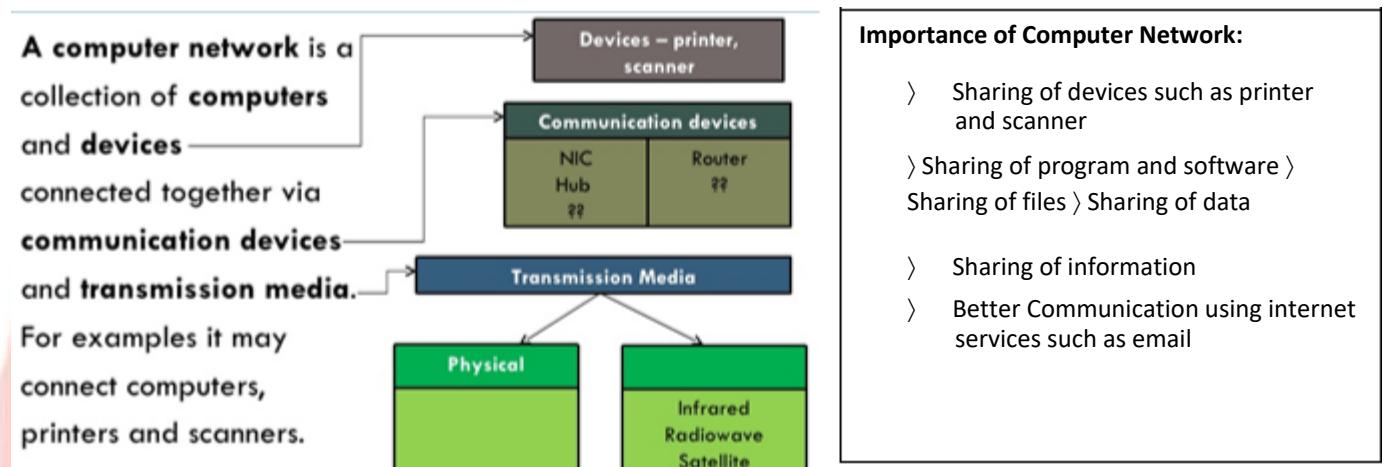


Unit 1: Introduction

Introduction to Networking:



Types of computer Network:

<i>Different LAN</i>	<i>MAN</i>	<i>WAN</i>
Cost	Low	High
Network Size	Small	Largest
Speed	Fastest	Slowest
Transmission Media	Twisted-pair	Twisted pair, fiber optics
No. of Computers	Smallest	Large

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:

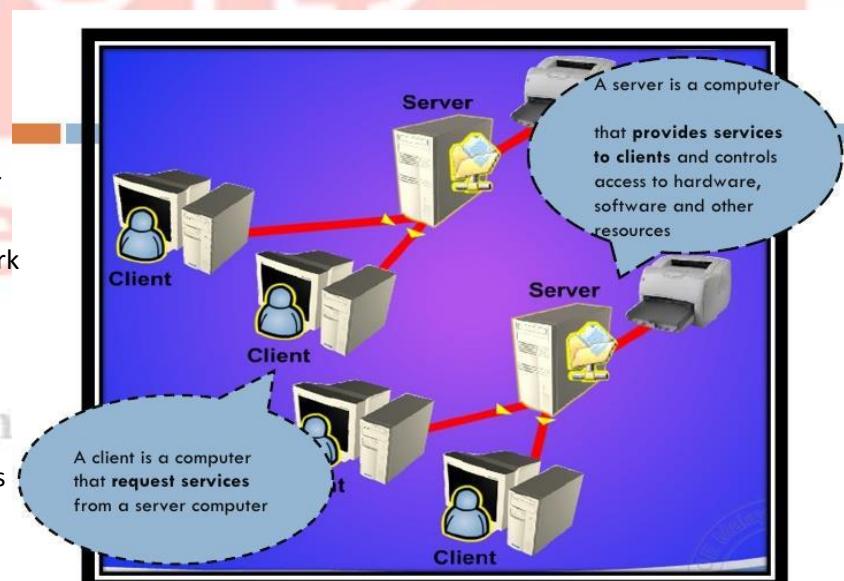
- 1) Client/server Network
- 2) Peer to peer Network

Client/Server:

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.

Peer-to peer

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.

WWW : 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 and
- › 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
<u></u>	Defines bold text
<u></u>	Defines emphasized text
<u><i></u>	Defines a part of text in italic
<u><small></u>	Defines smaller text
<u></u>	Defines important text
<u><sub></u>	Defines subscripted text
<u><sup></u>	Defines superscripted text
<u><ins></u>	Defines inserted text
<u></u>	Defines deleted text
<u><mark></u>	Defines marked/highlighted text

HTML "Computer Output" Tags

Tag	Description
<u><code></u>	Defines computer code text
<u><kbd></u>	Defines keyboard text
<u><samp></u>	Defines sample computer code
<u><var></u>	Defines a variable
<u><pre></u>	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: Visit Me !

