Chapter – 1

Internet and WWW

INTRODUCTION:

The Internet is the global system of interconnected computer networks that use the Internet protocol suite (TCP/IP) to link devices worldwide. It is a network of networks that consists of private, public, academic, business, and government networks of local to global scope, linked by a broad array of electronic, wireless, and optical networking technologies. The Internet carries an extensive range of information resources and services, such as the inter-linked hypertext documents and applications of the World Wide Web (WWW), electronic mail, telephony, and peer-to-peer networks for file sharing.

HISTORY:

Internet was originally conceived by the Department of Defense as a way to protect government communications systems in the event of a military strike. The original network, dubbed ARPANet (for the Advanced Research Projects Agency that developed it) evolved into a communications channel among contractors, military personnel, and university researchers who were contributing to ARPA projects.

The network employed a set of standard protocols to create an effective way for these people to communicate and share data with each other.

ARPAnet's popularity continued to spread among researchers, and in the 1980's the National Science Foundation, whose NSFNet, linked several high speed computers, and took charge of what had come to be known as the Internet.

By the late 1980's, thousands of cooperating networks were participating in the Internet.

In 1991, the U.S. High Performance Computing Act established the NREN (National Research & Education Network). NREN's goal was to develop and maintain high-speed networks for research and education, and to investigate commercial uses for the Internet.

The rest, as they say, is history in the making. The Internet has been improved through the developments of such services as Gopher and the World Wide Web.

Even though the Internet is predominantly thought of as a research oriented network, it continues to grow as an informational, creative, and commercial resource every day and all over the world.

APPLICATION AREAS OF INTERNET:

The Internet has many important applications. Of the various services available via the Internet, the following are the areas where internet use:

1. **Communication:**

It is used for sending and receiving message from one and other through internet by using electronic mail. Some of the web sites providing this service are yahoomail.com Hotmail.com rediffmail.com etc.

2. **Job Searches:**

Getting information regarding availability of job in different sectors and areas. We can publish our resume in online for prospective job. Some of the web sites providing this service are naukri.com, monster.com, summerjob.com, recuritmentindia.com etc.

3. **Finding Books And Study Material :**

Books and other study material stored around the world can be easily located through internet. Latest encyclopedias are available online.

4. **Health And Medicine:**

Internet provide information and knowledge about field of health medicine people can have information about various disease and can receive help .patient can be taken to virtual check room where they can meet doctors. Some of the web sites providing this service are

5. **Travel:**

One can use internet to gather information about various tourist place. It can be used for booking Holiday tours, hotels, train and flights. Some of the web sites providing this service are indiatravelog.com, rajtravel.com, makemytrip.com.

6. **Entertainment:**

One can download jokes, songs movies, and latest sports updates through internet. Some of the web sites providing this service arecricinfo.com, movies.com espn.com. We can also watch online videos through internet with the help of YouTube.

7. **Shopping:**

Internet is also used for online shopping. By just giving accounts details we can perform the transaction. We can even pay our bills and perform bank related transaction.

8. **Stock Market Updates:**

We can sell or buy shares while sitting on computer through internet. Several websites like ndtvprofit.com, moneypore.com, provide information regarding investment.

9. **Research :**

A large number of people are using internet for research purposes we can download any kind information by using internet.

10. Business Use Of Internet:

There are different ways by which internet can be used for business are:

- Information about the product can be provided online to the customer
- Provide market information to the business e.g. Stock Market
- It help business to recruit talented people.
- Help in locating suppliers of the product
- Feedback and reviews about companies product
- Eliminate middle men and have a direct contact with customer
- Providing information to the investor by providing companies back ground and financial information on web site.

INTERNET SERVICE PROVIDER:

An Internet service provider (ISP) is a company that provides customers with Internet access. The ISP may be organized in various form such as: commercial community owned, Non-profit or privately owned. Data may be transmitted using several technologies, including dial-up, DSL, cable modem, wireless or dedicated high-speed interconnects.
Typically, ISPs also provide their customers with the ability to communicate with one another by providing Internet email accounts, usually with numerous email addresses at the customer's discretion. Other services, such as telephone and television services, may be provided as well. The services and service combinations may be unique to each ISP.
An Internet service provider is also known as an Internet access provider (IAP). Some ISP in Nepal are:

- WorldLink Communication Pvt Ltd
- Vianet Communication
- Websurfer Nepal Pvt Ltd
- Mercantile Communication
- Websurfer Nepal Pvt Ltd

DOMAIN NAME SERVER (DNS):

Domain Name Servers (DNS) are the Internet's equivalent of a phone book. They maintain a directory of domain names and translate them to Internet Protocol (IP) addresses.
This is necessary because, although domain names are easy for people to remember, computers or machines, access websites based on IP addresses.
Information from all the domain name servers across the Internet are gathered together and housed at the Central Registry. Host companies and Internet Service Providers interact with the Central Registry on a regular schedule to get updated DNS information.
When we type in a web address, e.g., www.jimsbikes.com, our Internet Service Provider views the DNS associated with the domain name, translates it into a machine friendly IP address (for example

216.168.224.70 is the IP for jimsbikes.com) and directs our Internet connection to the correct website.
After we register a new domain name or when we update the DNS servers on our domain name, it usually takes about 12-36 hours for the domain name servers world-wide to be updated and able to access the information. This 36-hour period is referred to as propagation.

NETWORK PROTOCOL:

A network protocol defines rules and conventions for communication between network devices. Network protocols include mechanisms for devices to identify and make connections with each other, as well as formatting rules that specify how data is packaged into messages sent and received. Some protocols also support message acknowledgement and data compression designed for reliable and/or high-performance network communication.
Modern protocols for computer networking all generally use packet switching techniques to send and receive messages in the form of packets - messages subdivided into pieces that are collected and re-assembled at their destination. Hundreds of different computer network protocols have been developed each designed for specific purposes and environments.

PROTOCOLS USED IN INTERNET:

TCP/IP (Transmission Control Protocol/Internet Protocol):

TCP (Transmission Control Protocol) and IP (Internet Protocol) are two different procedures that are often linked together. The linking of several protocols is common since the functions of different protocols can be complementary so that together they carry out some complete task. The combination of several protocols to carry out a particular task is often called a "stack" because it has layers of operations. In fact, the term "TCP/IP" is normally used to refer to a whole suite of protocols, each with different functions. This suite of protocols is what carries out the basic operations of the Web. TCP/IP is also used on many local area networks.
TCP/IP defines how electronic device (like computer) should be connected over the internet and how data should be transmitted between them. It is a set of rule that an application can use to package its information for sending across the network of networks.

TCP (Transmission Control Protocol):

When information is sent over the Internet, it is generally broken up into smaller pieces or "packets". The use of packets facilitates speedy transmission since different parts of a message can be sent by different routes and then reassembled at the destination. It is also a safety measure to minimize the chances of losing information in the transmission process. TCP is the means for creating the packets, putting them back together in the correct order at the end, and checking to make sure that no packets got lost in transmission. If necessary, TCP will request that a packet be resent.

❖ IP (Internet Protocol):

Internet Protocol (IP) is the method used to route information to the proper address. Every computer on the Internet has to have its own unique address known as the IP address. Every packet sent will contain an IP address showing where it is supposed to go. A packet may go through a number of computer routers before arriving at its final destination and IP controls the process of getting everything to the designated computer. Note that IP does not make physical connections between computers but relies on TCP for this function. IP is also used in conjunction with other protocols that create connections.

COMMON TYPES OF TCP/IP ARE:

a. HTTP (Hyper Text Transfer Protocol):

HTTP is a communication protocol. It defines mechanism for communication between browser and the web server. It is also called request and response protocol because the communication between browser and server takes place in request and response pairs.

b. HTTPS(Hypertext Transfer Protocol over Secure Socket Layer):

HTTPS was first introduced by Netscape. HTTPS takes care of secure communication between a web server and web browser. HTTPS typically handles credit card transaction and other sensitive data. A Web page using this protocol will have https: at the front of its URL.

c. FTP (File Transfer Protocol):

File Transport Protocol is a program that allows users and computers to send and receive large portions of data through a private or public network. It can also be used to send configuration files and software updates to network devices, such as switches and routers. FTP uses ports for communications and also uses encryption to protect the information being received and sent.

d. SMTP (Simple Mail Transfer Protocol):

SMTP is a TCP/IP protocol used in sending and receiving e-mail. However, since it is limited in its ability to queue massages at the receiving end, it is usually used with one of two other protocols POP3 or IMAP that let the user save the messages in a server mailbox and download them periodically from the server. In other words, users typically use a program that uses SMTP for sending e-mail and either POP3 or IMAP for receiving e-mail. On UNIX based systems, sendmail is the most widely used SMTP server for e-mail.

e. UDP (User Datagram Protocol):

The User Datagram Protocol is one of the core members of the internet protocol suite. UDP uses a simple connectionless transmission model with a minimum of protocol mechanism. UDP provides checksums for data integrity, and port numbers for addressing different functions at the source and destination of the datagram. It has no handshaking dialogues, and thus exposes the user's program to any unreliability of the underlying network and so there is no guarantee of delivery, ordering, or duplicate protection. If error correction facilities are needed at the network interface level, an

application may use the TCP or SCTP (Stream Control Transmission Protocol) which are design for this purpose.

WWW (WORLD WIDE WEB):

The World Wide Web (abbreviated WWW or the Web) is an information space where documents and other web resources are identified by Uniform Resource Locators (URLs), interlinked by hypertext links, and can be accessed via the Internet. English scientist Tim Berners-Lee invented the World Wide Web in 1989. He wrote the first web browser computer program in 1990 while employed at CERN in Switzerland. The Web browser was released outside of CERN in 1991, first to other research institutions starting in January 1991 and to the general public on the Internet in August 1991.

The World Wide Web has been central to the development of the Information Age and is the primary tool billions of people use to interact on the Internet. Web pages are primarily text documents formatted and annotated with Hypertext Markup Language (HTML). In addition to formatted text, web pages may contain images, video, audio, and software components that are rendered in the user's web browser as coherent pages of multimedia content. Embedded hyperlinks permit users to navigate between web pages. Multiple web pages with a common theme, a common domain name, or both, make up a website. Website content can largely be provided by the publisher, or interactive where users contribute content or the content depends upon the user or their actions. Websites may be mostly informative, primarily for entertainment, or largely for commercial, governmental, or non-governmental organizational purposes.

WEB SERVER:

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.

The process is an example of the client/server model. All computers that host Web sites must have Web server programs. Leading Web servers include Apache (the most widely-installed Web server), Microsoft's Internet Information Server (IIS) and nginx (pronounced engine X) from NGNIX. Other Web servers include Novell's NetWare server, Google Web Server (GWS) and IBM's family of Domino servers.

Web servers often come as part of a larger package of Internet- and intranet-related programs for serving email, downloading requests for File Transfer Protocol (FTP) files, and building and publishing Web pages. Considerations in choosing a Web server include how well it works with the operating system and other servers, its ability to handle server-side programming, security characteristics, and the particular publishing, search engine and site building tools that come with it.

WEB BROWSER:

A browser is an application program that provides a way to look at and interact with all the information on the World Wide Web. The word "browser" seems to have originated prior to the Web as a generic term for user interfaces that let you browse (navigate through and read) text files online.
Technically, a Web browser is a client program that uses HTTP (Hypertext Transfer Protocol) to make requests of Web servers throughout the Internet on behalf of the browser user. Most browsers support e-mail and the File Transfer Protocol (FTP) but a Web browser is not required for those Internet protocols and more specialized client programs are more popular.
The first Web browser, called WorldWideWeb, was created in 1990. That browser's name was changed to Nexus to avoid confusion with the developing information space known as the World Wide Web. The first Web browser with a graphical user interface was Mosaic, which appeared in 1993.
The most popular web browsers are Google Chrome, Microsoft Edge (preceded by Internet Explorer), Safari, Opera and Firefox.

SEARCH ENGINE:

A search engine is a software program or script available through the Internet that searches documents and files for keywords and returns the results of any files containing those keywords. Today, there are thousands of different search engines available on the Internet, each with their own abilities and features. The first search engine ever developed is considered Archie, which was used to search for FTP files and the first text-based search engine is considered Veronica. Today, the most popular and well-known search engine is Google.
Large search engines contain millions and sometimes billions of pages, many search engines not only just search the pages but also display the results depending upon their importance. This importance is commonly determined by using various algorithms.
For users, a search engine is accessed through a browser on their computer, smartphone, tablet, or another device.

META SEARCH ENGINE:

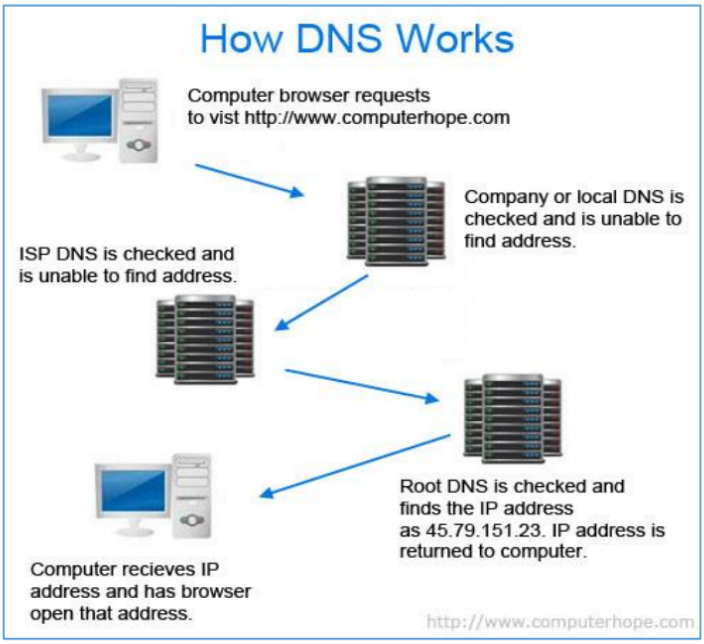
A search engine that queries other search engines and then combines the results that are received from all. In erect, the user is not using just one search engine but a combination of many search engine at once to optimize web searching. For example: Dogpile, Mamma are metasearch engine. A metasearch engine (or aggregator) is a search tool that uses another search engine's data to produce their own results from the internet. Metasearch engines take input from a user and simultaneously send out queries to third party search engines for results.

DOMAIN NAME:

When referring to an Internet address or name a **domain** or **domain name** is the location of a website. For example, the domain name "computerhope.com" points to the IP address "45.79.151.23", but it is generally easier to remember a name rather than a long string of numbers. A domain name can be a maximum of sixty-three characters with one character minimum, and is entered after the protocol in the URL, as we can see in the following example.

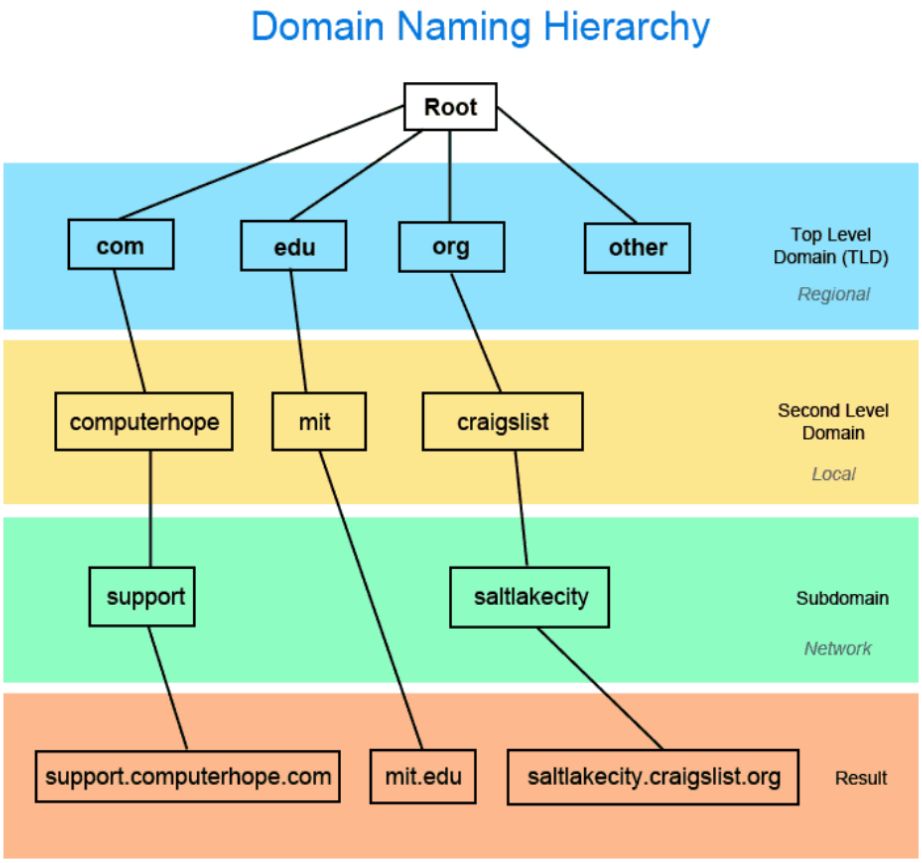
When deciding on a domain name, it's a good idea to keep it simple, something that is easy to remember. Additional promoting tips for websites is on our promotion page. To register or lookup a domain name, we recommend visiting GoDaddy or Network solutions; both companies are domain name registrars.

When referring to a computer network, a domain is a group of resources assigned to a specific group of individuals. It is used to divide global areas or departments of a corporation. A domain may need to be specified when mapping a network computer or drive.



DOMAIN NAME HIERARCHY:

There are millions of websites available on the Internet to make finding an address the domain naming is broken into a hierarchical structure. Below is how DNS is structured on the Internet and an example of three different addresses.



