UNIT I
INTRODUCTION TO HTML

INTRODUCTION:

HTML stands for Hypertext Markup Language. It is a Language used to createWeb Pages or Hypertext document. A Markup Language is a set of instructions often called TAGS which can be addedto text files. HTML is only a formatting language are not a programming language. The idea behind hypertext is that instead of reading text in a right linear structure we can easily jump from one point to another point.HTML is all about specifying the structure and format of our webpage i.e, it is mainly used for describing the structure document.

HTML is platform independent i.e, for example if we can access internet, we can access WORLD WIDE WEB (WWW) irrespective of client OS and OS of the webserver are accessing. So, we can view one download HTML files on www throughbrowser.

Elements of a web document are labelled through the usage of HTML tags. It is the tags that describe the document. Anything that is not a tag will bedisplayed in the document itself. HTML does not describe any page layout i.e, for example, word for windows havedifferent styles for headings, font size etc. But HTML doesn't have all these. Based on the Platforms, appearance of any element will change. The formatted textwill appear differently on different machines / Platforms. By separating the Structure of the document and appearance, a Program that reads and Understands HTML can make formatting decision based on capabilities of individual Platform. Web Browsers are best examples of HTML formatters.

Advantages of HTML:-

- A HTML document is small and hence easy to send over the net. It is smallbecause it doesn't include format information.
- HTML documents are cross platform compatible and device independent.We need a HTML readable browser to view them.

Basic HTML tags:-

(1) **<!doctype> :**

This tag formally starts an HTML document and it also indicates theversion of HTML used.

<!doctype HTML PUBLIC "//w3c//DTDHTML Q.o//EN">

(2) **<HTML>:**

Every HTML document starts with a <html> tag and it is always the firsttag in a html page and indicates that the document is a HTML document.The end tag <html> is </html>.

Example:

<html>......... </html>

(3) **<head>:**

It contains the head of an html document, which holds about the document such as title. Each property defined html page should have a head which we create with <head> tag. It has header information and it isdisplayed at the top of the browser. Each tag for <head> is </head>.

<head>.........</head>

(4) **<title>:**

It contains the title of the html document which includes the content thatwill actually appear in the web browser. The entire content of the web page is placed in the pages <body> tag. The end tag <body> is </body>

<title>........ </title>

(5) **<body>:**

It contains the body of the HTML Document , which includes the contentthat will actuall appear in the web browser. The entire content of the webpage will be placed in the pages <body> tag. The end tag of the
<body> tag will be </body>.

<body>........... </body>

STRUCTURE OF THE HTML PROGRAM:-

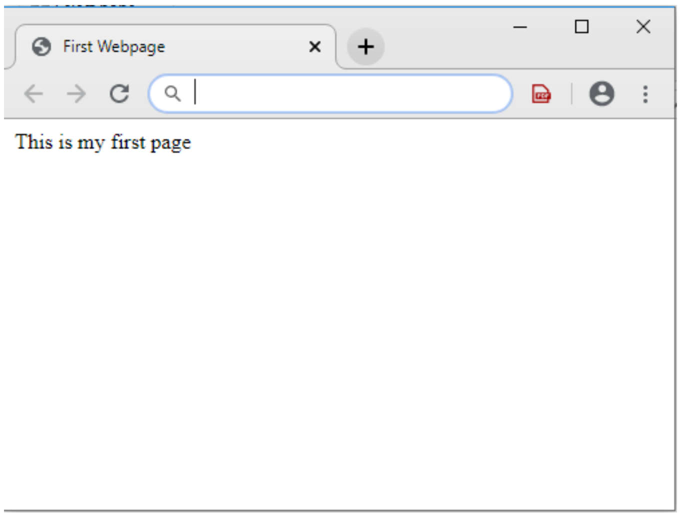
The HTML Program is generally divided into two sections i.e head and body. Weuse <head> and <body> tags to indicate these two sections. <head> section holds the header information of a webpage document indicated by a title that is provided by using <title> tag in the <head>. The title helps us to referto the webpage. <body> section contains the content which we want to display which the webpage.Anything that is not a tag will be displayed within the webpage.

Example:

```
<html>
        <head>
                <title>First Webpage</title>

        </head>
        <body>
                This is my first page
        </body>
</html>
```

Output:



This is my first page

Attribute:

An Attribute is a Keyword we use in an opening tag to give more information tothe web browser. HTML tags tell the web browsers how to format and organizeour webpages. But we can customize tags using attributes. The Format of an attribute is:

<tagname Attribute=value>

Attributes of the <body> tag:

(1) Background:

The URL or a graphic file to be used in the filling the browser'sBackground.

(2) Bgcolor:

The color of the browser's background.

(3) Bgproperties:

It Indicates if the background should scroll when text does. If we set it to"FIXED", the background will not scroll when the text does.

(4) Bottommargin:

Specifies the bottom margin ,the empty space at the bottom of the documentin pixels.

(5) Id:

It is a unique alphanumeric identifier for the tag which we can use to refer toit.

(6) Language:

Scripting language used for the tag.

(7) Leftmargin:

Specifies the left margin, the empty space at the left of the document.

(8) Marginheight:

Gives the height of the margin at the top and bottom of the page in pixels.

(9) MarginWidth:

Gives the width of the left and right margins of the page in pixels.

(10) Rightmargin:

It specifies the right margin, the empty space to the right margin of thedocument in pixels.

(11) Scroll:

It specifies whether a vertical scrollbar appears to the right of the documentcan be yes (or) no.

(12) Style:

Inline style indicating how to render the element.

(13) Text:

Color of the in the document.

(14) Topmargin:

It specifies the top margin the space at the top of the document in pixels.

(15) Link:

It specifies the color of hyperlinks that have not yet been visited.

(16) Alink:

It specifies the color of hyperlinks as they are being clicked.

(17) Vlink:

It specifies the color of hyperlinks as they have been visited.

(18) <! -------> Comment tag:

Annotates a web page with a comment. In the HTML that we can by lookingat the HTML but it will not be displayed in the web browser.

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

Formatting with HTML tags:

To set the actual style of text as displayed in a web page we can text style tags.There are a number of ways to apply styles to text.

(1) <b>:

It creates a bold text i.e, sets the text style to bold.

Attributes:

a. Id:

It is a unique alphanumeric identifier for the tag which we can use to refer toit.

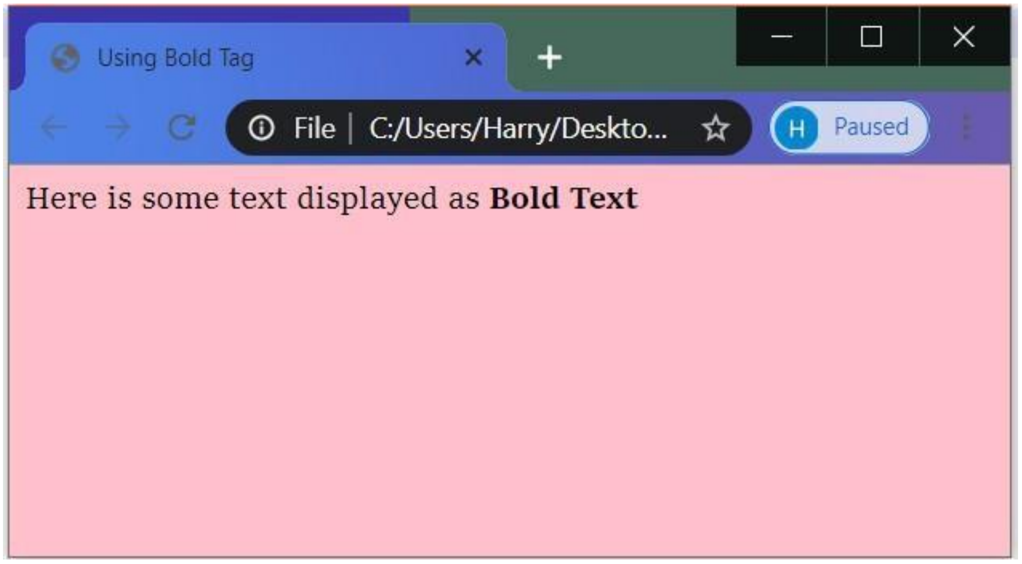
b. Style:

The Inline style indicating how to render the element.

Example:

```
<html>
        <head>
                <title>Using Bold Tag </title>
        </head>
        <body bgcolor=”pink”>
                Here is some text displayed as <b> Bold Text </b>
        </body>
</html>
```

Output:



Here is some text displayed as **Bold Text**

(2)<I>:

It displays text in Italics.(3)

<U>:

It displays text in Underlined text.(4)

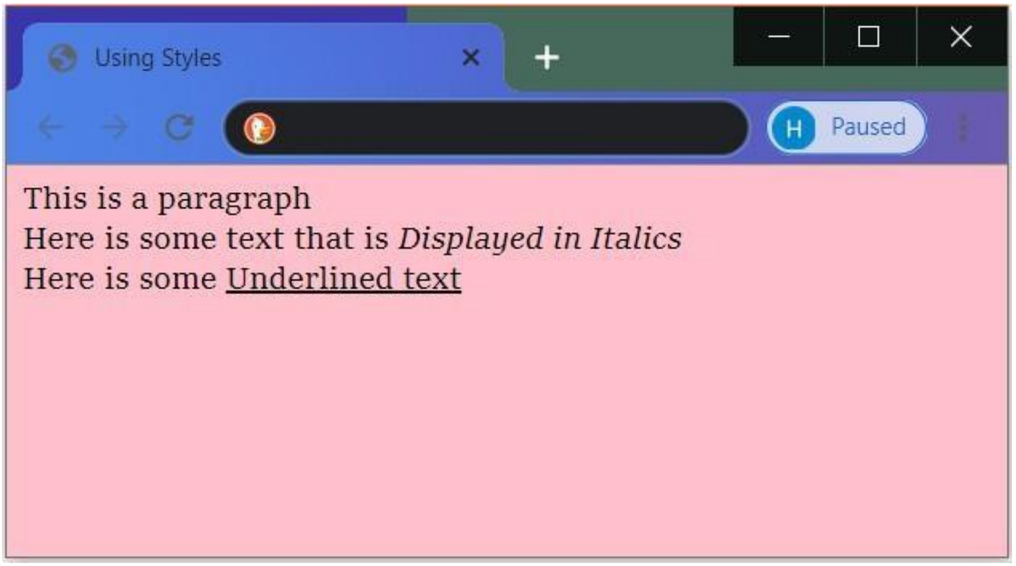
<P>:

It displays the Paragraph text.

Example:

```
<html>
        <head>
                <title> Using Styles </title>
        </head>
        <body bgcolor="pink">
                <p> This is a paragraph <br>
                Here is some text that is <i> Displayed in Italics </i>
                <br>Here is some <u> Underlined text </u>
        </body>
</html>
```

Output:



This is a paragraph
Here is some text that is *Displayed in Italics*
Here is some Underlined text

(5) <s> and <strike>:

It Displays text in smile through style. The <s> and <style> tags are used forthe same effect. HTML 2 used <strike> , HTML 3 called it <s>, HTML 3.2 caused it <strike> again.

(6) <big>:

Renders text in a bigger font than the current default.

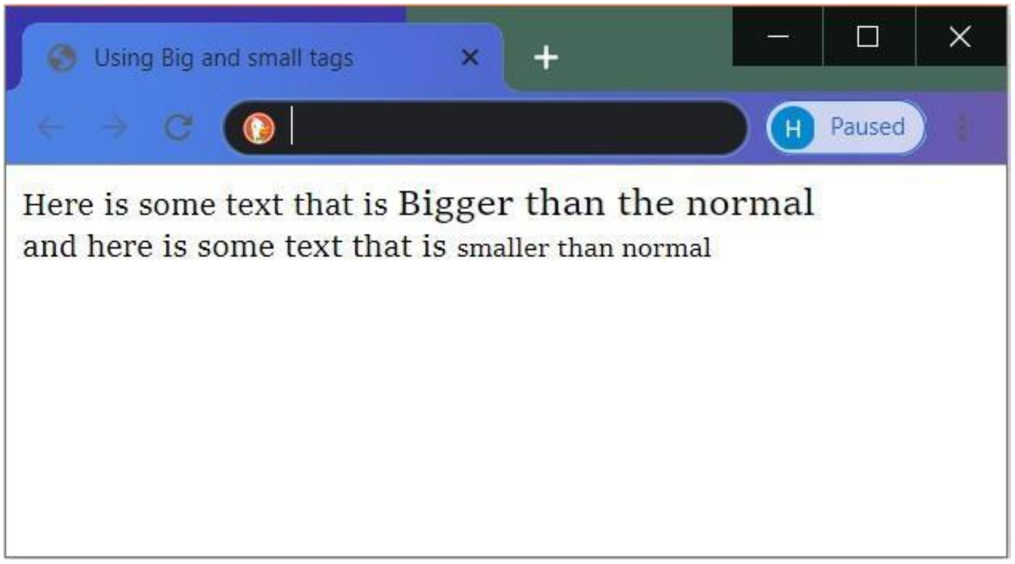
(7) <small>:

Renders text in a smaller font than the current default.

Example:

```
<html>
        <head>
                <title> Using Big and small tags </title>
        </head>
        <body>
                Here is some text that is <big> Bigger than the normal </big> <br>and here is
                some text that is <small> smaller than normal </small>
        </body>
</html>
```

Output:



Here is some text that is Bigger than the normal
and here is some text that is smaller than normal

(8) <sub>: It Styles the text as a subscript.

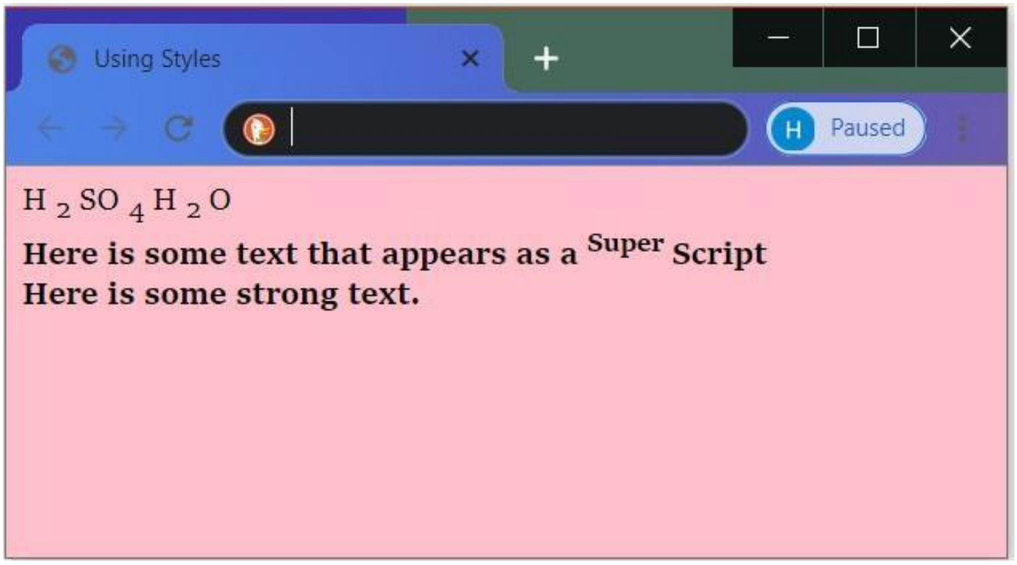
(9) <sup>: It Styles the text as a superscript.

(10) <strong>: Emphasizes text strongly, usually rendered in bold.

Example:

```
<html>
        <head>
                <title>Using Styles </title>
        </head>
        <body bgcolor=”pink”>
                H <sub> 2 </sub> SO <sub> 4 </sub> H
                <sub> 2 </sub> O
                <br>
        <b>Here is some text that appears as a <sup> Super </sup> Script <br>Here is some
<strong> strong </strong> text.
        </body>
</html>
```

Output:



H $_2$ SO $_4$ H $_2$ O

Here is some text that appears as a $^{\text{Super}}$ Script
Here is some strong text.

(11) Headings:

<h1>,<h2>,<h3>,<h4>,<h5> & <h6>

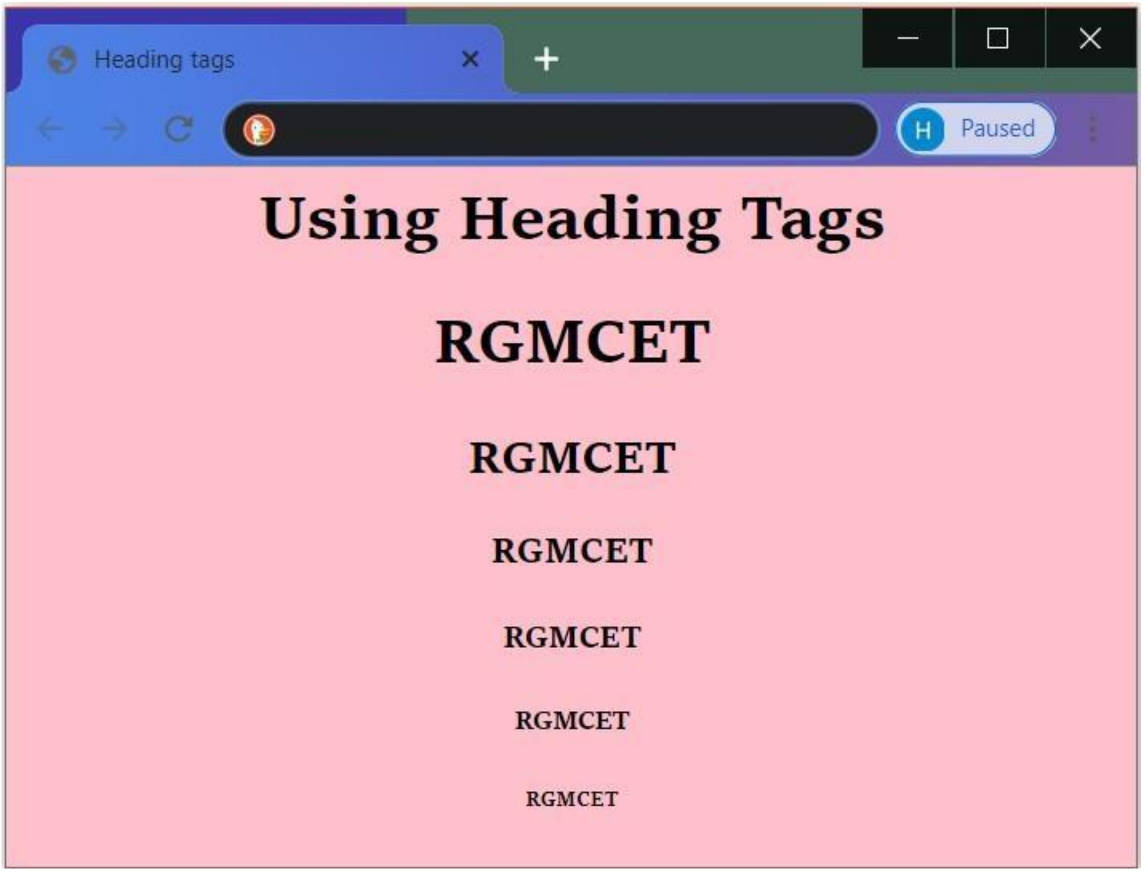
The heading element tags are <h1>,<h2>,<h3>,<h4>,<h5><h6>. These elements create the headings in our web pages by displaying bold text in avariety of sizes <h1> being larger <h6> being smaller.

Example:

```
<html>
        <head>
                <title> Heading tags </title>
        </head>
        <body bgcolor=”pink”>
                <center>
                <h1> Using Heading Tags</h1><br>
                <h1> RGMCET </h1><br>
                <h2> RGMCET </h2><br>
                <h3> RGMCET </h3><br>
                <h4> RGMCET </h4><br>
                <h5> RGMCET </h5><br>
```

```
                    <h6> RGMCET </h6><br>
                    </center>
          </body>
</html>
```

Output



Using Heading Tags

RGMCET

RGMCET

RGMCET

RGMCET

RGMCET

RGMCET

(12)<font>:

This tag will give us an option to select text size, color and face.

Attributes:

a. color: Color of the text.

b. Size: Size of the text in points

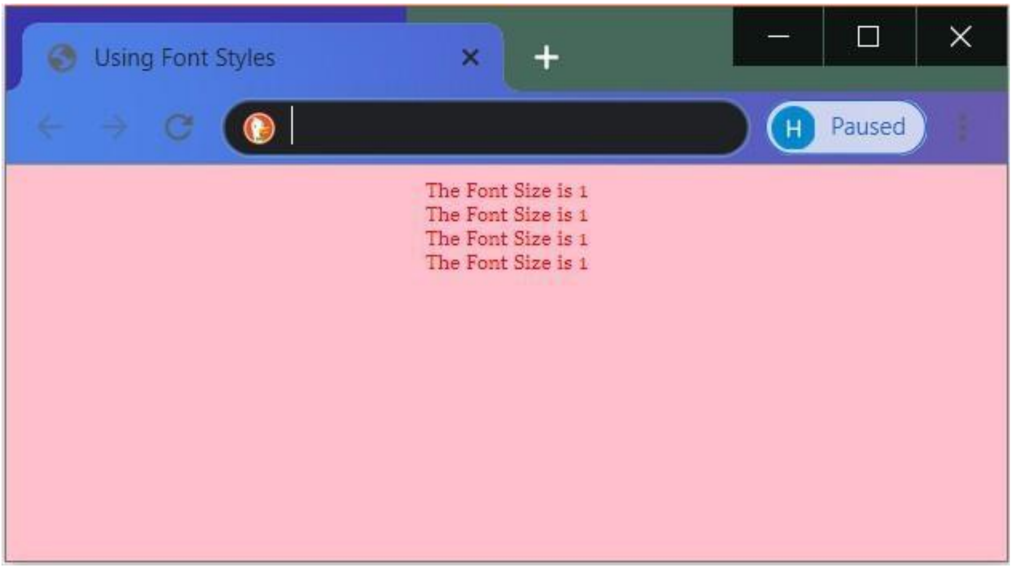
c. Face: The font face can be a list of names separated by commas.

d. Id: Unique alphanumeric identifier for a tag, which we can use to refer to it.

Example:

```
<html>
        <head>
                <title> Using Font Styles </title>
        </head>
        <body bgcolor=”pink”>
                <center>
                <font size=”1” color=”red”> The Font Size is 1 </font> <br>
                <font size=”10” color=”yellow”> The Font Size is 10 </font> <br>
                <font size=”20” color=”orange”> The Font Size is 20</font> <br>
                <font size=”30” color=”aqua”> The Font Size is 30 </font> <br>
                </center>
        </body>
</html>
```

Output



```
The Font Size is 1
The Font Size is 1
The Font Size is 1
The Font Size is 1
```

(13) <marquee> tag:

Displays scrolling text in a marquee style.

Attributes:

a. Align:

Sets the alignment of the text relative to marquee.Set to:

Top(default), middle (or) bottom.

b. behavior:

Sets how the text in the marquee should move can be scroll (default), slide(text enters from one side and stops at the other side), alternate (text seemsto bounce from one side to the other).

c. bgcolor:

It sets the background color for the marquee box.

d. Direction:

Sets the direction the text should scroll can be left, right, down or up.

e. Height:

It specifies the height of the marquee.

f. Loop:

Sets how many times we want the marquee to cycle. Is set to positive integeror -1 for continuous cycling.

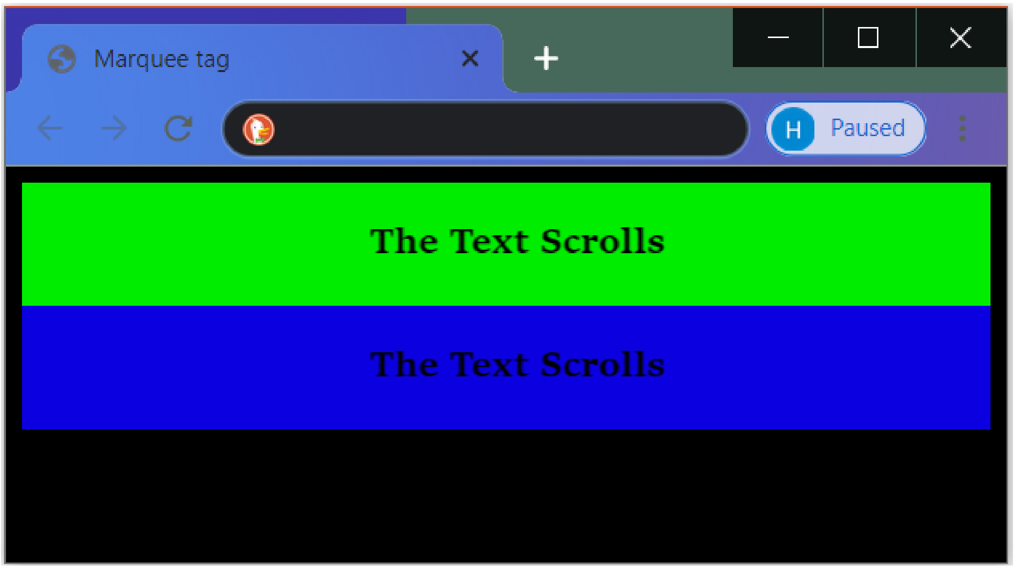
g. Scrolldelay:

Sets the number of the milliseconds between each successive display text.

Example:

```
<html>
        <head>
                <title> Marquee tag </title>
        </head>
        <body bgcolor=”pink”>
                <marquee    align=”top”    loop=”infinite”    behavior=”scroll”    bgcolor=”red”
direction=”right”> <h3> The Text Scrolls </h3></marquee>
<marquee    align=”middle”    loop=”infinite”    behavior=”slide”    bgcolor=”blue”
direction=”left”> <h3> The Text Scrolls </h3></marquee>
        </body>
</html>
```

Output



The Text Scrolls

The Text Scrolls

(14) <pre> tag(preformatted text):

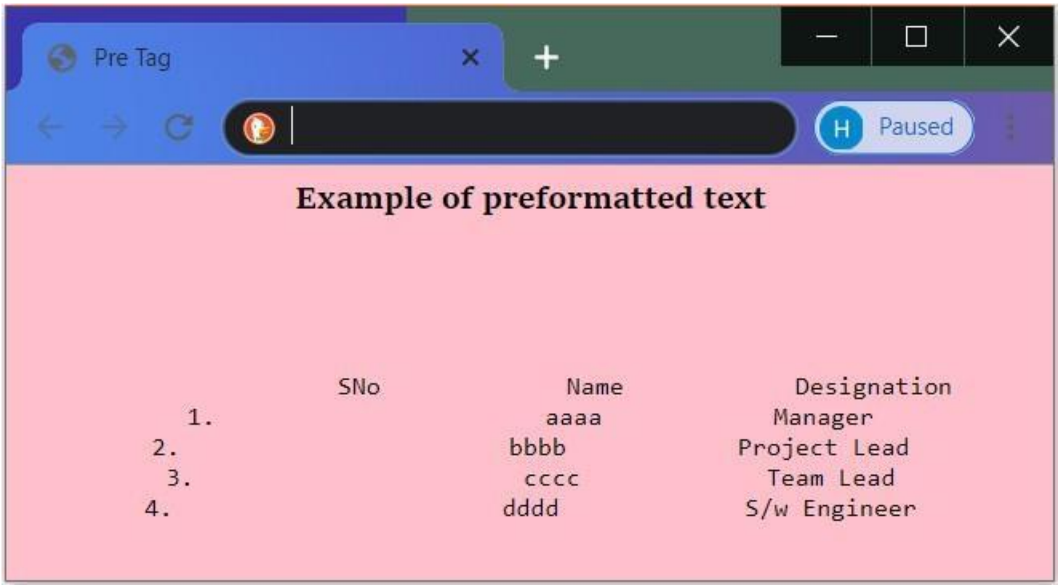
<pre> marks the text as preformatted text i.e, all the spaces and carriagereturns as rendered exactly as you type them.

Example:

```
<html>
        <head>
                <title> Pre Tag </title>
        </head>
        <body bgcolor="pink">
                <center>
                <h4> Example of preformatted text </h4> <br> <br>
                <pre>

                SNo         Name              Designation
                1.          aaaa               Manager
                2.          bbbb               Project Lead
                3.          cccc               Team Lead
                4.          dddd                S/w Engineer
                </pre>
        </center>
    </body>
</html>
```

Output:



Example of preformatted text

```
                SNo             Name            Designation
      1.                       aaaa            Manager
   2.                       bbbb            Project Lead
    3.                       cccc            Team Lead
  4.                       dddd            S/w Engineer
```

Lists:

Lists lets us display information in a compact, right format. There are three kindsof lists:

1. Unordered List
2. Ordered List
3. Definition List

Unordered List:

An Unordered list is a list of items that are marked with burden. The Unordered list is created by using <ul>tag are the list items in the list are createdby </ul> tag and the list items in the list are created by <li> tag.

