are then stored on the client machine and deleted from the server. The client can contain multiple folders for organizing email. Filters can place mail into specific folders during the download process. The user can mark mail with flags such as read, unread and urgent. A change to the POP standard includes the option to leave email on the server after downloading it. This enables a user to download the same mail using multiple clients on more than one computer. However, there are no server-side file manipulation capabilities, such as marking mail as read or unread. There are also no facilities for creating server-side directories. Instead, leaving email on a server, allows each client to download the same messages one time.

POP Benefits

Local Storage: When not connected, the user can still access and read downloaded email.
Server Saving: POP frees server disk space because it downloads emails and attachments then deletes them from the server.
Legacy Systems: For people with older systems, POP may be the only choice.

The POP3 protocol has three process states for handling the connection between the mail server and the POP3 e-mail client:

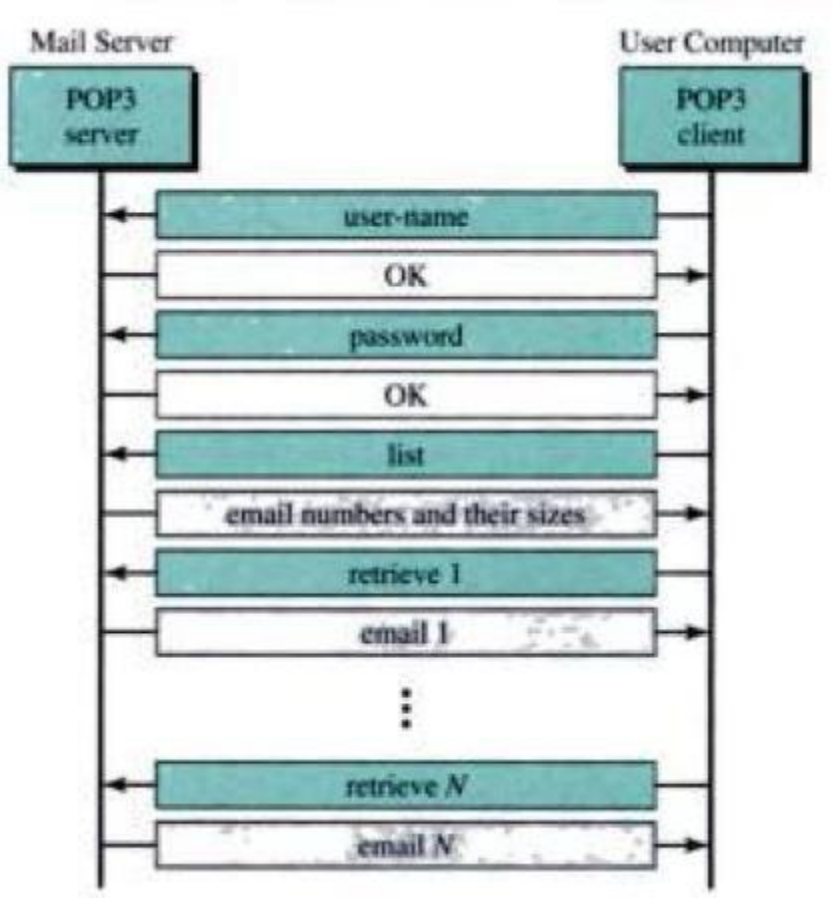
- **The authentication state**
- **The transaction state**
- **The update state.**

Authentication State:
During the authentication state, the POP3 e-mail client that is connected to the server must be authenticated before users can retrieve their e-mail. If the user name and password that are provided by the e-mail client match those on the server, the user is authenticated and proceeds to the transaction state. If not, the user receives an error message and is not allowed to connect to retrieve e-mail.

Transaction State:
During the transaction state, the client sends POP3 commands and the server receives and responds to them according to the POP3 protocol.

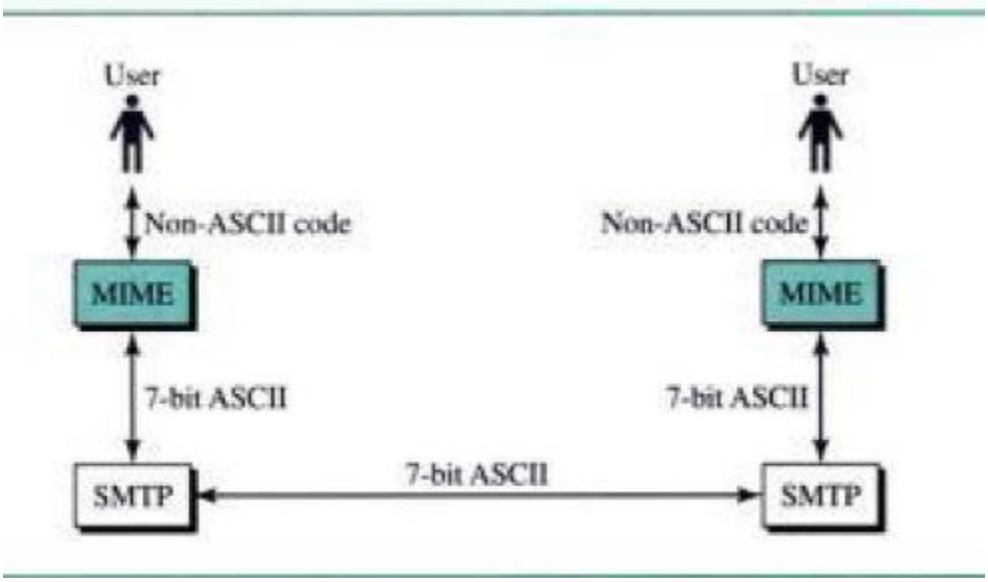
Update State:

The update state closes the connection between the client and server. It is the last command the client transmits. After the connection is closed, the mail store is updated to reflect the changes made while the user was connected to the mail server.



Multipurpose Internet Mail Extension (MIME)

MIME means **M**ultipurpose **I**nternet **M**ail **E**xtensions, and refers to an official Internet standard that specifies how messages must be formatted so that they can be exchanged between different email systems. MIME is a very flexible format, permitting one to include virtually any type of file or document in an email message. Specifically, MIME messages can contain text, images, audio, video, or other application-specific data.



IMAP

IMAP is a more recent development in email technology. Its current implementation is IMAP4. IMAP can do all POP functions, plus many more. It can be simple or complex to configure, operate and maintain.

IMAP was designed for users to stay connected to one or more email servers while reading, creating and organizing messages. Users can manipulate both mail and mailboxes on the server side. Mailboxes can be added, nested, renamed, moved and deleted; they can be for private and public access. Shared mail boxes are available for such uses as mailing lists, announcements and common projects.

Messages can be freely copied and moved between mailboxes on one or more accounts; they can also be moved between clients and servers. Email accounts can be accessed from various clients on many machines. All clients see the same mailboxes and messages in each account.

While IMAP email is stored on the server, messages can also be cached on a local machine for offline use. Deleting the cached messages does not delete the messages on the server. According to its standard, IMAP clients can filter messages to mailboxes on the client.

IMAP can potentially save bandwidth by downloading just the headers (To, From, Subject) of emails without transferring the message or any attachments to the client. This gives the user the choice to download the message and any attachments.

IMAP Benefits

• Multiple Client Support. Messages can be viewed on any computer with an IMAP client.
• Public and group folders. Because they are on the server everyone can see and use them.
• Configurability. It has dozens of options based on all three mail delivery models.

Introduction to HTML

What is an HTML File?

- HTML stands for **H**yper **T**ext **M**arkup **L**anguage
- An HTML file is a text file containing small **markup tags**
- The markup tags tell the Web browser **how to display** the page
- An HTML file must have an **htm** or **html** file extension

An HTML file can be created using a **simple text editor**

HTML Elements

HTML documents are text files made up of HTML elements. HTML elements are defined using HTML tags.

HTML Tags (Elements)

- HTML tags are used to mark-up HTML **elements**
- HTML tags are surrounded by the **two characters < and >**
- The surrounding characters are called **angle brackets**
- HTML tags normally **come in pairs** like and
- The first tag in a pair is the **start tag,** the second tag is the **end tag**
- The text between the start and end tags is the **element content**
- HTML tags are **not case sensitive,** means the same as <B>

In Html there are two types of Tags :

1.Paired: requires a ending tag which ends with a Slash /

<html>….</html>

<body>….</body>

2.Single: No need to specify the ending tag.

 , <li> , <td>

Tag Attributes

Tags can have attributes. Attributes can provide additional information about the HTML elements on your page.

Basic Structure of a HTML Document:

HTML documents are enclosed by html tags *<html>...</html>*. The rest of the document has two sections : the head and body.

<html>

<head>…</head>
<body>…</body>
</html>

The head section contains information about the document.

- e.g. the title of the document.

- For example:

<head>
<title>My Home Page</title>
</head>

"My Home Page" will appear in title bar of the browser.

Head tag Will contain the following tags

- <title>
  - Tells browser what to put in the title bar of page
- <script>
  - Tells browser what, if any, scripting language is being used
- <meta>
  - Designates meta tags for search engines
- <link>
  - Directs other pages appearance
- <style>
  - Use in Cascading Style Sheet (CSS)

<base> - tag:

Establishes a base URL for the website. This allows the user to direct the link to a page of specified website.

The body section contains text that will be displayed in the browser window.

- Some useful/common tags used in the body:
  - *
* - line break. White spaces are ignored.
  - *<p>* - paragraph break.
  - * … * - Centers the enclosed text.

Body tag Attributes:

- Bgcolor : <body bgcolor="red">
- Background : <body background="c:\imgs\flower.jpg">
- Bgproperties :<body bgproperties=FIXED>
- Bottommargine : <body bottommargine=100>
- Topmargine : <body topmargine=100>
- Leftmargin : <body leftmargine=100>
- Rightmargine :<body rightmargine=100>
- Link : <body link="blue">
- Alink : <body alink="black">

- Vlink : <body vlink="red"
- Scroll : <body scroll=yes/no>
- Text : <body text = "brown">

Basic HTML Tags:

Text Formatting Tags

- <b> : To apply **Bold** on the text
- <i> : To apply *italic* on the text.
- <u> : To apply the underline on the text.
- : Display the text center.
- <strike> : Text will be strike
- <tt> : Displays teletype text. Usually rendered in monospace font.
- <big> : Displays the text larger than usual.
- <small> :Displays the text smaller than usual.
- <sub> : Displays the text as subscript
- <sup> : Displays the text as superscript

Advanced Text Formatting Tags

- <Address> : To display the address information.
- <code> : program code.
- : Emphasized text
- <samp> : Sample output.
- : Strongly emphasized text
- <Abbr> : Abbreviated text
- <acronym> : Acronym text.

DOCUMENT HEADING

- Use as document header in HTML body.
- Starting and closing tags for header:

<h1>……</h1>

- There are six levels of headings, from Heading 1 through Heading 6.
- Various types of header:

<h1></h1>
<h2></h2>
<h3></h3>
<h4></h4>
<h5></h5>
<h6></h6>

EXAMPLE:

```
<html>
  <head>
    <title>Web Authoring</title>
  </head>
```

```
  <body>
    HTML is easy!!
    <h1>This line using h1</h1>
    <h2>This line using h2</h2>
    <h3>This line using h3</h3>
    <h4>This line using h4</h4>
    <h5>This line using h5</h5>
    <h6>This line using h6</h6>
  </body>
</html>
```

**
 and <hr> Tags**

-
 tag is a break tag. Used to break up the current line and cursor goes to next line.
- <hr> tag is used to display the horizontal line. Attributes of <hr> tag is
  - Align = left/right/center
  - Width – horizontal width of the line
  - Size – vertical size of the line
  - Noshade – displayed in two-dimensional

HTML Comments

<!--> Defines a comment

** tag**

- Used to sets the font properties to the text on the webpage.
- Attributes:
  - Face : sets the font type like Arial,tahoma.
  - Size : Sets the size of the text. possible values are 1 through 7.
  - Color : Sets the color of the text.
  - Example :

 This is font tag

Font Attributes

Attribute	Example	Purpose
Size="number"	size="2"	Defines the font size
Size="+number"	size="+1"	Increases the font size
Size="-number"	size="-1"	Decreases the font size
Face="face-name"	face="Times"	Defines the font-name
color="color-value"	color="#eeff00"	Defines the font color
color="color-name"	color="red"	Defines the font color

<p> tag

- Formats the text into a paragraph and adds space before the paragraph.
- Attributes:
  - Align : sets the alignment of the text in the paragraph. Possible values are LEFT, RIGHT,CENTER,JUSTIFY

<marquee> tag

- Displays scrolling text in a marquee style.
- Attributes
  - Align: Sets the alignment of the text relative to marquee box. Possible values are TOP(default),MIDDLE,BOTTOM.
  - Behavior : Sets how the text in the marquee should move. Can be SCROLL,SLIDE,ALTERNATE.
  - Bgcolor : Sets the background color for the marquee box.
  - Direction : Sets the direction of the text should scroll. Can be LEFT(default),RIGHT,DOWN,UP.

<pre> - Displaying Preformatted Text

- Tells the browser that the enclosed text is preformatted and display it as it is on the browser.

```
<pre>
  preformatted text
</pre>
```

<DIV> - Formatting a block of text

- Used for putting the text in terms of blocks so we can apply the styles to individual blocks.
- Attributes
  - Align : Sets the horizontal alignment of the element in the page. Possible values are LEFT(default),RIGHT,CENTER,JUSTIFY.

Working with Images

<img> - Adding an image to a web page

Inserts an image into a webpage.

- **Attributes**
  - **Src** : should be required. Specifies the path of the actual image to display.
  - **Align** : Sets the alignment of the image relative to text. Can be LEFT,RIGHT,TOP, BOTTOM, MIDDLE.
  - **Alt** : Displays the text to be displayed in place of an image for browsers that can't handle the images.
  - **Border** : Sets the border size of the image . Set to 0 for no border or a positive integer pixel value.
  - **Height** : Specifies the height of the image. if we specify , can speed up the downloading of the image.
  - **Width** : Specifies the width of the image. if we specify , can speed up the downloading of the image.
  - **Hspace** : Sets the horizontal spacing (both right & left sides) around the image.
  - **Vspace** : Sets the vertical spacing (both top & bottom sides) around the image.

Working with Hyperlinks: Anchor tag

<A> - Creating a Hyperlink Document

- Refer to other sources such as HTML documents and images. Both text and images can act as hyperlinks.
- Attributes :
  - **Href** : Holds the target URL of the hyperlink. Either this attribute or NAME attribute must be used.
  - **Name** : Specifies an anchor name, the name we want to use is as the target of a hyperlink.
  - **Accesskey** : Assigns a keyboard shortcut to the hyperlink. Set to a single alphanumeric value.
  - **Title**: Holds the text that will be displayed in tool tips when the mouse moves over the hyperlink.
  - **Target**: Defines where the linked document has to be opened.

- To link to another page on same web
  -
- To link to another web
  -
- To link to another page in other web
  -

Setting Hyperlink text colors

- In the <body> tag **link,alink** and **vlink** attributes specify the colors for the hyperlink text depending upon the visiting status.
  - Link : <body link="blue"> - color of the hyperlink that have not yet been visited.
  - Alink : <body alink="black"> - color of the hyperlinks as they are being clicked
  - Vlink : <body vlink="red"> - color of the hyperlink that have been visited.

BASE> - Setting the base for hyperlinks

- Sets the base URL for the hyperlinks in a page. Can only be used in <head> tag.
- Relative hyperlinks are defined relative to current page, unless we specify a different base to use for hyperlinks with <base> element.
- Attributes:
  - Href : Holds the URL we want to use as the base URL of hyperlinks in the page.

Eg:

<head>

<base href=http://jntu.ac.in/imgs>

</head>

<body>

< a href=http://jntu.ac.in/results.html>

<img src="clickme.gif">

Syllabus

</body>

Linking to a section of a Document

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

Syntax :-

Text to be displayed

We can refer to that anchor as #label in <a> tag href attribute.

Syntax :-

Text to be displayed

Creating Lists

- Lists may be used for a piece of information and for providing a straightforward index to the site.

Types of Lists

- HTML provides three types of lists.
  - Unordered Lists -

- Ordered Lists -
- Definition Lists - <dl>

The ordered and unordered lists are each made up of sets of list items. Elements of a list may be formatted with <li> tag.

Unordered Lists - Tag

- tag is used to create an unordered list. This list has a bullet in front of each list item. Every list item in <ul> tag must be encapsulated within <li>.. tag.
- Attributes
  - **Type** : Specifies the type of list item. For unordered list sets to DISC(solid bullet), SQUARE(solid square),CIRCLE(hollow bullet).
  - **Compact** : stand alone attribute specifying that compact rendering be used.

eg : <ul compact>

Ordered Lists - tag

- tag is used to Create an ordered list. This list has a sequence of numbers in front of each list item. Every list item in <ol> tag must be encapsulated within <li>.. tag.
- Attributes
  - Type : Specifies the type of list item. For ordered list sets to A, a, I, i ,1.
  - Compact : stand alone attribute specifying that compact rendering be used.

eg : <ol compact>

Definition Lists - <DL> , <DT> , <DD>

- The definition list is a special kind of list for providing terms followed by a description for them.
- Definition lists are contained inside the <dl> element. The <dl> element then contains alternating <dt> and <dd> elements.
- The content of the <dt> element is the term we will be defining.
- The <dd> element contains the definition of the previous <dt> element

Nesting Lists

- We can nest lists inside other lists.
- For example, we might want a numbered list with separate points corresponding to one of the list items. Each list will be numbered separately . And each new list should be placed inside a <li> element

```
<ol>
<li>Speed
    <ul>
        <li>1.5 Ghz
        <li>2.2 Ghz
        <li>2.7 Ghz
    </ul>
<li>RAM
    <ul>
        <li>128 MB
        <li>256 MB
        <li>512 MB
        <li>1 GB
        <li>2 GB
    </ul>
<li>Hard Disk
    <ul>
        <li>20 GB
        <li>40 GB
        <li>80 GB
        <li>120 GB
        <li>320 GB
```

```
    </ul>
  <li>External Disks
    <ul>
        <li>CD-ROM
        <li>CD-Writer
        <li>Combo Driver
        <li>DVD Driver
    </ul>
  <li>Monitor
    <ul>
        <li>15"
        <li>17"
    </ul>
</ol>
```

Working With Tables:

<TABLE> - Creating a Table

- The TABLE tag defines a table for multi-dimensional data arranged in rows and columns.
- Tables are commonly used to display all manner of data, such as timetables, financial reports, and sports results.
- In order to work with tables, we need to start thinking in *grid* of rectangles. Each rectangle is known as a ***cell***.
- A ***row*** is made up of a set of cells on the same line from left to right, while a ***column*** is made up of a line of cells going from top to bottom.
- Inside the <table> element, the table is written out row by row. A row is contained inside a <tr> element — which stands for *table row*. And each cell is then written inside the row element using a <td> element — which stands for *table data*.

<TABLE> Attributes

- ***BORDER :*** To display a table with borders, we will have to use the border attribute.

- ***CELLSPACING :*** This attribute allows control over the space used between cells in a table. The value should be a pixel value.
- ***CELLPADDING :*** This attribute allows control over the space inserted between the cell data and cell wall in a table. The value should be a pixel value.
- ***WIDTH :*** This attribute is used to describe the desired width of this table, either as an absolute width in pixels, or a percentage of document width.
- ***HEIGHT :*** This attribute describes the height of the table, either as a particular pixel value, or as a percentage of the display window.
- ***ALIGN :*** It allows a table to be aligned to the left or right of the page, allowing text to flow around the table.
- ***BGCOLOR :*** It allows the background color of the table to be specified, using either the specified color names, or a rrggbb hex triplet.
- ***BORDERCOLOR :*** This attribute used to set the border color of the table.

<TR> - Creating a Table Row

- Creates a row in a table. Contains multiple <TD> or <TH> elements.
- Attributes
  - **Align** : Specifies the horizontal alignment of the text in this table row. Set to LEFT, CENTER,RIGHT.
  - **Bgcolor** : Sets the background color of the table cells.
  - **Bordercolor** : Sets the external border color for the row.
  - **Valign** :Sets the vertical alignment of the data in this row. Sets to TOP,MIDDLE,BOTTOM.

<TD> - Creating Table Data

- Creates a table headings; just like table data but usually bold and centered vertically and horizontally.
- Attributes
  - **Align** : Specifies the horizontal alignment of the text in this table row. Set to LEFT, CENTER,RIGHT.
  - **Bgcolor** : Sets the background color of the table cells.
  - **Bordercolor** : Sets the external border color for the row.

- **Valign** :Sets the vertical alignment of the data in this row. Sets to TOP,MIDDLE,BOTTOM.
- **Width** : Specifies the width of the cell. Set to either pixel value or percentage of the display area.
- **Height** : Specifies the height of the cell. Set to either pixel value or percentage of the display area.
- **Colspan** : Indicates how many cell columns of the table this cell should span. Set to a positive integer.
- **Rowspan** : Indicates how many rows of the table this cell should span. Set to a positive integer.
- **Nowrap** : Specifies that data in the cell should not be wrapped by the browser , meaning the table cell will be made long enough to fit the contents without line breaks.

<TH> - Creating Table Headings

- Creates a table headings; just like table data but usually bold and centered vertically and horizontally.
- Attributes
  - **Align** : Specifies the horizontal alignment of the text in this table row. Set to LEFT, CENTER,RIGHT.
  - **Bgcolor** : Sets the background color of the table cells.
  - **Bordercolor** : Sets the external border color for the row.
  - **Valign** :Sets the vertical alignment of the data in this row. Sets to TOP,MIDDLE,BOTTOM.
  - **Width** : Specifies the width of the cell. Set to either pixel value or percentage of the display area.
  - **Height** : Specifies the height of the cell. Set to either pixel value or percentage of the display area.
  - **Colspan** : Indicates how many cell columns of the table this cell should span. Set to a positive integer.
  - **Rowspan** : Indicates how many rows of the table this cell should span. Set to a positive integer.

- **Nowrap** : Specifies that data in the cell should not be wrapped by the browser , meaning the table cell will be made long enough to fit the contents without line breaks.

<CAPTION> - Creating a Table Caption

- Specifies the data for a table cell. Used inside the <TABLE> tag.
- Attributes
  - **Align**
  - **VAlign**

Working with Frames:

Frames

- With frames, we 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.
- *Frames* divide a browser window into several separate pieces or panes, each pane containing a separate HTML page.
- One of the key advantages of frames is that we can load and reload individual panes without having to reload the entire contents of the browser window.
- A collection of frames in the browser window is known as a *frameset*.

<FRAMESET> Tag

- The <frameset> tag defines how to divide the window into frames each frameset defines a set of rows **or** columns. The values of the rows/columns indicate the amount of screen area each row/column will occupy.
- Attributes :
  - **Rows** : Sets the number of rows (horizontal frames) in the frameset. Separate the values assigned to this attribute with commas; each value represents the height of the row. Set to pixel value or percentages or use an asterisk to specify the remaining space.
  - **Cols** : Sets the number of cols (Vertical frames) in the frameset. Separate the values assigned to this attribute with commas; each value represents the height of the column. Set to pixel value or percentages or use an asterisk to specify the remaining space.

- **Border :**It accepts a pixel value, which determines the thickness of any borders used within the frame set.

- Example
  - **<FRAMESET ROWS="20%,60%,20%">**
  - **<FRAMESET COLS="100,*,100">**

<FRAME> Tag

- This tag defines a single frame in a frameset. Used inside the <FRAMESET> element.
- Attributes
  - **SRC:** The SRC attribute takes the URL of the document to be displayed in this particular frame. FRAMEs without SRC attributes are displayed as a blank space the size the frame would have been.
  - **NAME :** The NAME attribute is used to assign a name to a frame so it can be targeted by links in other documents. The NAME attribute is optional; by default all windows are unnamed.
  - **MARGINWIDTH** : Sets the size of the right and left margins used in the frame.
  - **MARGINHEIGHT :** Sets the size of the top and bottom margins used in the frame.
  - **NORESIZE** : Stand alone attribute indicating that the frame may not be resized. The default is that frames may be resized by dragging the border.
  - **SCROLLING** : Determines scrollbar action. Possible values are **AUTO, YES,NO.**

<NOFRAMES> Tag

- <NOFRAMES> tag used inside the <FRAMESET> tag.
- If the browser does not support the <frame> tag, it will display the text enclosed in the <noframes> tag.
- Example

```
<frameset rows=50%,50%>
    <noframes> Browser can't support to divide into frames </noframes>
    <frame src="frame1.html">
    <frame src="frame2.html">
```

<frameset>

<IFRAME> Tag

- **Inline Frame** is a special kind of frame which is used to create a frame inside a web page.
- It doesn't need to appear in a <frameset> element or even in a document that uses the frameset document type declaration
- It is created using <iframe> element and it can have a text around it and we can set borders and margins around the inline frame.
- It is also known as Floating Frame.