In the above example, all websites are broken into regional sections based on the top level domain (TLD). In the example of http://support.computerhope.com it has a ".com" TLD, with "computerhope" as its second level domain that is local to the .com TLD, and "support" as its subdomain, which is determined by its server.

ISSUES RELATED WITH DOMAIN NAME REGISTRATION:

- **Typosquatting:**

Where a person registers a domain name similar to a real domain name, but with a typo, in hopes that web surfers reach it by accident. These sites are usually filled with paid advertising links that generate revenue for the typosquatter, not to mention the web surfer has been tricked into believing he is on the correct site. This diverts traffic away from the intended site. Sometimes they are routed to a competitor's site or a pornographic site.

- **Cybersquatting:**

When someone registers a domain name, in bad faith, violating the rights of the trademark owner. They usually intend to extort payment from the trademark owner, and they keep the names to sell later to the highest bidder.

- **Pagejacking:**

When the offender copies part of an existing website, and then puts it up on a different website to make it look like the original. Pagejacking is used in phishing schemes, where the fake page gathers account numbers, passwords, and personal information from the unsuspecting user.

- **The Uniform Domain Name Dispute Resolution Policy (UDRP):**

It is a cost-effective and faster alternative to a lawsuit, when there is a domain name dispute that needs to be resolved. This was set up by the Internet Corporation for Assigned Names and Numbers (ICANN), the group responsible for domain name registration.

DOMAIN NAME SYSTEM:

Short for Domain Name System or Domain Name Service, a DNS is an Internet or network server that helps to point domain names or hostnames to their associated Internet Protocol address and was introduced by Paul Mockapetris and Jon Postel in 1983. Without a DNS to resolve a domain name or the proper rights, users would have to know the IP address of each of the web pages or computers you wanted to access.

CLIENT/SERVER ARCHITECTURE:

Client/server architecture is a computing model in which the server hosts, delivers and manages most of the resources and services to be consumed by the client. This type of architecture has one or more client computers connected to a central server over a network or Internet connection. This system shares computing resources.
Client/server architecture may also be referred to as a networking computing model because all the requests and services are delivered over a network.
Client/server architecture works when the client computer sends a resource or process request to the server over the network connection, which is then processed and delivered to the client. A server computer can manage several clients simultaneously, whereas one client can be connected to several servers at a time, each providing a different set of services. In its simplest form, the Internet is also based on client/server architecture where the Web server serves many simultaneous users with Web page and or website data.

CROSS BROWSER COMMUNICATION:

When a software program is developed for multiple computer platforms, it is called a crossplatform program. Similarly, when a website is developed for multiple browsers, it is called a cross-browser website.
The job of a Web developer would be much easier if all browsers were the same. While most browsers are similar in both design and function, they often have several small differences in the way they recognize and display websites. For example, Apple's Safari uses a different HTML rendering engines than Internet Explorer. This means the browsers may display the same Web page with slightly different page and text formatting. Since not all browsers support the same HTML tags,

some formatting may not be recognized at all in an incompatible Web browser. Furthermore, browsers interpret JavaScript code differently, which means a script may work fine in one browser, but not in another.

Because of the differences in the way Web browsers interpret HTML and JavaScript, Web developers must test and adapt their sites to work with multiple browsers. For example, if a certain page looks fine in Firefox, but does not show up correctly in Internet Explorer, the developer may change the formatting so that it works with Internet Explorer. Of course, the page may then appear differently in Firefox. The easiest fix for browser incompatibility problems is to use a more basic coding technique that works in both browsers. However, if this solution is not possible, the developer may need to add code that detects the type of browser, then outputs custom HTML or JavaScript for that browser.

Making a cross-browser site is usually pretty simple for basic websites. However, complex sites with a lot of HTML formatting and JavaScript may require significant extra coding in order to be compatible with multiple browsers. Some developers may even generate completely different pages for each browser. While CSS formatting has helped standardize the appearance of Web pages across multiple browsers, there are still several inconsistencies between Web browsers. Therefore, cross-browser design continues to be a necessary aspect of Web development.

CHAPTER – 2

HTML AND GRAPHICS

INTRODUCTION:

1. HTML:

HTML stands for Hyper Text Markup Language, which is the most widely used language on web to develop web pages. A markup language is a set of markups tags. HTML documents are described by HTML tags. Each HTML tag describes different document content.

HTML tags are keywords (tag names) surrounded by angle brackets. For example: <tagname> contents </tagname>. 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, but with a slash before the tag name.

HTML was created by Tim Berners Lee in late 1991 but "HTML 2.0" was the first standard HTML specification which was published in 1995. HTML 4.01 was a major version of HTML and it was published in late 1999. Though HTML 4.01version is widely used but currently we are having HTML 5 version which is an extension to HTML 4.01 and this version was published in 2012.

2. STRUCTURE OF HTML:

```
<! DOCTYPE HTML>
<html>
<head>
<title> </title>
</head>
<body>
</body>
</html>
```

3. < ! DOCTYPE>:

The <! DOCTYPE> declaration must be the very first thing in our HTML document, before the <html> tag. 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 HTML 4.01, the <! DOCTYPE> declaration refers to a Document Type Definition (DTD), because HTML 4.01 was based on Standard Generalized Markup Language (SGML). The DTD specifies the rules for the markup language, so that the browsers render the content correctly. HTML 5 is not based on SGML, and therefore does not require a reference to a DTD.

4. ELEMENTS:

An HTML element usually consists of a start tag and end tag, with the content inserted in between, for example: <tagname>Content goes here...</tagname>. The HTML element is everything from the start tag to the end tag, for example: My first paragraph .

HTML elements can be nested (elements can contain elements). All HTML documents consist of nested HTML elements.

HTML elements with no content are called empty elements.
 is an empty element without a closing tag (the <br> tag defines a line break). Empty elements can be "closed" in the opening tag like this: <br />. HTML5 does not require empty elements to be closed. But if we want stricter validation, or if we need to make our document readable by XML parsers, we must close all HTML elements properly.

HTML tags are not case sensitive: <P> means the same as <p>. The HTML5 standard does not require lowercase tags, but W3C recommends lowercase in HTML, and demands lowercase for stricter document types like XHTML.

```
<! DOCTYPE HTML>
<html>
<head>
<title> Elements Example</title>
</head>
<body>
<h1>This is <i>italic</i> heading</h1>
<p >This is  <u>underlined</u> paragraph </p>
</body>
</html>
```

5. Attributes:

An attribute is used to define the characteristics of an HTML element and is placed inside the element's opening tag. All attributes are made up of two parts: a name and a value.

The name is the property we want to set. For example, the paragraph <p> element in the example carries an attribute whose name is align, which we can use to indicate the alignment of paragraph on the page.

The value is what we want the value of the property to be set and always put within quotations. The below example shows three properties values of align attribute: left, center, right.

```
<! DOCTYPE HTML>
<html>
<head>
<title>Align Attribute Example</title>
</head>
<body>
<p align = “left”>This is left aligned </p>
<p align = “center”>This is center aligned </p>
<p align = “right”>This is right aligned </p>
</body>
</html>
```

6. Comment Tag:

Comment is a piece of code which is ignored by web browser. It is a good practice to add comment into HTML code, especially in complex documents, to indicate sections of a document and any other notes to anyone looking at the code. Comments help us and others understand our code and increases code readability.

HTML comments are placed in between <! -- Comment --> tags. So any content placed with in <! -- Comment -- > tags will be treated as comment and will be completely ignored by the browser.

7. Meta Tag:

HTML lets us specify metadata: additional important information about a document is a variety of ways. The META elements can be used to include name/value pairs describing properties of the HTML document, such as author, expiry data, a list of keywords, document author, etc.

The <meta> tag is used to provide such additional information. This tag is an empty element and so does not have a closing tab but it carries information within its attributes.

We can include one or more meta tags in our document based on what information we want to keep in our document but in general, meta tag do not impact physical appearance of the document so from appearance point of view, it does not matter if we include them or not.

We can add metadata to our web pages by placing <meta> tags inside the header of the document which is represented by <head> and </head> tags.

```
<head>
  <meta charset="UTF-8">
  <meta name="description" content="Free Web tutorials">
  <meta name="keywords" content="HTML,CSS,XML,JavaScript">
  <meta name="author" content="John Doe">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
</head>
```

HTML BASIS:

1. Text Formatting:

HTML also defines special elements for defining text with a special meaning. HTML uses elements like <b> and <i> for formatting output, like bold or italic text. Formatting elements were designed to display special types of text:

- <b> - Bold text
- - Important text
- <i> - Italic text
- - Emphasized text
- <mark> - Marked text
- <small> - Small text
- <del> - Deleted text
- <ins> - Inserted text

- <sub> - Subscript text
- <sup> - Superscript text

2. Phrase Element:

The phrase elements have been designed for specific purpose, though they are displayed in a similar way as other basic tags like <b>, <i>, <pre> and <tt>.

- **Emphasized Text:** Anything that appears within   element is displayed as emphasized text.
- **Marked Text:** Anything that appears within <mark> </mark> element is displayed as marked with yellow ink.
- **Strong Text:** Anything that appears within   element is displayed as important text.
- **Short Quotations:** The <q> </q> element is used when we want to add a double quote within a sentence.

3. Listing:

HTML offers web authors three ways for specifying lists of information. All lists must contain one or more list elements.

a. 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.

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

```
<ul type="square">
<ul type="disc">
<ul type="circle">
```

Example:

```
<!DOCTYPE html>
<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>
```

b. Ordered Lists:

If we are required to put our 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>.

We can use type attribute for <ol> tag to specify the type of numbering we 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.

We can use start attribute for <ol> tag to specify the starting point of numbering we 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:

```
<!DOCTYPE html>
<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>
</body>
</html>
```

c. Definition Lists:

HTML and XHTML support a list style which is called definition lists where entries are listed like in a dictionary or encyclopedia. The definition 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

Example

```
<!DOCTYPE html>
<html>
<head>
<title>HTML Definition List</title>
</head>
<body>
<dl>
<dt><b>HTML</b></dt>
<dd>This stands for Hyper Text Markup Language</dd>
<dt><b>HTTP</b></dt>
<dd>This stands for Hyper Text Transfer Protocol</dd>
</dl>
</body>
</html>
```

Nested List:

When we create a list with in another list then it is called nested list. For example:

```
<!DOCTYPE html>
<html>
<head>
<title>HTML Nested List</title>
</head>
<body>
  <ol type="square">
<li>Vegetable </li>
<ul>
  <li>Beetroot</li>
  <li>Ginger</li>
  <li>Potato</li>
  <li>Radish</li>
</ul>
  </ol>
</body>
</html>
```

4. Using Character Entities for Special Character:

There are few special characters or symbols which are not available to be typed directly from keyboard. Character Entities can be used to display those symbols/special characters also. Browsers may not support all entity names, but the support for numbers is good.

Some Useful HTML Character Entities

Result	Description	Entity Name	Entity Number
	Non-Breaking Space		
<	Less Than	&lt;	&#60;
>	Greater Than	&gt;	&#62;
&	Ampersand	&amp;	&#38;
"	Double Quotation Mark	&quot;	&#34;
'	Single Quotation Mark (Apostrophe)	&apos;	&#39;
¢	Cent	&cent;	&#162;
£	Pound	&pound;	&#163;
¥	Yen	&yen;	&#165;
€	Euro	&euro;	&#8364;
©	Copyright	&copy;	&#169;
®	Registered Trademark	&reg;	&#174;

5. <Font> Element and Attributes:

Fonts play very important role in making a website more user friendly and increasing content readability. Font face and color depends entirely on the computer and browser that is being used to view our page but we can use HTML tag to add style, size, and color to the text on our website.

The font tag is having three attributes called size, color, and face to customize our fonts. To change any of the font attributes at any time within our webpage, simply use the tag. The text that follows will remain changed until we close with the tag. We can change one or all of the font attributes within one <font> tag.

```
<!DOCTYPE html>
<html>
<head>
<title> Font Exmple </title>
</head>
<body>
<font face="Times New Roman" size="5" color = "Green">Use of Font element and attribute</font><br />
<font face="Impact" size="5" color = "red">Use of Font element and attribute</font><br />
</body>
</html>
```

6. Grouping Element:

There are two important tags which we use vary frequently to group various other HTML tags and they are:

a. ** tag:**

This is the very important block level tag which plays a big role in grouping various other HTML tags and applying CSS on group of elements. Even now tag can be used to create webpage layout where we define different parts (Left, Right, Top, and Bottom) of the page using tag. This tag does not provide any visual change on the block but this has more meaning when it is used with CSS.

```
<!DOCTYPE html>
<html>
<head>
<title>HTML div Tag</title>
<style type = "text/css" media = "all">
#contentinfo p {
   line-height: 20px;
         margin: 30px;
         padding-bottom: 20px;
         text-align: justify;
         width: 140px;
         color: red;
}
</style>
</head>
<body>
<div id="contentinfo">
<p>Welcome to our website. We provide tutorials on various subjects.</p>
</div>
</body>
</html>
```

b. ** tag:**

The HTML is an inline element and it can be used to group inline elements in an HTML document. This tag also does not provide any visual change on the block but has more meaning when it is used with CSS.

The main difference between the tag and the tag is that the tag is used with inline elements whereas the tag is used with block level elements.

```
<!DOCTYPE html>
<html>
<head>
<title>HTML span Tag</title>
</head>
<body>
<p>This is a paragraph <span style="color:#FF0000;">
This is a paragraph</span>
This is a paragraph</p>
<p><span style="color:#8866ff;">
```

```
This is another paragraph</span></p>
</body>
</html>
```

- **Block Elements:**

They appear on the screen as if they have a line break before and after them. For example: , <h1> </h1>, <ul> , <ol> <hr/>, <blockquote>, etc. They all start on their own new line, and anything that follows them appears on its own new line.

- **Inline Elements:**

They appear within sentence and do not have to appear on a new line of their own. For example: <b>, <i>, <u>, , , <sup>, <sub>, <big>, <small>, <li>, etc.

DIFFERENCE BETWEEN <DIV> TAG AND <SPAN> TAG:

<Div>	<Span>
is a block-level-element	is an inline element.
If we need to modify a large division, the height, move an element, or contain other elements we should use a	if we want to adjust a small portion of text and not break it out of the current line we should use a
Example: This is an example of a div tag that has a maximum width of 200 pixels and a silver background. 	Example: This text is red text and an example of a span tag.

LINKS AND NAVIGATION:

1. LINKING TO OTHER WEB PAGES:

A webpage can contain various links that take us directly to other pages and even specific parts of a given page. These links are known as hyperlinks. Hyperlinks allow visitors to navigate between Web sites by clicking on words, phrases, and images. Thus we can create hyperlinks using text or images available on a webpage.

- **Linking Documents:**

A link is specified using HTML tag <a>. This tag is called anchor tag and anything between the opening <a> tag and the closing tag becomes part of the link and a user can click that part to reach to the linked document. Following is the simple syntax to use <a> tag.

Link Text

Example:

```
<!DOCTYPE html>
<html>
```

```
<head>
<title>Hyperlink Example</title>
</head>
<body>
<p>Click following link</p>
<a href="https://www.facebook.com" target="_self">Tutorials Point</a>
</body>
</html>
```

The Target Attribute:

We have used target attribute in our previous example. This attribute is used to specify the location where linked document is opened. Following are possible options:

Option	Description
_blank	Opens the linked document in a new window or tab.
_self	Opens the linked document in the same frame.
_parent	Opens the linked document in the parent frame.
_top	Opens the linked document in the full body of the window.
targetframe	Opens the linked document in a named *targetframe*.

Example:

```
<!DOCTYPE html>
<html>
<head>
<title>Hyperlink Example</title>
<base href="https://www.facebook.com/">
</head>
<body>
<p>Click any of the following links</p>
<a href="/html/index.htm" target="_blank">Opens in New</a> |
<a href="/html/index.htm" target="_self">Opens in Self</a> |
<a href="/html/index.htm" target="_parent">Opens in Parent</a> |
<a href="/html/index.htm" target="_top">Opens in Body</a>
</body>
</html>
```

Use of Base Path:

When we link HTML documents related to the same website, it is not required to give a complete URL for every link. We can get rid of it if we use <base> tag in our HTML document header. This tag is used to give a base path for all the links. So our browser will concatenate given relative path to this base path and will make a complete URL.

Example:

```
<!DOCTYPE html>
<html>
<head>
<title>Hyperlink Example</title>
<base href="https://www.google.com/">
</head>
<body>
<p>Click following link</p>
<a href="/html/index.htm" target="_blank">Google</a>
</body>
</html>
```

- **Linking to a Page Section:**

We can create a link to a particular section of a given webpage by using name attribute. This is a two-step process:

- First create a link to the place where we want to reach with-in a webpage and name it using <a...> tag as follows:
  <h1>HTML Text Links </h1>
- Second step is to create a hyperlink to link the document and place where we want to reach:
  Go to the Top

This will produce following link, where we can click on the link generated **Go to the Top** to reach to the top of the HTML Text Link tutorial.

- **Setting Link Colors:**

We can set colors of our links, active links and visited links using link, alink and vlink attributes of <body> tag.

Example:

```
<!DOCTYPE html>
<html>
<head>
<title>Hyperlink Example</title>
</head>
<body alink="#54A250" link="#040404" vlink="#F40633">
<p>Click following link</p>
<a href="https://www.facebook.com" target="_blank" >Facebook</a>
</body>
</html>
```

- **Download Links:**

We can create text link to make our PDF, or DOC or ZIP files downloadable. This is very simple, we just need to give complete URL of the downloadable file as follows:

Example:

```
<!DOCTYPE html>
<html>
<head>
<title>Hyperlink Example</title>
</head>
<a href="https://www.tutorialspoint.com/page.pdf">Download PDF File</a>
</body>
</html>
```

2. URL (Uniform Resource Locator):

A **URL** (**Uniform Resource Locator**) is a form of URI (Uniform Resource Identifier) and is a standardized naming convention for addressing documents accessible over the Internet and Intranet. An example of a URL is http://www.computerhope.com, which is the URL for the Computer Hope website.