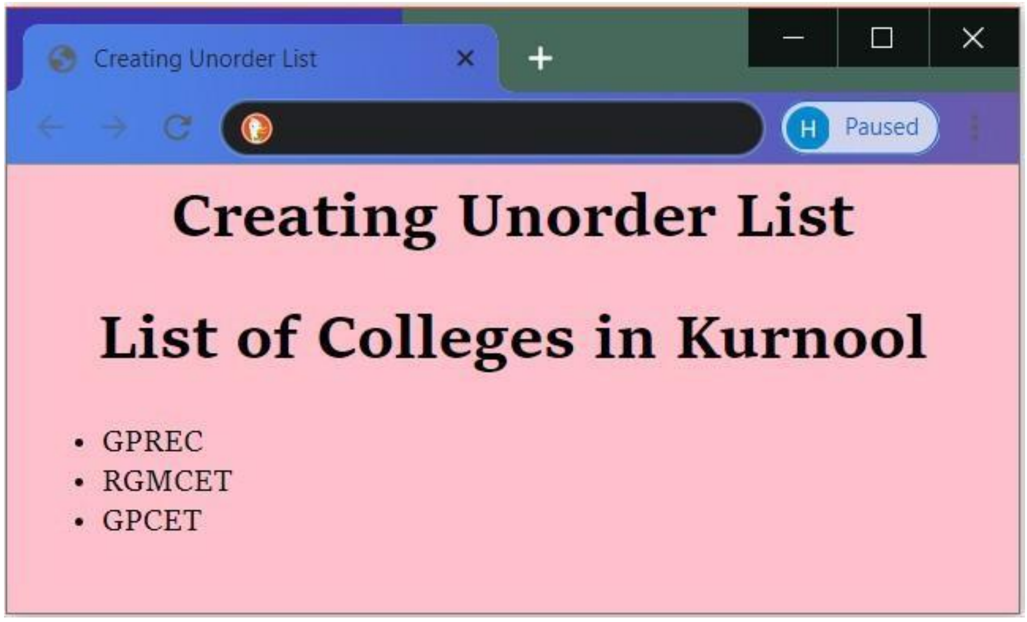
```
<ul>
   <li>List Item 1 </li>
  <li>List Item 2 </li>
</ul>
```

Example:

```
<html>
```

```
<head>
      <title> Creating Unorder List </title>
</head>
<body bgcolor=”pink”>
  <h1 align=”center”> Creating Unorder List</h1>
  <h1 align=”center”>List of Colleges in Kurnool</h1>
<ul>
   <li>GPREC</li>
   <li>RGMCET</li>
   <li>GPCET</li>
 </ul>
</body>
</html>
```

Output



Creating Unorder List

List of Colleges in Kurnool

- GPREC
- RGMCET
- GPCET

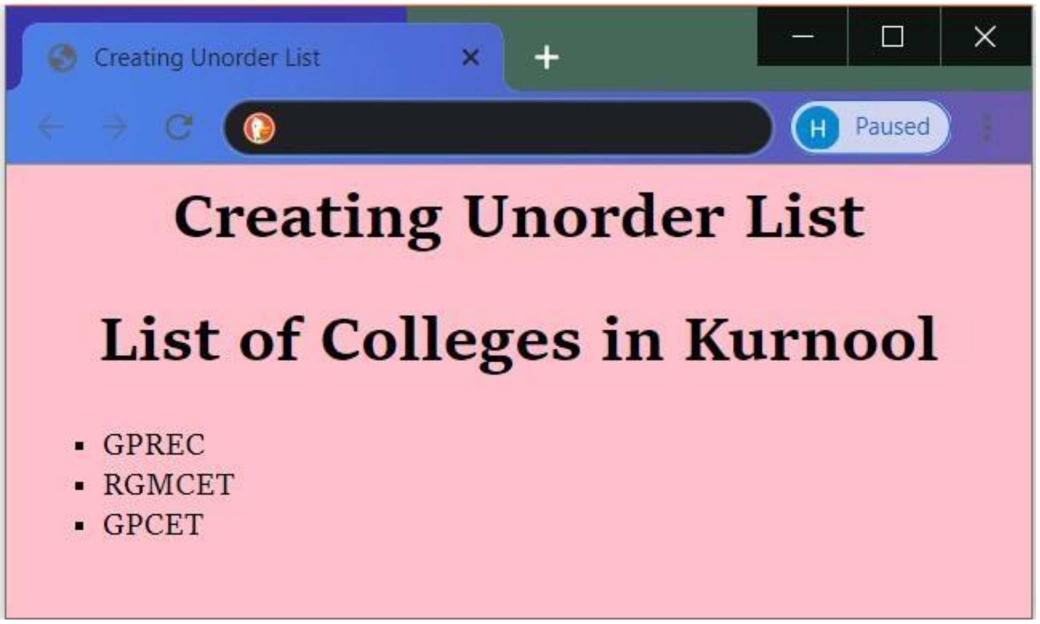
Creating Customized Unordered Lists:

We customized unordered lists by setting the “Type” attribute to three different values. DISC (default), SQUARE and CIRCLE which sets the type of bullet thatappears before the list item.

Example:

```
<html>
<head>
      <title> Creating Unorder List </title>
</head>
<body bgcolor=”pink”>
  <h1 align=”center”> Creating Unorder List</h1>
  <h1 align=”center”>List of Colleges in Kurnool</h1>
<ul type=”square”>
   <li>GPREC</li>
   <li>RGMCET</li>
   <li>GPCET</li>
 </ul>
</body>
</html>
```

Output



Creating Unorder List

List of Colleges in Kurnool

- GPREC
- RGMCET
- GPCET

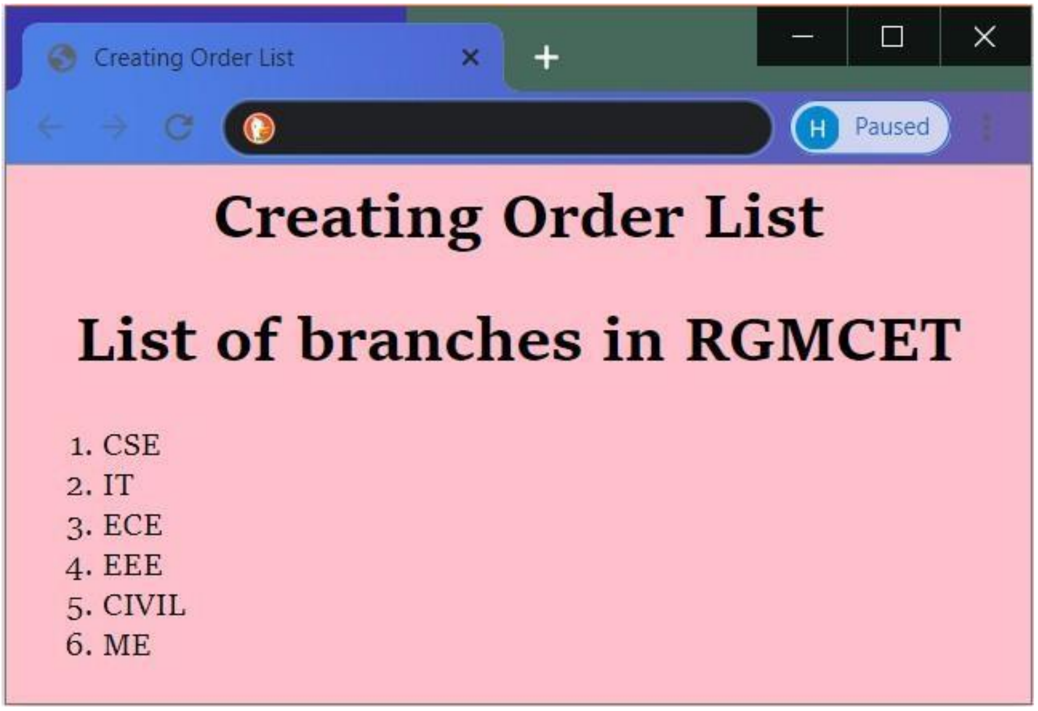
Ordered List:

While the unordered lists display simple bullet before each list item. Ordered lists use a number system / lettering scheme to indicate that the items are ordered in some ways, ordered lists are created by <ol> tag and the list items are created using <li> tag.

Example:

```
<html>
<head>
      <title> Creating Order List </title>
</head>
<body bgcolor=”pink”>
  <h1 align=”center”> Creating Order List</h1>
  <h1 align=”center”>List of branches in RGMCET</h1>
<ol>
     <li>CSE</li>
     <li>IT</li>
     <li>ECE</li>
     <li>EEE</li>
     <li>CIVIL</li>
     <li>ME</li>
 </ol>
</body>
</html>
```

Output



Creating Order List

List of branches in RGMCET

1. CSE
2. IT
3. ECE
4. EEE
5. CIVIL
6. ME

Creating Customized Ordered Lists:-

We can customize the numbering system used in ordered lists by using the TYPEattribute, which we can set to these values:

1. Default numbering system (1, 2, 3, ….)

A. Uppercase Letters (A, B, C, …..)

a. Lowercase Letters (a, b, c, …)

I. Large Roman Numerals (I, II , III, ….)

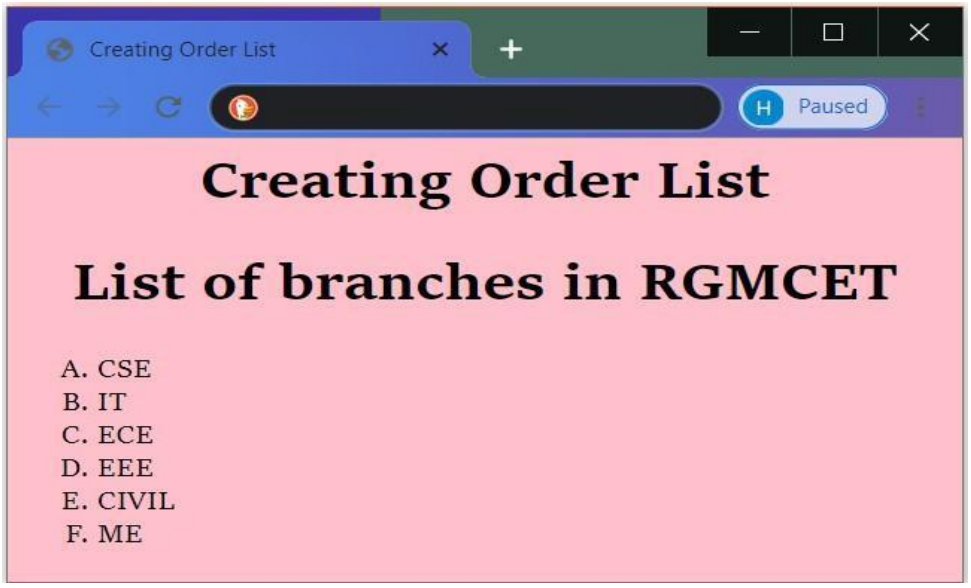
i. Small Roman Numerals (i, ii, iii, …..)

Example:

```
<html>
<head>
 <title> Creating Order List </title>
 </head>
 <body bgcolor="pink">
    <h1 align="center"> Creating Order List</h1>
        <h1 align="center">List of branches in RGMCET</h1>
```

```
            <ol type="A">
                  <li>CSE</li>
                  <li>IT</li>
                  <li>ECE</li>
                  <li>EEE</li>
                  <li>CIVIL</li>
                  <li>ME</li>
         </ol>
 </body>
</html>
```

Output



Creating Order List

List of branches in RGMCET

A. CSE
B. IT
C. ECE
D. EEE
E. CIVIL
F. ME

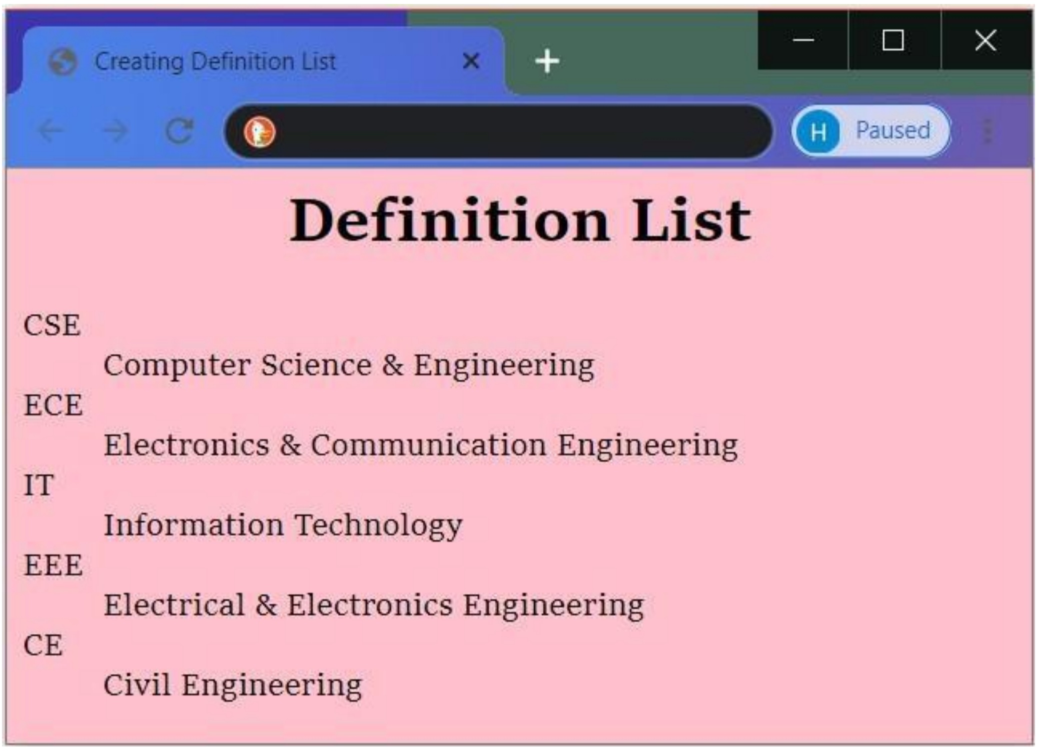
Definition List:-

These lists include both definition terms as well as their definition. To create the definition lists we use <dl> tag. For creating definition terms we use <dt> tag andfor data definitions we use <dd> tag.

Example:

```
<html>
    <head>
         <title>Creating Definition List</title>
    </head>
    <body bgcolo=”pink”>
        <h1 align=”center”>Definition List</h1>
        <dl>
                <dt>CSE<dd>Computer Science & Engineering
                <dt>ECE<dd>Electronics & Communication Engineering
                <dt>IT<dd>Information Technology
                <dt>EEE<dd>Electrical & Electronics Engineering
                <dt>CE<dd>Civil Engineering
       </dl>
    </body>
</html>
```

Output



```
                Definition List

CSE
        Computer Science & Engineering
ECE
        Electronics & Communication Engineering
IT
        Information Technology
EEE
        Electrical & Electronics Engineering
CE
        Civil Engineering
```

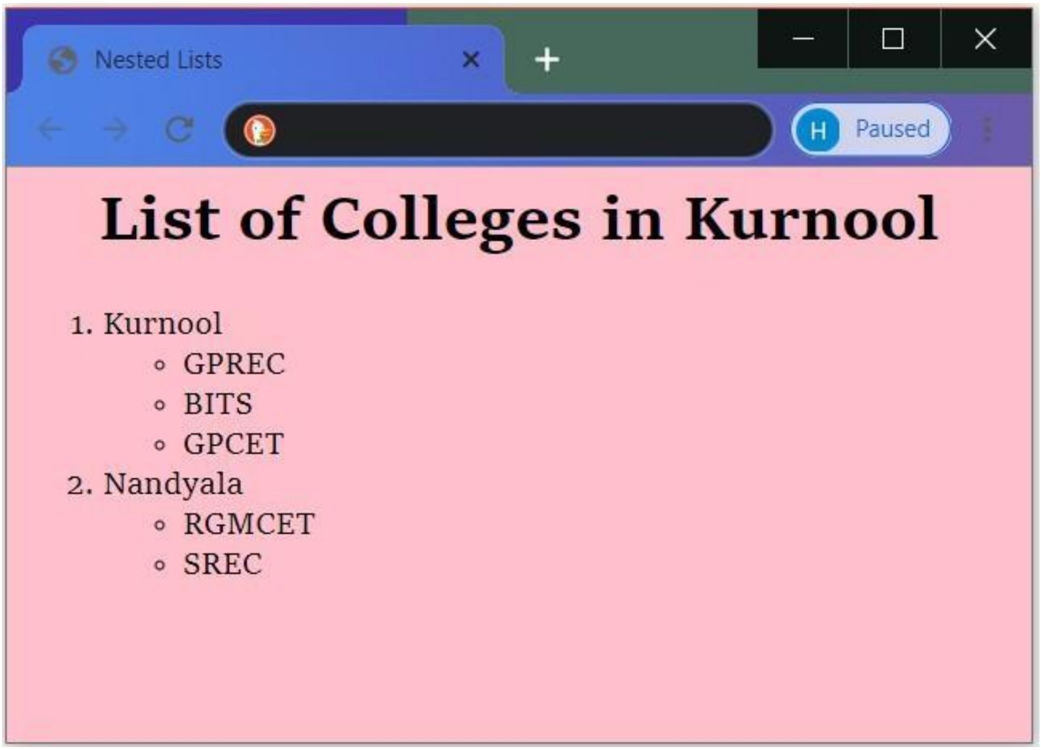
Nesting Lists:-

We have the capability of nesting lists inside other lists.

Example:

```
<html>
    <head>
         <title>Nested Lists</title>
    </head>
    <body bgcolor=”pink”>
        <h1 align=”center”>List of Colleges in Kurnool</h1>
 <ol>
     <li>Kurnool</li>
       <ul>
           <li>GPREC</li>
           <li>BITS</li>
             <li>GPCET</li>
      </ul>
   <li>Nandyala</li>
      <ul>
           <li>RGMCET</li>
           <li>SREC</li>
      </ul>
 </ol>
</body>
</html>
```

Output



List of Colleges in Kurnool

1. Kurnool
   - GPREC
   - BITS
   - GPCET
2. Nandyala
   - RGMCET
   - SREC

Creating Hyperlinks:

What makes the web so effective is the ability to define links from one page to another. In web terms, a “hyperlinks” is a reference on the web. Hyperlinks can point to any resources on the web. An anchor is a term used to define a hyperlinkdestination inside a document. Format of anchor tag is:

<a href=”address”> Line Text </a>

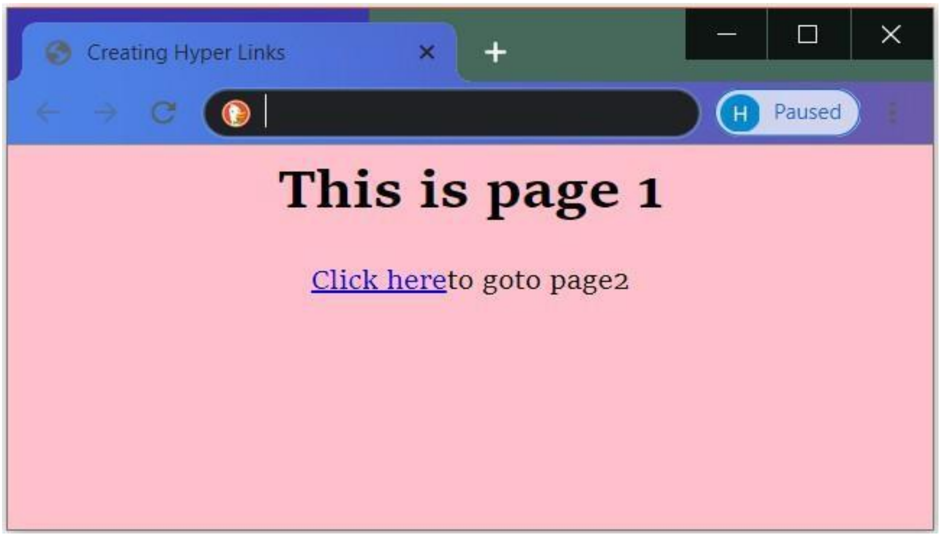
The <a> anchor tag has the following attributes.

1. href: It holds the target URL of the hyperlink.
2. Id: A unique alphanumeric identifier for the tag, which we can use to refer toit.
3. name: It specifies an anchor name, the name we want to use when referringto enclose items.
4. Target: This attribute defines where the linked document will be opened.

Example:

```
<html>
  <head>
    <title>Creating Hyper Links</title>
  </head>
  <body bgcolor="pink">
    <center><h1>This is page 1</h1>
    <a href=”page2.html”>Click here</a>to goto page2
    </center>
  </body>
</html>
```

Output



This is page 1

Click hereto goto page2

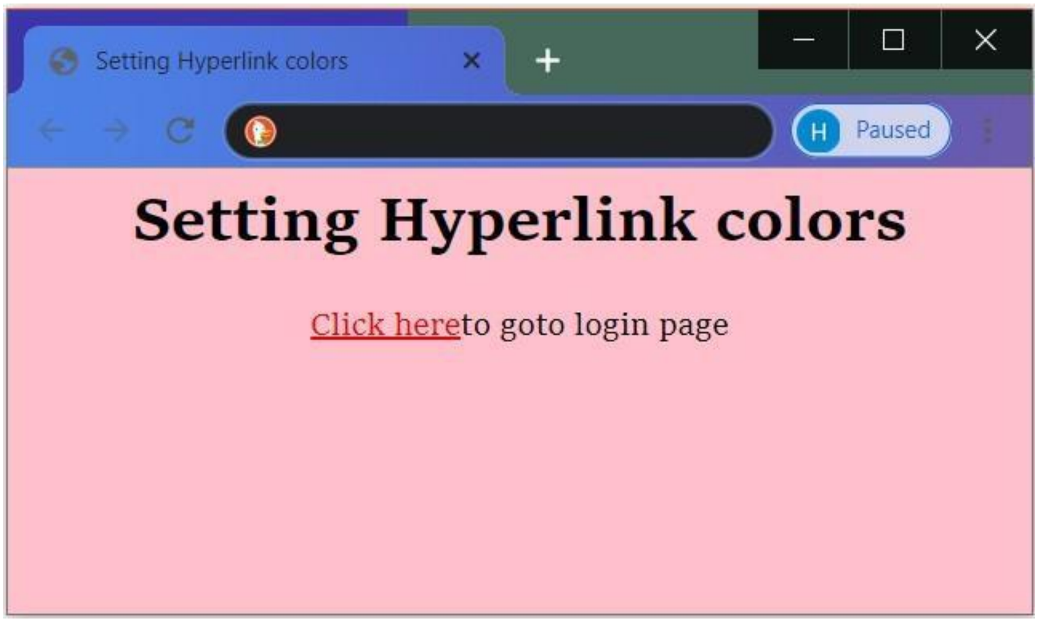
Setting hyperlink colors:

The default color of hyperlinks in a page is blue. Hyperlink that we have already visited are displayed in violet and when we click a hyperlink,it turns red when themouse button is down.We can set these colos in <body> tag attributes link,vlink(visited link),alink(active link).

Example

```
<html>
 <head>
   <title>Setting Hyperlink colors</title>
 </head>
 <body bgcolor="pink" link=”green” vlink=”blue” alink=”red”>
 <center><h1>Setting Hyperlink colors</h1>
 <a href=”login.html”>Click here</a>to goto login page
 </body>
</html>
```

Output



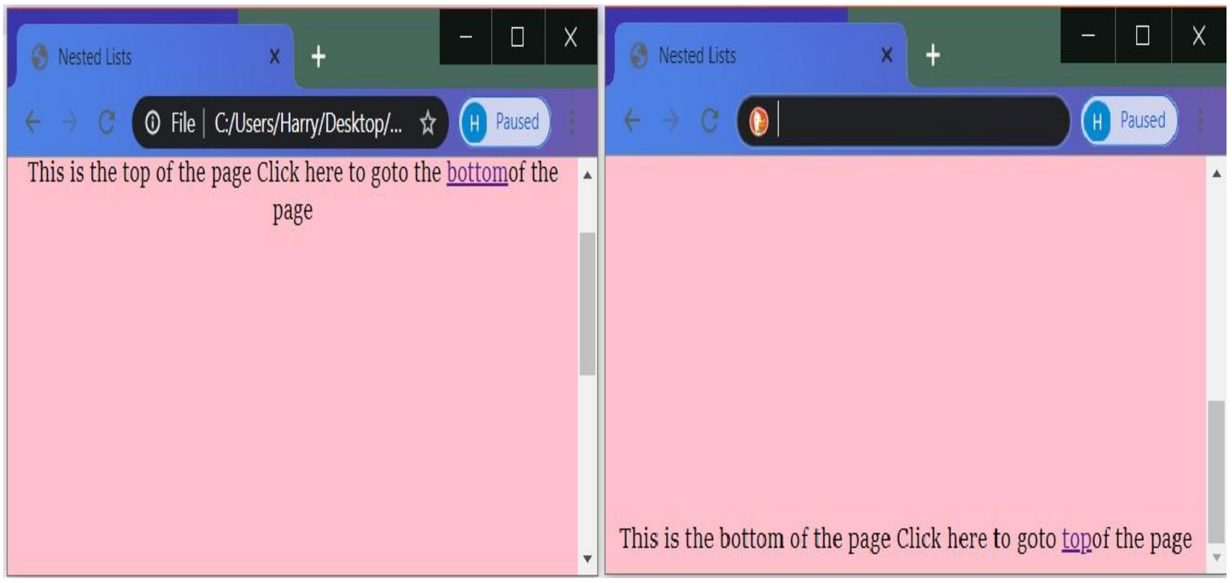
Setting Hyperlink colors

Click hereto goto login page

Providing navigation with in the page:

```
<html>
  <head>
   <title>Nested Lists</title>
 </head>
 <body bgcolor="pink">
 <center><h1>Linking to a section in a page</h1>
 <a name=”top”>This is the top of the page</a>
 Click here to goto the <a target=”#bottom”>bottom</a>of the page
<br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br>
<br><br><br><br><br>
<a name=”bottom”>This is the bottom of the page</a> Click here to
goto <a target=”#top”>top</a>of the page
</center>
</body>
</html>
```

Output



This is the top of the page Click here to goto the bottomof the page

This is the bottom of the page Click here to goto topof the page

Creating HML tables:

A HTML table arranges data/information in terms of rows and columns.Tables are defined in HTML using <table> tag.A table is divided into rows and each row each divided into data cells(columns).The rows of table are created using <tr> tag and data cells are created by <td> tag.

<tr> - Table row

<td> - Table data

Format

```
<table>
      <tr>
           <td>row1,col1</td>
           <td>row1,col2</td>
       </tr>
      <tr>
           <td>row2,col1</td>
           <td>row2,col2</td>
       </tr>
 </table>
```

- Heading in a table are defined with <th> tag

Format

```
<table>
      <tr>
           <th>heading 1</th>
           <th>heading 2</th>
       </tr>
      <tr>
           <td>data1</td>
           <td>data2</td>
       </tr>
 </table>
```

Attributes of <table> tag:

- align : specifies the horizontal alignment of the table in the browserwindow,set to ”left,center,right”.
- background : specifies the URL of a background image to be used asbackground for the table.
- bgcolor : sets the background color of the table cells.
- border : sets the border width.
- bordercolor : sets the external border color of the entire table.
- cellpadding : sets the spacing between cell walls and content.
- cellspacing : sets the spacing between table cells.
- height : sets the height of the whole table.
- width : sets the width of the table.

Attributes of <tr> tag:

- align : specifies the horizontal alignment content in the table cells set to “left,center,right”.
- bgcolor : sets the background color of the table cells.
- bordercolor : sets the external border color of the entire table.
- Valign : sets the vertical alignment of data,set to top,middle,bottom.

Alignment of <td> tag

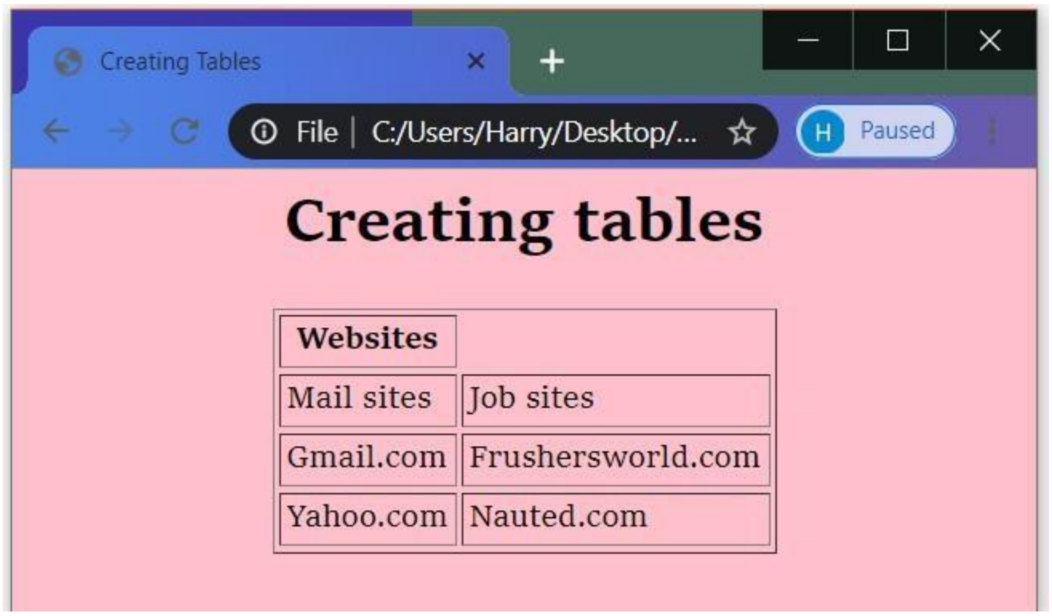
- align : specifies the horizontal alignment content in the table cells set to “left,center,right”.
- bgcolor : sets the background color of the table cells.
- bordercolor : sets the external border color of the entire table.
- colspan : indicates the how many cell columns of the table this cell shouldspan.
- rowspan : indicates the how many cell rows of the table this cell shouldspan.