Attributes:

Src, Align, height, width, frameborder, scrolling.

Working With Forms:

Forms provide a means of submitting information from the client to the server. Using Form , we will be able to handle HTML controls like textfields , buttons , textareas , checkboxes.

- HTML forms are used to create GUIs on Web pages
  - Usually the purpose is to ask the user for information
  - The information is then sent back to the server
- A form is an area that can contain form elements
  - The syntax is: <form *parameters*> ...*form elements*... </form>
  - Form elements include: buttons, checkboxes, text fields, radio buttons, drop-down menus, etc
    - Other kinds of HTML tags can be mixed in with the form elements
  - A form usually contains a Submit button to send the information in he form elements to the server
  - The form's *parameters* tell JavaScript how to send the information to the server.
- Creates an HTML form; used to enclose HTML controls like buttons and textfields.
- Attributes :

- **Action** : Gives the URL that will handle the form data when the Submit button is clicked.
- **Method** : Indicates a method or protocol for sending data to the target action URL. Possible values are GET(default) / POST.
- **Name** : Gives a name to the form so we can reference it in code.
- **Target** : Indicates a named frame for the browser to display the form results in.

- Creates an HTML form; used to enclose HTML controls like buttons and textfields.
- Attributes :
  - **Action** : Gives the URL that will handle the form data when the Submit button is clicked.
  - **Method** : Indicates a method or protocol for sending data to the target action URL. Possible values are GET(default) / POST.
  - **Name** : Gives a name to the form so we can reference it in code.
  - **Target** : Indicates a named frame for the browser to display the form results in.

<INPUT> tag

- Form elements use the input tag, with a type="..." attribute to tell which kind of element it is
  - type can be text, checkbox, radio, password, hidden, submit, reset, button, file, or image
- Other common input tag attributes include:
  - name: the name of the element
  - accesskey : can assign a keyboard shortcut key.
  - value: the "value" of the element; used in different ways for different values of type
  - disabled: the user can't do anything with this element
  - Size : Sets the size of the control.

Other arguments are defined for the input tag but have meaning only for certain values of type

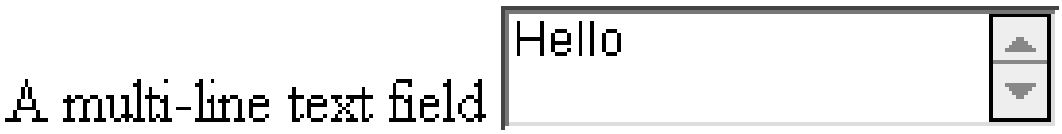
Text input

A text field:

<input type="text" name="textfield" value="with an initial value">

A multi-line text field :

<textarea name="textarea" cols="24" rows="2">Hello</textarea>



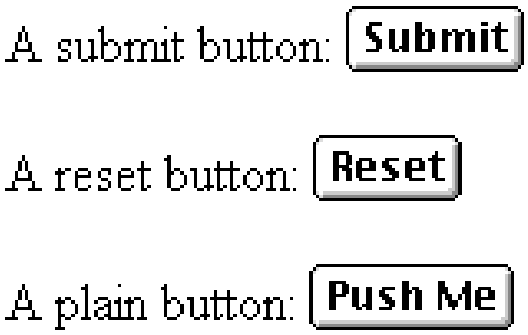
A password field:

<input type="password" name="textfield3" value="secret">

• Note that two of these use the input tag, but one uses textarea

Buttons:

- A submit button:
  <input type="submit" name="Submit" value="Submit">
- A reset button:
  <input type="reset" name="Submit2" value="Reset">
- A plain button:
  <input type="button" name="Submit3" value="Push Me">



- submit: send data
- reset: restore all form elements to their initial state
- button: take some action as specified by JavaScript

Checkboxes:

A checkbox:
<input type="checkbox" name="chkbx" value="checkbox" checked>

- type: "checkbox"
- name: used to reference this form element from JavaScript
- value: value to be returned when element is checked
- Note that there is *no text* associated with the checkbox—you have to supply text in the surrounding HTML

Radio buttons:

Radio buttons:

<input type="radio" name="radiobutton" value="myValue1">male
n

<input type="radio" name="radiobutton" value="myValue2" checked>female

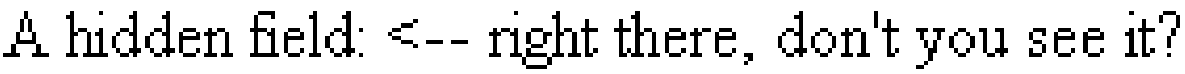
- If two or more radio buttons have the same name, the user can only select one of them at a time
  - This is how you make a radio button "group"
- If you ask for the value of that name, you will get the value specified for the selected radio button
- As with checkboxes, radio buttons do not contain any text
- A menu or list:

```
<select name="select">
   <option value="red">red</option>
   <option value="green">green</option>
   <option value="BLUE">blue</option>
</select>
```

- Additional arguments:
  - size: the number of items visible in the list (default is "1")
  - multiple: if set to "true", any number of items may be selected (default is "false")

Hidden fields:

```
<input type="hidden" name="hiddenField" value="nyah">&lt;-- right there, don't you see it?
```



- What good is this?

- All input fields are sent back to the server, including hidden fields
- This is a way to include information that the user doesn't need to see (or that you don't want her to see)
- The value of a hidden field can be set programmatically (by JavaScript) before the form is submitted

Example:

```
<html>
<head>
<title>Get Identity</title>
</head>
<body>
<p><b>Who are you?</b></p>

<form method="post" action="">
 <p>Name:
  <input type="text" name="textfield">
 </p>
 <p>Gender:
  <input type="radio" name="gender" value="m">Male
  <input type="radio" name="gender" value="f">Female</p>
 </form>

</body>
</html>
```

Output:

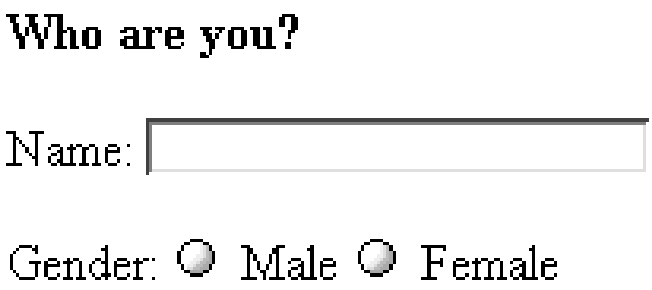


Image Maps:

When a hyperlink is created on an image, we can open a webpage specified by the hyperlink. However HTML also provides the facility to link multiple web pages to a single image through an Image Map. Image Map is a technique that divides image in to multiple sections and allows linking of each section to different web pages. Linked regions of an Image Map are called Hot Regions and each Hot Region is associated with an Html File that is loaded when a Hot Region is clicked. Hot Regions are specified by the Shape attribute of an Area Tag **<area>.**

Example:

```
<img src="planets.gif" width="145" height="126" alt="Planets" usemap="#planetmap">

<map name="planetmap">

 <area shape="rect" coords="0,0,82,126" href="sun.htm" alt="Sun">

 <area shape="circle" coords="90,58,3" href="mercur.htm" alt="Mercury">

 <area shape="circle" coords="124,58,8" href="venus.htm" alt="Venus">

</map>
```

- The usemap attribute specifies an image as a client-side image-map (an image-map is an image with clickable areas).
- The usemap attribute is associated with a <map> element's name or id attribute, and creates a relationship between the <img> and the <map>.

CASCADING STYLE SHEET

Cascading Style Sheets (CSS) is a language that allows us to define how the **HTML** document to appear. This concerns features such as typeface, background, link colors, margins, and placement of objects on a page. The set of instructions for these features are called **"style rules"** in CSS language. They allow web page designers to improve and change the appearance of their web pages very efficiently. Cascading Style Sheet (CSS) provides the way of incorporating style information in a style sheet. The term **"cascading"** indicates that several style sheets can be blended to present a document on the browser's screen.

A **style sheet** is a document that contains style information about one or more documents written in markup languages. It enables us to control rendering of styles such as fonts, color, typeface, size, spacing, margins, and other aspects of documents style. A style sheet is composed of a set of style rules written in a specified format. This set of style rules instructs browsers on how to present a document on the screen.

Advantages of CSS

- ✓ The primary intention of **CSS** is to separate document presentation from document content written in markup languages.
- ✓ CSS will allow designers to create tighter, more dynamic content for the Web.
- ✓ Web pages using CSS will transfer faster to users and will be easy to maintain.
- ✓ CSS will lead to improved accessibility, maintainability and performance on the Web.

CSS Syntax

There are two parts in a style rule. The ***selector*** determines what HTML tag we are styling, and the ***declaration*** is made up of one or more ***property: value*** pairs, ending in a semi-colon and enclosed in curly brackets. In this case a *property* is a display feature such as color, and a *value* is what we want that property to look like.

Syntax:

```
selector {
property1: value1;
property2: value2;
……
propertyN: valueN;
}
```

For example,

```
h1      {  color:    red;     }
selector   property  value
           |_____declaration_____|
```

means "All <h1> headers' text color to be red in entire HTML Document."

LINKING CSS TO A WEB DOCUMENT

For any style sheet to affect an HTML document, it must be attached. This can be done in several ways, depending on which kind of style sheet we are attaching. There are three basic methods of embedding our CSS into the HTML.

- **External style sheet**
- **Embedded style sheet**
- **Inline style sheet**

External Style Sheet

In this case, style information is written in a separate file and is referenced from an HTML document. An external style sheet is useful when the same style is applied on different documents. If the document is an HTML document, the external style sheet is specified using the HTML **<link>** tag. Following is an example:

<link rel = ”stylesheet” type = ”text/css” href = ”mystyle.css”/>

This tag specifies that the style information to be used to display this document is stored in a file named **mystyle.css**. It is inserted in the <**head**> section of an HTML document.

According to W3C specification, external style sheets can be of three types: **Persistent, Alternate, and Preferred**. A persistent style sheet is one that is always applied. Document authors may also specify a set of alternate style sheets, one of which is selected by the user depending on their choice. Authors may also specify one of the alternate style sheets as the preferred style sheet which is applied when no style sheet is selected.

The attribute **rel** specifies the type of style sheet used. Its value is **“stylesheet”** for persistent and preferred style sheets and **“alternate stylesheet”** for alternate style sheets.

Example **CSS Document**(**styles.css)** file

```
hr {color:#6845ff;}
p {margin-left:20px;}
body {font-family:arial,Tahoma; }
```

HTML Document

```
<html>
<head>
<title>Example Web Page</title>
    <link rel = ”stylesheet” type = ”text/css” href = ”styles.css”/>
```

```
</head>
<body>
<p>      Text that will be formatted.    </p>
<hr>
<p>      This will also be formatted.    </p>
</body>
</html>
```

Embedded Style Sheet

An internal style sheet should be used when a single document has a unique style. In this method, style information is placed under the **<style>** tag in the **<head>** section of an HTML page.

```
<html>
<head>
<title>Example Web Page</title>
<style type="text/css">
        hr {color:#6845ff;}
        p {margin-left:20px;}
        body {font-family:arial,Tahoma; }
</style>
</head>
<body>
<p>  Text that will be formatted.    </p>
<hr>
<p>      This will also be formatted.    </p>
</body>
</html>
```

Inline Style Sheet

Style may be in lined using the STYLE attribute. The STYLE attribute may be applied to any BODY elements including BODY itself. The attribute takes as its value any number of CSS declarations, where each declaration is separated by a semicolon. An example follows:

```
<html>
<body>
<p style=”margin-left:20px; direction=ltr”>
Text that will be left aligned paragraph.
</p>
<hr style=”color:#6845ff;”>
```

```
<p style=”margin-right:20px; direction=rtl”>
This will be right aligned paragraph
</p>
</body>
</html>
```

This is the least flexible styling method. If we want to modify the style of an element, we have to work through the document and modify it.

CASCADING RULE

There are several ways to specify style rules such as external, embedded and inline methods. If more than one rule is specified for a tag, then the rules are conflicted. For conflicting rules, latter rules get preference over the earlier rules. That means the inline style sheet definition overrides the embedded style sheet, which in turn overrides the external style sheet for the HTML document.

TEXT FORMATTING CSS

The text displayed in the web page is styled with some of the text formatting properties. The heading uses the text-align, text-transform, and color properties. The paragraph is indented, aligned, and the space between characters is specified. Some of the text formatting properties are

- Text Color
- Text Alignment
- Text Vertical Alignment
- Text Decoration
- Text Transformation
- Text Indentation
- Letter Spacing
- Word Spacing
- Text Direction

Text Color

The color property is used to set the color of the text. The color can be specified by:

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

The default color for a page is defined in the body selector.

Example

```
Body{color:blue;}
h1{color:#00ff00;}
h2 {color:rgb(255,0,0);}
```

Text Alignment

The text-align property is used to set the horizontal alignment of a text. Text can be centered, or aligned to the left or right, or justified.

When text-align is set to "justify", each line is stretched so that every line has equal width, and the left and right margins are straight

Example

```
h1 {text-align: center;}
p {text-align: right;}
td {text-align: justify;}
```

Text Vertical Alignment

The vertical-alignment property is useful when working with inline elements such as images, tables. It allows us to control their vertical positioning within the containing element. It can take several values like.

Value	Description
length	Raises or lower an element by the specified length. Negative values are allowed
%	Raises or lower an element in a percent of the "line-height" property. Negative values are allowed
baseline	Align the baseline of the element with the baseline of the parent element. This is default
sub	Aligns the element as it was subscript
super	Aligns the element as it was superscript
top	The top of the element is aligned with the top of the tallest element on the line
text-top	The top of the element is aligned with the top of the parent element's font
middle	The element is placed in the middle of the parent element
bottom	The bottom of the element is aligned with the lowest element on the line
text-bottom	The bottom of the element is aligned with the bottom of the parent element's font

Examples

```
img { vertical-align: text-top; }
table   { vertical-align: middle; }
```

Text Decoration

The text-decoration property is used to set or remove decorations from text. The possible values for this property are none, overline, line-through, underline and blink.

Examples:

```
a {text-decoration: none;}
h1 {text-decoration: overline;}
h2 {text-decoration: line-through;}
h3 {text-decoration: underline;}
h4 {text-decoration: blink;}
```

Text Transformation

The text-transform property is used to specify either uppercase or lowercase letters in a text. It can be used to turn everything into uppercase or lowercase letters, or capitalize the first letter of each word.

Examples:

```
p {text-transform: uppercase;}
h1 {text-transform: lowercase;}
td{text-transform: capitalize;}
```

Text Indentation

The text-indentation property is used to specify the indentation of the first line of a text with in an element. The value for this property is in terms of px, pc, in, mm

Example

```
p {text-indent:50px;}
```

Letter-spacing Property

The letter-spacing property increases or decreases the space between characters in a text within the element. The value for this property is in terms of px, pc, in, mm.

Example:

```
p {letter-spacing:2px}
h2 {letter-spacing:1px}
```

Word-spacing Property

The word-spacing property increases or decreases the white space between words. The value for this property is in terms of px, pc, in, mm.

Example

```
P {  word-spacing:30px; }
```

Text Direction

The direction property specifies the direction in which the text should flow. The possible values are ltr (left to right), rtl (right to left).

Example

```
p  { direction: rtl; }
```

Line – height Property

The line-height property is used to specify the gap between lines in the text. This should be specified in terms of px, pc, in, mm and percentages.

Example

```
p.small {line-height:90%}
p.big {line-height:200%}
```

Text Pseudo- Classes

There are two very useful pseudo-classes that can help us to work with text. These pseudo-classes allows us to render either the first letter or the first line of an element in a different way than the rest of that element. The name of the pseudo-class is separated from the element by a colon (:).

1. First - letter pseudo-class
2. First – line pseudo-class

first-letter pseudo-class

The first-letter pseudo-class allows us to specify a rule just for the first letter of an element. This is most commonly used on the first character of a new page.

Example

```
p:first-letter {  font-size:200%;  color:#8A2BE2; }
```

first-line pseudo-class

The first-line pseudo-class allows us to render the first line of any paragraph differently from the rest of the paragraph.

Example

```
p:first-line {  background-color : yellow; }
```

TEXT FORMATTING CSS PROPERTIES		
Property	**Description**	**Values**
color	Sets the color of a text	*color*
text-align	Aligns the text in an element	Left, right, center, justify
text-decoration	Adds decoration to text	None, underline, overline, line-through, blink
text-indent	Indents the first line of text in an element	*Length, %*
text-transform	Controls the letters in an element	None, capitalize, uppercase, lowercase
vertical-align	Sets the vertical alignment of an element	Baseline, sub, super, top, text-top middle, bottom text-bottom, *length, %*
letter-spacing	Increase or decrease the space between characters	Normal, *length*
word-spacing	Increase or decrease the space between words	Normal, *length*
direction	Sets the text direction	Ltr, rtl
line-height	Sets the distance between lines	Normal, *number length, %*

CONTROLLING FONTS USING CSS

Several properties related to font allow us to control the appearance of text in our document. CSS font properties define the font family, boldness, size, and the style of a text. Some of the font properties are

- ✓ font
- ✓ Font-family
- ✓ Font-size
- ✓ Font-weight
- ✓ Font-style
- ✓ Font-stretch
- ✓ Font-variant

Font Family

The font family of a text is set with the font-family property. The font-family property should hold several font names as a comma separated list. If the browser does not support the first font, it tries the next font. We provide the list of fonts starts with the font we want, and ends with a generic family, to let the browser pick a similar font in the generic family, if no other fonts are available.

In CSS, there are two types of font family names:

- **generic family** - A group of font families with a similar look such as "Serif" or “Sans Serif”, "Monospace".
- **font family** - a specific font family like "Times New Roman" or "Arial".

Generic family	Font family
Serif	Times New Roman Georgia
Sans-serif	Arial Verdana
Monospace	Courier New Lucida Console

Example

```
p{font-family:"Times New Roman", Times, serif;}
```

Font Style

The font-style property is mostly used to specify italic text. This property has three values:

- normal - The text is shown normally
- italic - The text is shown in italics
- oblique - The text is "leaning" which is very similar to italic

Example

```
p.normal {font-style:normal;}
p.italic {font-style:italic;}
p.oblique {font-style:oblique;}
```

Font-size Property:

The font-size property enables us to specify a size for the font. We can specify a value for this property in several ways like

- Absolute size
- Relative size
- Length
- Percentage

xx-small, x-small, small, medium, large, x-large, xx-large are the values for the absolute sizes.

Smaller, larger are the values for relative sizes.

Length can be expressed in the form of **px, em, pt, in, cm, pc, mm**.

A **percentage** is calculated as a proportion of the element that contains the text like 20%, 10%

Example

```
H1{ font-size:xx-large;}
H6{font-size:smaller;}
P{font-size:12px;}
B{font-size:10%;}
```

Font-weight Property

The font-weight property sets how thick or thin characters in text should be displayed. The possible values for font-weight are nomal, bold, bolder, lighter, 100, 200, 300, 400, 500, 600, 700, 800, 900.

Example

```
p{font-weight:normal;}
h1 {font-weight:bold;}
h6{font-weight:100;}
```

Font-stretch Property

The font-stretch property allows us to make text wider or narrower that means the width of the actual letters in a font. It can take the values like normal, wider, narrower, and expanded.

Font-variant Property

The font-variant property specifies whether or not a text should be displayed in a small-caps font. In a small-caps font, all lowercase letters are converted to uppercase letters. However, the converted uppercase letters appears in a smaller font size than the original uppercase letters in the text.

There are two possible values for the font-variant property like normal and small-caps.

```
p.small{font-variant:small-caps;}
```

font property

The font shorthand property sets all the font properties in one declaration. The properties that can be set, are in order of "font-style font-variant font-weight font-size/line-height font-family". If one of the other values are missing, the default values will be inserted.0

Example

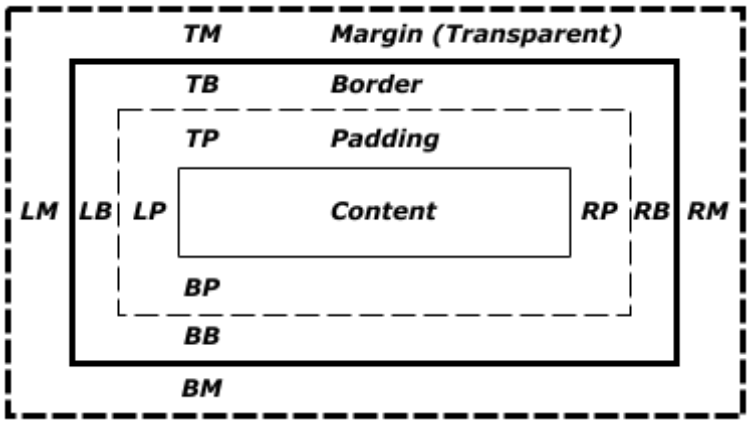
p.first { font:15px arial,sans-serif; }

p.second { font:italic bold 12px/30px Georgia, serif; }

FONT PROPERTIES		
Property	**Description**	**Values**
font	Sets all the font properties in one declaration	*font-style,font-variant, font-weight,font-size, font-family.*
font-family	Specifies the font family for text	*family-name, generic-family*
font-size	Specifies the font size of text	xx-small, x-small, small medium, large ,x-large, xx-large, smaller, larger *length, %*
font-style	Specifies the font style for text	Normal, italic, oblique
font-variant	Specifies whether or not a text should be displayed in a small-caps font	Normal, small-caps
font-weight	Specifies the weight of a font	Normal, bold, bolder, lighter,100,200,300,400 500, 600, 700, 800,900
font-stretch	make text wider or narrower	normal, wider, narrower, expanded.

THE CSS BOX MODEL

All HTML elements can be considered as boxes. The CSS box model is essentially a box that wraps around HTML elements, and it consists of margins, borders, padding, and the actual content. The box model allows us to place a border around elements and arrange the elements inside the border.



The parts of Box Model are:

- **Margin** – The Margin is the distance between the edge of a box and the box next to it.
- **Border** - A border that goes around the padding and content. The border is affected by the background color of the box
- **Padding** – This padding is the space between the content of the box and its border.
- **Content** - The content of the box, where text and images appear

Margin Property for Box Model

The CSS margin properties define the space around elements. The margin does not have a background color, and is completely transparent. 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 margins at once. The value of this property is either a length in terms of px, pm, in or percentage.

In CSS, it is possible to specify different margins for different sides like

margin-top:100px;
margin-bottom:100px;
margin-right:50px;
margin-left:50px;

It is possible to specify all the margin properties in one property. This is called a shorthand property. The shorthand property for all the margin properties is "margin".

The margin property can have from one to four values.

- **margin:25px 50px 75px 100px;**
  - top margin is 25px
  - right margin is 50px
  - bottom margin is 75px
  - left margin is 100px
- **margin:25px 50px 75px;**
  - top margin is 25px
  - right and left margins are 50px
  - bottom margin is 75px
  -
- **margin:25px 50px;**
  - top and bottom margins are 25px
  - right and left margins are 50px
- **margin:25px;**
  - all four margins are 25px

MARGIN CSS PROPERTIES		
Property	**Description**	**Values**
margin	A shorthand property for setting the margin properties in one declaration	*margin-top, margin-right, margin-bottom margin-left*
margin-bottom	Sets the bottom margin of an element	Auto, *length,%*
margin-left	Sets the left margin of an element	Auto, *length, %*
margin-right	Sets the right margin of an element	Auto, *length, %*
margin-top	Sets the top margin of an element	Auto, *length, %*

Border Property for Box Model

The CSS border properties allow us to specify the style and color of an element's border. There are three properties of a border.

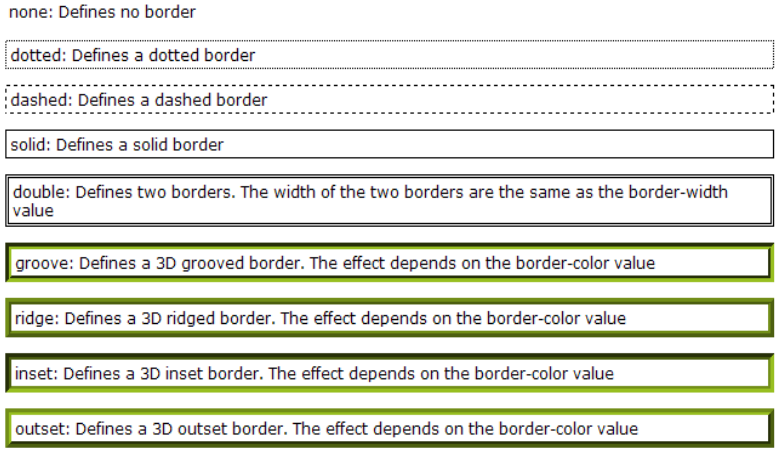
1. border-style
2. border-color
3. border-width

border-style Property

The border –style property allows us to specify the style of the border.