- **Overview of a URL:**

- **http://**

The "http" stands for HyperText Transfer Protocol and is what enables the browser to know what protocol it is going to use to access the information specified in the domain. After the http is the colon (:) and two forward slashes (//) that separate the protocol from the remainder of the URL.

Tip: A URL is not explicit to HTTP addresses; HTTPS, FTP, TFTP, Telnet, and other addresses are also considered URLs and may not follow the same syntax as above example.

- **www.**

Next, www. Stands for World Wide Web and is used to distinguish the content. This portion of the URL is not required and many times can be left out. For example, typing "http://computerhope.com" would still get us to the Computer Hope web page. This portion of the address can also be substituted for an important subpage known as a subdomain. For example, http://support.computerhope.com forwards us to the main help section of Computer Hope.

- **computerhope.com**

Next, computerhope.com is the domain name for the website. The last portion of the domain is known as the "domain suffix", or TLD, and is used to identify the type or location of the website. For example, **.com** is short for commercial, .org is short for an organization, and .co.uk is the United Kingdom. There are dozens of other domain suffixes available. To get a domain, we would register the name through a domain registrar.

- **/jargon/u/**

Next, the "jargon" and "u" portions of the above URL are the directories of where on the server the web page is located. In this example, the web page is two directories deep, so if we were trying to find the file on the server, it would be in the */public_html/jargon/u* directory. With most servers, the public_html directory is the default directory containing the HTML files.

- **url.htm**

Finally, url.htm is the actual web page on the domain we're viewing. The trailing .htm is the file extension of the web page that indicates the file is an HTML file. Other common file extensions on the Internet include .html, .php, .asp, .cgi, .xml, .jpg, and .gif. Each of these file extensions performs a different function, just like all the different types of files on your computer.

- **What characters are not allowed in a URL?**

Most people realize that a space is not allowed in a URL. However, it is also important to realize, as documented in RFC 1738, the URL string can only contain alphanumeric characters and the ! $ -_ + * ' ( ) , characters. Any other characters that are needed in the URL must be encoded.

- **Understanding more complex URLs and parameters**

When a URL points to a script that performs additional functions, such as a search engine pointing to a search results page, additional information (parameters) is added to the end of the URL. Below is additional information about a URL that points to the Computer Hope Search page, with the search query of "example search".

http://www.computerhope.com/cgi-bin/search.cgi?q=example%20search

In this URL, the script file being pointed to is **search.cgi** in the cgi-bin directory. Because this file ends with .cgi, it is assumed to be a Perl script.

After the script file name there is a **?** (question mark). The question mark in a URL separates the URL from all the parameters or variables that are being sent to the script. In the above example, the parameter being sent is **q=example%20search**. The "q" is a variable name, and the "example%20search" is the value being sent to that variable. Because no spaces are allowed in a URL, the space has been encoded as %20. In many scripts, a **+** (plus) is also used to represent a space.

In our example, because there is a variable the script would use it as it is executed. Scripts are also not limited to only one variable. If the script needs multiple variables, each variable can be separated with an **&** (ampersand) as shown in the example below.

http://www.computerhope.com/cgi-bin/search.cgi?q=example%20search&example=test

In the above example, there are two different variables. The "q" variable equals "example search" and the "example" variable equals "test". If the script was looking for an *example* variable, it could be processed and perform an additional feature.

- **URI (Uniform Resource Identifier):**

Short for Uniform Resource Identifier, URI is defined in RFC (Remote Function Call) 1630 as a reference to addresses, names, or objects that apply to registered protocols or name spaces on the Internet. For example, URL and URN are forms of Uniform Resource Identifiers.

3. Types of URL:

- **Absolute URL:**

In addition to several other meanings, the word **absolute**, in English, means "*not dependent on anything else*". It also means "*free from doubt*". An **Absolute URL** is, thus, something that is *independent* or *free from any relationship*. When we use an absolute URL, we point directly to a file. Hence, an absolute URL specifies the *exact* location of a file/directory on the internet. It also follows that each absolute URL is unique, which means that if two URLs are identical, they point to the same file.

For example:

- **http://www.webdevelopersnotes.com/images/email.gif** specifies an image file *email.gif* located in the *images* directory, under www.webdevelopersnotes.com domain name.
- Similarly, the absolute URL of the document we are viewing is **http://www.webdevelopersnotes.com/design/relative_and_absolute_urls.php3** which is a page in the directory called *design* on this web site.

- **Relative URL:**

A relative URL points to a file/directory in relation to the present file/directory. Let us understand relative URLs through a small exercise.
Look at the two URL above. We want to include (display) the image file **email.gif** stored in the images directory of **www.webdevelopersnotes.com** domain on this (**relative_and_absolute_urls.php3** stored in the **design** directory) page.

There are two ways to do this. We can either refer to it using an absolute URL or use a relative URL. The tag for this image display will be as follows:

- **Using an Absolute URL in an tag:**

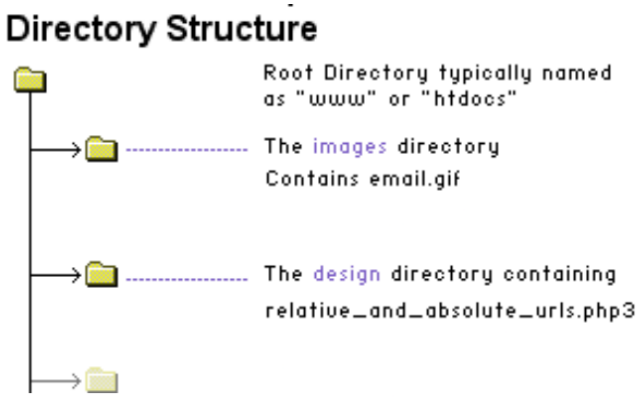
```
<img src="http://www.webdevelopersnotes.com/images/email.gif"width="..." height="..." />
```

- **Using a Relative URL in an tag**

```
<img src="../images/email.gif" width="..." height="..." />
```

The absolute URL is straight forward but in the relative URL we'll notice that we have referred to the image with **../images/email.gif**. In order to understand the relative URL, we need to know about the directory structure of this web site.
This web site has several sections and the files and web pages for each section have been segregated into different directories. This helps us to keep things organized and uncluttered on the web site. Under the document or server root directory (the main directory of the web site),

we have a directory called images which stores all common images used on the pages of this web site. The **image** email.gif resides in this directory. We have another directory called **design** which is at the same level as images i.e. it is also in the document root directory. This design directory contains the files and web pages for the "Web Page Design" section of this web site. Diagrammatically, the scenario can be represented as:



Now to access email.gif file from relative_and_absolute_urls.php3 page using a relative URL we put ../images/email.gif in the SRC attribute. We, thus, instruct the browser to first go one level up (i.e. to the document root) and then move to the images directory and pick up the file email.gif. The two periods (..) instruct the server to move up one directory (which is the root directory), then enter images directory (/images) and finally point at email.gif.

4. HTML Email Tag:

HTML <a> tag provides us option to specify an email address to send an email. While using <a> tag as an email tag, we will use **mailto:email address** along with *href* attribute. Following is the syntax of using **mailto** instead of using http.

Send Email

This code will generate following link which we can use to send email: Send Email

Now if a user clicks this link, it launches one Email Client (like Lotus Notes, Outlook Express etc.) installed on our user's computer. There is another risk to use this option to send email because if user do not have email client installed on their computer then it would not be possible to send email.

- **Default Settings:**

We can specify a default *email subject* and *email body* along with our email address. Following is the example to use default subject and body.

**Send Feedback **

This code will generate following link which we can use to send email: Send Feedback

- **Host Address:**

The host address is where a website can be found, either IP address (four sets of numbers from 0 to 258, for example 68.178.157.132) or more commonly the domain name for a site such as www.computerhope.com. Note that "www" is not actually part of the domain name though it is often used in the host address.

- **File Path:**

The file path always begins with a forward slash character, and may consist of one or more directory or folder names. Each directory name is separated by forward slash characters and the file path may end with a filename at the end. Here index.html is the filename which is available in html directory: https://www.computerhope.com/html/index.htm.

- **Local Links:**

A local link (link to the same web site) is specified with a relative URL (without http://www....).

Example

```
<a href="html_images.asp">HTML Images</a>
```

- **External Paths:**

External pages can be referenced with a full URL or with a path relative to the current web page. This example uses a full URL to link to a web page:

Example

```
<a href="https://www.w3schools.com/html/default.asp">HTML tutorial</a>
```

5. Advance Email Links:

Now a days due to improvement in technology and different approaches of programming we have seen different methods of sending emails. These all methods used are especially for security purpose. They are used with anti-spam, certain formats, anti-spam sounds.

Images, Audio and Video:

1. Image:

- **Inserting Image:**

Images are very important to beautify as well as to depict many complex concepts in simple way on our web page. We can insert any image in our web page by using **<img>** tag. Following is the simple syntax to use this tag.

******

The <img> tag is an empty tag, which means that it can contain only list of attributes and it has no closing tag.

Example:

```
<!DOCTYPE html>
<html>
<head>
<title>Using Image in Webpage</title>
</head>
<body>
<p>Simple Image Insert</p>
<img src="/html/images/test.png" alt="Test Image" />
</body>
</html>
```

We can use PNG, JPEG or GIF image file based on our comfort but make sure we specify correct image file name in **src** attribute. Image name is always case sensitive. The **alt** attribute is a mandatory attribute which specifies an alternate text for an image, if the image cannot be displayed.

- **Set Image Location:**

Usually we keep our all the images in a separate directory. So let's keep HTML file test.htm in our home directory and create a subdirectory **images** inside the home directory where we will keep our image test.png. ***Example:*** "/html/image/test.png"

- **Set Image Width/Height:**

We can set image width and height based on our requirement using **width** and **height** attributes. We can specify width and height of the image in terms of either pixels or percentage of its actual size.

Example

```
<!DOCTYPE html>
<html>
<head>
<title>Set Image Width and Height</title>
</head>
<body>
<p>Setting image width and height</p>
<img src="/html/images/test.png" alt="Test Image" width="150" height="100"/>
</body>
</html>
```

- **Set Image Border:**

By default image will have a border around it, we can specify border thickness in terms of pixels using **border** attribute. A thickness of 0 means, no border around the picture.

Example

```
<!DOCTYPE html>
```

```
<html>
<head>
<title>Set Image Border</title>
</head>
<body>
<p>Setting image Border</p>
<img src="/html/images/test.png" alt="Test Image" border="3"/>
</body>
</html>
```

- **Set Image Alignment:**

By default image will align at the left side of the page, but we can use **align** attribute to set it in the center or right.

Example

```
<!DOCTYPE html>
<html>
<head>
<title>Set Image Alignment</title>
</head>
<body>
<p>Setting image Alignment</p>
<img src="/html/images/test.png" alt="Test Image" border="3" align="right"/>
</body>
</html>
```

2. Images as Link:

We have seen how to create hypertext link using text and we also learnt how to use images in our webpages. Now we will learn how to use images to create hyperlinks. It's simple to use an image as hyperlink. We just need to use an image inside hyperlink at the place of text as shown below:

Example:

```
<!DOCTYPE html>
<html>
<head>
<title>Image Hyperlink Example</title>
</head>
<body>
<p>Click Image for Facbook</p>
<a href="https://www.facebook.com" target="_self">
<img src="\HTML Programs\on.jpg" alt="Facbook" border="0"/>
</a>
</body>
</html>
```

Mouse-Sensitive Images:

The HTML and XHTML standards provide a feature that lets us embed many different links inside a single image. We can create different links on the single image based on different coordinates available on the image. Once different are links attached to different coordinates, we can click different parts of the image to open target documents. Such mouse-sensitive images are known as image maps.

There are two ways to create image maps:

- **Server-side image maps -** This is enabled by the **ismap** attribute of the <img> tag and requires access to a server and related image-map processing applications.
- **Client-side image maps -** This is created with the **usemap** attribute of the <img> tag, along with corresponding <map> and <area> tags.

Server-Side Image Maps:

Here we simply put our image inside a hyper link and use **ismap** attribute which makes it special image and when the user clicks some place within the image, the browser passes the coordinates of the mouse pointer along with the URL specified in the <a> tag to the web server. The server uses the mouse-pointer coordinates to determine which document to deliver back to the browser.

When *ismap* is used, the href attribute of the containing <a> tag must contain the URL of a server application like a cgi or PHP script etc. to process the incoming request based on the passed coordinates.

The coordinates of the mouse position are screen pixels counted from the upper-left corner of the image, beginning with (0, 0). The coordinates, preceded by a question mark, are added to the end of the URL.

Example:

```
<!DOCTYPE html>
<html>
<head>
<title>ISMAP Hyperlink Example</title>
</head>
<body>
<p>Click following link</p>
<a href="/cgi-bin/ismap.cgi" target="_self">
  <img ismap src="/images/logo.png" alt="Tutorials Point" border="0"/>
</a>
</body>
</html>
```

Then the browser sends the following search parameters to the web server which can be processed by **ismap.cgi** script or **map file** and we can link whatever documents we like to these coordinates: ***/cgi-bin/ismap.cgi?20,30***

This way we can assign different links to different coordinates of the image and when those coordinates are clicked, we can open corresponding linked document.

- **Client-Side Image Maps:**

Client side image maps are enabled by the **usemap** attribute of the <img /> tag and defined by special <map> and <area> extension tags.

The image that is going to form the map is inserted into the page using the <img /> tag as a normal image, except it carries an extra attribute called **usemap**. The value of the usemap attribute is the value which will be used in a <map> tag to link map and image tags. The <map> along with <area> tags define all the image coordinates and corresponding links.

The <area> tag inside the map tag, specifies the shape and the coordinates to define the boundaries of each clickable hotspot available on the image. Here's an example from the image map:

Example:

```
<!DOCTYPE html>
<html>
<head>
<title>USEMAP Hyperlink Example</title>
</head>
<body>
<p>Search and click the hotspot</p>
<img src=/images/html.gif alt="HTML Map" border="0" usemap="#html"/>
<!-- Create  Mappings -->
<map name="html">
  <area shape="circle"
  coords="80,80,20" href="/css/index.htm" alt="CSS Link" target="_self" />
  <area shape="rect"
  coords="5,5,40,40" alt="jQuery Link" href="/jquery/index.htm" target="_self" />
</map>
</body>
</html>
```

- **Coordinate System:**

The actual value of coords is totally dependent on the shape in question. Here is a summary, to be followed by detailed examples:

rect = x_1 , y_1 , x_2 , y_2

x_1 and y_1 are the coordinates of the upper left corner of the rectangle; x_2 and y_2 are the coordinates of the lower right corner.

circle = x_c , y_c , radius

x_c and y_c are the coordinates of the center of the circle, and radius is the circle's radius. A circle centered at 200, 50 with a radius of 25 would have the attribute *coords="200, 50,25"*

poly = x_1 , y_1 , x_2 , y_2 , x_3 , y_3 , ... x_n , y_n

The various x-y pairs define vertices (points) of the polygon, with a "line" being drawn from one point to the next point. A diamond-shaped polygon with its top point at 20, 20 and 40 pixels across at its widest points would have the attribute *coords="20,20,40,40,20,60,0,40"*.

All coordinates are relative to the upper-left corner of the image (0, 0). Each shape has a related URL. We can use any image software to know the coordinates of different positions.

3. Image Format:

Each of image file types has their own pros and cons. They were created for specific, yet different, purposes. What's the difference, and when is each format appropriate to use?

- **JPEG:**

JPEG is short for Joint Photographic Experts Group, and is the most popular among the image formats used on the web. JPEG files are very 'lossy', meaning so much information is lost from the original image when you save it in a JPEG file.

This is because JPEG discards most of the information to keep the image file size small; which means some degree of quality is also lost.

Almost every digital camera can shoot and save in the JPEG format. JPEG is very web friendly because the file is smaller, which means it takes up less room, and requires less time to transfer to a site. Moreover it is less grainy then GIF, the old king of the internet roost. Since 1994, JPEG has been considered the standard.

Pros of JPEG:

- 24-bit color, with up to 16 million colors
- Rich colors, great for photographs that need fine attention to color detail
- Most used and most widely accepted image format
- Compatible in most OS (Mac, PC, Linux)

Cons of JPEG:

- They tend to discard a lot of data
- After compression, JPEG tends to create artifacts
- Cannot be animated
- Does not support transparency

- **GIF:**

GIF, short for Graphics Interchange Format, is limited to the 8 bit palette with only 256 colors. GIF is still a popular image format on the internet because image size is relatively small compared to other image compression types.

GIF is most suitable for graphics, diagrams, cartoons and logos with relatively few colors. GIF is still the chosen format for animation effects.

Compared to JPEG, it is lossless and thus more effective with compressing images with a single color, but pales in detailed or dithered pictures. In other words, GIF is lossless for images with 256 colors and below. So for a full color image, it may lose up to 99.998% of its colors.

One edge of the GIF image format is the interlacing feature, giving the illusion of fast loading graphics. When it loads in a browser, the GIF first appears to be blurry and fuzzy, but as soon as more data is downloaded, the image becomes more defined until all the date has been downloaded.

Pros of GIF:

- Can support transparency
- Can do small animation effects
- 'Lossless' quality–they contain the same amount of quality as the original, except of course it now only has 256 colors
- Great for images with limited colors, or with flat regions of color

Cons of GIF:

- Only supports 256 colors
- It's the oldest format in the web, having existed since 1989. It hasn't been updated since, and sometimes, the file size is larger than PNG.