Example

```
<html>
  <head>
```

```
    <title>Creating Tables</title>
  </head>
  <body bgcolor="pink">
  <center><h1>Creating tables</h1>
  <table border=”1” cellpadding=”3” cellspacing=”3”>
       <tr>
             <th colspa=”2”>Websites</th>
       </tr>
        <tr>
               <td>Mail sites</td>
               <td>Job sites</td>
       </tr>
       <tr>
               <td>Gmail.com</td>
               <td>Frushersworld.com</td>
       </tr>
    <tr>
               <td>Yahoo.com</td>
               <td>Nauted.com</td>
   </tr>
</center>
</table>  </body></html>
```

Output



Creating tables

Websites	
Mail sites	Job sites
Gmail.com	Frushersworld.com
Yahoo.com	Nauted.com

Advanced Table elements :

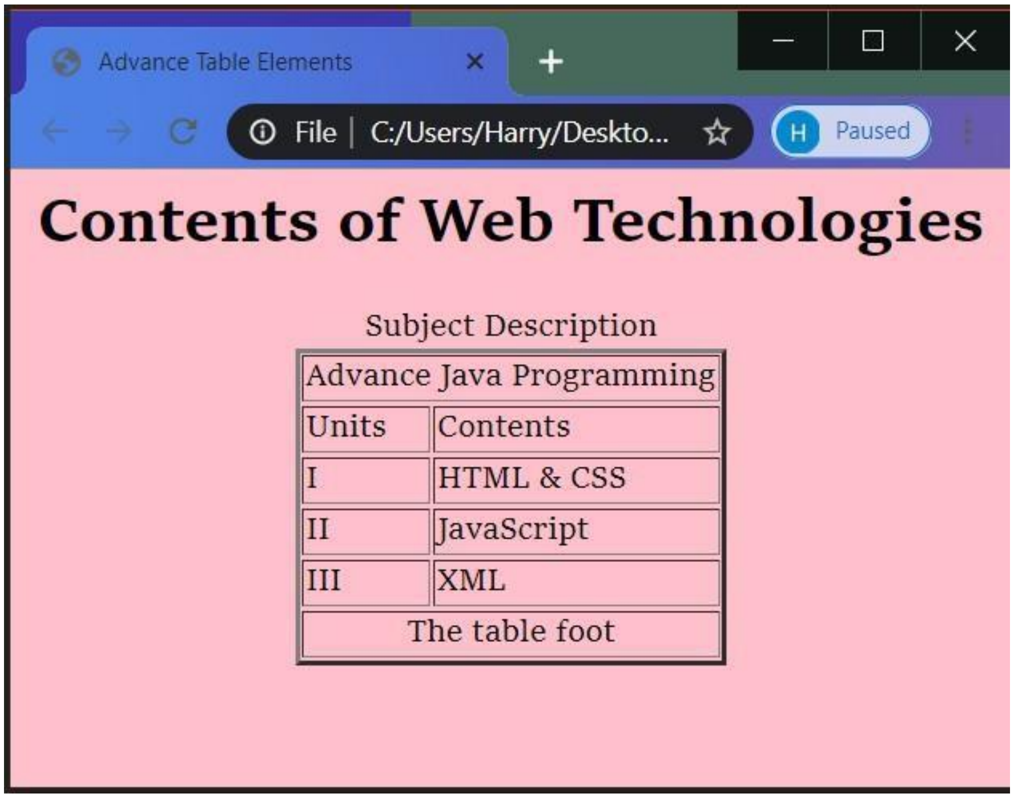
- <caption> : the element is an ptional element and it used to provide a stringwhich describes,the contern of the table ,it must follow the table element.
- <thead> : The rows in a table can bc groudated one are more time we cancreate a table by using this <thead>.
- <tbody> : creates a table body when grouping rows.
- <tfoot> : Creates a table foot when groupin rows

Example:

```
<html>
  <head>
    <title>Advance Table Elements</title>
  </head>
  <body bgcolor="pink">
  <h1 align="center">Contents of Web Technologies</h1>
  <center>
     <table border="2">
```

```
                <caption>Subject Description</caption>
        <thead>
                <tr> <td colspan="2">Advance Java Programming</td>
        </thead>
        <tbody>
                <tr> <td>Units</td>
                        <td>Contents</td>
                </tr>
                <tr> <td>I</td>
                        <td>HTML & CSS</td>
                </tr>
                <tr> <td>II</td>
                        <td>JavaScript</td>
                </tr>
                <tr> <td>III</td>
                        <td>XML</td>
                 </tr>
        </tbody>
        <tfoot align="center">
                <tr>
                        <td colspan="2">The table foot</td>
                </tr>
</tfoot>
</table>
</center>
        </body>
      </html>
```

Output



Contents of Web Technologies

Subject Description

Advance Java Programming	
Units	Contents
I	HTML & CSS
II	JavaScript
III	XML
The table foot	

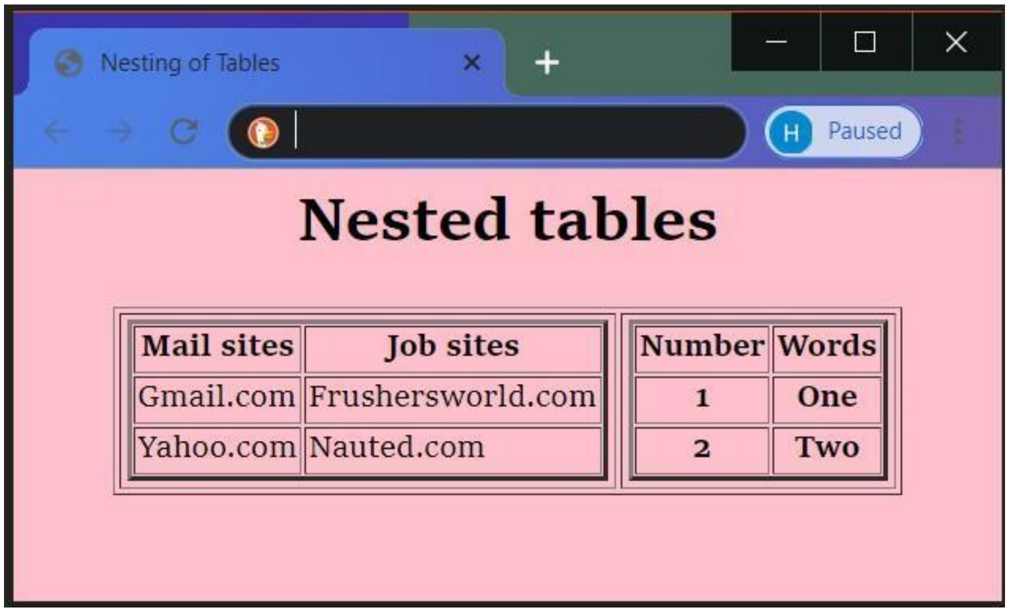
Nesting of Tables:

```
<html>
  <head>
    <title>Nesting of Tables</title>
  </head>
  <body bgcolor="pink">
  <center><h1>Nested tables</h1>
  <table border="1" cellpadding="3" cellspacing="3">
    <tr>
     <td>
       <table border="2">
        <tr>
               <th>Mail sites</th>
               <th>Job sites</th>
       </tr>
```

```
          <tr>
                  <td>Gmail.com</td>
                  <td>Frushersworld.com</td>
          </tr>
          <tr>
                  <td>Yahoo.com</td>
                  <td>Nauted.com</td>
          </tr>
         </table>
        </td>
       <td>
          <table border="2">
             <tr>
                <th>Number</th>
                <th>Words</th>
             </tr>
             <tr>
                <th>1</th>
                <th>One</th>
             </tr>
             <tr>
                <th>2</th>
                <th>Two</th>
             </tr>
           </table>
         </td>
       </tr>
   </table>
   </center>
    </body>
```

</html>

Output



Nested tables

Mail sites	Job sites
Gmail.com	Frushersworld.com
Yahoo.com	Nauted.com

Number	Words
1	One
2	Two

Images in HMTL:

In HTML we have the capability of displaying images in a webpage.This imagesmust be in a format that the web browser can handle , such as Graphics Interchange Format(GIF) , Joint Photograph Expert Group(JPEG) , and for somebrowser Portable Network Graphics(PNG) formats.

Displaying images in webpage is done by using <img> tag

Format

<img src=”URL of image source”>

Attributes of <img> tag:

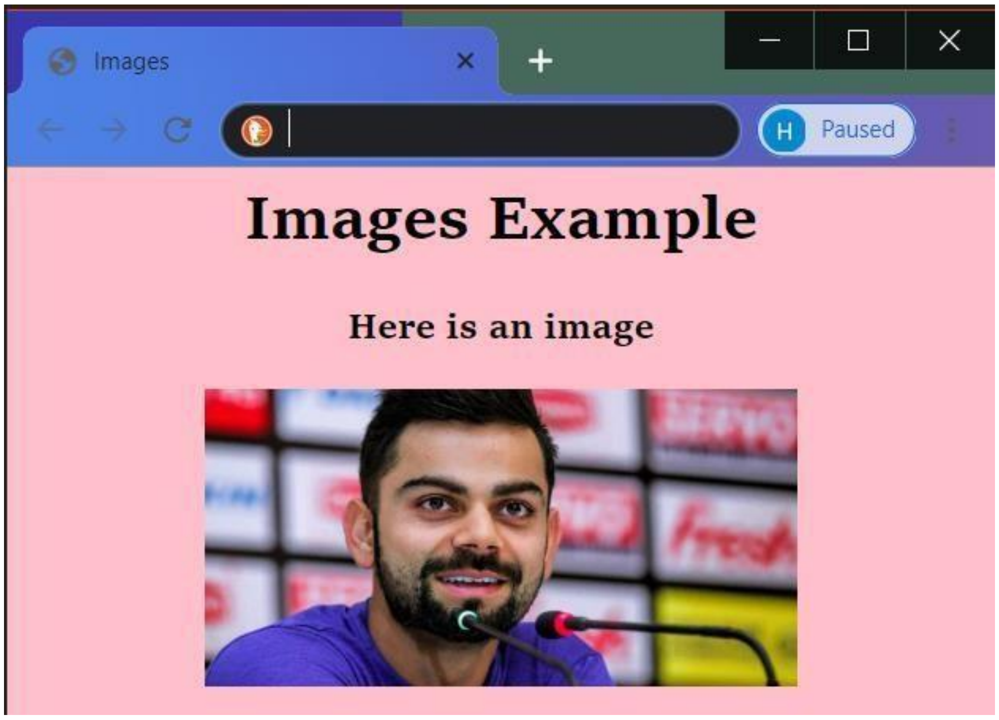
- alt : this attribute is used to specify text to be displayed in place of image forbrowser that cannot handle graphics.
- src : specifies the URL of the image to display.
- border : sets the border for the image.
- height : indicates the height of the image.
- width : indicates the width of the image.

- hspace : sets the horizontal space around the image.
- vspace : sets the vertical space around the image.

Example

```
<html>
<head>
  <title>Images</title>
</head>
<body bgcolor="pink">
 <center>
   <h1>Images Example</h1>
   <h3>Here is an image</h3>
   <img src="one.jpg" alt="here is an image" width="300" height="150">
  </center>
</body>
</html>
```

Output



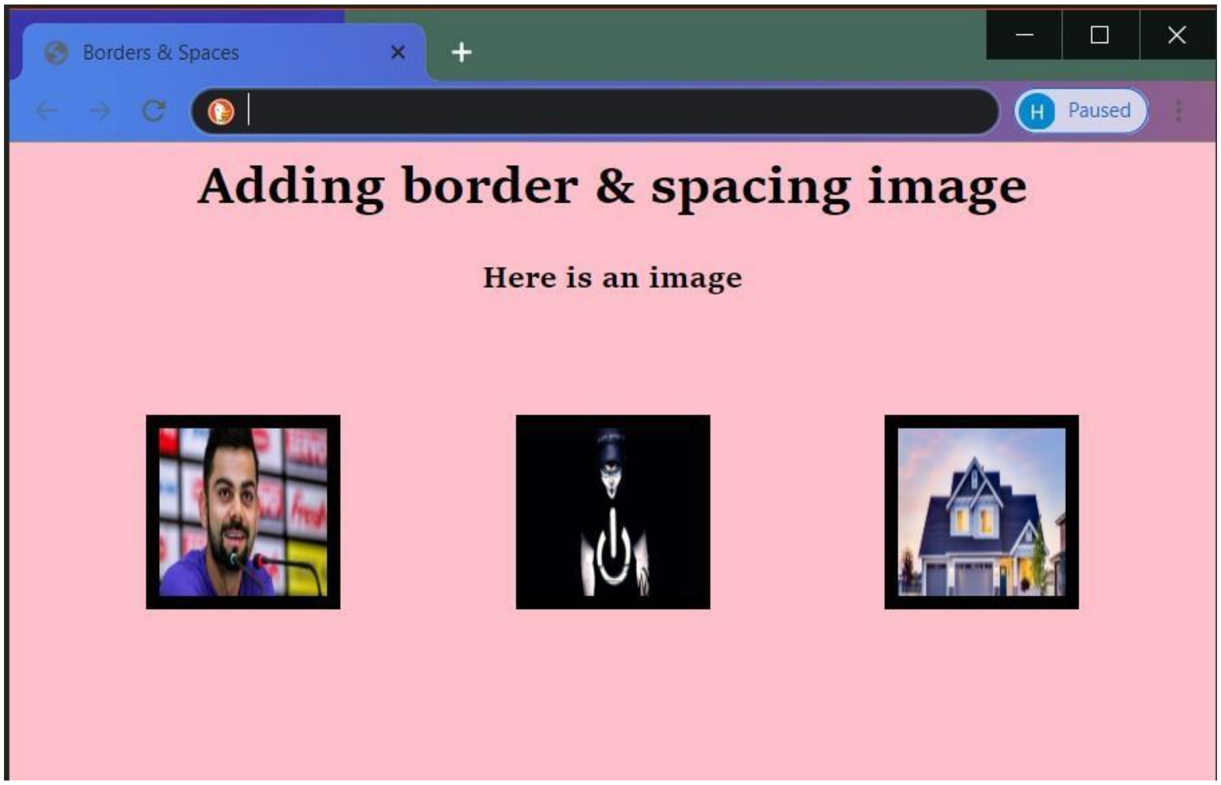
Images Example

Here is an image

Adding borders and spaces around image:

```
<html>
<head>
  <title>Borders & Spaces</title>
</head>
<body bgcolor="pink">
 <center>
   <h1>Adding border & spacing image</h1>
   <h3>Here is an image</h3>
   <img src="one.jpg" alt="here is an image" width="100" height="100"
hspace="50" vspace="50" border="8">
   <img src="two.png" alt="here is an image" width="100" height="100"
hspace="50" vspace="50" border="8">
   <img src="three.png" alt="here is an image" width="100" height="100"
hspace="50" vspace="50" border="8">
  </center>
</body>
</html>
```

Output:



Adding border & spacing image

Here is an image

Creating HTML Forms:

Form is a collection of various HTML control files , buttons ,checkboxes , radio buttons ,text fields et., and they use to send the data to the server. There are severalform elements.

- Button : <input type=”button”> :- are the standard clickable buttons.
- Checkbox : <input type=”checkbox”> :- displayed usually as a small boxwith a check mark in it. The use can toggle the checkbox on or off by checking the checkbox
- Customizable Buttons : <button> :- display images one other HMTL insideitself.
- File uploading controls : <input type=”file”> :- allow the user to upload filesto the server.
- Hidden controls : store data that is not visible to users unless they view theweb page source code.
- Image controls : <input type=”image”> :- are like submit buttons except thatthey are images the user can click.
- Password controls : <input type=”password”> :- are like text fields , buteach typed character displaying by an asterisk or instead any character.

- Radio buttons : <input type="radio"> :- displaying usually as a circle whichwhen selected displayed a dot in the middle. These controls are much like checkboxes except that they work int mutually exclusive at a time.
- Reset button : <input type="reset"> : - allow the user to clear all the data they has entered.When the user click reset button all controls in the form areremoved to that original state displaying the data they had when they first appeared.
- Selection : Works much like drop down list boxes also called select controlsFormat is:

```
<select>
        <option>Item1</option>
        <option>Item1</option>
        <option>Item1</option>
</select>
```

- Submit button : when we click the button all the data in the form will be sentto web server for processing.
- Text area : are two dimensional text fields allowing user to enter more thanone line of text.

  Format is: <textarea>
- Text fields : allow the user to enter one line of text also called a textboxFormat is :

  <input type="text">

In order to create form we use <form> tagFormat
is :

```
<form>
 |
 |
</form>
```

Attributes of <form> tag:

- name : gives the name of the form so that we can return it in code . Set to analphanumeric string.
- target : indicates a named frame for browser to display the form results.

- method : indicates a method or protocol for sending data to the target actionURL.
- action : gives the URL that that will handle the form data.

Example

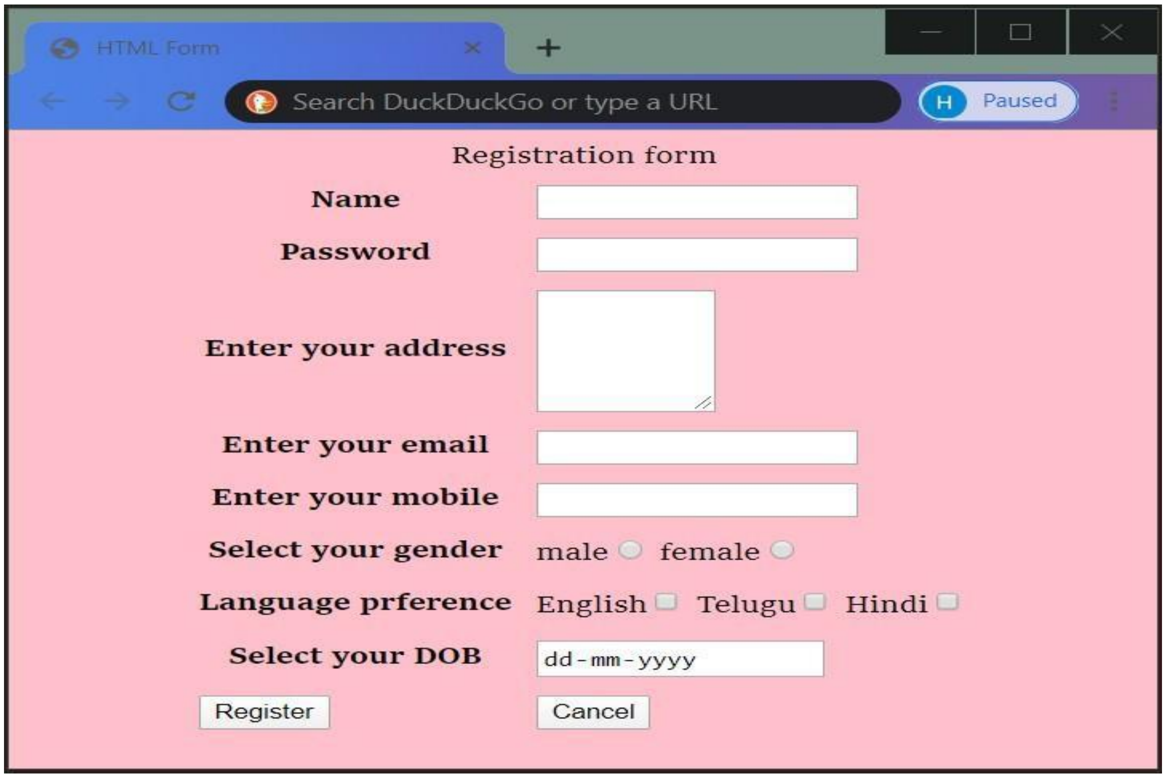
Registration.html

```
<html>
<head>
<title>HTML Form</title>
</head>
<body bgcolor="pink">
<center>
<form name="form1">
<table border="0" cellpadding="4" cellspacing="4">
<caption>Registration form</caption>
<tr>
<th>Name</th>
<td><input type="text" name="name" /></td>
</tr>
<tr>
<th>Password</th>
<td><input type="password"/></td>
</tr>
<tr>
<th>Enter your address</th>
<td><textarea rows="5" cols="10"></textarea></td>
</tr>
<tr>
<th>Enter your email</th>
<td><input type="email"/></td>
</tr>
<tr>
```

```
<th>Enter your mobile</th>
<td><input type="number"/></td>
</tr>
<tr>
<th>Select your gender</th>
<td>
male<input      type="radio"      name="g"      value="m"/>
female<input type="radio" name="g" value="f"/>
</td>
</tr>
<tr>
<th>Language prference</th>
<td>
English<input    type="checkbox"    value="    "/>
Telugu<input    type="checkbox"    value="    "/>
Hindi<input type="checkbox"  value=" "/>
</td>
</tr>
<tr>
<th>Select your DOB</th>
<td><input type="date"/></td>
</tr>
<tr>
<td ><input type="submit" value="Register"/></td>
<td><input type="reset" value="Cancel"/></td>
</td>
</tr>
</table>
</form>
</body>
```

</html>

Output



Registration form

Name	
Password	
Enter your address	
Enter your email	
Enter your mobile	
Select your gender	male ○ female ○
Language prference	English ☐ Telugu ☐ Hindi ☐
Select your DOB	dd-mm-yyyy
Register	Cancel

Working with Frames:

HMTL frames allow user to present documents in multiple views which may beindependent windows or sub windows. To divide a webpage into multiple parts and load different pages in a single web page we use the concept of frames. To dothis we use "<frameset>" tag which indicates the browser that the webpage window has a frame. We can divide it into rows and columns by using attributes such as 'rows' & 'cols'. In order to provide definition or each frame we use "<frame>" tag.

Format is :

```
<frameset rows="30%,70%">
    <frame src="source page URL" name="frame name">
    <frame src="source page URL" name="frame name">
</frameset>
```

- <frameset> element actually takes place of <body> tag.

Attributes of <frameset> tag:

- border : used in the outermost <frameset> tag to set the border thickness forframes.
- bordercolor : set the color of the borders for all frames in the frameset.
- frameborder : set whether or not border for all frames in the frameset.Can be set to ‘yes’ or ‘no’ or ‘1’ or ‘0’.
- framespacing : set the pixel spacing between frames. set to the positiveintegers.
- cols : set the number of columns in the frameset. Separate the values assigned to this attribute with comma(,) each value represents width of acolumn. Can be set to pixel values,percentages.
- Rows : set the number of rows in the frameset. Separate the values assigned to this attribute with comma(,) each value represents width of a column. Can be set to pixel values,percentages.

Attributes of <frame> tag:

- bordercolor : set the color used for the frmae border. This setting overridesthe color specified in the surrounding <frameset> element.
- frameborder : sets whether or not border surround the frame. Can be set to‘yes’ or ‘no’ or ‘1’ or ‘0’.
- name : sets the name of the frame we can use named frames as target for <a>tag.
- scrolling : determines scrolling possible values are : auto,yes or no.
- src : specifies the URL of the frame document. If we don’t specify a URLthe frame will appear blank.

Creating vertical frames:

In order to display vertical frame we have use of ‘cols’ attribute.

Example

```
<html>
<head>
<title>Vertical Frames</title>
</head>
```

```
<frameset cols="30%,70%">
        <frame src=frame1.html>
        <frame src=frame2.html>
</frameset>
</html>
```

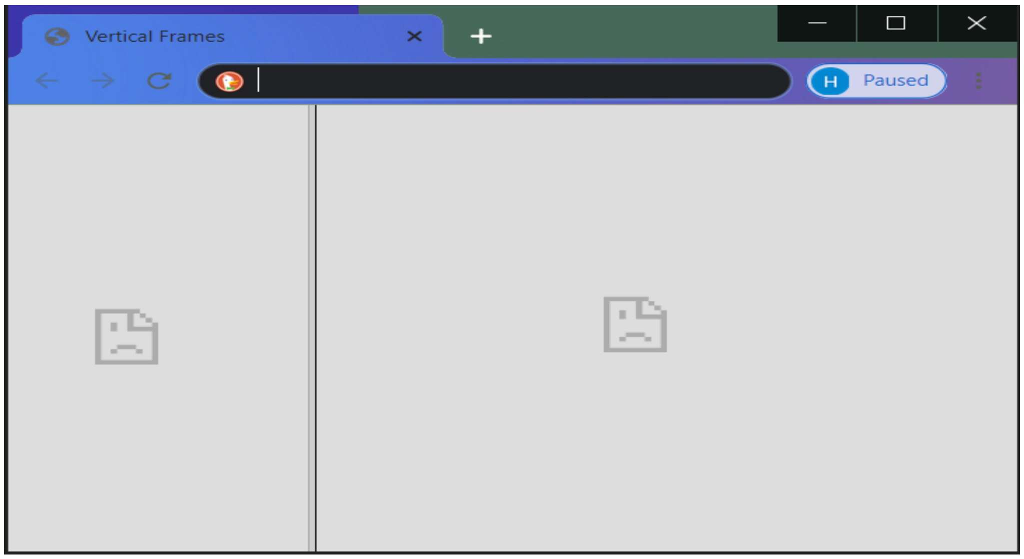
Frame1.html

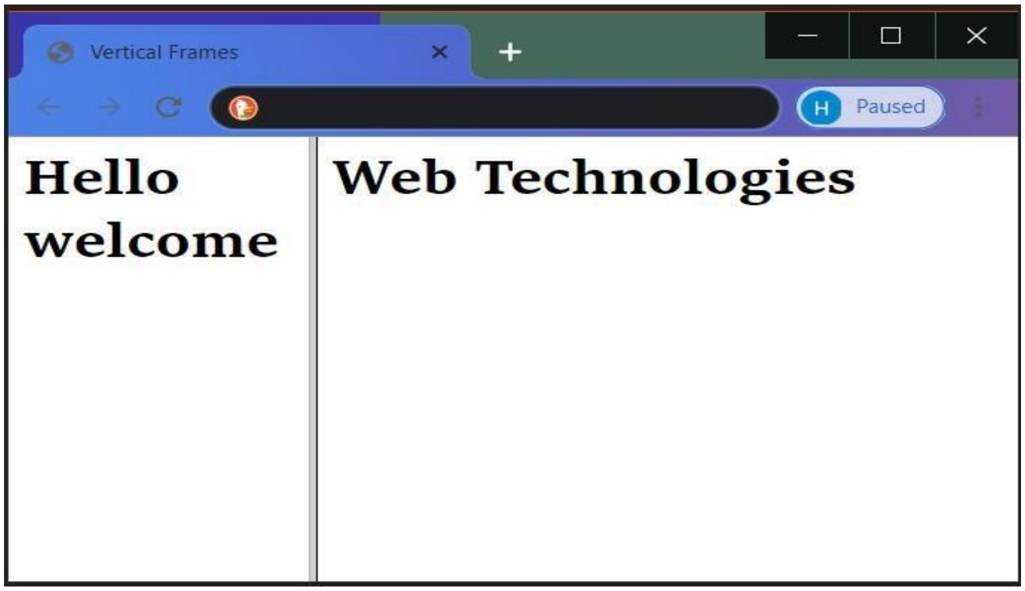
```
<html>
<head>
        <title>page1</title>
</head>
<body>
 <h1>Web Technologies</h1>
</body>
</html>
```

Frame2.html

```
<html>
<head>
        <title>page2</title>
</head>
<body>
 <h1>Web Technologies</h1>
</body>
</html>
```

Output





Hello welcome

Web Technologies

Creating horizontal frames:

In order to display horizontal frame we have use of ‘cols’ attribute.

Example

```
<html>
<head>
        <title>Horizontal Frames</title>
</head>
<frameset cols="50%,50%">
        <frame src=page1.html>
        <frame src=page2.html>
</frameset>
</html>
```

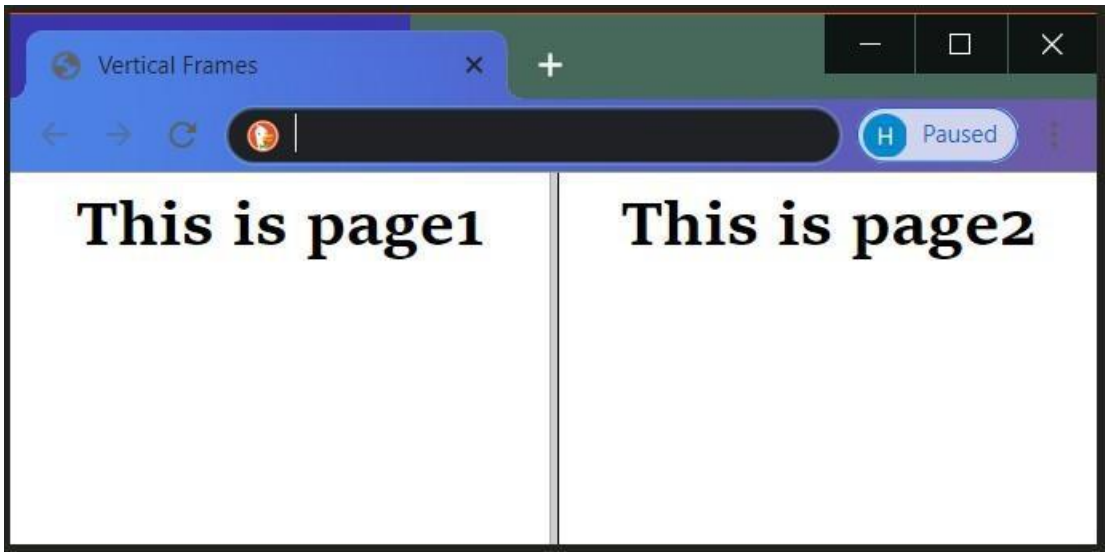
page1.html

```
<html>
<head>
        <title>page1</title>
</head>
<body>
 <h1 align=”center”>This is page1</h1>
</body>
</html>
```

page2.html

```
<html>
<head>
        <title>page2</title>
</head>
<body>
 <h1 align=”center”>This is page2</h1>
</body>
</html>
```

Output



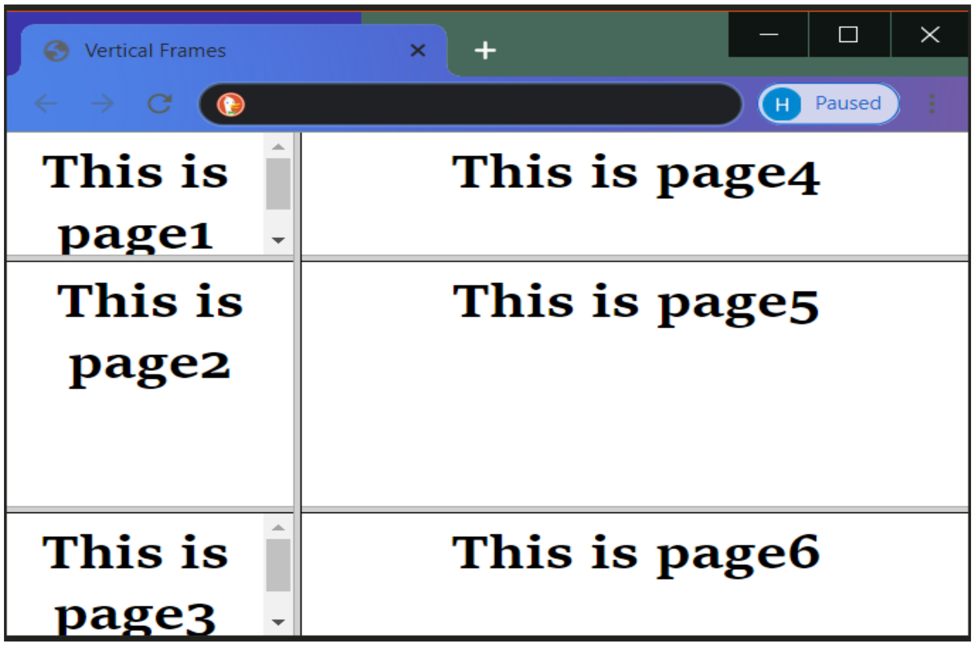
This is page1

This is page2

Creating horizontal & vertical frames :

```
<html>
<head>
        <title>Horizontal Frames</title>
</head>
<frameset cols="30%,70%">
    <frameset cols="25%,50%,25%">
        <frame src=page1.html>
        <frame src=page2.html>
        <frame src=page3.html>
    </frameset>
    <frameset cols="25%,50%,25%">
        <frame src=frame1.html>
        <frame src=frame2.html>
        <frame src=frame3.html>
    </frameset>
</frameset>  </html>
```

Output



This is page1

This is page4

This is page2

This is page5

This is page3

This is page6

<noframes> tag:

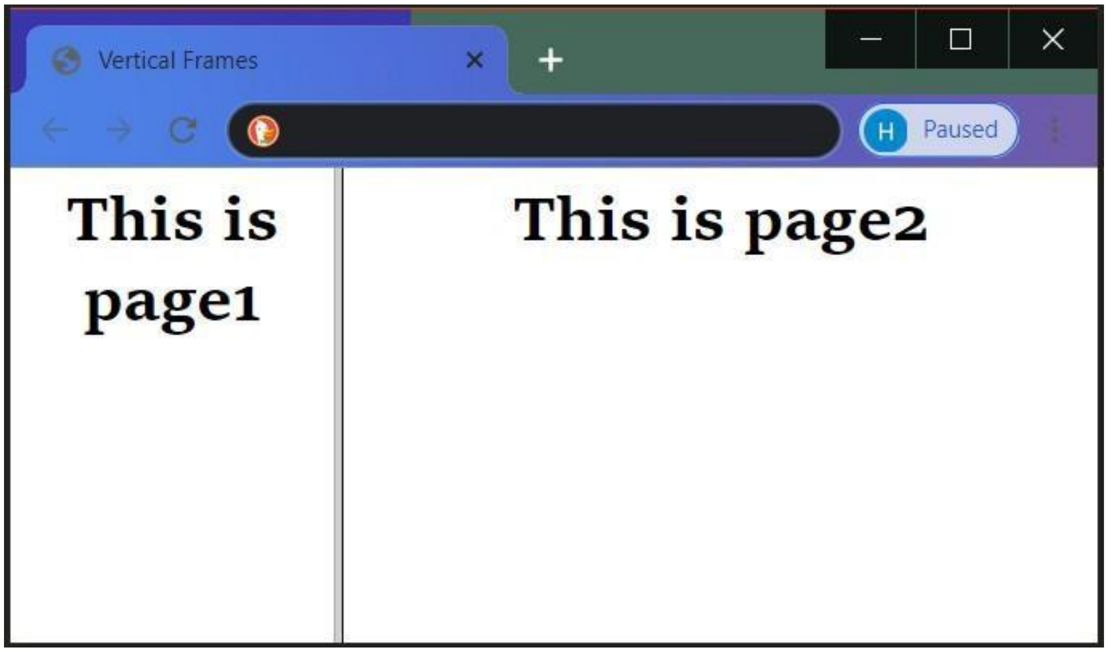
When the browser does not support frameset use <noframes> element to indicateto users that the browser doesnot support frames.The <noframes> element is ignored that handle frames.

