FOLLOW US

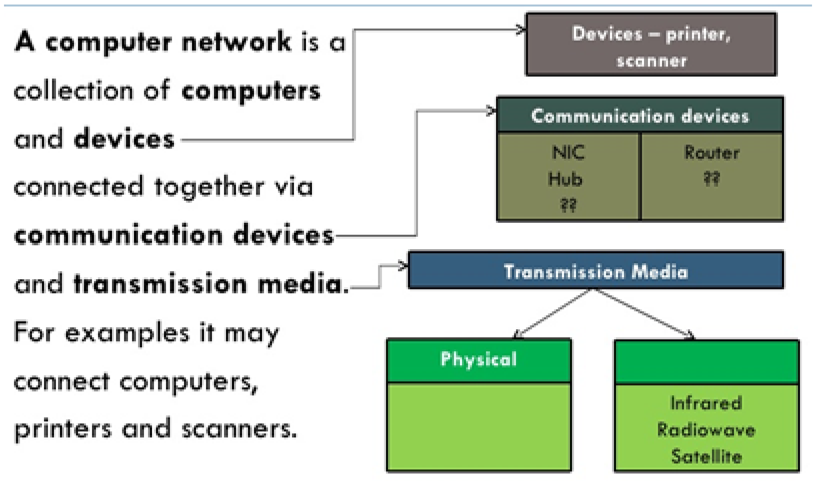
Website: https://bcanepaltu.com
Page:
https://www.facebook.com/bcanepaltu/

QUESTION PAPERS AND NOTES

Web Technologies (Csc-353)

Unit 1: Introduction

Introduction to Networking:



A computer network is a collection of computers and devices connected together via communication devices and transmission media. For examples it may connect computers, printers and scanners.

Importance of Computer Network:

- Sharing of devices such as printer and scanner
- Sharing of program and software
- Sharing of files
- Sharing of data
- Sharing of information
- Better Communication using internet services such as email

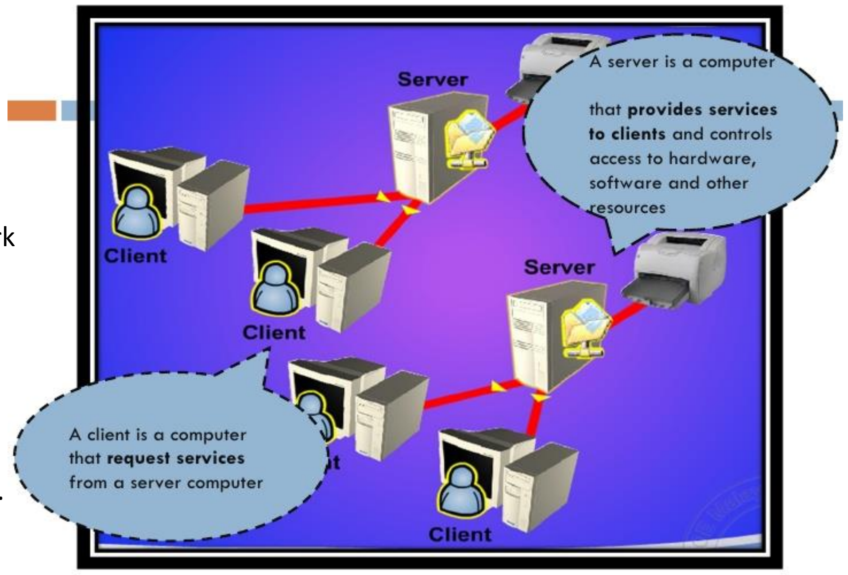
Types of computer Network:

Different	*LAN*	*MAN*	*WAN*
Cost	Low	High	Higher
Network Size	Small	Larger	Largest
Speed	Fastest	Slower	Slowest
Transmission Media	Twisted-pair	Twisted pair, fiber optics	Fiber optics, Radio waves, satellite
No. of Computers	Smallest	Large	Largest

Network Architecture:

Network architecture is the overall design of a computer network that describes how a computer network is configured and what strategies are being used. It is also called network model or network design. Two main network architecture are:

1) Client/server Network
2) Peer to peer Network


Server
A server is a computer that provides services to clients and controls access to hardware, software and other resources
Client
Client
Server
A client is a computer that request services from a server computer
Client

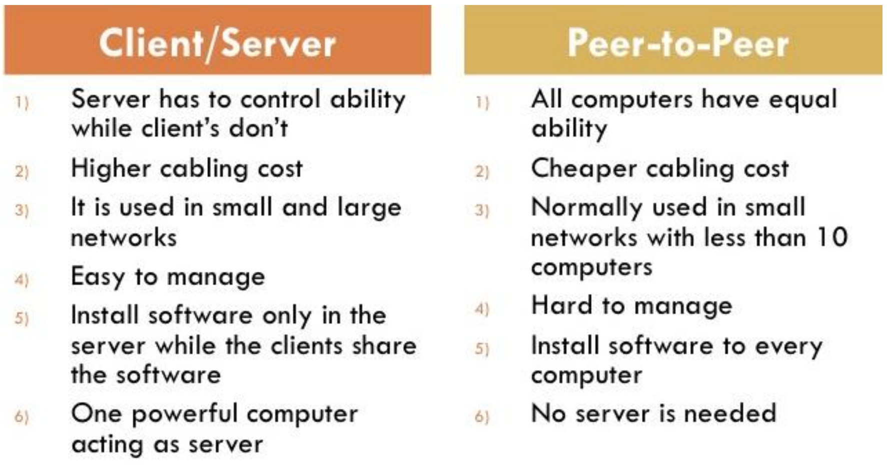
<u>Client/Server:</u>

On a client/server network, one computer act as a server, that provides services and the other computers(client) on the network request services from the server.

<u>Peer-to peer</u>

It is a simple, inexpensive network that typically connects fewer computers. There is no central server in this type of network.

Difference between client/server and peer-to-peer



Client/Server	Peer-to-Peer
1) Server has to control ability while client's don't	1) All computers have equal ability
2) Higher cabling cost	2) Cheaper cabling cost
3) It is used in small and large networks	3) Normally used in small networks with less than 10 computers
4) Easy to manage	4) Hard to manage
5) Install software only in the server while the clients share the software	5) Install software to every computer
6) One powerful computer acting as server	6) No server is needed

Internet and its Evolution:

The Internet is a global network connecting millions of computers. More than 190 countries are linked into exchanges of data, news and opinions. No one actually owns the Internet, and no single person or organization controls the Internet in its entirety.

<u>**WWW :**</u> WWW is a system of interlinked hypertext documents accessed via the Internet. The World Wide Web, or simply Web, is a way of accessing information over the medium of the Internet. It is an information-sharing model that is built on top of the Internet. The Web uses the HTTP protocol, only one of the languages spoken over the Internet, to transmit data. Web services, which use HTTP to allow applications to communicate in order to exchange business logic, use the Web to share information. The web also utilizes browsers, such as Internet Explorer or Firefox, to access Web documents called Web pages that are linked to each other via hyperlinks. Web documents also contain graphics, sounds, text and video.

Web Page: A web page is a document commonly written in Hyper Text Markup Language (HTML) that is accessible through the Internet network using a browser. A web page is accessed by entering a URL address and may contain text, graphics, and hyperlinks to other web pages and files.

Web site: A connected group of pages on the World Wide Web regarded as a single entity, usually maintained by one person or organization and devoted to a single topic or several closely related topics.

URI: Uniform Resource Identifier (URI) is a string of characters used to identify the name of a resource. Such identification enables interaction with representations of the resource over a network, typically the World Wide Web, using specific protocols.

URI looks like: public://myfile.jpeg

URL: Web browsers request pages from web servers by using a URL.
The URL is the address of a web page, like: http://sites/default/files/myfile.jpeg
This provides location and protocol which is http part.

* URI is basically a name, it is representation of an entity or some kind of resource somewhere.
* URL specifies where that resource can be found. So URL is a URI but URI is not a URL because URL belongs to the subset of URI.

Web server: Web servers are computers that deliver (serves) Web pages. Every Web server has an IP address and possibly a domain name. A Web server is a program that uses HTTP (Hypertext Transfer Protocol) to serve the files that form Web pages to users, in response to their requests, which are forwarded by their computers' HTTP clients. Dedicated computers and appliances may be referred to as Web servers as well.

Web client: It typically refers to the Web browser in the user's machine.

Web Browser: Browsers are software programs that allow you to search and view the many different kinds of information that's available on the World Wide Web. The information could be web sites, video or audio information.

SMTP : Simple Mail Transfer Protocol(SMTP), a protocol for sending e-mail messages between servers. Most e-mail systems that send mail over the Internet use SMTP to send messages from one server to another; the messages can then be retrieved with an e-mail client using either POP or IMAP. In addition, SMTP is generally used to send messages from a mail client to a mail server. This is why you need to specify both the POP or IMAP server and the SMTP server when you configure your e-mail application. SMTP by default uses TCP port 25.

POP: Post Office Protocol (POP) is an application-layer Internet standard protocol used by local e-mail clients to retrieve e-mail from a remote server over a TCP/IP connection.

Review of HTML

Introduction to HTML

HTML is a language for describing web pages. It is not a programming language. A markup language specifies the layout and style of a document. The browser does not display the HTML tags, but uses the tags to interpret the content of the page.

- HTML stands for Hyper Text Markup Language
- A markup language is a set of markup tags
- The tags describe document content
- HTML documents contain HTML tags and plain text
- HTML documents are also called web pages

HTML Tags

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

HTML Elements

An HTML element is everything from the start tag to the end tag

HTML Attributes

Attributes provide additional information about HTML elements.

The <!DOCTYPE> Declaration

The <!DOCTYPE> declaration helps the browser to display a web page correctly.
There are many different documents on the web, and a browser can only display an HTML page 100% correctly if it knows the HTML type and version used.

- The <!DOCTYPE> declaration is not an HTML tag; it is an instruction to the web browser about what version of HTML the page is written in.
- The <!DOCTYPE> tag does not have an end tag.

Common DOCTYPE Declarations

HTML 5	<!DOCTYPE html>
HTML 4.01	<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN""http://www.w3.org/TR/html4/loose.dtd">

HTML Text Formatting Tags

Tag	Description
<b>	Defines bold text
<em>	Defines emphasized text
<i>	Defines a part of text in italic
<small>	Defines smaller text
<strong>	Defines important text
<sub>	Defines subscripted text
<sup>	Defines superscripted text
<ins>	Defines inserted text
<del>	Defines deleted text
<mark>	Defines marked/highlighted text

HTML "Computer Output" Tags

Tag	Description
<code>	Defines computer code text
<kbd>	Defines keyboard text
<samp>	Defines sample computer code
<var>	Defines a variable
<pre>	Defines preformatted text

HTML Comment Tags

You can add comments to your HTML source by using the following syntax:
<!-- Write your comments here -->
Comments are not displayed by the browser, but they can help document your HTML.

HTML Hyperlinks (Links)

The HTML <a> tag defines a hyperlink. A hyperlink (or link) is a word, group of words, or image that you can click on to jump to another document. When you move the cursor over a link in a Web page, the arrow will turn into a little hand.
The most important attribute of the <a> element is the href attribute, which indicates the link's destination.
By default, links will appear as follows in all browsers:

- An unvisited link is underlined and blue
- A visited link is underlined and purple
- An active link is underlined and red

Example: <a href="http://www.devcpgn.blogspotcom/">Visit Me !</a>

HTML Links - The id Attribute
The id attribute can be used to create a bookmark inside an HTML document. Bookmarks are not displayed in any special way. They are invisible to the reader.
Example:
An anchor with an id inside an HTML document:
<a id="tips">Useful Tips Section</a>
Create a link to the "Useful Tips Section" inside the same document:
<a href="#tips">Visit the Useful Tips Section</a>
Or, create a link to the "Useful Tips Section" from another page:
<a href="http://www.devcpgn.blogspot.com/html_links.htm#tips">
Visit the Useful Tips Section</a>

Html Links- Target attribute: The target attribute specifies where to open the linked document.
Syntax:

<link target="_blank|_self|_parent|_top|framename">

Attribute Value	Description
_blank	Load in a new window
_self	Load in the same frame as it was clicked
_parent	Load in the parent frameset
_top	Load in the full body of the window
framename	Load in a named frame

The HTML <head> Element:
Inside <head> can include scripts, instruct the browser where to find style sheets, provide meta information, and more. The following tags can be added to the head section: <title>, <style>, <meta>, <link>, <script>, <noscript>, and <base>.

Tag	Description
<head>	Defines information about the document
<title>	Defines the title of a document
<base>	Defines a default address or a default target for all links on a page
<link>	Defines the relationship between a document and an external resource
<meta>	Defines metadata about an HTML document
<script>	Defines a client-side script
<style>	Defines style information for a document

The HTML <title> Element
The <title> tag defines the title of the document. The <title> element is required in all HTML/XHTML documents.
The HTML <link> Element
The <link> tag defines the relationship between a document and an external resource.
The <link> tag is most used to link to style sheets:

```
<head>
<link rel="stylesheet" type="text/css" href="mystyle.css">
</head>
```

The HTML <style> Element

The <style> tag is used to define style information for an HTML document. Inside the <style> element you specify how HTML elements should render in a browser:

```
< head>
<style type="text/css">
body {background-color:yellow;}
p {color:blue;}
</style>
</head>
```

The HTML <meta> Element

Metadata is data (information) about data. The <meta> tag provides metadata about the HTML document. Metadata will not be displayed on the page, but will be machine parsable.
Meta elements are typically used to specify page description, keywords, author of the document, last modified, and other metadata. The metadata can be used by browsers (how to display content or reload page), search engines (keywords), or other web services. <meta> tags always go inside the <head> element. Example of meta tag
Define keywords for search engines:
<meta name="keywords" content="HTML, CSS, XML, XHTML, JavaScript">
Define a description of your web page:
<meta name="description" content="Online help for students of IT ">
Define the author of a page:
<meta name="author" content="Devendra Chapagain”>
Refresh document every 30 seconds:
<meta http-equiv="refresh" content="30">

The HTML <script> Element

The <script> tag is used to define a client-side script, such as a JavaScript. Common uses for JavaScript are image manipulation, form validation, and dynamic changes of content.

HTML <noscript> Tag

The <noscript> tag defines an alternate content for users that have disabled scripts in their browser or have a browser that doesn't support script. The <noscript> element can be used in both <head> and <body>.When used inside the <head> element: <noscript> must contain <link>, <style>, and <meta> elements. The content inside the <noscript> element will be displayed if scripts are not supported, or are disabled in the user's browser.
Example:

```
<script>
document.write("Hello World!")
</script>
<noscript>Your browser does not support JavaScript!</noscript>
```

HTML Images - The <img> Tag and the Src Attribute

Syntax for defining an image: <img src="*url*" alt="*some_text*">

<img> Tag Attributes	Definations
Alt	specifies an alternate text for an image, if the image cannot be displayed
Width, height	Sets width and height of an image

HTML <map> Tag

The <map> tag is used to define a client-side image-map. An image-map is an image with clickable areas.
The <map> element contains a number of <area> elements, that defines the clickable areas in the image map.

Attribute	Value	Description
name	*Mapname*	Required. Specifies the name of an image-map

Example:

```
<img src="planets.gif" width="145" height="126" alt="Planets" usemap="#planetmap">
<map name="planetmap">
  <area shape="rect" coords="0,0,82,126" href="sun.htm" alt="Sun">
  <area shape="circle" coords="90,58,3" href="mercur.htm" alt="Mercury">
  <area shape="circle" coords="124,58,8" href="venus.htm" alt="Venus">
</map>
```

HTML Tables:

Tag	Description
<table>	Defines a table
<th>	Defines a header cell in a table
<tr>	Defines a row in a table
<td>	Defines a cell in a table
<caption>	Defines a table caption
<thead>	Groups the header content in a table
<tbody>	Groups the body content in a table
<tfoot>	Groups the footer content in a table

Example:

```
<table border="1" style="width: 300px">
<tr>
  <td>Jill</td>
  <td>Smith</td>
  <td>50</td>
</tr>
<tr>
  <td>Eve</td>
  <td>Jackson</td>
  <td>94</td>
</tr>
</table>
```

HTML Lists:

The most common HTML lists are ordered and unordered lists:

List properties allow you to:

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

Lists may contain:
<ul> - An unordered list. This will list items using plain bullets.
<ol> - An ordered list. This will use different schemes of numbers to list your items.
<dl> - A definition list. This arranges your items in the same way as they are arranged in a dictionary.

Unordered Lists

An unordered list is a collection of related items that have no special order or sequence. This list is created by using HTML <ul> tag. Each item in the list is marked with a bullet.

Example

```
<html>
<head>
<title>HTML Unordered List</title>
</head>
<body>
<ul>
<li>Beetroot</li>
<li>Ginger</li>
<li>Potato</li>
</ul>
</body>
</html>
```

This will produce following result:

- Beetroot
- Ginger
- Potato
- Radish

The type Attribute

You can use type attribute for <ul> tag to specify the type of bullet you like. By default it is a disc. Following are the possible options:

▪ <ul type="square">
● <ul type="disc">
○ <ul type="circle">

Example:

```
<html>
<head>
<title>HTML Unordered List</title>
</head>
<body>
  <ul type="square">
  <li>Beetroot</li>
  <li>Ginger</li>
  <li>Potato</li>
  <li>Radish</li>
  </ul>
</body>
</html>
```

This will produce following result:

▪ Beetroot
▪ Ginger
▪ Potato
▪ Radish

HTML Ordered Lists

If you are required to put your items in a numbered list instead of bulleted then HTML ordered list will be used. This list is created by using <ol> tag. The numbering starts at one and is incremented by one for each successive ordered list element tagged with <li>.

Example

```
<html>
<head>
<title>HTML Ordered List</title>
</head>
<body>
<ol>
<li>Beetroot</li>
<li>Ginger</li>
<li>Potato</li>
<li>Radish</li>
</ol>
</body>
</html>
```

This will produce following result:

```
    1.  Beetroot
    2.  Ginger
    3.  Potato
    4.  Radish
```

The type Attribute

You can use type attribute for <ol> tag to specify the type of numbering you like. By default it is a number. Following are the possible options:

```
<ol type="1"> - Default-Case Numerals.
<ol type="I"> - Upper-Case Numerals.
<ol type="i"> - Lower-Case Numerals.
<ol type="a"> - Lower-Case Letters.
<ol type="A"> - Upper-Case Letters.
```

The start Attribute

You can use start attribute for <ol> tag to specify the starting point of numbering you need. Following are the possible options:

```
<ol type="1" start="4">   - Numerals starts with 4.
<ol type="I" start="4">   - Numerals starts with IV.
<ol type="i" start="4">   - Numerals starts with iv.
<ol type="a" start="4">   - Letters starts with d.
<ol type="A" start="4">   - Letters starts with D.
```

Example

Following is an example where we used <ol type="i" start="4" >

```
<html>
<head>
<title>HTML Ordered List</title>
</head>
<body>
  <ol type="i" start="4">
  <li>Beetroot</li>
  <li>Ginger</li>
  <li>Potato</li>
  <li>Radish</li>
  </ol>
```

This will produce following result:

```
    iv. Beetroot
    v. Ginger
    vi. Potato
    vii. Radish
```

</body>
</html>

HTML Description Lists

The definition/Description list is the ideal way to present a glossary, list of terms, or other name/value list.

Definition List makes use of following three tags.