P{border-style:solid;}

The values of the border are



In CSS it is possible to specify different borders for different sides.

For example

```
p
{
border-top-style:dotted;
border-right-style:solid;
border-bottom-style:dotted;
border-left-style:solid;
}
```

border-color Property

The border-color property is used to set the color of the border. The color can be set by two methods.

- name - specify a color name, like "red"
- Hex - specify a hex value, like "#ff0000"

Example

P{border-style:solid; border-color:red; }

In CSS it is possible to specify different borders for different sides

```
P                                                                    {
border-top-style:dotted;
border-right-style:solid;
border-bottom-style:dotted;
border-left-style:solid;
}
```

border-width property

The border-width property is used to set the width of the border. The width is set in pixels, or by using one of the three pre-defined values: thin, medium, or thick.

p{ border-style:solid; border-width:5px;}

Border Property

It is also possible to specify all the individual border properties in one property. The border property is a shorthand for the following individual border properties:

- border-width
- border-style
- border-color

Example

P{border:5px solid red;}

Padding Property for Box Model

The CSS padding properties define the space between the element border and the element content. The top, right, bottom, and left padding can be changed independently using separate properties. A shorthand padding property can also be used, to change all paddings' at once.

Padding - Individual sides

In CSS, it is possible to specify different padding for different sides

padding-top:25px;
padding-bottom:25px;
padding-right:50px;
padding-left:50px;

Padding - Shorthand property

It is possible to specify all the padding properties in one property. The shorthand property for all the padding properties is "padding":

P{padding:25px 50px 75px 100px;}

Property	Description	Values
padding	A shorthand property for setting all the padding properties in one declaration	*padding-top, padding-right, padding-bottom padding-left*
padding-bottom	Sets the bottom padding of an element	*Length, %*
padding-left	Sets the left padding of an element	*Length, %*
padding-right	Sets the right padding of an element	*Length, %*
padding-top	Sets the top padding of an element	*Length, %*

Dimensions for the Box Model

We can set the dimensions for the box in terms of width and height.

Property	Purpose
Height	Sets the height of the box
Width	Sets the width of the box
Max-height	Sets the maximum height of a box
Min-height	Sets the minimum height of a box
Max-width	Sets the maximum width of a box
Min-width	Sets the minimum width of a box

BACKGROUND PROPERTIES

CSS treats each element as in its own box. We can control the background of these boxes using the background properties.

Property	Purpose
background-color	Specifies a background color
background-image	Specifies an image to use as the background
background-repeat	Indicates whether the background image should be repeated
background-attachment	Indicates a background image should be fixed in one position on the page, and whether it should stay in that position when the user scrolls down the page
background-position	Indicates where an image should be positioned
background	A shorthand form that allows you to specify all of these properties

background Color: background-color: <color> | transparent

The background-color property sets the background color of an element.

For example:

body { background-color: white }

h1 { background-color: #000080 }

The value transparent indicates that whatever is behind the element can be seen.

background-image : background-image: <url> | none

The background-image property sets the background image of an element. For example:

body { background-image: url("/images/monkey.gif") }

p { background-image: url("http://www.cclrc.com/pretty.png") }

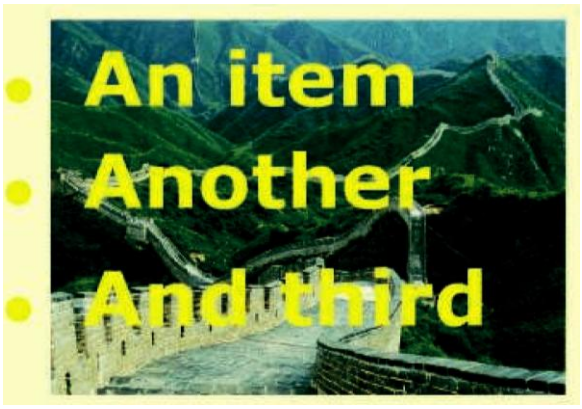
h1 { background-image: none }

We have the HTML like

```
<ul style="background-image:url('wall.png')"> <li>An item</li>

<li>Another
</li>
<li>And
third</li>
</ul>
```

It shows the output like



Background-repeat : background-repeat: repeat | repeat-x | repeat-y | no-repeat

If we do not want our image to repeat all over the background of the box, we should use the background - repeat property, which has four helpful values

Value	Purpose
repeat	This causes the image to repeat to cover the whole page (it is the default therefore rarely used).
repeat-x	The image will be repeated horizontally across the page (not down the whole page vertically).
repeat-y	The image will be repeated vertically down the page (not across horizontally).
no-repeat	The image is displayed only once.

For example

```
body { background-image:
url(/images/monkey.gif) } body { background-
repeat: repeat-x }
```

In the above example, the monkey will only be tiled horizontally

Background-attachment : background-attachment: scroll | fixed

When we specify a background image we can use the background - attachment property to specify whether the image is fixed in its position, or whether it moves as the user scrolls up and down the page.

For example

```
body { background-image:
url("monkey.jpg") } body { background-
attachment: fixed }
```

background-position : background-position: [percentage> | length]{1,2} | [[top | center | bottom] || [left | center | right]]

We may want to alter the position of this image, and we can do this using the background – position property, which takes the values like

Value	Meaning
x% y%	Percentages along the *x* (horizontal) and *y* (vertical) axis
x y	Absolute lengths along the *x* (horizontal) and *y* (vertical) axis in pixels
left	Shown to the left of the page or containing element
center	Shown to the center of the page or containing element
right	Shown to the right of the page or containing element
top	Shown at the top of the page or containing element
center	Shown at the center of the page or containing element
bottom	Shown at the bottom of the page or containing element

Background

The background property allows us to specify several of the background properties at once. The values can be given in any order, and if we do not supply one of the values, the default value will be used.

background: [<background-color> || <background-image> || <background-repeat> || <background-attachment> || <background-position>]

For Example

```
body { background: white url(http://www.clrc/xyz.gif) }
p { background: url(../backgrounds/lion.png) #F0F000 fixed }
h1 { background: #0F0 url(grass.jpg) no-repeat bottom left }
ul { background: white url('monkey.jpg') repeat scroll 0% 50%}
```

SELECTORS

The basic building blocks of style rules are selectors. Selectors determine elements on which rules are to be applied. The elements selected by selectors are called **subjects** of selectors. A selector may simply be the name of an element called **simple selector** or may consist of a rich contextual pattern called **complex selector**. A complex selector consists of one or more simple selectors separated by combinatory such as white space, “>”, and “+”.

Grouping

Selectors having common declarations are grouped into a comma-separated (,) list. For example the below code

h1 { color ; red; }
h2 { color ; red; }
h3 { color ; red; }

This is equivalent to: **h1, h2, h3 { color: red; }**

Type Selectors

A type selector is a simple selector, which is the name of a document element and it matches every single element of the document. For example, the selector **p** selects every **<p>** element in the document. Similarly, the selector **b** selects every **<b>** element.

Universal Selectors

CSS has a special selector *, which matches with every single element in the document. For example,

*** { color: red; }**

It makes all the text in the document red.

The universal selector is useful when element names are not known in advance during the development of a style sheet.

Descendant Selectors

Descendant selectors, also called **contextual selectors**, allow us to determine the elements depending upon their hierarchical relationship. A descendant selector selects only those elements that are descendants of a specified element. A descendant selector is made up of two or more selectors separated by white space. A descendant selector of the form "A B" matches when an element B is the descendant of some ancestor element A.

For Example

```
h1 { color: red }
em { color: red }
```

Child Selector

Child selectors select elements that are immediate children of a specified element. The combinatory used for child selector is **“>”.**

For example

```
body > P { line-height: 1.3 }
```

Attribute Selectors

Attribute selectors provide a ways of selecting elements depending on the presence of an attribute or the presence of certain attribute values. Attribute selectors are not supported by IE but are supported by Firefox and opera.

Simple Attribute Selector

It selects elements having a specified attribute. The general syntax of simple attribute selectors is as follows:

element[attribute_name]

or

[attribute_name]

For example

a [href]

selects all <a> elements having the attribute href. If the element name is omitted, all elements are assumes.

Attribute Value Selector

It allows us to select elements having an attribute with a value equal to given value in CSS. It is used as:

element[attribute_name=”value”]

or

[attribute_name=”value”]

The following example

P[class=”bold”]

matches with the following:

…

One of many attribute value selector

It allows us to select elements having an attribute with a value equal to any one of the values separated by white spaces. It is used as:

element[attribute_name ~ =”value”]

or

[attribute_name~=”value”]

The following example

P[class~=”bold”]

matches with the following:

<p class=”**bold** italic”>…

<p class=”underlined **bold**”>…

Hyphen attribute value selector

This selector selects an element having attribute value exactly equal to the specified value or beginning with the specified value immediately followed by **"-"**. The basic purpose of this selector is to match language sub-code. Here is an example:
p[lang |="en"]
that matches … and … .

Starts-with attribute value selector

This selector selects elements having an attribute value that starts with the value specified. The general syntax is

element[attribute_name^="attribute_value"]

or

[attribute_name^="attribute_value"]

For example, the selector p[type^= "copy"] matches copyright… as well as copyleft…

Ends-with attribute value selector

This attribute selects elements having attribute value that ends with the value specified. The general syntax is

element[attribute_name$="attribute_value"]

or

[attribute_name$="attribute_value"]

For example, a[href$=".com"] selects those anchor tags that point to .com website. So, it matches

yahoo

rediff mail

but not

jadavpur university

Substring match attribute value selector

This selector selects those elements having an attribute value containing at least one occurrence of the value specified. The general syntax is

element[attribute_name*="attribute_value"]

or [attribute_name*="attribute_value"]

For example, a[href*="image"] selects those anchor tags that have image in the href attribute.

Pseudo Classes and Elements

A pseudo-class is way of selecting certain parts of a HTML document, based in principle not on the HTML document tree itself, but on other phantom conditions like language encoding or the dynamic state of an element. Some of the pseudo classes and elements are

- :link
- :visited
- :active
- :hover
- :first-child
- :last-child
- :only-child
- :after
- :before

:first-child Pseudo-class

The :first-child pseudo-class matches a specified element that is the first child of another element.

In the following example, the selector matches any <p> element that is the first child of any element:

Example

```
<html>
<head>
<style type="text/css">
p:first-child { color:blue; }
</style>
</head>
<body>
<p>I am a strong man.</p>
<p>I am a strong man.</p>
</body>
</html>
```

:last-child Pseudo-class

The :last-child selector matches every element that is the last child of its parent.

Example

```
p:last-child { background:#ff0000; }
```

:only-child Pseudo-class

The :only-child selector matches every element that is the only child of its parent

Example

p:only-child { background:#ff0000; }

:before Pseudo-element

The ":before" pseudo-element can be used to insert some content before the content of an element.

Example

h1:before { content:url(smiley.gif); }

:after Pseudo-element

The ":after" pseudo-element can be used to insert some content after the content of an element.

Example

h1:after { content:url(smiley.gif); }

CSS FOR LINKS

Links can be styled with any CSS property such as color, font-family, background, etc. Special CSS for links are in such a way that they can be styled differently depending on what state they are in.

CSS defined some pseudo classes for links that can give greater control over presentation of links. They are

- **a: link - a normal, unvisited link**
- **a: visited - a link the user has visited**
- **a: hover - a link when the user moves over it**
- **a: active - a link the moment it is clicked**

- The **:link** pseudo class applies to those hyper links that have not yet been visited.
- The **:visited** pseudo class applies to hyperlinks that have already been visited at least once.
- The **:hover** pseudo class selects the hyperlinks that are being designated by the user with a pointing device.
- The **:active** pseudo class applies to an element that is currently being activated by the user.

For example :

```
body {background-color:#ffffff;}
a { font-family: arial, verdana, sans-serif; font-size:12px; font-weight:bold;}
a:link { color:#0000ff; text-decoration:none;}
a:visited { color:#333399; text-decoration:none;}
a:link:hover { background-color:#e9e9e9; text-decoration:underline;}
a:active { color:#0033ff; text-decoration:underline;}
```

CSS FOR LISTS

CSS allows us to customize the lists that can be made with HTML. The CSS list properties facilitate us to:

- Set different list item markers for ordered lists
- Set different list item markers for unordered lists
- Set an image as the list item marker

Property	Purpose
list-style-type	Allows you to control the shape or appearance of the marker (the marker is another name for the bullet point or number).
list-style-position	When a list item takes up more than one line, this property specifies where the marker should appear in relation to the text.
list-style-image	Specifies an image for the marker rather than a bullet point or number.
list-style	Serves as shorthand for the preceding properties.
marker-offset	Specifies the distance between a marker and the text in the list.

The list - style - type Property

The list - style - type property allows you to control the shape or style of bullet point in the case of unordered lists and the style of numbering characters in ordered lists.

The values to this property for unordered list are

Value	Marker
none	None
disc (default)	A filled-in circle
circle	An empty circle
square	A filled-in square

The values to this property for ordered list are

Value	Meaning	Example
decimal	Number	1, 2, 3, 4, 5
decimal-leading-zero	0 before the number	01, 02, 03, 04, 05
lower-alpha	Lowercase alphanumeric characters	a, b, c, d, e
upper-alpha	Uppercase alphanumeric characters	A, B, C, D, E
lower-roman	Lowercase Roman numerals	i, ii, iii, iv, v
upper-roman	Uppercase Roman numerals	I, II, III, IV, V

The list - style - type property can either be used on the < ul > and < ol > elements in which case it applies to the entire list or on the individual < li > elements.

For example

ul {list-style-type: circle;}
ol {list-style-type: upper-roman;}
li {list-style-type:disc;}

The list - style - position Property

The list - style - position property indicates whether the marker should appear inside or outside of the box containing the main points. There are two values for this property. They are

Value	Purpose
inside	The marker is inside the block of text (which is indented).
outside	The marker sits to the left of the block of text (this is the default value if this is not specified).

Example

ul {list-style-position:outside;}
ol {list-style-position:inside;}

The list - style - image Property

The list - style - image property allows us to specify an image so that we can use our own bullet style. For this property we must supply the URL of marker image as the value.

For example

li {list-style-image: url("images/bulletpoint.gif");}

The list - style Property

The list - style property is a way of expressing more than one of these properties at once. They can appear in any order. For example:

ul {list-style: inside circle;}

marker-offset property

The *marker-offset* property defines the distance between the marker and text associated to that marker. The value for this property is the length to be defined in terms of px, em, pc, mm. For example

Li { marker-offset: 10em;}

CSS FOR TABLES

Properties that are commonly used with the < table > , < td > , and < th > elements are

- border to set the properties of the border of a table.
- padding to set the amount of space between the border of a table cell and its content — this property is very important to make tables easier to read.
- Properties to change text and fonts.
- text - align to align writing to the left, right, or center of a cell.
- vertical - align to align writing to the top, middle, or bottom of a cell.
- width to set the width of a table or cell.
- height to set the height of a cell (often used on a row as well).
- background - color to change the background color of a table or cell.
- background - image to add an image to the background of a table or cell.

Table specific properties

CSS defined some properties that can only be used with tables. They are

Property	Purpose
border-collapse	Where the borders of two table cells touch, this property indicates whether *both* borders should be visible, or whether the browser should pick just one of the borders to show.
border-spacing	Specifies the width of the space that should appear between table cells.
caption-side	Specifies which side of a table the caption should appear on.
empty-cells	Specifies whether the border should be shown if a cell is empty.
table-layout	If the space you have allocated for a table is not enough to fit the contents, browsers will often increase the size of the table to fit the content in — this property can force a table to use the dimensions you specify.

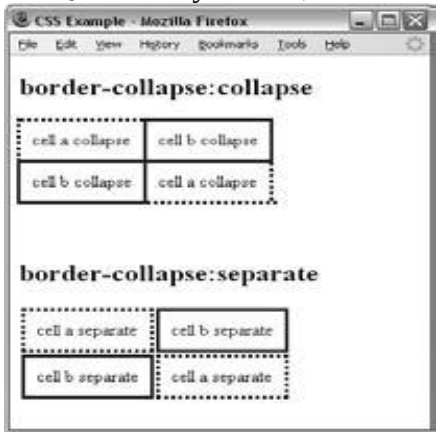
The border - collapse Property

When two table cells meet, we can tell the browser to show just one of the borders. We can do this using the border - collapse property, which can take two values.

Value	Purpose
collapse	Horizontal borders will be collapsed and vertical borders will abut one another.
separate	Separate rules are observed. This value opens up additional properties to give you further control.

For example

```
table.one {border-collapse:collapse;}
table.two {border-collapse:separate;}
td.a {border-style:dotted; border-width:3px; border-color:#000000; padding:10px;}
td.b {border-style:solid; border-width:3px; border-color:#333333; padding:10px;}
```



The border - spacing Property

The border - spacing property specifies the distance that separates adjacent cell's borders. If we provide one value, it will apply to both vertical and horizontal borders.

table.one {border-spacing:15px;}

We can also can specify two values, in which case the first refers to the horizontal spacing and the second to the vertical spacing:

table.two {border-spacing:2px 4px;}

The empty - cells Property

The empty - cells property indicates whether a cell without any content should have a border displayed. It can take one of two values.

Value	Purpose
show	Borders will be shown even if the cell is empty (this is the default value).
hide	Borders will be hidden if cell is empty.

CSS Code

table { width:350px; border-collapse:separate; empty-cells:hide;}

td {padding:5px; border-style:solid; border-width:1px; border-color:#999999;}

HTML Code

< table > < tr > < th > < /th >

< th > Title one < /th > < th > Title two < /th > < /tr >

< tr > < th > Row Title < /th > < td > value < /td > < td > value < /td >

< /tr >

< tr > < th > Row Title < /th > < td > value < /td > < td > < /td > < /tr >

< /table >

The caption - side Property

The caption - side property is for use with the < caption > element to indicate on which side of the table the caption should go. The values for the caption-side property are

Value	Purpose
top	The caption will appear above the table (the default).
right	The caption will appear to the right of the table.
bottom	The caption will appear below the table.
left	The caption will appear on the left side of the table.

For example

caption {caption-side:bottom}

The table - layout Property

The table-layout property sets the table layout algorithm to be used for a table.

Value	Description
auto	Automatic table layout algorithm (this is default): • The column width is set by the widest unbreakable content in the cells • Can be slow, since it needs to read through all the content in the table, before determining the final layout
fixed	Fixed table layout algorithm: • The horizontal layout only depends on the table's width and the width of the columns, not the contents of the cells • Allows a browser to lay out the table faster than the automatic table layout • The browser can begin to display the table once the first row has been received

Understanding Page Layout and Page Size

i. Page Layout:

The look and feel of the web site is generally depends of the style and layout of web page. So we must give more importance on design and construction of the page layout. While designing the page it is important to consider following things.

- Purpose or aim of the site.
- Size of the page.
- Contents on the page.
- Position of each content on the page.

Understanding the Site

Before designing the web site, it is necessary for us to know whether the site is for ourselves or for the client. The web site designer must keep the following questions in mind.

- Who are the visitors for the site?
- What kind of information the clients want from the site?
- How often they visit our site?
- What you want your site to do for the visitors?

Understanding a site's Aims

For knowing the purpose of the website, following questions can be asked-

- Is the website for product or service?
- How often people need this product or service?
- After placing the order for the product or service, do people visit the site again for getting more information?
- If the product/ service is new to the people or promoting it?
- If the people have already heard about it or explaining it in detail?
- Is the site provide on-line sell facility or through the sales person?

The aim of designing the web site can be anything, but it is necessary to know the aim of the web site before the actual design. At the beginning of each new site we should try to list all of the aims. After doing so we can go for structuring the information.

Understanding kind of Visitors

It is equally important to keep it in mind that the design of the web site is for the target audience and not for the clients. Hence ask the following questions.

- Who will be visiting the site? The customers of the website are such as Investors, students, hobbyists, press, researchers and so on.
- What is their purpose of visiting the site?
- What we know about these visitors?

New Content

The contents of the web site must be updated and latest. For example – If we are designing the web site for the books or a product then it is important to put the updates on the web site for new book or product release.

Updating the web site is a time consuming process. Hence there should be somebody to take the responsibility of this task because it should be done on regular basis.

Defining Site's Content

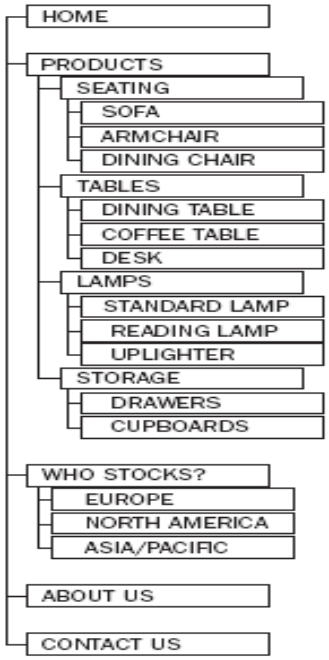
- Deciding the contents for the website is the immediate next step after understanding the goals/aims.
- The list of the contents for the website includes information about the product or service, photos, description, dimensions, features, examples, how it works, contact details and some information about the company.
- The web site designers must also look at the other sites that address the similar topic and try to understand what others do well.
- Add the things like Logo, or branding to most of the pages for advertising purpose.
- Add the copyright notice, terms and conditions and a privacy policy.
- Use various ideas to place the contents on the page to attract the visitors. Unused ideas can be used in future updated of the site.

Categorization or Grouping of Contents

- After deciding the contents for the web site, it is necessary to group them in various categories.
- For instance, the information of the company, its history, how it is formed, its current status and so on can be grouped in 'About Us' category. The phone numbers, email addresses, fax, opening hours, maps-all such information can be grouped in 'Content Us' category.
- Most of the sites have six to seven sections in which all the contents are put together. For example product, purchases, enquiries, About Us, Content Us and so on. These sections are called 'primary or global navigation' of the items.
- Some of the sections might contain the subsections contain more detailed information. These subsections are also called as 'secondary or category navigation'.
- These categories and subcategories are like table of contents and will form the basis for the navigation of our site.
- If the web site is organized properly then it is very easy for users to find what they are looking for. This method of grouping the contents will make the site easier and to navigate better.

Creating a Site Map

- Creating a site map is the next step after deciding the sections and subsections of the web site. The concept of site map is similar to folder list in windows explorer.
- If the contents are arranged in categories and subcategories then the subcategory pages should appear as child to the category page. For example if 'product' is the main category then 'mobile phones, laptops, computer accessories' should appear as "child" in the site map.
- After creating the site map we will know following things:
    - Number of pages on the web site.
    - Information appearing on each page.
    - The links appearing on each page for navigation.
- The site map must be verified against the user's expectations. Also check how users navigate through the site map, step by step to get the information that they need.



Identifying key elements for Every Page

- Before we start actual design, we need to perform a last step and that is deciding the key elements of web page.
- The key elements are those elements that must appear on each page.
- These elements are company logo, name, branding, search box, heading, main contents, primary and secondary navigation, and so on.
- These elements will reflect the purpose of the web site. Good design and simple navigation is basic requirement of any web site.

ii. Page size :

Before the actual design of the web page it is essential to decide its size.

Hence even if the web page that looks good at developer's place perhaps may not look good at user's place as different visitors have different size monitors. By considering this factor, it is necessary to decide the sensible dimensions for the page. Hence while deciding the width of the page, the web designer must think from visitor's point of view. The design should be such that it can be used for most of the monitors.

There are two different techniques for determining the size of the web page.

1. Fixed Width design
2. Liquid Design

Fixed width vs. liquid designs

The design which is based on the viewport rather than the contents of the page are called the **"liquid design"**. The liquid lay out will move in and out to 'resize' the contents in the browser window.

In contrast, the design which is independent of the viewport and has fixed width or height is called **'fixed width design'**. A fixed lay out will 'not move in' and out when we resize our browser window.

Advantages of liquid design

1. The web page gets expanded to fill browser window. Hence there will be 'no space around' the page.
2. If the user opens a 'small browser' window then the page gets compressed and gets 'fitted within the browser' window
3. The design is tolerant of the user's settings.

Disadvantages of liquid design

1. Sometimes the page might 'look' much 'different' than intended.

Advantages of fixed width design

- Accurate pixel values which are useful for controlling the width and positioning the elements.
- The designer can have 'control' over the appearance and position of the elements on the page
- The 'size' of the image will remain the 'same' relative to the page.
- The 'lengths of lines' of text can be 'controlled easily regardless of the user's window.