Example

```
<html>
<head>
        <title>Vertical Frames</title>
</head>
<frameset cols="30%,70%">
        <noframes>Your browser does not support frames...</noframes>
        <frame src=on.html>
        <frame src=two.html>
</frameset>
```

</html>

Output



This is page1

This is page2

Named frames:

One important aspect of working with frames is using named frames. When wegive frmae a name , we can use as a target to load new page into the frame.

Example

```
<html>
<head>
        <title>Vertical Frames</title>
</head>
<frameset cols="40%,60%">
        <frame src=menu.html>
        <frame src=default.html name="display">
</frameset>
</html>
```

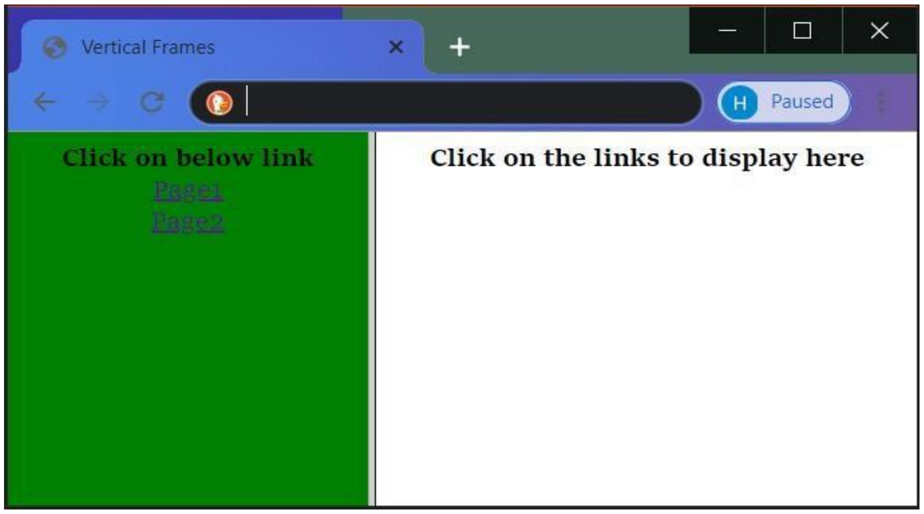
menu.html

```
<html>
<head>
            <title>Menu</title>
</head>
<body bgcolor="green">
  <center><b>Click on below link</b>
            <br><a href="page1.html" target="display">Page1
            <br><a href="page2.html" target="display">Page2
  </center>
</body>
</html>
```

Defual.html

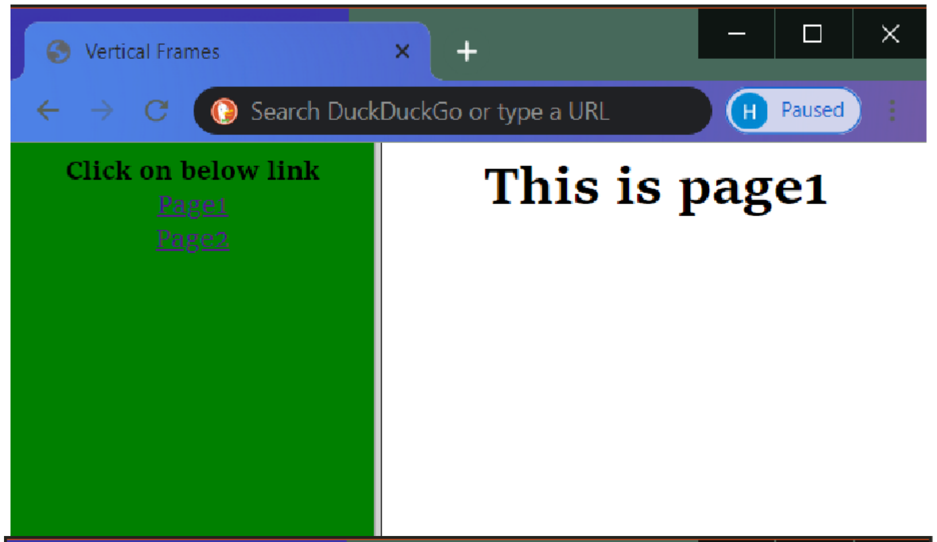
```
<html>
<head>
            <title>Vertical Frames</title>
</head>
<frameset cols="40%,60%">
            <frame src=menu.html>
            <frame src=default.html name="display">
</frameset>
</html>
```

Output



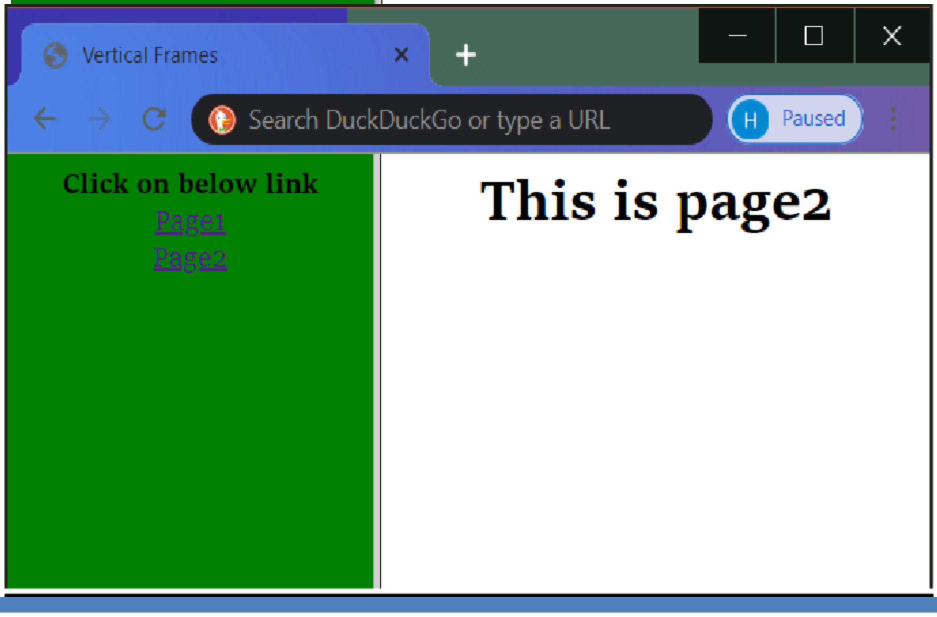
Click on below link
Page1
Page2

Click on the links to display here



Click on below link
Page1
Page2

This is page1



Click on below link
Page1
Page2

This is page2

Cascading Style Sheets:

Style sheets represent the World Wide Web consortium"s effort to improve on the tag andattribute based style of formatting. Style sheets provide a way of customizing whole pages all at once and in much richer detail than the simple use of tags and attributes. The format of style sheet will be:

```
<style type=”text/css”>
        selector{property:value;property:value;}
        selector{property:value;property:value;}
</style>
```

Every line in <style> tag is called as a „**Rule**" and a style rule has two parts:

a. Selector.
b. Set of declarations.

A selector is used to create a link between the rule and the HTML tag. The declaration has two parts again:

a. Property.
b. Value.

A property specifies additional information and value specifies property value. For example:

```
<style type=”text/css”>
body {background-color: #d0e4fe;}
 h1 {
color: orange;
 text-align: center;
}
p {
font-family: "Times New Roman";
font-size: 20px;
}
</style>
```

If we add above code in the <head> element of web page , entire web page will be displayed in various styles given in style element.

Style sheets are implemented with cascading style sheets specification. Conventionally styles are cascaded i.e., we don‟t have to use just a single set of styles inside a document, but we can import as many styles as we like. There are three mechanisms by which we can apply styles to our HTML documents:

1. Inline Style sheets.
2. Embedded Style sheets.
3. External Style sheets.

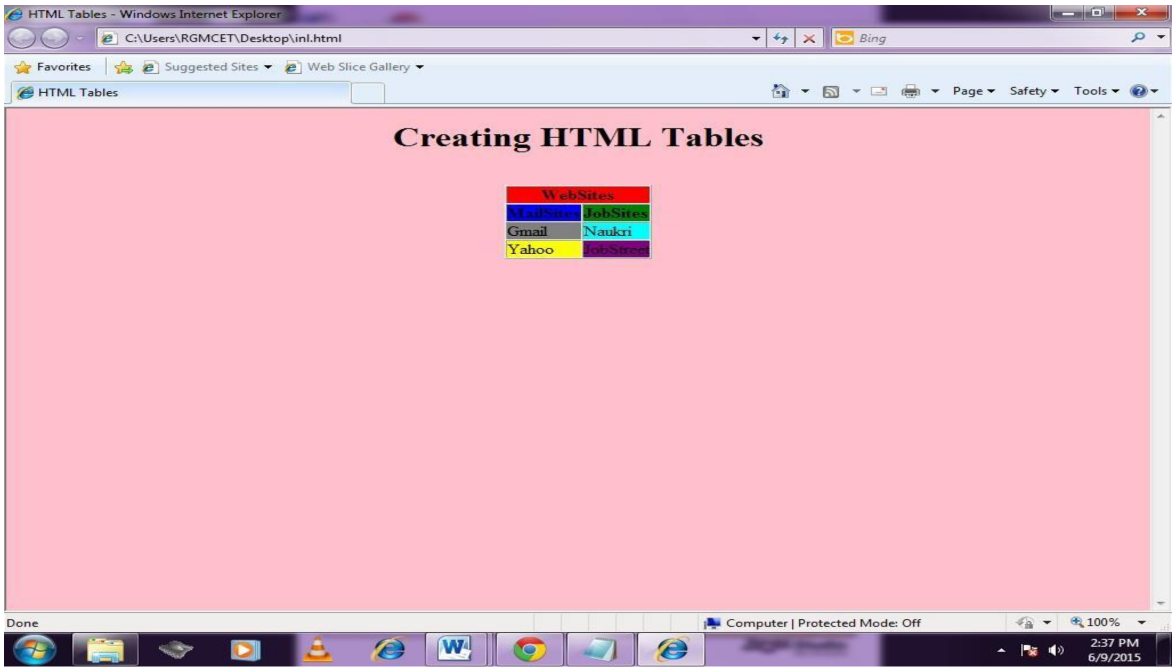
Inline Style Sheets:

Inline style sheets mix content with presentation. To use inline styles we use style attribute in the relevant tag.

Example:

```
<html>
        <head>
                <title>HTML Tables</table>
        </head>
        <body bgcolor="pink">
                <center>
                        <h1>Creating HTML Tables</h1><br>
                        <table border=”2” cellpadding=”4” cellspacing=”4”>
                                <tr>
                                    <th colspan=”2” style=”background-color:red”> WebSites</th>
                                </tr>
                                <tr>
                                        <th style=”background-color:blue”>MailSites</th>
                                        <th style=”background-color:green”>JobSites</th>
                                </tr>
                                <tr>
                                        <td style=”background-color:grey”>Gmail</td>
                                        <td style=”background-color:aqua”>Naukri</td>
                                        </tr>
                                        <tr>
                                                <td style="background-color:yellow">Yahoo</td>
                                                <td style="background-color:purple">JobStreet</td>
                                        </tr>
                                </table>
                        </center>
                </body>
        </html>
```

Output:



Creating HTML Tables

WebSites	
MailSites	JobSites
Gmail	Naukri
Yahoo	JobStreet

HTML Tables - Windows Internet Explorer
C:\Users\RGMCET\Desktop\inl.html
Bing
Favorites
Suggested Sites
Web Slice Gallery
HTML Tables
Page
Safety
Tools
Done
Computer | Protected Mode: Off
100%
2:37 PM
6/9/2015

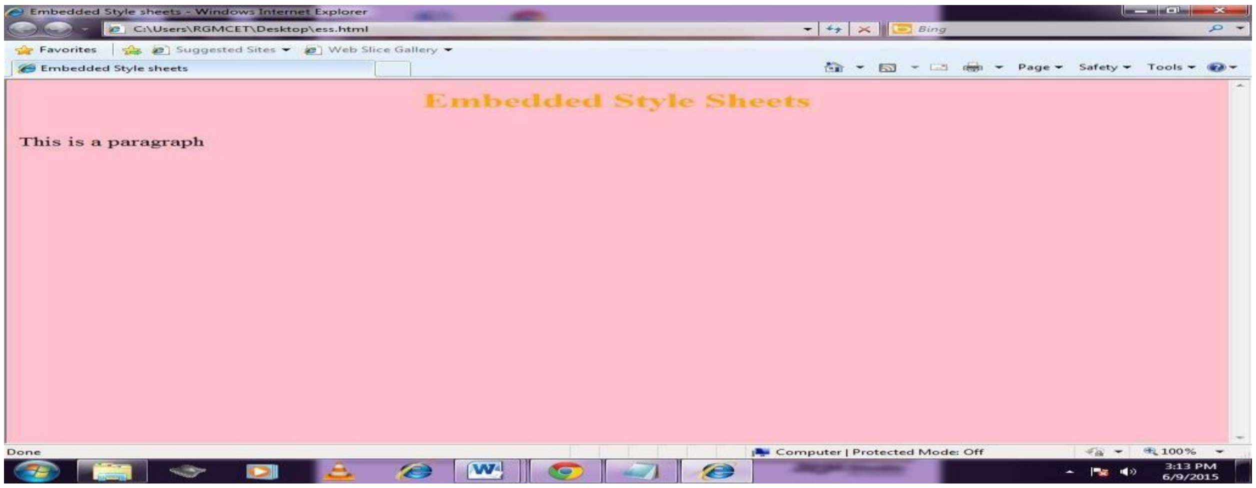
Embedded Style sheets:

An embedded style sheet is used when a single document has a unique style. We define internal styles in the head section of a HTML page by using „**<style>**" tag. The styles defined using embedded style sheets are applied throughout the page and we put the styles into oneplace.

Example:

```
<html>
        <head>
                <title>Embedded Style sheets</title>
                <style type=”text/css”>
                                body{background-color:
                                pink;}
                                h1 {
                                  color:orang
                                  e;
                                  text-align:
                                  center;
                                   }
                                p {
                                  font-family:    "Times    New
                                  Roman";
                                  font-size: 20px;
                                  }
                </style>
        </head>
        <body>
                <h1>Embedded Style Sheets</h1><br>
                <p>This is a paragraph
        </body>
</html>
```

Output:


Embedded Style sheets - Windows Internet Explorer
C:\Users\RGMCET\Desktop\ess.html
Embedded Style Sheets
This is a paragraph
Done
Computer | Protected Mode: Off
100%
3:13 PM
6/9/2015

External Style Sheets:

External style sheets are just that the style sheets are stored separately from our web page.These are useful especially if we are setting the styles for an entire website. When we change the styles in external style sheet we change the styles of all pages. We use „**<link>**" element to access the style sheet file defined into our web page. The format of <link> element is:

<link rel="stylesheet" type="text/css" href="extstylesheet.css">

Example:

extern.css:

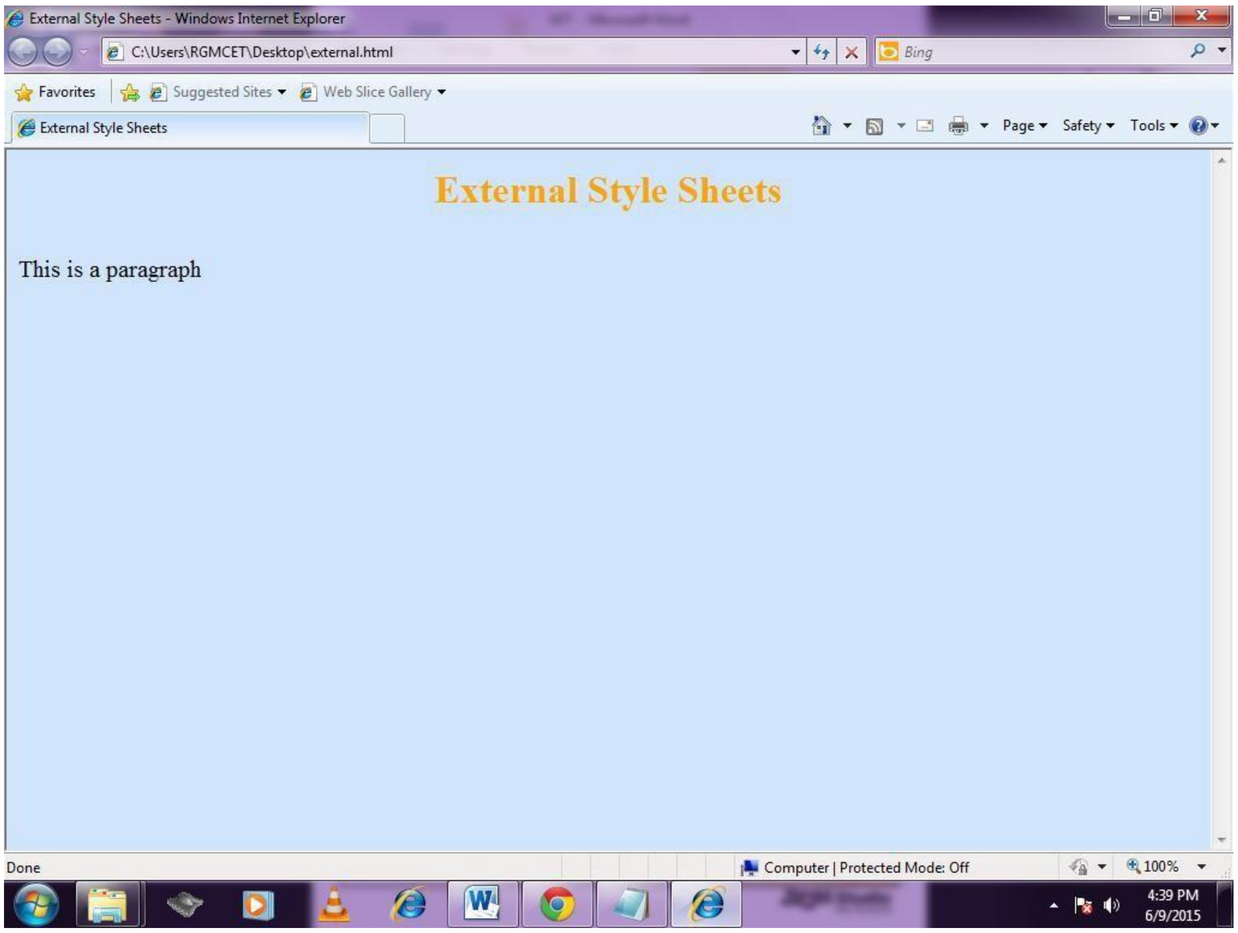
```
body {background-color: #d0e4fe;}
h1 {
color: orange; text-align: center;
}
p {
font-family: "Times New Roman"; font-size: 20px;
}
```

extern.html:

```
<html>
<head>
<title>External Style Sheets</title>
```

```
<link rel=”stylesheet” type=”text/css” href=”extern.css”>
</head>
<body>
<h1>External Style Sheets</h1><br>
<p>This is a paragraph
</body>
</html>
```

Output:


External Style Sheets - Windows Internet Explorer
C:\Users\RGMCET\Desktop\external.html
Bing
Favorites
Suggested Sites
Web Slice Gallery
External Style Sheets
Page
Safety
Tools
Done
Computer | Protected Mode: Off
100%
4:39 PM
6/9/2015

External Style Sheets

This is a paragraph

UNIT – II
JAVASCRIPT

INTRODUCTION TO JAVASCRIPT:

Static web pages are useful and can be informative. A number of technologies have been developed that enable the creation of web applications rather than static web pages. The java programming language is probably the best known such technology. Few programming languages other than java have been adapted for use in client-side web applications. One such language that is used in programming client-side web applications is javascript.

Javascript originates from a language called Live Script and was developed by Sun microsystems and Netscape navigator. Scripts are small pieces of code which accomplish a single relatively simple task. Javascript is a scripting language that is used for the development of Client-side-in-browser applications. Javascript is a simple script based language which is only suitable for fairly simple tasks. Javascript is a language that is best suited to tasks that run for a short time. Most of the developers experience problems when they try to build web pages which have embedded javascript.

Important things about Javascript:

1. **Javascript is embedded into HTML:**

   Javascript code is usually embedded into HTML code and is executed within the HTML document. Javascript has no user interface and it relies on HTML to provide a means of interaction with the users. Most of the javascript objects have HTML tags and provide event-driven code to execute it.

2. **Javascript is browser dependent:**

   Javascript depends on the web browser to support it. If the browser does not support it javascript will be ignored. Javascript was given support from I.E 3.0 & N.N 2.0 onwards.

3. **Javascript is an interpreted language:**

   Javascript is interpreted at runtime by the browser before it is executed. It is not compiled into a separate program like .exe but remains part of HTML file.

4. **Javascript is a loosely typed language:**

   Javascript is very flexible when compared to java. There is no need to specify the data type of a variable while declaring it. We can declare variables whenever it is necessary i.e., variables can be declared explicitly.

5. **Javascript is an object-based language:**

   Javascript is an object-based language that means that we can work with objects that encapsulate data and behaviour. However, javascript object model is instance based and there is no inheritance.

6. **Javascript is Event-Driven:**

   HTML objects such as buttons are enhanced to support event handlers. We can specify functionality.

7. **Javascript is not Java:**

   Java is object oriented language, whereas javascript is object-based, interpreted, loosely-typed language meant for creating scripts.

8. **Javascript is multi functional:**

   Javascript can be used to:

   - Enhance HTML pages.
   - Develop client-side applications.
   - Build to a certain extent client/server web applications.
   - Create extensions to a web server.
   - Provide database connectivity without using CGI.

9. **Javascript language spans context:**

   Javascript can be used in server side Netscape live wire pro environment and microsoft's Active X server framework. It is not just a client side scripting tool.

Benefits of Javascript:

- It is widely supported in web browsers.
- It gives easy access to the document objects and can manipulate most of them.
- Javascript can give interesting animations without long download times associated with any multimedia data types.
- Web surfers do not need any special plug-in to use your scripts.
- Javascript is relatively secure i.e., javascript can neither read from a local hard drive nor write to it and we cannot get a virus directly from javascript.

Problems with Javascript:

- If our script does not work then our page is useless.
- Most of the web surfers disable javascript support in their browsers because of the problems of broken scripts.
- Scripts can run slowly and complex scripts can take a long time to start up.

Javascript Basics:

Javascript programs contain variables, objects and functions. The key points that we need to apply in all scripts are:

- Each line of code is terminated by semicolon.
- Blocks of code must be surrounded by a pair of curly braces. A block of code is a set of instructions that are to be executed together as a unit.
- Functions have parameters which are passed inside parentheses.
- Variables are declared using keyword **"var"**.
- Scripts neither require a main function nor an exit condition. Execution of scripts starts with the first line of code and runs until there is no more code.

Javascript and HTML page:

Javascript need to be included in a HTML page. We cannot 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.

Javascript is embedded in HTML using <SCRIPT> element. Usually <SCRIPT> is included in <HEAD> element. A simple javascript code:

```
<html>
        <head>
                <title>Javascript</title>
                <script language="javascript">
                        document.write("Welcome to Javascript");
                </script>
        </head>
        <body>
```

```
            <h1>Javascript First Page</h1>
        </body>
</html>
```

If javascript is written as a separate file, then it must be saved with .js extension. External javascript file is included into HTML by using src attribute of <SCRIPT> element. Then also the script is evaluated when page loads and before any script action takes place.

In case of functions, all javascript statements within the function block are interpreted but executed only when the function is called from a javascript event. Javascript statements which are not in a function block are executed after document loads into browser.

JavaScript alert()

- The alert() method in JavaScript is used to display a virtual alert box. It is mostly used to give a warning message to the users. It displays an alert dialog box that consists of some specified message (which is optional) and an OK button. The syntax is:

alert("message");

JavaScript prompt()

The prompt() method in JavaScript is used to display a prompt box that prompts the user for the input. It is generally used to take the input from the user before entering the page. When the prompt box pops up, we have to click "OK" or "Cancel" to proceed. Syntax is:

prompt("message", "default")

Javascript Statements:

Programs are composed of data and code which manipulates that data. Program instructions are grouped into units called statements. We create programs from lot of statements.

If-else:

The JavaScript if-else statement is used to execute the code whether condition is true or false. There are three forms of if statement in JavaScript.