<dl> - Defines the start of the list
<dt> - A term
<dd> - Term definition
</dl> - Defines the end of the list

A description list is a list of terms/names, with a description of each term/name.

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

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

HTML Colors:

CSS colors are defined using a hexadecimal (hex) notation for the combination of Red, Green, and Blue color values (RGB). The lowest value that can be given to one of the light sources is 0 (hex 00). The highest value is 255 (hex FF). Color names are also defined in HTML like: Aqua, Black, Blue, Brown, Cyan, Red, Gold, Indigo, etc. Hex values are written as 3 double digit numbers, starting with a # sign.

Color	Color HEX	Color RGB
⬛	#000000	rgb(0,0,0)
🟥	#FF0000	rgb(255,0,0)
🟩	#00FF00	rgb(0,255,0)
🟦	#0000FF	rgb(0,0,255)
🟨	#FFFF00	rgb(255,255,0)
🟦	#00FFFF	rgb(0,255,255)
🟪	#FF00FF	rgb(255,0,255)
▨	#C0C0C0	rgb(192,192,192)
	#FFFFFF	rgb(255,255,255)

HTML Forms:

HTML forms are used to pass data to a server. An HTML form can contain input elements like text fields, checkboxes, radio-buttons, submit buttons and more. A form can also contain select lists, textarea, fieldset, and label elements. HTML Forms are used to select different kinds of user input. A form is defined with a <form> tag.

A form has two duties: to collect information from the user and to send that information to a separate web page for processing. For example, whenever you submit personal information to a web, you are using a form. Or whenever you type the keyword into your search engine, you are using a form. Forms are the heart and soul of the World wide web.

HTML Forms - The Input Element

The most important form element is the <input> element. The <input> element is used to select user information. An <input> element can vary in many ways, depending on the type attribute. An <input> element can be of type text field, checkbox, password, radio button, submit button, and more.
The most common input types are given below:

Syntax:
<input type="value">

Attribute value	Description
Text	A text field
Password	A password text field where each keystroke appears as an *
Button	A new button other than submit and reset button
Checkbox	A checkbox
Radio	A radio button
Reset	A reset button
Submit	A submit button
Select	A selection list
TextArea	A multiline text entry field
Hidden	A field that may contain value but is not displayed within a form

Creating Forms:
A form is created using a <form> tag
<form method=POST action="samepage url.asp">

Method:
The method tag can be set either GET or POST
GET: GET method sends the data captured by form element to the web server encoded into URL, which points to web server. The data captured in form element is appended to the URL.

POST: POST method sends the data captured by form element back to the web server as a separate bit stream of data. When there is a large amount of data to be send back to the web server, this method is used.

- If the method attribute is not specified to the <form> </form> tags, the default method used by the browser to send data back to the web server is GET method.
  Action:

The action tag specifies what page will process the information entered by the user. The server side program that process this data can be written in any scripting language that web server understand. Commonly used server side scripting are: JavaScript, VB Script and ASP.

Text element:
Text element are data entry field used in a HTML forms. Text field accept a single line of text entry.
<input type="text" name=txt_name value="some value">

Events:
Focus()
Blur()
Select()
Change()

JavaScript provides the following event handlers for the text object's events.

- onFocus()
- onBlur()
- onSelct()
- onChande()

Password Element:
All keystroke for this field are displayed as an asterisk (*). This make password element ideal for accepting input of confidential information, such as password, bank a/c number etc.
<input type="password" name="txt_pwd" value="">

Events:
Focus()
Blur()
Select()
Change()

JavaScript provides the following event handlers for the password object's events.

- onFocus()
- onBlur()
- onSelct()
- onChande()

Button Element:
The HTML Button element is a commonly used form object. It is generally used to trigger appropriate form level processing.
<input type="button" name="btn_ok" value="ok">

Events:
Click()

JavaScript provides the following event handlers for the buttonobject's events.

- onClick()

Submit Button Element:
The submit button is a special purpose button. The submit button submits the current data held in each data element to webserver for further processing.
<input type="Submit" name="btn_submit" value="SUBMIT">

Events:
Click()

JavaScript provides the following event handlers for the submit object's events.

- onClick()

The Reset Button Element:
<input type="Reset" name="btn_reset" value="RESET">

Events:
Click()

JavaScript provides the following event handlers for the Reset object's events.

- onClick()

The checkbox element
<input type="checkbox" value="yes" name="yes/no" CHECKED>

Events:

JavaScript provides the following event handlers for the checkbox object's events.

- onClick()

Click()

The Radio Element:

<input type= “radio” name=”Radio group name” value=”” CHECKED>

Event

Clicked()

JavaScript provides the following event handlers for the radio object’s events.

- onClicked()

TextArea Element:

The textarea form element provides a way to create a customized size, multiple line text entry.

<textarea rows="4" cols="50">

</textarea>

Event:

Focus()

Blur()

Select()

JavaScript provides the following event handlers for the TextArea object’s events.

- onFocus()
- onBlur()
- onSelect()

The select and Option element:

A select object on HTML form appears as srop-down list or scrollable list of selectable items.

```
<SELECT Name=”Items”>
        <option SELECTED> Computer
        <option> Mouse
        <option> Keyboard
</SELECT>
```

If the <SIZE> attribute is set to a value less than the actual choice available in the select list a scrollable list will be created.

```
<SELECT name=”items” size=2 MULTIPLE>    * to select multiple objects MULTIPLE attribute must be used.
        <option SELECTED> Computer
        <option> Mouse
        <option> Keyboard
</SELECT>
```

Special Characters:

Certain characters are taken to have special meaning within the context of an HTML document.

Char	Number	Entity	Description
©	&#169;	&copy;	COPYRIGHT SIGN
®	&#174;	&reg;	REGISTERED SIGN
€	&#8364;	&euro;	EURO SIGN
™	&#8482;	&trade;	TRADEMARK
←	&#8592;	&larr;	LEFTWARDS ARROW

↑	&#8593;	&uarr;	UPWARDS ARROW
→	&#8594;	&rarr;	RIGHTWARDS ARROW
↓	&#8595;	&darr;	DOWNWARDS ARROW
♠	&#9824;	&spades;	BLACK SPADE SUIT
♣	&#9827;	&clubs;	BLACK CLUB SUIT
♥	&#9829;	&hearts;	BLACK HEART SUIT
♦	&#9830;	&diams;	BLACK DIAMOND SUIT

Some Mathematical Symbols:

Char	**Number**	**Entity**	**Description**
∀	&#8704;	&forall;	FOR ALL
∂	&#8706;	&part;	PARTIAL DIFFERENTIAL
∃	&#8707;	&exist;	THERE EXISTS
∅	&#8709;	&empty;	EMPTY SETS
∇	&#8711;	&nabla;	NABLA
∈	&#8712;	&isin;	ELEMENT OF
∉	&#8713;	&notin;	NOT AN ELEMENT OF
∋	&#8715;	&ni;	CONTAINS AS MEMBER
∏	&#8719;	&prod;	N-ARY PRODUCT
∑	&#8721;	&sum;	N-ARY SUMMATION

HTML Div tags:

The <div> tag defines a division or a section in an HTML document. The <div> element is very often used together with CSS, to layout a web page. By default, browsers always place a line break before and after the <div> element. However, this can be changed with CSS.

Attribute	**Value**	**Description**
align	left right center justify	Not supported in HTML5. Specifies the alignment of the content inside a <div> element

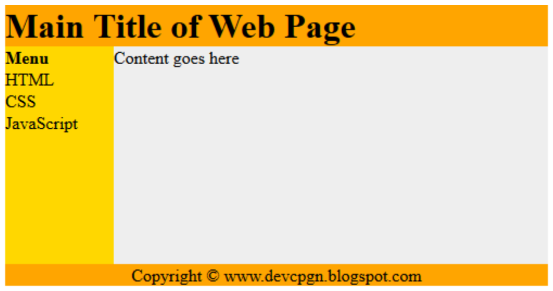
```
<body>
<div id="container" style="width:500px">
<div id="header" style="background-color:#FFA500;">
<h1 style="margin-bottom:0;">Main Title of Web Page</h1></div>

<div id="menu" style="background-
color:#FFD700;height:200px;width:100px;float:left;">
<b>Menu</b><br>
HTML<br>
CSS<br>
JavaScript</div>

<div id="content" style="background-
color:#EEEEEE;height:200px;width:400px;float:left;">
Content goes here</div>
<div id="footer" style="background-color:#FFA500;clear:both;text-
align:center;">
Copyright © www.devcpgn.blogspot.com</div>
```

gn.

```
<body>
<table width="500">
<tr>
<td colspan="2" style="background-color:#FFA500;">
<h1>Main Title of Web Page</h1>
</td></tr>
<tr>
<td style="background-color:#FFD700;width:100px;">
<b>Menu</b><br>HTML<br>CSS<br>JavaScript
</td>
<td style="background-
color:#EEEEEE;height:200px;width:400px;">
Content goes here</td>
</tr>
<tr>
<td colspan="2" style="background-color:#FFA500;text-
align:center;">
Copyright ©www.devcpgn.blogspot.com </td>
```



HTML <span> Tag:

A <span> element used to color a part of a text:
Example: <p>My mother has <span style="color:blue">blue</span> eyes.</p>

Events:

Events can be user actions, such as clicking a mouse button or pressing a key, or system occurrences, such as running out of memory.

Review of CSS:

Introduction:

Cascading Style Sheets (CSS) is a style sheet language used for describing the look and formatting of a document written in a markup language. It was intended to allow developers to separate content from design and layout so that HTML could perform more of the function without worry about the design and layout. It is used to separate style from content.

Advantages of CSS

<u>CSS saves time</u> − You can write CSS once and then reuse same sheet in multiple HTML pages. You can define a style for each HTML element and apply it to as many Web pages as you want.

<u>Pages load faster</u> − If you are using CSS, you do not need to write HTML tag attributes every time. Just write one CSS rule of a tag and apply it to all the occurrences of that tag. So less code means faster download times.

<u>Easy maintenance</u> − To make a global change, simply change the style, and all elements in all the web pages will be updated automatically.

Superior styles to HTML – CSS has a much wider array of attributes than HTML, so you can give a far better look to your HTML page in comparison to HTML attributes.

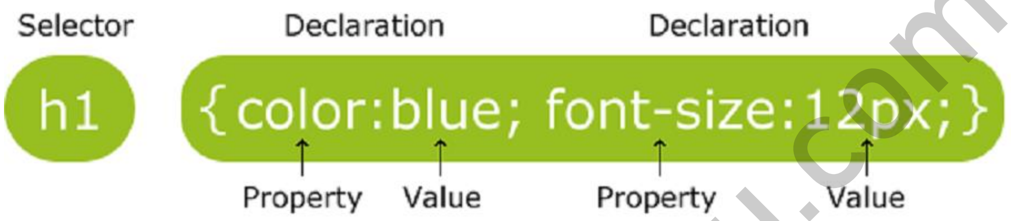
Multiple Device Compatibility – Style sheets allow content to be optimized for more than one type of device. By using the same HTML document, different versions of a website can be presented for handheld devices such as PDAs and cell phones or for printing.

Global web standards – Now HTML attributes are being deprecated and it is being recommended to use CSS. So it's a good idea to start using CSS in all the HTML pages to make them compatible to future browsers.

Platform Independence – The Script offer consistent platform independence and can support latest browsers as well.

Syntax :

A CSS rule has two main parts: a *selector* and one or more *declarations*. **Selector** is normally the HTML element you want to style and each **declaration** consists of a *property* and *value*. The property is the style attribute we want to use and each property has a value associated with it.



* For Detail CSS Properties reference to CSS.doc

Selector – A selector is an HTML tag at which a style will be applied. This could be any tag like <h1> or <table> etc.
Property - A property is a type of attribute of HTML tag. Put simply, all the HTML attributes are converted into CSS properties. They could be color, border etc.
Value - Values are assigned to properties. For example, color property can have value either red or #F1F1F1 etc.

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

Inserting CSS

We can use style sheets in three different ways in our HTML document. CSS can be added to HTML in the following ways:

- **Inline** - using the style attribute in HTML elements
- **Internal** - using the <style> element in the <head> section
- **External** - using an external CSS file

The preferred way to add CSS to HTML, is to put CSS syntax in separate CSS files.

When there more than one style specified for an HTML element than inline style (inside an HTML element) has the highest priority, which means that it will override a style defined inside the <head> tag, or in an external style sheet, or in a browser (a default value).

* If the link to the external style sheet is placed after the internal style sheet in HTML <head>, the external style sheet will override the internal style sheet!

Inline Styles

An inline style can be used if a unique style is to be applied to one single occurrence of an element. To use inline styles, use the style attribute in the relevant tag. The style attribute can contain any CSS property. The example below shows how to change the text color and the left margin of a paragraph:

```
<p style="color:blue;margin-left:20px;">This is a paragraph.</p>
```

Examples:

```
<html>
<body style="background-color:yellow;">
<h2 style="background-color:red; text-align:center;">This is a heading</h2>
<p style="background-color:green;">This is a paragraph.</p>
<h1 style="font-family:verdana;">A heading</h1>
<p style="font-family:arial;color:red;font-size:20px;">A paragraph.</p>
</body>
</html>
```

Internal Style Sheet

An internal style sheet can be used if one single document has a unique style. Internal styles are defined in the <head> section of an HTML page, by using the <style> tag, like this:

```
<head>
<style>
body {background-color:yellow;}
p {color:blue;}
</style>
</head>
```

External Style Sheet

An external style sheet is ideal when the style is applied to many pages. With an external style sheet, you can change the look of an entire Web site by changing one file. Each page must link to the style sheet using the <link> tag. The <link> tag goes inside the <head> section:

```
<head>
<link rel="stylesheet" type="text/css" href="mystyle.css">
</head>
```

CSS Comments

Comments are used to explain your code, and may help you when you edit the source code at a later date. Comments are ignored by browsers. A CSS comment begins with "/*", and ends with "*/"

Eg. `/* this is a css comment */`

ID and Class Selectors

The Class Selectors

You can define style rules based on the class attribute of the elements. All the elements having that class will be formatted according to the defined rule.

```
.black {
  color: #000000;
}
```

This rule renders the content in black for every element with class attribute set to black in our document.

The ID Selectors

You can define style rules based on the id attribute of the elements. All the elements having that id will be formatted according to the defined rule.

```
#black {
  color: #000000;
}
```

This rule renders the content in black for every element with id attribute set to black in our document.

Grouping and Nesting Selectors

Pseudo Classes and Elements

Client-side Programming (Review of JavaScript):

Introduction to JavaScript:

JavaScript is a Scripting Language. A scripting language is a lightweight programming language. JavaScript code can be inserted into any HTML page, and it can be executed by all types of web browsers.
JavaScript is used to make web pages interactive. It runs on your visitor's computer and doesn't require constant downloads from your website. JavaScript and Java are completely different languages, both in concept and design.

✦ Unlike HTML, JavaScript is case sensitive - therefore watch your capitalization closely when you write JavaScript statements, create or call variables, objects and functions.

Why JavaScript?

- JavaScript can be used to create cookies
- Transmitting information about the user's reading habits and browsing activities to various websites. Web pages frequently do this for web analytics, ad tracking, personalization or other purposes.
- Interactive content, for example games, and playing audio and video
- Validating input values of a Web form to make sure that they are acceptable before being submitted to the server.
- Loading new page content or submitting data to the server via AJAX without reloading the page (AJAX: Asynchronous JavaScript and XML: Using AJAX, webpages get updated in the background without reloading and refreshing the webpage)
- Animation of page elements, fading them in and out, resizing them, moving them, etc.

The HTML <script> tag is used to insert a JavaScript into an HTML page. A simple example is given below:

```
<html>
<body>
<script type="text/javascript">
document.write("Hello World!");
</script>
</body>
</html>
```

The example below shows how to add HTML tags to the JavaScript:

```
<html>
<body>
<script type="text/javascript">
document.write("<h1>Hello World!</h1>");
</script>
</body>
</html>
```

JavaScript Comments

JavaScript comments can be used to explain JavaScript code, and to make it more readable. It can also be used to prevent execution, when testing alternative code.

Single Line Comments

Single line comments start with //.
Any text between // and the end of the line, will be ignored by JavaScript (will not be executed).
The given example uses a single line comment at the end of each line, to explain the code:

```
var x = 5;      // Declare x, give it the value of 5
var y = x + 2;  // Declare y, give it the value of x + 2
```

Multi-line Comments

Multi-line comments start with /* and end with */. Any text between /* and */ will be ignored by JavaScript.

```
/*
This is the example of
Multiline comment
*/
```

JavaScript Statements:

JavaScript statements are "instructions" to be "executed" by the web browser. This JavaScript statement tells the browser to write "Hello World" to the web page:

```
    document.write("Hello World");
```

The semicolon is optional (according to the JavaScript standard), and the browser is supposed to interpret the end of the line as the end of the statement.

JavaScript Code:

JavaScript code (or just JavaScript) is a sequence of JavaScript statements. Each statemen is executed by the browser in the sequence they are written. Following example will write a heading and two paragraphs to a web page:

Example

```
<script type="text/javascript">
document.write("<h1>This is a heading</h1>");
document.write("<p>This is a paragraph.</p>");
document.write("<p>This is another paragraph.</p>");
</script>
```

JavaScript Code Blocks:

JavaScript statements can be grouped together in code blocks, inside curly brackets {...}. The purpose of code blocks is to define statements to be executed together. One place you will find statements grouped together in blocks, are in JavaScript functions:

Example

```
function myFunction() {
   document.write("hello");
  document.write("Dev");
}
```

JavaScript Keywords

JavaScript statements often start with a keyword to identify the JavaScript action to be performed. Some of the JavaScript keywords or reserved words are:

abstract	default	float
boolean	delete	for
break	do	function
byte	double	goto
case	else	if
catch	enum	implements
char	export	import
class	extends	in
const	false	instanceof
continue	final	int
debugger	finally	interface

long	short	true
native	static	try
new	super	typeof
null	switch	var
package	synchronized	void
private	this	volatile
protected	throw	while
public	throws	with
return	transient	

JavaScript Variables:

As with algebra, JavaScript variables are used to hold values or expressions. A variable can have a short name, like x, or a more descriptive name, like lastname.

Rules for JavaScript variable names:

→ Variable names are case sensitive (a and A are two different variables)
→ Variable names must begin with a letter or the underscore character
→ Names can contain letters, digits, underscores, and dollar signs.
→ Names must begin with a letter
→ Names can also begin with $ and _
→ Reserved words (like JavaScript keywords) cannot be used as names

You can declare JavaScript variables with the var statement:
Example
var x = 5;
var y = 6;
var z = x + y;

Or simply you can write
X=5;

JavaScript Data types

One of the most fundamental characteristics of a programming language is the set of data types it supports. These are the type of values that can be represented and manipulated in a programming language. JavaScript variables can hold many data types: numbers, strings, arrays, objects and more:

Example:

```
var length = 10;                               // Number
var firstName = "Kamal";                      // String
var colors = ["Red", "Green", "Blue"];           // Array
var x = {firstName:"Binod", lastName:"Aryal"};    // Object
```

JavaScript has dynamic types. This means that the same variable can be used as different types:

```
var x;              // Now x is undefined
var x = 10;          // Now x is a Number
var x = "Kamal";     // Now x is a String
```

JavaScript Operators:

1. Arithmetic Operators
2. Comparison Operators
3. Logical (or Relational) Operators
4. Assignment Operators
5. Conditional (or ternary) Operators

- **Arithmetic Operators**

Arithmetic operators are used to perform arithmetic on numbers (literals or variables).