Disadvantages of fixed width design

1. If the font size is too large then text might not fit within the allotted space.
2. The design works only on the devices that have similar size and resolution of desktop computers. The design can not be displayed properly on the other devices such as PDA, mobile and so on.

3. There can be big gaps around the page.

Designing pages

During the design of the pages, the contents are fit on the page. Some contents are grouped together and their positions are decided. All this is before building of the first page.

Sketching the placement of elements

- The sketching of the elements is done by some blocks of the spaces and few lines of text.
- At this stage we will not consider about the color, background color, font,image and other related issues.
- After creating the wireframe model, you can check the visitor for your site and can make sure that they can easily find whatever they want.
- Preparing wireframe model is important for following two reasons:
    1. in fully designed web site, the user focuses more on look and feel. But in wireframe model they focus on the accessibility and functionality of the elements.
    2. secondly, if the user suggests some changes is a complex task. But of course, you can make the necessary changes is a complex task. But of course, you can make the necessary changes quite easly in the wireframe model.
- Most of the required functions should be at the places in the main page of the site. Generally following items are of top priority and should be placed on the main page-
    - Brand name.
    - Company logo.
    - Title of the site.
    - Global navigation.
    - Secondary or subsection navigation.
    - The contents that tell us the purpose of the page.
    - Option for 'search' the site.
    - Advertisement for self/new arrival.
- Following items should not appear at the front glances on the page.
    - The detailed information about rest of the pages.
    - Links or information about other sites.
    - Advertising.
    - Copyright, terms and conditions, privacy policies must appear at the bottom of the web page.
    - Visitors visit the site to look for something. Hence focus the necessary elements appropriately so that the visitor can quickly find out what they want. It is observed that people generally scan the page in F shape. That means scan the page in two horizontal stripes and then in vertical stripe.

The heading of the page should be clear and concise and text should be easy to scan.

Many times the sites get updated very often. Hence there should be some space allotted for regular changing contents.

Introducing Style

- Adding style means using various colors, background colors, various types of fonts, images and so on.
- If you are designing the web page for some company which has '**logo**', then you must use the same logo. If the company has some **color** representing its identity then you must use that color scheme for web pages. Avoid making changes in these company identities.
- Make use of **digital copy of logo** if any.
- Make use of the '**photographs**' of products or the work done for previous customer along with some text. If the client can supply good photographs then the site will look more professional.
- There should be '**consistency in the design'**. This helps the user to learn how to use the site more quickly.
- There are **header and footer** for every page. The 'header' is generally used to describe the heading of any page on the site, which remains consists across the pages of the web site. Many often logo and primary navigation is a part of header. The 'footer' can contain anything that should appear at the bottom of any page. Between header and footer the content of the webpage are residing.
- The '**use of images**' such as great logos, graphics and photographs make the site more attractive. But remember that these images must be of high quality. Before using these images.
- While '**grouping**' the several elements on the page consider following aspects-
    1. Location: Similar elements must be close to each other.
    2. Color: Make use of similar control wizards(e.g. buttons, links) for navigation of the items.
    3. styles: Make use of similar control wizards(e.g. buttons, links) for navigation of the items.
    4. Borders and Padding: Create appropriate gap between the elements so that they can grouped appropriately.

Navigation

- Placement of navigation should be decided very early while you design the web page. Avoid making use of more than ten primary links on the page.
- **Top Navigation**: It is very common practice to place the navigation directly under the header. It navigation is typically aligned at the left or at the center of the page.
- **Left Navigation**: Placing the navigation at the left is less common. But there are some sites which place the primary navigation at the left side.
- **Top and Left Navigation**: Many sites make use of both top and left navigation. Generally, the primary navigation is at the top and the secondary or sub navigation is at the left.
- **Bottom Navigation**: Avoid putting the primary navigation links at the bottom because there is no guarantee that these links will be in view when the page gets loaded. However, you can place the copyright statement, privacy, policy and terms and conditions links at the bottom.

- **Right Navigation**: It is very rare to have right navigation. This is because, there are chances that user may ignore these links with narrow browser window. More often, the right hand column is for additional contents and links to other part of the site.

Home

- Home page is the first page of the web site. The visitor should be able to understand the main purpose of the site using this home page. For this purpose, company's logo and one sentence tag line can also be added on this page.
- The tasks that user most likely to use must be emphasized on this page. So that people can find the desired information quickly.
- On the home page, all the headings and link names must begin with important keywords. This will enable the visitor to understand the sections or link.

Content Pages

- The content pages should display the contents in such a way that it makes user to read the information easily.
- Place the equivalent information on the same place on each page.
- The page should not be overloaded with information. There should be a clean presentation of information. If you need to display a lot of information, then there should be separation between different group of information.
- Image should be related to product, service or topic. These should be left or right aligned with the text following around these images.
- If the web site is for product sell, then the page should be action oriented –that means user must be able to select particular item or browse through catalog and so on.
- The text should not spread too widely across the pages. All the text must be split in appropriate ate heading and subheadings. The paragraphs should be short.

Structuring Pages

- Structuring pages means translating the wireframe design into the usable and attractive web site. In this process the design is translated into the code.
- If you want to specify some order to the elements that appear on the web page then you need to create a layout that has more than one column. For instance : The first left column may contains some links and the second column can display the main contents.
- Professional designers layout the pages using grids (i.e. set of rows and columns). The rows and columns of this grid can be of any height and width.

Single Column Layout

- The single column layout is the easiest way of structuring the page. It is ideal for the smaller web sites. Single column layout normally have three to four rows.
- Normally, the first row contains the logo of the company, the second row contains the navigational links and the third row contains the main contents

and fourth row might contain the copyright statements, privacy, policies and terms and conditions.

- For this type of structuring it is necessary to control the width of the text. Using CSS, the width property can be set for the page. Similarly the left and right margins can be set by assigning 'auto to margin-left and margin-right properties.
- Along with these properties the text-align property can also be used for aligning the contents of the page.
- For example – Following CSS style can be used to layout the contents of the page.

Two Column Layout

- Two column layout usually contains one of the following scenario –
- The left hand column contains the navigation and the second next column contains the main contents of the page.
- The left-hand column contains the main contents of the page and the right hand contains the related contents or advertisements ets.
- The heading can span both the columns of this page.

- The float property of the CSS can be used to adjust the contents in the two columns. Along with float property it is necessary to specify the width of each block; otherwise contents can occupy the full width of its containing block.
- Different background-color or the shading of the two columns can be added to differentiate the two columns.

Three Columns Layout

Three columns layout usually contains following scenario

- The left hand column contains the navigation.
- The second next column contains the main contents.
- The third column contains the related links, advertisements and so on.

Sacrificial column

The sacrificial column is normally the third right hand column. The contents of this column help the visitor to understand the site in better manner.

- Event-if the user's browser does not show the sacrificial column within the width of the window the user does not miss the meaning of the page because the main contents of the page are always displayed to him/her.

Advanced Layout using CSS

It is common to use Cascading Style Sheet (CSS) to control the layout of the page.
There are various sites that help to create attractive layouts using CSS.

Creating a Layout using Nested Tables

- Before the use of CSS, use nested tables for structuring the web page was a common practice.
- One table can used to layout the entire page. For instance : The header can be in one row, the main contents can be in another row, the copyright statements can be in some bottom-most row.
- The use of nested table allows the page to occupy the whole browser window but at the same time to control the width of the content itself.
- The varying background colors, text and images can be used by the tables.
- Having nested tables with fixed width inside the row ensures the contents of the page to remain as of fixed width.
- Note that while working with tables a cell must wholly contain the elements. The element must not straddle two table cells.

Design Issues in Structuring Web Pages

The design issues affect the specific parts of the web page. These are –Text, menu, tables and forms. We will discuss each of these issues one by one.

i. Text

Various issues regarding the placement of text, spacing of text and regarding the font are

- **Adding the white space in the page makes it more attractive** :
  - The page with space between navigation, text, images and other items make it more readable and attractive. This space is called as white space.
  - Using the padding and margin properties of table the space can be created between the elements. Adding white space in the text doesn't mean to have white colored space only. You can simply use different shades of the background for identifying the contents. using the CSS we can add extra space in the web page.

- **Careful alignment of the text makes it readable :**
  - Mostly the left aligned text is more readable; however you can have the right-aligned or centrally aligned text. Generally the paragraphs are left aligned.

- **Columns should not be too wide** :
  - Most of the people find it heard to read the long line on computer screen. It makes it hard to follow the correct line if it is too long. Hence limit the width of the column of text.

- **There should not be background images for the text because then the text becomes hard to read**
  - If you to use background images then make sure that the text will stand out of it. Do not make use of photographs as the background images

- **Increased vertical space makes text more readable:**

        - If there is large amount of data, then increase the amount of space between the lines of text. Following are the two reasons why the increased vertical space is helpful in the web page –
    - When you come to an end of a line it becomes easy to find the start of the line.
    - People often scan the page instead of reading it. Extra space makes it easier to read the text quickly.
    -
- **Fonts must be chosen carefully :**
    - When choosing which font to use on the web page make sure that the selected font is installed on the user's computer otherwise the whole design will be of no use. The common list of fonts with preferences that are applied normally is-
        - Sans- serif : Helvetica, Arial, sans-serif.
        - Serif : Times new roman, times serif
        - Mono Space : courier new , courier mono space.

- **Fixed font size may vary on the screens with different resolutions :**
    - Some times the fonts with 12px high will look very small on the screen with resolution. 1280 X 1024 than 800 X 600. using fixed size fonts also makes it harder to read it on the screen with high resolution.

ii. Navigation

Navigation through the web site is the most important factor. It helps the people to find the information quickly and as easily as possible. User navigate through the web site by following ways

- Using menus
- By browsing links
- Searching for relevant items.

iii. Menus

Following are some guidelines that must be used while designing menus for the web page.

- Menus must focus on what user wants to achieve.
- Menus must be concise.
- Menus must be quick and easy to read.
- menus should load efficiently.
- Menu items must be easy to select.
- Grouping of menus must be logical.
- Menus must be consistent.
- If icons are used then those icons must be easy to understand. Add text links as well.

iv. Links

There are two types of links that are normally used.

1. Text link
2. Image Link.

By default the text link are blue and under lined. Make use of different color to different the text link from the main text. We can change the appearance of the links when user hovers over them and when the user already visited them.
The images can also be used as links. They are used in advertisements, for photo graphs to click on, graphics icons and so on.

v. Searching for relevant items

- Using search feature user can navigate through the site. Using the keyword the user can search for information which he wants
- Searching can save user from learning of navigation scheme.
- Search feature is an essential for the larger sites.
- Large commercial sites where the content is stored in database can use programming queries for the finding the desired contents. Some times make use of indexing techniques to index your site. These services allow using a search box and sending a query to service's site.
- Google offers a very powerful and flexible service whereby you can use the search engine to provide the search feature to your site.

vi. Tables

- If the table contains multiple rows, then make use of alternate back ground colors for the tables.
- there must be good contrast between the background color and the text for the table. This enhances the readability of the table.

vii. Forms

If the website has a form which is to be filled by the user when they must be good design for it. Following issues must be considered for having form on the web site

- what to do before designing the form.
- How to design the form.
- how to have best layout for the form.

Before designing the form

- Following guidelines must be used before designing the form
- List the information that is needed for the form. For instance: Login information, user's name, user's address, user's contact details and so on.
- Then whatever information is collected just try to make the logical grouping of that information, sothat some items can go together. For example name, phone number and email address, login details and so on.

- For the online application create a form which is just similar paper form. Due to this user can fill such online forms easily.
- User need not have to give the same information again and again. For instance: once the user is registered to your site, his name , address and other relevant details must appear on the web page. He need not have to fill it up again.
- Along with this basic information, there is some other information that might appear on the form. For example: price list , legal notice, options for payment, shipping details and so on.

Designing the form

- After understanding what information must be captured by the form is the time to design the form. For actual design, the appropriate controls must be selected and grouped together. These controls must be labeled appropriately.
- There are various form controls such as text boxes, text areas, radio buttons, check boxes and so on.
- If you wish to enter single line text then make use of <input type =text> element.
    - If you wish to enter multi line text then make use of <text area> element.
    - If user wants to select only one option from the several one use radio buttons or drop down select box.
    - Check box can be used to select more than one items.
- These controls must be named appropriately so that the contents of these controls can be understood. For instance instead of naming value1, make use of the name txtvalue.
- After deciding what controls to be used for the form, can you start putting them on the form. The related information can be grouped together. This will enhance the users understanding of filling up the form.
- The form elements can be grouped in following way.
    - Field sets
    - Labels
    - Splitting the form in across several pages.
- The <fieldset> element allows to make one entity by grouping some controls. The controls can also be separated out by adding a line, styles are background colors.
- Some times user has to fill up a very long form. This makes the code more complicated. In such a case, the form is split into several pages. Following are the reasons why the form is split across multiple pages.
    - Smaller forms are easy to fill up.
    - If all the related information is put on same page, then the user can find it easy to fill up.
- While filling up the form, user need not have to scroll up a lot. If there are multiple pages for the form, then the page numbering must be displayed to the user, so that the user can jump to desired section of the form very easily.

Layout of Forms

- Layout of the form is based on the expectation. Each control must be labeled clearly so that the user can understand where to add the information. Following guideline is generally used for labeling the controls.
    - The labels are to left of the input or directly above the text box.
    - The labels are to the right of the check box or radio button.
    - The label must be directly on the button.
- Many designers and programmer use tables to layout and position the form elements on the web page. However normally use of CSS is more beneficial than using tables. This is because by using CSS the form elements can be controlled in better manner.
- Always keep the relevant information next to or above the form controls.
- If it is compulsory that some fields must be filled up by the user then the asterisk or * can be used in front of the required fields.
- The buttons must be placed carefully on the web page. It must be closed to the relevant part of the form. For example the online reservation, after filling up the form the form submit button should at the bottom of the page.
- There should be next, proceed or submit buttons on the form.
- The title attribute is used for the form controls. This attribute provides a tool tip for the form controls, by which the user can understand how to fill up the form.
- For the form, the user must be able to make use of the tab key on the keyboard in order to move from one field to another. This is known as tab index.
- Do not use color to the convey the information. For example if some field is compulsory to get filled then instead of using different colors for this filled make use of asterisk.
- The stylish form can be created by using the CSS styles. Hence use of CSS for form design is a poplar option.
- After creating the form it is equally important to test the form thoroughly.

UNIT - 4

Java Script

JavaScript is the premier client-side ***interpreted scripting language***. It's widely used in tasks ranging from the validation of form data to the creation of complex user interfaces. ***Dynamic HTML*** is a combination of the content formatted using HTML, CSS, Scripting language and DOM. By combining all of these technologies, we can create interesting and interactive websites.

History of JavaScript:

Netscape initially introduced the language under the name **LiveScript** in an early beta release of Navigator 2.0 in 1995, and the focus was on form validation. After that, the language was renamed JavaScript. After Netscape introduced JavaScript in version 2.0 of their browser, Microsoft introduced a clone of JavaScript called **JScript** in Internet Explorer 3.0.

What a JavaScript can do?

JavaScript gives web developers a programming language for use in web pages & allows them to do the following:

→JavaScript gives HTML designers a programming tool

→JavaScript can be used to validate data

→ JavaScript can read and write HTML elements

→Create pop-up windows

→Perform mathematical calculations on data

→React to events, such as a user rolling over an image or clicking a button

→Retrieve the current date and time from a user's computer or the last time a document was modified

→Determine the user's screen size, browser version, or screen resolution

→JavaScript can put dynamic text into an HTML page

→JavaScript can be used to create cookies

Advantages of JavaScript:

→Less server interaction

→ Immediate feedback to the visitors

→ Increased interactivity

→ Richer interfaces

→ Web surfers don't need a special plug-in to use your scripts

→ Java Script is relatively secure.

Limitations of JavaScript:

We cannot treat JavaScript as a full-fledged programming language. It lacks some of the important features like:

→ Client-side JavaScript does not allow the reading or writing of files. This has been kept for security reason.

→ JavaScript cannot be used for networking applications because there is no such support available.

→ JavaScript doesn't have any multithreading or multiprocess capabilities.

→ If your script doesn't work then your page is useless.

Points to remember:

JavaScript is case-sensitive

Each line of code is terminated by a semicolon

Variables are declared using the keyword **var**

Scripts require neither a **main** function nor an **exit** condition. There are major differences between scripts and *proper* programs. Execution of a script starts with the first line of code & runs until there is no more code

#JavaScript comments:

In JavaScript, each line of comment is preceded by two slashes and continues from that point to the end of the line.

//this is javascript comment

Block comments or Multiline comments: **/*** ***/**

JavaScript is not the same as Java, which is a bigger programming language (although there are some similarities)

JavaScript and HTML Page

Having written some JavaScript, we need to include it in an HTML page. We can't execute these scripts from a command line, as the interpreter is part of the browser. The script is included in the web page and run by the browser, usually as soon as the page has been loaded. The browser is able to debug the script and can display errors.

Embedding JavaScript in HTML file:

If we are writing small scripts, or only use our scripts in few pages, then the easiest way is to include the script in the HTML code. The syntax is shown below:

```
<html>
    <head>
    <script language="javascript">
    <!- -
        Javascript code here
    //- - >
    </head>
    <body>
        ……
    </body>
</html>
```

Using External JavaScript in HTML file:

If we use lot of scripts, or our scripts are complex then including the code inside the web page will make the source file difficult to read and debug. A better idea is to put the JavaScript code in a separate file and include that code in the HTML file. By convention, JavaScript programs are stored in files with the **.js** extension.

```
<html>
    <head>
    <script language=”javascript” src=”sample.js”> </script>
    </head>
    <body>
        ……
    </body>
</html>
```

POPUP BOXES IN JAVASCRIPT

alert(“string”)	opens box containing the message
confirm(“string”)	displays a message box with OK and CANCEL buttons
prompt(“string”)	displays a prompt window with field for the user to enter a text string

Example:

```
<html>
<head>
    <script language="javascript">
    function show_alert()
    {
        alert("Hi! This is alert box!!");
    }
    </script>
</head>
<body>
    <input type="button" onclick="show_alert()" value=“Display alert box" > </input>
</body>
</html>
```

JavaScript Programming Elements

- Variables, datatypes, operators
- Statements
- Arrays
- Functions
- Objects in JavaScript
- Exception Handling
- Events
- Dynamic HTML with JavaScript

VARIABLES

Like any programming language, JavaScript has variables. A variable is a name assigned to computer memory location to store data. As the name suggests, the value of the variable can vary, as the program runs. We can create a variable with the **var** statement:

var <variablename> = <some value>;

Example:

```
var sum = 0;

var str;
```

We can initialize a variable like this:

```
str = "hello";
```

Rules for variable names:

They must begin with a letter, digit or underscore character # We can't use spaces in names

Variable names are case sensitive # We can't use reserved word as a variable name.

Weakly Typed Language:

→ Most high-level languages, including C and Java, are ***strongly typed*.** That is, a variable must be declared before it is used, and its type must be included in its declaration. Once a variable is declared, its type cannot be changed.

→ As the JavaScript is ***weakly typed*** language, data types are not explicitly declared.

Example:

```
var  num;

num = 3;

num = "San Diego";
```

First, when the variable ***num*** is declared, it is empty. Its data type is actually the type **undefined**. Then we assign it to the number 3, so its data type is **numeric**. Next we reassign it to the string "San Diego", so the variable's type is now **string.**

Example:

```
<html>
<body>
    <script language="javascript" type="text/javascript">
        var  s;
        s = "Hello";
        alert(typeof s);
        s = 54321;
        alert(typeof s);
    </script>
</body>
</html>
```

DATATYPES

- JavaScript supports five primitive data types:

  number string boolean undefined null.

- These types are referred to as ***primitive types*** because they are the basic building blocks from which more complex types can be built.
- Of the five, only **number, string**, and **boolean** are real data types in the sense of actually storing data.
- Undefined and null are types that arise under special circumstances.

Numeric Data Type:

→These are numbers and can be integers (such as 2, 22 and 2000) or floating-point values (such as 23.42, -56.01, and 2E45).

→ Valid ways to specify numbers in JavaScript

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

→We can represent integers in one of the following 3 ways:

Decimal: The usual numbers which are having the base 10 are the decimal numbers

Octal: Octal literals begin with a leading zero, and they consist of digits from zero through seven. The following are all valid octal literals:

00 0777 024513600

HexaDecimal: Hexadecimal literals begin with a leading 0x, and they consist of digits from 0 through 9 and letters A through F. The following are all valid hexadecimal literals:

0x0 0XF8f00 0x1a3C5e7

Special Numeric Values:

Special value	Result of	Comparisions
Infinity, -Infinity	Number too large or too small to represent	All infinity values compare equal to each other
NaN	Undefined Operation	NaN never compares equal to anything, even to itself

Example Operations for these special values are

- Division of a positive number by zero results in "**Infinity**" .
- Division of a negative number by zero results in "**- Infinity**"
- Division of zero by zero is "undefined" (**NaN)**.
- The **isNaN()** method returns a Boolean indicating whether the value is **NaN**

String:

String means set of characters. In JavaScript, strings are enclosed in a pair of single quotes or double quotes. The value of a string can even contain spaces and may be totally made from digits. All of the following are strings:

"Chris" "Chris Bates" "234.56"

Boolean:

Boolean variables hold the values **true** and **false**. These are used to hold the result of conditional tests.

Example:

```
var   flag= true;
//set of statements
if(flag)
{
     x = x+1;
}
```

Undefined:

The ***undefined*** type is used for variables or object properties that either do not exist or have not been assigned a value. For example, declaring a variable without assigning it a value,

```
var  x;
```

Gives ***x*** the **undefined** type and value.

```
var x = String.noSuchProperty;
```

Also results in the assignment of **undefined** to ***x***.

Datatype Conversion

- 3 + 3 // results in 6
- 3 + "3" // results in "33"
- 3 + 3 + "3" /results in "63"

Converting String to Number

- parseInt("33"); // results in 33
- parseInt("33.33); // results in 33
- parseFloat("33"); // results in 33
- parseFloat("33.33"); // results in 33.33
- 3 + 3 + parseInt("3"); // results in 9

Converting Number to String

- "" + 2500 // results in "2500"
- ("" + 2500).length // results in 4
-

Keywords

Keywords are the words which are part of JavaScript language syntax & have special meaning. These may not be used when choosing identifier names for variables, functions, or labels

break	else	new	var
case	finally	return	void
catch	for	switch	while
continue	function	this	with
default	if	throw	
delete	in	try	
do	instanceof	typeof	

Reserved Words:

JavaScript reserves all the words listed below for future inclusion into the JavaScript syntax – so these too may not be used as identifier names.

```
abstract        enum        int         short
boolean         export      interface   static
byte            extends     long        super
char            final       native      synchronized
class           float       package     throws
const           goto        private     transient
debugger        implements  protected   volatile
double          import      public
```

Operators in JavaScript

The operators in JavaScript can be classified as follows:

→Arithmetic operators

→Relational operators

→Logical operators

→Assignment operators

Arithmetic operators:

Operator	Description	Example	Result
+	Addition	x=2 y=2 x+y	4
-	Subtraction	x=5 y=2 x-y	3
*	Multiplication	x=5 y=4 x*y	20
/	Division	15/5 5/2	3 2.5
%	Modulus (division remainder)	5%2 10%8 10%2	1 2 0
++	Increment	x=5 x++	x=6
--	Decrement	x=5 x--	x=4

Note: If the arguments of + are numbers then they are added together. If the arguments are strings then they are concatenated and result is returned.

Example:

```
<body>
<script language="JavaScript">
<!--
var a = 5;
++a;
alert("The value of a = " + a );
-->
</script>
</body>
</html>
```

String (+) Operator:

Example:

txt1="Welcome";

txt2="to L&T Infotech Ltd.!";

txt3=txt1 + " " + txt2;

(or)

txt1="Welcome ";

txt2="to L&T Infotech Ltd.!";

txt3=txt1 + txt2;

Relational operators/Comparison operators: Relational operators are used to compare quantities.

Operator	Description	Example
==	is equal to	5==8 returns false
===	is equal to (checks for both value and type)	x=5 y="5" x==y returns true x===y returns false
!=	is not equal	5!=8 returns true
>	is greater than	5>8 returns false
<	is less than	5<8 returns true
>=	is greater than or equal to	5>=8 returns false
<=	is less than or equal to	5<=8 returns true

Conditional Operator: Conditional operator is one the JavaScript's comparison operator, which assigns a value to a variable based on some condition.