- if Statement
- if else statement
- if else if statement

If statement:

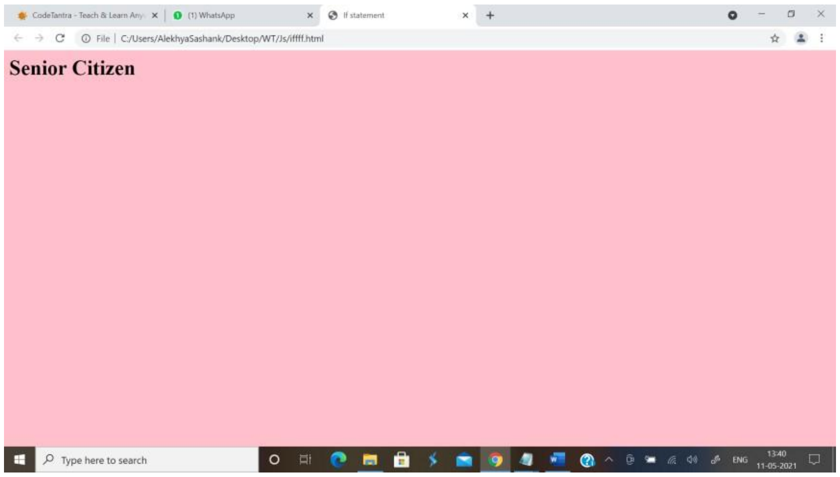
It evaluates the content only if expression is true. The signature of JavaScript if statement is:

```
if(expression){
    //content to be evaluated
}
```

Example:

```
<html>
<head>
<title>If statement</title>
</head>
<body bgcolor="pink">
<script type="text/javascript">
var age=parseInt(prompt("Enter age of the person",""));
if(age>60){
document.write("<h1>Senior Citizen</h1>");
}
</script>
</body>
</html>
```

Output:



Senior Citizen

JavaScript If...else Statement:

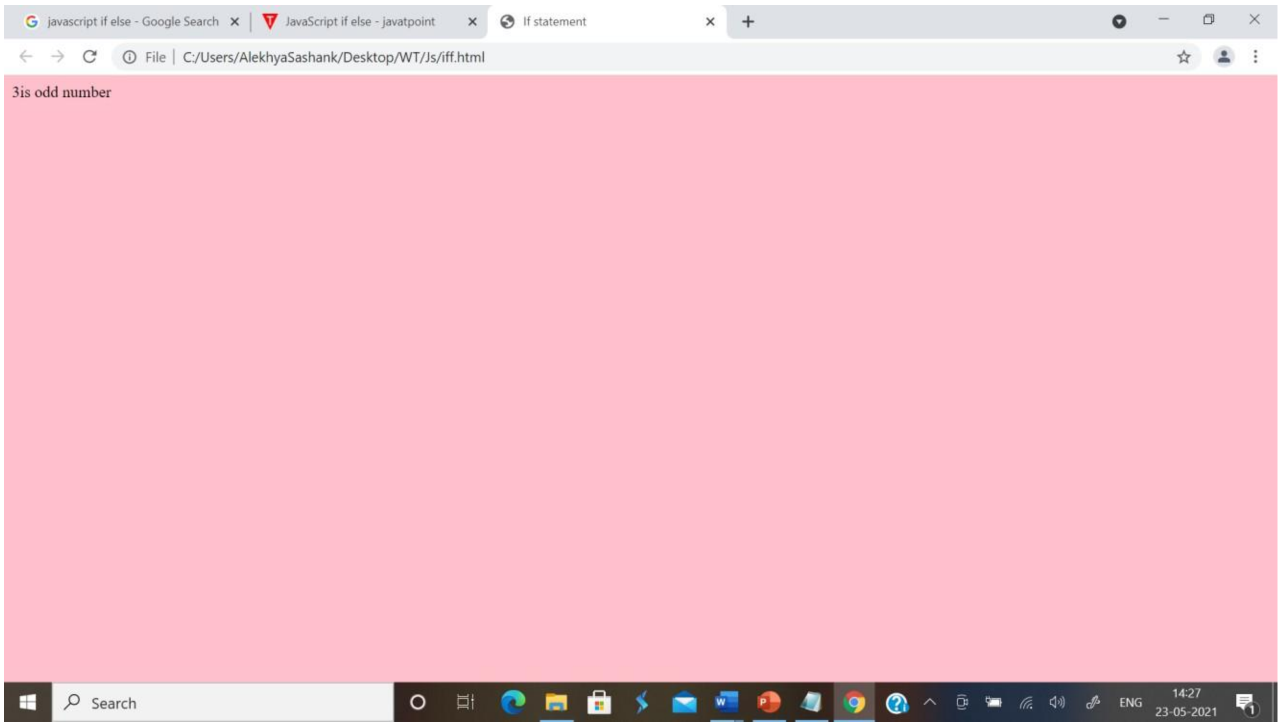
It evaluates the content whether condition is true of false. The syntax of JavaScript if-else statement is:

```
if(expression){
//content to be evaluated if condition is true
}
else{
//content to be evaluated if condition is false
}
```

Example:

```
<html>
<head>
<title>If statement</title>
</head>
<body bgcolor="pink">
<script language="javascript">
var n=parseInt(prompt("Enter a value"," "));
if(n%2==0){
document.write(n+"is even number");
}
else{
document.write(n+"is odd number");
}
</script>
</body>
</html>
```

Output:



```
3is odd number
```

JavaScript If...else if statement:

It evaluates the content only if expression is true from several expressions. The signature of JavaScript if else if statement is:

```
if(expression1){
//content to be evaluated if expression1 is true
}
else if(expression2){
//content to be evaluated if expression2 is true
}
else if(expression3){
//content to be evaluated if expression3 is true
}
else{
//content to be evaluated if no expression is true
}
```

Example:

```
<html>
<head>
<title>if statement</title>
</head>
<body bgcolor="pink">
<script language="javascript">
var m1=parseInt(prompt("Enter Marks1","0"));
var m2=parseInt(prompt("Enter Marks2","0"));
var m3=parseInt(prompt("Enter Marks3","0"));
var avg=parseInt((m1+m2+m3)/3);
if(avg>70){
document.write("<h1>Distinction</h1>");
}
else if(avg<70&&avg>60){
document.write("<h1>First Class</h1>");
}
else if(avg<60&&avg>50){
document.write("<h1>Second class</h1>");
}
else{
document.write("<h1>Fail</h1>");
}
</script>
</body>
</html>
```

JavaScript Switch:

The JavaScript switch statement is used to execute one code from multiple expressions. The signature of JavaScript switch statement is :

```
switch(expression){
```

```
case value1:
 code to be executed;
 break;
case value2:
 code to be executed;
 break;
......

default:
 code to be executed if above values are not matched;
}
```

Example:

```
<html>
<head>
<title>Switch</title>
</head>
<body bgcolor="pink">
<script language="javascript">
var a=parseInt(prompt("Enter a value"," "));
var b=parseInt(prompt("Enter b value"," "));
var ch=parseInt(prompt("Enter your choice"," "));
switch(ch){
case 1: document.write("<h1>Addition is:"+(a+b)+"<h1>");
            break;
case 2: document.write("<h1>Subtraction is:"+(a-b)+"<h1>");
            break;
case 3: document.write("<h1>Multiplication is:"+(a*b)+"<h1>");
            break;
case 4: document.write("<h1>Division is:"+(a/b)+"<h1>");
            break;
default: document.write("<h1>Invalid Choice</h1>");
}
</script>
</body>
</html>
```

Javascript Loops:

The JavaScript loops are used to iterate the piece of code. Various types of loops are:

1. for loop
2. while loop
3. do-while loop

JavaScript For loop:

The JavaScript for loop iterates the elements for the fixed number of times. Syntax is:

```
 for (initialization; condition; increment)
{
   code to be executed
}
```

JavaScript while loop:

The JavaScript while loop iterates the elements for the infinite number of times. Syntax is:

```
while (condition)
{
   code to be executed
}
```

JavaScript do while loop:

The JavaScript do while loop iterates the elements for the infinite number of times like while loop. But, code is executed at least once whether condition is true or false. Syntax is:

```
do{
   code to be executed
}while (condition);
```

Example:

```
<html>
<head>
<title>Multiplication</title>
</head>
<body bgcolor="pink">
<script language="javascript">
var n=parseInt(prompt("Enter a number",""));
var i;
for(i=1;i<=10;i++){
document.write("<b>"+n+"*"+i+"="+(n*i)+"<br>");
}
</script>
</body>
</html>
```

Javascript Events:

JavaScript's interaction with HTML is handled through events that occur when the user or the browser manipulates a page. When the page loads, it is called an event. When the user clicks a button, that click too is an event. Events are a part of the Document Object Model (DOM) Level 3 and every HTML element contains a set of events which can trigger JavaScript Code.

Mouse events:

Event Performed	Event Handler	Description
click	onclick	When mouse click on an element
mouseover	onmouseover	When the cursor of the mouse comes over the element
mouseout	onmouseout	When the cursor of the mouse leaves an element
mousedown	onmousedown	When the mouse button is pressed over the element
mouseup	onmouseup	When the mouse button is released over the element
mousemove	onmousemove	When the mouse movement takes place.

Keyboard events:

Event Performed	Event Handler	Description
Keydown & Keyup	onkeydown & onkeyup	When the user press and then release the key

Form Events:

Event Performed	Event Handler	Description
focus	onfocus	When the user focuses on an element
submit	onsubmit	When the user submits the form
blur	onblur	When the focus is away from a form element
change	onchange	When the user modifies or changes the value of a form element

Window/Document events:

Event Performed	Event Handler	Description
load	onload	When the browser finishes the loading of the page
unload	onunload	When the visitor leaves the current webpage, the browser unloads it
resize	onresize	When the visitor resizes the window of the browser

Javascript Functions:

A javascript function contains code that will be executed by an event or by a call to function. We may call a function from anywhere within a page. Functions can be defined both in <head> and <body> section of a document. Syntax to define a function is:

```
function function_name(var1, var2,…..,varn)
{
//Block of code
}
```

- A function with no parameters must include the parentheses () after function name.
- The keyword function must be written in lower case, otherwise javascript error occurs.

Example:

```
<html>
<head>
<title>functions</title>
<script language="javascript">
function displayMessage(){
alert("This is a function");
}
</script>
</head>
```

```
<body bgcolor="pink">
<form>
<input type="button" value="Click Me" onClick="displayMessage()">
</form>
</body>
</html>
```

Function with parameters:

```
<html>
<head>
<title>Function</title>
</head>
<body onLoad="pinfo('abc',30)">
<script language="javascript">
function pinfo(name,age){
document.write("<center><h1>User Information</h1><br>");
document.write("<h3>Name is:"+name+"</h3>");
document.write("<br><h3>Age is:"+age+"</h3>");
document.write("</center>");
document.close();
}
</script>
</body>
</html>
```

Objects in Javascript:

Window object:

Various properties & methods supported by window object are:

- **open():** It is used to open a new window. Two arguments are provided. **URL** that specifies the path of documents which should be loaded in the window and **Name of window.**
- **close():** This method is used to close the current window.
- **scroll():** By using this method contents of a given window can be easily scrolled.

Apart from above methods, window object may also have properties such as **toolbar, location, menubar, scrollbar, resizeable etc..**

Examples:

Creating a new window and loading existing page:

```
<html>
<head>
<title>Window</title>
<script language="javascript">
function showNewPage(){
window.open("login.html");
}
</script>
</head>
<body bgcolor="pink">
<center>
<form>
<input type="button" value="SeeLogin" onClick="showNewPage()">
</form>
</center>
</body>
</html>
```

Creating a new window:

```
<html>
<head>
<title>Window</title>
<script language="javascript">
function showNewPage(){
var newWindow=window.open("window.html","height=200,width=450");
newWindow.document.write("<center><h1>This is a new Window</h1></center>");
}
```

```
</script>
</head>
<body bgcolor="pink">
<center>
<form>
<input type="button" value="Open page" onClick="showNewPage()">
</form>
</center>
</body>
</html>
```

Document object:

Document refers to page which will be displayed as soon as the browser window is opened. Various methods/properties supported by the document object are:

- **write()/writeln():** We can create HTML pages using javascript by using write()/writeln() methods. Data can also be inserted.
- **bgcolor/fgcolor:** These are the same properties that can be set in the <body> tag. The only difference here is that the values can be set from javascript.
- **anchors:** It is an array holding the names of anchor elements appearing on web page.
- **links:** It is an array holding all links appearing on web page.
- **forms:** It is an array that contains all of the HTML forms.
- **close():** The document is not completely written until the close() method is called. The browser keep waiting for more data if user do not call this method.

Form object:

Two aspects of form can be manipulated through javascript.

- Most commonly the data that is entered onto the form can be checked at submission.
- We can actually build forms through javascript.

Various form events are:

- **onClick:** This event is triggered when the user clicks on an element.
- **onSubmit:** This event can only be triggered when the form is submitted.
- **onReset:** This event is triggered when the form is reset by the user.

Simple form validation:

```
<html>
<head>
<title>Validation</title>
<script language="javascript">
function validate(){
var value=document.forms[0].elements[0].value;
if(value!="abc"){
document.forms[0].reset();
}
else{
alert("Hi abc");
}
}
</script>
</head>
<body bgcolor="pink">
<form>
Name:<input type="text" name="txt1" value=""><br>
<input type="submit" value="Submit" onClick="validate()">
</form>
</body>
</html>
```

Math object:

Methods supported by math object are:

- **min():** Displays minimum of two numeric values entered. (min(45,65)).
- **max():** Displays maximum of two numeric values entered. (min(45,65)).
- **abs():** Displays the absolute value of numeric entity entered into it.
- **ceil():** Displays the rounded value of the integer entered into it. (ceil(5.2)=6)
- **round():** It rounds the value entered to its nearest integer. (round(5.2)=5)

- **sqrt():** Displays the square root of the value entered into it.
- **sin():** Displays the trigonometric sine value.
- **cos():** Displays the trigonometric cosine value.
- **tan():** Displays the trigonometric tangent value.
- **log():** Displays the logarithmic equivalent value.

Browser object:

The browser object is also called as navigator object. Methods supported by browser object are:

- **navigator.appcodeName:** The internal name for the browser.
- **navigator.appName:** This is the public name of the browser.
- **navigator.appversion:** The version number, platform on which the browser is running and the version of navigator with which it is compatible.
- **navigator.userAgent:** The strings appcodeName and appVersion concatenated together.
- **navigator.plugins:** An array containing details of all installed plugins.
- **navigator.mimeTypes:** An array of all supported MIME types.

Date object:

Javascript Date provides functions to perform many different date manipulations. In javascript dates and times represent the number of milliseconds since 01/01/1970 UTC. Javascript like most programming systems has two separate notions of time:

- UTC is universal time, also known as Greenwich Mean Time, which is the standard time throughout the world.
- Local time is the time on the machine which is executing the script.

Date object Methods:

1. **Date():** Construct an empty date object.
2. **Date(milliseconds):** Construct a new date object based upon the number of milliseconds which have elapsed since 00:00:00 hours on 01/01/1970.
3. **Date(String):** Create a date object based upon the contents of a text string.
4. **Date(year,month,day[hour,minute,second]):** Create a new date object based upon numeric values for year, month and day. Optional time values may also be supplied.
5. **Parse(string):** Returns the number of milliseconds since midnight on 01/01/1970.

6. **getDate():** Return the day of the month.
7. **getDay():** Return the day of the week.
8. **getHours():** Return the hours.
9. **getMilliseconds():** Return the milliseconds.
10. **getMinutes():** Return the minutes.
11. **getSeconds():** Return the seconds.
12. **getMonth():** Return the month.
13. **getFullYear():** Return the year as a four digit number.
14. **getTime():** Return the number of milliseconds since midnight on 01/01/1970.
15. **setDate(day):** Set the day value of the object. Accept values in the range 1 to 31.
16. **setFullYear(year[,month,day]):** sets the year value of the object.
17. **setHours(hours[,mins,secs,ms]):** Set the hours value of the object to an integer in the range 0 – 23.
18. **setMilliseconds(ms):** Set the milliseconds value of the object in the range 0 - 999.
19. **setMinutes(min[,secs,ms]):** Set the minutes value of the object in the range 0 - 59.
20. **setSeconds(secs[,ms]):** Set the seconds value of the object to an integer in the range 0 - 59.
21. **setMonth(month[,day]):** Set the month value of the object to an integer in the range 0 - 11.

Example:

```
<html>
<head>
<title>Date</title>
</head>
<body onLoad="Dater()">
<script language="javascript">
function Dater(){
var today=new Date();
var  yesterday=new  Date();
var diff=today.getDate()-1;
Yesterday.setDate(diff);
document.write("<h3>The date is:"+today+"</h3>");
document.write("<h3>The date yesterday was:"+yesterday+"</h3>");
document.close();
}
</script>
</body>
</html>
```

Javascript Arrays:

The basic array operations are:

- Creation of Arrays.
- Adding elements.
- Accessing individual elements.
- Removing elements.

Creating Arrays:

Javascript arrays can be created in three different ways:

1. The easiest way is simply to declare a variable and pass it some elements in array format.

    var days=["Monday","Tuesday","Wednesday"];

2. The second approach is to create an array object using the keyword new and a set of elements to store:

    var days=new Array("Monday","Tuesday","Wednesday");

3. Finally, an empty array object which has a space for a number of elements can be created:

    var days=new Array(3);

Adding elements to Arrays:

Adding an element to an array can be done as shown below:

var days[4]="Thursday";

In javascript we have a benefit. If the array is full, then also we can add elements to array. The interpreter simply extends the array and inserts new item. For example:

Accessing Array elements:

The elements in the array are accessed through their index. The same access method is used to find elements and to change their values.

Length:

When accessing array elements we need to know how many elements have been stored into the array. This is done through the length attribute. The index number runs from 0 – length-1.

Example:

```
<html>
<head>
<title>Arrays</title>
</head>
<body>
<script language="javascript">
document.write("<h1> Looping through the Array</h1>");
var data =[10,20,30,40];
var len=data.length;
for(var count=0;count<len;count++){
        document.write(data[count]+",");
}
document.close();
</script>
</body>
</html>
```

Removing Array elements:

Javascript does not provide a built-in function to remove array members. To remove array elements we use the following procedure:

1. Read each element in the array.
2. If the element is not the one which we want to delete, copy it to temporary array.
3. If the element is the one we want to delete, do nothing.
4. Increment loop counter.
5. Repeat the process and copy the array elements again into original array.

Example:

```
<html>
<head>
```

```
<title>Arrays</title>
</head>
<body>
<script language="javascript">
document.write("<h1> Removing Array elements</h1>");
var data =[10,20,30,40];
var len=data.length;
for(var count=0;count<len;count++){
        document.write(data[count]+",");
}
var rem=prompt("Enter item to remove","");
var tmp=new Array(data.length-1);
var count2=0;
for(count=0;count<len;count++){
if(data[count]==rem){
}
else{
tmp[count2]=data[count];
count2++;
}
}
data=tmp;
var len=data.length;
for(var count=0;count<len;count++){
        document.write(data[count]+",");
}
document.close();
</script>
</body>
</html>
```

Dynamic HTML using Javascript:

In case of HTML dynamic means subject to change at any time. The DHTML is simply HTML that has the ability to change after the browser has loaded the page. DHTML is also called as **"Animated HTML"** i.e., anything that is written in HTML can be redone after the page loads.

This dynamic capability is achieved through a combination of several key DHTML components that work in conjunction with HTML. DHTML can also be described as the combination of HTML, stylesheets and scripts that allow documents to be animated. The features of dynamic HTML are:

1. **Changing tags and contents:**

   In this we deal with technology of changing the contents and tags of text using the concept of DOM.
2. **Live positioning of elements:**

   In this DHTML has introduced a concept of dynamically positioning tags in the document.
3. **Dynamic fonts:**

   We can increase the font of text dynamically.
4. **Data Binding:**

   This enables page elements such as table cells to attach themselves to data records. Changes to a record are updated on page, and user modification updated on data record.

Example:

```
<html>
<head>
<title>DHTML</title>
</head>
<body bgcolor="pink">
<center>
<h1 onMouseOver="this.style.color='red';" onMouseOut="this.style.color='blue';">
Text changes to red with the mouse
</h1>
</center>
```

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

Dynamic content – Changing web pages:

There are several methods that insert HTML in pages:

1. insertAdjacentHTML.
2. insertAdjacentText.
3. innerText.
4. outerText.
5. innerHTML.
6. outerHTML.

insertAdjacentHTML & insertAdjacentText:

The insertAdjacentHTML method lets us to insert HTML next to an element that already exists and insertAdjacentText method lets us insert text in the same way.

We can determine where the new text or HTML will go with respect to existing element by passing the constraints **"BeforeBegin, AfterBegin, BeforeEnd or AfterEnd"**.

innerText: Let us change the text between the start and end tags.

outerText: Let us change all the text including start and end tags.

innerHTML: Changes the contents of elements between start and end tags.

outerHTML: Changes contents of an element including the start and end tags.

createElement: It is used to create new web page elements and use methods like insertBefore() and insertAfter() to insert those elements into web page.

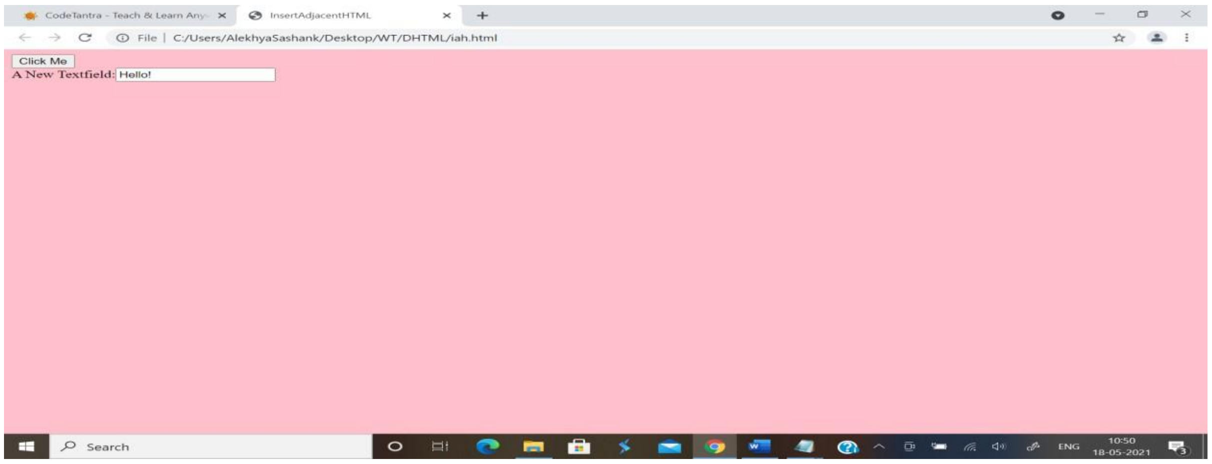
Examples:

insertAdjacentHTML():

```
<html>
<head>
<title>InsertAdjacentHTML</title>
<script language="javascript">
function showMore(){
div1.insertAdjacentHTML('afterend','A New Textfield:<input type="text" value="Hello!">');
}
```

```
</script>
</head>
<body bgcolor="pink">
<div id="div1">
<input type="button" value="Click Me" onClick="showMore()">
</div>
</body>
</html>
```

Output:



innerText:

```
<html>
<head>
<title>InnerText</title>
<script language="javascript">
function changeHeader(){
header.innerText="This is new Header";
}
</script>
</head>
```

```
<body bgcolor="pink">
<center>
<h1 id="header" onClick="changeHeader()">Dynamic HTML</h1>
</center>
</body>
</html>
```

innerHTML:

```
<html>
<head>
<title>InnerText</title>
<script language="javascript">
function changeHeader(){
header.innerHTML="<marquee>This is new Header</marquee>";
}
</script>
</head>
<body bgcolor="pink">
<center>
<h1 id="header" onClick="changeHeader()">Dynamic HTML</h1>
</center>
</body>
</html>
```

outerHTML:

```
<html>
<head>
<title>InnerText</title>
<script language="javascript">
function changeHeader(){
header.outerHTML="<marquee style='font-size:50'>This is new Header</marquee>";
}
```

```
</script>
</head>
<body bgcolor="pink">
<center>
<h1 id="header" onClick="changeHeader()">Dynamic HTML</h1>
</center>
</body>
</html>
```

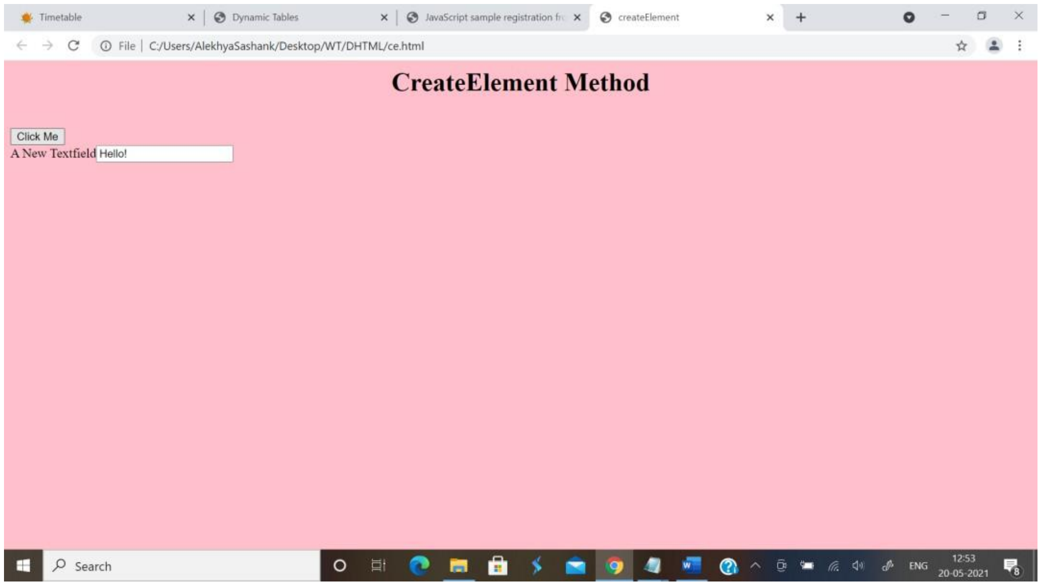
createElement() & createTextNode():

```
<html>
<head>
<title>createElement</title>
<script language="javascript">
function showMore(){
var newDiv,newTextfield,newText;
newDiv=document.createElement("DIV");
newDiv.id="div1";
newTextfield=document.createElement("INPUT");
newTextfield.type="text";
newTextfield.value="Hello!";
newText=document.createTextNode("A New Textfield");
newDiv.insertBefore(newText,null);
newDiv.insertBefore(newTextfield,null);
document.body.insertBefore(newDiv,null);
}
</script>
</head>
<body bgcolor="pink">
<h1 align="center">CreateElement Method</h1><br>
<input type="button" value="Click Me" onClick="showMore()">
```

</body>

</html>

Output:


Timetable
Dynamic Tables
JavaScript sample registration fr
createElement
File | C:/Users/AlekhyaSashank/Desktop/WT/DHTML/ce.html
Click Me
Search
ENG
12:53
20-05-2021

CreateElement Method

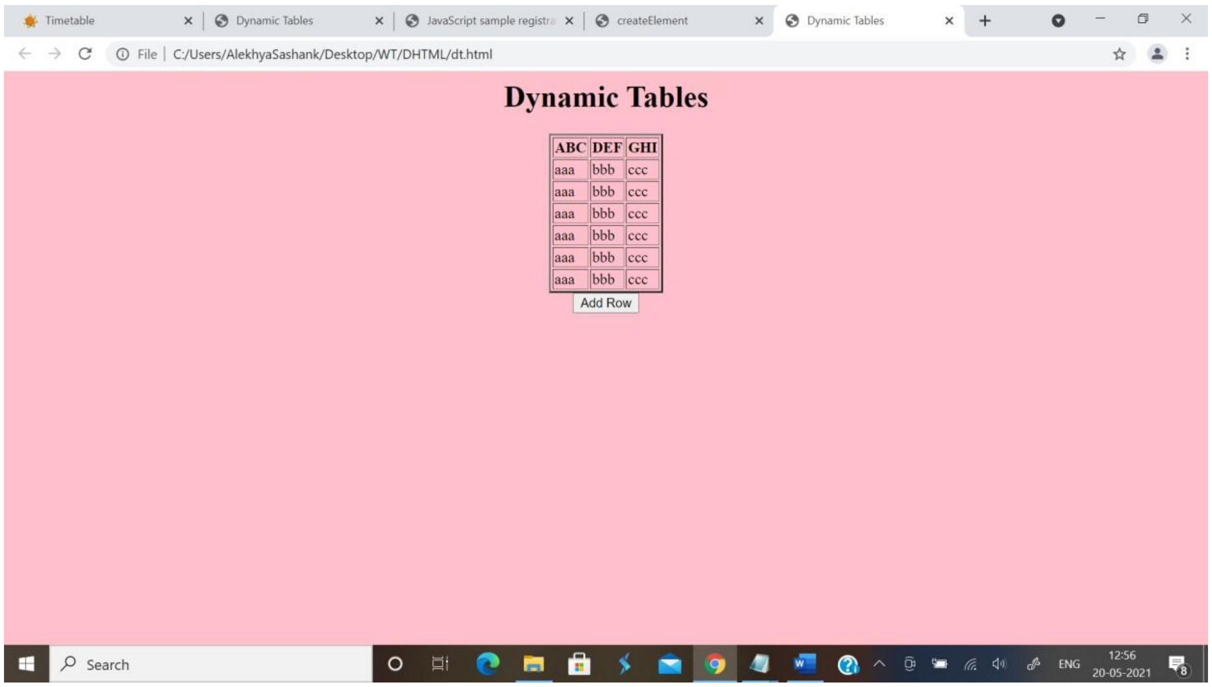
A New Textfield Hello!

Creating Dynamic Tables:

```
<html>
<head>
<title>Dynamic Tables</title>
<script language="javascript">
function addRow(){
var newRow=table1.insertRow(3);
var newCell=newRow.insertCell(0);
newCell.innerText="aaa";
var newCell=newRow.insertCell(1);
newCell.innerText="bbb";
var newCell=newRow.insertCell(2);
newCell.innerText="ccc";
}
</script>
</head>
```

```
<body bgcolor="pink">
<center>
<h1>Dynamic Tables</h1>
<table id="table1" border="2">
<tr>
<th>ABC</th>
<th>DEF</th>
<th>GHI</th>
</tr>
<tr>
<td>aaa</td>
<td>bbb</td>
<td>ccc</td>
</tr>
<tr>
<td>aaa</td>
<td>bbb</td>
<td>ccc</td>
</tr>
</table>
<input type="button" value="Add Row" onClick="addRow()">
</center>
</body>
</html>
```

Output:



Dynamic Tables

ABC	DEF	GHI
aaa	bbb	ccc
aaa	bbb	ccc
aaa	bbb	ccc
aaa	bbb	ccc
aaa	bbb	ccc
aaa	bbb	ccc

Data Validation using Javascript:

Data validation is a process of verifying/validating the data entered by the user. Data validation can be done in two ways:

1. Client-side data validation.
2. Server-side data validation.

Client-side data validation is a process of verifying the data entered by the user before it is submitted to the server.

Server-side data validation is a process of verifying the data at server side with the database data.