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.devcpn.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:

```
<
<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.

```
<script>
```

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

```
</script>
```

```
<noscript>Your browser does not support JavaScript!</noscript>
```

Example:

BCA

NOTES

N e p a l

www.bcanotesnepal.com

HTML Images - The Tag and the Src Attribute

Syntax for defining an image:

 Tag Attributes	Definitions
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:

```
 <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:

 - An unordered list. This will list items using plain bullets.

 - 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 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 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 tag. The numbering starts at one and is incremented by one for each successive ordered list element tagged with .

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 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 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
Black	#000000	rgb(0,0,0)
Red	#FF0000	rgb(255,0,0)
Green	#00FF00	rgb(0,255,0)
Blue	#0000FF	rgb(0,0,255)
Yellow	#FFFF00	rgb(255,255,0)
Cyan	#00FFFF	rgb(0,255,255)
Magenta	#FF00FF	rgb(255,0,255)
Grey	#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 drop-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
©	©	©	COPYRIGHT SIGN
®	®	®	REGISTERED SIGN
€	€	€	EURO SIGN
™	™	™	TRADEMARK
←	←	←	LEFTWARDS ARROW

↑	↑	↑	UPWARDS ARROW
→	→	→	RIGHTWARDS ARROW
↓	↓	↓	DOWNTWARDS ARROW
♠	♠	♠	BLACK SPADE SUIT
♣	♣	♣	BLACK CLUB SUIT
♥	♥	♥	BLACK HEART SUIT
♦	♦	♦	BLACK DIAMOND SUIT

Some Mathematical Symbols:

Char	Number	Entity	Description
∀	∀	∀	FOR ALL
∂	∂	∂	PARTIAL DIFFERENTIAL
∃	∃	∃	THERE EXISTS
∅	∅	∅	EMPTY SETS
∇	∇	∇	NABLA
∈	∈	∈	ELEMENT OF
∉	∉	∉	NOT AN ELEMENT OF
∋	∋	∋	CONTAINS AS MEMBER
∏	∏	∏	N-ARY PRODUCT
Σ	∑	∑	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 <code><div></code> 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>

```

Source: <http://www.devcp.gn.blogspot.com>

```

<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-

```

Unit1:14

>

Copyright © www.devcpgn.blogspot.com

>

Copyright © www.devcpgn.blogspot.com

BCA NOTES

N e p a l

www.bcanotesnepal.com



HTML Tag:

A element used to color a part of a text:

Example: <p>My mother has blue 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

CSS saves time – 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.

Pages load faster – 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.

Easy maintenance – 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.

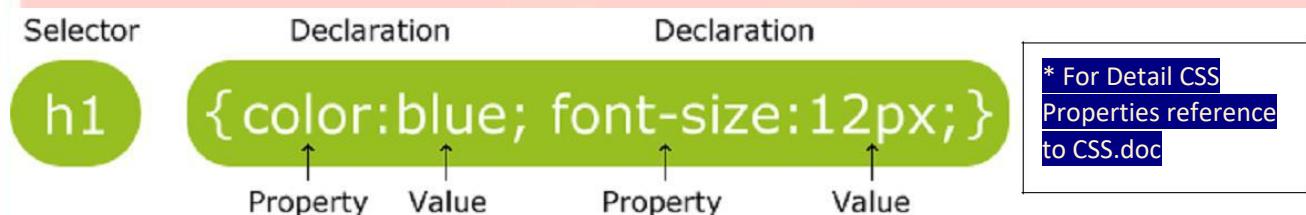
Multiple Device Compatibility – Style sheets allow content to be optimized for more than one type of device. By using the same HTML document, different versions of a website can be presented for handheld devices such as PDAs and cell phones or for printing.

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.



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

```
*/
```

BCA NOTES

N e p a l

www.bcanotesnepal.com

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 statement 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(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	
+=	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: www.bcanotesnepal.com

```

<html>
<body>
<script type="text/javascript">
    var a = 30;
    var b = 20;
    //var linebreak = "<br />";

```

BCA NOTES

N e p a l

www.bcanotesnepal.com

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

```

</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
Onclick	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
Oncontextmenu	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
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 );
            }
        }
    </script>
</head>
<body>
    <h1>Hello World!</h1>
</body>
</html>
```

```

        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 JavaScript</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>

```