Syntax :

variablename=(condition)? value1 : value2;

Logical operators: Logical operators are used to combine two or more conditions.

Operator	Description	Example
&&	and	x=6 y=3 (x < 10 && y > 1) returns true
||	or	x=6 y=3 (x==5 || y==5) returns false
!	not	x=6 y=3 !(x==y) returns true

Example (Logical operators):

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

Assignment Operators: are used to assign the result of an expression to a variable.

Operator	Example	It is Same as
=	x=y	x=y
+=	x+=y	x=x+y
-=	x-=y	x=x-y
*=	x*=y	x=x*y
/=	x/=y	x=x/y
%=	x%=y	x=x%y

The *typeof* operator

typeof operator is used to verify the type of the variable or value.

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

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

- "undefined" if the variable is of the Undefined type.
- "boolean" if the variable is of the Boolean type.
- "number" if the variable is of the Number type.
- "string" if the variable is of the String type.
- "object" if the variable is of a reference type or of the Null type

Example:

```
var  s = "test string";
alert(typeof  s);            //outputs "string"
alert(typeof  95);           //outputs "number"
alert(typeof window);    //outputs "object"
```

STATEMENTS

Programs are composed of two things : data and code (set of statements) which manipulates the data. Java script Statements can be divided into the following categories:

- Conditional Statements
- Looping Statements
- Jumping Statements

Conditional statements: Conditional statements are used to make decisions.

Various conditional statements in JavaScript:

→Various forms of **if**

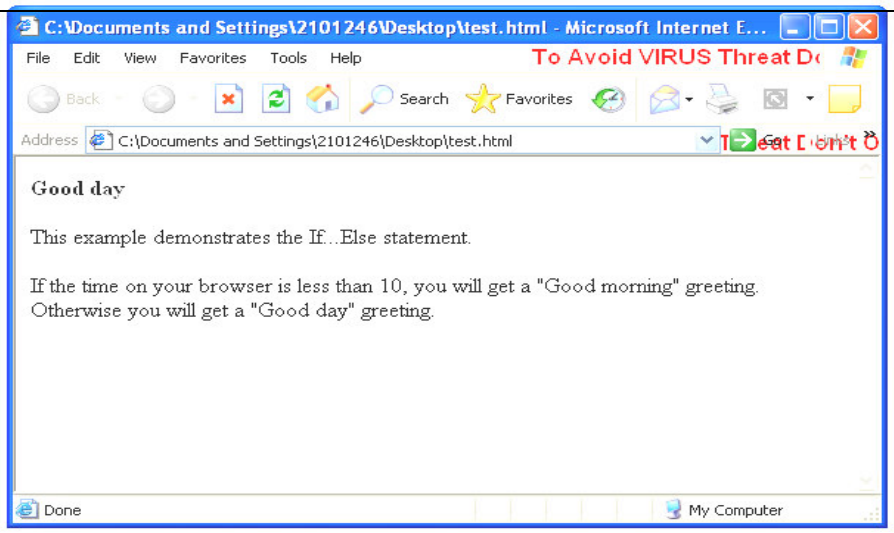
→switch

Various forms of if:

Simple if
if-else Statement
nested if
else-if ladder

Example:

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



switch statement:

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

Syntax

```
switch (expression)
{
        case  label1:    block1
                         break;
        case  label2:    block2
                         break;
         ….
        default:         def block;
}
```

Example:

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

Looping statements: Loops are used to execute certain statements repeatedly. The various loops in JavaScript are:

- **for** loop
- **while** loop
- **do-while** loop

for loop syntax:

```
for(initialization; condition; loop variable update)
{
    Set of statements
}
```

Example (for loop):

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

for- in loop:

The **for-in** loop has a special use to enumerate all the properties contained within an object. This loop is rarely used in regular JavaScript.

Example: Display **window** object properties

```
var  i, a = "";
for( i  in window)
        a +=  i  +  "…";
alert(a);
```

while loop: executes the statements as long as the condition is true.

Syntax:

```
while(condition)
    {
        Set of statements
    }
```

Example:

```
<html>
<body>
<script language="javascript">
```

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

do-while syntax:

```
        do
        {
                Set of statements
        }while(condition);
```

Jumping Statements:

break statement:

break statement is used to terminate a loop, switch, or label statement.

When we use break without a label, it terminates the innermost enclosing while, do-while, for, or switch immediately and transfers control to the following statement.

When we use break with a label, it terminates the specified labeled statement

Syntax:

- **break;**
- **break *label*;**

continue statement:

When we use **continue** without a label, it terminates the current iteration of the innermost enclosing while, do-while or for statement and continues execution of the loop with the next iteration.

When we use **continue** with a label, it applies to the looping statement identified with that label.

Syntax:

- **continue;**
- **continue *label*;**

WORKING WITH ARRAYS

An array is an ordered set of data elements which can be accessed through a single variable name. In many programming languages arrays are contiguous areas of memory which means that the first element is physically located next to the second and so on. In JavaScript, an array is slightly different because it is a special type of object and has functionality which is not normally available in other languages.

Basic Array Functions: The basic operations that are performed on arrays are creation, addition of elements (inserting elements), accessing individual elements, removing elements.

Creating Arrays: We can create arrays in several ways:

- var arrayObjectName = [element0, element1, ..., element N];
- var arrayObjectName = new Array(element0, element1, ..., element N);
- var arrayObjectName = new Array(arrayLength);

Ex:

- var colors = ["Red", "Green", "Blue"];
- var colors = new Array("Red", "Green", "Blue");
- var colors = Array("Red", "Green", "Blue");
- var thirdArray = [,,,,];
- var fourthArray = [,,35,,,16,,23,];

Note: JavaScript arrays can hold mixed data types as the following example shows:

var a = [“Monday”, 34, 45.7, “Tuesday”];

Accessing Array Elements:

Array elements are accessed through their ***index***. The **length** property can be used to know the length of the array. The index value runs from **0** to **length-1**.

Example:

```
<script language="javascript">
        var a = [1,2,3];
        var s = "";
        for(var i=0;i<a.length;i++)
        {
                s += a[i] + "  ";
        }
        alert(s);
</script>
```

Adding elements to an array:

What happens if we want to add an item to an array which is already full? Many languages struggle with this problem. But JavaScript has a really good solution: ***the interpreter simply extends the array and inserts the new item.***

Ex: var a = [“vit”,”svecw”,”sbsp”];

a[3] = “bvrit”;

a[10] = “bvrice”; //this extends the array and the values of elements a[4] to a[9] will be **undefined**.

Modifying array elements:

Array element values can be modified very easily.

Ex: To change a[1] value to "vdc" simply write:

a[1] = "vdc";

Searching an Array:

To search an array, simply read each element one by one & compare it with the value that we are looking for.

Removing Array Members:

JavaScript doesn't provide a builtin function to remove array element. To do this, we can use the following approach:

→read each element in the array

→if the element is not the one you want to delete, copy it into a temporary array

→if you want to delete the element then do nothing

→increment the loop counter

→repeat the process

→finally store the temporary array reference in the main array variable

Note: The statement **delete a[0]** makes the value of a[0] **undefined**

OBJECT-BASED ARRAY FUNCTIONS:

In JavaScript, an array is an object. So, we can use various member functions of the object to manipulate arrays.

concat()

The **concat()** method returns the array resulting from concatenating argument arrays to the array on which the method was invoked. The original arrays are unaltered by this process.

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

Example:

```
<script language="javascript">
    var a = [1,2,3];
    var b = ["a","b"];
    alert(a.concat(b));
</script>
```

join()

join() method allows to join the array elements as strings separated by given specifier.The original array is unaltered by this process.

Syntax: arrayname.join(separator);

Example:

```
<script language="javascript">
        var a = [1,2,3];
        alert(a.join("#"));
</script>
```

push()

push() function adds one or more elements to the end of an array and returns the last element added.

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

Example:

```
<script language="javascript">
        var a = [1,2,3];
        alert(a.push(4,5));          //displays 5
        alert(a);                    //displays 1,2,3,4,5
</script>
```

pop()

pop() removes the last element from the array and returns that element

Syntax: arrayname.pop();

reverse()

reverse() method transposes the elements of an array: the first array element becomes the last and the last becomes the first. The original array is altered by this process.

Syntax: arrayname.reverse();

Example:

```
<script language="javascript">
        var a = [1,2,3];
        a.reverse();
        alert(a);
</script>
```

shift()

shift() removes the first element of the array and in doing so shortens its length by one. It returns the first element that is removed.

Ex:

```
var a = [1, 2, 3];
var first = a.shift();
 alert(a);        // 2,3
alert(first);     //1
```

unshift()

unshift() adds one or more elements to the front of an array.

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

Example:

```
var a = ["x","y","z"];
a.unshift("p","q");
alert(a);          //p,q,x,y,z
```

slice()

slice() returns a "slice" (subarray) of the array on which it is invoked. The method takes two arguments, the *start* and *end* index, and returns an **array** containing the elements from index *start* up to but not including index *end*. If we specify only first parameter, then array containing the elements from *start* to the end of the array are returned. The original array is unaltered by this process.

Syntax: **a**rrayname.slice(startindex , endindex);

Example:

```
var  a = [1, 2, 3, 4, 5];
a.slice(2);        // returns [3, 4, 5]
a.slice(1, 3);    // returns [2, 3]
```

splice()

The **splice()** method can be used to add, replace, or remove elements of an array in place. Any elements that are removed are returned. It takes a variable number of arguments, the first two arguments are mandatory. The original array is altered by this process.

Syntax : arrayname.splice(*start*, *deleteCount*, *replacevalues*);

The first argument *start* is the index at which to perform the operation.

The second argument is *deleteCount*, the number of elements to delete beginning with index *start*. If we don't want to delete any elements specify this value as 0.

Any further arguments represented by *replacevalues* (that are comma-separated, if more than one) are inserted in place of the deleted elements.

Example:

```
var myArray = [1, 2, 3, 4, 5];
myArray.splice(3,2,"a","b");          // returns 4,5
alert(a);                             //1,2,3,a,b
```

sort()

sort() method sorts the array into lexicographic order. Elements which are not text are converted into strings before the sort operation is performed. This means, for example,732 will be placed before 80 in the sorted array. Original array is altered by this process.

Example1:

```
var myArray = ["vit","svecw","bvrice"];
myArray.sort();
alert(myArray);          //bvrice,svecw,vit
```

Example2:

```
var a = [80,732,450];
a.sort();
alert(a);                //450,732,80
```

****

STRINGS

String is a set of characters enclose in a pair of single quotes or double quotes. In JavaScript using string object, many useful string related functionalities can be done. Some commonly used methods of string object are concatenating two strings, converting the string to uppercase or lowercase, finding the substring of a given string and so on.

PROPERTY:

length

This property of string returns the length of the string

Example: "vit".length //gives 3

METHODS:

charAt(index)

This method returns the character specified by index

Example: alert("vitsvecw".charAt(2)**);** //displays **t**

indexOf(substring [, offset])

This method returns the index of substring found in the main string. If the substring is not found returns **-1**. By default the indexOf() function starts index **0**, however, an optional offset may be specified, so that the search starts from that position.

Example: "vitsvecwsvecw".indexOf("sv"); //returns 3

"vitsvecwsvecw".indexOf("sv",5); //returns 8

lastIndexOf(substring [,offset])

This method returns the index of substring found in the main string (i.e. last occurence). If the substring is not found returns **-1**. By default the lastIndexOf() function starts at index **string.length-1**, however, an optional offset may be specified, so that the search starts from that position in backwards.

Example: "vitsvecwsvecw".lastIndexOf("sv"); //returns 8

"vitsvecwsvecw".lastIndexOf("sv",6); //returns 3

str1.concat(str2 [,str3 ..strN])

This method is used to concatenate strings together. For example, s1.concat(s2) returns the concatenated string of s1 and s2. The original strings don't get altered by this operation.

substring(*start* [,*end*] **)**

This method returns the substring specified by ***start*** and ***end*** indices (upto ***end*** index, but not the character at ***end*** index). If the ***end*** index is not specified, then this method returns substring from ***start*** index to the end of the string.

Example: "vitsvecw".substring(3,6); //returns **sve**

"vitsvecw".substring(3); //returns **svecw**

substr(index [,length])

This method returns substring of specified number of characters (**length**), starting from **index**.

If the length is not specified it returns the entire substring starting from **index**.

Example: "vitsvecw".substr(3,2); //returns **sv**

"vitsvecw".substr(3); //returns **svecw**

toLowerCase( )

returns the string in lower case. The original string is not altered by this operation.

Example:

```
<script language="javascript">
    var s="VITsvecw";
    alert(s.toLowerCase());      //displays  vitsvecw
    alert(s);                    //displays  VITsvecw
</script>
```

toUpperCase( )

returns the string in upper case. The original string is not altered by this operation.

Example:

```
<script language="javascript">
    var s="VITsvecw";
    alert(s.toUpperCase());      //displays  VITSVECW
    alert(s);                    //displays  VITsvecw
</script>
```

split(separator [,limit])

Splits the string based on the ***separator*** specified and returns that array of substrings. If the **limit** is specified only those number of substrings will be returned

Example1:

```
<script language="javascript">
    var s="vit#svecw#bvrice#sbsp";
    var t =s.split("#");
    alert(t);          //displays vit,svecw,bvrice,sbsp
</script>
```

Example2:

```
<script language="javascript">
        var s="vit#svecw#bvrice#sbsp";
        var t =s.split("#");
        alert(t);              //displays vit,svecw,bvrice
    </script>
```

Write Javascript that determines whether the given string is palindrome or not

```
<html>
<body>
<script language="javascript">
        var s = prompt("enter any string");
        var n = s.length;
        var flag=true;
        for(var i=0; i<n;i++)
        {
            if(s.charAt(i) != s.charAt(n-1-i))
            {
                flag=false;   break;
            }
        }
        if(flag)  alert("palendrome");
        else  alert("not palendrome");
</script>
</body>
</html>
```

STRING METHODS USED TO GENERATE HTML:

string.anchor("anchorname")	string.link(url)
string.blink( )	string.big( )
string.bold( )	string.small( )
string.fixed( )	string.strike( )
string.fontcolor(colorvalue)	string.sub( )
string.fontsize(integer 1 to 7)	string.sup( )
string.italics( )	

Example:

```
<script language="javascript">
    var s = "vit".bold();          //s value is:  <b>vit</b>
    document.write(s);
    document.write("<br>");
    document.write("svecw".italics());
</script>
```

anchor ("*name*")	Creates a named anchor specified by the <**A**> element using the argument name as the value of the corresponding attribute.	`var x = "Marked point".anchor("marker");` `// <A NAME="marker">Marked point</A>`
big()	Creates a <**BIG**> element using the provided string.	`var x = "Grow".big();` `// <BIG>Grow</BIG>`
blink()	Creates a blinking text element enclosed by <**BLINK**> out of the provided string despite Internet Explorer's lack of support for the <**BLINK**> element.	`var x = "Bad Netscape".blink();` `// <BLINK>Bad Netscape</BLINK>`
bold()	Creates a bold text element indicated by <**B**> out of the provided string.	`var x = "Behold!".bold();` `// <B>Behold!</B>`
fixed()	Creates a fixed width text element indicated by <**TT**> out of the provided string.	`var x = "Code".fixed();` `// <TT>Code</TT>`

fontcolor (color)	Creates a <**FONT**> tag with the color specified by the argument color. The value passed should be a valid hexadecimal string value or a string specifying a color name.	`var x = "green".font("green");`
		`// <FONT COLOR="green">Green</FONT>`
		`var x = "Red".font("#FF0000");`
		`// <FONT COLOR="#FF0000">Red</FONT>`

Fontsize (*size*)	Takes the argument specified by size that should be either in the range 1–7 or a relative +/– value of 1–7 and creates a <**FONT**> tag.	`var x = "Change size".font(7);`
		`// <FONT SIZE="7">Change size</FONT>`
		`var x = "Change size".font("+1");`
		`// <FONT SIZE="+1">Change size</FONT>`

italics()	Creates an italics element <I>.	`var x = "Special".italics();` `// <I>Special</I>`

Link (location)	Takes the argument location and forms a link with the <**A**> element using the string as the link text.	`var x = "click here".location("http://www.pint.com/");`
		`// <A HERF="http://www.pint.com">`
		`// click here</A>`

small()	Creates a <**SMALL**> element out of the provided string.	`var x = "Shrink".small();` `// <SMALL>Shrink</SMALL>`
strike()	Creates a <**STRIKE**> element out of the provided string.	`var x = "Legal".strike();` `// <STRIKE>Legal</STRIKE>`
Sub()	Creates a subscript element specified by <**SUB**> out of the provided string.	`var x = "test".sub()` `// <SUB>test</SUB>`
Sup()	Creates a superscript element specified by <**SUP**> out of the provided string.	`var x = "test".sup()` `// <SUP>test</SUP>`

FUNCTIONS

A function is a piece of code that performs a specific task. The function will be executed by an event or by a call to that function. We can call a function from anywhere within the page (or even from other pages if the function is embedded in an external **.js** file). JavaScript has a lot of builtin functions.

Defining functions:

JavaScript function definition consists of the **function** keyword, followed by the name of the function.

A list of arguments to the function are enclosed in parentheses and separated by commas. The statements within the function are enclosed in curly braces { }.

Syntax:

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

Parameter Passing:

Not every function accepts parameters. When a function receives a value as a parameter, that value is given a name and can be accessed using that name in the function. The names of parameters are taken from the function definition and are applied in the order in which parameters are passed in.

- Primitive data types are passed by value in JavaScript. This means that a copy is made of a variable when it is passed to a function, so any manipulation of a parameter holding primitive data in the body of the function leaves the value of the original variable untouched.
- Unlike primitive data types, composite types such as arrays and objects are passed by reference rather than value.

Examining the function call:

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

***functionname*.arguments**

This is an array of parameters that have been passed

***functionname*.arguments.length**

This is the number of parameters that have been passed into the function

Example:

```
<html>
<body>
<script language="javascript">
function fun(a,b){
        var  msg = fun.arguments[0]+".."+fun.arguments[1];   //referring a,b values
        alert(msg);
}
```

```
fun(10,20);            //function call
fun("abc","vit");      //function call
</script>
</body>
</html>
```

Returning values

The **return** statement is used to specify the value that is returned from the function. So functions that are going to return a value must use the **return** statement.

Example:

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

Scoping Rules:

Programming languages usually impose rules, called scoping, which determine how a variable can be accessed. JavaScript is no exception. In JavaScript variables can be either ***local*** or ***global.***

global

Global scoping means that a variable is available to all parts of the program. Such variables are declared outside of any function.

local

Local variables are declared inside a function. They can only be used by that function.

GLOBAL FUNCTIONS

>**parseInt()**	>Converts the string argument to an integer and returns the value. If the string cannot be converted, it returns **NaN**. Like **parseFloat ()**, this method should handle strings starting with numbers and peel off what it needs, but other mixed strings will not be converted.	>var x;
		x = parseInt("-53");
		// x is -53
		
		x = parseInt("33.01568");
		// x is 33
		
		x = parseInt("47.6k-red-dog");
		// x is 47
		
		x = parseInt("a567.34");
		// x is NaN
		
		x = parseInt("won't work");
		// x is NaN

>**unescape ()**	>Takes a hexadecimal string value containing some characters of the form %xx and returns the ISO-Latin-1 ASCII equivalent of the passed values.	>Var aString="O%27Neill%20%26%20Sons";
		aString = unescape(aString);
		// aString = "O'Neill & Sons"
		aString = unescape("%64%56%26%23");
		// aString = "dV&#"

Method	Description	Example

>escape()	>Takes a string and returns a string where all non-alphanumeric characters such as spaces, tabs, and special characters have been replaced with their hexadecimal equivalents in the form %xx.	>var aString="O'Neill & Sons";
		// aString = "O'Neill & Sons"
		aString = escape(aString);
		// aString="O%27Neill%20%26%20Sons"

>eval()	>Takes a string and executes it as JavaScript code.	>var x;
		var aString = "5+9";
		x = aString;
		// x contains the string "5+9"
		x = eval(aString);
		// x will contain the number 14

>isFinite()	>Returns a Boolean indicating whether its number argument is finite.	>var x;
		x = isFinite('56');
		// x is true
		x = isFinite(Infinity)
		// x is false

>isNaN()	>Returns a Boolean indicating whether its number argument is **NaN**.	>var x;
		x = isNaN('56');
		// x is False
		x = isNaN(0/0)
		// x is true
		x = isNaN(NaN);
		// x is true

>parseFloat()	>Converts the string argument to a floating-point number and returns the value. If the string cannot be converted, it returns **NaN**. The method should handle strings starting with numbers and peel off what it needs, but other mixed strings will not be converted.	>var x;
		x = parseFloat("33.01568");
		// x is 33.01568
		x = parseFloat("47.6k-red-dog");
		// x is 47.6
		x = parseFloat("a567.34");
		// x is NaN
		x = parseFloat("won't work");
		// x is NaN

EXCEPTION HANDLING

Runtime error handling is vitally important in all programs. Many OOP languages provide a mechanism for handling with general classes of errors. The mechanism is called *Exception Handling.*

An exception in object-based programming language is an object, created dynamically at run-time, which encapsulates an error and some information about it. In Java Script it returns an error object when an error is generated at browser while the code is executing.

try-catch block

The block of code that might cause the exception is placed inside **try** block. **catch** block contains statements that are to be executed when the exception arises.

```
try
{
        statement one

        statement two

        statement three

}
catch(Error)
{
        //Handle errors here
}
finally
{
       // execute the code even regardless of above catches are matched
}
```

Example:

```
try{
        alert('This is code inside the try clause');
        ablert ('Exception will be thrown by this code');
}
catch(exception)
{
        alert("Internet Explorer says the error is " + exception.description);
}
```

throw statement

The **throw** statement allows us to create an exception. If we use this statement together with the **try...catch** statement, we can control program flow and generate accurate error messages.