Example:

Registration Form Validation:

```
<html>
<head>
<title>JavaScript sample registration from validation</title>
<script type='text/javascript'>
function formValidation()
```

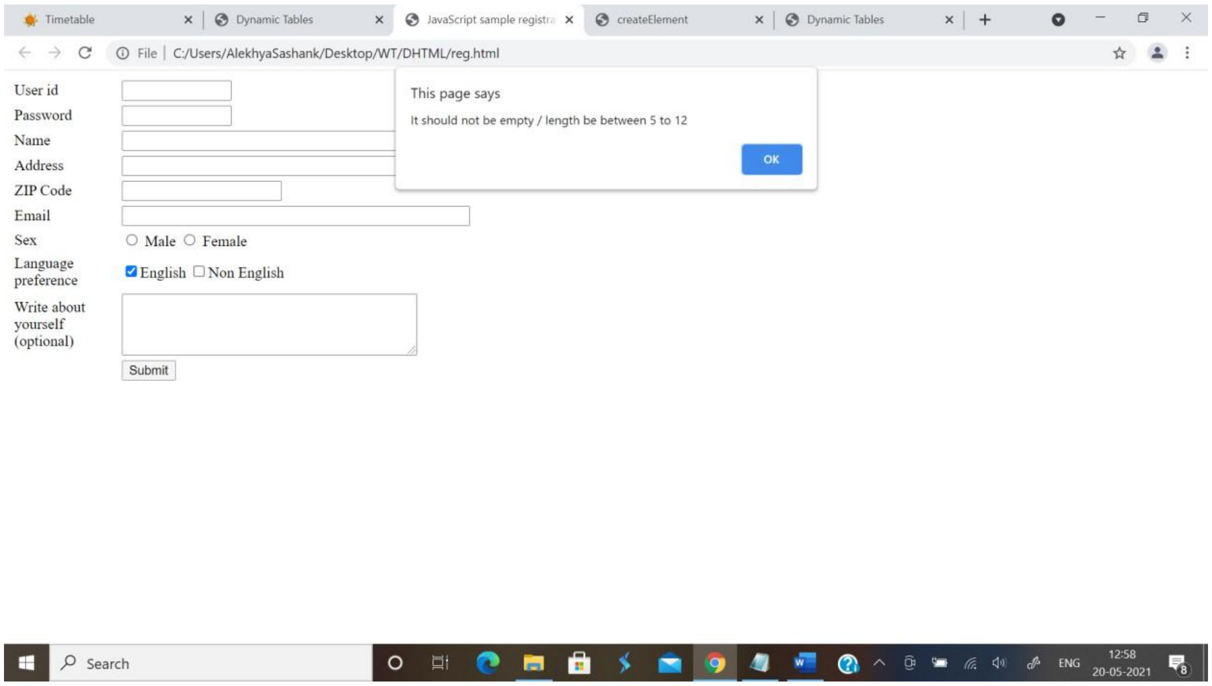
```
{
var uid = document.form1.userid;
var passid = document.form1.passid;
var uname = document.form1.username;
var uadd = document.form1.address;
var uzip = document.form1.zip;
var uemail = document.form1.email;
if(userid_validation(uid,5,12))
{
if(userid_validation(passid,7,12))
{
if(allLetter(uname))
{
if(alphanumeric(uadd))
{
if(allnumeric(uzip))
{
if(ValidateEmail(uemail))
{
}
}
}
}
}
}
return false;
}
function userid_validation(uid,mx,my)
{
var uid_len = uid.value.length;
```

```
if (uid_len == 0 || uid_len >= my || uid_len < mx)
{
alert("It should not be empty / length be between "+mx+" to "+my);
uid.focus();
return false;
}
return true;
}
function allLetter(uname)
{
var letters = /^[A-Za-z]+$/;
if(uname.value.match(letters))
{
return true;
}
else
{
alert('Please input alphabet characters only');
uname.focus();
return false;
}
}
function alphanumeric(uadd)
{
var letters = /^[0-9a-zA-Z]+$/;
if(uadd.value.match(letters))
{
return true;
}
else
```

```
{
alert('Please input alphanumeric characters only');
uadd.focus();
return false;
}
}
function allnumeric(uzip)
{
var numbers = /^[0-9]+$/;
if(uzip.value.match(numbers))
{
return true;
}
else
{
alert('Please input numeric characters only');
uzip.focus();
return false;
}
}
function ValidateEmail(uemail)
{
var mailformat = /^\w+([\.-]?\w+)*@\w+([\.-]?\w+)*(\.\w{2,3})+$/;
if(uemail.value.match(mailformat))
{
return true;
}
else
{
alert("You have entered an invalid email address!");
uemail.focus();
return false;
}
}
</script>
</head>
<body>
<form name='form1' onsubmit='return formValidation()' >
<table width="500" cellpadding="3" style="border-collapse: collapse;">
<tr>
<td>User id </td>
<td><input type="text" name="userid" size="12" /></td>
</tr>
<tr>
<td>Password</td>
<td><input type="password" name="passid" size="12" /></td>
</tr>
<tr>
<td>Name</td>
<td><input type="text" name="username" size="50" /></td>
</tr>
<tr>
<td>Address</td>
<td><input type="text" name="address" size="50" /></td>
</tr>
<tr>
<td>ZIP Code </td>
<td><input type="text" name="zip" /></td>
</tr>
<tr>
```

```
<td>Email</td>
<td><input type="text" name="email" size="50" /></td>
</tr>
<tr>
<td>Sex</td>
<td><input type="radio" name="msex" value="Male" /> Male
<input type="radio" name="fsex" value="Female" /> Female</td>
</tr>
<tr>
<td>Language preference</td>
<td><input type="checkbox" name="en" value="en" checked />English
<input type="checkbox" name="nonen" value="noen" />Non English</td>
</tr>
<tr>
<td>Write about yourself<br>
(optional)</td>
<td><textarea name="desc" rows="4" cols="40"></textarea></td>
</tr>
<tr>
<td>&nbsp;</td>
<td><input type="submit" name="submit" value="Submit" /></td>
<td>&nbsp;</td>
</tr>
</table>
</form>
</body>
</html>
```

Output:



This page says

It should not be empty / length be between 5 to 12

User id	
Password	
Name	
Address	
ZIP Code	
Email	
Sex	○ Male ○ Female
Language preference	☑ English ☐ Non English
Write about yourself (optional)	

UNIT – III
XML

Introduction:

XML stands for "Extensible Markup Language". XML is a text-based markup language. A markup language is a set of instructions often called as tags which can be added to text files. When the file is processed by a suitable application the tags are used to control the structure and presentation of data contained in the file. Most commonly tags are used by application when presenting data.

XML is more than HTML because, HTML only deals with presentation and carries no information about the data, whereas XML deals with the representation of data and carries information about the data. In XML, we create our own tags and also syntax for those tags that can let the document structure follow the data structure. As we are defining our own tags and there are no predefined tags, XML is defined to be "self-descriptive". Using scripting languages like javascript we can reach to various elements of an xml page and make use of data.

When compared with HTML, XML provides a way of structuring the data and store data with focus on "What data is", where HTML was designed to display data with focus on "How data looks".

XML is nothing special, it is just a plain text. Software that can handle plain text can also handle XML. XML is a software/hardware independent tool for carrying information.

Advantages of XML:

- **Readability:**

  XML document is plain text and human readable.
- **Hierarchal:**

  XML document has a tree like structure which is enough to express complex data as simple as possible.
- **Language Independent:**

  XML documents are language neutral. For example, a java program can generate an XML which can be parsed by a program written in C++ or perl.
- **Platform Independent:**

  XML files are operating system independent.

Uses of XML:

- XML is used to describe the contents of a document.
- XML is used in messaging, where applications exchange data between them.
- The data can be extracted from the database, can be preserved with original information and can be used in more than one application in different ways.

Important points about XML:

- XML is case sensitive.
- All the XML files are stored with .xml extension.
- Every XML document begin with <?xml version="1.0"?>

Example: XML to store customer information of a supermarket.

```
<?xml version="1.0"?>
<document>
        <customer>
                <name>
                        <firstname>aaa</firstname>
                        <lastname>bbb</lastname>
                </name>
                <date>01 jan 2017</date>
                <orders>
                        <item>
                                <product>Chocolates</product>
                                <number>777</number>
                                <price>250</price>
                        </item>
                        <item>
                                <product>Sweets</product>
                                <number>888</number>
                                <price>450</price>
                        </item>
        </orders>
    </customer>
</document>
```

Valid & Well-formed XML document:

An XML document is said to be valid only if it is associated with a **"Document Type Definition"** or **"XML schema"**.

An XML document is said to be well-formed only if it satisfies the below constraints:

- All tags must have corresponding end tag.

Well-Formed	Not Well-Formed

`<personnel>`
`  <employee>`
`    <name>aaa</name>`
`    <id>504</id>`
`    <age>39</age>`
`  </employee>`
`</personnel>`
	`<personnel>`
	`  <employee>`
	`    <name>aaa</name>`
	`    <id>504</id>`
	`    <age>39`
	`  </employee>`
	`</personnel>`

- Must have no overlapping of tags.

Well-Formed	Not Well-Formed

`<personnel>`
`  <employee>`
`    <name>aaa</name>`
`    <id>504</id>`
`    <age>39</age>`
`  </employee>`
`</personnel>`
	`<personnel>`
	`  <employee>`
	`    <name>aaa</name>`
	`    <id>504</id>`
	`    <age>39`
	`  </personnel>`
	`</employee>`

Most of the XML parsers require XML documents to be well-formed, but not necessarily valid.

Document Type Definition(DTD):

- Document Type Definition specifies syntax for XML document i.e., it specifies rules that apply to the data.
- The XML document contains data, whereas DTD contains rules that apply to the data.
- A DTD defines the document structure with a list of legal elements and attributes.

Syntax of DTD:

```
<!DOCTYPE ROOTELEMENT[
<!ELEMENT ELEMENT_NAME1(subelement1,subelement2….)>
<!ELEMENT ELEMENT_NAME2(subelement1,subelement2….)>
.
.
<!ELEMENT ELEMENT_NAMEn(subelement1,subelement2….)>
]>
```

There are two types of DTD:

1. Internal DTD.
2. External DTD.

Internal DTD is the one in which we specify the DTD within the XML document. The scope of Internal DTD is limited to one document only.

Example:

```
<?xml version="1.0"?>
<!DOCTYPE document[
<!ELEMENT document (customer)*>
<!ELEMENT customer (name,date,orders)>
<!ELEMENT name (firstname,lastname)>
<!ELEMENT firstname(#PCDATA)>
<!ELEMENT lastname (#PCDATA)>
<!ELEMENT date (#PCDATA)>
<!ELEMENT orders (item)*>
<!ELEMENT item (product,number,price)>
<!ELEMENT product (#PCDATA)>
<!ELEMENT number (#PCDATA)>
<!ELEMENT price (#PCDATA)>
]>
```

```
<document>
        <customer>
                <name>
                        <firstname>aaa</firstname>
                        <lastname>bbb</lastname>
                </name>
                <date>01 jan 2017</date>
                <orders>
                        <item>
                                <product>Chocolates</product>
                                <number>777</number>
                                <price>250</price>
                        </item>
                        <item>
                                <product>Sweets</product>
                                <number>888</number>
                                <price>450</price>
                        </item>
                </orders>
        </customer>
</document>
```

External DTD is the one where DTD is specified as a separate file and is saved with .dtd extension and contains only set of rules. To import the DTD file into XML we include the below statement in XML file:

<!DOCTYPE ROOT_ELEMENT SYSTEM “filename.dtd”>

Example:

Extern.dtd:

```
<!ELEMENT document (customer)*>
<!ELEMENT customer (name,date,orders)>
<!ELEMENT name (firstname,lastname)>
```

```
<!ELEMENT firstname(#PCDATA)>
<!ELEMENT lastname (#PCDATA)>
<!ELEMENT date (#PCDATA)>
<!ELEMENT orders (item)*>
<!ELEMENT item (product,number,price)>
<!ELEMENT product (#PCDATA)>
<!ELEMENT number (#PCDATA)>
<!ELEMENT price (#PCDATA)>
```

Cust.xml:

```
<?xml version="1.0"?>
<!DOCTYPE document SYSTEM "extern.dtd">
<document>
        <customer>
                <name>
                        <firstname>aaa</firstname>
                        <lastname>bbb</lastname>
                </name>
                <date>01 jan 2017</date>
                <orders>
                        <item>
                                <product>Chocolates</product>
                                <number>777</number>
                                <price>250</price>
                        </item>
                        <item>
                                <product>Sweets</product>
                                <number>888</number>
                                <price>450</price>
                        </item>
                </orders>
```

```
    </customer>
</document>
```

Specifying attributes in a DTD:

We can specify the attributes of the elements in DTD by using <!ATTLIST> element. This element holds a list of attributes for an element. We can specify default values for the attribute and also can specify whether the attribute is necessary for an element or not. The format of defining an attribute in DTD is:

<!ATTLIST ELEMENT_NAME ATTRIBUTE_NAME TYPE DEFAULT_VALUE>

The value supplied for attribute can be:

Value – The default value of an attribute.

#REQUIRED – The attribute is required.

#IMPLIED – The attribute is not required.

#FIXEDVALUE – The attribute value is fixed.

The default type used for attributes is unparsed character data.

Example:

```
<?xml version="1.0"?>
<!DOCTYPE document[
<!ELEMENT document (customer)*>
<!ELEMENT customer (name,date,orders)>
<!ELEMENT name (firstname,lastname)>
<!ELEMENT firstname(#PCDATA)>
<!ELEMENT lastname (#PCDATA)>
<!ELEMENT date (#PCDATA)>
<!ELEMENT orders (item)*>
<!ELEMENT item (product,number,price)>
<!ELEMENT product (#PCDATA)>
<!ELEMENT number (#PCDATA)>
<!ELEMENT price (#PCDATA)>
<!ATTLIST customer nationality CDATA "INDIAN"
                   Age         CDATA #IMPLIED
```

```
                        Type        CDATA #REQUIRED>
]>
<document>
        <customer>
                <name>
                        <firstname>aaa</firstname>
                        <lastname>bbb</lastname>
                </name>
                <date>01 jan 2017</date>
                <orders>
                        <item>
                                <product>Chocolates</product>
                                <number>777</number>
                                <price>250</price>
                        </item>
                        <item>
                                <product>Sweets</product>
                                <number>888</number>
                                <price>450</price>
                        </item>
                </orders>
        </customer>
</document>
```

XML Namespaces:

XML document allows us to create our own XML elements with our own element names. This can result in naming collision i.e., different XML elements can have the same name in one XML document. For example:

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

```
            <name>aaa</name>
            <dept>CSE</dept>
      </staff>
      <student>
            <name>bbb</name>
            <dept>CSE</dept>
      </student>
</college>
```

In the above example the XML elements <name> and <dept> of both staff and student have the common element names. This could provide naming collisions. To solve such problems in XML documents, we use the “XML Namespaces”. The XML namespaces allows us to prevent collision by differentiating the elements with same names by using namespace prefixes.

The xmlns attribute:

XML namespaces use the keyword xmlns to create namespace prefixes and assign corresponding URI that uniquely identifies the namespace. For the above example we can define the namespace prefixes to avoid naming collisions.

```
 <?xml version=”1.0”>
<college xmlns:staff=”web technologies:staffInfo”
        xmlns:student=”web technologies:studentInfo”>
      <staff>
            < staff:name>aaa</ staff:name>
            < staff:dept>CSE</ staff:dept>
      </staff>
      <student>
            < student:name>bbb</ student:name>
            < student:dept>CSE</ student:dept>
      </student>
</college>
```

XML Schema:

An XML schema describes the structure of an XML document. XML schema is an XML-based alternative to DTD.

Limitations of DTD:

1. DTD's do not have the angled bracket syntax.
2. We cannot use multiple DTD's to validate one XML document.
3. DTD's have very limited basic types.
4. DTD's are not object oriented.

An XML schema language is also referred to as "XML Schema Definition(XSD)". An XML schema begins by declaring the XML schema namespace as follows:

```
<xsd:schema xmlns:xsd=http://www.w3.org/2001/XMLSchema>
```

Defining an Element:

There are two types of elements defined in an XML schema. They are:

1. Simple elements.
2. Complex elements.

1. Simple Elements:

Simple elements can contain only data and they cannot have sub elements or attributes. The text they contain can be of various data types. The format for defining a simple element is:

```
<xsd:element name=”element_name” type=”xsd:type”/>
```

2. Complex Elements:

Complex elements can contain sub elements or attributes. The complex elements are made up of simple elements. The format of defining a complex element is:

```
<xsd:complexType name=”element_name”>
        <xsd:sequence>
                <xsd:element name=”element_name” type=”xsd:type”/>
                <xsd:element name=”element_name” type=”xsd:type”/>
        </xsd:sequence>
</xsd:complexType>
```

Defining an Attribute:

An attribute provides additional information used by an element besides the element's content. The format for defining an attribute in XML schema is:

```
<xsd:complexType name=”element_name”>
        <xsd:complexContent>
              <xsd:extension>
                    <xsd:attribute name=”attribute_name” type=”xsd:type”/>
              </xsd:extension>
        </xsd:complexContent>
</xsd:complexType>
```

Example:

```
<xsd:complexType name=”employee”>
        <xsd:complexContent>
              <xsd:extension>
                    <xsd:attribute name=”gender” type=”xsd:string”/>
              </xsd:extension>
        </xsd:complexContent>
</xsd:complexType>
```

An XML schema is a file that we can create using any text editor. The file must be saved with .xsd extension and the contents of the file must confirm to the XML schema language.

Referencing an XML schema:

We link an XML schema by referencing the XML schema file in the <root> tag of the XML document. The format of <root> is:

<root xmlns:xsi=http://www.w3.org/2000.10.XMLSchema-instance xsi:schemalocation=”URI”>

Example of XML schema:

St.xsd:

```
<?xml version=”1.0”?>
      <xsd:schema xmlns:xsd=http://www.w3.org/2001/XMLSchema>
```

```
                <xsd:complexType name="student">
                        <xsd:sequence>
                                <xsd:complexType name="studentType">
                                        <xsd:sequence>
                                        <xsd:element name="rollnumber" type="xsd:string"/>
                                        <xsd:element name="name" type="xsd:string"/>
                                        <xsd:element name="marks" type="xsd:integer"/>
                                        <xsd:element name="result" type="xsd:string"/>
                                        </xsd:sequence>
                                </xsd:complexType>
                        </xsd:sequence>
        </xsd:complexType>
</xsd:schema>
```

studentInfo.xml:

```
<?xml version="1.0"?>
<root xmlns:xsi=http://www.w3.org/2000.10.XMLSchema-instance xsi:schemalocation="st.xsd">
<student>
        <studentType>
                <rollno>95A0</rollno>
                <name>aaa</name>
                <marks>500</marks>
                <result>Firstclass</result>
        </studentType>
        <studentType>
                <rollno>95A1</rollno>
                <name>bbb</name>
                <marks>450</marks>
                <result>Firstclass</result>
        </studentType>
</student>
```

Presenting XML:

In order to present the data in XML documents we use web presentation technologies. Presenting data is again dependent to a specific technology. It is better to separate the data from the presentation technology and stored in XML document and the extensible stylesheet language (XSL) is used to present the data.

Extensible stylesheet language (XSL):

The extensible stylesheet is a language used to express stylesheets which are then used to present XML documents. XSL stylesheets are not same as HTML stylesheets. Rather than creating a style for a particular XML element or class of elements with XSL, a template is created. This template is used to format XML elements which match a specific pattern.

The XSL transformation language (XSLT) is used to transform one XML document from one form to another. The result of applying XSLT to an XML document could be another XML, HTML text or any other document.

Example:

People.xml:

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="people.xsl"?>
<people>
        <person born="1993">
                <name>
                        <firstname>aaa</firstname>
                        <lastname>bbb</lastname>
                </name>
                <profession>Software Engineer</profession>
        </person>
        <person born="1982">
                <name>
                        <firstname>ccc</firstname>
```

```
                    <lastname>ddd</lastname>
                </name>
                <profession>Project Manager</profession>
        </person>
</people>
```

People.xsl:

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl=http://www.w3.org/1999/XSL/Transform>
<xsl:output method="html" omit-xml-declaration="no"/>
<xsl:template match="/">
<html>
        <head>
                <title>XSLT</title>
        </head>
        <body>
                <table border="2">
                        <tr>
                                <th>Born</th>
                                <th>FirstName</th>
                                <th>LastName</th>
                                <th>Profession</th>
                        </tr>
                        <xsl:for-each select="people/person">
                        <tr>
                                <td><xsl:value-of select="@born"/></td>
                                <td><xsl:value-of select="name/firstname"/></td>
                                <td><xsl:value-of select="name/lastname"/></td>
                                <td><xsl:value-of select="profession"/></td>
                        </tr>
                        </xsl:for-each>
                </table>
        </body>
</html>
</xsl:template>
</xsl:stylesheet>
```

The XSL stylesheet begins with a tag called stylesheet which binds with namespace prefix xsl to the URI http://www.w3.org/1999/XSL/Transform which uniquely identifies the XSL namespace.

The <xsl:output> is used to write a XHTML document type declaration to the result tree. Attribute omit-xml-declaration specifies whether the transformation should write the XML declaration to the result tree. <xsl:template> element describes how to transform a particular node from the source tree into the result tree. The match is used to select the document root of the source document.

The element <xsl:for-each> is used to iterate through the source XML document and search for person element. The attribute select specifies the node on which the <xsl:for-each> operates. The element <xsl:value-of>is used to retrieve value of the attribute and elements. The @ symbol specifies that we need to retrieve an attributes value.

Document Object Model (DOM):

There are two models that are commonly used for XML parsers:

1. SAX (Simple API for XML).
2. DOM (Document Object Model).

SAX parser is mainly used when we are dealing with streams of data i.e., the data the XML is passing from one place to another with parser acting as an intermediate way point. Typically this model is used when passing XML data across a network between applications and is widely used by java programmers. SAX based parsers does not have to build any static models of the document in memory and are intended to run quickly.

The SAX parser is not suitable to use in websites where repeated querying and updating of XML document is required. Here it is sensible to build some sort of representation which can be held in memory for the duration of the use of application. In such cases the DOM based parser is a better choice.

DOM stands for Document Object Model and is an API for XML documents. Basically an API is a set of data items and operations which can be used by the developers of application programs. For example, the Microsoft windows environment has a very rich API which is used by the developers when creating windows programs.

The DOM API specifies the logical structure of XML documents and the ways in which they can be accessed and manipulated. The DOM API is just a specification. There isn't a single reference piece of software associated with it which everyone must use. DOM complaint applications include all of the functionality that is needed to handle XML documents. They can build static documents, navigate and search through them, add new elements, delete elements and modify the content of existing elements.

DOM views XML document as a tree, but this is very much logical view of the document. Each XML element is modelled as an object. This means that the node encompass both data and behaviour and that the whole document can be seen as a single complex object. Object oriented theory lets each object have a unique identity. If each node has a unique identity then the tree can be searched for individual nodes. DOM exposes the whole document to application so that the application can manipulate individual nodes.

Accessing XML data using DOM object:

School.xml:

```
<?xml version="1.0"?>
<school>
<class>
<title>XML</title>
<students>
<student>
<first_name>aaa</first_name>
<last_name>bbb</last_name>
</student>
<student>
<first_name>aaa</first_name>
<last_name>bbb</last_name>
```

```
</student>
</students>
</class>
</school>
```

School.html:

```
<html>
<head>
<title>Accessing XML data</title>
<script type="text/javascript">
function getStudentData()
{
var xmldoc;
xmldoc=new ActiveXObject("Microsoft.XMLDOM");
xmldoc.load("school.xml");
nodeSchool=xmldoc.documentElement;
nodeClass=nodeSchool.firstChild;
nodeStudents=nodeClass.lastChild;
nodeStudent=nodeStudents.lastChild;
nodeFirstname=nodeStudent.firstChild;
nodeLastname=nodeFirstname.nextSibling;
outputMessage="Name:"+nodeFirstname.firstChild.nodeValue+'
'+nodeLastname.firstChild.nodeValue;
message.innerHTML=outputMessage;
}
</script>
</head>
<body bgcolor="pink">
<center>
<h1>Accessing XML Data</h1>
<div id="message"></div>
```

```
<input type="button" value="GET DATA" onClick="getStudentData()">
</center>
</body>
</html>
```

Using XML processors: DOM & SAX:

Parsing is a process of reading and validating a program written in one format and converting it to the desired format. XML documents are organized as hierarchal structure similar to a tree. They are well-formed and valid documents. Thus if we have something equivalent to a compiler for XML that can read, validate and convert it. So we can make use of parser for XML documents.

The program that do the process of reading, validating and convert an XML document into the desired format is called as a parser or processor. The parser has to access the XML file and bring it into the memory. Then the parser converts the file into an object, where the object is accessed by an application program.

When an XML document is presented to a program as an object there are two possibilities:

a) Present the document in bits and pieces as we encounter a certain selection or portions of the document. This approach where it goes through the XML document item by item till the end of XML file is called “Simple API for XML (SAX)”.
b) To present the entire document tree at once where the program has access to the XML document as a single object. This approach of presenting entire document as a single object is done by “Document Object Model (DOM)”.

UNIT – IV
WEB SERVERS & SERVLETS
JSP APPLICATION DEVELOPMENT

Introduction:

Java Servlet is a server side program that is called by the user interface or another J2EE in component and contains the business logic to process a request. The Java Servlet can call other components such as EJB that handle processing.

J2EE applications use a light weight client, typically a browser that contains little (or) no processing logic. Consider a request for a static web page. A user enters a Uniform Resource Locator (URL) into a browser. The browser generates an HTTP request to the appropriate Web server. The web server maps this request to a specific file. That file is returned in an HTTP response to the browser. The HTTP header in the response indicates the type of the content. The Multi Purpose Internet Mail Extension (MIME) are used for this purpose. For example, ordinary ASCII text has a MIME type of text/plain. The Hypertext Markup Language source code of a web page has a MIME type of text/HTML.

Now consider the dynamic content. Assuming that an online store uses a database to store information about business, This would include items for sale, prices, availability, orders and so on. It wishes to make this information accessible to customer via web pages. The contents of those web pages must be dynamically generated in order to reflect the largest information in the database.
In the early days of web, a server could be dynamic construct a page by creating a separate process to handle each client request. The process would open connecting to one or more database in order to obtain the necessary information. It communicated within the web server buyer and interface known as Common Gateway Interface(CGI). CGI allowed the separate process to read data from the HTTP request write the data to the HTTP response. A variety of different languages were used to build CGI programs. These include C, C++, Perl etc. However, CGI suffered serious performance problems. It was expensive in terms of processor and memory resources to create a separate process for each client request. It was also expensive to open and close database connections for each client request. In addition, CGI programs were not platform - independent. Therefore, other techniques were introduced, Among these are "Servlets".

Benefits of Servlets over CGI:

Service offer several advantages in comparison with CGI:

- The performance is significantly better. Servlets execute within the address space of a web server. It is not necessary to create a separate process to handle each client request.
- Servlets are Platform - independent because they are written in Java. A number of web servers from different vendors offer the servlet API. Programs developed for this API can be moved to any of these environment without recompilation.
- The Java security manager on the server enforces a set of restrictions to protect the resources on a server machine.
- The full functionality of the Java class libraries is available to a servlet. It can communicate with applets, databases (or) other software via the sources and RMI mechanisms.

The Life Cycle of a Servlet:

1. init()
2. service()
3. destroy()

They are implemented by every servlet and invoked at specific times by the server. Assuming that a user enters a URL to a web browser. The browser then generates an HTTP request for this ?URL. This request is then sent to the appropriate server. This HTTP request is received by the web server. The server maps this request to a particular sevlet. The servlet is dynamically retrived and loaded into the address space of the server. The server invokes “init()” method of the servlet. This method is invoked only when the servlet is first loaded in to the memory. It is possible to pass the initialization parameters to the servlet. So, it may configure itself.

The server invokes the “service()” method of the servlet. This method is called to process the HTTP request. It is possible to the servlet to read the data that has been provided in the HTTP request. It may also formulate an HTTP response for the-client.

The servlet remaining in the address space of the server and is available to process any other HTTP request received from clients. The service() method is called for each HTTP request.

The server may decide to unload the servlet from its memory. The algorithms by which this determination is made or specific to each server. The server calls “destroy()” to relinquish any resources as file handles that are allocated for the servlet. Important data may be stored to a persistent store. The memory allocated for the servlet and its objects can then be garbage collected.

A Simple Servlet:

To create and test a simple servlet we have to follow the following steps:

1. Create a project folder hierarchy as follows:

```
P1
 |
WEB-INF
|
Classes
```

2. Open notepad and write servlet class as follows:

Source Code:

```
import java.io.*;
import javax.servlet.*;
public class MyServlet extends GenericServlet{
public void service(ServletRequest request, ServletResponse response) throws ServletException,
IOException{
response.setContentType("text/html");
PrintWriter pw = response.getWriter();
pw.println("<html><head><title>This is My Servlet </title>");
pw.println("</head><body><h1>Hi This message came from my server</h1>");
pw.println("</body></html>");
plw.close();
}
}
```

Above code imports "javax.servlet" package. This package contains the classes and interfaces required to build servlets. The program defines "MyServlet" as a subclass of "GenericServlet". The GenericServlet class provides functionality that makes it is easy to handle requests and responses. Inside MyServlet, "service()" is overridden. This method handles requests from a client. The first argument "ServletRequest" object that enables the servlet to read data that is provided via client requests. The second argument is a "ServletResponse" object that enables the servlet to formulate a response for the client.

The call to “setContentType()” method establishes the “MIME” type of the HTTP response. In this the MIME type is text/html that indicates that the browser should interpret the content as HTML data.

Now, the getWriter() method obtains the “PrintWriter()” anything written to this stream is sent to the client as part of a HTTP response. The println() method is used to write some simple HTML.

3. Save the above file with the name MyServlet.java.
4. Set the classpath to servlet-api.jar file by the following command

```
  setClassPath= %CLASSPATH%;D:/tomcat6.0/lib/servlet-api.jar ;
                              (or)
setClassPath= %CLASSPATH%;D:/tomcat5.5/commom/lib/servlet-api.jar;
```

1. Compile the servlet by the following command

```
javac MyServlet.java
```

2. Copy te MyServlet.class file in the P1/WEB-INF/classes folder
3. Open notepad and write the description of te servlet as shown below:

```
<web-app>
<servlet>
<servlet-name>abc</servlet-name>
<sservlet-class>MyServlet</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>abc</servlet-name>
<url-pattern>/hello</url-pattern>
</servlet-mapping>
</web-app>
```

4. Save this file with the name web.xml in P1/WEB-INF folder.
5. Copy the project folder P! in d:/tomcat6.0/webapps folder.
6. Start tomcat server. The application P1 is deployed and running.
7. Open web browser window.
8. Type the following in address field of the browser window:

http://localhost:8080/P1/hello

Where P1=name of the application

Hello= url-pattern of MyServlet

9. Servlet executes one generates the response to clients as:

10. Hi, this message came from MyServlet in client's browser window.

The Servlet API:

Two packages contain the classes and interfaces that are required to build servlets. These are:

1. Javax.servlet
2. Javax.servlet.http

The above two packages constitute the servlet API. These two packages are not part of the java core packages. Instead they are standard extensions. Therefore, they are not included in the Java Software Development Kit. We must download tomcat to obtain their functionality.

The javax.servlet Package:

The javax.servlet package contains a number of interfaces and classes that establish the framework in which the servlets operate.

List of Interfaces:

Interface	**Description**
Servlet	Declares lifecycle methods for a servlet
ServletConfig	Allows servlet to get initialization parameters.
ServletContext	Enables servlet to events and access information about their environment
ServletRequest	Used to read data from a client request
ServletResponse	Used to write data to a client response
SingleThreadModel	Indicates that the Servlet is thread safe

1. The Servlet Interface:

All Servlets must implement the "Servlet" interface. It declares the init(), service and destroy() methods that are called by server during the life cycle of servlet. A method is also provided that allows a servlet to obtain any initialization parameters. Various methods designed by the servlet interfaces are:

Method	Description
void destroy()	Called when the servlet is unloaded
ServletConfig getServletConfig()	Returns a ServletConfig object that contains any initialization parameters
String getServletInfo()	Returns a string describing the servlet
void init(ServletConfig sc) throws ServletException	Called when the servlet is initialized. Initialization parmeters for the servlet can be obtained from sc. An unavilable Exception should be thrown if the servlets cannot be initialized.
void service(ServletRequest req, ServletResponse res) throws ServletException, IoException	Called to process a request from a client .The request from req and the response to the client can be written res. An exception is generated if a servlet (or) Io problem occurs.

The init(), servlet() and destroy() methods are the life cycle methods of the server. These are invoked by the server the getServeltConfig() method is called by the servlet to obtain imitialize paramaters. A servlet devloper overrides getServletInfo() method to provide a string useful information.

2. The ServletConfig Interface:

The servletConfig interface is implemented by the server. It allows a servlet to obtain configuration data when it is loaded. The methods declared by this interface are:

Method	Description
servletContext getServletContext()	Returns the context of the servlet.
string getInitParameter(String param)	Returns the value of the initialization parameter named param.
string getServletParameterName()	Returns an enumeration of all initialization parameter names.
string getServletName()	Returns the name of the invoking servlet.