BMP:

The Windows Bitmap or BMP files are image files within the Microsoft Windows operating system. In fact, it was at one point one of the few image formats. These files are large and uncompressed, but the images are rich in color, high in quality, simple and compatible in all Windows OS and programs. BMP files are also called raster or paint images.

BMP files are made of millions and millions of dots called 'pixels,' with different colors and arrangements to come up with an image or pattern. It might be an 8-bit, 16-bit or 24-bit image. Thus when we make a BMP image larger or smaller, we are making the individual pixels larger, and thus making the shapes look fuzzy and jagged.

BMP files are not great and not very popular. Being oversized, bitmap files are not what we call 'web friendly', nor are they compatible in all platforms and they do not scale well.

Pros of BMP:

- Works well with most Windows programs and OS, you can use it as a Windows wallpaper

Cons of BMP:

- Does not scale or compress well
- Again, very huge image files making it not web friendly
- No real advantage over other image formats

TIFF:

TIFF was created by Aldus for 'desktop publishing', and by 2009 it was transferred to the control of Adobe Systems. TIFF is popular among common users, but has gained recognition in the graphic design, publishing and photography industry. It is also popular among Apple users.

The TIFF image format is easy to use with software that deals with page layout, publishing and photo manipulation via fax, scanning, word processing, etc. TIFF is very flexible, it can be lossy or lossless. TIFF is a rich format and supported by many imaging programs.

It is capable of recording halftone image data with different pixel intensities, thus is the perfect format for graphic storage, processing and printing. This makes TIFF the superior raster image format.

Pros of TIFF:

- Very flexible format, it supports several types of compression like JPEG, LZW, ZIP or no compression at all.
- High quality image format, all color and data information are stored
- TIFF format can now be saved with layers

Cons of TIFF:

- Very large file size–long transfer time, huge disk space consumption, and slow loading time.

PNG

PNG or (Portable Network Graphics) is a recently introduced format, so not everyone is familiar with it. But PNG has been approved as a standard since 1996. It is an image format specifically designed for the web. PNG is, in all aspects, the superior version of the GIF.

Just like the GIF format, the PNG is saved with 256 colors maximum but it saves the color information more efficiently. It also supports an 8 bit transparency.

PNG was actually created for the intent to replace the GIF as an image format that doesn't require a patent license. PNG can support 24 bit RGB color images, grayscale images, both with and without alpha channels. RGB cannot support CMYK color spaces, and is not designed for print graphics.

Pros of PNG:

- Lossless, so it does not lose quality and detail after image compression
- In a lot ways better then GIF. To start, PNG often creates smaller file sizes than GIF
- Supports transparency better than GIF

Cons of PNG:

- Not good for large images because they tend to generate a very large file, sometimes creating larger files than JPEG.
- Unlike GIF however, it cannot be animated.
- Not all web browsers can support PNG.

4. Working with Multimedia:

Sometimes we need to add music or video into our web page. The easiest way to add video or sound to our web site is to include the special HTML tag called **<embed>**. This tag causes the browser itself to include controls for the multimedia automatically provided browser supports <embed> tag and given media type.

We can also include a **<noembed>** tag for the browsers which don't recognize the <embed> tag. We could, for example, use <embed> to display a movie of our choice, and <noembed> to display a single JPG image if browser does not support <embed> tag.

Example:

```
<!DOCTYPE html>
<html>
<head>
<title>HTML embed Tag</title>
</head>
<body>
<embed src="/html/yourfile.mid" width="100%" height="60" >
  <noembed><img src="yourimage.gif" alt="Alternative Media" ></noembed>
</embed>
</body>
</html>
```

- **The <embed> Tag Attributes:**

Following is the list of important attributes which can be used with <embed> tag.

Attribute	Description
align	Determines how to align the object. It can be set to either *center, left or right*.
autostart	This Boolean attribute indicates if the media should start automatically. We can set it either true or false.
loop	Specifies if the sound should be played continuously (set loop to true), a certain number of times (a positive value) or not at all (false)
playcount	Specifies the number of times to play the sound. This is alternate option for *loop* if we are using IE.
hidden	Specifies if the multimedia object should be shown on the page. A false value means no and true values means yes.
width	Width of the object in pixels
height	Height of the object in pixels
name	A name used to reference the object.
src	URL of the object to be embedded.
volume	Controls volume of the sound. Can be from 0 (off) to 100 (full volume).

- **Multimedia Formats:**

Multimedia elements (like audio or video) are stored in media files. The most common way to discover the type of a file, is to look at the file extension. Multimedia files have formats and different extensions like: .swf, .wav, .mp3, .mp4, .mpg, .wmv, and .avi.

- **Common Video Formats:**

MP4 is the new and upcoming format for internet video.

- MP4 is recommended by YouTube.

- MP4 is supported by Flash Players.
- MP4 is supported by HTML5.

Format	File	Description
MPEG	.mpg .mpeg	MPEG. Developed by the Moving Pictures Expert Group. The first popular video format on the web. Used to be supported by all browsers, but it is not supported in HTML5 (See MP4).
AVI	.avi	AVI (Audio Video Interleave). Developed by Microsoft. Commonly used in video cameras and TV hardware. Plays well on Windows computers, but not in web browsers.
WMV	.wmv	WMV (Windows Media Video). Developed by Microsoft. Commonly used in video cameras and TV hardware. Plays well on Windows computers, but not in web browsers.
QuickTime	.mov	QuickTime. Developed by Apple. Commonly used in video cameras and TV hardware. Plays well on Apple computers, but not in web browsers. (See MP4)
RealVideo	.rm .ram	RealVideo. Developed by Real Media to allow video streaming with low bandwidths. It is still used for online video and Internet TV, but does not play in web browsers.
Flash	.swf .flv	Flash. Developed by Macromedia. Often requires an extra component (plug-in) to play in web browsers.
Ogg	.ogg	Theora Ogg. Developed by the Xiph.Org Foundation. Supported by HTML5.
WebM	.webm	WebM. Developed by the web giants, Mozilla, Opera, Adobe, and Google. Supported by HTML5.
MPEG-4 or MP4	.mp4	MP4. Developed by the Moving Pictures Expert Group. Based on QuickTime. Commonly used in newer video cameras and TV hardware. Supported by all HTML5 browsers. Recommended by YouTube.

Example:

```
<!DOCTYPE html>
<html>
<head>
<title>HTML embed Tag</title>
</head>
<body>
<embed src="/html/yourfile.swf" width="200" height="200" >
  <noembed><img src="yourimage.gif" alt="Alternative Media" ></noembed>
</embed>
</body>
</html>
```

Audio Formats:

MP3 is the newest format for compressed recorded music. The term MP3 has become synonymous with digital music. If our website is about recorded music, MP3 is the choice.

Format	File	Description
MIDI	.mid .midi	MIDI (Musical Instrument Digital Interface). Main format for all electronic music devices like synthesizers and PC sound cards. MIDI files do not contain sound, but digital notes that can be played by electronics. Plays well on all computers and music hardware, but not in web browsers.
RealAudio	.rm .ram	RealAudio. Developed by Real Media to allow streaming of audio with low bandwidths. Does not play in web browsers.
WMA	.wma	WMA (Windows Media Audio). Developed by Microsoft. Commonly used in music players. Plays well on Windows computers, but not in web browsers.
AAC	.aac	AAC (Advanced Audio Coding). Developed by Apple as the default format for iTunes. Plays well on Apple computers, but not in web browsers.
WAV	.wav	WAV. Developed by IBM and Microsoft. Plays well on Windows, Macintosh, and Linux operating systems. Supported by HTML5.
Ogg	.ogg	Ogg. Developed by the Xiph.Org Foundation. Supported by HTML5.
MP3	.mp3	MP3 files are actually the sound part of MPEG files. MP3 is the most popular format for music players. Combines good compression (small files) with high quality. Supported by all browsers.
MP4	.mp4	MP4 is a video format, but can also be used for audio. MP4 video is the upcoming video format on the internet. This leads to automatic support for MP4 audio by all browsers.

Background Audio:

We can use HTML **<bgsound>** tag to play a soundtrack in the background of our webpage. This tag is supported by Internet Explorer only and most of the other browsers ignore this tag. It downloads and plays an audio file when the host document is first downloaded by the user and displayed. The background sound file also will replay whenever the user refreshes the browser.

This tag is having only two attributes *loop* and *src*. Both these attributes have same meaning as explained above.

Here is a simple example to play a small midi file:

```
<!DOCTYPE html>
<html>
<head>
<title>HTML embed Tag</title>
</head>
<body>
<bgsound src="/html/yourfile.mid">
  <noembed><img src="yourimage.gif" ></noembed>
</bgsound>
</body>
```

```
</html>
```

HTML Object tag:

HTML 4 introduces the **<object>** element, which offers an all-purpose solution to generic object inclusion. The <object> element allows HTML authors to specify everything required by an object for its presentation by a user agent

Here are few examples:

a. *Example - 1*

We can embed an HTML document in an HTML document itself as follows:

```
<object data="data/test.htm" type="text/html" width="300" height="200">
 alt : <a href="data/test.htm">test.htm</a>
</object>
```

Here *alt* attribute will come into picture if browser does not support *object* tag.

b. *Example - 2*

We can embed a PDF document in an HTML document as follows:

```
<object data="data/test.pdf" type="application/pdf" width="300" height="200">
 alt : <a href="data/test.pdf">test.htm</a>
</object>
```

c. *Example - 3*

We can specify some parameters related to the document with the **<param>** tag. Here is an example to embed a wav file:

```
<object data="data/test.wav" type="audio/x-wav" width="200" height="20">
 <param name="src" value="data/test.wav">
 <param name="autoplay" value="false">
 <param name="autoStart" value="0">
 alt : <a href="data/test.wav">test.wav</a>
</object>
```

d. *Example - 4*

We can add a flash document as follows:

```
<object classid="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000" id="penguin"
  codebase="someplace/swflash.cab" width="200" height="300">
  <param name="movie" value="flash/penguin.swf" />
  <param name="quality" value="high" />
  <img src="penguin.jpg" width="200" height="300" alt="Penguin" />
</object>
```

e. *Example - 5*

We can add a java applet into HTML document as follows:

```
<object classid="clsid:8ad9c840-044e-11d1-b3e9-00805f499d93"
  width="200" height="200">
  <param name="code" value="applet.class">
</object>
```

The **classid** attribute identifies which version of Java Plug-in to use. We can use the optional *codebase* attribute to specify if and how to download the JRE.

TABLE:

The HTML tables allow web authors to arrange data like text, images, links, other tables, etc. into rows and columns of cells. The HTML tables are created using the **<table>** tag in which the **<tr>** tag is used to create table rows and **<td>** tag is used to create data cells.

Example:

```
<!DOCTYPE html>
<html>
<head>
<title>HTML Tables</title>
</head>
<body>
<table border="1">
<tr>
<td>Row 1, Column 1</td>
<td>Row 1, Column 2</td>
</tr>
<tr>
<td>Row 2, Column 1</td>
<td>Row 2, Column 2</td>
</tr>
</table>
</body>
</html>
```

1. Table Heading:

Table heading can be defined using **<th>** tag. This tag will be put to replace <td> tag, which is used to represent actual data cell. Normally we will put our top row as table heading as shown below, otherwise we can use <th> element in any row.

Example

```
<!DOCTYPE html>
<html>
```

```
<head>
<title>HTML Table Header</title>
</head>
<body>
<table border="1">
<tr>
<th>Name</th>
<th>Salary</th>
</tr>
<tr>
<td>Ramesh Raman</td>
<td>5000</td>
</tr>
<tr>
<td>Shabbir Hussein</td>
<td>7000</td>
</tr>
</table>
</body>
</html>
```

2. Cellpadding and Cellspacing Attributes:

There are two attribiutes called *cellpadding* and *cellspacing* which we will use to adjust the white space in our table cells. The cellspacing attribute defines the width of the border, while cellpadding represents the distance between cell borders and the content within a cell.

Example

```
<!DOCTYPE html>
<html>
<head>
<title>HTML Table Cellpadding</title>
</head>
<body>
<table border="1" cellpadding="5" cellspacing="5">
<tr>
<th>Name</th>
<th>Salary</th>
</tr>
<tr>
<td>Ramesh Raman</td>
<td>5000</td>
</tr>
<tr>
<td>Shabbir Hussein</td>
<td>7000</td>
</tr>
```

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

3. Colspan and Rowspan Attributes:

We will use **colspan** attribute if we want to merge two or more columns into a single column. Similar way we will use **rowspan** if we want to merge two or more rows.

Example

```
<!DOCTYPE html>
<html>
<head>
<title>HTML Table Colspan/Rowspan</title>
</head>
<body>
<table border="1">
<tr>
<th>Column 1</th>
<th>Column 2</th>
<th>Column 3</th>
</tr>
<tr><td rowspan="2">Row 1 Cell 1</td><td>Row 1 Cell 2</td><td>Row 1 Cell 3</td></tr>
<tr><td>Row 2 Cell 2</td><td>Row 2 Cell 3</td></tr>
<tr><td colspan="3">Row 3 Cell 1</td></tr>
</table>
</body>
</html>
```

4. Tables Backgrounds:

We can set table background using one of the following two ways:

- **bgcolor** attribute - We can set background color for whole table or just for one cell.
- **background** attribute - We can set background image for whole table or just for one cell.

We can also set border color also using **bordercolor** attribute.

Example

```
<!DOCTYPE html>
<html>
<head>
<title>HTML Table Background</title>
</head>
<body>
<table border="1" bordercolor="green" bgcolor="yellow">
<tr>
```

```
<th>Column 1</th>
<th>Column 2</th>
<th>Column 3</th>
</tr>
<tr><td rowspan="2">Row 1 Cell 1</td><td>Row 1 Cell 2</td><td>Row 1 Cell 3</td></tr>
<tr><td>Row 2 Cell 2</td><td>Row 2 Cell 3</td></tr>
<tr><td colspan="3">Row 3 Cell 1</td></tr>
</table>
</body>
</html>
```

Here is an example of using background attribute. Here we will use an image available in /images directory.

```
<!DOCTYPE html>
<html>
<head>
<title>HTML Table Background</title>
</head>
<body>
<table border="1" bordercolor="green" background="/images/test.png">
<tr>
<th>Column 1</th>
<th>Column 2</th>
<th>Column 3</th>
</tr>
<tr><td rowspan="2">Row 1 Cell 1</td><td>Row 1 Cell 2</td><td>Row 1 Cell 3</td></tr>
<tr><td>Row 2 Cell 2</td><td>Row 2 Cell 3</td></tr>
<tr><td colspan="3">Row 3 Cell 1</td></tr>
</table>
</body>
</html>
```

5. Table Height and Width:

We can set a table width and height using **width** and **height** attrubutes. We can specify table width or height in terms of pixels or in terms of percentage of available screen area.

Example

```
<!DOCTYPE html>
<html>
<head>
<title>HTML Table Width/Height</title>
</head>
<body>
<table border="1" width="400" height="150">
```

```
<tr>
<td>Row 1, Column 1</td>
<td>Row 1, Column 2</td>
</tr>
<tr>
<td>Row 2, Column 1</td>
<td>Row 2, Column 2</td>
</tr>
</table>
</body>
</html>
```

6. Table Caption:

The **caption** tag will serve as a title or explanation for the table and it shows up at the top of the table. This tag is deprecated in newer version of HTML/XHTML.

Example

```
<!DOCTYPE html>
<html>
<head>
<title>HTML Table Caption</title>
</head>
<body>
<table border="1" width="100%">
<caption>This is the caption</caption>
<tr>
<td>row 1, column 1</td><td>row 1, columnn 2</td>
</tr>
<tr>
<td>row 2, column 1</td><td>row 2, columnn 2</td>
</tr>
</table>
</body>
</html>
```

7. Table Header, Body, and Footer:

Tables can be divided into three portions: a header, a body, and a foot. The head and foot are rather similar to headers and footers in a word-processed document that remain the same for every page, while the body is the main content holder of the table.

The three elements for separating the head, body, and foot of a table are:

- **<thead> -** to create a separate table header.
- **<tbody> -** to indicate the main body of the table.
- **<tfoot> -** to create a separate table footer.

A table may contain several <tbody> elements to indicate different *pages* or groups of data. But it is notable that <thead> and <tfoot> tags should appear before <tbody>

Example

```
<!DOCTYPE html>
<html>
<head>
<title>HTML Table</title>
</head>
<body>
<table border="1" width="100%">
<thead>
<tr>
<td colspan="4">This is the head of the table</td>
</tr>
</thead>
<tfoot>
<tr>
<td colspan="4">This is the foot of the table</td>
</tr>
</tfoot>
<tbody>
<tr>
<td>Cell 1</td>
<td>Cell 2</td>
<td>Cell 3</td>
<td>Cell 4</td>
</tr>
</tbody>
</table>
</body>
</html>
```

8. Nested Tables:

We can use one table inside another table. Not only tables we can use almost all the tags inside table data tag <td>.

Example

Following is the example of using another table and other tags inside a table cell.

```
<!DOCTYPE html>
<html>
<head>
<title>HTML Table</title>
</head>
<body>
```

```
<table border="1" width="100%">
<tr>
<td>
  <table border="1" width="100%">
  <tr>
  <th>Name</th>
  <th>Salary</th>
  </tr>
  <tr>
  <td>Ramesh Raman</td>
  <td>5000</td>
  </tr>
  <tr>
  <td>Shabbir Hussein</td>
  <td>7000</td>
  </tr>
  </table>
</td>
</tr>
</table>
</body>
</html>
```

9. Grouping Section of Table:

HTML <colgroup> Tag:

The <colgroup> tag specifies a group of one or more columns in a table for formatting. The <colgroup> tag is useful for applying styles to entire columns, instead of repeating the styles for each cell, for each row.