Syntax

throw(exception)

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

Example:

```
<html>
<body>
<script type="text/javascript">
var x=prompt("Enter a number between 0 and 10:","");
try
  {
        if(x>10)
        {
                throw  "Error! The value is too high";

        }
        else if(x<0)
        {
                throw "Error! The value is too low";

        }
        else if(isNaN(x))
        {
                 throw  "Error! The value is not a number";
        }
  }
catch(er)
  {
            alert(er);
  }

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

Document Object Model (DOM)

*The **DOM** defines what properties of a document can be retrieved and changed, and the methods that can be performed.* **(or)**

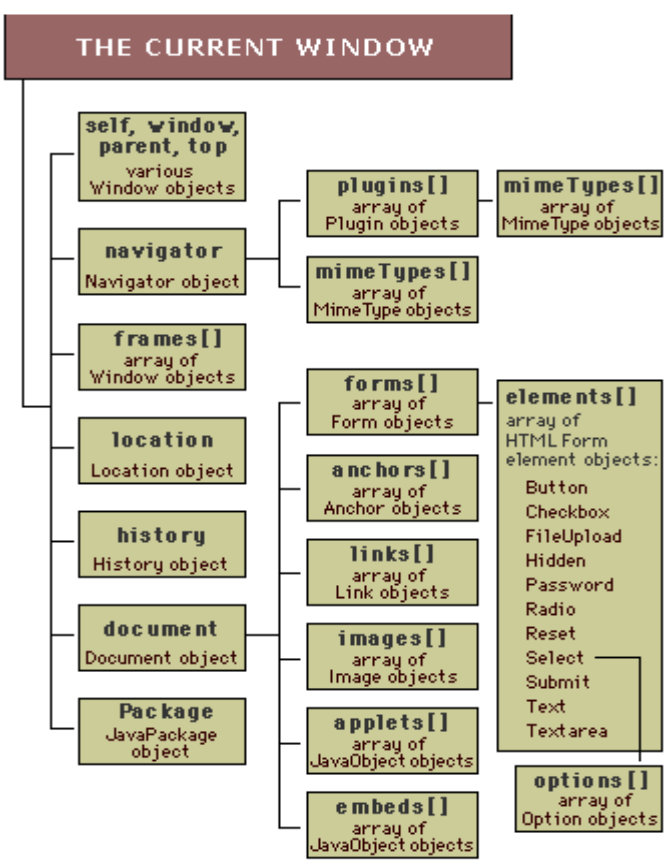
*The Browser **DOM** specifies how JavaScript (and other programming languages) can be used to access information about a document. Then you can perform calculations and decisions based on the values retrieved from the document using JavaScript*

Example: We can retrieve properties such as the value of the **height** attribute of any image, the **href** attribute of any link, or the length of a password entered into a text box in a form. Meanwhile, methods allow us to perform actions such as **reset()** or **submit()** methods on a form that allows you to reset or submit a form.

DOM Hierarchy

→ The objects in the web page follow a strict hierarchy, where the **window** object is the very top level. Because **window** is the top level "root" object it can be omitted in the address syntax. For instance, the **window.document.bgColor** property, which stores the value of the window's current background color, can be addressed simply as **document.bgColor**

→ Several of the DOM objects have properties that contain an array of elements in that web page. For example, with **document.images[],** the **images[]** array is a property of the document object that will store the URL address of each image contained on that web page.The URL of the first image in the HTML code is stored in the array at **document.images[0]**



BUILT-IN OBJECTS IN JAVASCRIPT

Javascript has many built-in objects which posses the capability of performing many tasks. Hence sometimes Javascript is referred to as an ***Object based programming language***.

Now we will discuss few commonly used objects of javascript along with their attributes & behaviors.

THE WINDOW OBJECT

PROPERTIES:

frames[]

array of frames stored in the order in which they are defined in the document

frames.length

number of frames

self

current window

opener

the window (if any) which opened the current window

parent

parent of the current window if using a frameset

status

message in the status bar

defaultStatus

default message for the status bar

name

the name of the window if it was created using the open( ) method and a name was specified

location

this object contains the full URL address of the document that is currently loaded in the browser, and assigning a new value to this will load the specified URL into the browser.

A typical URL address may comprise these parts:

Protocol: // host / pathname ? #hash

We can use the following properties of **location** object to extract individual pieces of information from URL

window.location.href	**window.location.protocol**	**window.location.host**
window.location.pathname	**window.location.hash**	

history

The **window.history** object contains history (i.e. array of URL addresses previously visited during a browser session). For security reasons, these are not directly readable but they are used to navigate back to previous pages. The back( ) and forward( ) methods of the window.history object emulate the browser's Back and Forward buttons. More flexible navigation is often provided by the window.history.go( ) method.

Example: window.history.go(1) → goes forward to the next page in the history

window.history.go(-2) → goes backward by 2 pages in the history

window.history.go(0) → causes the browser to reload the current document.

Example:

```
<html>
<head>
<script language="javascript">
function fun()
{
        var n = prompt("enter any number");
        window.history.go(n);
}
</script>
</head>
<body>
<form>
<h1>Vishnu Institute of Technology</h1>
<h2>bhimavaram</h2>
<input type="button" value="navigate" onclick=fun()>
</form>
</body>
</html>
```

onload

This object can be used to specify the name of a function to be called immediately after a document has completely loaded in the browser

Example:

```
<html>
<head>
<script language="javascript">
```

```
window.onunload=fun;
function fun(){
        alert("number of frames = " + window.frames.length);
}
</script>
</head>
<frameset rows="30%,30%,*">
        <frame name="row1" src="page1.html">
        <frame name="row2" src="page2.html">
        <frame name="row3" src="page3.html">
</frameset>
</html>
```

onunload

This object can be used to specify the name of a function to be called when the user exits the web page.

METHODS:

alert("string")

opens box containing the message

confirm("string")

displays a message box with OK and CANCEL buttons

prompt("string")

displays a prompt window with field for the user to enter a text string

blur( )

remove focus from current window

focus( )

give focus to current window

scroll(x,y)

move the current window to the chosen x,y location

open("URL", "name", "options string")

The open() method has 3 arguments:

- ➔ URL to load in the pop-up window
- ➔ Name for the pop-up
- ➔ List of options

Example:

```
newWin = window.open(address,"newWin", "status=0, width=100, height=100, resizable=0");
```

The open( ) method can take the following options:

toolbar = [1|0]	location = [1|0]	directories = [1|0]	status = [1|0]	menubar = [1|0]
scrollbars = [1|0]	resizable = [1|0]	width = pixels	height = pixels	

Many of the attributes of a browser window are undesirable in a pop-up window. They can be switched on and off individually

close( )

This shuts the current window

Note: Because **window** is the top level "root" object, it can be omitted in the address syntax. Therefore we can refer its properties directly

Example: window.document.bgColor (or) document.bgColor

window.alert( ) (or) alert( )

THE DOCUMENT OBJECT

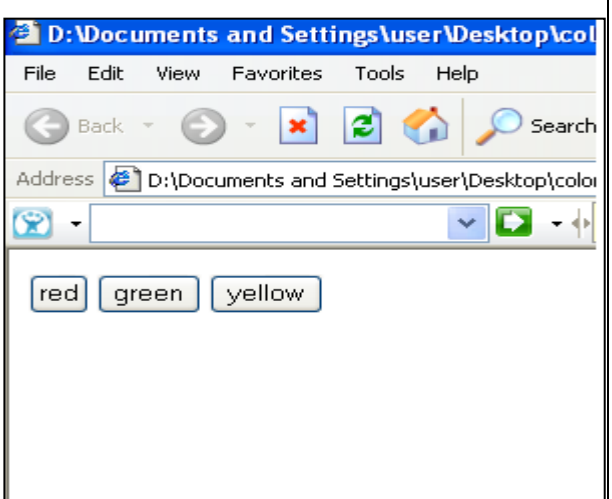
A document is a web page that is being either displayed or created. The document has a number of properties that can be accessed by JavaScript programs and used to manipulate the content of the page.

PROPERTIES:

bgColor

Background color of the document

Example: write a javascript that designs 3 buttons "red", "green", and "yellow". When ever the button is clicked, the document color should change accordingly




```
<html>
<head>
<script language="javascript">
function changecolor(s)
{
        window.document.bgColor=s;
}
</script>
</head>
<body>
<form>
<input type="button" value="red" onclick="changecolor('red')">
<input type="button" value="green" onclick="changecolor('green')">
<input type="button" value="yellow" onclick="changecolor('yellow')">
</form>
</body>        </html>
```

fgColor

Foreground color of the document

title

Title of the current document

location

This object contains the full URL address of the document that is currently loaded in the browser, and assigning a new value to this will load the specified URL into the browser.

Example:

```
<html>
<body>
<script language="javascript">
        document.title="vit";
        function fun(){
                document.location="page1.html";
        }
</script>
<input type="button" value="change url" onclick="fun()">
</body>
</html>
```

lastModified

Object that provides information about date and time when a webpage was last modified. This data is usually supplied to **document.lastModified** from the HTTP file header that is sent by the web server

Example:

```
<html>
<body>
<script language="javascript">
        window.status = "Last updated " + document.lastModified;
</script>
<h1> VIT </h1>
VIT was established in the year 2008.
</body>
</html>
```

linkColor, vlinkColor,alinkColor

These can be used to set the colors for the various types of links

forms[] array of forms on the current page

forms.length

the number of form objects on the page

links[]

array of links in the current page in the order in which they appear in the document

anchors[]

an array of anchors. Any named point inside an HTML document is an anchor. Anchors are create using <a name=…>. These will be commonly used for moving around inside a large page. The anchors property is an array of these names in the order in which they appear in the HTML document. Anchors can be accessed like this: **document.anchors[0]**

images[]

an array of images

applets[]

an array of applets

cookie

object that stores information about cookie

Methods:

write("string")

write an arbitrary string to the HTML document

writeln("string")

write a string to the HTML document and terminate it with a newline character. HTML pages can be created on the fly using JavaScript. This is done using the write or writeln methods of the document object.

Example:

```
document.write("<body>");

document.write("<h1>VIT</h1>");

document.write("<form>");
```

clear()

clear the current document

close()

close the current document

getElementById()

Returns the reference of the form control given by its Id

DATE OBJECT

This object is used to obtain the date and time. This date and time is based on computer's local time (system's time) or it can be based on GMT. This GMT is also known as UTC i.e. Universal Coordinated Time. This is basically a world time standard.

Following are the commonly used methods of Date object:

Method	**Meaning**
getTime()	It returns the number of milliseconds. This value is the difference between the current time and the time value from 1st January 1970
getDate()	Returns the current date based on computers local time
getUTCDate()	Returns the current date obtained from UTC
getDay()	Returns the current day. The day number is from 0 to 6 i.e. from Sunday to Saturday
getUTCDay()	Returns the current day based on UTC. The day number is from 0 to 6
getHours()	Returns the hour value ranging from 0 to 23
getUTCHours()	Returns the hour value ranging from 0 to 23, based on UTC timing zone
getMinutes	Returns the minute value ranging from 0 to 59
getUTCMinutes()	Returns the minute value ranging from 0 to 59, based on UTC
getSeconds()	Returns the seconds value ranging from 0 to 59
getUTCSeconds()	Returns the seconds value ranging from 0 to 59, based on UTC
getMilliseconds()	Returns the milliseconds value ranging from 0 to 999, based on local time
getUTCMilliseconds()	Returns the milliseconds value ranging from 0 to 999, based on UTC
setDate(value)	This is used to set the Date
setHour(hr,min,sec,ms)	This is used to set the Hour

Example:

```
<html>
<head>
<title>Date Object</title>
</head>
<body>
<script type="text/javascript">
        var  d = new Date();
        document.write("The Date is: "+d.toString()+"<br>");

        document.write("Today date is: "+d.getDate()+"<br>");
        document.write("UTC date is: "+d.getUTCDate()+"<br>");

        document.write("Minutes: "+d.getMinutes()+"<br>");
        document.write("UTC Minutes: "+d.getUTCMinutes()+"<br>");
</script>
</body>
</html>
```

MATH OBJECT

For performing the mathematical computations there are some useful methods available from math object.

sqrt(num) abs(num) ceil(num) floor(num) log(num) pow(a,b) min(a,b) max(a,b)

sin(num) cos(num) tan(num) exp(num) asin(value) acos(value) atan(value)

random() → returns a psuedorandom number between 1 to 1

round(value)

In addition to the above methods, it has several properties (Numeric constants) like:

Math.E → Euler constant
Math.PI → 3.14159
Math.SQRT_2 → The square root of 2
Math.SQRT1_2 → The square root of ½
Math.LN2 → Log of 2
Math.LN10 → Log of 10

Example:

```
<html>
<body>
<script language="javascript">
        var n = prompt("enter any number");
        alert("square root is "+Math.sqrt(n));
  </script>
</body>
</html>
```

Exercise1:

1.Write a JavaScript program that generates the following table for the given value of n

Number	Square
1	1
2	4
3	9
4	16

```
<html>
<body>
<table border=1>
<tr>  <th>Number  <th>Square  </tr>
<script language="javascript">
        var n = prompt("enter n");
        for(i=1;i<=n;i++)
        {
                document.write("<tr><td>"+i+"<td>"+(i*i)+"</tr>");
        }
</script>
</table>
</body>
</html>
```

FORM OBJECT

The **window.document.forms** object contains an array of all the forms in a HTML document, indexed in the order in which they appear in the HTML code.

For example, **window.document.forms[0]** addresses the first form to appear in the HTML code of a web page.

If the **id** attribute of the **<form>** element has been assigned a value, then the form can be addressed by name.

For example, a form named **reg** can be addressed as **document.forms.reg**

All the attributes assigned in the **<form>** tag can be accessed as properties of that form object.

Example:

```
<html>
<head>
<script language=”javascript”>
window.onload = fun;
function fun()
{
        var msg = “Form name: “ + document.forms.reg.id;
        msg += “\nMethod: “ + document.forms.reg.method;
        msg += “\nAction: “ + document.forms.reg.action;
        window.alert(msg);
}
</script>
</head>
<body>
<form id=”reg” method=”post”  action=mailto:abc@xyz.com>


        Name: <input type="text" size=10> <br>
        Age: <input type="text" size=5> <br>
</form>
</body>
</html>
```

Form elements:

The elements of the form are held in the array **window.document.forms[].elements[]**

The properties of the form elements can be accessed and set using Javascript.

Example1:

```
<html>
<head>
<script language="javascript">
function fun(){
        var msg = "Element type: " + document.forms.reg.elements[0].type;
         msg += "\nElement value: " + document.forms.reg.elements[0].value;
        window.alert(msg);
}
</script>
</head>
<body>
<form id="reg">
<input type="button" value="click" name="btn1" onClick="fun()">
</form>
</body>
</html>
```

Example2:

<!--It is useful to change the label that is displayed on a button by its value attribute if that button performs dual actions

-->

```
<html>
<head>
<script language="javascript">
var running=false;      var num=0;
var tim;
function startstop(){
        running = !running;    count();
        document.forms[0].btn1.value = (running) ? "stop" : "start";
}
function count(){
        if(running){
             num++;
             window.status = "seconds elapsed: " + num;
             tim = setTimeout("count()", 1000);
        }
        else{
                num=0;        clearTimeout(tim);
        }
}
```

```
</script>
</head>
<body>
<form>
<input type="button" value="start" name="btn1" onClick="startstop()">
</form>
</body>
</html>
```

form Object properties, methods

Properties:

name

the name of the form

method

submission method in numeric form. 0 = GET, 1 = POST

action

the action attribute of the form

target

if specified this is the target window for responses to the submission of the form

elements[]

an array containing the form elements in the order in which they are declared in the document

length

the number of elements in the form

Methods:

submit()

submits the form

reset()

resets the form i.e. form controls will be set to default values

Radio Buttons

→Radio Buttons allow to select one option from a list of options.

→Radio Buttons can be created using <input type=”radio”>

→In the browser DOM the radio button group creates a **document.form** object with the given name

Example: Radio Buttons

```
<html>
<head>
<script language="javascript">
function radio_info()
{
```

```
        var msg="1st radio value = " + document.forms[0].rbnBranch[0].value;
         msg += "\n 2nd radio value = " + document.forms[0].rbnBranch[1].value;
          msg += "\n 3rd radio value = " + document.forms[0].rbnBranch[2].value;
         msg += "\n 4th radio value = " + document.forms[0].rbnBranch[3].value;
          alert(msg);
}
</script>
</head>
<body>
<form id="form1">
        The branches in our college are: <br>
        <input type="radio" name="rbnBranch" value="cse" selected>CSE<br>
        <input type="radio" name="rbnBranch" value="it">IT<br>
        <input type="radio" name="rbnBranch" value="ece">ECE<br>
        <input type="radio" name="rbnBranch" value="eee">EEE<br><br>
        <input type="button" value="Get Radio Info" onclick="radio_info()">
</form>
</body>
</html>
```

Radio Polling: The important feature of a radio button is whether it is checked or not. This can be ascertained from the radio button's **checked** property. It will return **true**, if the button is checked otherwise **false.**

```
<html>
<head>
<script language="javascript">
function radio_info()
{
        var msg="1st radio status = " + document.forms[0].rbnBranch[0].checked;
          msg += "\n 2nd radio status = " + document.forms[0].rbnBranch[1].checked;
          msg += "\n 3rd radio status = " + document.forms[0].rbnBranch[2].checked;
          msg += "\n 4th radio status = " + document.forms[0].rbnBranch[3].checked;
          alert(msg);
}
</script>
</head>
```

```
<body>
<form id="form1">
        The branches in our college are: <br>
        <input type="radio" name="rbnBranch" value="cse" selected>CSE<br>
        <input type="radio" name="rbnBranch" value="it">IT<br>
        <input type="radio" name="rbnBranch" value="ece">ECE<br>
        <input type="radio" name="rbnBranch" value="eee">EEE<br><br>
        <input type="button" value="Get Radio Info" onclick="radio_info()">
</form>
</body>
</html>
```

Check Boxes

Example:

```
<html>
<head>
<script language="javascript">
function info()
{
  var msg="1st radio status = " + document.forms[0].branch[0].checked;
  msg += "\n 2nd radio status = " + document.forms[0].branch[1].checked;
  msg += "\n 3rd radio status = " + document.forms[0].branch[2].checked;
  msg += "\n 4th radio status = " + document.forms[0].branch[3].checked;
  alert(msg);
}
</script>
</head>
<body>
<form id="form1">
The branches in our college are: <br>
<input type="checkbox" name="branch" value="cse">CSE<br>
<input type="checkbox" name="branch" value="it">IT<br>
<input type="checkbox" name="branch" value="ece">ECE<br>
<input type="checkbox" name="branch" value="eee">EEE<br><br>

<input type="button" value="Get Info" onclick="info()">
</form>
</body>
</html>
```

Option Lists

The <option> tag can be used in the <select> tag to specify various options in the drop down menu list.All menu items are stored in the <select> object's **options[]** array. We can use the **selectedIndex** property of **option list** to identify the index of the selected item

Example:

```
<html>
<head>
<script language="javascript">
function get_selected(){
   var s = document.forms[0].sltVehicle.selectedIndex;
   document.forms[0].txtVehicle.value = document.forms[0].sltVehicle.options[s].text;
}
</script>
</head>
<body>
<form name="form1">
<select name="sltVehicle">
        <option value="v"> Volvo </option>
        <option value="s" selected> Saab </option>
         <option value="m"> Mercedes </option>
        <option value="a"> Audi </option>
</select>
<input type="button" value="Show Selected" onClick="get_selected()">
<input type="text" size=15 name="txtVehicle">
</form>
</body>
</html>
```

Exercise: Write a program that designs a Simple Calculator


```
<html>
<head>
<script language="javascript">
var exp="";
function fun(ch){
  if(ch=='=')   {
        calc.txt1.value=eval(exp);      exp = "";
  }
  else{
        exp = exp + ch;    calc.txt1.value=exp;
  }
}
```

```
</script>
</head>
<body>
<form name="calc">
<table border=1>
<tr>
<th colspan=3>Simple calculator</th>
</tr>
<tr>
<th colspan=3><input type="text" name="txt1" size=15></th>
</tr>

<tr>
<td><input type="button" value="1" onclick="fun('1')"></td>
<td><input type="button" value="2" onclick="fun('2')"></td>
<td><input type="button" value="3" onclick="fun('3')"></td>
</tr>

<tr>
<td><input type="button" value="4" onclick="fun('4')"></td>
<td><input type="button" value="5" onclick="fun('5')"></td>
<td><input type="button" value="6" onclick="fun('6')"></td>
</tr>

<tr>
<td><input type="button" value="7" onclick="fun('7')"></td>
<td><input type="button" value="8" onclick="fun('8')"></td>
<td><input type="button" value="9" onclick="fun('9')"></td>
</tr>

<tr>
<td><input type="button" value="+" onclick="fun('+')"></td>
<td><input type="button" value="-" onclick="fun('-')"></td>
<td><input type="button" value="=" onclick="fun('=')"></td>
</tr>
</table>
</form>
</body>
</html>
```

The Browser / Navigator object

No two browser models will process our javascript in the same way. Its important that we find out which browser is being used to view our page. Then we can make a choice from our visitors:

- ➔ Redirect them to a non-scripted version of our site
- ➔ Present scripts that are tailored to suit each browser

For historical reasons, browser object is called **navigator** object

The **navigator** object has properties that provide information about the browser that is being used to view a document.

Properties:

Navigator.appCodeName→ The internal name for the browser. For both major products, this is Mozilla, which was the name of the original Netscape code source

navigator.appName→ public name of the browser

navigator.appVersion→the version number, platform on which the browser is running

navigator.userAgent→ appCodeName + appVersion

navigator.platform→ platform in which browser in running

navigator.plugins[] → array containing details of installed plugins

navigator.mimeTypes→ array of all supported MIME types. Useful to make sure that the browser can handle our data

Methods:

navigator.javaEnabled() →This method returns true or false depending upon whether java is enabled or not in the system

Example:

```
<html>
<head>
<script language="javascript">
function fun()
{
    if(navigator.javaEnabled()) window.location = "appletpage.html";
    else  window.location = "nonapplet.html";
}
</script>
</head>
<body>
Vishnu Institute of Technology was established in the year 2008 with four branches. For further details visit our  <a href="javascript:fun()" > website  </a>
</body>
</html>
```

COOKIES

- Cookies are small bits of key-value pair information that a Web server sends to a browser through HTTP response and that the browser later returns unchanged through HTTP Request when visiting the same Web site or domain.
- A cookie is a small piece of information that is passed back and forth in the HTTP request and response. The cookie sent by a servlet to the client will be passed back to the server when the client requests another page from the same application.
- Cookies are tiny files that can be written by javascript to store small amounts of data on the local hard drive. There are limitations to the use of cookies that restrict their size to 4 kilobytes and web browsers are not required to retain more than 20 cookies per web server. Typically a cookie may often retain user data for use across web pages or on subsequent visits to a web site
- Depending on the *maximum age* of a cookie, the Web browser either maintains the cookie for the duration of the browsing session (i.e., until the user closes the Web browser) or stores the cookie on the client computer for future use. When the browser requests a resource from a server, cookies previously sent to the client by that server are returned to the server as part of the request formulated by the browser. Cookies are deleted automatically when they *expire* (i.e., reach their maximum age).

Benefits of Cookies:

- **Identifying a user during an e-commerce session**
- **Remembering usernames and passwords :** Cookies let a user log in to a site automatically, providing a significant convenience for users of unshared computers.
- **Customizing sites:** Sites can use cookies to remember user preferences.
- **Focusing advertising:** Cookies let the site remember which topics interest certain users and show advertisements relevant to those interests.
- **Security Issue:** Browsers generally only accept 20 cookies per site and 300 cookies total, and since browsers can limit each cookie to 4 kilobytes, cookies cannot be used to fill up someone's disk or launch other denial-of-service attacks.

History:

Cookies were originally invented by Netscape to give 'memory' to web servers and browsers. The HTTP protocol, which arranges for the transfer of web pages to your browser and browser requests for pages to servers, is *state-less*, which means that once the server has sent a page to a browser requesting it, it doesn't remember a thing about it. So if you come to the same web page a second, third, hundredth or millionth time, the server once again considers it the very first time you ever came there.

This can be annoying in a number of ways. The server cannot remember if you identified yourself when you want to access protected pages, it cannot remember your user preferences, it cannot remember anything. As soon as personalization was invented, this became a major problem.

Cookies were invented to solve this problem. There are other ways to solve it, but cookies are easy to maintain and very versatile.

How cookies work?

A cookie is nothing but a small text file that's stored in your browser. It contains some data:

1. A name-value pair containing the actual data
2. An expiry date after which it is no longer valid
3. The domain and path of the server it should be sent to

As soon as you request a page from a server (which was requested earlier & the server sent cookie to the client), the cookie is added to the HTTP header. Server side programs can then read out the information and give response accordingly. So every time you visit the site the cookie comes from, information about you is available. This is very nice sometimes, at other times it may somewhat endanger your privacy.

Cookies can be read by JavaScript too. They're mostly used for storing user preferences.

name-value

Each cookie has a ***name-value pair*** that contains the actual information. The name of the cookie is for your benefit, you will search for this name when reading out the cookie information.

Expiry date

Each cookie has an ***expiry date*** after which it is trashed. If you don't specify the expiry date the cookie is trashed when you close the browser. This expiry date should be in UTC (Greenwich) time.

Domain and path

Each cookie also has a ***domain*** and a ***path***. The domain tells the browser to which domain the cookie should be sent. If you don't specify it, it becomes the domain of the page that sets the cookie.

document.cookie

Cookies can be created, read and erased by JavaScript. They are accessible through the property document.cookie. Though you can treat document.cookie as if it's a **string**, it isn't really, and you have only access to the *name-value* pairs.

If I want to set a cookie for this domain with a name-value pair 'ppkcookie1=testcookie' that expires in seven days from the moment I write this sentence, I do