3. The ServletContext Interface:

The servletContext interface is implemented by the server it enables servlet to obtain information about their enviroment several methods designed by this interface are:

Methods	Description
string getiIMEType(String file)	Returns the MIME type of file
string getServletInfo()	Returns information about the server.
void log(string s)	Writes to the servlet log
void log(string s, throwable e)	Write s and stack trace for e to the servlet log

The ServletRequest Interface:

The ServletRequest Interface is implemented by the server. It enables a servlet to obtain the information a client request. The various methods are:

Method	Description
Int getContentLength()	Returns the size of the request. The value -1 is returned if the size is unavailable.
String getContentType()	Returns the type of the request. A null value is returned if the type cannot be determined.
ServletInputStream getInputStream() throws IOException	Returns a servlet that can be used to read binary data from the request. An IllegalStateException is thrown if getReader has already been invoked for this request
String getCharacterEncoding()	Returns the character encoding of the request
String getParameter(string pname)	Returns the value of the parameter named pname
Enumeration getParameterNames()	Returns an enumeration of the parameter names for this request.
String[] getParameterValues(string name)	Returns an array containing values associated with the parameter specify by name

4. The ServletResponse Interface:

The ServletResponse Interface is implemented by the server. It enables a server to formulate a response for a client. Several methods defined by this interface are:

Method	Description
String getCharacterEncoding()	Returning the character encoding for the response
ServletOutputStream getOutputStream() throws IOException	Returns a ServletOutputStream that can be used to write binary data to a response. An IlleagalStateException is thrown if getWriter() has already been invoked for this request
PrintWriter getWriter() throws IOException	Returns a PrintWriter that can be used to write character data to the response. An IllegalStateException is thrown if getOutputStream() has already been invoked for this request
Void SetContentLength(int size)	Sets the content length for the response to size
Void SetContentType(string type)	Sets the content type for the response to type.

The various core classes that are provided in the javax.servlet Package are:

Class	Description
GenericServlet	Implement the servlet and servletConfig Interfaces.
ServletInputStream	Provides an input stram for reading request from a client.
ServletOutputStream	Provides an outputStream for writing responses to a client.
ServletException	Indicates a servlet error has occures.
Unavailable Exception	Indicates a servlet in unavailable.

The SingleThread Model Interface:

This interface is used to indicate that only a single thread will execute the servlet() metod of a servlet at a given time. It defines no constants and declares no methods.

If a servlet implements this interface, the server has two options:

1. It can create servlet instances of the servlet when a client request arrives it is dent to an available

instance of the servlet.

2. It can synchronize access to the servlet.

Javax.servlet classes:

1. The GenericServlet Class:

The GenericServlet class provides implementations of the basic life cycle methods of a servlet and is typically subclass by servlet developers. GenericServlet implements "servlet" and ServletConfig interfaces. In addition, a method to append a String to the server log file is available.the signatures of this method are shown as:

- Void log(String s)
- Void log(String s)

Here, s is the strong to be appended to the log , and e is the exception that occured.

2. The ServletInputStream class:

The ServletInputStream class extends InputStream. It is implemented by the server and provides an input stream that a servlet developer can use to read the data from a client request .It defines the default constructor. In addition, a method is provided to read bytes from the stream. Its syntax is:
int readLine(byte[] buffer,int offset, int size) throws IOException
Here, buffer is the array into which size bytes are placed starting at offfset. The method returns the actual number of bytes read (or) -1 if an end-of-stram condition is encountered.

1. The ServletOutputStream class:

The ServletOutputStream class extends OutputSream. It is implemented by the server and provides on OutputStream, that a developer can use to write data to a client response. A default constructor is defined. It also defines print() and println() methods, which output data to the stream.

2. The ServletException class:

Javax.Servlet defines two exceptions. The first is ServletException, which indicates a servlet problem as occurred. The second is UnavailableException, which extends ServletException and indicates tha a servlet is unavailable.

<u>The javax.Servlet.http package:</u>

The javax.Servlet.http package contains a number of interfaces and classes that are commonly used by servlet developers. Various interfaces defined by the javax.Servlet.http are:

Interface	Description
HttpServletRequest	Enables Servlet to read data from a HTTP Request
HttpServletResponse	Enables Servlet to write data to an Http Response
HttpSession	Allows session data to be read or written
HttpSessionBindingListener	Informs an object that is bound to (or) unbound from a session.

1. The HttpServerRequest Interface:

The HttpServletRequest interface is implemented by the server and it enables a servlet to obtain information about a client request. Several methods defined by this interface are:

Method	Description
Cookie[] getCookies()	Returns an array of cookies in this request
Long getDateHeader(String field)	Returns the value of the date header field named field
String getHeader(string field)	Returns the value of the header field named field
Enumeration getHeaderNames()	Returns an enumeration of the header names
String getQueryString()	Returns any query string in the URL
StringBuffer getRequestURL()	Returns the URL
HttpSession getSession()	Returns session for this request. If a session does not exist, one is created and then returned.

2. The HttpServletResponse interface:

The HttpServletResponse interface is implemented by the server. It enables a servlet to formulate an Http Response to a client. Several constraints are defined. These correspond to the different status codes that can be assigned to an Http Response. Several methods of this interface are:

Methods and it's Description:

- void addCookie(Cookie cookie) : adds cookie to the HTTPresponse
- String encodeURL(String url) : determines if the session ID must be encoded in the URL identified as URL. If so it returns the modified version of URL. Otherwise returns URL. All URL's generated by a servlet should be processed by this method
- void sendError(int c) throws IOExecption : sends the error code c to the client.
- void sendError(int c, String S) throws IOExecption : sends the error code c and message S to the client.

- void sendRedirect(String url) throws IOExecption : redirects the client to url.
- void sendHeader(String field, String value) : adds field to the header with value equal to value.
- void setState(int code) : sets the states code for this response to code.

3. HttpSession interface :

The HttpSession interface is implemented by the server and it enables a servlet to read and write the state information that is associate with an HttpSession. Various methods defined by this interface are

Methods :

Object getAttribute(String attr) : returns the value associated with the name passed in attr. Returns null if attr is not found.

Enumeration getAttributeNames() : returns an enumeration of the attribute names associated with the session.

long getCreationTime() : returns the time (in milliseconds) when this session was created.

long getLastAccessedTime() : returns the time (int milliseconds) when the user last made a request for this session.

void invalidate() : invalidates this session and removes it from the context.

3. HttpSessionBindingListener interface :

The HttpSessionBindingListener is an interface extending from base interface java.util.EventListener interface. This interface will notify an object when it is bound to or unbound from a session.

Methods :

Void valueBound(HttpSessionBindingEvent e)

Void valueUnbound(HttpSessionBindingEvent e)

Here, e is the event object that describes the binding various classes that are provided by javax.servlet.httpPacakge are

Class & Their Description:

Cookie : allows state information to be stored on a client machine

HttpServlet : provides methods to handle Http requests and responses

HttpSessionEvent : encapsulates a session changed event

HttpSessionBindingEvent : indicates when a listener is bound or unbound from a session value or that a session attributes

1) The Cookie Class:

The Cookie class encapsulates a cookie. A cookie contains state information sent by a servlet to web browser, and is saved by th e browser and stored on a client.

It later is sent back to server. Cookies are valuable for tracking user's activities. For example, assuming that a user visits an online store. A cookie can save the user's name, address and other information. The user does not need to enter this data each time he (or) she visits the store.

A servlet can write a cookie in to a user's machine via the addcookie() method of the HttpServletResponse interface, The data for that cookie is then included in the header of the Http response that is sent to the browser.

The names and values of cookies are stored on the user's machine. Some of the information that is saved for each cookie includes the following:

- The name of the cookie
- The value of the cookie
- The expression date of the cookie
- The domain and path of the cookie

The expiration date determines when this cookie is deleted from the user's machine. If an expiration date is not explicitly assigned to a cookie, it is
deleted when the current browser Session ends. Otherwise, the cookie is saved in a file on the user's machine.

The domain and path of the cookie determine when it is included in the header of the HTTP request. If the user enters a URL whose domain and path match
these values, the cookie is then supplied to web server. Otherwise it is not. There is one constructor defined for cookie. It has signature:

Cookie(string name, string value)

Here, name and value of the cookie are supplied as arguments to the constructor. Various other methods defined by cookie class are:

Method	**Description**
string getDomain()	Return the domain
int getMaxAge()	Return the age(in seconds)

string getName()	Return the name
string getPath()	Return the path
string getValue()	Return the value
void setDomain(string d)	Sets the domain to d
void setMaxAge(int secs)	Sets the maximum age of the cookie to secs. This is the number of seconds after which the cookie is deleted.passing -1 causes the cookie to be removed when the browser is terminuted
void setValue(string v)	sets the value to v.
void setpath(string p)	sets the path to p

2) The HttpServlet class:

The HttpServlet class extends GenericServlet, It is commonly used when developing servlets that receive and process HTTP requests.The methods of HttpServlet can be summarized as:

Method	Description
void doGet(HttpServletRequest req, HttpServletResponse res)throws IOException, ServletException	Performs on HTTP GET
void doPost(HttpServletRequest req, HttpServletResponse res)throws IOException, ServletException	Performs on HTTP POST
void service(HttpServletRequest req, HttpServletResponse res)throws IOException, ServletException	Called by the server, When an HTTP request arrives for the servlet. The arguments provide access to HTTP request and response

3) The HttPSessionEvent Class:

The HttpSessionEvent Class encapsulates session events.It extends EventObject and is generated when a change occurs to the session.It defines a constructor. HttpSessionEvent(HttpSession session):- Here,Session is the source of the event.

HttPSessionEvent defines one method:

HttPSession getSession():- It returns the session in which the event is occurred.

4) The HttpSessionBindingEvent Class:

The HttpSessionBindingEvent Class extends The HttpSessionEvent Class.It is generated when a listener is bound (or) unbound from a value in a HttpSession object.It is also generated when an attribute is bound (or) unbound.

<u>The constructors defined are:</u>

HttPSessionBindingEvent(HttPSession session String name)

HttPSessionBindingEvent(HttPSession session String name,Object val)

Here,Session is the source of the event and name is the name associated with the object that is being bound (or) unbound.If an attribute is being bound (or) unbound,its value is passed in val.

<u>The various other methods are:</u>

String getName():

This method obtains the name that is being bound (or) unbound.

HttpSession getSession():

Obtains the session to which the listeneris bound (or) unbound.

Object getValue():

This method obtains the value of the attribute that is being bound (or) unbound.

Reading Servlet Parameters

The ServletRequest includes methods that allow us to read the names and values of parameters that are included in a client request. In order to illustrate this consider the below example. The example contains two files PostParameter.html and PostParameterServlet.java. The PostParameter.html defines two labels and two text fields and a submit button.

<u>PostParameters.html:</u>

```
<html>
        <head>
                <title>Reading Servlet Parameters</title>
        </head>
        <body>
                <center>
```

```
<form name=”form1” method=”post”      action=http://localhost:8080/p1/PostParamservlet>
                    <table>
                            <tr>
                            <td><b>Employee</td>
                             <td><input type=”text” name=”ename” size=”25”      value=””></td>
                            </tr>
                            <tr>
<td><b>phone</td>
          <td><input type=”text” name=”phone” size=”25”       value=””></td>
                            </tr>
                            <tr>
                                      <td><input type=”submit” value=”submit”></td>
                                      <td><input type=”reset” value=”reset”></td>
                            </tr>
                   </table>
          </body>
</html>
```

The source code for PostParametersServlet.java can be shown as:

PostParametersServlet.java:

```
import java.io.*  ;
import javax.Servlet.*;
import java.util.*;
public class PostParametersServlet extends GenericServlet{
          public void Service(ServletRequest req, ServletResponse res) throws
ServletException,IOException {
          PrintWriter pw=res.getwriter();
          Enumeration e=req.getParameterNames();
          While(e.hasMoreElements()){
                            String pname=(string)e.nextElement();
pw.println(pname+"=");
                        String pvalue=req.getParameter(pname);
                        Pw.println();
                }
                pw.close();
        }
}
```

The getParameterNames() method return an enumeration of the parameter names. The parameter value is obtained through the getParameter() method. The output will be the parameter name and value are displayed to the client.

Place the html file in P1 folder and .class file in classes folder and web.xml in WEB-INF folder and follow the following steps:

1. Start Tomcat
2. Display the web page in the browser
3. Enter employee name and phone number in the text fields
4. Submit the web page

The browser will display the response that is dynamically generated by the servlets.

Reading Initialization Parameters:

The Initialization Parameters are supplied by server developer in the development descriptor i.e web.xml. The Initialization Parameters are given to a server as follows:

```
<init-parameter>
<param-name>name</param-name>
<param-value>value</param-value>
</init-parameter>
```

The servlet container reads the parameters and supplies them to the servlet using servletConf8g object. We call the getParameter() method on the ServletConfig object in the Servlet to receive the parameter values .

For ex: init() method can be implemented as follows.

```
public void init(ServletConfig con) throws ServletException
{
        String value=con.getInitParameter("name");
```

}

Whatever name is given to the parameter in the deployment descriptor file that name should be supplied to the getInitParameter() .

It receives the value specified the deployment descriptor file as as string object .

We can supply any number of Initialization Parameters to the Servlet.

Installing the Java Software development kit(JSDK):

1. Download the Java development kit from the sun's website https://java.sun.com/
2. Double click on the downloaded jdk installer to start JDK and now we could see the window as license agreement. Change the 4adio button option for selecting accepting terms and conditions.
3. Now just click on the next button then cases for the description folder where we go to install the Java. By default it 8s installed in c drive
   C:/Program Files/java/jdk1.5.0_01
4. Now click the next button and the installation progresses that will take few minutes.

After the installation of Java we need to set the environment variables. The steps are:

1. Go to desktop find MyComputer icon and right click on it
2. From the pop up menu select properties option
3. Click on the advanced which appears on the top right corner.
4. Click on environment variables button and we could see a window which contains the user variables and system variables.
5. Now click on the new variable button in the user variable option to create the PATH variable
   Variable Name: PATH
   Variable Value:C:\Program files\Java\jdk1.5.0.

Then click on OK button.

6. Now again click on the new variable button in the user variable option to set the JAVA_HOME Variable as:
   Variable Name: JAVA_HOME
   Variable Value:C:\Programfiles\Java\jdk1.5.0
7. Click on ok button.

Handling HTTP Requests and Responses:

The HttpServlet class provides specialized methods that handles the various types of Http requests. A servlet developer typically overrides one of these methods. However, the GET and POST methods are commonly used when handling from input.

Handling HTTP GET Requests and Responses: In GET method the servlet container must write the headers before committing the response, because in Http the header must be sent before the response body. If the response is incorrectly formatted, do Get returns an Http “Bad Request” message.

Example:

Here we will develop a servlet that handles an HTTP GET request. The Servlet is invoked when a form on a web page is submitted. The example contains two files. A web page is defined and a servlet is defined.

ColorGet.html:

```
<html>
<head>
        <title>Handling GET Request</title>
</head>
<body>
<center>
<form name=”form1” method=”GET” action= http://localhost:8080/p1/ColorGetServlet>
<b>Color:
<select name=”Color” size=”1”>
        <option value=”red”>Red</option>
          <option value=”Green”>Green</option>
          <option value=”Blue”>Blue</option>
</select><br><br>
<input type=”submit” value=”submit”>
</form>
</center>
</body>
</html>
```

The above html defines a select element and a submit button .The action of the form tag specifies a URL that identifies a servlet to process the HTTP GET request.

ColorGetServlet.java:

```
Import java.io.*;
Import javax.servlet.*;
Import javax.servlet.http.*;
Public class ColorGetServlet extends HttpServlet{
        Public void doGet(HttpServletRequest req,HttpServletResponse res) throws
ServletException, IoException{
        String color=req.getParameter("color");
        Res.setContentType("text/html");
        PrintWriter pw= res.getWriter();
        Pw.println("<b> The Selected color is:");
        Pw.println(color);
        Pw.close();
}
}
```

Compile the servlet and perform these steps to test this example:

1. Start Tomcat.
2. Display web page in the browser.
3. Select a color.
4. Submit the web page.

After Completing these steps ,the browser will display the response that is dynamically generated by the servlet. The Parameters for an HTTP GET request are included as part of URL. That is sent to the web server. For example, assuming that the user selects red option and submits the form. The URL sent from the browser to the server is : http://localhost:8080/p1/ColorGetServlet?color=Red. The Characters to the right of the question mark are known as "Query String" .

Handling HTTP POST Request and Responses:

The HTTP POST method allows the client to send data of unlimited length to the webserver at a time .The servlet container must write the headers before committing the responses, because in HTTP the header must be sent before the response body. If the HTTP POST Request is incorrectly formatted,doPost returns an HTTP "Bad Request" message.

Example:

Here we will devlop a servlet that handles an HTTP POST request. The Servlet is invoked when a form on a web page is submitted. The example contains two files. A webpage is defined and a servlet is defined.

ColorPost.html:

```
<html>
<head>
<title>Handling post req</title>
</head>
<body>
<center>
 <form name=”form1” method=”POST” action=http://localhost:8080/p1/ColorPostServlet>
<b>Color:
<select name=”color” size=”1”>
<option value=”red”> Red</option>
<option value=”Blue”>Blue</option>
</select><br><br>
<input type=”submit” value=”submit”>
</form>
</center>
</body>
</html>
```

ColorPostServlet.java:

```
Import java.io.*;
Import javax.servlet.*;
Import javax.servlet.http.*;
Public class ColorPostServlet extends HttpServlet{
Public void doPost(HttpServletRequest req,HttpServletResponse res) throws
ServletException,IOException{
String color=req.getParameter(“color”);
Res.setContentType(“text/html”);
PrintWriter pw=res.getWriter();
pw.println(“<b> the selected color is:”);
pw.println(color);
pw.close();
}
}
```

Parameters for an HTTP POST request are not included as part of URL that is sent to web server URL sent will be: http://localhost:8080/p1/ColorPostServlet.

Handling Cookies:

AddCookie.html:-

```
<html>
    <head>
        <title> Cookies demo </title>
    </head>
    <body>
        <center>
<form name="form1" method="Post"
action="http://localhost:8080/P1/AddCookieServlet">
        <b>Enter a value for my cookie:
        <input type="text" name="data" size=25 value=" ">
        <br><input type=submit value="submit">
        </form>
    </center>
  </body>
</html>
```

The above page contains a text field in which a value can be entered. There is also a submit button on the page when this button is pressed, the value in the text field is sent to AddCookieServlet via an HTTP POST request.

AddCookieServlet.Java:-

```
import java.io.*;
import javax.Servlet.*;
import javax.Servlet.http.*;
```

```
public class AddCookieServlet extends HttpServlet
{
    Public void doPost (HttpServletRequest req, HttpServletResponse res)throws ServletException,
IOException
   {
         String data=req.getParameter("data");
          Cookie cookie=new Cookie("MyCookie",data);
           res.addCookie(cookie);
           PrintWriter pw=res.getWriter();
           pw.println("<br>MyCookie has been sent to");
           pw.println(data);
           pw.close();
   }
}
```

Above code gets the value of parameter name data. It then creates cookie object that has the name "MyCookie" and contains the value of the "data" parameter. The cookie is then added to the header of the HTTP response via the addCookie() method. A feedback message is then written to the browser.

GetCookiesServlet.java:-

```
import java.io.*;
import javax.Servlet.*;
import javax.Servlet.http.*;
public class GetCookieServlet extends HttpServlet
{
    Public void doGet (HttpServletRequest req, HttpServletResponse res)throws ServletException,
IOException
   {
          Cookie[] cookies=req.getCookies();
          res.setContentType("text/html");
          PrintWriter pw=res.gerWriter();
```

```
            pw.println("<b>");
            for(int i=0;i<cookies.length;i++)
            {
                    String name=cookies[i].getName();
                     String value=cookies[i].getValue();
                     pw.println("name="+name+";value="+value);
            }
            pw.close();
    }
}
```

Compile the servlet and perform these steps:-

1. Start Tomcat
2. Display AddCookie.html in the bottom
3. Enter a value for my cookie
4. Submit the webpage

   After completing these steps we will observe that a feedback message is displayed by the browser.

Next request the following URL via browser. http://localhost:8000/p1/GetCookiesServlet

The name and value of the cookie will be displayed in the browser.

Session Tracking:

Http is a stateless protocol. Each request is independent of the previous one. However in some applications it is necessary to save state information so that information can be collected from several interactions between a browser and a server sessions provide such a mechanism.

A session can be defined as a series of related interactions between a single client and the web server, which take place over a period of time.

A session can be created via the getSession() method of HttpServerRequest. An HttpSession object is requested this object can store a set of bindings that associate names in objects the SetAttribute(), getAttribute(), getAttributeNames() and removeAttributes() methods of HttpSession manage these bindings. It is important to note that session state is shared among all the servlets that are associated with a particular client.

Example:

```
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class DateServlet extends HttpServlet{
 public void doGet(HttpServletRequest req,HttpServletResponse res)throws
ServletException,IOException{
  res.setContentType("text/html");
  HttpSession hs=req.getSession(true);
  PrintWriter pw=res.getWriter();
  pw.print("<b>");
  Date d=(Date)hs.getAttribute("date");
  If (d!=null){
   Pw.print("Last accessed date:"+d+"<br>");
  }
  date = new Date();
  hs.setAttribute("date",d);
  pw.print("current date:"+d);
  pw.close();
 }
```

JSP Application Development

Generating Dynamic Content:

JSP is all about generating dynamic content i.e. content that differs based on user input, time of day, the state of an external system or any other runtime conditions. JSP provides us with loss of tools for generating this content.

JSP pages should have the file extension. JSP which tells the server that the page needs to be processed by the JSP container without this extension, the server is unable to distinguish a JSP page from the other type of file and sends it unprocessed to the browser.

When working with the JSP's, we need a regular text editor such as notepad on windows. However, some Interactive Development Environment (IDE) includes a small web container that allows us to easily execute and debug page during development. There are also several webpage authoring tools that are often used when developing HTML pages that support JSP.

Example JSP that shows the current date and time:

```
<% page import =”java.util.Date” session=”false” %>
<html>
<head><title>Example JSP </title></head>
<body>
<h1>Hello, Welcome to JSP world </h1>
<% out.println(“<p><b>Hello to the JSP world”);
Out.println(“<p><b>Current Date is:”);%>
<%=new Date()%>
</body></html>
```

Using Scripting Element:

We can create Dynamic content by accessing Java programming language objects within scripting elements.

We can access a variety of objects, including enterprise beans and Java Bean components within a JSP page. JSP technology automatically makes some objects available, and we can also create and access application application-specific objects.

Implicit JSP Objects:

Implicit Objects are created by the web container and contain information related to a particular request, Page (or) application. Many of the objects are defined by the Java Server technology underlying JSP Technology.

Implicit objects are those objects which are available in JSP by default. Web container executes the JSP page as a servlet. Servlet operates in the request-response model. All objects that are associated with any servlet by default are part of this implicit object group. The following is a list of JSP implicit Objects:

1. Request
2. Response
3. Page content
4. Session
5. Application
6. Out
7. Config
8. Page
9. Exception

JSP implicit objects:

These implicit objects are created by the JSP engine during translation phase. They are being created inside service method, so we can directly use them within scriptlet without initializing ad declaring them. There are total 9 implicit objects available in JSP.

1. Out – Instance of the javax.servlet.JSP.JSPWriter
2. Request – javax.servlet.http.HttpServletRequest
3. Response – javax.servelet.http.HttpServletRequest
4. Session – javax.servlet.http.HttpSession
5. Application – javax.servlet.ServletContext
6. Exception – javax.servlet.JSP.JSPException
7. Page – java.lang.object
8. Page Context – javax.Sevlet.JSP.PageContext
9. Config – javax.servlet.Servletconfig

- **Out:**

This object is used for writing content to the client. It has several methods which can be used for properly formatting output message to the browser.

Differences between JSP writer and PrintWriter:

- ➔ Every JSP Writer is associated with 8kb of the internal buffer. Whereas PrintWriter doesn't associated with any buffer.
- ➔ The methods of the JSPWriter class are designed to throw java.io.IOException whereas methods of PrintWriter class doesn't throw any exception.
- ➔ JSPWriter is an abstract class present in the javax.servlet package.
- ➔ PrintWriter is a class defined in java.io package.

Example:

```
<html>
<body>
<% int a=10; Int b=20;
Int sum=a+b;
out.print("a value is:"+a);
out.print("b value is:"+b);
out.print("sum is:"+sum); %></body></html>
```

- **Request:**

The main purpose of request implicit object is to get data on a JSP page which has entered by the user on the previous JSP page.

Example:

```
<% string name=request.getParameter("vid");
out.print("Welcome"+name); %>
```

- **Response:**

It is basically used for modifying on dealing with the response which is being sent to the client after processing the request.

Example:

```
request.setContentType("text\html");
```

out.print(name);

- **Session:**

It is most frequently used implicit object, which is used for storing the user’s data to make it available on the other JSP pages till the uses session is awake.

Example:

```
<html><body>
<form action=”/welcome”>
<input type=”text” name=”uname”>
<input type=”submit” value=”submit”>
</form></body></html>
```

Welcome.jsp:

```
<html><body><% string name=request.getParameter(“uname”);
out.print(“Welcome:”+name);
session.setAttribute(“user”,name);
< a href=”showuserinfo.jsp”>Show underable </a> %>
</body>
</html>
```

Showuserinfo.jsp:

```
<html><body> <% string name=(string)session.getAttribute(“user”);
out.print(“Hello”+name);
%></body></html>
```

- **Application:**

This is used for getting application wide initialization processing and to maintain useful data across whole JSP application.

- **Exception:**

Exception implicit object is used in exception handling for exploring the error messages. This object is only available to the JSP pages which has ErrorPage set to true.

```
<% @page isErrorPage=”true” %>
<html><body><%= exception %></body></html>
```

- **Page:**

Page is implicit object used to represent the servlet instance i.e. converted servlet, generated during translation phase from a JSP page.

<% =page.getServletInfo() %>

- **Config:**

This is a Servlet configuration object and mainly used for accessing getting configuration information such as ServletContext, Servlet name,configuration parameters etc.

Conditional Processing:

Stud.html:

```
<html><body><h3>Conditional Processing </h3>
<form method=”get” action=”http://localhost:page/p1/con.jsp”>
<b>Enter Student Name:<input type=”text” name=”t1”><br>
<b>Enter Student Marks:<input type=”text” name=”t2”><br>
<input type=”submit” value=”Result”>
</form></body></html>
```

Con.jsp:

```
<html><body>
<%! String name;
String marks1;
Int marks;
%>
<b>Following is the Student Result
<%
name=request.getParameter(“t1”);
marks1=request.getParameter(“t2”);
marks=Integer.ParseInt(marks1);
if(marks>70)
out.println(“Distinction”);
else if(marks>60 && marks<70)
out.println(“First Class”);
else if(marks>50 && marks<60)
```

```
out.println("Second class");
else
out.println("Failed");
%>
<% out.println("<b>Student name:"+name);
out.println("<b>Student marks are:"+marks); %>
</body></html>
```

Declaring Variables are Methods:

Declaration tag is a block of java code for declaring class wide variables, methods and classes.

<%! Declaration %>

Example:

```
<html><head><title>Declaration Tag</title></head>
<% String name="abc"
        int age=30; %>
<b> The Name is:<%= name %><br>
<b>The age is:<%= age %><br>
<%! int sum(int num1,int num2,int num3){
        return num1+num2+num3;
} %>
<b>The Result is: <%= sum(10,40,50) %>
</body></html>
```

Displaying values using an expression to set an attribute:

```
<html><head><title>JSP Expression</title></head>
<body><h2>JSP Expression </h2>
<ul>
<li>current time: <%= new java.util.Date()%>
<li>host name: <%= request.getResponseHost()%>
<li>Session id: <%= session.getId() %>
</ul></body></html>
```

Sharing data between JSP pages using Session Object:

first.jsp:

```
<%@ Page language=”java”%>
<html><head><title>First</title></head>
<body><form method=”post” action=”second.jsp”>
<b>Username: <input type=”text” name=”uname”><br>
<b> Password: <input type=”password” name=”password”><br>
<input type=”submit” value=”Enter”>
</form></body></html>
```

Second.jsp:

```
<%@Page language=”java” %>
<% string uname=request.getParameter(“uname”);
String password=request.getParameter(“Password”);
Session.getAttribute(“user”,uname);
Session.getAttribute(“password”,password);
%><html><body>< a href=”third.jsp”>Welcome </u> to view Session data</body></html>
```

third.jsp:

```
<@Page language=”java” %>
<% string uname=(string)session.getAttribute(“user”);
String pwd=(string)session.getAttribute(“password”); %>
<html><body>
<b>username is: <%= uname%> <br>
<b> password is:<%= pwd%> </body></html>
```

Scripting Elements:

The Basic Functionality of scripting elements is to allow jsp developers to embed java code directly into jsp. We can classify scripting elements as follows:

i) Declaration
ii) Expression
iii) Scriptlet

1. Declaration:

A JSP declaration lets us define methods or variables in a JSP page. The methods and variables become direct members of the containers generated servlet code. We can define both instance and static members in a declaration. A declaration has the following form:

Example:

```
<%! private int count=1;
Static int total;
Int getcount(){
return count;
}
static int getTotal(){
return total;
}
%>
```

(1) In a declaration we can define a static block of variables and a class also.

(2) In a JSP page we can have any number of declarations in any number of declaring in any order.

(3) Variables declared in a declaration are initialized to java default values.

2. Expressions:

An Expression scripting element is an embedded java.expr that is evaluated and converted to text string. The resulting text string is placed in a JSP output in the location where the element appear. Expressions act as Placeholders for java language expressions. An Expression is evaluated each time the page is accessed and in the value is then embedded into the output HTML.

Example:

```
<%= a+b %>
<%= new java.util.Date()%>
```

(1) Expressions are inserted into the service method of the container generated servlet.

(2) We can have any number of expression per page.

(3) Embedded Java expression should not be terminated with a semicolon.

(4) We can print the value of any object of any primitive datatype to the output stream usng an expression.

(5) We can also use as expression in a JSP action attribute. In this case the value of the expression does not go to the output Stream. It is evaluated at the request time and its value is assigned to the attribute. An expression used in this way is known as request time attribute expression.