Note: The <colgroup> tag must be a child of a <table> element, after any <caption> elements and before any <thead>, <tbody>, <tfoot>, and <tr> elements.

Tip: To define different properties to a column within a <colgroup>, use the <col> tag within the <colgroup> tag.

Example:

```
<!DOCTYPE html>
<html>
<head>
<title>Table Example</title>
</head>
<body>
<table border = "1" cellpadding = "3" cellspacing = "0">
 <colgroup>
  <col span="2" style="background-color:cyan">
   <col style="background-color:yellow">
```

ISBN	Title	Quantity	Price
3476896	Motorola	15	$53
3476897	Samsung	20	$55

```
        <col style="background-color:red">

  </colgroup>
  <tr>
   <th>ISBN</th>
   <th>Title</th>
        <th>Quantity</th>
   <th>Price</th>
  </tr>
  <tr>
   <td>3476896</td>
   <td>Motorola</td>
        <td>15</td>
   <td>$53</td>
  </tr>
   <tr>
   <td>3476897</td>
   <td>Samsung</td>
        <td>20</td>
   <td>$55</td>
  </tr>
 </table>
 </body>
 </html>
```

- **HTML <thead> Tag:**

The <thead> tag is used to group header content in an HTML table. The <thead> element is used in conjunction with the <tbody> and <tfoot>elements to specify each part of a table (header, body, footer).

Browsers can use these elements to enable scrolling of the table body independently of the header and footer. Also, when printing a large table that spans multiple pages, these elements can enable the table header and footer to be printed at the top and bottom of each page.

The <thead> tag must be used in the following context: As a child of a <table> element, after any <caption>, and <colgroup> elements, and before any <tbody>, <tfoot>, and <tr> elements.

Example:

Month	Savings
January	$100
February	$80
Sum	$180

```
<!DOCTYPE html>
<html>
<head>
<title>Table Example</title>
</head>
<body>
<table border = "1" cellpadding = "3" cellspacing = "0">
 <colgroup>
  <col style="background-color:green">
```

```
        <col style="background-color:red">
  </colgroup>
  <thead>
  <tr>
    <th>Month</th>
    <th>Savings</th>
  </tr>
 </thead>
 <tfoot>
  <tr>
    <td>Sum</td>
    <td>$180</td>
  </tr>
 </tfoot>
 <tbody>
  <tr>
    <td>January</td>
    <td>$100</td>
  </tr>
  <tr>
    <td>February</td>
    <td>$80</td>
  </tr>
 </tbody>
</table>
</body>
</html>
```

10. Accessible Table:

Data tables are used to organize data with a logical relationship in grids. Accessible tables need HTML markup that indicates header cells and data cells, and defines their relationship. Assistive technologies use this information to provide context to users.

To make tables accessible, header cells must be marked up with <th>, and data cells with <td>. For more complex tables, explicit associations may be needed using scope, id, and headers attributes.

Using id and headers attributes to associate data cells with header cells in data tables

The objective of this technique is to associate each data cell (in a data table) with the appropriate headers. This technique adds a headers attribute to each data cell (td element). It also adds an **id** attribute to any cell used as a header for other cells. The headers attribute of a cell contains a list of the id attributes of the associated header cells. If there is more than one id, they are separated by **spaces**.

This technique is used when data cells are associated with more than one row and/or one column header. This allows screen readers to speak the headers associated with each data cell when the relationships are too complex to be identified using the **th** element alone or

the th element with the scope attribute. Using this technique also makes these complex relationships perceivable when the presentation format changes.

This technique is not recommended for layout tables since its use implies a relationship between cells that is not meaningful when tables are used for layout.

Example:

Homework	Exams			Projects		
	1	2	Final	1	2	Final
15%	15%	15%	20%	10%	10%	15%

```
<!DOCTYPE html>
<html>
<head>
<title>Table Example</title>
<style>
th{
        background-color:lime;
}
td{
        background-color:cyan;
}
</style>
</head>
<body>
<table border = "1" cellspacing = "0">
<table>
  <tr>
   <th rowspan="2" id="h">Homework</th>
   <th colspan="3" id="e">Exams</th>
   <th colspan="3" id="p">Projects</th>
  </tr>
  <tr>
   <th id="e1" headers="e">1</th>
   <th id="e2" headers="e">2</th>
   <th id="ef" headers="e">Final</th>
   <th id="p1" headers="p">1</th>
   <th id="p2" headers="p">2</th>
   <th id="pf" headers="p">Final</th>
  </tr>
  <tr>
   <td headers="h">15%</td>
   <td headers="e e1">15%</td>
   <td headers="e e2">15%</td>
   <td headers="e ef">20%</td>
   <td headers="p p1">10%</td>
   <td headers="p p2">10%</td>
   <td headers="p pf">15%</td>
  </tr>
 </table>
</body>
```

```
</html>
```

Using the scope attribute to associate header cells and data cells in data tables

The objective of this technique is to associate header cells with data cells in complex tables using the scope attribute. The scope attribute may be used to clarify the scope of any cell used as a header. The scope identifies whether the cell is a header for a row, column, or group of rows or columns. The values row, col, rowgroup, and colgroup identify these possible scopes respectively.

For simple data tables where the header is not in the first row or column, like the one in Example 1, this technique can be used. Based on screen reader support today, its use is suggested in two situations both relating to simple tables:

- data cells marked up with td that also function as row header or column header
- header cells marked up with td instead of th. Sometimes, authors use this to avoid the display characteristics associated with th and also do not choose to use CSS to control the display for th.

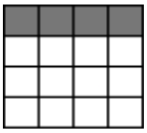
Example:

Contact Information

	Name	Phone	Fax	City
1.	Joel Garner	412-212-5421	412-212-5400	Pittsburgh
2.	Clive Lloyd	410-306-1420	410-306-5400	Baltimore
3.	Gordon Greenidge	281-564-6720	281-511-6600	Houston

```
<!DOCTYPE html>
<html>
<head>
<title>Table Example</title>
<style>
th{
        background-color:lime;
}
td{
        background-color:cyan;
}
</style>
</head>
<body>
<table border = "1" cellspacing = "0">
<caption>Contact Information</caption>
 <tr>
  <td></td>
  <th scope="col">Name</th>
  <th scope="col">Phone</th>
  <th scope="col">Fax</th>
  <th scope="col">City</th>
 </tr><tr>
  <td>1.</td>
  <th scope="row">Joel Garner</th>
  <td>412-212-5421</td>
  <td>412-212-5400</td>
  <td>Pittsburgh</td>
```

```
 </tr><tr>
  <td>2.</td>
  <th scope="row">Clive Lloyd</th>
  <td>410-306-1420</td>
  <td>410-306-5400</td>
  <td>Baltimore</td>
 </tr><tr>
  <td>3.</td>
  <th scope="row">Gordon Greenidge</th>
  <td>281-564-6720</td>
  <td>281-511-6600</td>
  <td>Houston</td>
 </tr>
</table>
</body>
</html>
```



Tables with one header for rows or columns:

For tables with content that is easy to distinguish, mark up header cells with <th> and data cells with <td> elements.

Example 1: Table with header cells in the top row only

```
<!DOCTYPE html>
<html>
<head>
<title>Table Example</title>
<style>
.head{
        background-color:wheat;
}
</style>
</head>
<body>
<table border = "1" cellpadding = "3" cellspacing = "0">
<caption>
<p>Phone Detail</p>
</caption>
 <colgroup>
  <col span="2" style="background-color:cyan">
   <col style="background-color:yellow">
        <col style="background-color:red">
 </colgroup>
 <tr class = "head">
  <th>ISBN</th>
```

Phone Detail

ISBN	Title	Quantity	Price
3476896	Motorola	15	$53
3476897	Samsung	20	$55

```
  <th>Title</th>
        <th>Quantity</th>
  <th>Price</th>
 </tr>
 <tr>
  <td>3476896</td>
  <td>Motorola</td>
        <td>15</td>
  <td>$53</td>
 </tr>
  <tr>
  <td>3476897</td>
  <td>Samsung</td>
        <td>20</td>
  <td>$55</td>
 </tr>
</table>
</body>
</html>
```

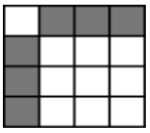
Example 2: Table with header cells in the first column only

```
<!DOCTYPE html>
<html>
<head>
<title>Table Example</title>
</head>
<body>
<table border = "1" cellpadding = "3" cellspacing = "1">
<caption>
<p>Phone Detail</p>
</caption>
 <colgroup>
  <col style="background-color:cyan">
  <col span = "2" style="background-color:lime">
 </colgroup>
 <tr>
  <th>ISBN</th>
        <td>3476896</td>
        <td>3476897</td>
</tr>
<tr>
  <th>Title</th>
        <td>Motorola</td>
        <td>Samsung</td>
</tr>
<tr>
```

Phone Detail

ISBN	3476896	3476897
Title	Motorola	Samsung
Quantity	15	20
Price	$53	$55

```
        <th>Quantity</th>
        <td>15</td>
        <td>20</td>
</tr>
 <tr>
  <th>Price</th>
  <td>$53</td>
        <td>$55</td>
 </tr>
</table>
</body>
</html>
```



Tables with two headers have a simple row header and a simple column header:

For tables with unclear header directions, define the direction of each header by setting the scope attribute to col or row.

Example 1: Table with header cells in the top row and first column

```
<!DOCTYPE html>
<html>
<head>
<title>Table Example</title>
<style>
.head{
        background-color:red;
}
</style>
</head>
<body>
<table border = "1" cellpadding = "3" cellspacing = "0">
<colgroup>
<col span = "6" style = "background-color:cyan"
</colgroup>
<caption>
<p><mark>Time Table</mark></p>
</caption>
<tr class = "head">
                <th>Days/Time</th>
                <th>Monday</th>
                <th>Tuesday</th>
                <th>Wednesday</th>
                <th>Thursday</th>
                <th>Friday</th>
</tr>
        <tr>
```

Time Table

Days/Time	Monday	Tuesday	Wednesday	Thursday	Friday
09:00 - 11:00	Closed	Open	Open	Closed	Closed
11:00 - 13:00	Open	Open	Closed	Closed	Closed
13:00 - 15:00	Open	Open	Open	Closed	Closed
15:00 - 17:00	Closed	Closed	Closed	Open	Open

```
                <th class = "head">09:00 - 11:00</th>
                <td>Closed</td>
                <td>Open</td>
                <td>Open</td>
                <td>Closed</td>
                <td>Closed</td>
        </tr>
        <tr>
                <th class = "head">11:00 - 13:00</th>
                <td>Open</td>
                <td>Open</td>
                <td>Closed</td>
                <td>Closed</td>
                <td>Closed</td>
        </tr>
        <tr>
                <th class = "head">13:00 - 15:00</th>
                <td>Open</td>
                <td>Open</td>
                <td>Open</td>
                <td>Closed</td>
                <td>Closed</td>
        </tr>
        <tr>
                <th class = "head">15:00 - 17:00</th>
                <td>Closed</td>
                <td>Closed</td>
                <td>Closed</td>
                <td>Open</td>
                <td>Open</td>
        </tr>
</table>
</body>
</html>
```

Example 2: Table with an offset column of header cells

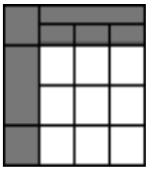
```
<!DOCTYPE html>
<html>
<head>
<title>Table Example</title>
<style>
th{
        background-color:yellow;
}
td{
        background-color:cyan;
```

Holidays taken in the last six months

ID	Name	July	August	September	October	November	December
215	Abel	5	2	0	0	0	3
231	Annette	0	5	3	0	0	6
173	Bernard	2	0	0	5	0	0
141	Gerald	0	10	0	0	0	8
99	Michael	8	8	8	8	0	4

```
}
</style>
</head>
<body>
<table border = "1" cellspacing = "0">
 <caption>
  Holidays taken in the last six months
 </caption >
 <thead>
  <tr>
   <th scope="col"><abbr title="Identification Number">ID</abbr></th>
   <th scope="col">Name</th>
   <th scope="col">July</th>
   <th scope="col">August</th>
   <th scope="col">September</th>
   <th scope="col">October</th>
   <th scope="col">November</th>
   <th scope="col">December</th>
  </tr>
 </thead>

 <tbody>
  <tr>
   <td>215</td>
   <th scope="row">Abel</th>
   <td>5</td>
   <td>2</td>
   <td>0</td>
   <td>0</td>
   <td>0</td>
   <td>3</td>
  </tr>

  <tr>
   <td>231</td>
   <th scope="row">Annette </th>
   <td>0</td>
   <td>5</td>
   <td>3</td>
   <td>0</td>
   <td>0</td>
   <td>6</td>
  </tr>

  <tr>
   <td>173</td>
```

```
    <th scope="row">Bernard</th>
    <td>2</td>
    <td>0</td>
    <td>0</td>
    <td>5</td>
    <td>0</td>
    <td>0</td>
   </tr>

   <tr>
    <td>141</td>
    <th scope="row">Gerald</th>
    <td>0</td>
    <td>10</td>
    <td>0</td>
    <td>0</td>
    <td>0</td>
    <td>8</td>
   </tr>

   <tr>
    <td>99</td>
    <th scope="row">Michael</th>
    <td>8</td>
    <td>8</td>
    <td>8</td>
    <td>8</td>
    <td>0</td>
    <td>4</td>
   </tr>
  </tbody>
 </table>
 </body>
 </html>
```



Tables with irregular headers have header cells that span multiple columns and/or rows:

For these tables, define column and row groups and set the range of the header cells using the colgroup and rowgroup values of the scope attribute.

Example 1: Table with two tier headers

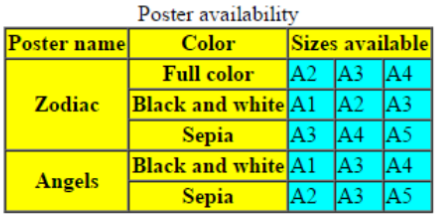
```
<!DOCTYPE html>
<html>
<head>
<title>Table Example</title>
```

Inventory

	Mars		Venus	
	Produced	Sold	Produced	Sold
Teddy Bears	50,000	30,000	100,000	80,000
Board Games	10,000	5,000	12,000	9,000

```
<style>
th{
        background-color:yellow;
}
td{
        background-color:cyan;
}
</style>
</head>
<body>
<table border = "1" cellspacing = "0">
 <caption>  Inventory  </caption >
 <colgroup span="2"></colgroup>
 <colgroup span="2"></colgroup>
 <tr>
  <td rowspan="2"></td>
  <th colspan="2" scope="colgroup">Mars</th>
  <th colspan="2" scope="colgroup">Venus</th>
 </tr>
 <tr>
  <th scope="col">Produced</th>
  <th scope="col">Sold</th>
  <th scope="col">Produced</th>
  <th scope="col">Sold</th>
 </tr>
 <tr>
  <th scope="row">Teddy Bears</th>
  <td>50,000</td>
  <td>30,000</td>
  <td>100,000</td>
  <td>80,000</td>
 </tr>
 <tr>
  <th scope="row">Board Games</th>
  <td>10,000</td>
  <td>5,000</td>
  <td>12,000</td>
  <td>9,000</td>
 </tr>
</table>
</body>
</html>
```

Example 2: Table with headers spanning multiple rows or columns




```
<!DOCTYPE html>
<html>
<head>
<title>Table Example</title>
<style>
th{
        background-color:yellow;
}
td{
        background-color:cyan;
}
</style>
</head>
<body>
<table border = "1" cellspacing = "0">
 <caption>
  Poster availability
 </caption>
 <col>
 <col>
 <colgroup span="3"></colgroup>
 <thead>
  <tr>
   <th scope="col">Poster name</th>
   <th scope="col">Color</th>
   <th colspan="3" scope="colgroup">Sizes available</th>
  </tr>
 </thead>
 <tbody>
  <tr>
   <th rowspan="3" scope="rowgroup">Zodiac</th>
   <th scope="row">Full color</th>
   <td>A2</td>
   <td>A3</td>
   <td>A4</td>
  </tr>
  <tr>
   <th scope="row">Black and white</th>
   <td>A1</td>
   <td>A2</td>
   <td>A3</td>
  </tr>
  <tr>
   <th scope="row">Sepia</th>
   <td>A3</td>
```

```
    <td>A4</td>
    <td>A5</td>
   </tr>
  </tbody>
  <tbody>
   <tr>
    <th rowspan="2" scope="rowgroup">Angels</th>
    <th scope="row">Black and white</th>
    <td>A1</td>
    <td>A3</td>
    <td>A4</td>
   </tr>
   <tr>
    <th scope="row">Sepia</th>
    <td>A2</td>
    <td>A3</td>
    <td>A5</td>
   </tr>
  </tbody>
</table>
</body>
</html>
```

Tables with multi-level headers have multiple header cells associated per data cell: For tables that are so complex that header cells can't be associated in a strictly horizontal or vertical way, use id and headers attributes to explicitly associate header and data cells.