```
document.cookie='ppkcookie1=testcookie; expires=Thu, 2 Aug 2001 20:47:11 UTC; path=/'
```

Event Handling

An ***event*** is defined as "something that takes place" and that is exactly what it means in web programming as well.

An ***event handler*** is JavaScript code that is designed to run each time a particular event occurs.

Syntax for handling the events:

<tag Attributes **event="handler"**>

Table: JavaScript Events

Event	Handler	Description
blur	onBlur	The input focus is moved from the object
change	onChange	The value of a field in a form has been changed by the user entering or deleting data
click	onClick	The mouse is clicked over an element of a page
dblclick	onDblClick	A form element or link is clicked twice in rapid succession
dragdrop	onDragDrop	A system file is dragged with a mouse and dropped onto the browser
focus	onFocus	Input focus is given to an element. The reverse of blur
keydown	onKeyDown	A key is pressed but not released
keypress	onKeyPress	A key is pressed
keyup	onKeyUp	A pressed key is released
load	onLoad	The page is loaded by the browser
mousedown	onMouseDown	A mouse button is pressed
mousemove	onMouseMove	The mouse is moved
mouseout	onMouseOut	The mouse pointer moves off an element
mouseover	onMouseOver	The mouse pointer moves over an element
mouseup	onMouseUp	The mouse button is released
move	onMove	A window is moved, maximized or restored either by the user or by a script
resize	onResize	A window is resized by the user or by script
submit	onSubmit	A form is submitted (the **submit** button is clicked)
unload	onUnload	The user leaves the webpage

Table: Objects and Event Handlers

Object	Event Handlers			
window	onload	onunload	onblur	onfocus
link	onclick	onmouseout	onmouseover	
image	onabort	onerror	onload	
form	onreset	onsubmit		
text, password	onblur	onchange	onfocus	
textarea	onblur	onchange	onfocus	
button	onclick			
reset	onclick			
submit	onclick			
radio	onclick			
checkbox	onclick			
select	onblur	onchange	onfocus	
fileupload	onblur	onchange	onfocus	

Handling Events

There are two ways to set and execute the JavaScript event handler for an HTML tag:

- Set the event handler property inside HTML
- Set the event handler property inside JavaScript

Set the event handler property inside HTML: Example

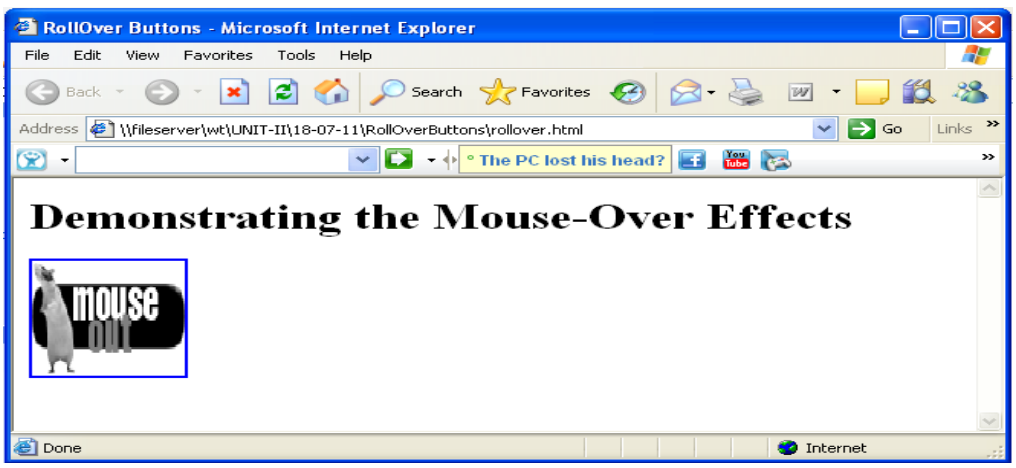
```
<img src="img.jpg"  id="SampleImage"  onmouseover="changeimage(1)"
onmouseout="changeimage(2)" >
```

Set the event handler property inside JavaScript: Example

```
<img src="img.jpg" id="SampleImage" >
<script type="text/javascript" language="javascript">
var img = document.getElementById("SampleImage");
img.onmouseover = changeimageover;
img.onmouseout = changeimageout;
function changeimageover() {
        img.src = "flower.jpg";
}
```

```
function changeimageout()
 {
        img.src = “img.jpg";
}
</script>
```

Exercise: Write JavaScript that illustrates Mouse-Over Effects




```
<html>
<head>
<script language="javascript">
function change(v)
{
        var i = document.getElementById("mouse");
         if(v==1) i.src="over.gif";
        else i.src="out.gif";
}
</script>
</head>
<body>
<form name="form1">
<h1>Demonstrating Rollover Buttons</h1>
<img id="mouse" src="out.gif" width=100 height=100 onMouseOver=change(1)
onMouseOut=change(0)>
</form>
</body>
</html>
```

Exercise:

```
<html>
<head>
<script language="javascript">
```

```
function fun(s)
{
        if(s==0) document.form1.bulb.src="off.jpg";
        else document.form1.bulb.src="on.jpg";
}
</script>
</head>
<body>
<form name="form1">
        <input type="button" value="switch off" onclick=fun("0")>
        <input type="button" value="switch on" onclick=fun("1")>
        <img name="bulb" src="off.jpg" width=150 height=200>
</form>  </body>   </html>
```

Regular Expressions & Pattern Matching

A Regular expression is a way of describing a pattern in a piece of text. It's an easy way of matching a string to a pattern.

We could write a simple regular expression and use it to check, quickly, whether or not any given string is a properly formatted user input. This saves us from difficulties and allows us to write clean and tight code.

For instance, a script might take "***name"*** data from a user and have to search through it checking that no digits have been entered. This type of problem can be solved by reading through the string one character at a time looking for the target pattern. Although it seems like a straightforward approach, it is not (Efficiency & speed matters, so any code that does has to be written carefully). The usual approach in scripting languages is to create a pattern called a ***regular expression,*** which describes a set of characters that may be present in a string.

Creating Regular Expressions:

A regular expression is a JavaScript object. We can create regular expressions in one of two ways.

→Static regular expressions

Ex: var regex = /fish/fowl/ ;

→Dynamic regular expressions

Ex: var regex = new RegExp("fish|fowl");

Note: If performance is an issue for our script, then we should try to use static expressions whenever possible. If we don't know what we are going to be searching until runtime (for instance, the pattern may depend on user input) then we create dynamic patterns

A regular expression pattern is composed of simple characters, such as /abc/, or a combination of simple and special characters, such as /ab*c/ or /Chapter (\d+)\.\d*/.

Using Simple Patterns:

- Simple patterns are constructed of characters for which we want to find a direct match.
- For example, the pattern /abc/ matches character combinations in strings only when exactly the characters 'abc' occur together and in that order.
- Such a match would succeed in the strings

  "Hi, do you know your abc's?“

  "The latest airplane designs evolved from slabcraft."
- In both cases the match is with the substring 'abc'.
- There is no match in the string "Grab crab" because it does not contain the substring 'abc'.

Using special characters:

- When the search for a match requires something more than a direct match, such as finding one or more b's, or finding white space, the pattern includes special characters.
- For example, the pattern /ab*c/ matches any character combination in which a single 'a' is followed by zero or more 'b's (* means 0 or more occurrences of the preceding item) and then immediately followed by 'c'. In the string "cbbabbbbcdebc," the pattern matches the substring 'abbbbc'

Table: JavaScript Regular Expression Grammar

Token	Description/Meaning
\	Either of the following: • For characters that are usually treated literally, indicates that the next character is special and not to be interpreted literally. For example, /b/ matches the character 'b'. By placing a backslash in front of b, that is by using /\b/, the character becomes special to mean match a word boundary. • For characters that are usually treated specially, indicates that the next character is not special and should be interpreted literally. For example, * is a special character that means 0 or more occurrences of the preceding item should be matched; for example, /a*/ means match 0 or more a's. To match * literally, precede it with a backslash; for example, /a*/ matches 'a*'.
^	Matches beginning of input. If the multiline flag is set to true, also matches immediately after a line break character. For example, /^A/ does not match the 'A' in "an A", but does match the first 'A' in "An A".
$	Matches end of input. If the multiline flag is set to true, also matches immediately before a line break character. For example, /t$/ does not match the 't' in "eater", but does match it in "eat".
*	Matches the preceding character 0 or more times. For example, /bo*/ matches 'boooo' in "A ghost booooed" and 'b' in "A bird warbled", but nothing in "A goat grunted".
+	Matches the preceding character 1 or more times. Equivalent to {1,}. For example, /a+/ matches the 'a' in "candy" and all the a's in "caaaaaaandy".

	Matches the preceding character 0 or 1 time. For example, `/e?le?/` matches the 'el' in "angel" and the 'le' in "angle."
?	If used immediately after any of the quantifiers *, +, ?, or {}, makes the quantifier non-greedy (matching the minimum number of times), as opposed to the default, which is greedy (matching the maximum number of times). For example, using /\d+/ non-global match "123abc" return "123", if using /\d+?/, only "1" will be matched. Also used in lookahead assertions, described under x(?=y) and x(?!y) in this table.
.	(The decimal point) matches any single character except the newline character. For example, /.n/ matches 'an' and 'on' in "nay, an apple is on the tree", but not 'nay'.
(x)	Matches 'x' and remembers the match. These are called capturing parentheses. For example, /(foo)/ matches and remembers 'foo' in "foo bar." The matched substring can be recalled from the resulting array's elements [1], ..., [n].
(?:x)	Matches 'x' but does not remember the match. These are called non-capturing parentheses. The matched substring can not be recalled from the resulting array's elements [1], ..., [n].
x(?=y)	Matches 'x' only if 'x' is followed by 'y'. For example, /Jack(?=Sprat)/ matches 'Jack' only if it is followed by 'Sprat'. /Jack(?=Sprat|Frost)/ matches 'Jack' only if it is followed by 'Sprat' or 'Frost'. However, neither 'Sprat' nor 'Frost' is part of the match results.
x(?!y)	Matches 'x' only if 'x' is not followed by 'y'. For example, /\d+(?!\.)/ matches a number only if it is not followed by a decimal point. The regular expression /\d+(?!\.)/.exec("3.141") matches '141' but not '3.141'.
x|y	Matches either 'x' or 'y'. For example, /green|red/ matches 'green' in "green apple" and 'red' in "red apple."
{n}	Where n is a positive integer. Matches exactly n occurrences of the preceding character. For example, /a{2}/ doesn't match the 'a' in "candy," but it matches all of the a's in "caandy," and the first two a's in "caaandy."
{n,}	Where n is a positive integer. Matches at least n occurrences of the preceding character. For example, /a{2,}/ doesn't match the 'a' in "candy", but matches all of the a's in "caandy" and in "caaaaaaandy."
{n,m}	Where n and m are positive integers. Matches at least n and at most m occurrences of the preceding character. For example, /a{1,3}/ matches nothing in "cndy", the 'a' in "candy," the first two a's in "caandy," and the first three a's in "caaaaaaandy" Notice that when matching "caaaaaaandy", the match is "aaa", even though the original string had more a's in it.
[xyz]	A character set. Matches any one of the enclosed characters. You can specify a range of characters by using a hyphen. For example, [abcd] is the same as [a-d]. They match the 'b' in "brisket" and the 'c' in "city".
[^xyz]	A negated or complemented character set. That is, it matches anything that is not enclosed in the brackets. You can specify a range of characters by using a hyphen. For example, [^abc] is the same as [^a-c]. They initially match 'r' in "brisket" and 'h' in "chop."

[\b]	Matches a backspace. (Not to be confused with \b.)
\b	Matches a word boundary, such as a space or a newline character. (Not to be confused with [\b].) For example, /\bn\w/ matches the 'no' in "noonday";/\wy\b/ matches the 'ly' in "possibly yesterday."
\B	Matches a non-word boundary. For example, /\w\Bn/ matches 'on' in "noonday", and /y\B\w/ matches 'ye' in "possibly yesterday."
\d	Matches a digit character. Equivalent to [0-9]. For example, /\d/ or /[0-9]/ matches '2' in "B2 is the suite number."
\D	Matches any non-digit character. Equivalent to [^0-9]. For example, /\D/ or /[^0-9]/ matches 'B' in "B2 is the suite number."
\s	Matches a single white space character, including space, tab, form feed, line feed. Equivalent to [\f\n\r\t\v\u00A0\u2028\u2029]. For example, /\s\w*/ matches ' bar' in "foo bar."
\S	Matches a single character other than white space. Equivalent to [^ \f\n\r\t\v\u00A0\u2028\u2029]. For example, /\S\w*/ matches 'foo' in "foo bar."
\t	Matches a tab.
\w	Matches any alphanumeric character including the underscore. Equivalent to [A-Za-z0-9_]. For example, /\w/ matches 'a' in "apple," '5' in "$5.28," and '3' in "3D."
\W	Matches any non-word character. Equivalent to [^A-Za-z0-9_]. For example, /\W/ or /[^A-Za-z0-9_]/ matches '%' in "50%."

Remembering the Result

Sometimes we may be looking for a pattern that we use elsewhere. This means that, we need to remember the result of our search. The **RegExp** object holds the result of its operations in an array which it returns to the calling script.

Example: Write JavaScript that takes 2 strings as input. In the 1st input string search for the pattern “a followed by zero or more b’s followed by c”. If the pattern appears in 1st string, then the matched string should appear in 2nd string

```
<html>
<body>
<script language="javascript">
var s1 = prompt("enter string1");
var s2 = prompt("enter string2");
var reg = /ab*c/;
var r = s1.match(reg);
if(r){
        if(s2.match(r[0]))   alert("the matched string is " + r[0]);
        else   alert("the string is not available in string2");
}
else
       alert("the pattern is not available in string1");
</script>
</body>
</html>.
```

Using Parenthesized Substring Matches

- Including parentheses in a regular expression pattern causes the corresponding submatch to be remembered.
- For example, /a(b)c/ matches the characters 'abc' and remembers 'b'.
- To recall these parenthesized substring matches, use the Array elements [1], ..., [n].
- The number of possible parenthesized substrings is unlimited. The returned array holds all that were found.

Example

```
<script language="javascript">
 re = /(\w+)\s(\w+)/;
str = "John Smith";
newstr = str.replace(re, "$2, $1");
document.write(newstr);
</script>
```

Functions used to manipulate Regular Expressions:

Regular Expressions are manipulated using functions which belong to either **RegExp** or **String** classes

String class Functions:

match(pattern)

Searches for a matching pattern. Returns an array holding the results, or **null** if no match is found

replace(pattern1, pattern2)

Searches for **pattern1**. If the search is successful **pattern1** is replaced with **pattern2**

search(pattern)

Searches for **pattern** in the string. If the match is successful, the index of the start of the match is returned. If the search fails, the function returns **-1**

split(pattern)

Splits the string into parts based upon **pattern**, or regular expression, which is supplied as a parameter

RegExp class Functions:

exec(string)

Executes a search for a matching pattern in its parameter string. Returns an array holding the result(s) of the operation.

test(string)

Searches for a match in its parameter string. Returns true is a match is found, otherwise returns false

Advanced Searching With Flags

- Regular expressions have 3 optional flags that allow for global and case insensitive searching.
- To indicate a global search across all input values rather than stopping when a match is found, use the ***g*** flag.
- To indicate a case-insensitive search, use the ***i*** flag.
- To indicate a multi-line search, use the ***m*** flag.
- To include a flag with the regular expression, use this syntax:
  - re = /pattern/flags
  - re = new RegExp("pattern", ["flags"])

Example1:

```
<script type="text/javascript">
re = /\w+\s/g;
str = "fee fi fo fum";
myArray = str.match(re);
document.write(myArray);
</script>
```

- This displays ["fee ", "fi ", "fo "]. In this example, we could replace the line:
  - re = /\w+\s/g; with: re = new RegExp("\\w+\\s", "g");

Example2:

```
function  validate1(){
        var valid=false;
        var id=document.forms[0].elements[0].value;
        var  pass=document.forms[0].elements[1].value;
        var  id_re= new RegExp("^[0-9][0-9a-zA-Z]+$","g");
        var  pass_re=new RegExp("[0-9A-Za-z]{6,}","g")
        if(id.match(id_re))        {
                if(pass.match(pass_re))   valid=true;
                else   alert("password is invalid");
         }
          else
                    alert("login id is invalid");
          return valid;
}
```

Example3:

```
<html>
<head>
<script language="javascript">
          var s = prompt("enter any string");
          var re1 = /^\w+\s/;
          var re2 = /\s\w+$/;
          var w1 = s.match(re1).toString();
          var w2 = s.match(re2).toString();
          var mw1 = w1.substring(0,w1.length-1);
          var mw2 = w2.substring(1);
          if(mw1 == mw2)    alert("same");
          else   alert("not same");
</script>
</head>
</html>
```

Miscellaneous program:

```
<html>
<body>
<script language="javascript">
function fun(){
  return true;
}
</script>
For further details visit our <a href="page1.html" onclick="return fun()"> website</a>
</body></html>
```

UNIT-V

XML

Introduction:

XML stands for eXtensible Markup Language

Differences between HTML & XML:

HTML	XML
HTML is used to mark up text so it can be displayed to users	XML is used to mark up data so it can be processed by computers
HTML describes both structure (e.g. <p>, <h2>,) and appearance (e.g. , <font>, <i>)	XML describes only content, or "meaning"
HTML uses a fixed, unchangeable set of tags	In XML, you make up your own tags

- HTML and XML look similar, because they are both SGML languages (SGML = Standard Generalized Markup Language)
  - Both HTML and XML use elements enclosed in tags (e.g. <body>This is an element</body>)
  - Both use tag attributes (e.g., <font face="Verdana" size="+1" color="red">)
  - Both use entities (<, >, &, ", ')
- More precisely,
  - HTML is defined in SGML
  - XML is a (very small) subset of SGML
- HTML is for humans
  - HTML describes web pages
  - You don't want to see error messages about the web pages you visit

- Browsers ignore and/or correct as many HTML errors as they can, so HTML is often sloppy

- XML is for computers
  - XML describes data
  - The rules are strict and errors are not allowed
    - In this way, XML is like a programming language
  - Current versions of most browsers can display XML
    - However, browser support of XML is spotty at best

XML-related technologies:

- DTD (Document Type Definition) and XML Schemas are used to define legal XML tags and their attributes for particular purposes
- CSS (Cascading Style Sheets) describe how to display HTML
- XSLT (eXtensible Stylesheet Language Transformations) and XPath are used to translate from one form of XML to another
- DOM (Document Object Model), SAX (Simple API for XML, and JAXP (Java API for XML Processing) are all APIs for XML parsing

XML-based markup languages include

- XHTML
- MathML (for mathematics)
- VoiceXML (for speech)
- SMIL(the Synchronous Multimedia Integration Language—for multimedia presentations)
- CML (Chemical Markup Language—for chemistry)
- XBRL (Extensible Business Reporting Language—for financial data exchange)

Example:

```
<?xml version="1.0"?>
```

```
<weatherReport>
  <date>7/14/97</date>
  <city>North Place</city>, <state>NX</state>
  <country>USA</country>
  High Temp: <high scale="F">103</high>
  Low Temp: <low scale="F">70</low>
  Morning: <morning>Partly cloudy, Hazy</morning>
  Afternoon: <afternoon>Sunny &amp;amp; hot</afternoon>
  Evening: <evening>Clear and Cooler</evening>
</weatherReport>
```

Another Example:

```
<?xml version = "1.0"?>
<article>
<title>Simple XML</title>
 <date>September 19, 2001</date>
 <author>
        <firstName>Tem</firstName>
         <lastName>Nieto</lastName>
 </author>
 <summary>XML is pretty easy.</summary>
 <content>Once you have mastered XHTML, XML is easily
learned. You must remember that XML is not for
displaying information but for managing information.
 </content>
```

</article>

Overall structure:

- An XML document may start with one or more processing instructions (PIs) or directives:

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/css" href="ss.css"?>
```

- Following the directives, there must be exactly *one* root element containing all the rest of the XML:

```
<weatherReport>
   ...
</weatherReport>
```

XML building blocks:

- Aside from the directives, an XML document is built from:
  - elements: high in <high scale="F">103</high>
  - tags, in pairs: <high scale="F">103</high>
  - attributes: <high scale="F">103</high>
  - entities: <afternoon>Sunny & hot</afternoon>
  - character data, which may be:
    - parsed (processed as XML)--this is the default
    - unparsed (all characters stand for themselves)

Elements and attributes:

- Attributes and elements are somewhat interchangeable
- Example using just elements:

```
<name>
   <first>David</first>
   <last>Matuszek</last>
</name>
```

- Example using attributes:

  <name first="David" last="Matuszek"></name>

- You will find that elements are easier to use in your programs--this is a good reason to prefer them
- Attributes often contain metadata, such as unique IDs
- Generally speaking, browsers display only elements (values enclosed by tags), not tags and attributes

Well-formed XML Document:

- Every element must have *both* a start tag and an end tag, e.g. <name> ... </name>
  - But empty elements can be abbreviated: <break />.
  - XML tags are case sensitive
  - XML tags may not begin with the letters xml, in any combination of cases
- Elements must be properly nested, e.g. *not* bold and italic
- Every XML document must have one and only one root element
- The values of attributes must be enclosed in single or double quotes, e.g. <time unit="days">
- Character data cannot contain < or &

Valid XML Document:

- You can make up your own XML tags and attributes, *but*...
  - ...any program that *uses* the XML must know what to expect!
- A DTD (Document Type Definition) defines what tags are legal and where they can occur in the XML
- An XML document *does not require* a DTD
- XML is well-structured if it follows the rules and Syntax of XML
- In addition, XML is valid if it declares a DTD and conforms to that DTD
- A DTD can be included in the XML, but is typically a separate document

- Errors in XML documents will *stop* XML programs

Some alternatives to DTDs are XML Schemas and RELAX NG

Entities:

- Five special characters must be written as entities:

  & for & (almost always necessary)

  < for < (almost always necessary)

  > for > (not usually necessary)

  " for " (necessary inside double quotes)

  ' for ' (necessary inside single quotes)

- These entities can be used even in places where they are not absolutely required
- These are the *only* predefined entities in XML

XML declaration:

- The XML declaration looks like this:
  <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
  - The XML declaration is not required by browsers, but *is* required by most XML processors (so include it!)
  - If present, the XML declaration must be first--*not even whitespace* should precede it
  - Note that the brackets are <? and ?>
  - version="1.0" is required (this is the *only* version so far)
  - encoding can be "UTF-8" (ASCII) or "UTF-16" (Unicode), or something else, or it can be omitted
  - standalone tells whether there is a separate DTD

Processing instructions:

- PIs (Processing Instructions) may occur anywhere in the XML document (but usually first)

- A PI is a command to the program processing the XML document to handle it in a certain way
- XML documents are typically processed by more than one program
- Programs that do not recognize a given PI should just ignore it
- General format of a PI: <?*target instructions*?>
- Example: <?xml-stylesheet type="text/css" href="mySheet.css"?>

Comments:

- <!-- This is a comment in both HTML and XML -->
- Comments can be put anywhere in an XML document
- Comments are useful for:
    - Explaining the structure of an XML document
    - Commenting out parts of the XML during development and testing
- Comments are not elements and do not have an end tag
- The blanks after <!-- and before --> are optional
- The character sequence -- cannot occur in the comment
- The closing bracket *must* be -->
- Comments are not displayed by browsers, but can be seen by anyone who looks at the source code

Names in XML:

- Names (as used for tags and attributes) must begin with a letter or underscore, and can consist of:
    - Letters, both Roman (English) and foreign
    - Digits, both Roman and foreign
    - . (dot)
    - - (hyphen)

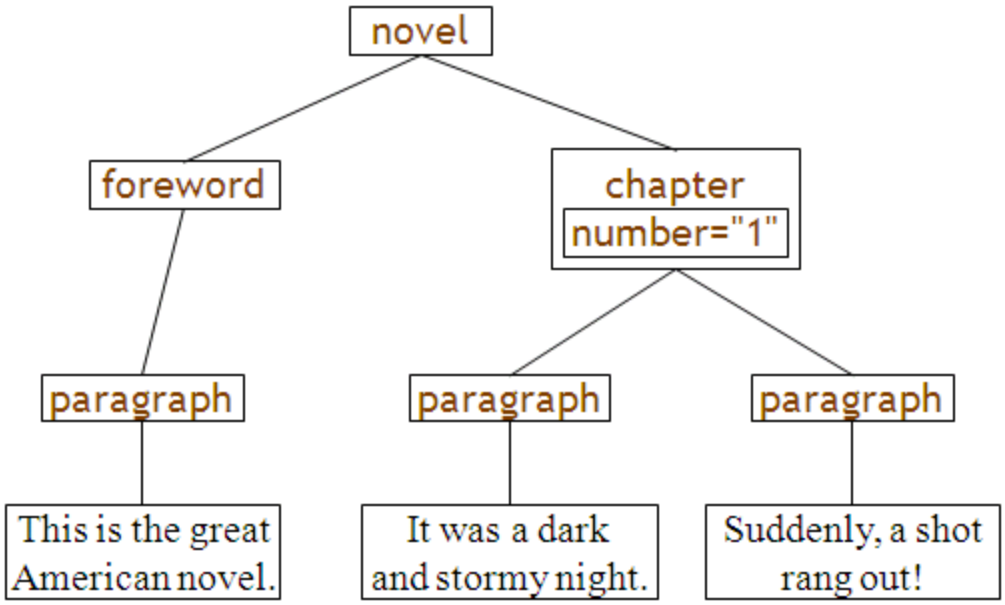
- _ (underscore)
- : (colon) should be used only for namespaces
- Combining characters and extenders (not used in English)

Another well-structured example:

```
<novel>
  <foreword>
    <paragraph> This is the great American novel.
    </paragraph>
</foreword>
  <chapter number="1">
    <paragraph>It was a dark and stormy night.
   </paragraph>
    <paragraph>Suddenly, a shot rang out!
    </paragraph>
  </chapter>
</novel>
```

XML as a tree:

- An XML document represents a hierarchy; a hierarchy is a tree



Document Type Definition (DTD)

- The purpose of a DTD is to define the building blocks of an XML document.
- It defines the Document structure with a list of legal elements.
- A DTD can be declared in two ways
  - Inline DTD Declaration in the XML document itself
  - External DTD reference to XML Document.