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus
++	Increment
--	Decrement

Example:

```
<html>
  <body>
    <script type="text/javascript">
        var a = 22;b = 5;
        var c = "Test";
        var linebreak = "<br />";

        document.write("a + b = ");
        result = a + b;
        document.write(result);
        document.write(linebreak);

        document.write("a - b = ");
        document.write(a-b);
        document.write(linebreak);

        document.write("a / b = ");
        result = a / b;
        document.write(result);
        document.write(linebreak);

        document.write("a % b = ");
        result = a % b;
        document.write(result);
        document.write(linebreak);

        document.write("a + b + c = ");
        result = a + b + c;
        document.write(result);
        document.write(linebreak);

        a = a++;
        document.write("a++ = ");
        result = a++;
        document.write(result);
        document.write(linebreak);

        b = b--;
        document.write("b-- = ");
        result = b--;
        document.write(result);
        document.write(linebreak);
    </script>
  </body>
</html>
```

- **JavaScript Assignment Operators**

Assignment operators assign values to JavaScript variables.

Operator	Example	Same As	Example2

=	x = y	x = y	var x = 10;
			x += 5;
			x -= 5;
			txt1 = "Dev";
			txt2 = "Cpgn";
			txt3 = txt1 + " " + txt2;

+=	x += y	x = x + y	
-=	x -= y	x = x – y	
*=	x *= y	x = x * y	
/=	x /= y	x = x / y	
%=	x %= y	x = x % y	

- **JavaScript Comparison Operator**

Operator	Description	**Example** (let x=3)
==	equal to	X==3 True
===	is exactly equal to value and type	X===3 True
!=	not equal	X!=3 False
!==	not equal value or not equal type	X!==5 True
>	greater than	x>5 False
<	less than	X<5 True
>=	greater than or equal to	x>=5 False
<=	less than or equal to	X<=2 True

- **Logical Operators:**

Logical operators are used to determine the logic between variables or values.

Operator	Description	**Example** (let x=5 and y=2)
&&	AND	(x<7 && y>1) is true
||	OR	(x<7 || y>5) is true
!	NOT	!(x==y) is true

- **Conditional Operator (? :)**

The conditional operator first evaluates an expression for a true or false value and then executes one of the two given statements depending upon the result of the evaluation.

Example:

```
<html>
  <body>
    <script type="text/javascript">
        var a = 30;
        var b = 20;
        //var linebreak = "<br />";
        result = (a > b) ? "A is greater" : "B is greater ";
        document.write(result);
        document.write("<br/>");
    </script>
  </body>
</html>
```

JavaScript Functions:

Like any other advanced programming language, JavaScript also supports all the features necessary to write modular code using functions. A JavaScript function is a block of code designed to perform a particular task.

JavaScript allows us to write our own functions as well.

Function Definition

Before we use a function, we need to define it. The most common way to define a function in JavaScript is by using the function keyword, followed by a unique function name, a list of parameters (that might be empty), and a statement block surrounded by curly braces.

Syntax:

```
<script type="text/javascript">
      function functionname(parameter-list)
      {
         statements
      }
</script>
```

Example:

```
<script type="text/javascript">
      function sayHello()
      {
         alert("Hello there");
      }
</script>
```

Calling a Function

To invoke a function somewhere later in the script, you would simply need to write the name of that function as shown in the following code.

```
<html>
   <head>

      <script type="text/javascript">
         function sayHello()
         {
            document.write ("Hello there!");
         }
      </script>

   </head>
   <body>
      <p>Click the button to call the function</p>
      <form>
         <input type="button" onclick="sayHello()" value="Say Hello">
      </form>
   </body>
</html>
```

Function Parameters

There is a facility to pass different parameters while calling a function. These passed parameters can be captured inside the function and any manipulation can be done over those parameters. A function can take multiple parameters separated by comma.

```
<html>
   <head>

      <script type="text/javascript">
         function sayHello(name, age)
         {
            document.write (name + " is " + age + " years old.");
         }
      </script>

   </head>
   <body>
      <form>
         <input type="button" onclick="sayHello('Dev', 25)" value="Say Hello">
      </form>
   </body>
</html>
```

The return Statement

A JavaScript function can have an optional return statement. This is required if you want to return a value from a function. This statement should be the last statement in a function.

For example, you can pass two numbers in a function and then you can expect the function to return their multiplication in your calling program.

```
<html>
   <head>
      <script type="text/javascript">
         function concatenate(first, last)
         {
            var full;
            full = first + last;
            return full;
         }

         function secondFunction()
         {
            var result;
            result = concatenate('Ram', 'Sita');
            document.write (result );
         }
      </script>
   </head>
   <body>
      <form>
         <input type="button" onclick="secondFunction()" value="Call Function">
      </form>
  </body>
</html>
```

JavaScript - Dialog Boxes

JavaScript supports three important types of dialog boxes. These dialog boxes can be used to raise and alert, or to get confirmation on any input or to have a kind of input from the users.

1) Alert Dialog Box

An alert dialog box is mostly used to give a warning message to the users. For example, if one input field requires to enter some text but the user does not provide any input, then as a part of validation, you can use an alert box to give a warning message.

Nonetheless, an alert box can still be used for friendlier messages. Alert box gives only one button "OK" to select and proceed.

Example:

```
<html>
   <head>
         <script type="text/javascript">
            function Warn() {
               alert ("This is a warning message!");
               document.write ("This is a warning message!");
            }
      </script>

   </head>
   <body>
      <form>
         <input type="button" value="Click Me" onclick="Warn();" />
      </form>
   </body>
</html>
```

2) Confirmation Dialog Box

A confirmation dialog box is mostly used to take user's consent on any option. It displays a dialog box with two buttons: Ok and Cancel.

If the user clicks on the OK button, the window method confirm() will return true. If the user clicks on the Cancel button, then confirm() returns false. You can use a confirmation dialog box as follows.

```
<html>
```

```
   <head>

      <script type="text/javascript">
            function getConfirmation(){
               var retVal = confirm("Do you want to continue ?");
               if( retVal == true ){
                  document.write ("User wants to continue!");
                  return true;
               }
               else{
                  Document.write ("User does not want to continue!");
                  return false;
               }
            }
      </script>

   </head>
   <body>
      <form>
         <input type="button" value="Click Me" onclick="getConfirmation();" />
      </form>

   </body>
</html>
```

3) Prompt Dialog Box

The prompt dialog box is very useful when you want to pop-up a text box to get user input. Thus, it enables you to interact with the user. The user needs to fill in the field and then click OK.

This dialog box is displayed using a method called prompt() which takes two parameters: (i) a label which you want to display in the text box and (ii) a default string to display in the text box.

This dialog box has two buttons: OK and Cancel. If the user clicks the OK button, the window method prompt() will return the entered value from the text box. If the user clicks the Cancel button, the window method prompt() returns null.

Example:

```
<html>
   <head>

      <script type="text/javascript">
         <!--
            function getValue(){
               var retVal = prompt("Enter your name : ", "your name here");
               document.write("You have entered : " + retVal);
            }
         //-->
      </script>

   </head>

   <body>
      <p>Click the following button to see the result: </p>

      <form>
```

```
            <input type="button" value="Click Me" onclick="getValue();" />
        </form>

    </body>
</html>
```

JavaScript - Events

What is an Event?

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. Other examples include events like pressing any key, closing a window, resizing a window, etc. Developers can use these events to execute JavaScript coded responses, which cause buttons to close windows, messages to be displayed to users, data to be validated, and virtually any other type of response imaginable.

HTML 5 Standard Events

The standard HTML 5 events are listed below: Here script indicates a JavaScript function to be executed against that event.

Attribute	**Value**	**Description**
Offline	script	Triggers when the document goes offline
Onabort	script	Triggers on an abort event
Onafterprint	script	Triggers after the document is printed
Onbeforeonload	script	Triggers before the document loads
Onbeforeprint	script	Triggers before the document is printed
Onblur	script	Triggers when the window loses focus
Oncanplay	script	Triggers when media can start play, but might has to stop for buffering
Oncanplaythrough	script	Triggers when media can be played to the end, without stopping for buffering
Onchange	script	Triggers when an element changes
Onclick	script	Triggers on a mouse click
Oncontextmenu	script	Triggers when a context menu is triggered
Ondblclick	script	Triggers on a mouse double-click
Ondrag	script	Triggers when an element is dragged
Ondragend	script	Triggers at the end of a drag operation
Ondragenter	script	Triggers when an element has been dragged to a valid drop target
Ondragleave	script	Triggers when an element is being dragged over a valid drop target
Ondragover	script	Triggers at the start of a drag operation
Ondragstart	script	Triggers at the start of a drag operation
Ondrop	script	Triggers when dragged element is being dropped
Ondurationchange	script	Triggers when the length of the media is changed
Onemptied	script	Triggers when a media resource element suddenly becomes empty.
Onended	script	Triggers when media has reach the end
Onerror	script	Triggers when an error occur
Onfocus	script	Triggers when the window gets focus

Onformchange	script	Triggers when a form changes
Onforminput	script	Triggers when a form gets user input
Onhaschange	script	Triggers when the document has change
Oninput	script	Triggers when an element gets user input
Oninvalid	script	Triggers when an element is invalid
Onkeydown	script	Triggers when a key is pressed
Onkeypress	script	Triggers when a key is pressed and released
Onkeyup	script	Triggers when a key is released
Onload	script	Triggers when the document loads
Onloadeddata	script	Triggers when media data is loaded
Onloadedmetadata	script	Triggers when the duration and other media data of a media element is loaded
Onloadstart	script	Triggers when the browser starts to load the media data
Onmessage	script	Triggers when the message is triggered
Onmousedown	script	Triggers when a mouse button is pressed
Onmousemove	script	Triggers when the mouse pointer moves
Onmouseout	script	Triggers when the mouse pointer moves out of an element
Onmouseover	script	Triggers when the mouse pointer moves over an element
Onmouseup	script	Triggers when a mouse button is released
Onmousewheel	script	Triggers when the mouse wheel is being rotated
Onoffline	script	Triggers when the document goes offline
Onoine	script	Triggers when the document comes online
Ononline	script	Triggers when the document comes online
Onpagehide	script	Triggers when the window is hidden
Onpageshow	script	Triggers when the window becomes visible
Onpause	script	Triggers when media data is paused
Onplay	script	Triggers when media data is going to start playing
Onplaying	script	Triggers when media data has start playing
Onpopstate	script	Triggers when the window's history changes
Onprogress	script	Triggers when the browser is fetching the media data
Onratechange	script	Triggers when the media data's playing rate has changed
Onreadystatechange	script	Triggers when the ready-state changes
Onredo	script	Triggers when the document performs a redo
Onresize	script	Triggers when the window is resized
Onscroll	script	Triggers when an element's scrollbar is being scrolled
Onseeked	script	Triggers when a media element's seeking attribute is no longer true, and the seeking has ended
Onseeking	script	Triggers when a media element's seeking attribute is true, and the seeking has begun
Onselect	script	Triggers when an element is selected
Onstalled	script	Triggers when there is an error in fetching media data
Onstorage	script	Triggers when a document loads
Onsubmit	script	Triggers when a form is submitted
Onsuspend	script	Triggers when the browser has been fetching media data, but stopped before the entire media file was fetched
Ontimeupdate	script	Triggers when media changes its playing position
Onundo	script	Triggers when a document performs an undo
Onunload	script	Triggers when the user leaves the document
Onvolumechange	script	Triggers when media changes the volume, also when volume is set to "mute"

Onwaiting	script	Triggers when media has stopped playing, but is expected to resume

Some Examples:

onclick Event

This is the most frequently used event type which occurs when a user clicks the left button of his mouse. You can put your validation, warning etc., against this event type.

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

      <form>
         <input type="button" onclick="sayHello()" value="Say Hello" />
      </form>

   </body>
</html>
```

Error Handling

Error can occur due to wrong input, from a user, or from an Internet server. It can be syntax error or any unexpected things. There are three types of errors in programming:

(a) Syntax Errors
(b) Runtime Errors
(c) Logical Errors

Syntax Errors

Syntax errors, also called parsing errors are typically coding errors or typing mistakes made by the programmer. Syntax error occur at compile time in traditional programming languages and at interpret time in JavaScript. For example, the following line causes a syntax error because it is missing a closing parenthesis.

```
<script type="text/javascript">
      Document.write("Hello ";
</script>
```

Runtime Errors

Runtime errors, also called exceptions, occur during execution (after compilation/interpretation).

For example, the following line causes a runtime error because here the syntax is correct, but at runtime, it is trying to call a method that does not exist.

```
<script type="text/javascript">
      window.printme();
</script>
```

Exceptions also affect the thread in which they occur, allowing other JavaScript threads to continue normal execution.

Logical Errors

Logic errors can be the most difficult type of errors to track down. These errors are not the result of a syntax or runtime error. Instead, they occur when you make a mistake in the logic that drives your script and you do not get the result you expected.

You cannot catch those errors, because it depends on your business requirement what type of logic you want to put in your program.

The try...catch...finally Statement

JavaScript implements the try...catch...finally construct as well as the throw operator to handle exceptions. You can catch programmer-generated and runtime exceptions, but you cannot catch JavaScript syntax errors. Here is the try...catch...finally block syntax –

```
<script type="text/javascript">
      try {
         // Code to run
         [break;]
      }

      catch ( e ) {
         // Code to run if an exception occurs
         [break;]
      }

      [ finally {
         // Code that is always executed regardless of
         // an exception occurring
      }]
</script>
```

The try block must be followed by either exactly one catch block or one finally block (or one of both). When an exception occurs in the try block, the exception is placed in “e” and the catch block is executed. The optional finally block executes unconditionally after try/catch.

The given below example describes the try catch and finally block

```
<html>
   <head>
      <script type="text/javascript">
            function myFunc()
            {
               var a = 100;

               try {
                  alert("Value of variable a is : " + a );
               }
```

```
                catch ( e ) {
                   alert("Error: " + e.description );
                }

                finally {
                   alert("Finally block will always execute!" );
                }
             }
      </script>

   </head>
   <body>
      <p>Click the following to see the result:</p>

      <form>
         <input type="button" value="Click Me" onclick="myFunc();" />
      </form>
   </body>
</html>
```

How to get the value of textbox in JavaScript

Syntax:

```
var x = document.getElementById("myText").value;
or
var x = document.FormName.TextFieldName.value;
```

Example:

```
<html>
 <head>
 <title>Get value of text box in Javascript</title>

 <script language="javascript" type="text/javascript">
 function GetValue()
 {
  var name = document.getElementById('txtUserName').value;
  var address=document.MyForm.txtaddress.value;
  alert(name);
  alert(address);
 }
 </script>
 </head>
 <body>
 <h1>Get Value of Text Box in JavacScript</h1>
<form name="MyForm">
 <input id="txtUserName" type="text" /><br/>
  <input id="txtaddress" type="text"  name="txtUser"/> <br/>
  <input type="button" value="Btn1" onclick="GetValue()" />
</form>
 </body>
```

</html>

Form Validation

JavaScript provides a way to validate form's data on the client's computer before sending it to the web server.

Form validation is the process of checking that a form has been filled in correctly before it is processed. For example, if your form has a box for the user to type their email address, you might want your form handler to check that they've filled in their address before you deal with the rest of the form. You can check your form from either server side or client side. Client side validation is easier and simpler than server side validation. Client side form validation is usually done using JavaScript. You can validate your form against the following.

- Checking for empty text boxes
- Numbers validation
- Check for letters
- Check the selection made or not
- Email validation
- Password Validation

Here is a simple example of validation.

```
<html>

   <head>
      <title>Form Validation</title>
         <script type="text/javascript">

      // Form validation code will come here.

      function validate()

      {

         if( document.myForm.Name.value == "" )

         {

            alert( "Please provide your name!" );

            document.myForm.Name.focus() ;

            return false;

         }


         if( document.myForm.EMail.value == "" )

         {

            alert( "Please provide your Email!" );

            document.myForm.EMail.focus() ;

            return false;

         }


         if( document.myForm.Zip.value == "" ||
         isNaN( document.myForm.Zip.value ) ||

         document.myForm.Zip.value.length != 5 )

         {

            alert( "Please provide a zip in the format #####." );

            document.myForm.Zip.focus() ;

            return false;

         }


         if( document.myForm.Country.value == "-1" )

         {

            alert( "Please provide your country!" );

            return false;

         }

         return( true );

      }

</script>
   </head>

   <body>
      <form name="myForm" onsubmit="return(validate());">
         <table cellspacing="2" cellpadding="2" border="1">

            <tr>
               <td align="right">Name</td>
               <td><input type="text" name="Name" /></td>
            </tr>

            <tr>
               <td align="right">EMail</td>
               <td><input type="text" name="EMail" /></td>
            </tr>

            <tr>
               <td align="right">Zip Code</td>
               <td><input type="text" name="Zip" /></td>
            </tr>

            <tr>
               <td align="right">Country</td>
               <td>
                  <select name="Country">
                     <option value="-1" selected>[choose yours]</option>
                     <option value="1">USA</option>
                     <option value="2">UK</option>
                     <option value="3">INDIA</option>
                  </select>
               </td>
            </tr>

            <tr>
               <td align="right"></td>
               <td><input type="submit" value="Submit" /></td>
            </tr>

         </table>
      </form>
   </body>
</html>
```

Cookies

Cookies are data, stored in small text files, on your computer. When a web server has sent a web page to a browser, the connection is shut down, and the server forgets everything about the user. Cookies were invented to solve the problem "how to remember information about the user":

→ When a user visits a web page, his name can be stored in a cookie.
→ Next time the user visits the page, the cookie "remembers" his name.

Cookies are saved in name-value pairs like:
Username=Dolly

When a browser request a web page from a server, cookies belonging to the page is added to the request. This way the server gets the necessary data to "remember" information about users.

Create a Cookie with JavaScript
JavaScript can create cookies, read cookies, and delete cookies with the property document.cookie.
With JavaScript, a cookie can be created like this:
document.cookie="username=Dev cpgn";
You can also add an expiry date (in UTC or GMT time). By default, the cookie is deleted when the browser is closed.

Name	Description	
Name="Value"	Specifies the name of the cookie	Required
PATH="path"	Specifies the path of the URLs for which the cookie is valid. If the URL matches both the PATH and the DOMAIN, then the cookie is sent to the server in the request in the request header. (If left unset , the value of the PATH is the same as the document that set the cookie). This may be blank if you want to retrieve the cookie from any directory or page.	Optional
EXPIRES=date	Specifies the expiry date of the cookie. After this date the cookie will no longer be stored by the client or sent to the server. If this is blank the cookie will expires when browser is closed.	Optional
DOMAIN=domain	Specifies the domain portion of the URLs for which the cookie is valid. The default value for this attribute is the domain of the current document setting the cookie.	Optional
SECURE	Specifies that the cookie should only be transmitted over a secure over a secure link	Optional

Example:
document.cookie="username=Dev cpgn; expires=Thu, 18 Dec 2013 12:00:00 GMT; path=/";

Read a Cookie with JavaScript
With JavaScript, cookies can be read like this:
var x = document.cookie;

* document.cookie will return all cookies in one string much like: cookie1=value; cookie2=value; cookie3=value;

Delete a Cookie with JavaScript

Deleting a cookie is very simple. Just set the expires parameter to a passed date:
document.cookie = "username=; expires=Thu, 01 Jan 1970 00:00:00 GMT";
Note that you don't have to specify a cookie value when you delete a cookie.

Creating Cookies Example:

```
<html>
   <head>

      <script type="text/javascript">
             function WriteCookie()
            {
               if( document.myform.customer.value == "" ){
                  alert("Enter some value!");
                  return;
               }
               cookievalue= escape(document.myform.customer.value) + ";";
               document.cookie="name=" + cookievalue;
               document.write ("Setting Cookies : " + "name=" + cookievalue );
            }
      </script>

   </head>

   <body>

      <form name="myform" action="">
         Enter name: <input type="text" name="customer"/>
         <input type="button" value="Set Cookie" onclick="WriteCookie();"/>
      </form>

   </body>
</html>
```

Reading Cookies Example:

```
<html>
   <head>

      <script type="text/javascript">
            function ReadCookie()
            {
               var allcookies = document.cookie;
               document.write ("All Cookies : " + allcookies );

               // Get all the cookies pairs in an array
               cookiearray = allcookies.split(';');

               // Now take key value pair out of this array
               for(var i=0; i<cookiearray.length; i++){
                  name = cookiearray[i].split('=')[0];
                  value = cookiearray[i].split('=')[1];
                  document.write ("Key is : " + name + " and Value is : " + value);
               }
            }
      </script>

   </head>
   <body>

      <form name="myform" action="">
         <p> click the following button and see the result:</p>
         <input type="button" value="Get Cookie" onclick="ReadCookie()"/>
      </form>

   </body>
</html>
```

Setting Cookie Expiry Date:

```
<html>
   <head>

      <script type="text/javascript">
            function WriteCookie()
            {
               var now = new Date();
               now.setMonth( now.getMonth() + 1 );
               cookievalue = escape(document.myform.customer.value) + ";"

               document.cookie="name=" + cookievalue;
               document.cookie = "expires=" + now.toUTCString() + ";"
               document.write ("Setting Cookies : " + "name=" + cookievalue );
            }
      </script>

   </head>
   <body>

      <form name="formname" action="">
         Enter name: <input type="text" name="customer"/>
         <input type="button" value="Set Cookie" onclick="WriteCookie()"/>
      </form>

   </body>
</html>
```

Unit 2: Issue of Web Technology

Architectural issues of web layer

The web layer is also referred to as the UI layer. The web layer is primarily concerned with presenting the user interface and the behavior of the application (handling user interactions/events). While the web layer can also contain logic, core application logic is usually located in the services layer. The three Layers within the Web Layer are:

HTML-The Content Layer: The content layer is where you store all the content that your customers want to read or look at. This includes text and images as well as multimedia. It's also important to make sure that every aspect of your site is represented in the content layer. That way, your customers who have JavaScript turned off or can't view CSS will still have access to the entire site, if not all the functionality.

CSS - the Styles Layer: Store all your styles for your Web site in an external style sheet. This defines the way the pages should look, and you can have separate style sheets for various media types. Store your CSS in an external style sheet so that you can get the benefits of the style layer across the site.

JavaScript - the Behavior Layer: JavaScript is the most commonly used language for writing the behavior layer; ASP, CGI and PHP can also generate Web page behaviors. However, when most developers refer to the behavior layer, they mean that layer that is activated directly in the Web browser - so JavaScript is nearly always the language of choice. You use this layer to interact directly with the DOM or Document Object Model.

Hyper Text Transfer Protocol (HTTP):

HTTP is the basic protocol used by the World Wide Web. HTTP defines how messages are formatted and transmitted, and what actions Web servers and browsers should take in response to various commands. For example, when you enter a URL in your browser, this actually sends an HTTP command to the Web server directing it to fetch and transmit the requested Web page.

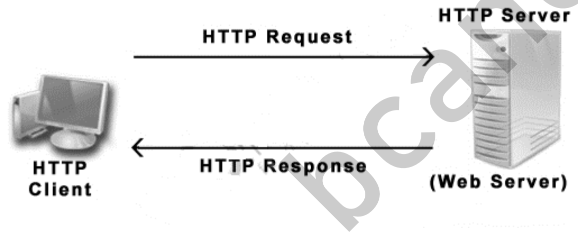


Figure: Internet is based on the client server model

The client, in this case the browser communicates with the server, requesting a web page. The server processes that request and responds by sending the web page contents to the browser.
A browser is works as an HTTP client because it sends requests to an HTTP server which is called Web server. The Web Server then sends responses back to the client. The standard and default port for HTTP servers to listen on is 80 but it can be changed to any other port like 8080 etc.

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

- <u>HTTP is media independent:</u> Any type of data can be sent by HTTP as long as both the client and server know how to handle the data content. How content is handled is determined by the MIME

(Multipurpose Internet Mail Extensions- a specification for formatting non-ASCII messages so that they can be sent over the Internet. Many e-mail clients now support MIME, which enables them to send and receive graphics, audio, and video files via the Internet) specification.

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

File Transfer Protocol (FTP):

FTP is a protocol for exchanging files over the Internet. FTP works in the same way as HTTP for transferring Web pages from a server to a user's browser and SMTP for transferring electronic mail across the Internet in that, like these technologies, FTP uses the Internet's TCP/IP protocols to enable data transfer.

FTP is most commonly used to download a file from a server using the Internet or to upload a file to a server (e.g., uploading a Web page file to a server). Sites that have a lot of downloading (software sites, for example) will often have an FTP server to handle the traffic. If FTP is involved, the URL will have ftp: at the front. FTP can be run in active or passive mode, which determines how the data connection is established.

- Active FTP

Active mode was originally the only method of FTP, and is therefore often the default mode for FTP. In active mode, the client sends the server the IP address and port number on which the client will listen, and the server initiates the TCP connection.

* The client issues a PORT command to the server signaling that it will "actively" provide an IP and port number to open the Data Connection back to the client.

- Passive FTP

In this mode the client sends a PASV command to the server and receives an IP address and port number in return. The client uses these to open the data connection to the server. At the condition when the client is behind a firewall and unable to accept incoming TCP connections, passive mode may be used.

* The client issues a PASV command to indicate that it will wait "passively" for the server to supply an IP and port number, after which the client will create a Data Connection to the server.

Client Server Model

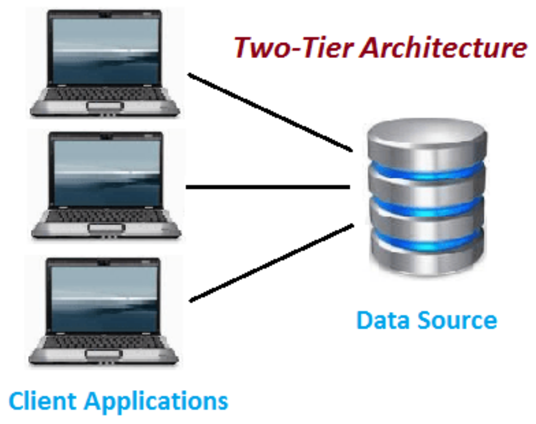
The client–server model is a computing model that acts as distributed application which partitions tasks or workloads between the providers of a resource or service, called servers, and service requesters, called clients. Often clients and servers communicate over a computer network on separate hardware, but both client and server may reside in the same system. A server machine is a host that is running one or more server programs which share their resources with clients. A client does not share any of its resources, but requests a server's content or service function. Clients therefore initiate communication sessions with servers which await incoming requests.

A server machine is a host that is running one or more server programs which share their resources with clients. A client does not share any of its resources, but requests a server's content or service function. Clients therefore initiate communication sessions with servers which await incoming requests.

Client/Server architecture can be of different model based on the number of layers it holds. Some of them are:

- **2-Tier Architecture**

A two-tier client/server is a computing architecture in which an entire application is distributed as two distinct layers or tiers. It divides the application logic, data and processing between client and server devices.
The two-tier is based on Client Server architecture. The two-tier architecture is like client server application. The direct communication takes place between client and server. There is no intermediate between client and server. Because of tight coupling a 2 tiered application will run faster



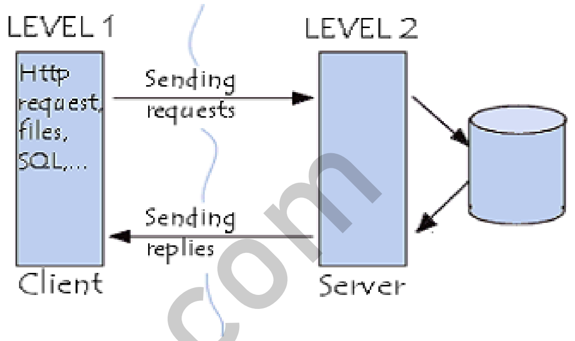


Figure: Two tire architecture

The above figure shows the architecture of two-tier. Here the direct communication between client and server, there is no intermediate between client and server.
The Two-tier architecture is divided into two parts:

1) Client Application (Client Tier)
2) Database (Data Tier)

On client application side the code is written for saving the data in the SQL server database. Client sends the request to server and it process the request & send back with data. The main problem of two tier architecture is the server cannot respond multiple request same time, as a result it cause a data integrity issue.

Advantages:
Easy to maintain and modification is bit easy
Communication is faster

Disadvantages:
In two tier architecture application performance will be degrade upon increasing the users.
Cost-ineffective

- **3-Tier Architecture**

Three-tier architecture typically comprise a presentation tier, a business tier, and a data tier. Three layers in the three tier architecture are as follows:

1) Client layer (Presentation layer)
2) Business layer (Logic layer)
3) Data layer (Data Access Layer)

1) Client layer:

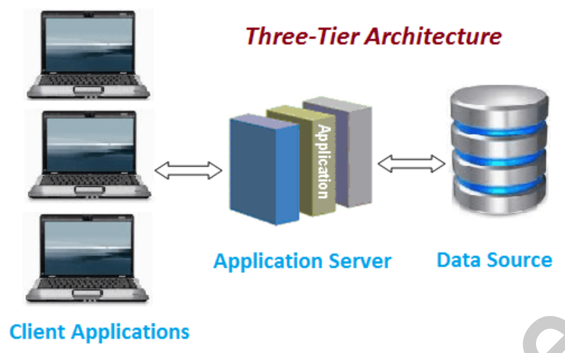
It is also called as Presentation layer which contains user interface part of application. This layer is used for the design purpose where data is presented to the user or input is taken from the user. For example designing registration form which contains text box, label, button etc.

2) Business layer:

In this layer all business logic written like validation of data, calculations, data insertion etc. This acts as an interface between Client layer and Data Access Layer. This layer is also called the intermediary layer helps to make communication faster between client and data layer.

3) Data layer:

In this layer actual database is comes in the picture. Data Access Layer contains methods to connect with database and to perform insert, update, delete, get data from database based on our input data.



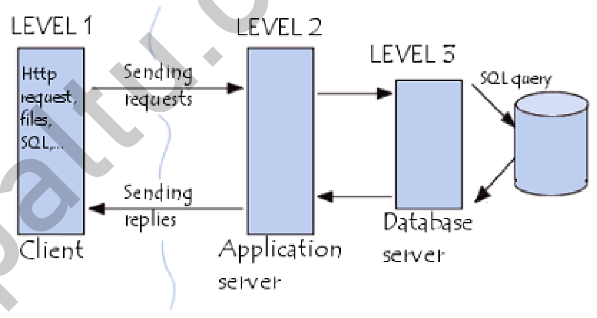


Figure: Three-tier architecture

<u>Advantages</u>

- High performance
- Scalability – Each tier can scale horizontally
- Performance – Because the Presentation tier can cache requests, network utilization is minimized, and the load is reduced on the Application and Data tiers.
- High degree of flexibility in deployment platform and configuration
- Better Re-use
- Improve Data Integrity
- Improved Security – Client is not direct access to database.
- Easy to maintain and modification is bit easy, won't affect other modules
- In three tier architecture application performance is good.

<u>Disadvantages</u>

- Increase Complexity/Effort

- **N-Tier Architecture (multi-tier)**

N-tier architecture (with N more than 3) is really 3 tier architectures in which the middle tier is split up into new tiers. The application tier is broken down into separate parts.
The primary advantage of N-tier architectures is that they make load balancing possible. Since the application logic is distributed between several servers, processing can then be more evenly distributed among those servers. N-tiered architectures are also more easily scalable, since only servers experiencing high demand, such as the application server, need be upgraded. The primary disadvantage of N-tier architectures is that it is also more difficult to program and test an N-tier architecture due to its increased complexity.

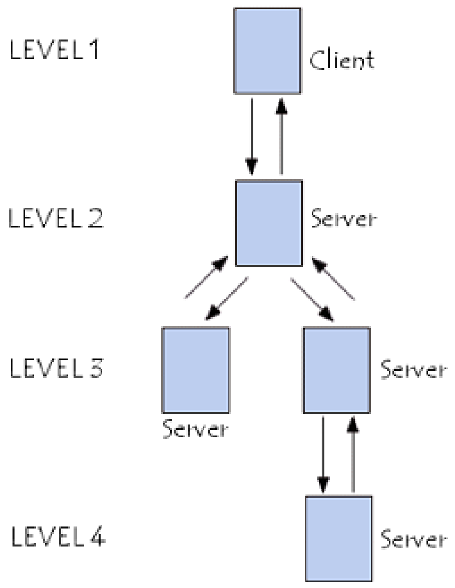


Figure: N-tier architecture

Client/Server Architecture:
Client-server architecture (client/server) is a network architecture in which each computer or process on the network is either a client or a server. Servers are powerful computers or processes dedicated to managing disk drives (file servers), printers (print servers), or network traffic (network servers). Clients are PCs or workstations on which users run applications. Clients rely on servers for resources, such as files, devices, and even processing power.

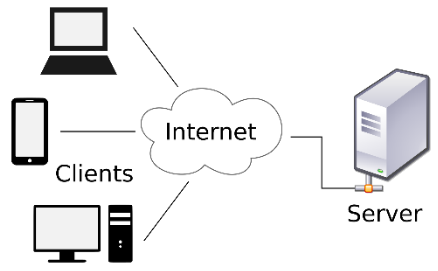


Figure: Client server model

Advantages

1. Centralization: Unlike P2P, where there is no central administration, here in this architecture there is a centralized control. Servers help in administering the whole set-up. Access rights and resource allocation is done by Servers.
2. Proper Management: All the files are stored at the same place. In this way, management of files becomes easy. Also it becomes easier to find files.
3. Back-up and Recovery possible: As all the data is stored on server its easy to make a back-up of it. Also, in case of some break-down if data is lost, it can be recovered easily and efficiently. While in peer computing we have to take back-up at every workstation.
4. Upgradation and Scalability in Client-server set-up: Changes can be made easily by just upgrading the server. Also new resources and systems can be added by making necessary changes in server.
5. Accessibility: From various platforms in the network, server can be accessed remotely.
6. As new information is uploaded in database, each workstation need not have its own storage capacities increased (as may be the case in peer-to-peer systems). All the changes are made only in central computer on which server database exists.
7. Security: Rules defining security and access rights can be defined at the time of set-up of server.
8. Servers can play different roles for different clients.

Disadvantages

1. Failure of the server causes whole network to be collapsed
2. Expensive than P2P, Dedicated powerful servers are needed
3. Extra effort are needed for administering and managing the server.

Difference between 2 tier and 3 tier architecture

S.N	**2 tier**	**3 tier**
1	2Tier is Client server architecture	3Tier is Client, Server and Database architecture
2	In 2Tier client directly get communication to server.	3Tier has a Middle stage to communicate client to server
3	Client -Server Architecture	Web -based application
4	Client will hit request directly to server and client will get response directly from server	Here in between client and server middle ware will be there, if client hits a request it will go to the middle ware and middle ware will send to server and vice versa.
5	2-tier means 1) Design layer 2) Data layer	3-tier means 1) Design layer 2) Business layer or Logic layer 3) Data layer
6	Fig.	Fig.

Unit 3: The Client Tier

Introduction to XML:

- ~ XML stands for EXtensible Markup Language derived from Standard Generalized Markup Language (SGML).
- ~ XML was designed to store, carry and exchange data.
- ~ XML was designed to be both human- and machine-readable.
- ~ XML is not a replacement for HTML
- ~ XML Does Not Use Predefined Tags
- ~ XML is self-descriptive
- ~ XML is Extensible
- ~ XML is a software- and hardware-independent tool for storing and transporting data.
- ~ HTML is about displaying information, while XML is about carrying information.
- ~ XML does not DO anything. XML was created to structure, store, and transport information.

XML is a software- and hardware-independent tool for storing and transporting data. HTML is designed to display data. In contrast, XML is designed to transport and store data. XML stands for EXtensible Markup Language and is much like HTML. XML was designed to carry data, not to display data. XML tags are not predefined. You must define your own tags. XML is designed to be self-descriptive.

Uses of XML

1. <u>XML is an international standard</u>

XML is a document standard that is maintained by the W3C (World Wide Web Consortium), an organization that is responsible for Web standards. XML documents are vendor-neutral, and they are not tied to one application or one company.

2. <u>It Separates Data from HTML</u>

If you need to display dynamic data in your HTML document, it will take a lot of work to edit the HTML each time the data changes. With XML, data can be stored in separate XML files.

3. <u>It Simplifies Data Sharing</u>:

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

4. <u>XML Simplifies Data Transport</u>:

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

5. <u>It Makes Your Data More Available</u>:

Different applications can access your data, not only in HTML pages, but also from XML data sources.

Where XML is used?

- XML can separate data from HTML
- XML is used to exchange data
- XML can be used to share data
- XML can be used to store data
- XML can make your data more useful
- XML can be used to create new *ML language

What is Markup?
Markup is information added to a document that enhances its meaning in certain ways, in that it identifies the parts and how they relate to each other. More specifically, a markup language is a set of symbols that can be placed in the text of a document to demarcate and label the parts of that document. Following example shows how XML markup looks, when embedded in a piece of text:

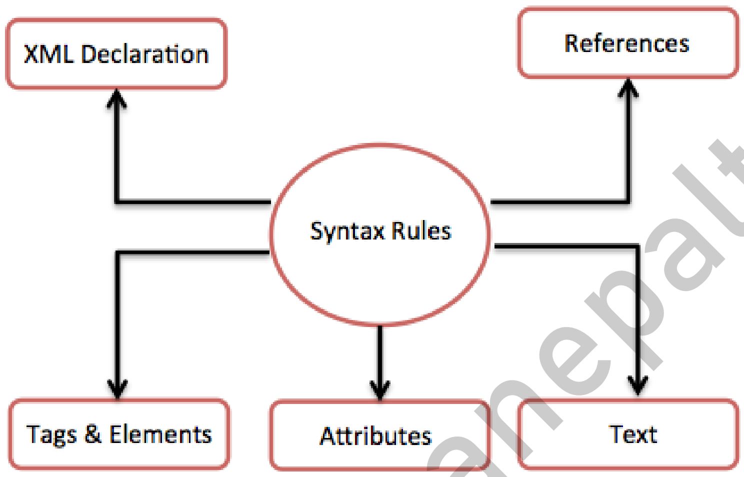
```
<message>
  <text>Hello, world!</text>
</message>
```

Rules for writing XML:

Following is a complete XML document:

```
<?xml version="1.0"?>
<contact-info>
<name>Dev Cpgn</name>
<Address>Kawasoti</Address>
<phone>9867031614</phone>
</contact-info>
```

The following diagram depicts the syntax rules to write different types of markup and text in an XML document.



1) <u>XML Declaration</u>

The XML document can optionally have an XML declaration. It is written as below:

```
<? xml version="1.0" encoding="UTF-8"?>
```

-> This line is also called the XML prolog:

Where version is the XML version and encoding specifies the character encoding used in the document.
Rules for XML Declaration:

~ The XML declaration is case sensitive and must begin with "<?xml>" where "xml" is written in lower-case.
~ If document contains XML declaration, then it strictly needs to be the first statement of the XML document.
~ An HTTP protocol can override the value of encoding that you put in the XML declaration.

2) <u>Tags and Elements</u>

An XML file is structured by several XML-elements, also called XML-nodes or XML-tags. XML-elements' names are enclosed by triangular brackets < > as shown below:

```
<element>
```

Rules for Tags and Elements:

- ~ Each XML-element needs to be closed either with start or with end elements as shown below
- ~ If the XML declaration does not have a closing tag, this is not an error. The declaration is not a part of XML.
  <element>....</element> or in simple-cases, just this way: <element/>

Nesting of elements

An XML-element can contain multiple XML-elements. Following example shows correct nested tags:

```
<?xml version="1.0"?>
<contact-info>
<company>TutorialsPoint</company>
<contact-info>
```

Root element: An XML document can have only one root element. For example, following is not a correct XML document, because both the x and y elements occur at the top level without a root element:

```
<x>...</x>
<y>...</y>
```

The following example shows a correctly formed XML document:

```
<root>
  <x>...</x>
  <y>...</y>
</root>
```

Case sensitivity: The names of XML-elements are case-sensitive. That means the name of the start and the end elements need to be exactly in the same case.

For example <contact-info> is different from <Contact-Info>.

3) Attributes

An attribute specifies a single property for the element, using a name/value pair. An XML-element can have one or more attributes.

Rules for XML Attributes:

- ~ Attribute names in XML (unlike HTML) are case sensitive. That is, HREF and href are considered two different XML attributes.
- ~ Attribute names are defined without quotation marks, whereas attribute values must always appear in quotation marks.

Eg.

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

4) XML References

References usually allow you to add or include additional text or markup in an XML document. References always begin with the symbol "&", which is a reserved character and end with the symbol ";". XML has two types of references:

- Entity References:

Some characters have a special meaning in XML. If you place a character like "<" inside an XML element, it will generate an error because the parser interprets it as the start of a new element.

This will generate an XML error:
<message>salary < 1000</message>
To avoid this error, replace the "<" character with an entity reference:
<message>salary &lt; 1000</message>
There are 5 pre-defined entity references in XML:

not allowed character	**replacement-entity**	**character description**
<	&lt;	less than
>	&gt;	greater than
&	&amp;	ampersand
'	&apos;	apostrophe
"	&quot;	quotation mark

- Character References: These contain references, such as &#65;, contains a hash mark (“#”) followed by a number. The number always refers to the Unicode code of a character. In this case, 65 refers to alphabet "A".

5) XML Text

The names of XML-elements and XML-attributes are case-sensitive, which means the name of start and end elements need to be written in the same case.
To avoid character encoding problems, all XML files should be saved as Unicode UTF-8 or UTF-16 files.
Whitespace characters like blanks, tabs and line-breaks between XML-elements and between the XML-attributes will be ignored.

Comments in XML

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

Well Formed XML

XML documents that conform to the syntax rules above are said to be "Well Formed" XML documents.
A "Well Formed" XML document has correct XML syntax. The syntax rules as described in previous sections are:

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

XML Naming Rules

- XML elements must follow these naming rules:
- Names can contain letters, numbers, and other characters
- Names cannot start with a number or punctuation character

- ~ Names cannot start with the letters xml (or XML, or Xml, etc)
- ~ Names cannot contain spaces

Best Naming Practices

- Make names descriptive. Names with an underscore separator are nice: <first_name>, <last_name>.

- Names should be short and simple, like this: <book_title> not like this: <the_title_of_the_book>.

- Avoid "-" characters. If you name something "first-name," some software may think you want to subtract name from first.

- Avoid "." characters. If you name something "first.name," some software may think that "name" is a property of the object "first."

- Avoid ":" characters. Colons are reserved to be used for something called namespaces (more later).

- XML documents often have a corresponding database. A good practice is to use the naming rules of your database for the elements in the XML documents.

- Non-English letters like éòá are perfectly legal in XML, but watch out for problems if your software vendor doesn't support them.

XML Namespace:

A Namespace is a set of unique names. Namespace is a mechanisms by which element and attribute name can be assigned to group. The Namespace is identified by URI (Uniform Resource Identifiers). A Namspace is declared using reserved attributes. Such an attribute name must either be xmlns or begin with xmlns: shown as below:
<element xmlns:name="URL">

- ¤ The Namespace starts with the keyword **xmlns**.
- ¤ The word **name** is the Namespace prefix.
- ¤ The **URL** is the Namespace identifier.

XML Namespaces provide a method to avoid element name conflicts. In XML, element names are defined by the developer. This often results in a conflict when trying to mix XML documents from different XML applications.
Consider following examples;
This XML carries HTML table information:

```
<table>
  <tr>
    <td>Apples</td>
    <td>Bananas</td>
  </tr>
</table>
```

This another XML carries information about a table (a piece of furniture):

```
<table>
  <name>Dining table</name>
  <width>80</width>
  <length>120</length>
</table>
```

If these XML fragments were added together, there would be a name conflict. Both contain a <table> element, but the elements have different content and meaning.
An XML parser will not know how to handle these differences. Thus XML namespaces are used for providing uniquely named elements and attributes in an XML document.

Example:
Namespace affects only a limited area in the document. An element containing the declaration and all of its descendants are in the scope of the Namespace. Following is a simple example of XML Namespace:

```
<?xml version="1.0" encoding="UTF-8"?>
<cont:contact xmlns:cont="www.devcpgn.blogspot.com/profile">
  <cont:name>Bindu</cont:name>
  <cont:company>Abc</cont:company>
  <cont:phone>9867031614</cont:phone>
</cont:contact>
```

Here, the Namespace prefix is cont, and the Namespace identifier (URI) as www.devcpgn.blogspot.com/profile. This means, the element names and attribute names with the cont prefix (including the contact element), all belong to the www.devcpgn.blogspot.com/profile namespace.

Document Type Declaration (DTD)

The XML Document Type Declaration, commonly known as DTD, is a way to describe XML language precisely. DTDs check vocabulary and validity of the structure of XML documents against grammatical rules of appropriate XML language. A DTD defines the document structure with a list of legal elements and attributes.
An XML DTD can be either specified inside the document, or it can be kept in a separate document and then liked separately.
Basic syntax of a DTD is as follows:

```
<!DOCTYPE element DTD identifier
[
  declaration1
 declaration2
 ........
]>
```

In the above syntax,

- The **DTD** starts with <!DOCTYPE delimiter.
- An **element** tells the parser to parse the document from the specified root element.
- **DTD identifier** is an identifier for the document type definition, which may be the path to a file on the system or URL to a file on the internet. If the DTD is pointing to external path, it is called **External Subset**.
- **The square brackets [ ]** enclose an optional list of entity declarations called Internal Subset.

- ### Internal DTD

A DTD is referred to as an internal DTD if elements are declared within the XML files. To refer it as internal DTD, standalone attribute in XML declaration must be set to yes. This means, the declaration works independent of external source.
The syntax of internal DTD is as shown:

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

where root-element is the name of root element and element-declarations is where you declare the elements.

Following is a simple example of internal DTD:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<!DOCTYPE address [
  <!ELEMENT address (name, company, phone)>
  <!ELEMENT name (#PCDATA)>
  <!ELEMENT company (#PCDATA)>
  <!ELEMENT phone (#PCDATA)>
]>
<address>
  <name>Binod </name>
  <company>Xyz company</company>
  <phone>(011) 123-4567</phone>
</address>
```

So this PCDATA it will give the value of employee:
Binod Xyz company (011) 123-4567

The DTD above is interpreted like this:

- !DOCTYPE address defines that the root element of this document is address
- !ELEMENT address defines that the address element contains three elements: "name, company, phone"
- !ELEMENT name defines the name element to be of type "#PCDATA"
- !ELEMENT company defines the company element to be of type "#PCDATA"
- !ELEMENT phone defines the phone element to be of type "#PCDATA"

Rules

- The document type declaration must appear at the start of the document (preceded only by the XML header) — it is not permitted anywhere else within the document.
- Similar to the DOCTYPE declaration, the element declarations must start with an exclamation mark.
- The Name in the document type declaration must match the element type of the root element.

Example of CDATA:

```
<?xml version="1.0"?>
<!DOCTYPE employee SYSTEM "employee.dtd">
<employee>
<![CDATA[
  <firstname>vimal</firstname>
  <lastname>jaiswal</lastname>
  <email>vimal@javatpoint.com</email>
]]>
</employee>
```

PCDATA - Parsed Character Data.
XML parsers normally parse all the text in an XML document.

- Basically used for ELEMENTS

CDATA – Just a Character Data (Unparsed)
The term CDATA is used about text data that should not be parsed by the XML parser. Characters like "<" and "&" are illegal in XML elements.

- Used for Attributes of XML i.e ATTLIST

* By default, everything is PCDATA

In the above CDATA example, CDATA is used just after the element employee to make the data/text unparsed, so it will give the value of employee:

```
<firstname>vimal</firstname><lastname>jaiswal</lastname><email>vimal@javatpoint.com</e
mail>
```

- **External DTD**

In external DTD elements are declared outside the XML file. They are accessed by specifying the system attributes which may be either the legal .dtd file or a valid URL. To refer it as external DTD, standalone attribute in the XML declaration must be set as no. This means, declaration includes information from the external source.
Following is the syntax for external DTD:

```
<!DOCTYPE root-element SYSTEM "file-name">
```

where file-name is the file with .dtd extension.

Example:

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<!DOCTYPE address SYSTEM "address.dtd">
<address>
  <name>Binod Babu</name>
  <company>Xyz company</company>
  <phone>(011) 123-4567</phone>
</address>
```

The content of the DTD file address.dtd are as shown:

```
<!ELEMENT address (name,company,phone)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT company (#PCDATA)>
<!ELEMENT phone (#PCDATA)>
```

XML - Schemas

XML Schema is commonly known as XML Schema Definition (XSD). It is used to describe and validate the structure and the content of XML data. XML schema defines the elements, attributes and data types. Schema element supports Namespaces. It is similar to a database schema that describes the data in a database.

XML Schema is an XML-based alternative to DTD.
The purpose of an XML Schema is to define the legal building blocks of an XML document, just like a DTD.
An XML Schema:

- defines elements that can appear in a document
- defines attributes that can appear in a document
- defines which elements are child elements
- defines the order of child elements
- defines the number of child elements
- defines whether an element is empty or can include text
- defines data types for elements and attributes
- defines default and fixed values for elements and attributes
- XML Schemas are extensible to future additions

- XML Schemas are richer and more powerful than DTDs
- XML Schemas are written in XML
- XML Schemas support data types
- XML Schemas support namespaces

Why Use XML Schemas?

- It is easier to describe allowable document content
- It is easier to validate the correctness of data
- It is easier to work with data from a database
- It is easier to define data facets (restrictions on data)
- It is easier to define data patterns (data formats)
- It is easier to convert data between different data types

Syntax:

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

Example:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="contact">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="name" type="xs:string" />
            <xs:element name="company" type="xs:string" />
            <xs:element name="phone" type="xs:int" />
        </xs:sequence>
    </xs:complexType>
</xs:element>
</xs:schema>
```

The basic idea behind XML Schemas is that they describe the valid format that an XML document can take.

XSD Simple Elements

A simple element is an XML element that contains only text. It cannot contain any other elements or attributes. The text can be of many different types. It can be one of the types included in the XML Schema definition (Boolean, string, date, etc.), or it can be a custom type that you can define yourself.

You can also add restrictions to a data type in order to limit its content, or you can require the data to match a specific pattern.
The syntax for defining a simple element is:
<xs:element name="xxx" type="yyy"/>
where xxx is the name of the element and yyy is the data type of the element.

XML Schema has a lot of built-in data types. The most common types are:

- xs:string
- xs:decimal
- xs:integer
- xs:boolean

And here are the corresponding simple element definitions:
<xs:element name="lastname" type="xs:string"/>
<xs:element name="age" type="xs:integer"/>
<xs:element name="dateborn" type="xs:date"/>

- xs:date
- xs:time

Example:
Here are some XML elements:

```
<lastname>Refsnes</lastname>
<age>36</age>
<dateborn>1970-03-27</dateborn>
```

Default and Fixed Values for Simple Elements:
Simple elements may have a default value OR a fixed value specified. A default value is automatically assigned to the element when no other value is specified.

In the following example the default value is "red":
<xs:element name="color" type="xs:string" default="red"/>

A fixed value is also automatically assigned to the element, and you cannot specify another value.
In the following example the fixed value is "red":
<xs:element name="color" type="xs:string" fixed="red"/>

XSD Attributes:
Simply attributes are associated with the complex elements. If an element has attributes, it is considered to be of a complex type. Simple elements cannot have attributes. But the attribute itself is always declared as a simple type. All attributes are declared as simple types.

The syntax for defining an attribute is:
<xs:attribute name="xxx" type="yyy"/> , where xxx is the name of the attribute and yyy specifies the data type of the attribute.
Example:

Here is an XML element with an attribute:
<name lname="aryal">Smith</name>

And here is the corresponding attribute definition:
<xs:attribute name="lname" type="xs:string"/>

Default and Fixed Values for Attributes:

Attributes may have a default value OR a fixed value specified.
A default value is automatically assigned to the attribute when no other value is specified.
In the following example the default value is “ENGLISH ":
<xs:attribute name="lang" type="xs:string" default="ENGLISH"/>

A fixed value is also automatically assigned to the attribute, and you cannot specify another value.
In the following example the fixed value is “ENGLISH ":

<xs:attribute name="lang" type="xs:string" fixed="ENGLISH"/>

Restrictions on Content

When an XML element or attribute has a data type defined, it puts restrictions on the element's or attribute's content. If an XML element is of type "xs:date" and contains a string like "Hello World", the element will not validate. With XML Schemas, you can also add your own restrictions to your XML elements and attributes. These restrictions are called **facets**.

XSD Restrictions/ Facets:

1. Restrictions on Values

The following example defines an element called "age" with a restriction. The value of age cannot be lower than 0 or greater than 120:

```
<xs:element name="age">
  <xs:simpleType>
    <xs:restriction base="xs:integer">
      <xs:minInclusive value="0"/>
      <xs:maxInclusive value="120"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

2. Restrictions on a Set of Values

To limit the content of an XML element to a set of acceptable values, we would use the enumeration constraint. The example below defines an element called "car" with a restriction. The only acceptable values are: Audi, Golf, BMW:

```
<xs:element name="car">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:enumeration value="Audi"/>
      <xs:enumeration value="Golf"/>
      <xs:enumeration value="BMW"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

The next example defines an element called "zipcode" with a restriction. The only acceptable value is FIVE digits in a sequence, and each digit must be in a range from 0 to 9:

```
<xs:element name="zipcode">
  <xs:simpleType>
    <xs:restriction base="xs:integer">
      <xs:pattern value="[0-9][0-9][0-9][0-9][0-9]"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

3. Restrictions on a Series of Values

To limit the content of an XML element to define a series of numbers or letters that can be used, we would use the pattern constraint.

The example below defines an element called "letter" with a restriction. The only acceptable value is ONE of the LOWERCASE letters from a to z:

```
<xs:element name="letter">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="[a-z]"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

The next example defines an element called "initials" with a restriction. The only acceptable value is THREE of the UPPERCASE letters from a to z:

```
<xs:element name="initials">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="[A-Z][A-Z][A-Z]"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

4. Restrictions on Whitespace Charact

To specify how whitespace characters should be handled, we would use the whiteSpace constraint. This example defines an element called "address" with a restriction. The whiteSpace constraint is set to "**preserve**", which means that the XML processor WILL NOT remove any white space characters:

```
<xs:element name="address">
<xs:simpleType>
  <xs:restriction base="xs:string">
    <xs:whiteSpace value="preserve"/>
  </xs:restriction>
</xs:simpleType>
</xs:element>
```

whitespace value= "**replace**", the XML processor WILL REPLACE all white space characters (line feeds, tabs, spaces, and carriage returns) with spaces
whiteSpace value= "**collapse**", which means that the XML processor WILL REMOVE all white space characters (line feeds, tabs, spaces, carriage returns are replaced with spaces, leading and trailing spaces are removed, and multiple spaces are reduced to a single space):

5. Restrictions on Length:
To limit the length of a value in an element, we would use the **length, maxLength, and minLength** constraints. This example defines an element called "password" with a restriction. The value must be exactly eight characters:

```
<xs:element name="password">
<xs:simpleType>
  <xs:restriction base="xs:string">
    <xs:length value="8"/>
  </xs:restriction>
</xs:simpleType>
</xs:element>
```

```
<xs:element name="password">
<xs:simpleType>
  <xs:restriction base="xs:string">
    <xs:minLength value="5"/>
    <xs:maxLength value="8"/>
  </xs:restriction>
```

Restrictions for Datatypes:

Constraint	Description
enumeration	Defines a list of acceptable values
fractionDigits	Specifies the maximum number of decimal places allowed. Must be equal to or greater than zero
length	Specifies the exact number of characters or list items allowed. Must be equal to or greater than zero
maxExclusive	Specifies the upper bounds for numeric values (the value must be less than this value)
maxInclusive	Specifies the upper bounds for numeric values (the value must be less than or equal to this value)
maxLength	Specifies the maximum number of characters or list items allowed. Must be equal to or greater than zero
minExclusive	Specifies the lower bounds for numeric values (the value must be greater than this value)
minInclusive	Specifies the lower bounds for numeric values (the value must be greater than or equal to this value)
minLength	Specifies the minimum number of characters or list items allowed. Must be equal to or greater than zero
pattern	Defines the exact sequence of characters that are acceptable
totalDigits	Specifies the exact number of digits allowed. Must be greater than zero
whiteSpace	Specifies how white space (line feeds, tabs, spaces, and carriage returns) is handled

XSD Complex Types:

A complex element is an XML element that contains other elements and/or attributes.
There are four kinds of complex elements:

- empty elements
- elements that contain only other elements
- elements that contain only text
- elements that contain both other elements and text

Note: Each of these elements may contain attributes as well!

Examples of Complex Elements

- A complex XML element, "product", which is empty:
  ```
  <product pid="1345"/>
  ```
- A complex XML element, "employee", which contains only other elements:
  ```
  <employee>
          <firstname>Dev</firstname>
          <lastname>Cpgn</lastname>
   </employee>
  ```
- A complex XML element, "food", which contains only text:
  ```
  <food type="dessert">Ice cream</food>
  ```
- A complex XML element, "description", which contains both elements and text:
  ```
  <description>
  It happened on <date lang="english">12/6/2015</date> ....
  </description>
  ```

How to Define a Complex Element

Look at this complex XML element, "employee", which contains only other elements:

```
      <employee>
   <firstname>John</firstname>
   <lastname>Smith</lastname>
  </employee>
```

We can define a complex element in an XML Schema two different ways:

1. The "employee" element can be declared directly by naming the element, like this:

```
      <xs:element name="employee">
   <xs:complexType>
    <xs:sequence>
     <xs:element name="firstname" type="xs:string"/>
     <xs:element name="lastname" type="xs:string"/>
    </xs:sequence>
   </xs:complexType>
  </xs:element>
```

If you use the method described above, only the "employee" element can use the specified complex type. Note that the child elements, "firstname" and "lastname", are surrounded by the <sequence> indicator. This means that the child elements must appear in the same order as they are declared.

2. The "employee" element can have a **type** attribute that refers to the name of the complex type to use:

```
      <xs:element name="employee" type="personinfo"/>

    <xs:complexType name="personinfo">
      <xs:sequence>
        <xs:element name="firstname" type="xs:string"/>
        <xs:element name="lastname" type="xs:string"/>
      </xs:sequence>
    </xs:complexType>
```

If you use the method described above, several elements can refer to the same complex type, like this:

```
<xs:element name="employee" type="personinfo"/>
<xs:element name="student" type="personinfo"/>
<xs:element name="member" type="personinfo"/>

<xs:complexType name="personinfo">
  <xs:sequence>
    <xs:element name="firstname" type="xs:string"/>
    <xs:element name="lastname" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
```

Types of XSD Complex Elements

1. XSD Empty Element

An empty XML element cannot have contents, only attributes. For eg

```
<product prodid="1345" />
```

The "product" element above has no content at all. To define a type with no content, we must define a type that allows elements in its content, but we do not actually declare any elements, like this:

```
<xs:element name="product">
  <xs:complexType>
    <xs:complexContent>
      <xs:restriction base="xs:integer">
        <xs:attribute name="prodid" type="xs:positiveInteger"/>
      </xs:restriction>
    </xs:complexContent>
  </xs:complexType>
</xs:element>
```

it is possible to declare the "product" element more compactly, like this:

```
<xs:element name="product">
  <xs:complexType>
    <xs:attribute name="prodid" type="xs:positiveInteger"/>
  </xs:complexType>
</xs:element>
```

2. XSD Elements only

An "elements-only" complex type contains an element that contains only other elements. Consider an XML element "person”, that contains only other elements:

An XML element, "person", that contains only other elements:

```
<person>
  <firstname>John</firstname>
  <lastname>Smith</lastname>
</person>
```

Notice the <xs:sequence> tag. It means that the elements defined ("firstname" and "lastname") must appear in that order inside a "person" element.

You can define the "person" element in a schema, like this:
```
<xs:element name="person">
 <xs:complexType>
   <xs:sequence>
     <xs:element name="firstname" type="xs:string"/>
     <xs:element name="lastname" type="xs:string"/>
   </xs:sequence>
 </xs:complexType>
</xs:element>
```

Or you can give the complexType element a name, and let the "person" element have a type attribute that refers to the name of the complexType (if you use this method, several elements can refer to the same complex type):

```
<xs:element name="person" type="persontype"/>

<xs:complexType name="persontype">
 <xs:sequence>
   <xs:element name="firstname" type="xs:string"/>
   <xs:element name="lastname" type="xs:string"/>
 </xs:sequence>
</xs:complexType>
```
3. XSD Text only Elements

A complex text-only element can contain text and attributes. This type contains only simple content (text and attributes), therefore we add a simpleContent element around the content. When using simple content, you must define an extension OR a restriction within the simpleContent element, like this:

```
<xs:element name="somename">
 <xs:complexType>
   <xs:simpleContent>
     <xs:extension base="basetype">
       ....
       ....
     </xs:extension>
   </xs:simpleContent>
 </xs:complexType>
</xs:element>
```

OR
```
<xs:element name="somename">
 <xs:complexType>
   <xs:simpleContent>
     <xs:restriction base="basetype">
       ....
       ....
     </xs:restriction>
   </xs:simpleContent>
 </xs:complexType>
</xs:element>
```

Note: You can use the extension/restriction element to expand or to limit the base simple type for the element.

4. XSD Mixed Content (that contain other element and text)

A mixed complex type element can contain attributes, elements, and text. Consider an XML element, "ordernote", that contains both text and other elements:
```
<ordernnote>
 Dear Mr.<name>Jagdish Bhatta</name>.
 Your gift order for the valentine day with order id
<orderid>9999</orderid>
 will be shipped on <shipdate>2012-02-13</shipdate>.
</ordernnote>
```

The following schema declares the "ordernote" element:

```
<xs:element name="ordernote">
 <xs:complexType mixed="true">
  <xs:sequence>
   <xs:element name="name" type="xs:string"/>
   <xs:element name="orderid" type="xs:positiveInteger"/>
   <xs:element name="shipdate" type="xs:date"/>
  </xs:sequence>
      </xs:complexType>
     </xs:element>
```

Note: To enable character data to appear between the child-elements of "ordernote", the mixed attribute must be set to "true". The <xs:sequence> tag means that the elements defined (name, orderid and shipdate) must appear in that order inside a "ordernote" element.

XSD Indicators:

XSD indicators are used to control how elements are to be used in documents with indicators. There are seven indicators:

1. Order indicators: They contain;

All

Choice

Sequence

2. Occurrence indicators: They include;

maxOccurs

minOccurs

3. Group indicators: They contain;

Group name

attributeGroup name

Order Indicators

Order indicators are used to define the order of the elements.

- All Indicator

The <all> indicator specifies that the child elements can appear in any order, and that each child element must occur only once:

```
<xs:element name="person">
  <xs:complexType>
    <xs:all>
      <xs:element name="firstname" type="xs:string"/>
      <xs:element name="lastname" type="xs:string"/>
    </xs:all>
  </xs:complexType>
</xs:element>
```

Note: When using the <all> indicator you can set the <minOccurs> indicator to 0 or 1 and the <maxOccurs> indicator can only be set to 1 (the <minOccurs> and <maxOccurs> are described later).

- **Choice Indicator**

The <choice> indicator specifies that either one child element or another can occur:

```
<xs:element name="person">
  <xs:complexType>
    <xs:choice>
      <xs:element name="employee" type="employee"/>
      <xs:element name="member" type="member"/>
    </xs:choice>
  </xs:complexType>
</xs:element>
```

- **Sequence Indicator**

The <sequence> indicator specifies that the child elements must appear in a specific order:

```
<xs:element name="person">
   <xs:complexType>
    <xs:sequence>
      <xs:element name="firstname" type="xs:string"/>
      <xs:element name="lastname" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Occurrence Indicators

Occurrence indicators are used to define how often an element can occur.
Note: For all "Order" and "Group" indicators (any, all, choice, sequence, group name, and group reference) the default value for maxOccurs and minOccurs is 1.

- **maxOccurs Indicator**

The <maxOccurs> indicator specifies the maximum number of times an element can occur:

```
<xs:element name="person">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="full_name" type="xs:string"/>
      <xs:element name="child_name" type="xs:string" maxOccurs="10"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

The example above indicates that the "child_name" element can occur a minimum of one time (the default value for minOccurs is 1) and a maximum of ten times in the "person" element.

- **minOccurs Indicator**

The <minOccurs> indicator specifies the minimum number of times an element can occur:

```
<xs:element name="person">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="full_name" type="xs:string"/>
      <xs:element name="child_name" type="xs:string"
      maxOccurs="10" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

The example above indicates that the "child_name" element can occur a minimum of zero times and a maximum of ten times in the "person" element.

Tip: To allow an element to appear an unlimited number of times, use the maxOccurs="unbounded" statement:

Group Indicators

Group indicators are used to define related sets of elements.

Element Groups

Element groups are defined with the group declaration, like this:

```
<xs:group name="groupname">
...
</xs:group>
```

You must define an all, choice, or sequence element inside the group declaration. The following example defines a group named "persongroup", that defines a group of elements that must occur in an exact sequence:

```
<xs:group name="persongroup">
  <xs:sequence>
    <xs:element name="firstname" type="xs:string"/>
    <xs:element name="lastname" type="xs:string"/>
    <xs:element name="birthday" type="xs:date"/>
  </xs:sequence>
</xs:group>
```

After you have defined a group, you can reference it in another definition, like this:

```
<xs:group name="persongroup">
  <xs:sequence>
    <xs:element name="firstname" type="xs:string"/>
    <xs:element name="lastname" type="xs:string"/>
    <xs:element name="birthday" type="xs:date"/>
  </xs:sequence>
</xs:group>

<xs:element name="person" type="personinfo"/>

<xs:complexType name="personinfo">
  <xs:sequence>
    <xs:group ref="persongroup"/>
    <xs:element name="country" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
```

Attribute Groups

Attribute groups are defined with the attributeGroup declaration, like this:

```
<xs:attributeGroup name="groupname">
...
</xs:attributeGroup>
```

The following example defines an attribute group named "personattrgroup":

```
<xs:attributeGroup name="personattrgroup">
  <xs:attribute name="firstname" type="xs:string"/>
  <xs:attribute name="lastname" type="xs:string"/>
  <xs:attribute name="birthday" type="xs:date"/>
</xs:attributeGroup>
```

After you have defined an attribute group, you can reference it in another definition, like this:

```
<xs:attributeGroup name="personattrgroup">
  <xs:attribute name="firstname" type="xs:string"/>
  <xs:attribute name="lastname" type="xs:string"/>
  <xs:attribute name="birthday" type="xs:date"/>
</xs:attributeGroup>

<xs:element name="person">
  <xs:complexType>
    <xs:attributeGroup ref="personattrgroup"/>
  </xs:complexType>
</xs:element>
```

XSL/XSLT:

XSL- EXtensible Stylesheet Language, and is a style sheet language for XML documents.
XSLT- XSL Transformations

HTML uses predefined tags, and the meaning of each tag is well understood.
The <table> tag in HTML defines a table - and a browser knows how to display it.
Adding styles to HTML elements are simple. Telling a browser to display an element in a special font or color, is easy with CSS.

XSL = Style Sheets for XML

CSS = Style Sheets for HTML
XSL = Style Sheets for XML

XML does not use predefined tags (we can use any tag-names we like), and therefore the meaning of each tag is not well understood.
A <table> tag could mean an HTML table, a piece of furniture, or something else - and a browser does not know how to display it.
XSL describes how the XML document should be displayed!

XSLT (XSL Transformations)

- XSLT is a language for transforming XML documents into XHTML documents or to other XML documents.
- XPath is a language for navigating in XML documents.

- XSLT stands for XSL Transformations
- XSLT is the most important part of XSL
- XSLT transforms an XML document into another XML document
- XSLT uses XPath to navigate in XML documents
- XSLT is a W3C Recommendation

The Extensible Stylesheet Language (XSL) has three major subcomponents:
<u>XSL-FO</u>
The Formatting Objects standard. By far the largest subcomponent, this standard gives mechanisms for describing font sizes, page layouts, and other aspects of object rendering.

<u>XSLT</u>
This is the transformation language, which lets you define a transformation from XML into some other format. For example, you might use XSLT to produce HTML or a different XML structure. You could even use it to produce plain text or to put the information in some other document format.

<u>XPath</u>
At bottom, XSLT is a language that lets you specify what sorts of things to do when a particular element is encountered. But to write a program for different parts of an XML data structure, you need to specify the part of the structure you are talking about at any given time. XPath is that specification language. It is an addressing mechanism that lets you specify a path to an element so that, for example, <article><title> can be distinguished from <person><title>. In that way, you can describe different kinds of translations for the different <title>elements.

Correct Style Sheet Declaration
The root element that declares the document to be an XSL style sheet is <xsl:stylesheet> or <xsl:transform>.
Note: <xsl:stylesheet> and <xsl:transform> are completely synonymous and either can be used!
The correct way to declare an XSL style sheet according to the W3C XSLT Recommendation is:
<xsl:stylesheet version="1.0"xmlns:xsl="http://www.w3.org/1999/XSL/Transform"> or:

```
<xsl:transform version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

To get access to the XSLT elements, attributes and features we must declare the XSLT namespace at the top of the document.
The xmlns:xsl="http://www.w3.org/1999/XSL/Transform" points to the official W3C XSLT namespace. If you use this namespace, you must also include the attribute version="1.0".

XSLT <xsl:template> Element

An XSL style sheet consists of one or more set of rules that are called templates. A template contains rules to apply when a specified node is matched.

The <xsl:template> element is used to build templates.
The **match** attribute is used to associate a template with an XML element. The match attribute can also be used to define a template for the entire XML document. The value of the match attribute is an XPath expression (i.e. match="/" defines the whole document).

The <xsl:value-of> Element
The <xsl:value-of> element can be used to extract the value of an XML element and add it to the output stream of the transformation:

The <xsl:for-each> Element
The <xsl:for-each> element allows you to do looping in XSLT.
The XSL <xsl:for-each> element can be used to select every XML element of a specified node-set:

Example:
We want to transform the following XML document ("cdcatalog.xml") into XHTML:

```
<?xml version="1.0" encoding="UTF-8"?>
<catalog>
  <cd>
    <title>Yeti Dherai maya </title>
    <artist>Narayan Gopal</artist>
    <country>Nepal</country>
    <company>Media Hub</company>
    <price>10.90</price>
    <year>1985</year>
  </cd>
.
.
</catalog>
```

Create an XSL Style Sheet

Then you create an XSL Style Sheet ("cdcatalog.xsl") with a transformation template:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
  <html>
  <body>
  <h2>My CD Collection</h2>
  <table border="1">
    <tr bgcolor="#9acd32">
      <th>Title</th>
      <th>Artist</th>
    </tr>
    <xsl:for-each select="catalog/cd">
    <tr>
      <td><xsl:value-of select="title"/></td>
      <td><xsl:value-of select="artist"/></td>
    </tr>
    </xsl:for-each>
  </table>
  </body>
  </html>
</xsl:template>
</xsl:stylesheet>
```

Link the XSL Style Sheet to the XML Document

Add the XSL style sheet reference to your XML document ("cdcatalog.xml"):

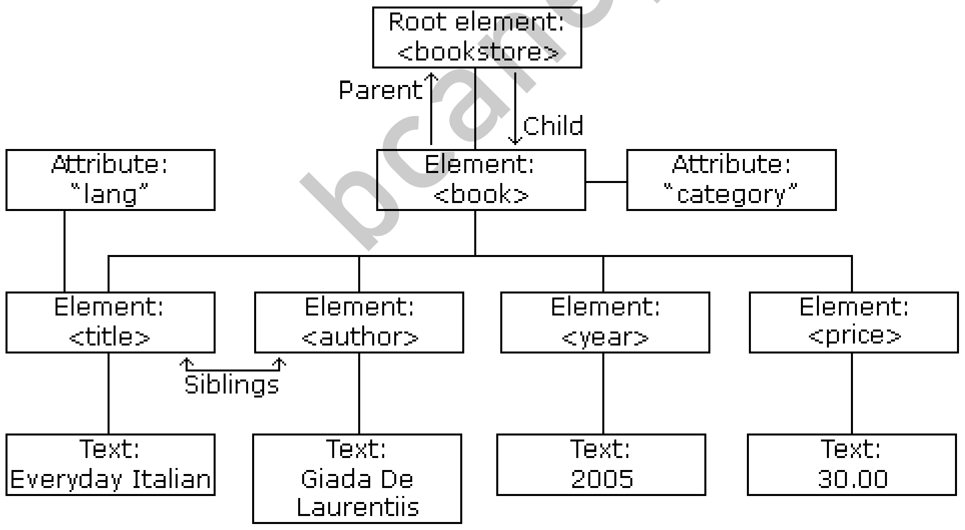
```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="cdcatalog.xsl"?>
<catalog>
  <cd>
<title>Yeti Dherai maya </title>
    <artist>Narayan Gopal</artist>
    <country>Nepal</country>
    <company>Media Hub</company>
    <price>10.90</price>
    <year>1985</year>
  </cd>
.
.
</catalog>
```

Example Explained

- Since an XSL style sheet is an XML document, it always begins with the XML declaration: <?xml version="1.0" encoding="UTF-8"?>.
- The next element, <xsl:stylesheet>, defines that this document is an XSLT style sheet document (along with the version number and XSLT namespace attributes).
- The <xsl:template> element defines a template. The match="/" attribute associates the template with the root of the XML source document.
- The content inside the <xsl:template> element defines some HTML to write to the output.
- The **select** attribute, in the example above, contains an XPath expression. An XPath expression works like navigating a file system
- The value of the **select** attribute is an XPath expression. An XPath expression works like navigating a file system

XML DOM

- A DOM (Document Object Model) defines a standard way for accessing and manipulating documents.
- The XML DOM defines a standard way for accessing and manipulating XML documents.
- The XML DOM views an XML document as a tree-structure.
- All elements can be accessed through the DOM tree. Their content (text and attributes) can be modified or deleted, and new elements can be created. The elements, their text, and their attributes are all known as nodes.
- The nodes of every document are organized in a tree structure, called the DOM tree.

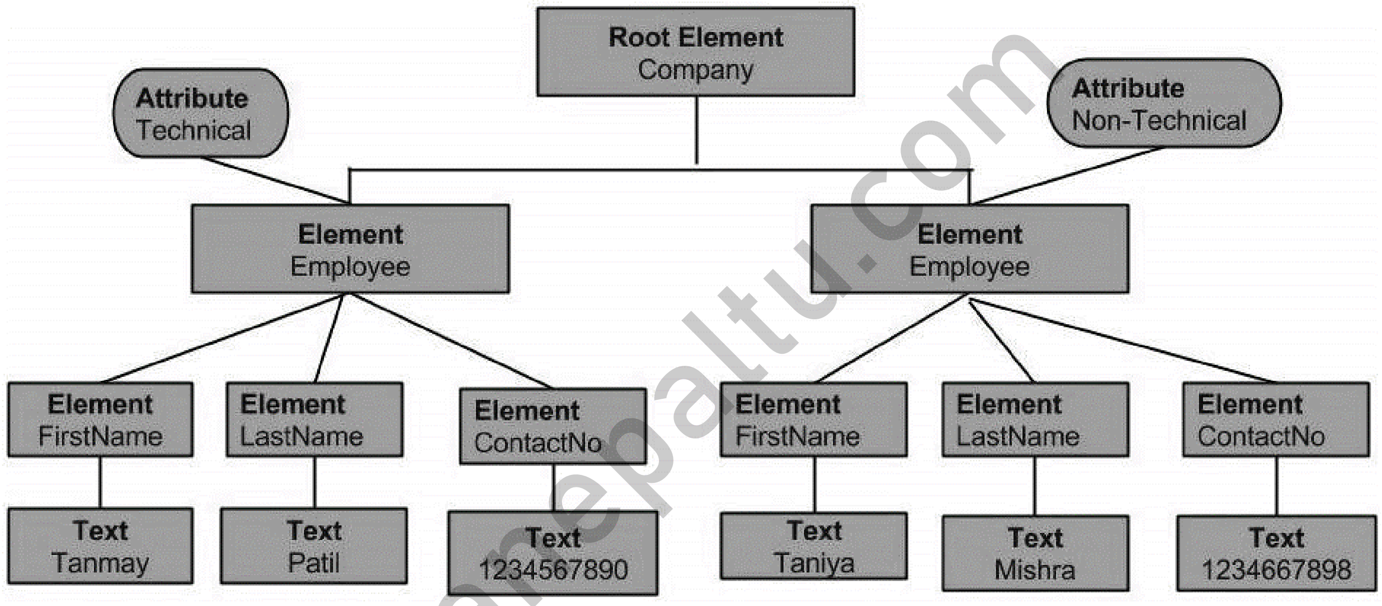


The XML Document Object Model (DOM) class is an in-memory representation of an XML document. The DOM allows you to programmatically read, manipulate, and modify an XML document.