Example 1: Table with multiple column headers in each column

```
<!DOCTYPE html>
<html>
<head>
<title>Table Example</title>
<style>
th{
        background-color:yellow;
}
td{
        background-color:cyan;
}
</style>
</head>
<body>
<table border = "1" cellspacing = "0">
 <caption>
   Supplier contacts
```

Supplier contacts

	Example 1 Ltd	Example 2 Co
Contact	James Phillips	Marie Beauchamp
Position	Sales Director	Sales Manager
Email	jp@1ltd.example.com	marie@2co.example.com
	Example 3 Ltd	Example 4 Inc
Contact	Suzette Jones	Alex Howe
Position	Sales Officer	Sales Director
Email	Suz@ltd3.example.com	howe@4inc.example.com

```
 </caption>
 <tr>
  <td id="blank">&amp;nbsp;</td>
  <th id="co1" headers="blank">Example 1 Ltd</th>
  <th id="co2" headers="blank">Example 2 Co</th>
 </tr>
 <tr>
  <th id="c1" headers="blank">Contact</th>
  <td headers="co1 c1">James Phillips</td>
  <td headers="co2 c1">Marie Beauchamp</td>
 </tr>
 <tr>
  <th id="p1"  headers="blank">Position</th>
  <td headers="co1 p1">Sales Director</td>
  <td headers="co2 p1">Sales Manager</td>
 </tr>
 <tr>
  <th id="e1"  headers="blank">Email</th>
  <td headers="co1 e1">jp@1ltd.example.com</td>
  <td headers="co2 e1">marie@2co.example.com</td>
 </tr>
 <tr>
  <td>&amp;nbsp;</td>
  <th id="co3" headers="blank">Example 3 Ltd</th>
  <th id="co4" headers="blank">Example 4 Inc</th>
 </tr>
 <tr>
  <th id="c2"  headers="blank">Contact</th>
  <td headers="co3 c2">Suzette Jones</td>
  <td headers="co4 c2">Alex Howe</td>
 </tr>
 <tr>
  <th id="p2" headers="blank">Position</th>
  <td headers="co3 p2">Sales Officer</td>
  <td headers="co4 p2">Sales Director</td>
 </tr>
 <tr>
  <th id="e2" headers="blank">Email</th>
  <td headers="co3 e2">Suz@ltd3.example.com</td>
  <td headers="co4 e2">howe@4inc.example.com</td>
 </tr>
</table>
</body>
</html>
```

Example 2: Table with three headers related to each data cell

```
<!DOCTYPE html>
<html>
<head>
<title>Table Example</title>
<style>
th{
        background-color:yellow;
}
td{
        background-color:cyan;
}
</style>
</head>
<body>
<table border = "1" cellspacing = "0">
<caption>
  Availability of holiday accommodation
</caption>
<thead>
  <tr>
    <td></td>
    <th id="stud" scope="col">
      Studio
    </th>
    <th id="apt" scope="col">
      <abbr title="Apartment">Apt</abbr>
    </th>
    <th id="chal" scope="col">
      Chalet
    </th>
    <th id="villa" scope="col">
      Villa
    </th>
  </tr>
</thead>
<tbody>
  <tr>
    <th id="par" class="span" colspan="5" scope="colgroup">
      Paris
    </th>
  </tr>
  <tr>
    <th headers="par" id="pbed1">
      1 bedroom
    </th>
```

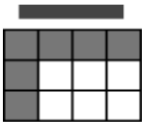
Availability of holiday accommodation

	Studio	Apt	Chalet	Villa
Paris				
1 bedroom	11	20	25	23
2 bedroom	-	43	52	32
3 bedroom	-	13	15	40
Rome				
1 bedroom	13	21	22	3
2 bedroom	-	23	43	30
3 bedroom	-	16	32	40

```
    <td headers="par pbed1 stud">
      11
    </td>
    <td headers="par pbed1 apt">
      20
    </td>
    <td headers="par pbed1 chal">
      25
    </td>
    <td headers="par pbed1 villa">
      23
    </td>
  </tr>
  <tr>
    <th headers="par" id="pbed2">
      2 bedroom
    </th>
    <td headers="par pbed2 stud">
      -
    </td>
    <td headers="par pbed2 apt">
      43
    </td>
    <td headers="par pbed2 chal">
      52
    </td>
    <td headers="par pbed2 villa">
      32
    </td>
  </tr>
  <tr>
    <th headers="par" id="pbed3">
      3 bedroom
    </th>
    <td headers="par pbed3 stud">
      -
    </td>
    <td headers="par pbed3 apt">
      13
    </td>
    <td headers="par pbed3 chal">
      15
    </td>
    <td headers="par pbed3 villa">
      40
    </td>
```

```
</tr>
<tr>
  <th id="rome" class="span" colspan="5" scope="colgroup">
    Rome
  </th>
</tr>
<tr>
  <th id="rbed1" headers="rome">
    1 bedroom
  </th>
  <td headers="rome rbed1 stud">
    13
  </td>
  <td headers="rome rbed1 apt">
    21
  </td>
  <td headers="rome rbed1 chal">
    22
  </td>
  <td headers="rome rbed1 villa">
    3
  </td>
</tr>
<tr>
  <th id="rbed2" headers="rome">
    2 bedroom
  </th>
  <td headers="rome rbed2 stud">
    -
  </td>
  <td headers="rome rbed2 apt">
    23
  </td>
  <td headers="rome rbed2 chal">
    43
  </td>
  <td headers="rome rbed2 villa">
    30
  </td>
</tr>
<tr>
  <th id="rbed3" headers="rome">
    3 bedroom
  </th>
  <td headers="rome rbed3 stud">
    -
```

```
      </td>
      <td headers="rome rbed3 apt">
        16
      </td>
      <td headers="rome rbed3 chal">
        32
      </td>
      <td headers="rome rbed3 villa">
        40
      </td>
    </tr>
  </tbody>
  </table>
  </body>
  </html>
```



Caption & Summary: A caption identifies the overall topic of a table and is useful in most situations. A summary provides orientation or navigation hints in complex tables.

Some document formats other than HTML, such as PDF, provide similar mechanisms to markup table structures. Word processing applications may also provide mechanisms to markup tables. Table's markup is often lost when converting from one format to another, though some programs may provide functionality to assist converting table markup.

Many web authoring tools and content management systems (CMS) provide functions to define header cells during table creation without having to manually edit the code.

- **Why is this important?**

Tables without structural markup to differentiate and properly link between header and data cells, create accessibility barriers. Relying on visual cues alone is not sufficient to create an accessible table. With structural markup, headers and data cells can be programmatically determined by software, which means that:

- **People using screen readers** can have the row and column headers read aloud as they navigate through the table. Screen readers speak one cell at a time and reference the associated header cells, so the reader doesn't lose context.
- **Some people use alternative ways to render the data**, for example by using custom stylesheets to display header cells more prominently. Techniques like this enable them to change text size and colors, and display the information as lists rather than grids. In order to allow alternative renderings, table code needs to be properly structured.

FORM:

1. Introduction Form:

HTML Forms are required when you want to collect some data from the site visitor. For example during user registration you would like to collect information such as name, email address, credit card, etc.

A form will take input from the site visitor and then will post it to a back-end application such as CGI, ASP Script or PHP script etc. The back-end application will perform required processing on the passed data based on defined business logic inside the application.

There are various form elements available like text fields, textarea fields, drop-down menus, radio buttons, checkboxes, etc.

The HTML **<form>** tag is used to create an HTML form and it has following syntax:

```
<form action="Script URL" method="GET|POST">
   form elements like input, textarea etc.
</form>
```

2. Form Attributes:

Apart from common attributes, following is a list of the most frequently used form attributes:

Attribute	Description
action	Backend script ready to process our passed data.
method	Method to be used to upload data. The most frequently used are GET and POST methods.
target	Specify the target window or frame where the result of the script will be displayed. It takes values like _blank, _self, _parent etc.
enctype	We can use the enctype attribute to specify how the browser encodes the data before it sends it to the server. Possible values are: ➢ **application/x-www-form-urlencoded** - This is the standard method most forms use in simple scenarios. ➢ **mutlipart/form-data** - This is used when you want to upload binary data in the form of files like image, word file etc.

3. HTML Form Controls:

There are different types of form controls that we can use to collect data using HTML form:

- **Text Input Controls:**

There are three types of text input used on forms:

a. Single-Line Text Input Controls:

This control is used for items that require only one line of user input, such as search boxes or names. They are created using HTML <input> tag.

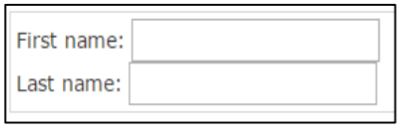
Attributes:

Following is the list of attributes for <input> tag for creating text field.

Attribute	Description
type	Indicates the type of input control and for text input control it will be set to **text**.
name	Used to give a name to the control which is sent to the server to be recognized and get the value.
value	This can be used to provide an initial value inside the control.
size	Allows to specify the width of the text-input control in terms of characters.
maxlength	Allows to specify the maximum number of characters a user can enter into the text box.

Example:

```
<!DOCTYPE html>
<html>
<head>
<title>Text Input Control</title>
</head>
<body>
<form >
First name:  <input type="text" name="first_name" />
<br>
Last name:  <input type="text" name="last_name" />
</form>
</body>
</html>
```



b. Password Input Controls:

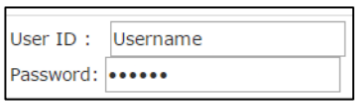
This is also a single-line text input but it masks the character as soon as a user enters it. They are also created using HTML <input> tag but type attribute is set to **password**.

Attributes

Following is the list of attributes for <input> tag for creating password field.

Attribute	Description
type	Indicates the type of input control and for password input control it will be set to **password**.
name	Used to give a name to the control which is sent to the server to be recognized and get the value.
value	This can be used to provide an initial value inside the control.
size	Allows to specify the width of the text-input control in terms of characters.
maxlength	Allows to specify the maximum number of characters a user can enter into the text box.

Example:




```
<!DOCTYPE html>
<html>
<head>
<title>Password Input Control</title>
</head>
<body>
<form >
User ID :  <input type="text" name="user_id" />
<br>
Password:  <input type="password" name="password" />
</form>
</body>
</html>
```

c. Multiple-Line Text Input Controls:

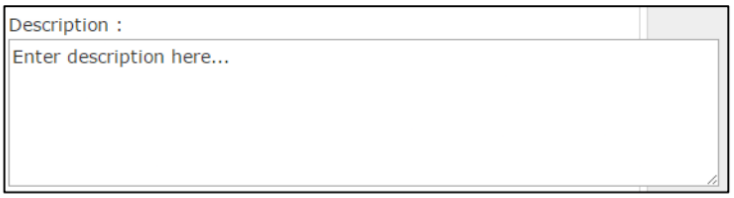
This is used when the user is required to give details that may be longer than a single sentence. Multi-line input controls are created using HTML <textarea> tag.

Attributes

Following is the list of attributes for <textarea> tag.

Attribute	Description
name	Used to give a name to the control which is sent to the server to be recognized and get the value.
rows	Indicates the number of rows of text area box.
cols	Indicates the number of columns of text area box

Example:




```
<!DOCTYPE html>
<html>
<head>
<title>Multiple-Line Input Control</title>
</head>
<body>
<form>
Description : <br />
<textarea rows="5" cols="50" name="description">
Enter description here...
</textarea>
</form>
</body>
</html>
```

Checkbox Control:

Checkboxes are used when more than one option is required to be selected. They are also created using HTML <input> tag but type attribute is set to **checkbox**.

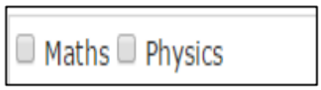
Attributes

Following is the list of attributes for <checkbox> tag.

Attribute	Description
type	Indicates the type of input control and for checkbox input control it will be set to **checkbox**.
name	Used to give a name to the control which is sent to the server to be recognized and get the value.
value	The value that will be used if the checkbox is selected.
checked	Set to *checked* if you want to select it by default.
Id	provide a unique identifier for the form so it can be more easily styled using CSS

Example:

```
<!DOCTYPE html>
<html>
<head>
<title>Checkbox Control</title>
</head>
<body>
<form>
<input type="checkbox" name="maths" value="on"> Maths
<input type="checkbox" name="physics" value="on"> Physics
</form>
</body>
</html>
```



Radio Button Control:

Radio buttons are used when out of many options, just one option is required to be selected. They are also created using HTML <input> tag but type attribute is set to **radio**.

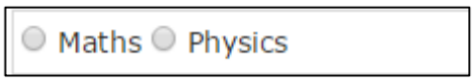
Attributes

Following is the list of attributes for radio button.

Attribute	Description
type	Indicates the type of input control and for checkbox input control it will be set to **radio**.
name	Used to give a name to the control which is sent to the server to be recognized and get the value.
value	The value that will be used if the radio box is selected.
checked	Set to *checked* if you want to select it by default.

id	provide a unique identifier for the form so it can be more easily styled using CSS

Example:




```
<!DOCTYPE html>
<html>
<head>
<title>Radio Box Control</title>
</head>
<body>
<form>
<input type="radio" name="subject" value="maths"> Maths
<input type="radio" name="subject" value="physics"> Physics
</form>
</body>
</html>
```

- **Select Box Control:**

A select box, also called drop down box which provides option to list down various options in the form of drop down list, from where a user can select one or more options.

Attributes

Following is the list of important attributes of <select> tag:

Attribute	Description
name	Used to give a name to the control which is sent to the server to be recognized and get the value.
size	This can be used to present a scrolling list box.
multiple	If set to "multiple" then allows a user to select multiple items from the menu.

Following is the list of important attributes of <option> tag:

Attribute	Description
value	The value that will be used if an option in the select box box is selected.
selected	Specifies that this option should be the initially selected value when the page loads.
label	An alternative way of labeling options

Example:


```
<!DOCTYPE html>
<html>
<head>
<title>Select Box Control</title>
</head>
<body>
<form>
```

```
<select name="dropdown">
<option value="Maths" selected>Maths</option>
<option value="Physics">Physics</option>
</select>
</form>
</body>
</html>
```

- **File Upload Box:**

If we want to upload a file to our web site, we will need to use a file upload box, also known as a file select box. This is also created using the <input> element but type attribute is set to **file**.

Attributes

Following is the list of important attributes of file upload box:

Attribute	Description
name	Used to give a name to the control which is sent to the server to be recognized and get the value.
accept	Specifies the types of files that the server accepts.

Example:

```
<!DOCTYPE html>
<html>
<head>
<title>File Upload Box</title>
</head>
<body>
<form>
<input type="file" name="fileupload" accept="image/*" />
</form>
</body>
</html>
```

- **Button Controls:**

There are various ways in HTML to create clickable buttons. We can also create a clickable button using <input> tag by setting its type attribute to **button**. The type attribute can take the following values:

Type	Description
submit	This creates a button that automatically submits a form.
reset	This creates a button that automatically resets form controls to their initial values.
button	This creates a button that is used to trigger a client-side script when the user clicks that button.
image	This creates a clickable button but we can use an image as background of the button.

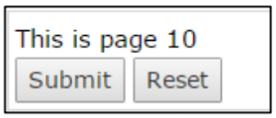
Example:


```
<!DOCTYPE html>
<html>
<head>
<title>File Upload Box</title>
</head>
<body>
<form>
<input type="submit" name="submit" value="Submit" />
<input type="reset" name="reset"  value="Reset" />
<input type="button" name="ok" value="OK"  />
<input type="image" name="imagebutton" src="/html/images/logo.png" />
</form>
</body>
</html>
```

- **Hidden Form Controls:**

Hidden form controls are used to hide data inside the page which later on can be pushed to the server. This control hides inside the code and does not appear on the actual page. For example, following hidden form is being used to keep current page number. When a user will click next page then the value of hidden control will be sent to the web server and there it will decide which page has be displayed next based on the passed current page.

Example:




```
<!DOCTYPE html>
<html>
<head>
<title>File Upload Box</title>
</head>
<body>
<form>
<p>This is page 10</p>
<input type="hidden" name="pagename" value="10" />
<input type="submit" name="submit" value="Submit" />
<input type="reset" name="reset"  value="Reset" />
</form>
</body>
</html>
```

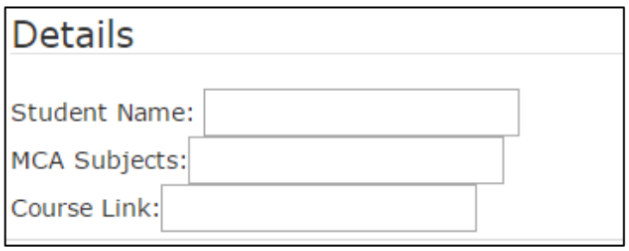
4. Fieldset and Legend:

The HTML <fieldset> tag is used for grouping related form elements. By using the fieldset tag and the legend tag, we can make our forms much easier to understand for our users.

The HTML <legend> tag s used to define a caption for <fieldset> tag.

Example:

```
<!DOCTYPE html>
<html>
<head>
<title>HTML fieldset Tag</title>
</head>
<body>
<form>
 <fieldset>
  <legend>Details</legend>
  Student Name: <input type="text"><br />
  MCA Subjects:<input type="text"><br />
  Course Link:<input type="url" name="websitelink">
 </fieldset>
</form>
</body>
</html>
```



- **Global Attributes:**

Not valid in base, head, html, meta, param, script, style, and title elements.

Attribute	HTML-5	Description
accesskey		Specifies a shortcut key for an element to be used in place of keyboard.
class		The class of the element
contenteditable	Yes	Specify whether the element is editable or not.
contextmenu	Yes	Specifies a context menu for an element.
data-*	Yes	Used to store custom data associated with the element.
draggable	Yes	Boolean attribute to specify whether the element can be dragged or not.
dropzone	Yes	Specifies whether the dragged data is copied, moved, or linked, when dropped.
hidden	Yes	Specifies whether element should be visible or not.
id		A unique id for the element
spellcheck	Yes	Specifies if the element must have it's spelling or grammar checked.
style		An inline style definition
tabindex		Specifies the tab order of an element.
title		A text to display in a tool tip
translate	Yes	Boolean attribute specifies whether the content of an element should be translated or not

Language Attributes:

The **lang** attribute indicates the language being used for the enclosed content. The language is identified using the ISO standard language abbreviations, such as **fr** for **French**, **en** for **English**, and so on. RFC 1766 (https://www.ietf.org/rfc/rfc1766.txt) describes these codes and their formats.

Not valid in base, br, frame, frameset, hr, iframe, param, and script elements.