3. Scriptlets:

A scriptlet is used to include complete fragments of java code in the body of JSP page. This Scripting element differs from others in two ways:

(1) It is not limited to the declaration of methods and variables.
(2) It does not generate a string output directly as expressions do.

```
<% int sum=0;
for(int i=0;i<10;i++)
sum++;
%>
```

(1) Scriptlet can be placed anywhere within the jsp body.
(2) We can place any number of scriptlet number of scriptlet in JSP.
(3) Each Scriptlet is inserted into the service method of container generated servlet.
(4) The scriptlet is executed each time the page is requested.
(5) The scriptlets contain any java code, they are used for embedding computing logic withing a JSP page.

Comments in a JSP:

In JSP there are three kinds of comments:

(1) Java Comments/ Scripting Comments
(2) HTML Comments/ Content Comments
(3) JSP Comments/ Hidden Comments

In a Declaration or a scriptlet we can have single line multiline java comments.

HTML Comment is of the form <!--------->

A JSP Comment is of the form <% -------- >

Simple JSP that makes use of the all Scripting Elements:

```
<html><head><title>Example JSP</title></head>
<body><unset><h1>Example JSP</h1>
<%! String names[]={“aaa”,”bbb”,”ccc”};
```

```
private string getName(int index){
return names[index];}
%>
<b>The third name is:<%= getName(2)%><br>
<% for(int i=0;i<=5;i++){  %>
<b>This line is displayed 5 times <br>
<% } %>
</center></body></html>
```

Including content in JSP Page:

There are two mechanisms for including another web resource in a JSP page:

(a) The include Directive

(b) The JSP include element

The include directive is processed when a JSP page is translated into a servlet class. The effect of the directive is to insert the text contained in another file either a static content or another JSP Page in the current JSP page. We would probables use the include directive to the include banner comment, copyright information or my piece of the content that we might want to reuse in another page. The Syntax for include directive is:

```
<%@ include file="file name" %>
```

Because we must practically put an include directive in each file that returns the resource referenced by the directive, this approach has its own limitations.

The JSP:include element is processed when a JSP page is executed. The include action allows us to include either a static or dynamic resource in a JSP file. The results of including static or dynamic resources are quite different. If the resource is static, its content is inserted into the calling JSP file. If the resource is dynamic, the request is sent to the included resource, the included page is executed and then the result is included in the response from the calling JSP page. The syntax for the jsp:include is as follows:

```
<%@ include page="include page"%>
```

UNIT – V
STRUTS

Introduction to Struts:

Definition:

Struts is a framework that promotes the use of Model-View-Controller(MVC) architecture for designing large-scale applications. The framework includes a set of custom tag libraries and their associated java classes along with various utility classes.

Model-View-Controller Architecture:

Definition:

MVC is a way to build applications that promotes complete separation between business logic and presentation logic. MVC represents three components:

- Model
- View
- Controller

View:

The View is the user interface i.e., the screens that the end-user of an application actually sees and interacts with. Examples are HTML,JSP's etc..

Controller:

When the user enters some data in the view and clicks on submit button the request is sent to the Controller. The controllers job is to take the data entered by the user and pass it to the Model. A JavaBean is a controller in a J2EE application.

Model:

A Model is a separate Java class that contains the actual business logic. The model executes the business logic and returns some result back to the controller. The controller takes this and presents the user with a new view. A servlet is an example of a Model.

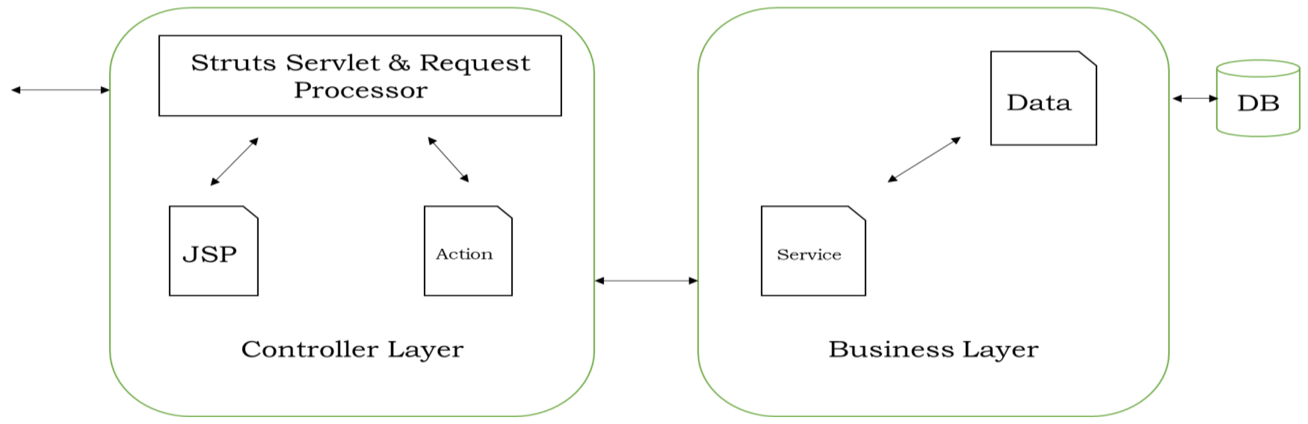
What is Struts Framework?

The struts framework is a standard for developing well-architectured web applications. It has the following features:

1. It is an Open source

2. It is based on MVC
3. It stores application routing information and request mapping information in a single core file called struts-config.xml.

Struts Architecture:



All the incoming requests are intercepted by the struts servlet controller. The struts-config.xml file is used by the controller to determine the routing information. The flow consists of an alteration between two transitions:

1. From View-to-Action.
2. From Action-to-View.

Struts Components:

Various Struts components are:

Controller:

This receives all incoming requests. Its primary function is the mapping of a request URL to an action class.

Struts-config.xml:

This file contains all of the routing and configuration information for the struts application.

Action classes:

Action classes act as a bridge between user-invoked URI's and business services.

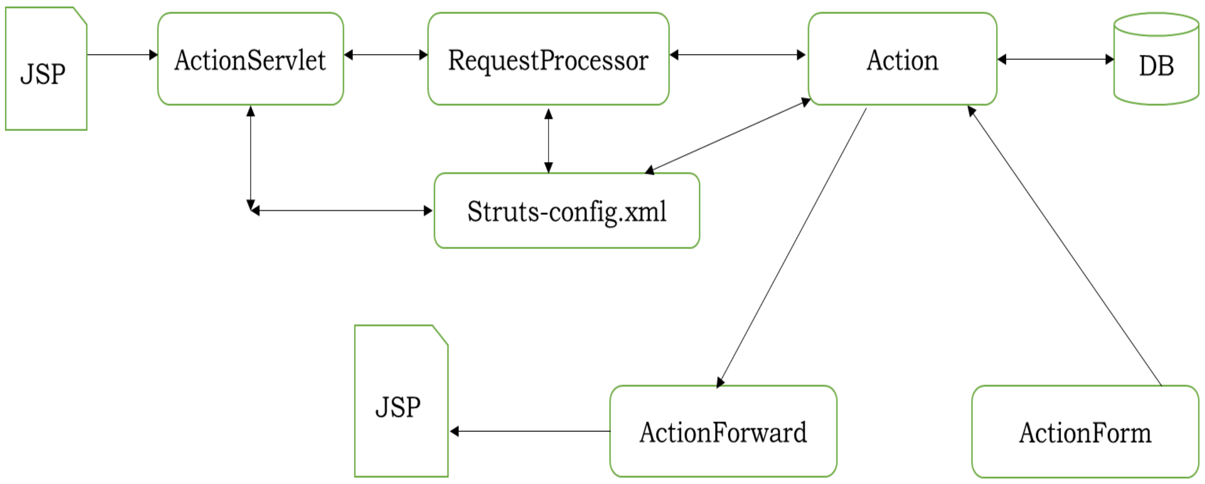
View resources:

View resources consists of JSP's, HTML, Java script and stylesheet files etc..

ActionForms:

ActionForms greatly simplify user form validation by capturing user data from the HTTP request.

Struts Request Lifecycle:



There are several struts controller classes. They are:

1. ActionServlet
2. RequestProcessor
3. ActionMapping
4. Action
5. ActionForm
6. ActionForward

All the above classes reside in org.apache.struts.action package.

ActionServlet:

ActionServlet is the concrete class and extends the javax.servlet.http.HttpServlet. It performs two important things:

1. On start-up it reads the struts configuration file and loads it into memory.
2. It then intercepts HTTP request and handles it appropriately.

In order to load the struts-config.xml into memory it's information from web.xml file must be retrieved into ActionServlet using init() method. The listing of web.xml can be as shown below:

```
<servlet>
        <servlet-name>action</servlet-name>
        <servlet-class>org.apache.struts.action.ActionServlet</servlet-class>
        <init-param>
                <param-name>config</param-name>
                <param-value>/WEB-INF/config/struts-config.xml</param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
</servlet>
```

<load-on-startup>:

When the user request is received for the first time, then to read struts-config.xml from deployment descriptor through init() method and loading into memory is time consuming process. In order to avoid this delay we specify <load-on-startup> as 1 that instructs the servlet container to retrieve struts-config file directly into memory.

The second task the ActionServlet performs is to intercept HTTP request based on the URL pattern and handle appropriately. The URL pattern can either be a path or suffix. This is specified in web.xml as shown below:

```
<servlet-mapping>
        <servlet-name>action</servlet-name>
        <url-pattern>*.do</url-pattern>
</servlet-mapping>
```

RequestProcessor:

ActionServlet intercepts the HTTP request and it delegates it to another class called RequestProcessor by invoking its process() method. RequestProcessor first retrieves the appropriate XML block from struts-config.xml to map the request to corresponding Action class. This process is done through ActionMapping.

ActionMapping:

ActionMapping is the class that holds the mapping between a URL and Action. A sample ActionMapping from the struts-config.xml file looks as follows:

```
<action path="/login" type="LoginAction">
```

ActionForm:

ActionForm is the class that reads data from user form and provides it to Action class.

Action:

The Action class provides an entry point where we can start with our application code to process the request.

RequestProcessor instantiates the Action class specified in the ActionMapping and invokes the execute() method. The signature of execute() method is as follows:

public ActionForward execute(ActionMapping mapping, ActionForm form, HttpServletRequest req,HttpServletResponse res)throws Exception

ActionForward:

The execute() method returns the next view shown to the end user. ActionForward is the class that encapsulates the next view information. The findForward() method on the ActionMapping instance retrieves the ActionForward as follows:

ActionForward forward=mapping.findForward("success");

"success" is the logical name that is passed as the keyword in findForward().

The findForward() method searches for the logical name success and retrieves the corresponding view to be displayed to the user.

Struts Configuration file:

```
<struts-config>
        <data-sources>
        </data-sources>
        <form-beans>
                <form-bean name="bean1" type="com.example.FormBeanOne"/>
                 <form-bean name="bean2" type="com.example.FormBeanTwo"/>
        </form-beans>
        <global-exceptions>
                <exception type="com.example.SomeException"
                path="/WEB-INF/jsp/error.jsp">
        </global-exceptions>
        <global-forwards>
```

```
            <forward name="login" path="/WEB-INF/jsp/login.jsp">
            <forward name="home" path="/WEB-INF/jsp/home.jsp">
      </global-forwards>
<action-mappings>
      <action path="/login" type="com.example.action.LoginAction" name="bean1"
             input="/WEB-INF/jsp/login.jsp">
            <forward name="ok" path="/WEB-INF/jsp/home.jsp"/>
            <forward name="fail" path="/WEB-INF/jsp/login.jsp"/>
      </action>
</action-mappings>
<controller ProcessorClass="org.apache.struts.files.RequestProcessor"/>
</struts-config>
```

Hello World Example:

HelloworldForm.java:

```
package com.example.form;
import org.apache.struts.action.ActionForm;
public class HelloWorldForm extends ActionForm {
    private String message;
    public String getMessage() {
        return message;
    }
    public void setMessage(String message) {
        this.message = message;
    }
}
```

HelloWorldAction.java:

```
package com.example.action;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.apache.struts.action.Action;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;
import com.example.form.HelloWorldForm;
public class HelloWorldAction extends Action {
public ActionForward execute(ActionMapping mapping, ActionForm form, HttpServletRequest
request, HttpServletResponse response) throws Exception {
    HelloWorldForm hwForm = (HelloWorldForm) form;
    hwForm.setMessage("Hello World");
    return mapping.findForward("success");
    }
}
```

struts-config.xml:

```
<struts-config>
    <form-beans>
            <form-bean name="helloWorldForm" type="com.example.form.HelloWorldForm"/>
    </form-beans>
   <global-forwards>
        <forward name="helloWorld" path="/helloWorld.do"/>
    </global-forwards>
    <action-mappings>
        <action path="/helloWorld“
                    type="com.example.action.HelloWorldAction" name="helloWorldForm">
            <forward name="success" path="/helloWorld.jsp" />
        </action>
    </action-mappings>
</struts-config>
```

web.xml:

```
<web-app>
<servlet>
```

```
    <servlet-name>action</servlet-name>
    <servlet-class>org.apache.struts.action.ActionServlet</servlet-class>
    <init-param>
      <param-name>config</param-name>
      <param-value>/WEB-INF/struts-config.xml</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
 </servlet>
  <servlet-mapping>
    <servlet-name>action</servlet-name>
    <url-pattern>*.do</url-pattern>
  </servlet-mapping>
  <welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
  </welcome-file-list>
</web-app>
```

index.jsp:

```
<%@ taglib uri="http://struts.apache.org/tags-logic" prefix="logic" %>
<logic:redirect forward="helloWorld"/>
```

helloWorld.jsp:

```
<%@taglib uri="http://struts.apache.org/tags-bean" prefix="bean" %>
<html>
<head>
<title>Hello World</title>
</head>
<body>
<bean:write name="helloWorldForm" property="message"/>
</body>
</html>
```

Form Validation in struts:

We can define our own validation logic in struts by implementing the Validateable interface in the action class. The workflow interceptor is used to get information about the error messages defined in the action class.

Workflow Interceptor:

The workflow interceptor checks if there is any validation error. It doesn't perform any validation. It is applied when action class implements the Validateable interface. The input is the default parameter for this interceptor that determines the result to be invoked for the action or field error.

Validateabale interface:

The Validateable interface must be implemented to perform validation logic in the action class. It contains only one method validate() that must be overridden in the action class to define the validation logic. Signature of the validate method is:

```
public void validate();
```

ActionSupport class implements Validateable and interface, so we can inherit the ActionSupport class to define the validation logic and error messages.

Steps to perform validation:

The steps are as follows:

1. create the form to get input from the user
2. Define the validation logic in action class by extending the ActionSupport class and overriding the validate method
3. Define result for the error message by the name input in struts.xml file

Example to perform validation:

In this example, we are creating 4 pages :

1. index.jsp for input from the user.
2. RegisterAction.java for defining the validation logic.
3. struts.xml for defining the result and action.
4. welcome.jsp for the view component.

Index.jsp:

```
<%@ taglib uri="/struts-tags" prefix="s" %>
<s:form action="register">
```

```
<s:textfield name="name" label="Name"></s:textfield>
<s:password name="password" label="Password"></s:password>
<s:submit value="register"></s:submit>
</s:form>
```

RegisterAction.java:

```
package com.example;
public class RegisterAction extends ActionSupport{
private String name,password;
public void validate() {
    if(name.length()<1)
        addFieldError("name","Name can't be blank");
    if(password.length()<6)
        addFieldError("password","Password must be greater than 5");
}
public String execute(){
    return "success";
}
}
```

struts.xml:

```
<struts>
<action name="register" class="com.example.RegisterAction">
<result>welcome.jsp</result>
<result name="input">index.jsp</result>
</action>
</struts>
```

welcome.jsp:

```
<%@ taglib uri="/struts-tags" prefix="s" %>
Name:<s:property value="name"/><br/>
Password:<s:property value="password"/><br/>
```

UNIT – VI

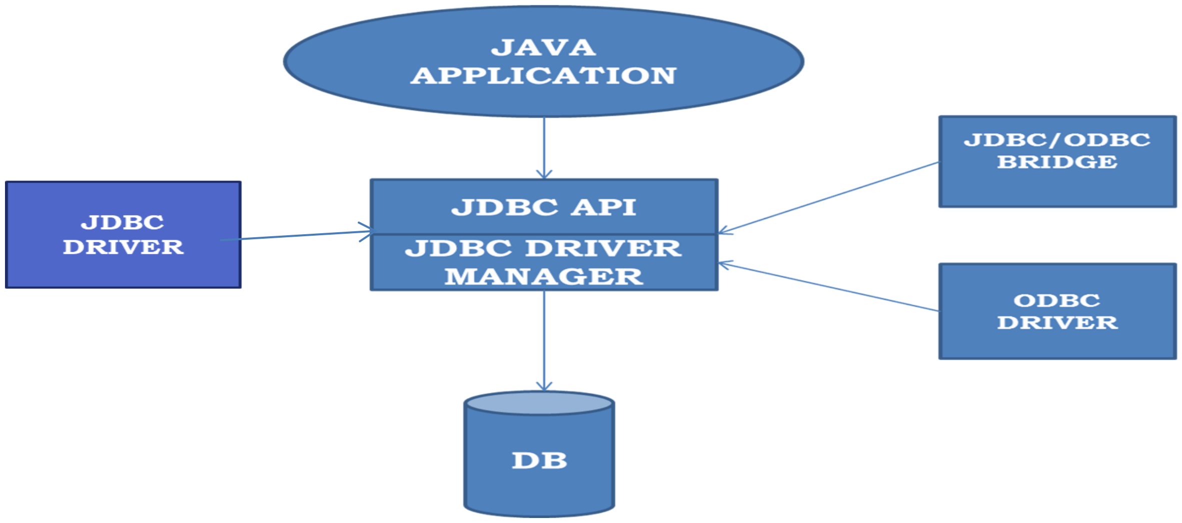
DATABASE ACCESS & AJAX

Introduction:

Every enterprise application is required to access a database either for retrieving the data to be processed (or) to store the processed data. Java offers a simple approach for database connectivity through JDBC using which a java application can be connected virtually to any database.

JDBC is not a s/w product. It is an API which defines interfaces and classes for writing database applications in java. In fact, JDBC is one of the service API's in the J2EE. The java.sql & javax.sql packages provide the necessary library support for the database aware java applications.

JDBC Architecture:



Java Application:

It can be a standalone java program (or) an applet (or) a servlet (or) an EJB, which uses the JDBC API to get connected and perform operations on the data present in database.

JDBC API:

It is a set of classes & interfaces used in a Java Program for database operations. The java.sql & javax.sql packages provide the necessary library support.

Driver Manager:

The primary purpose of the driver manager is to load specific drivers on behalf of user application.

JDBC Driver:

It is a s/w that translates the JDBC method calls into vendor-specific API calls.

ODBC Driver:

ODBC driver is dynamically loaded by the ODBC driver manager for making connection to target database.

Database:

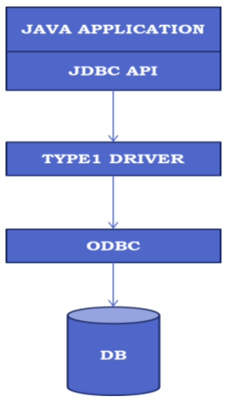
The real-time it is the database server.

The programmer needs a specific driver to connect to a specific database. Driver implementation come in four types:

1. Type 1: JDBC-ODBC Bridge Driver (Bridge)
2. Type 2: Native-API / Partly Java Driver (Native)
3. Type 3: All Java / Net - Protocol Driver (Middleware)
4. Type 4: All Java / Native – Protocol Driver (Pure)

1. Type 1: JDBC – ODBC Bridge Driver (Bridge):

Type 1 driver allows an application to access database through an intermediate ODBC driver. It provides a gateway to the ODBC API, since in purpose is to translate JDBC methods into the ODBC function calls. Here ODBC acts as a mediating layer between the JDBC driver and the vendors (Client libraries).



Advantages:

ODBC drivers are commonly available; hence it can work with huge number of ODBC drivers.

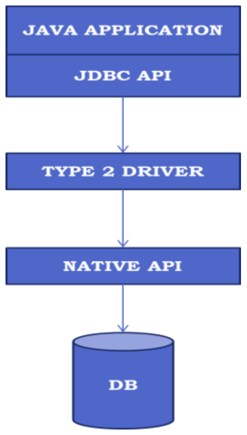
Disadvantages:

1. A ODBC binary code must be loaded on each client machine that uses this driver.

2. Translation overhead between JDBC & ODBC.
3. Does not support all features of java.
4. It works only under Microsoft windows and sun solaris OS.

2. Type 2: Native API / Partly Java Driver (Native):

Type 2 driver converts JDBC calls into data base specific calls for databases. The Type 2 driver communicates directly with database server & therefore requires some binary code to be present on client machine.



Advantages:

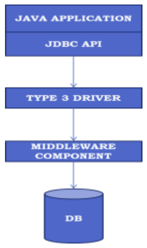
Type 2 driver offers significantly better performance than type 1 driver.

Disadvantages:

The vendor database library needs to be loaded on each client machine consequently. Type 2 drivers cannot be used for internet.

3. Type 3: All Java / Net – Protocol Driver (Middleware):

Type 3 driver translation JDBC calls into the middleware vendor's protocol which is subsequently translated to a DBMS protocol by middleware server. The Middleware provides connectivity to many different databases. The types of drivers are best suited for environment that need to provide connectivity to a variety of DBMS servers and heterogenous databases.



Advantages:

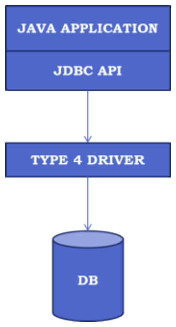
The Type 3 driver is server based. So, there is no need for any vendor database library to be present on client machines.

Disadvantages:

Type 3 drivers require database specific coding be done in the middle tier.

4. Type 4: All Java / Native – Protocol driver (Pure):

Type 4 driver talks directly to the database using Java sources. These type of drivers are completely implemented in java to achieve platform independence and eliminate deployment issues.



Advantages:

Performance is significantly better.

Disadvantages:

The user needs a different driver for each database.

Database Programming using JDBC:

In order to query a database using JDBC we need to follow the below steps:

1. Loading the JDBC driver
2. Defining a connection URL
3. Establishing a connection
4. Creating a statement object
5. Executing a query
6. Process the results
7. Close the Connection

1. Loading the JDBC driver:

To load a driver we specify the class name of the database driver in the class.forName() method. By doing so, we automatically create a driver instance and register it and the JDBC driver manager.

Class.forName(“oracle.jdbc.driver.OracleDriver”);

2. **Defining the connection URL:**

In JDBC, a connection URL specifies the server host port and database name with which we have to establish the connection

String url=”jdbc: oracle: thin:@localhost:1521:XE”;

3. **Establishing the connection:**

With the connection URL, username and password, a network connection to the database can be established and once the connection is established, database queries can be performed entities the connection is closed.

Connection con=DriverManager.getConnectio(Url,username,password);

4. **Create a statement Object:**

Creating a statement object enables us to bend queries and commands to databases.

Statement st=con.Statement();

5. **Executing a query:**

Given a statement object we can send SQL statements to the database by using execute(), executeQuery(), executeUpdate() or executeBatch() methods.

st.executeQuery(“select * from customer”);

6. **Process the Results:**

When a database query is executed , a result set object is returned. The Result set represents a set of rows and columns that we can process by using various methods.

```
ResultSet rs=st.executeQuery(“select * from customer”);
String first_name,last_name;
boolean records=rs.next();
if(!records){
system.out.println(“No records reference”);
}
else{
do{
first_name=rs.getString(“first_name”);
last_name=rs.getString(“last_name”);
system.out.println("First name"+first_name+""+"Last Name"+last_name);
} while(rs.next());
}
```

7. **Close the Connection:**

When we are finished performing queries and processing results, we should close the connection releasing resources to the database.

```
con.close();
```

Accessing the Database from a JSP page:

```
<%@ Page import=" java.sql.* "%>
<html><head><title>Database Access </title></head>
<body bgcolor="pink">
<center><h1>Accessing Data from a Database </h1><br>
<% class.forName("oracle.jdbc.driver.oracleDriver");
String url="jdbc.oracle.thin@localhost:1521:XE";
Connection con=DriverManager.getConnection(url,"nvn","nvn");
Statement st=con.createStatement();
ResultSet rs=st.executeQuery("select * from customer");
%>
<table border="2"><tr><th>Last Name</th>
<th>First Name</th></tr>
<% while(rs.next()){%>
<tr><td><%= rs.getString("lastname"); %></td>
<td><%=rs.getString("firstname");%></td></tr>
<% } %></table></center></body></html>
```

Application – Specific Database Actions:

The various application specific database actions are:

1. Creating a table
2. Inserting data into a table
3. Retrieving the data to be processed
4. Updating the table data

Creating a table:

```
<%@ Page import=”java.sql.*” %>
<html><head><title>Creating a table</title></head>
<body bgcolor=”pink”>
<center><h1>Creating a table</h1><br>
<%try {
Class.forName(“jdbc.oracle.driver.OracleDriver”);
String url=”jdbc:oracle:thin:@localhost:1521:XE”;
Connection con=DriverManager.getConnection(url,”nvn”,”nvn”);
Statement st=con.createStatement();
String cmd=”create table stafflist(ID(integer), name varchar2(30), Dept varchar2(30), Designation
varchar2(30);”;
St.executeUpdate(cmd);
System.out.println(“<b> Table created Successfully “);
}
Catch(Exception e){
Out.println(“An error occurred”);
}
%>
</center></body></html>
```

Inserting Data into a table:

```
<%@ Page import= “java.sql.*” %>
<html><head><title>Inserting Data </h1><br>
<% try {
Class.forName(“jdbc.oracle.driver.OracleDriver”);
String url=”jdbc:oracle:thin:@localhost:1521:XE”;
Connection con=DriverManager.getConnection(url,user,pwd);
Statement st=con.createStatement();
String cmd=”Insert Into stafflist values(1,’abc’,’cse’,”Asst.post”);”;
St.executeUpdate(“cmd”);
out.println("<b> Table data inserted successfully");
}
catch(Exception e){
out.println("An error occurred");
}%></center>
</body></html>
```

Studying javax.sql Package:

The javax.sql package provides the API for the serverside datasource access and processing from the java program.

Interfaces:

1. ConnectionEventListener:

An object that registers to be notified of events generated by a pooled connection object.

2. ConnectionPoolaDataSource:

A factory for pooled connection objects.

3. DataSource:

A factory for connections to the physical data Source that this DataSource represents.

4. PooledConnection:

An object that provides hooks for connection pool management.

5. RowSet:

The interface that adds support to the JDBC API for the JavaBean Component model.

6. Rowset Internal:

The interface that a Rowset object implements in order to present itself to a RowSetReader/RowSetWriter object.

7. RowsetListener:

An interface that must be implemented by a component that wants to be notified when a significant event happens in the life of a RowSet object.

8. RowSetMetaData:

An object that contains information about the columns in a RowSet object.

9. RowSet Reader:

The facility that a disconnected RowSet Object calls on to populate itself with rows of data.

10. RowSet Writer:

An object that implements the RowsetWriter interface called as Writer.

11. XAConnection:

An object that provides support for distributed transactions.

12. XADataSource:

A factory for XAConnection objects that are used internally.

Various Classes defined are:

1. ConnectionEvent:

An event object that provides information about the source of a Connection-related event.

2. RowSet Event:

An event object generated when an event occurs to a RowSet Object.

Javax.sql Package Description:

The javax.sql Package provides an API for Server side data Source access and processing from the java programming language. The package supplements the java.sql package and provides the following:

a. Data Source interface as an alternative to the Driver Manager for established a connection with a data Source.
b. Connection Pooling
c. Distributed transactions
d. RowSets

Applications use the Data Source and the Row Sets directly, out the connection pooling and distributed transactions are used internally by the middle-tier infrastructure.

a) Using the Data Source object for making a connection:

The javax.sql Package provides the preferred way to make a connection to the data source. It offers many advantages such as:

1. Applications do not need to hard code a driver class.
2. Changes can be made to data source properties which means that it is not necessary to make changes in application code when something about the data Source or driver changes.
3. Connection Pooling and distributed transaction are available through a DataSource object that is implements to work with the middle-tier infrastructure.

A Particular Data Source object represents a particular physical Data Source and each connection to Data Source object creates a connection to that physical Data Source.

A logical name for the data source is registered with a naming service that uses JNOI API. An Application can retrieve the Data Source object to create a connection to the physical Data Source it represents.

A Data Source Object can be implemented to work with the middle tier infrastructure so that the connections it provides will be pooled for reuse.

b) Connection Pooling:

Connections made via a Data Source object are implemented to work with a middle tier. Connection Pool Manager participate in connection pooling. Connection Pooling allows a connection to be used and reused thus cursing down substantially the need of making new connections.

Connection Pooling is totally transparent. It is done automatically in the middle tier of a J2EE configuration and so there is no need of change in code from application point of view. An Application simply uses the Data Source.getConnection() method to get the pooled connection and uses it the same way it uses any connection object. The classes are interfaces used for connection pooling are:

(1) Connection Pool DataSource

(2) Pooled Connection

(3) Connection Event

(4) Connection Event Listener

c) Distributed Transactions:

As with pooled connections, connections made via a Data Source objects are implemented to with the middle tier infrastructure may also participate in distributed transactions.

This gives an application the ablility to involve data sources on multiple servers in a single transaction.

d) RowSets:

The Rowset interface works with the various other classes & interfaces. These can be grouped into three categories:

i) Event Notification

ii) Meta data

iii) The Reader / Writer facilities.

Event Notifications:

(a) Rowset Listener:

A Rowset object is a JavaBean is a Java Bean Component and it has participates in the Java Bean event notification mechanism. The Rowset Listener interface is implemented by a component that wants to be notified about the events that occur to a particular Rowset object.

(b) Rowset Event:

A Rowset Object creates an instance of Rowset of Rowset Event and passes it to the listener. The listener can use this Rowset Event object to find which Rowset had the event.

1. <u>Meta Data:</u>

Data about Data (Information about Data)

1. <u>RowSetMetaData:</u>

This interface is derived from the Resultset MetaData and provides information about the columns in a Rowset object. It provides methods for setting information about columns.

2. The Reader / Writer facility:

A Rowset object that implements the Rowset Internal interface can call on the Rowset Reader object associated with it to populate itself with data.

It can also call on the Rowset writer object associated with it to write any changes to its rows back to the datasource from which it originally got the rows.