It is the common and structured way that XML data is represented in memory, although the actual XML data is stored in a linear fashion when in a file or coming in from another object. The following is XML data.

```
<?xml version="1.0"?>
<Company>
   <Employee category="Technical">
      <FirstName>Tanmay</FirstName>
      <LastName>Patil</LastName>
      <ContactNo>1234567890</ContactNo>
   </Employee>
   <Employee category="Non-Technical">
      <FirstName>Taniya</FirstName>
      <LastName>Mishra</LastName>
      <ContactNo>1234667898</ContactNo>
   </Employee>
</Company>
```

The following illustration shows how memory is structured when this XML data is read into the DOM structure.



- The topmost node of a tree is called the **root**. The **root** node is <Company> which in turn contains the two nodes of <Employee>. These nodes are referred to as child nodes.
- The child node <Employee> of root node <Company>, in turn consists of its own child node (<FirstName>, <LastName>, <ContactNo>).
- The two child nodes, <Employee> have attribute values Technical and Non-Technical, are referred as *attribute nodes.*
- The text within the every node is called as *text node*.

In the HTML DOM (Document Object Model), everything is a node:

- The document itself is a document node
- All HTML elements are element nodes
- All HTML attributes are attribute nodes

- Text inside HTML elements are text nodes
- Comments are comment nodes

Advantages of DOM

- XML DOM is language and platform independent.
- XML DOM is traversable - Information in XML DOM is organized in a hierarchy which allows developer to navigate around the hierarchy looking for specific information.
- XML DOM is modifiable - It is dynamic in nature providing developer a scope to add, edit, move or remove nodes at any point on the tree.
- You can append, delete or update a child node because data is available in the memory.

Disadvantages of DOM

- It consumes more memory (if the XML structure is large) as program written once remains in memory all the time until and unless removed explicitly.
- Due to the larger usage of memory its operational speed, compared to SAX is slower.
- if the XML contains a large data, then it will be very expensive
- the whole XML is loaded to memory although you are looking for something particular

SAX (Simple API for XML)

It is an event-driven online algorithm for parsing XML documents, with an API interface. SAX provides a mechanism for reading data from an XML document that is an alternative to that provided by the Document Object Model (DOM). Where the DOM operates on the document as a whole, SAX parsers operate on each piece of the XML document sequentially.

Advantages of SAX:

~ Do not need to process and store the entire document (low memory requirement)
~ Can quickly skip over parts not of interest
~ Fast processing

Disadvantages of SAX:

~ Only traverse the document once

Both SAX and DOM are used to parse the XML document

SAX	DOM
It stands for Simple API for XML	It stands for Document Object Model
SAX is event-based	DOM is tree model
In SAX, events are triggered when the XML is being parsed	In DOM, there are no events triggered while parsing.
Parses node by node	Stores the entire XML document into memory before processing
Doesn’t store the XML in memory	Occupies more memory
We can’t insert or delete a node	We can insert or delete nodes
Top to bottom traversing	Traverse in any direction.

Difference between DTD and XSD

	DTD	**XSD**
Abbreviation	DTD stands for Document Type Definition	XSD stands for XML Schema Definition
Uses	DTD is more suitable for small XML data example: bookname, companyname etc.	XSD is used in large XML data example: ADO.Net Dataset, Web services
Strongly/weakly Typed	DTD is weakly typed DTD lacks strong typing capabilities, and has no way of validating the content to data types.	XML Schema is strongly typed An XML Schema can define the data type of certain elements within specific length or values. This ability ensure that the data stored in the XML document is accurate.
Provisions of Inline Definition	DTD allows inline definitions	XML Schema does not allow inline definitions
extensible	DTD is not extensible	XML schema are extensible
Datatype support	DTD doesn't support any datatypes.	XML schemas define datatypes for elements and attributes

	The purpose of a DTD is to define the structure of an XML document. It defines the structure with a list of legal elements:
	
	`<!ATTLIST contact type CDATA #IMPLIED>`
	`<!ELEMENT address1 ( #PCDATA)>`
	`<!ELEMENT city ( #PCDATA)>`
	`<!ELEMENT state ( #PCDATA)>`
	`<!ELEMENT zip ( #PCDATA)>`
		XML Schema enables schema authors to specify that element quantity's data must be numeric or, even more specifically, an integer. Example:
		
		`<xs:element name="note">`
		`<xs:complexType>`
		`  <xs:sequence>`
		`   <xs:element name="address1" type="xs:string"/>`
		`   <xs:element name="city" type="xs:string"/>`
		`   <xs:element name="state" type="xs:string"/>`
		`   <xs:element name="zip" type="xs:string"/>`
		`  </xs:sequence>`
		`</xs:complexType>`

The difference between HTML and XML

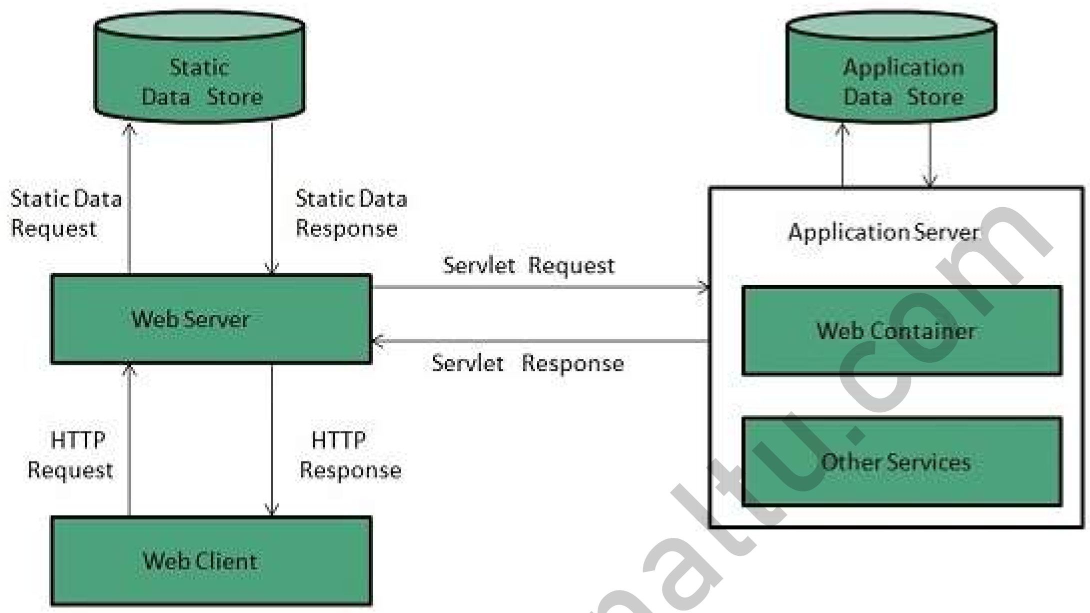
S.N	**HTML**	**XML**
1	Markup language for displaying web pages in a web browser. Designed to display data with focus on how the data looks	Markup language defines a set of rules for encoding documents that can be read by both humans and machines. Designed with focus on storing and transporting data.
2	HTML has a very strict set of predefined elements.	XML has an infinite number of possible elements
3	An HTML document can be XML	But an XML document can't be HTML unless it uses the named HTML elements and served as XHTML.
4	No strict rules. Browser will still generate data to the best of its ability	Strict rules must be followed or processor will terminate processing the file
5	HTML is Static	XML is dynamic
6	It displays a web page	It transport data between application and the database.
	HTML was designed to display data	XML was not designed to display data

Unit 4: The server tier

Web Server Concept

Web server is a computer where the web content is stored. Basically web server is used to host the web sites but there exists other web servers also such as gaming, storage, FTP, email etc.

* Web site is collection of web pages while web server is a software that respond to the request for web resources.



- When client sends request for a web page, the web server search for the requested page if requested page is found then it will send it to client with an HTTP response.
- If the requested web page is not found, web server will the send an **HTTP response: Error 404 Not found.**
- If client has requested for some other resources then the web server will contact to the application server and data store to construct the HTTP response.

Examples of web server

S. N.	Web Server Description
1	**Apache HTTP Server** This is the most popular web server in the world developed by the Apache Software Foundation. Apache web server is an open source software and can be installed on almost all operating systems including Linux, UNIX, Windows, FreeBSD, Mac OS X and more. About 60% of the web server machines run the Apache Web Server.

2.	**Internet Information Services (IIS)** The Internet Information Server (IIS) is a high performance Web Server from Microsoft. This web server runs on Windows NT/2000 and 2003 platforms (and may be on upcoming new Windows version also). IIS comes bundled with Windows NT/2000 and 2003; Because IIS is tightly integrated with the operating system so it is relatively easy to administer it.
3.	**Lighttpd** The lighttpd, pronounced lighty is also a free web server that is distributed with the FreeBSD operating system. This open source web server is fast, secure and consumes much less CPU power. Lighttpd can also run on Windows, Mac OS X, Linux and Solaris operating systems.
4.	**Sun Java System Web Server** This web server from Sun Microsystems is suited for medium and large web sites. Though the server is free it is not open source. It however, runs on Windows, Linux and UNIX platforms. The Sun Java System web server supports various languages, scripts and technologies required for Web 2.0 such as JSP, Java Servlets, PHP, Perl, Python, and Ruby on Rails, ASP and Coldfusion etc.
5.	**Jigsaw Server** Jigsaw (W3C's Server) comes from the World Wide Web Consortium. It is open source and free and can run on various platforms like Linux, UNIX, Windows, and Mac OS X Free BSD etc. Jigsaw has been written in Java and can run CGI scripts and PHP programs.

Using control flow to control dynamic content generation

Dynamic content in the context of HTML and the World Wide Web refers to website content that constantly or regularly changes based on user interactions, timing and other parameters that determine what content is delivered to the user. This means that the content of the site may differ for every user because of different parameters. Facebook is an excellent example of a site that delivers dynamic content, as every user gets different content based on their friends and social interactions, although the layout stays generally the same.
This could be different text, video, advertisements or even an entirely different layout and color. Any element in a page which contains movement and can change over time may be considered as dynamic content.
There are two ways to provide dynamic content:

1. **Using client-side scripting** and frameworks such as JavaScript, AJAX and Bootstrap to change the UI behavior within a specific Web page in response to specific user actions and timings. This gives dynamic behavior to the UI

presentation. This is normally used in Web applications and interactive websites.

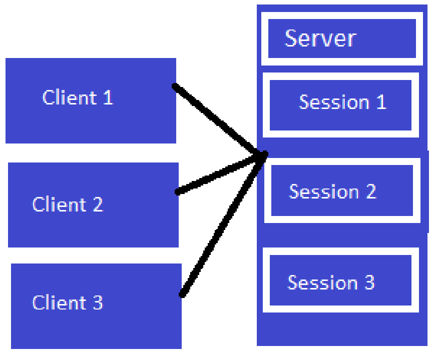
2. **Using server-side scripting** and processing to dynamically form both the layout and content to be delivered to the user based on parameters such as the user's location, time of day, browser being used or user preferences. Some good examples of this are social networking sites and content delivery sites. Social networking sites such as Facebook and Twitter provide entirely different content per user because of the difference of their connections and subscribed services, while sites like YouTube and Amazon provide dynamic content based on user-specific preferences based on past purchases or views; the server gives suggested items or content that the user may like based on historical data.

Sessions and State

A session is a way to store information (in variables) to be used across multiple pages.

Unlike a cookie, the information is not stored on the users computer.
We all know that the web uses the HTTP protocol and the HTTP protocol is a stateless protocol; in other words, when a client sends a request to the server, an instance of the page is created and the page is converted to HTML format and then the server provides the response and then the instance of the page and the value of the control are destroyed. So if we have a requirement to store the values of the controls and pass them into another web form then a State Management Technique is used.

Session is a State Management Technique. A Session can store the value on the Server. It can support any type of object to be stored along with our own custom objects. A session is one of the best techniques for State Management because it stores the data as client-based, in other words the data is stored for every user separately and the data is secured also because it is on the server.



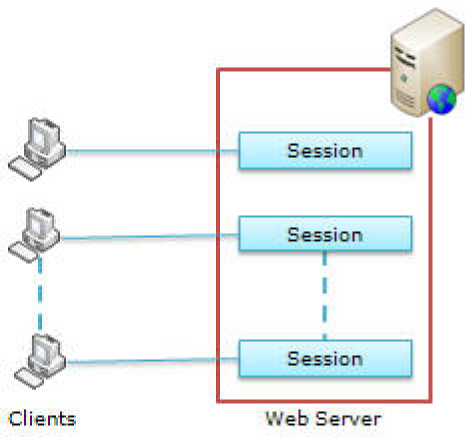


Fig: For every client, session data is stored separately

State management using session is one of the best ASP.NET features, because it is secure, transparent from users, and we can store any kind of object in it. Along with these advantages, some times session can cause performance issues in high traffic sites because it is stored in server memory and clients read data from the server. Now let's have a look at the advantages and disadvantages of using session in our web applications.

Advantages:

- It helps maintain user state and data all over the application.
- It is easy to implement and we can store any kind of object eg. Database, dataset etc.
- Stores client data separately.
- Session is secure and transparent from the user.

Disadvantages:

- As session is stored on the server memory, it is not advisable to use session state when there is a large volume of data.
- With the use of session state, it will affect the performance of memory, because it is stored in server memory until you destroy the state
- The problem with sessions is that when you close your browser you also lose the session so, if you had a site requiring a login, this couldn't be saved as a session like it could as a cookie, and the user would be forced to re-login every time they visit.

Storing and retrieving values from Session

The following code is used for storing a value to session:

```
//Storing UserName in Session
Session["UserName"] = txtUser.Text;
```

Now, let's see how we can retrieve values from session:

```
//Check weather session variable null or not
if (Session["UserName"] != null)
{
   //Retrieving UserName from Session
   lblWelcome.Text = "Welcome : " + Session["UserName"];
}
else
{
 //Do Something else
}
```

Session ID

When a client communicates with a server, only the session ID is transmitted between them. When the client requests for data, ASP.NET looks for the session ID and retrieves the corresponding data. This is done in the following steps:

- Client hits the web site and information is stored in the session.
- Server creates a unique session ID for that client and stores it in the Session State Provider.
- The client requests for some information with the unique session ID from the server.
- Server looks in the Session Providers and retrieves the serialized data from the state server and type casts the object.

Take a look at the the pictorial flow:

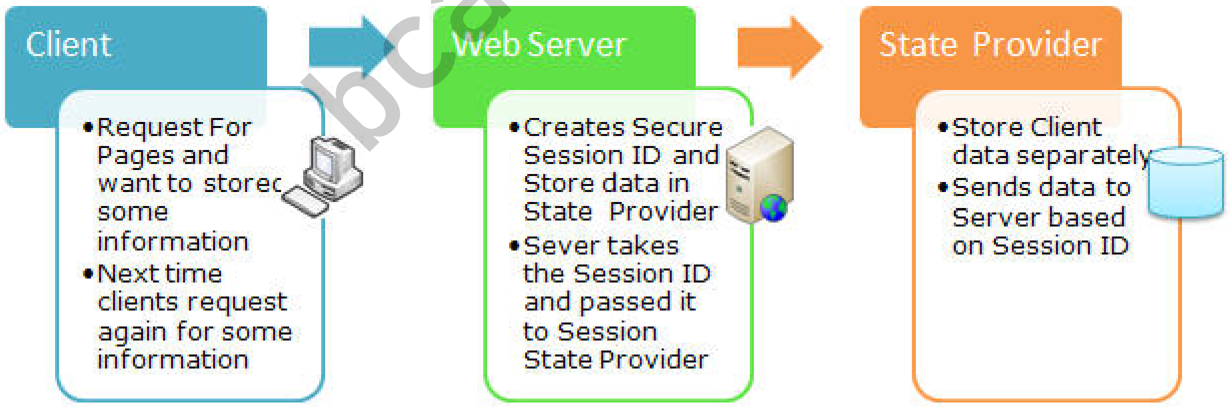

Client
•Request For Pages and want to storec some information
•Next time clients request again for some information
Web Server
•Creates Secure Session ID and Store data in State Provider
•Sever takes the Session ID and passed it to Session State Provider
State Provider
•Store Client data separately
•Sends data to Server based on Session ID

Fig: Communication of client, web server, and State Provider

Session Event

There are two types of session events available in ASP.NET:

- Session_Start
- Session_End

```
void Session_Start(object sender, EventArgs e)
{
    // Code that runs when a new session is started
}
void Session_End(object sender, EventArgs e)
{
    // Code that runs when a session ends.
}
```

Error Handling

There are three types of error in the programming.

a) Syntax Error
b) Logical Error
c) Runtime error

When executing JavaScript code, different errors can occur.
Error can be handled by try catch and finally block.

```
try {
      Block of code to try
}
catch(err) {
    Block of code to handle errors
}
finally {
    Block of code to be executed regardless of the try / catch result
}
```

- The **try** statement lets you test a block of code for errors.
- The **catch** statement lets you handle the error.
- The **throw** statement lets you create custom errors.
- The **finally** statement lets you execute code, after try and catch, regardless of the result.

Example:

```
<!DOCTYPE html>
<html>
<body>
<p>Please input a number between 5 and 10:</p>
<input id="demo" type="text">
<button type="button" onclick="myFunction()">Test Input</button>
<p id="p01"></p>

<script>
function myFunction() {
    var message, x;
    message = document.getElementById("p01");
    message.innerHTML = "";
    x = document.getElementById("demo").value;
    try {
        if(x == "") throw "empty";
        if(isNaN(x)) throw "not a number";
        x = Number(x);
        if(x < 5) throw "too low";
        if(x > 10) throw "too high";
    }
    catch(err) {
        message.innerHTML = "Input is " + err;
    }
}
</script>

</body>
</html>
```

<u>Architecting Web Application</u>
Web application architecture defines the interactions between applications, middleware systems and databases to ensure multiple applications can work together. When a user types in a URL and taps "Go," the browser will find the Internet-facing computer the website lives on and requests that particular page.

The server then responds by sending files over to the browser. After that action, the browser executes those files to show the requested page to the user. Now, the user gets to interact with the website. Of course, all of these actions are executed within a matter of seconds. Otherwise, users wouldn't bother with websites.

What's important here is the code, which has been parsed by the browser. This very code may or may not have specific instructions telling the browser how to react to a wide swath of inputs. As a result, web application architecture includes all sub-components and external applications interchanges for an entire software application.
Of course, it is designed to function efficiently while meeting its specific needs and goals. Web application architecture is critical since the majority of global network traffic, and every single app and device uses web-based communication. It deals with scale, efficiency, robustness, and security.

Best Practices for Good Web Application Architecture

You may have a working app, but it also needs to have good web architecture. Here are several attributes necessary for good web application architecture:

- Solves problems consistently and uniformly
- Make it as simple as possible
- Supports the latest standards include
- Offers fast response times
- Utilizes security standards to reduce the chance of malicious penetrations
- Does not crash
- Heals itself
- Does not have a single point of failure
- Scales out easily
- Allows for easy creation of known data
- Errors logged in a user-friendly way
- Automated deployments

Using tag libraries

In a Web application, a common design goal is to separate the display code from business logic. Java tag libraries are one solution to this problem. Tag libraries allow you to isolate business logic from the display code by creating a Tag class (which performs the business logic) and including an HTML-like tag in your JSP page. When the Web server encounters the tag within your JSP page, the Web server will call methods within the corresponding Java Tag class to produce the required HTML content.
Tag libraries were designed so that Java code could be executed within a JSP page without using Java script blocks.
It separates the display code from business logic. Instead of script blocks, tag libraries allow you to create custom HTML-like tags that map to a Java class that performs the business logic. Groups of these HTML-like tags are called tag libraries.