Attribute	Value	Description
dir	ltr | rtl	Sets the text direction
lang	language_code	Sets the language code

Window Events Attributes:

Following events have been introduced in older versions of HTML but all the tags marked with  are part of HTML-5.

Events	HTML-5	Description
onafterprint		Triggers after a document is printed
onbeforeprint		Triggers before a document is printed
onbeforeonload		Triggers before a document loads
onerror		Triggers when an error occurs
onhaschange		Triggers when a document has changed
onload		Triggers when a document loads
onmessage		Triggers when a message is triggered
onoffline		Triggers when a document goes offline
ononline		Triggers when a document comes online
onpagehide		Triggers when a window is hidden
onpageshow		Triggers when a window becomes visible
onpopstate		Triggers when a window's history changes
onredo		Triggers when a document performs a redo
onresize		Triggers when a window is resized
onstorage		Triggers when a document loads
onundo		Triggers when a document performs an undo
onunload		Triggers when a user leaves the document

Form Events:

Following tags have been introduced in older versions of HTML but all the tags marked with  are part of HTML-5.

Events	HTML-5	Description
onblur		Triggers when a window loses focus
onchange		Triggers when an element changes
oncontextmenu		Triggers when a context menu is triggered
onfocus		Triggers when a window gets focus
onformchange		Triggers when a form changes
onforminput		Triggers when a form gets user input

oninput		Triggers when an element gets user input
oninvalid		Triggers when an element is invalid
onreset		Triggers when a form is reset
onselect		Triggers when an element is selected
onsubmit		Triggers when a form is submitted

- **Keyboard Events:**

Events	HTML-5	Description
onkeydown		Triggers when a key is pressed
onkeypress		Triggers when a key is pressed and released
onkeyup		Triggers when a key is released

- **Mouse Events:**

Following tags have been introduced in older versions of HTML but all the tags marked with  are part of HTML-5.

Events	HTML-5	Description
onclick		Triggers on a mouse click
ondblclick		Triggers on a mouse double-click
ondrag		Triggers when an element is dragged
ondragend		Triggers at the end of a drag operation
ondragenter		Triggers when an element has been dragged to a valid drop target
ondragleave		Triggers when an element leaves a valid drop target
ondragover		Triggers when an element is being dragged over a valid drop target
ondragstart		Triggers at the start of a drag operation
ondrop		Triggers when a dragged element is being dropped
onmousedown		Triggers when a mouse button is pressed
onmousemove		Triggers when the mouse pointer moves
onmouseout		Triggers when the mouse pointer moves out of an element
onmouseover		Triggers when the mouse pointer moves over an element
onmouseup		Triggers when a mouse button is released
onmousewheel		Triggers when the mouse wheel is being rotated
onscroll		Triggers when an element's scrollbar is being scrolled

- **Media Events:**

Following tags have been introduced in older versions of HTML but all the tags marked with  are part of HTML-5.

Events	HTML-5	Description
onabort		Triggers on an abort event
oncanplay		Triggers when a media can start play, but might has to stop for buffering

oncanplaythrough		Triggers when a media can be played to the end, without stopping for buffering
ondurationchange		Triggers when the length of a media is changed
onemptied		Triggers when a media resource element suddenly becomes empty.
onended		Triggers when a media has reached the end
onerror		Triggers when an error occurs
onloadeddata		Triggers when media data is loaded
onloadedmetadata		Triggers when the duration and other media data of a media element is loaded
onloadstart		Triggers when the browser starts loading the media data
onpause		Triggers when media data is paused
onplay		Triggers when media data is going to start playing
onplaying		Triggers when media data has started playing
onprogress		Triggers when the browser is fetching the media data
onratechange		Triggers when the playing rate of media data has changed
onreadystatechange		Triggers when the ready-state changes
onseeked		Triggers when the seeking attribute of a media element is no longer true, and the seeking has ended
onseeking		Triggers when the seeking attribute of a media element is true, and the seeking has begun
onstalled		Triggers when there is an error in fetching media data
onsuspend		Triggers when the browser has been fetching media data, but stopped before the entire media file was fetched
ontimeupdate		Triggers when media changes its playing position
onvolumechange		Triggers when a media changes the volume, also when volume is set to "mute"
onwaiting		Triggers when media has stopped playing, but is expected to resume

5. Tab Index Attribute:

The tabindex attribute specifies the tab order of an element (when the “tab” button is used for navigating).

For example:

```
<a href = “http://www.w3school.com/” tabindex = “2”>W3School</a>
<a href = “http://www.google.com/” tabindex = “1”>Google</a>
<a href = “http://www.microsoft.com/” tabindex = “3”>Microsoft</a>
```

6. Sending Form Data to the Server:

There are two ways the browser client can send information to the web server.

1. The GET Method
2. The POST Method

Before the browser sends the information, it encodes it using a scheme called URL encoding. In this scheme, name/value pairs are joined with equal signs and different pairs are separated by the ampersand.

name1=value1&name2=value2&name3=value3

Spaces are removed and replaced with the + character and any other nonalphanumeric characters are replaced with a hexadecimal values. After the information is encoded it is sent to the server.

1. **The GET Method:**

The GET method sends the encoded user information appended to the page request. The page and the encoded information are separated by the **?** character.

http://www.test.com/index.htm?name1=value1&name2=value2

- The GET method produces a long string that appears in your server logs, in the browser's Location: box.
- The GET method is restricted to send upto 1024 characters only.
- Never use GET method if you have password or other sensitive information to be sent to the server.
- GET can't be used to send binary data, like images or word documents, to the server.
- The data sent by GET method can be accessed using QUERY_STRING environment variable.
- The PHP provides **$_GET** associative array to access all the sent information using GET method.

Example:

```
<?php
  if( $_GET["name"] || $_GET["age"] ) {
    echo "Welcome ". $_GET['name']. "<br />";
    echo "You are ". $_GET['age']. " years old.";

    exit();
  }
?>
<html>
  <body>
    <form action = "<?php $_PHP_SELF ?>" method = "GET">
      Name: <input type = "text" name = "name" />
      Age: <input type = "text" name = "age" />
      <input type = "submit" />
    </form>
  </body>
</html>
```

2. **The POST Method:**

The POST method transfers information via HTTP headers. The information is encoded as described in case of GET method and put into a header called QUERY_STRING.

- The POST method does not have any restriction on data size to be sent.
- The POST method can be used to send ASCII as well as binary data.
- The data sent by POST method goes through HTTP header so security depends on HTTP protocol. By using Secure HTTP you can make sure that your information is secure.
- The PHP provides **$_POST** associative array to access all the sent information using POST method.

Example:

```
<?php
  if( $_POST["name"] || $_POST["age"] ) {
    if (preg_match("/[^A-Za-z'-]/",$_POST['name'] )) {
      die ("invalid name and name should be alpha");
    }
    echo "Welcome ". $_POST['name']. "<br />";
    echo "You are ". $_POST['age']. " years old.";
    exit();
  }
?>
<html>
  <body>
    <form action = "<?php $_PHP_SELF ?>" method = "POST">
      Name: <input type = "text" name = "name" />
      Age: <input type = "text" name = "age" />
      <input type = "submit" />
    </form>
  </body>
</html>
```

FRAMES:

HTML frames are used to divide our browser window into multiple sections where each section can load a separate HTML document. A collection of frames in the browser window is known as a frameset. The window is divided into frames in a similar way the tables are organized: into rows and columns.

Disadvantages of Frames:

There are few drawbacks with using frames, so it's never recommended to use frames in your webpages:

- Some smaller devices cannot cope with frames often because their screen is not big enough to be divided up.
- Sometimes your page will be displayed differently on different computers due to different screen resolution.

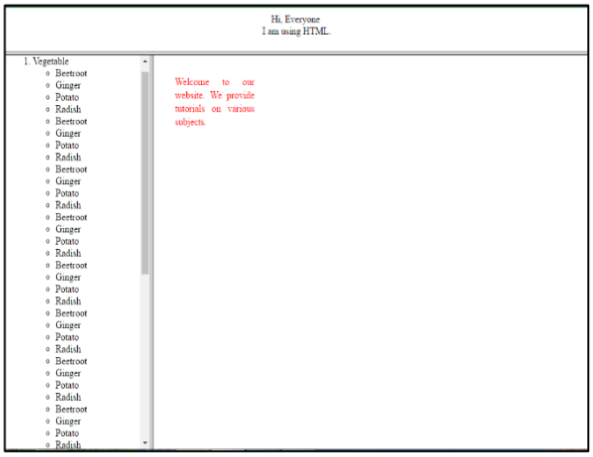
- The browser's *back button* might not work as the user hopes.
- There are still few browsers that do not support frame technology.

1. Creating Frames:

To use frames on a page we use <frameset> tag instead of <body> tag. The <frameset> tag defines how to divide the window into frames. The rows attribute of <frameset> tag defines horizontal frames and cols attribute defines vertical frames. Each frame is indicated by <frame> tag and it defines which HTML document shall open into the frame.

Example:

```
<!DOCTYPE html>
<html>
<head>
<title>HTML Frames</title>
</head>
<frameset rows="10%,*">
  <frame name="top" src="/HTML Programs/FirstProgram.html" />
  <frameset cols = "25%,*">
  <frame name="main" src="/HTML Programs/NestedList.html" />
  <frame name="bottom" src="/HTML Programs/Font.html" />
  </frameset>
  <noframes>
  <body>
    Your browser does not support frames.
  </body>
  </noframes>
</frameset>
</html>
```



2. The <frameset> Tag Attributes:

Following are important attributes of the <frameset> tag:

Attribute	Description
cols	Specifies how many columns are contained in the frameset and the size of each column. We can specify the width of each column in one of four ways: ➢ Absolute values in pixels. For example to create three vertical frames, use *cols="100, 500,100"*. ➢ A percentage of the browser window. For example to create three vertical frames, use *cols="10%, 80%, 10%"*. ➢ Using a wildcard symbol. For example to create three vertical frames, use *cols="10%, *, 10%"*. In this case wildcard takes remainder of the window.

	➢ As relative widths of the browser window. For example to create three vertical frames, use *cols="3*, 2*, 1*"*. This is an alternative to percentages. We can use relative widths of the browser window. Here the window is divided into sixths: the first column takes up half of the window, the second takes one third, and the third takes one sixth.
rows	This attribute works just like the cols attribute and takes the same values, but it is used to specify the rows in the frameset. For example to create two horizontal frames, use *rows="10%, 90%"*. We can specify the height of each row in the same way as explained above for columns.
border	This attribute specifies the width of the border of each frame in pixels. For example border="5". A value of zero means no border.
frameborder	This attribute specifies whether a three-dimensional border should be displayed between frames. This attribute takes value either 1 (yes) or 0 (no). For example frameborder="0" specifies no border.
framespacing	This attribute specifies the amount of space between frames in a frameset. This can take any integer value. For example framespacing="10" means there should be 10 pixels spacing between each frames.

3. The <frame> Tag Attributes:

Following are important attributes of <frame> tag:

Attribute	Description
src	This attribute is used to give the file name that should be loaded in the frame. Its value can be any URL. For example, src="/html/top_frame.htm" will load an HTML file available in html directory.
name	This attribute allows us to give a name to a frame. It is used to indicate which frame a document should be loaded into. This is especially important when we want to create links in one frame that load pages into an another frame, in which case the second frame needs a name to identify itself as the target of the link.
frameborder	This attribute specifies whether or not the borders of that frame are shown; it overrides the value given in the frameborder attribute on the <frameset> tag if one is given, and this can take values either 1 (yes) or 0 (no).
marginwidth	This attribute allows you to specify the width of the space between the left and right of the frame's borders and the frame's content. The value is given in pixels. For example marginwidth="10".
marginheight	This attribute allows you to specify the height of the space between the top and bottom of the frame's borders and its contents. The value is given in pixels. For example marginheight="10".
noresize	By default we can resize any frame by clicking and dragging on the borders of a frame. The noresize attribute prevents a user from being able to resize the frame. For example noresize="noresize".
scrolling	This attribute controls the appearance of the scrollbars that appear on the frame. This takes values either "yes", "no" or "auto". For example scrolling="no" means it should not have scroll bars.

longdesc	This attribute allows you to provide a link to another page containing a long description of the contents of the frame. For example longdesc="framedescription.htm"

4. Browser Support for Frames:

If a user is using any old browser or any browser which does not support frames then <noframes> element should be displayed to the user.

So we must place a <body> element inside the <noframes> element because the <frameset> element is supposed to replace the <body> element, but if a browser does not understand <frameset> element then it should understand what is inside the <body> element which is contained in a <noframes> element.

We can put some nice message for our user having old browsers. For example *Sorry!! your browser does not support frames.*

5. Frame's name and target attributes:

One of the most popular uses of frames is to place navigation bars in one frame and then load main pages into a separate frame.

Let's see following example where a test.htm file has following code:

```
<!DOCTYPE html>
<html>
<head>
<title>HTML Target Frames</title>
</head>
<frameset cols="200, *">
  <frame src="/html/menu.htm" name="menu_page" />
  <frame src="/html/main.htm" name="main_page" />
  <noframes>
  <body>
    Your browser does not support frames.
  </body>
  </noframes>
</frameset>
</html>
```

Here we have created two columns to fill with two frames. The first frame is 200 pixels wide and will contain the navigation menubar implemented by **menu.htm** file. The second column fills in remaining space and will contain the main part of the page and it is implemented by **main.htm** file. For all the three links available in menubar, we have mentioned target frame as **main_page**, so whenever we click any of the links in menubar, available link will open in main_page.

Following is the content of menu.htm file

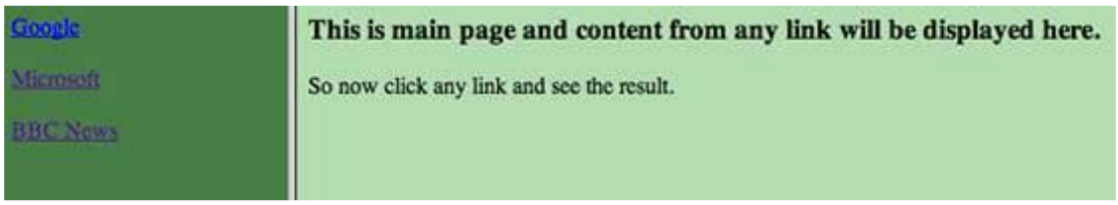
```
<!DOCTYPE html>
```

```
<html>
<body bgcolor="#4a7d49">
<a href="https://www.google.com" target="main_page">Google</a>
<br /><br />
<a href="https://www.microsoft.com" target="main_page">Microsoft</a>
<br /><br />
<a href="https://news.bbc.co.uk" target="main_page">BBC News</a>
</body>
</html>
```

Following is the content of main.htm file:

```
<!DOCTYPE html>
<html>
<body bgcolor="#b5dcb3">
<h3>This is main page and content from any link will be displayed here.</h3>
<p>So now click any link and see the result.</p>
</body>
</html>
```

When we load test.htm file, it produces following result:



Now we can try to click links available in the left panel and see the result. The *target* attribute can also take one of the following values:

Option	Description
_self	Loads the page into the current frame.
_blank	Loads a page into a new browser window.opening a new window.
_parent	Loads the page into the parent window, which in the case of a single frameset is the main browser window.
_top	Loads the page into the browser window, replacing any current frames.
targetframe	Loads the page into a named targetframe.

6. IFRAME:

We can define an inline frame with HTML tag **<iframe>**. The <iframe> tag is not somehow related to <frameset> tag, instead, it can appear anywhere in your document. The <iframe> tag defines a rectangular region within the document in which the browser can display a separate document, including scrollbars and borders.

The **src** attribute is used to specify the URL of the document that occupies the inline frame.

Example:


```
<!DOCTYPE html>
<html>
<head>
<title>HTML Iframes</title>
</head>
<body>
<p>Document content goes here...</p>
<iframe src="/html/menu.htm" width="555" height="200">
  Sorry your browser does not support inline frames.
</iframe>
<p>Document content also go here...</p>
</body>
</html>
```

7. The <Iframe> Tag Attributes:

Most of the attributes of the <iframe> tag, including *name, class, frameborder, id, longdesc, marginheight, marginwidth, name, scrolling, style, and title* behave exactly like the corresponding attributes for the <frame> tag.

Attribute	Description
src	This attribute is used to give the file name that should be loaded in the frame. Its value can be any URL. For example, src="/html/top_frame.htm" will load an HTML file avalaible in html directory.
name	This attribute allows you to give a name to a frame. It is used to indicate which frame a document should be loaded into. This is especially important when you want to create links in one frame that load pages into an another frame, in which case the second frame needs a name to identify itself as the target of the link.
frameborder	This attribute specifies whether or not the borders of that frame are shown; it overrides the value given in the frameborder attribute on the <frameset> tag if one is given, and this can take values either 1 (yes) or 0 (no).
marginwidth	This attribute allows you to specify the width of the space between the left and right of the frame's borders and the frame's content. The value is given in pixels. For example marginwidth="10".
marginheight	This attribute allows you to specify the height of the space between the top and bottom of the frame's borders and its contents. The value is given in pixels. For example marginheight="10".
noresize	By default you can resize any frame by clicking and dragging on the borders of a frame. The noresize attribute prevents a user from being able to resize the frame. For example noresize="noresize".
scrolling	This attribute controls the appearance of the scrollbars that appear on the frame. This takes values either "yes", "no" or "auto". For example scrolling="no" means it should not have scroll bars.

longdesc	This attribute allows you to provide a link to another page containing a long description of the contents of the frame. For example longdesc="framedescription.htm"

EXPLORING NEW ELEMENTS OF HTML5:

HTML5 is the latest and most enhanced version of HTML. Technically, HTML is not a programming language, but rather a markup language.