Internal DTD Declaration:

- If the DTD is included in the XML source file, DTD definition should be wrapped in a DOCTYPE definition along with XML Definition.
- Syntax

<?xml version="1.0">

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

XML definition

Simple Example

```
<?xml version="1.0">

<!DOCTYPE note [

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

        <!ELEMENT to (#PCDATA)>

        <!ELEMENT from (#PCDATA)>

        <!ELEMENT heading (#PCDATA)>

        <!ELEMENT body (#PCDATA)>

 ]>

<note>

        <to> Tove </to>
```

<from> Jani</from>

<heading>Reminder </heading>

<body> Don't forget me this weekend </body>

</note>

External DTD Declaration:

- If the DTD is external to the XML source file, the external DTD file should be specified in a DOCTYPE definition without writing DTD definition again.
- Syntax

<?xml version="1.0">

<!DOCTYPE root-element SYSTEM "filename">

XML definition

Simple Example:

This is a copy of the file "note.dtd" containing the DTD

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

<!ELEMENT to (#PCDATA)>

<!ELEMENT from (#PCDATA)>

<!ELEMENT heading (#PCDATA)>

<!ELEMENT body (#PCDATA)>

This is the XML document with an external DTD

<?xml version="1.0">

<!DOCTYPE note SYSTEM "note.dtd">

<note>

<to> Tove </to>

<from> Jani</from>

<heading>Reminder </heading>

<body> Don't forget me this weekend </body>

</note>

Basic Building Blocks of XML Document:

- According to DTD point of view , all XML documents are made up by the building blocks like
  - Elements
  - Tags
  - Attributes
  - Entities
  - CDATA
  - PCDATA

CDATA and PCDATA:

- CDATA
  - CDATA means character data which is a text that will NOT be parsed by XML parsers.
  - Tags inside the text will NOT be treated as markup and entities will not be expanded.
- PCDATA
  - PCDATA means Parsable character data which is a text that will be parsed by XML Parser
  - Tags inside the text will NOT be treated as markup and entities will not be expanded.

Element Declaration in DTD:

- In the DTD, XML elements are declared with element declaration.
- Syntax of an element declaration

<!ELEMENT element-name category>

Or

<!ELEMENT element-name (element-content)>

Different types of Element Declarations:

- Empty Element Declaration
  - Empty elements are declared with the category keyword EMPTY
  - Syntax
    - <!ELEMENT element-name EMPTY>
  - DTD Example
    - <!ELEMENT br EMPTY>
  - XML Example
    - < br />
- Elements with only character data
  - Elements with only character data are declared with #PCDATA inside parenthesis
  - Syntax
    - <!ELEMENT element-name (#PCDATA)>
  - Example
    - <!ELEMENT from (#PCDATA)>
- Elements with a sequence of children
  - Elements with one or more children are defined with the name of the children elements inside parenthesis
  - Syntax
    - <!ELEMENT element-name (child element1, child element2, child element3,..) >
    - Example

- <!ELEMENT note (to , from ,heading ,body)>

- When children are declared in a sequence seperated by comma, the children must be appear in the same sequence in the document.

- Declaring only one occurrence of the element
  - Syntax
    - <!ELEMENT element –name (child-name)>
  - Example
    - <!ELEMENT note(to,from.heading,body)>
- Declaring minimum one occurrence of the same element
  - Syntax
    - <!ELEMENT element-name (child-name+)>
  - Example
    - <!ELEMENT students(students+)>
- Declaring zero or more occurrence of the same content
  - Syntax
    - <!ELEMENT element-name (child-name*)>
  - Example
    - <!ELEMENT students(students*)>
- Declaring zero or one occurrence of the same element
  - Syntax
    - <!ELEMENT element-name (child-name?)>
  - Example
    - <!ELEMENT note (message?)>
- Declaring either/or content

- Syntax
  - <!ELEMENT element-name(child-name1|child-name2)
- Example
  - <!ELEMENT note(to,from,header,(message|body))>

- Declaring Mixed Content
  - Syntax
    - <!ELEMENT element-name(child-name1 | child-name2,..)>
  - Example
    - <!ELEMENT note(#PCDATA | to | from | header | message)>

- **Declaring Attributes:**
  In the DTD, XML element attributes are declared with an ATTLIST declaration. An attribute declaration has the syntax like:
  - <!ATTLIST element-name attribute-name attribute-type default-value>

- For the attributes it have the properties like
  - Name of the attribute
  - Type of the attributes
  - Default value if any.

Attribute Types:

Value	Explanation
CDATA	The value is character data
(eval|eval|..)	The value must be an enumerated value
ID	The value is an unique id
IDREF	The value is the id of another element
IDREFS	The value is a list of other ids
NMTOKEN	The value is a valid XML name
NMTOKENS	The value is a list of valid XML names

ENTITY	The value is an entity
ENTITIES	The value is a list of entities
NOTATION	The value is a name of a notation
xml:	The value is predefined

Attribute Default Value:

- The last entry in the attribute specification determines the attributes default value, if any, and tells whether or not the attribute is required.

Specification	*Specifies...*
#REQUIRED	The attribute value must be specified in the document.
#IMPLIED	The value need not be specified in the document. If it isn't, the application will have a default value it uses.
"defaultValue"	The default value to use, if a value is not specified in the document.
#FIXED "fixedValue"	The value to use. If the document specifies any value at all, it must be the same.

Attribute declaration examples:

- Default attribute value
  - Syntax:
    - <!ATTLIST element-name attribute-name CDATA "default-value">
  - DTD example:
    - <!ATTLIST payment type CDATA "check">
  - XML example:
    - <payment type="check">

- Specifying #IMPLIED
  - Syntax:
    - <!ATTLIST element-name attribute-name attribute-type #IMPLIED>
  - DTD example: <!ATTLIST contact fax CDATA #IMPLIED>
  - XML example: <contact fax="555-667788">
- Specifying #REQUIRED
  - Syntax:
    - <!ATTLIST element-name attribute_name attribute-type #REQUIRED>
  - DTD example:
    - <!ATTLIST person number CDATA #REQUIRED>
  - XML example:
    - <person number="5677">
- Specifying #FIXED
  - Syntax:
    - <!ATTLIST element-name attribute-name attribute-type #FIXED "value">
  - DTD example:
    - <!ATTLIST sender company CDATA #FIXED "Microsoft">
  - XML example:
    - <sender company="Microsoft">
- Enumerated attribute values
  - Syntax:
    - <!ATTLIST element-name attribute-name (val1|val2|..) default-value>
  - DTD example:

        - <!ATTLIST payment type (check|cash) "cash">
    - XML example:
        - <payment type="check"> or <payment type="cash">

Declaring Entities:

- Entities are the variables used to define shortcuts to common text.
- <!ENTITY> PI is used to declare user-defined entities.
- Syntax
    - <!ENTITY entity-name "entity-value">
- DTD Example
    - <!ENTITY writer "Donald Duck.">
    - <!ENTITY copyright "Copyright W3Schools>
- XML Example

<author> &writer; &copyright; </author>

Namespaces:

- Recall that DTDs are used to define the tags that can be used in an XML document
- An XML document may reference more than one DTD
- Namespaces are a way to specify which DTD defines a given tag
- XML, like Java, uses qualified names
    - This helps to avoid collisions between names
    - Java: myObject.myVariable
    - XML: myDTD:myTag
    - Note that XML uses a colon (:) rather than a dot (.)

Namespaces and URIs:

- A namespace is defined as a unique string

- To guarantee uniqueness, typically a URI (Uniform Resource Indicator) is used, because the author "owns" the domain
- It doesn't have to be a "real" URI; it just has to be a unique string
- Example: http://www.matuszek.org/ns

- There are two ways to use namespaces:
  - Declare a default namespace
  - Associate a prefix with a namespace, then use the prefix in the XML to refer to the namespace

Namespace syntax:

- In *any* start tag you can use the reserved attribute name xmlns:

  <book xmlns="http://www.matuszek.org/ns">

  This namespace will be used as the default for all elements up to the corresponding end tag

  You can override it with a specific prefix

- You can use almost this same form to declare a prefix:

  <book xmlns:dave="http://www.matuszek.org/ns">

  Use this prefix on *every tag and attribute* you want to use from this namespace, including end tags--it is *not* a default prefix

  <dave:chapter dave:number="1">To Begin</dave:chapter>

- You can use the prefix in the start tag in which it is defined:

<dave:book xmlns:dave="http://www.matuszek.org/ns">

XML Schema:

- What is XML Schema?
  - XML Schema is vocabulary for expressing constraints for the validity of an XML document.

- A piece of XML is valid if it satisfies the constraints expressed in another XML file, the schema file.
- The idea is to check if the XML file is fit for a certain purpose.
- When we say "XML Schemas," we usually mean the W3C XML Schema Language
  - This is also known as "XML Schema Definition" language, or xs

Need of XML Schema:

- DTDs provide a very weak specification language
  - We can't put any restrictions on text content
  - We have very little control over mixed content (text plus elements)
  - We have little control over ordering of elements
- DTDs are written in a strange (non-XML) format
  - We need separate parsers for DTDs and XML
- The XML Schema Definition language solves these problems
  - xs gives us much more control over structure and content
  - xs is written in XML

Disadvantages of XML Schema:

- DTDs have been around longer than xs
  - Therefore they are more widely used
  - Also, more tools support them
- xs is very verbose, even by XML standards
- More advanced XML Schema instructions can be non-intuitive and confusing
- Nevertheless, xs is not likely to go away quickly

Referring to a schema:

- To refer to a DTD in an XML document, the reference goes *before* the root element:

- <?xml version="1.0"?>
  <!DOCTYPE rootElement SYSTEM "*url*">
  <rootElement> ... </rootElement>

- To refer to an XML Schema in an XML document, the reference goes *in* the root element:

  - <?xml version="1.0"?>
    <rootElement
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            (The XML Schema Instance reference is required)
      xsi:schemaLocation="*url* xs">
            (This is where *your* XML Schema definition can be found)
      ...
    </rootElement>

The xs Document:

- To prepare the XML document structure for the XML file, first we need to prepare the xs document.
- XML Schema is itself be XML file
- The file extension is .xs
- The root element is <schema>
- The xs starts like this:

  - <?xml version="1.0"?>
    <xs:schema xmlns:xs="http://www.w3.rg/2001/XMLSchema">

<schema>:

- The <schema> element may have attributes:

  - xmlns:xs="http://www.w3.org/2001/XMLSchema"

    - This is necessary to specify where all our xs tags are defined

  - elementFormDefault="qualified"

    - This means that all XML elements must be qualified with namespace.

  - targetNamespace=http://www.w3schools.com

- indicates that all elements defined by this schema come from the http://www.w3schools.com namespace.

- Xmlns=http://www.w3schools.com
  - Indicates that the default namespace is "http://www.w3schols.com".

"Simple" and "complex" elements:

- A "simple" element is one that contains text and nothing else
  - A simple element cannot have attributes
  - A simple element cannot contain other elements
  - A simple element cannot be empty
  - However, the text can be of many different types, and may have various restrictions applied to it
- If an element isn't simple, it's "complex"
  - A complex element may have attributes
  - A complex element may be empty, or it may contain text, other elements, or both text and other elements

Defining a simple element:

- A simple element is defined as
  <xs:element name="*name*" type="*type*" />
  where:
  - *name* is the name of the element
  - the most common values for *type* are

xs:boolean	xs:integer
xs:date	xs:string
xs:decimal	xs:time

- Other attributes a simple element may have:
  - default="*default value*" *if no other value is specified*
  - fixed="*value*" *no other value may be specified*

- **Simple Example:**
  xs Example

  <xs:element name="lastname" type="xs:string" />

  <xs:element name="age" type="xs:string" />

  <xs:element name="dateborn" type="xs:date" />

- XML Example

  <lastname> Vishnu </lastname>

  <age>34 </age>

  <dateborn> 1968-03-27 </dateborn>

- Providing Default value Example
  - <xs:element name="color" type=xs:string default="red" />
- Providing Fixed Value Example
  - <xs:element name="color" type="xs:string" fixed="red" />

Defining an attribute:

- Attributes themselves are always declared as simple types
- An attribute is defined as
  <xs:attribute name="*name*" type="*type*" />
  where:
  - *name* and *type* are the same as for xs:element
- Other attributes a simple element may have:
  - default="*default value*" *if no other value is specified*
  - fixed="*value*" *no other value may be specified*
  - use="optional" *the attribute is not required (default)*
  - use="required" *the attribute must be present*

Simple Example:

- xs Example
  - <xs:attribute name=“lang” type=“xs:string”>
- XML Example
  - <lastname lang=“EN”>Smith</lastname>
- Providing Default Value to the attribute
  - <xs:attribute name=“lang” type=“xs:string” default=“EN” />
- Providing Fixed Value to the attribute
  - <xs:attribute name=“lang” type=“xs:string” fixed=“EN” />

Restrictions on Content:

- With XML Schema , we can also add our own restrictions tyo our XML elements and attributes. These restrictions are called as “Facets”.
- The general form for putting a restriction on an element:
  - <xs:element name="*name*"> *(or* xs:attribute*)*
    <xs:restriction base="*type*">
    ... the restrictions ...
    </xs:restriction>
    </xs:element>
- For example:
  - <xs:element name="age">
    <xs:restriction base="xs:integer">
    <xs:minInclusive value="0">
    <xs:maxInclusive value=“60">
    </xs:restriction>
    </xs:element>

Restrictions on numbers:

- minInclusive -- number must be ≥ the given *value*
- minExclusive -- number must be > the given *value*
- maxInclusive -- number must be ≤ the given *value*

- maxExclusive -- number must be < the given *value*
- totalDigits -- number must have exactly *value* digits
- fractionDigits -- number must have no more than *value* digits after the decimal point

Restrictions on strings:

- length -- the string must contain exactly *value* characters
- minLength -- the string must contain at least *value* characters
- maxLength -- the string must contain no more than *value* characters
- pattern -- the *value* is a regular expression that the string must match
- whiteSpace -- not really a "restriction"--tells what to do with whitespace
    - value="preserve" Keep all whitespace
    - value="replace" Change all whitespace characters to spaces
    - value="collapse" Remove leading and trailing whitespace, and replace all sequences of whitespace with a single space

Enumeration – Restrictions on set of Values:

- An enumeration restricts the value to be one of a fixed set of values
- Example:

```
<xs:element name="season">
   <xs:simpleType>
      <xs:restriction base="xs:string">
         <xs:enumeration value="Spring"/>
         <xs:enumeration value="Summer"/>
         <xs:enumeration value="Autumn"/>
         <xs:enumeration value="Fall"/>
         <xs:enumeration value="Winter"/>
      </xs:restriction>
   </xs:simpleType>
</xs:element>
```

Complex elements:

- A complex element is defined as

```
<xs:element  name="name">
   <xs:complexType>
      ... information about the complex type...
   </xs:complexType>
</xs:element>
```

- Example:

```
<xs:element  name="person">
   <xs:complexType>
      <xs:sequence>
         <xs:element  name="firstName"  type="xs:string" />
         <xs:element  name="lastName"  type="xs:string" />
      </xs:sequence>
   </xs:complexType>
</xs:element>
```

- <xs:sequence> says that elements must occur in this order
- However attributes are always simple types

Global and local definitions:

- Elements declared at the "top level" of a <schema> are available for use throughout the schema
- Elements declared within a xs:complexType are local to that type
- Thus, in

```
<xs:element  name="person">
   <xs:complexType>
      <xs:sequence>
         <xs:element  name="firstName"  type="xs:string" />
         <xs:element  name="lastName"  type="xs:string" />
      </xs:sequence>
   </xs:complexType>
</xs:element>
```

  the elements firstName and lastName are only locally declared

- The order of declarations at the "top level" of a <schema> *do not* specify the order in the XML data document

- **xs:all:**
  xs:all allows elements to appear in any order

- ```
  <xs:element name="person">
    <xs:complexType>
      <xs:all>
        <xs:element name="firstName" type="xs:string" />
        <xs:element name="lastName" type="xs:string" />
      </xs:all>
    </xs:complexType>
  </xs:element>
  ```

- Despite the name, the members of an xs:all group can occur once or not at all

- You can use minOccurs="*n*" and maxOccurs="*n*" to specify how many times an element may occur (default value is 1)

In this context, *n* may only be 0 or 1

Mixed elements:

- Mixed elements may contain both text and elements

- We add mixed="true" to the xs:complexType element

- The text itself is not mentioned in the element, and may go anywhere (it is basically ignored)

- ```
  <xs:complexType name="paragraph" mixed="true">
    <xs:sequence>
      <xs:element name="someName" type="xs:anyType"/>
    </xs:sequence>
  </xs:complexType>
  ```

Predefined string types:

- A simple element is defined as:
  <xs:element name="*name*" type="*type*" />

- Here are a few of the possible string types:

  - xs:string -- a string

- xs:normalizedString -- a string that doesn't contain tabs, newlines, or carriage returns
- xs:token -- a string that doesn't contain any whitespace other than single spaces

- Allowable restrictions on strings:
  - enumeration, length, maxLength, minLength, pattern, whiteSpace

Predefined date and time types:

- xs:date -- A date in the format *YYYY-MM-DD*, for example, 2002-11-05
- xs:time -- A date in the format *hh:mm:ss* (hours, minutes, seconds)
- xs:dateTime -- Format is *YYYY-MM-DD*T*hh:mm:ss*
- Allowable restrictions on dates and times:
  - enumeration, minInclusive, maxExclusive, maxInclusive, maxExclusive, pattern, whiteSpace

Predefined numeric types:

- Here are some of the predefined numeric types:
- xs:decimal xs:positiveInteger
- xs:byte xs:negativeInteger
- xs:short xs:nonPositiveInteger
- xs:int xs:nonNegativeInteger
- xs:long
- Allowable restrictions on numeric types:
  - enumeration, minInclusive, maxExclusive, maxInclusive, maxExclusive, fractionDigits, totalDigits, pattern, whiteSpace

```
<?xml version="1.0"?>

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"

        targetNamespace="http://www.books.org"
```

```
            xmlns="http://www.books.org"
            elementFormDefault="qualified">
<xs:element name="BookStore">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="Book" minOccurs="1" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="Book">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="Title" minOccurs="1" maxOccurs="1"/>
      <xs:element ref="Author" minOccurs="1" maxOccurs="1"/>
      <xs:element ref="Date" minOccurs="1" maxOccurs="1"/>
      <xs:element ref="ISBN" minOccurs="1" maxOccurs="1"/>
      <xs:element ref="Publisher" minOccurs="1" maxOccurs="1"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="Title" type="xs:string"/>
<xs:element name="Author" type="xs:string"/>
<xs:element name="Date" type="xs:string"/>
<xs:element name="ISBN" type="xs:string"/>
```

```
    <xs:element name="Publisher" type="xs:string"/>
</xs:schema>
```

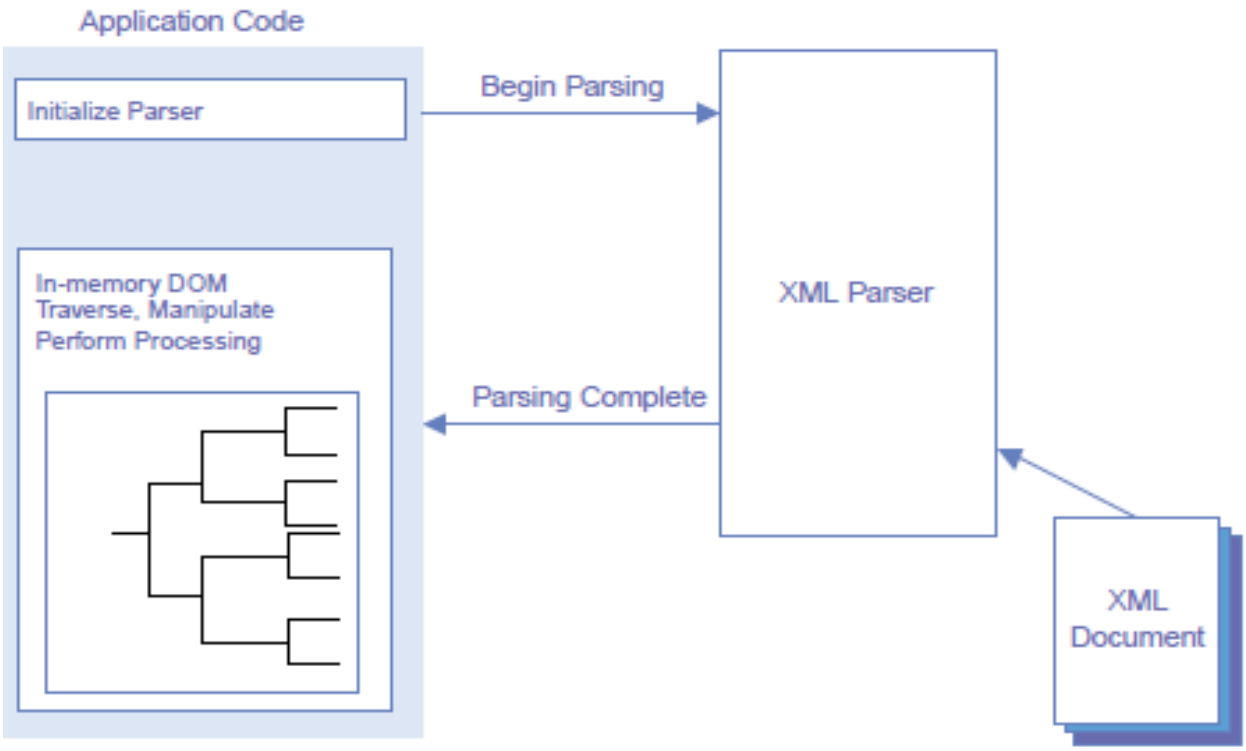
XML Parser:

- An *XML parser* is a software component that can read and validate any XML document.
- A parser makes data contained in an XML data structure available to the application that needs to use it.
- Numerous XML parsers have been developed like
  - DOM
  - SAX
  - Crimson
  - Xerces
  - JAXP

Document Object Model (DOM):

- In the Sun's implementation of DOM model, the parser will read in an entire XML data source and construct a treelike representation of it in memory.
- Under DOM, a pointer to the entire document is returned to the calling application.
- The application can then manipulate the document, rearranging nodes, adding and deleting content as needed by using DOM API.
- While DOM is generally easier to implement, it is far slower and more resource intensive .
- DOM can be used effectively with smaller XML data structures in situations when speed is not of paramount importance to the application.

Using the DOM API



DOM API Packages:

The Document Object Model implementation is defined in the following packages:

org.w3c.dom	Defines the DOM programming interfaces for XML (and, optionally, HTML) documents, as specified by the W3C.
com.sun.xml.tree	Sun's Java XML implementation of the DOM libraries, including the XmlDocument, XmlDocumentBuilder, and TreeWalker classes.
Javax.xml.parsers	Defines the DocumentBuilderFactory class and DocumentBuilder which returns an object that implements the W3C Document interface. This package also defines the ParserConfigurationException class for reporting errors.

Node Interfaces:

- All components in xml document are represented by different kind of nodes

- Package org.w3c.dom
  - Document
  - Element
  - Attribute
  - Text
  - ProcessingInstruction
  - Comment
  - CDATASection
  - PCDATASection
  - DocumentType
  - Entity
  - Namespace
  - Children

Some DOM Methods:

- Node information
  - short getNodeType();
  - String getNodeName();
  - String getNodeValue();
- Namespace information
  - String getPrefix();
  - String getLocalName();
  - String getNamespaceURI();
- Editing node values

- void setNodeValue(String value);
- void setPrefix(String prefix);

- Editing tree structure
  - Node appendChild(Node child);
  - Node insertBefore(Node newChild, Node refChild);
  - Node removeChild(Node child);
  - Node replaceChild(Node newChild, Node oldChild);
- Attributes
  - boolean hasAttribute(String name);
  - NamedNodeMap getAttributes();
- Children
  - Node getFirstChild();
  - Node getLastChild();
  - boolean hasChildNodes();
  - NodeList getChildNodes();
- Links to other nodes
  - Document getOwnerDocument();
  - Node getParentNode();
  - Node getPreviousSibling();
  - Node getNextSibling();
- Query methods
  - int getLength();
  - Node item(int index);
  - Node getNamedItem(String name);

- Node getNamedItemNS(String namespaceURI, String localName);

- Editing methods
  - Node setNamedItem(Node node);
  - Node setNamedItemNS(Node node);
  - Node removeNamedItem(String name);
  - Node removeNamedItemNS(String namespaceURI, String localName);