Unit 5: Introduction to Advanced Server Side Issues

Database Connectivity
In working with databases, the following are the concepts which are common to all databases. To work with the data in a database, the first step is the connection. The connection to a database normally consists of the given parameters.

Database name or Data Source – The first important parameter is the database name to which the connection needs to be established. Each connection can only work with one database at a time.

Credentials – The next important aspect is the username and password which needs to be used to establish a connection to the database. It ensures that the username and password have the necessary privileges to connect to the database.

Optional parameters - For each database type, you can specify optional parameters to provide more information on how .net should handle the connection to the database. For example, one can specify a parameter for how long the connection should stay active. If no operation is performed for a specific period of time, then the parameter would determine if the connection has to be closed.

Selecting data from the database – Once the connection has been established, the next important aspect is to fetch the data from the database. C# can execute 'SQL' select command against the database. The 'SQL' statement can be used to fetch data from a specific table in the database.

Inserting data into the database – C# can also be used to insert records into the database. Values can be specified in C# for each row that needs to be inserted into the database.

Updating data into the database – C# can also be used to update existing records into the database. New values can be specified in C# for each row that needs to be updated into the database.

Deleting data from a database – C# can also be used to delete records into the database. Select commands to specify which rows need to be deleted can be specified in C#.

Steps to creating Simple User Registration

1. Create a database
2. Creating table with five fields in database
    - username
    - email
    - password
3. Create form in html
4. Adding styles to form
5. Connect to the database using ASP.net or PHP
6. Writing logic for user registration script

Creating an SQL statement

What is SQL?

- SQL stands for Structured Query Language
- SQL lets you access and manipulate databases
- SQL became a standard of the American National Standards Institute (ANSI) in 1986, and of the International Organization for Standardization (ISO) in 1987

What Can SQL do?

- ◊ SQL can execute queries against a database
- ◊ SQL can retrieve data from a database
- ◊ SQL can insert records in a database
- ◊ SQL can update records in a database
- ◊ SQL can delete records from a database
- ◊ SQL can create new databases
- ◊ SQL can create new tables in a database
- ◊ SQL can create stored procedures in a database
- ◊ SQL can create views in a database
- ◊ SQL can set permissions on tables, procedures, and views

Some of The Most Important SQL Commands
SELECT - extracts data from a database
UPDATE - updates data in a database
DELETE - deletes data from a database
INSERT INTO - inserts new data into a database
CREATE DATABASE - creates a new database
ALTER DATABASE - modifies a database
CREATE TABLE - creates a new table
ALTER TABLE - modifies a table
DROP TABLE - deletes a table

The SQL CREATE DATABASE Statement
The CREATE DATABASE statement is used to create a new SQL database.

Syntax:
CREATE DATABASE databasename;

The SQL DROP DATABASE Statement
The DROP DATABASE statement is used to drop an existing SQL database.

Syntax
DROP DATABASE databasename;
The SQL SELECT Statement
The SELECT statement is used to select data from a database.
SELECT Syntax:
SELECT column1, column2, ...
FROM table_name;

The SQL SELECT TOP Clause
The SELECT TOP clause is used to specify the number of records to return.

The SELECT TOP clause is useful on large tables with thousands of records. Returning a large number of records can impact on performance.

Note: Not all database systems support the SELECT TOP clause. MySQL supports the LIMIT clause to select a limited number of records, while Oracle uses ROWNUM.

SQL Server / MS Access Syntax:

```
SELECT TOP number|percent column_name(s)
FROM table_name
WHERE condition;
```

The SQL SELECT DISTINCT Statement
The SELECT DISTINCT statement is used to return only distinct (different) values.

Inside a table, a column often contains many duplicate values; and sometimes you only want to list the different (distinct) values.
SELECT DISTINCT Syntax:

```
SELECT DISTINCT column1, column2, ...
FROM table_name;
```

The SQL WHERE Clause
The WHERE clause is used to filter records.

The WHERE clause is used to extract only those records that fulfill a specified condition.

WHERE Syntax:

```
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

The SQL ORDER BY Keyword
The ORDER BY keyword is used to sort the result-set in ascending or descending order.

The ORDER BY keyword sorts the records in ascending order by default. To sort the records in descending order, use the DESC keyword.

ORDER BY Syntax:

```
SELECT column1, column2, ...
FROM table_name
ORDER BY column1, column2, ... ASC|DESC;
```

The SQL CREATE TABLE Statement
The CREATE TABLE statement is used to create a new table in a database.

Syntax:

```
CREATE TABLE table_name (
    column1 datatype,
    column2 datatype,
    column3 datatype,
   ....
```

);

The SQL INSERT INTO Statement
The INSERT INTO statement is used to insert new records in a table.

INSERT INTO Syntax
It is possible to write the INSERT INTO statement in two ways.

The first way specifies both the column names and the values to be inserted:

```
INSERT INTO table_name (column1, column2, column3, ...)
VALUES (value1, value2, value3, ...);
```

If you are adding values for all the columns of the table, you do not need to specify the column names in the SQL query. However, make sure the order of the values is in the same order as the columns in the table. The INSERT INTO syntax would be as follows:

```
INSERT INTO table_name
VALUES (value1, value2, value3, ...);
```

The SQL UPDATE Statement
The UPDATE statement is used to modify the existing records in a table.

UPDATE Syntax:
```
UPDATE table_name
SET column1 = value1, column2 = value2, ...
WHERE condition;
```

The SQL DELETE Statement
The DELETE statement is used to delete existing records in a table.

DELETE Syntax
```
DELETE FROM table_name WHERE condition;
```

The SQL MIN() and MAX() Functions
The MIN() function returns the smallest value of the selected column.

The MAX() function returns the largest value of the selected column.

MIN() Syntax
```
SELECT MIN(column_name)
FROM table_name
WHERE condition;
```
Example:
```
SELECT MIN(Price) AS SmallestPrice
FROM Products;
```

The SQL COUNT(), AVG() and SUM() Functions
The COUNT() function returns the number of rows that matches a specified criteria.
The AVG() function returns the average value of a numeric column.
The SUM() function returns the total sum of a numeric column.

```
COUNT() Syntax:
SELECT COUNT(column_name)
FROM table_name
WHERE condition;

AVG() Syntax:
SELECT AVG(column_name)
FROM table_name
WHERE condition;

SUM() Syntax:
SELECT SUM(column_name)
FROM table_name
WHERE condition;
```

The SQL DROP TABLE Statement
The DROP TABLE statement is used to drop an existing table in a database.

Syntax
```
DROP TABLE table_name;
```

SQL ALTER TABLE Statement
The ALTER TABLE statement is used to add, delete, or modify columns in an existing table.

The ALTER TABLE statement is also used to add and drop various constraints on an existing table.

ALTER TABLE - ADD Column
To add a column in a table, use the following syntax:

```
ALTER TABLE table_name
ADD column_name datatype;
```

Example:
```
ALTER TABLE Customers
ADD Email varchar(255);
```

ALTER TABLE - DROP COLUMN
To delete a column in a table, use the following syntax (notice that some database systems don't allow deleting a column):

```
ALTER TABLE table_name
DROP COLUMN column_name;
```
The following SQL deletes the "Email" column from the "Customers" table:

Example:
```
ALTER TABLE Customers
DROP COLUMN Email;
```

SQL Constraints
SQL constraints are used to specify rules for data in a table.

SQL Create Constraints
Constraints can be specified when the table is created with the CREATE TABLE statement, or after the table is created with the ALTER TABLE statement.

Syntax:
```
CREATE TABLE table_name (
   column1 datatype constraint,
   column2 datatype constraint,
   column3 datatype constraint,
   ....
);
```

Constraints are used to limit the type of data that can go into a table. This ensures the accuracy and reliability of the data in the table. If there is any violation between the constraint and the data action, the action is aborted.

Constraints can be column level or table level. Column level constraints apply to a column, and table level constraints apply to the whole table.

The following constraints are commonly used in SQL:

NOT NULL - Ensures that a column cannot have a NULL value
UNIQUE - Ensures that all values in a column are different
PRIMARY KEY - A combination of a NOT NULL and UNIQUE. Uniquely identifies each row in a table
FOREIGN KEY - Uniquely identifies a row/record in another table
CHECK - Ensures that all values in a column satisfies a specific condition
DEFAULT - Sets a default value for a column when no value is specified
INDEX - Used to create and retrieve data from the database very quickly

Authentication:

Authentication: Authentication is the process of determining the identity of a user based on the user’s credentials. The user’s credentials are usually in the form of user ID and password, which is checked against any credentials' store such as database. If the credentials provided by the user are valid, then the user is considered an authenticated user.
In ASP.NET there are different ways in which authentication is performed as discussed below:

1) **Anonymous Access:** There is no authentication performed and the user is treated as anonymous user by IIS.

Sometimes it is necessary or possible to access any data from remote server or database without any authenticated person. If the data is easily accessible without any authentication (i.e username and password) that means the user accessing those data or file has an anonymous access. There is no difference between a user who is “anonymously authenticated” and an unauthenticated user. There can be many situations where anonymous authentication is useful.

2) **Windows Authentication:** Provides information on how to use Windows authentication in conjunction with Microsoft Internet Information Services (IIS) authentication to secure ASP.NET applications. This is the default authentication mode in ASP.NET and it is set in web.config file of the application.

Windows Authentication uses the security features of Windows clients and servers. Unlike Basic authentication, initially, it does not prompt users for a user name and password. The current Windows user information on the client computer is supplied by the web browser through a cryptographic exchange involving hashing with the Web server. If the authentication exchange initially fails to identify the user, the web browser will prompt the user for a Windows user account user name and password.

Windows authentication is generally used if the users accessing the application belong to same organization. This authentication method uses Windows accounts for validating users' credentials. This type of authentication is very good for intranet Web sites where we know our users.

Cookies

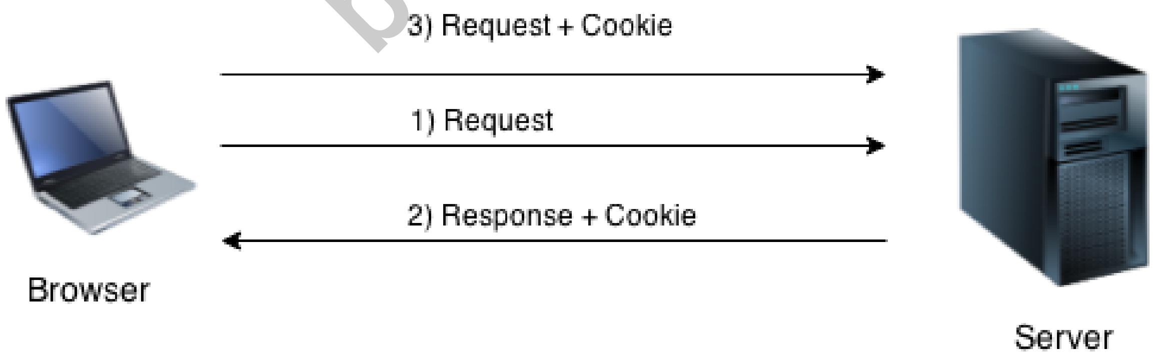
Cookies are data, stored in small text files, on your computer.

When a web server has sent a web page to a browser, the connection is shut down, and the server forgets everything about the user.

Cookies were invented to solve the problem "how to remember information about the user":

When a user visits a web page, his/her name can be stored in a cookie.
Next time the user visits the page, the cookie "remembers" his/her name.
Cookies are saved in name-value pairs like:

username = John Doe
When a browser requests a web page from a server, cookies belonging to the page is added to the request. This way the server gets the necessary data to "remember" information about users.



Session		Cookies	
1	Session expires when the browser is closed.	1	Cookies last longer than session
2	Session are stored on the server computer	2	Cookies are stored on the user's computer
3	Session are reliable and more secure as we cannot easily access the session value	3	Cookies are unreliable and less secure as we can easily access the cookie value
4	Session can store unlimited amount of data	4	Cookies can store only limited amount of data
5	Session is mainly used for login/logout purpose	5	cookies using for user activity tracking

File Handling

File Handling concept in C#.

File handling simply deal with creating, reading, deleting and editing files. Website can handle local files. It can read from, and write to, a text file on the disk. The code can run on any server with file system privileges—and also a local development machine

There are some modes of File which is mostly used in Real application.

1. **Create** :-In this mode New File will be created .If File is Exist then It will be Overridden.
2. **Append** :- In Append mode Existing File will be open and contents will be written from Last .If File is not Exist then new File will be created. Append mode is used Between two Files.
3. **Create New**:-In this mode New File will be created, if File is already Exist then it will thrown Exception.
4. **Open**:- In this Existing file will be opened if file is not Exist then it will be thrown Exception.
5. **OpenRead**:- In this Opens the Existing File for Reading the Contents.
6. **Truncate**:-In this Existing File will be open and the Entire contents of the file will be removed.But if the file is not Exist then it will be thrown Exception.

StreamReader Class:-

- **Flush()**:- Flush function is used for immediately save the File contents from **Buffer** to **Memory.**
- **Close():-**Close function is used for closing the file.If we do not write close() statement in our program then File will be always open mode ,then No other person will be used this File at that time. This is the reason for using close statement in the File Program.
- **Read**()- It is used for Reading the Value using File Stream.
- **Read-line()**:-It is used for Read the value using File Stream in a File Line by Line.
- **Peek()**:- It returns next value but not use it.
- **Seek()**:- It is used for Read/Write a values at any positions in a file.

StreamWriter Class:-

- **close()**-->Close function is used for closing the file.
- **Flush()**:-Flush function is used for immediately save the File contents from **Buffer** to **Memory.**
- **Write():-** It is used for writing a File using File stream class.
- **WriteLine()**:- It is used to write a File Line by Line using File stream.

HTTP Functions:

The HTTP functions let you manipulate information sent to the browser by the Web server, before any other output has been sent.

Function	**Description**
header()	Sends a raw HTTP header to a client Syntax: header(string, replace, http_response_code)
headers_list()	Returns a list of response headers sent (or ready to send) Syntax: headers_list() To determine whether or not the headers have been sent yet, use the headers_sent() function.
headers_sent()	Checks if / where the HTTP headers have been sent Syntax: headers_sent(file,line)
setcookie()	Defines a cookie to be sent along with the rest of the HTTP headers Syntax: setcookie(name,value,expire,path,domain,secure,httponly); A cookie is often used to identify a user. A cookie is a small file that the server embeds on the user's computer. Each time the same computer requests a page with a browser, it will send the cookie too.
setrawcookie()	Defines a cookie (without URL encoding) to be sent along with the rest of the

```
HTTP headers
Syntax: setrawcookie(name,value,expire,path,domain,secure);
```

FTP Functions:
The FTP functions give client access to file servers through the File Transfer Protocol (FTP).

The FTP functions are used to open, login and close connections, as well as upload, download, rename, delete, and get information on files from file servers. Not all of the FTP functions will work with every server or return the same results.

Some FTP Functions are given below:

ftp_alloc — Allocates space for a file to be uploaded
ftp_cdup — Changes to the parent directory
ftp_chdir — Changes the current directory on a FTP server
ftp_chmod — Set permissions on a file via FTP
ftp_close — Closes an FTP connection
ftp_connect — Opens an FTP connection
ftp_delete — Deletes a file on the FTP server
ftp_exec — Requests execution of a command on the FTP server
ftp_fget — Downloads a file from the FTP server and saves to an open file
ftp_fput — Uploads from an open file to the FTP server
ftp_get_option — Retrieves various runtime behaviours of the current FTP stream
ftp_get — Downloads a file from the FTP server
ftp_login — Logs in to an FTP connection
ftp_mdtm — Returns the last modified time of the given file
ftp_mkdir — Creates a directory
ftp_nb_continue — Continues retrieving/sending a file (non-blocking)
ftp_nb_fget — Retrieves a file from the FTP server and writes it to an open file (non-blocking)
ftp_nb_fput — Stores a file from an open file to the FTP server (non-blocking)
ftp_nb_get — Retrieves a file from the FTP server and writes it to a local file (non-blocking)
ftp_nb_put — Stores a file on the FTP server (non-blocking)
ftp_nlist — Returns a list of files in the given directory
ftp_put — Uploads a file to the FTP server
ftp_pwd — Returns the current directory name
ftp_quit — Alias of ftp_close
ftp_raw — Sends an arbitrary command to an FTP server
ftp_rawlist — Returns a detailed list of files in the given directory
ftp_rename — Renames a file or a directory on the FTP server
ftp_rmdir — Removes a directory
ftp_set_option — Set miscellaneous runtime FTP options
ftp_site — Sends a SITE command to the server
ftp_size — Returns the size of the given file
ftp_ssl_connect — Opens an Secure SSL-FTP connection
ftp_systype — Returns the system type identifier of the remote FTP server

Q. What are anti overload techniques in web server?
At any time web servers can be overloaded because of:

- Too much legitimate web traffic

- Thousands or even millions of clients hitting the web site in a short interval of time (DDoS) Distributed Denial of Service attacks
- Computer worms
- Millions of infected browsers and/or web servers
- limited on large web sites with very few resources (bandwidth, etc.)
- Client requests are served more slowly and the number of connections increases so much that server limits are reached
- Web servers required urgent maintenance or upgrade
- Hardware Software failures

To partially overcome load limits and to prevent overload we can use techniques like:

- Managing network traffic by using: Firewalls, Block unwanted traffic
- HTTP traffic managers- redirect or rewrite requests having bad HTTP patterns
- Bandwidth management and traffic shaping
- Deploying web cache techniques
- Use different domains to serve different content (static and dynamic) by separate Web servers,
- Add more hardware resources
- Use many web servers

These technique to limit the load on websites are called anti-overload techniques in web server.

Q. How a Domain Name is translated to an IP Address?

DNS (Domain Name System) is what translate domain name (for eg. www.google.com) into an IP address that our browser can use (for eg. 173.194.35.148). Before the page is loaded, the DNS must be resolved so the browser can establish a TCP connection to make the HTTP request. The DNS Resolution process starts when the user types a URL address on the browser and hits Enter.

On the Internet, many communications programs deal only with IP addresses, yet allow their users to specify machines in terms of their host names (or alias host names). Or a program which already knows the IP address must determine the domain name for the network to which the machine is connected. Such programs must somehow convert the host names into IP addresses (or vice versa) behind the scenes. How do they achieve this translation between IP addresses and host names?

The mapping of host names to IP addresses is handled through a service called Domain Name Service (DNS). Rather than require individual machines, applications, or users to keep up with the constant changes in host names and IP addresses, a series of special DNS servers across the world (known as "name servers") keep track of the name/address information for all the computers on the Internet. Applications that need to determine an IP address from a host name (or vice versa) contact the local "name server" to supply this information.

For instance, if you use a web browser to check out the site "web.mit.edu", the program actually first contacts your local DNS machine to obtain the IP-address that matches the host name you provided; then the program uses that IP address to complete your request.

DNS is used much more frequently than is usually supposed: virtually every activity that moves information across the network (getting web documents, transferring files, sending or receiving electronic mail) relies on DNS.