HTML5 is the next major revision of the HTML standard superseding HTML 4.01, XHTML 1.0, and XHTML 1.1. HTML5 is a standard for structuring and presenting content on the World Wide Web.

HTML5 is a cooperation between the World Wide Web Consortium (W3C) and the Web Hypertext Application Technology Working Group (WHATWG).

The new standard incorporates features like video playback and drag-and-drop that have been previously dependent on third-party browser plug-ins such as Adobe Flash, Microsoft Silverlight, and Google Gears.

1. Browser Support:

The latest versions of Apple Safari, Google Chrome, Mozilla Firefox, and Opera all support many HTML5 features and Internet Explorer 9.0 will also have support for some HTML5 functionality.

The mobile web browsers that come pre-installed on iPhones, iPads, and Android phones all have excellent support for HTML5.

2. New Features:

HTML5 introduces a number of new elements and attributes that helps in building a modern website. Following are great features introduced in HTML5.

- **New Semantic Elements**: These are like <header>, <footer>, and <section>.
- **Forms 2.0**: Improvements to HTML web forms where new attributes have been introduced for <input> tag.
- **Persistent Local Storage**: To achieve without resorting to third-party plugins.
- **WebSocket**: A next-generation bidirectional communication technology for web applications.
- **Server-Sent Events**: HTML5 introduces events which flow from web server to the web browsers and they are called Server-Sent Events (SSE).
- **Canvas**: This supports a two-dimensional drawing surface that you can program with JavaScript.
- **Audio & Video**: You can embed audio or video on your web pages without resorting to third-party plugins.
- **Geolocation**: Now visitors can choose to share their physical location with your web application.
- **Microdata**: This lets you create your own vocabularies beyond HTML5 and extend your web pages with custom semantics.

- **Drag and drop**: Drag and drop the items from one location to another location on a the same webpage.

3. Backward Compatibility:

HTML5 is designed, as much as possible, to be backward compatible with existing web browsers. New features build on existing features and allow us to provide fallback content for older browsers.

It is suggested to detect support for individual HTML5 features using a few lines of JavaScript.

4. HTML5 Document:

The following tags have been introduced for better structure –

- **Section**: This tag represents a generic document or application section. It can be used together with h1-h6 to indicate the document structure.
- **Article**: This tag represents an independent piece of content of a document, such as a blog entry or newspaper article.
- **Aside**: This tag represents a piece of content that is only slightly related to the rest of the page.
- **Header**: This tag represents the header of a section.
- **Footer:** This tag represents a footer for a section and can contain information about the author, copyright information, etc.
- **Nav**: This tag represents a section of the document intended for navigation.
- **Dialog**: This tag can be used to mark up a conversation.
- **Figure**: This tag can be used to associate a caption together with some embedded content, such as a graphic or video.

5. Deprecated Tags:

The following elements are not available in HTML5 anymore and their function is better handled by CSS:

Tags (Elements)	Description
<acronym>	Defines an acronym
<applet>	Defines an applet
<basefont>	Defines an base font for the page.
<big>	Defines big text
** **	Defines centered text
<dir>	Defines a directory list
<font>	Defines text font, size, and color
<frame>	Defines a frame
<frameset>	Defines a set of frames
<isindex>	Defines a single-line input field
<noframes>	Defines a noframe section
<s>	Defines strikethrough text
<strike>	Defines strikethrough text
<tt>	Defines teletype text
<u>	Defines underlined text

6. Deprecated Attributes:

HTML5 has none of the presentational attributes that were in HTML4 as their functions are better handled by CSS. Some attributes from HTML4 are no longer allowed in HTML5 at all and they have been removed completely.

Following is the table having removed attributed and their corresponding impacted tags (elements) ie. elements from which those attributes have been removed permanently

Removed Attributes	From the Elements
rev	link, a
charset	link and a
shape	a
coords	a
longdesc	img and iframe.
target	link
nohref	area
profile	head
version	html
name	img
scheme	meta
archive	object
classid	object
codebase	object
codetype	object
declare	object
standby	object
valuetype	param
type	param
axis	td and t
abbr	td and t
scope	td
align	caption, iframe, img, input, object, legend, table, hr, div, h1, h2, h3, h4, h5, h6, p, col, colgroup, tbody, td, tfoot, th, thead and tr.
alink	body
link	body
vlink	body
text	body
background	body
bgcolor	table, tr, td, th and body.
border	table and object.
cellpadding	table
cellspacing	table
char	col, colgroup, tbody, td, tfoot, th, thead and tr.
charoff	col, colgroup, tbody, td, tfoot, th, thead and tr.
clear	br
compact	dl, menu, ol and ul.
frame	table

compact	dl, menu, ol and ul.
frame	table
frameborder	iframe
hspace	img and object.
vspace	img and object.
marginheight	iframe
marginwidth	iframe
noshade	hr
nowrap	td and th
rules	table
scrolling	iframe
size	hr
type	li, ol and ul.
valign	col, colgroup, tbody, td, tfoot, th, thead and tr
width	hr, table, td, th, col, colgroup and pre.

7. HTML5 - New Tags (Elements):

The following tags (elements) have been introduced in HTML5.

Tags (Elements)	Description
<article>	Represents an independent piece of content of a document, such as a blog entry or newspaper article
<aside >	Represents a piece of content that is only slightly related to the rest of the page.
<audio>	Defines an audio file.
<canvas>	This is used for rendering dynamic bitmap graphics on the fly, such as graphs or games.
<command>	Represents a command the user can invoke.
<datalist>	Together with the a new list attribute for input can be used to make comboboxes
<details>	Represents additional information or controls which the user can obtain on demand
<embed>	Defines external interactive content or plugin.
<figure>	Represents a piece of self-contained flow content, typically referenced as a single unit from the main flow of the document.
<footer>	Represents a footer for a section and can contain information about the author, copyright information, et cetera.
<header>	Represents a group of introductory or navigational aids.
<hgroup>	Represents the header of a section.
<keygen>	Represents control for key pair generation.
<mark>	Represents a run of text in one document marked or highlighted for reference purposes, due to its relevance in another context.
<meter>	Represents a measurement, such as disk usage.
<nav>	Represents a section of the document intended for navigation.
<output>	Represents some type of output, such as from a calculation done through scripting.

<progress>	Represents a completion of a task, such as downloading or when performing a series of expensive operations.
<ruby>	Together with <rt> and <rp> allow for marking up ruby annotations.
<section>	Represents a generic document or application section
<time>	Represents a date and/or time.
<video>	Defines a video file.
<wbr>	Represents a line break opportunity.

8. New types for <input> tag:

The input element's type attribute now has the following new values.

Type	Description
color	Color selector, which could be represented by a wheel or swatch picker
date	Selector for calendar date
datetime-local	Date and time display, with no setting or indication for time zones
datetime	Full date and time display, including a time zone.
email	Input type should be an email.
month	Selector for a month within a given year
number	A field containing a numeric value only
range	Numeric selector within a range of values, typically visualized as a slider
search	Term to supply to a search engine. For example, the search bar atop a browser.
tel	Input type should be telephone number.
time	Time indicator and selector, with no time zone information
url	Input type should be URL type.
week	Selector for a week within a given year

Example:

```
<!DOCTYPE html>
<html>
<body bgcolor="#b5dcb3">
<form>
    Email: <input type = "email" name = "email" id = "01"/>
    Date: <input type = "date" name = "data" id = "001"/>
    <input type = "submit" name = "submit" value = "submit"/>
</form>
</body>
</html>
```

9. KEYGEN:

The <keygen> tag specifies a key pair generator field used for forms. When the for is submitted, the private key is stored locally, and the public key is sent to the server.

Example:

```
<!DOCTYPE html>
<html>
<body bgcolor="#b5dcb3">
<form action = "mailto:go@gmail.com" method = "get">
Username: <input type = "text" name = "username"/><br/>
Password: <input type = "password" name = "password"/><br/>
Encryption:<keygen name = "security"><br/>
<input type = "submit" name = "submit" value = "submit"/>
</form>
</body>
</html>
```

10. Progress:

The progress element represents the completion progress of a task. A progress must have both a start tag and an end tag, even though it looks like a replaced element like an input. This is good though, as it helps with fallback content as we will cover later.

Apart from the global attributes, it can have two more attributes:

1. **Max:**

Indicates how much task needs to be done before it can be considered as complete. If not specified the default value is 1.0.

2. **Value:**

Indicates the current status of the progress bar. It must be greater than or equal to 0.0 and less than or equal to 1.0 or the value of the max attribute (if present)

Example:

```
<!DOCTYPE html>
<html>
<body bgcolor="#b5dcb3">
<form>
        <progress max = “100” value = “80”></progress>
</form>
</body>
</html>
```

11. Meter:

The html <meter> element represents either a scalar value within a known range or a fractional value. The <meter> tag should not be used to indicate progress (as in progress bar). For progress bars, we use the <progress> tag.

Example:

```
<!DOCTYPE html>
```

```
<html>
<body bgcolor="#b5dcb3">
<form>
        <meter value = “5” min = “0” max = “10”> 2 out of 10 </meter>
</form>
</body>
</html>
```

12. Command:

The command element represents a command which the user can invoke. The type attribute specifies the type of command.

Syntax: <command type = “command or checkbox or radio”>

Attribute Values

Value	Description
command	Default. Specifies a normal command with an action
checkbox	Specifies a command that can be toggled using a checkbox
radio	Specifies a command that can be toggled using a radio button

Example:

```
<!DOCTYPE html>
<html>
<body bgcolor="#b5dcb3">
<form>
<menu>
        <command type="command" label="Save" onclick="save()">Save</command>
</menu>
</form>
</body>
</html>
```

13. Menu:

The <menu> tag defines a list or menu of commands. The <menu> tag is used for context menus, toolbars and for listing form controls and commands.

Example:

```
<!DOCTYPE html>
<html>
<body bgcolor="#b5dcb3">
<form>
<menu>
        <command type="command" label="Save" onclick="save()">Save</command>
</menu>
```

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

14. Spellcheck:

The spellcheck attributes specifies whether the element is to have its spelling and grammar checked or not. The following can be spellchecked:

- Text values in input elements except passwords.
- Text in <textarea> element
- Text in editable elements

Example:

```
<!DOCTYPE html>
<html>
<body bgcolor="#b5dcb3">
<form>
<p contenteditable = "true" spellcheck = "true"> this is a paragraph. </p>
</form>
</body>
</html>
```

15. Header:

The <header> element represents a container for introductory content or a set of navigational links. A <header> element typically contains:

- One Or More Heading Elements (<H1> - <H6>)
- Logo Or Icon
- Authorship Information

We can have several <header> elements in one document.

Note: A <header> tag cannot be placed within a <footer>, <address> or another <header> element.

Example:

```
<!DOCTYPE html>
<html>
<body bgcolor="#b5dcb3">
<form>
<article>
 <header>
  <h1>Most important heading here</h1>
  <h3>Less important heading here</h3>
  <p>Some additional information here</p>
 </header>
 <p>Lorem Ipsum dolor set amet....</p>
```

```
</article>
</form>
</body>
</html>
```

16. Footer:

The <footer> tag defines a footer for a document or section. A <footer> element should contain information about its containing element. A <footer> element typically contains:

- Authorship Information
- Copyright Information
- Contact Information
- Sitemap
- Back To Top Links
- Related Documents

We can have several <footer> elements in one document.

Example:

```
<!DOCTYPE html>
<html>
<body bgcolor="#b5dcb3">
<form>
<article>
<footer>
 <p>Posted by: Hege Refsnes</p>
 <p>Contact information: <a href="mailto:someone@example.com">
 someone@example.com</a>.</p>
</footer>
</article>
</form>
</body>
</html>
```

CHAPTER – 3

SCRIPTING LANGUAGE

INTRODUCTION TO SCRIPTING LANGUAGE:

A scripting language is a programming language designed for integrating and communicating with other programming languages. Some of the most widely used scripting languages are JavaScript, VBScript, PHP, Perl, Python, Ruby, ASP and Tcl. Since a scripting language is normally used in conjunction with another programming language, they are often found alongside HTML, Java or C++.

One common distinction between a scripting language and a language used for writing entire applications is that, while a programming language is typically compiled first before being allowed to run, scripting languages are interpreted from source code or bytecode one command at a time.

Although scripts are widely employed in the programming world, they have recently become more associated with the World Wide Web, where they have been used extensively to create dynamic Web pages. While technically there are many client-side scripting languages that can be used on the Web, in practice it means using JavaScript.

DIFFERENTIATE BETWEEN CLIENT SIDE AND SERVER SIDE SCRIPTING LANGUAGE:

Client Side Scripting Language	Server Side Scripting Language
Client side scripting is used when the user's browser already has all the code and the page is altered on the basis of the users input.	Server side scripting is used to create dynamic pages based a number of conditions when the users browser makes a request to the server.
The Web Browser executes the client side scripting that resides at the user's computer.	The Web Server executes the server side scripting that produces the page to be sent to the browser.
The browser receives the page sent by the server and executes the client-side scripts.	Server executes server-side scripts to send out a page but it does not execute client-side scripts.
Client side scripting cannot be used to connect to the databases on the web server.	Server side scripting is used to connect to the databases that reside on the web server.
Client side scripting can't access the file system that resides at the web server.	Server side scripting can access the file system residing at the web server.
The files and settings that are local at the user's computer can be accessed using Client side scripting.	The settings that belong to Web server can be accessed using Server side scripting.
Client side scripting is possible to be blocked by the user.	Server side scripting can't be blocked by the user.
Response from a client-side script is faster as compared to a server-side script	Response from a server-side script is slower as compared to a client-side script

because the scripts are processed on the local computer.	because the scripts are processed on the remote computer.
Examples of Client side scripting languages : JavaScript, VB script, etc.	Examples of Server side scripting languages: PHP, JSP, ASP, ASP.Net, Ruby, Perl and many more.

JAVASCRIPT:

JavaScript is a lightweight, interpreted programming language. It is designed for creating network-centric applications. It is complimentary to and integrated with Java. JavaScript is very easy to implement because it is integrated with HTML. It is open and cross-platform.

JavaScript is a dynamic computer programming language. It is lightweight and most commonly used as a part of web pages, whose implementations allow client-side script to interact with the user and make dynamic pages. It is an interpreted programming language with object-oriented capabilities.

JavaScript was first known as **LiveScript,** but Netscape changed its name to JavaScript, possibly because of the excitement being generated by Java. JavaScript made its first appearance in Netscape 2.0 in 1995 with the name **LiveScript**. The general-purpose core of the language has been embedded in Netscape, Internet Explorer, and other web browsers.

FEATURES OF JAVASCRIPT:

- JavaScript is a lightweight, interpreted programming language.
- Designed for creating network-centric applications.
- Complementary to and integrated with Java.
- Complementary to and integrated with HTML.
- Open and cross-platform it works.

CLIENT-SIDE JAVASCRIPT:

Client-side JavaScript is the most common form of the language. The script should be included in or referenced by an HTML document for the code to be interpreted by the browser. It means that a web page need not be a static HTML, but can include programs that interact with the user, control the browser, and dynamically create HTML content.

The JavaScript client-side mechanism provides many advantages over traditional CGI server-side scripts. For example, we might use JavaScript to check if the user has entered a valid e-mail address in a form field.

The JavaScript code is executed when the user submits the form, and only if all the entries are valid, they would be submitted to the Web Server.

JavaScript can be used to trap user-initiated events such as button clicks, link navigation, and other actions that the user initiates explicitly or implicitly.

ADVANTAGES OF JAVASCRIPT:

The merits of using JavaScript are:

- **Less Server Interaction**:

We can validate user input before sending the page off to the server. This saves server traffic, which means less load on our server.

- **Immediate Feedback To The Visitors**:

They don't have to wait for a page reload to see if they have forgotten to enter something.

- **Increased interactivity**:

We can create interfaces that react when the user hovers over them with a mouse or activates them via the keyboard.

- **Richer interfaces**:

We can use JavaScript to include such items as drag-and-drop components and sliders to give a Rich Interface to your site visitors.

LIMITATIONS OF JAVASCRIPT:

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

- Client-side JavaScript does not allow the reading or writing of files. This has been kept for security reason.
- JavaScript cannot be used for networking applications because there is no such support available.
- JavaScript doesn't have any multithreading or multiprocessor capabilities.

JAVASCRIPT SYNTAX:

JavaScript can be implemented using JavaScript statements that are placed within the **<script>... </script>** HTML tags in a web page. We can place the **<script>** tags, containing our JavaScript, anywhere within our web page, but it is normally recommended that we should keep it within the **<head>** tags.

The <script> tag alerts the browser program to start interpreting all the text between these tags as a script. A simple syntax of our JavaScript will appear as follows.

```
<script ...>
    JavaScript code
</script>
```

The script tag takes two important attributes:

- **Language**:

This attribute specifies what scripting language we are using. Typically, its value will be JavaScript. Although recent versions of HTML (and XHTML, its successor) have phased out the use of this attribute.

- **Type**:

This attribute is what is now recommended to indicate the scripting language in use and its value should be set to "text/javascript".

So JavaScript segment will look like:

```
<script language="javascript" type="text/javascript">
  JavaScript code
</script>
```

Example:

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

1. Whitespace and Line Breaks:

JavaScript ignores spaces, tabs, and newlines that appear in JavaScript programs. We can use spaces, tabs, and newlines freely in our program and we are free to format and indent our programs in a neat and consistent way that makes the code easy to read and understand.

2. Semicolons are Optional:

Simple statements in JavaScript are generally followed by a semicolon character, just as they are in C, C++, and Java. JavaScript, however, allows us to omit this semicolon if each of our statements are placed on a separate line. For example, the following code could be written without semicolons.

```
<script language="JavaScript" type="text/JavaScript">
    var1 = 10
    var2 = 20
</script>
```

But when formatted in a single line as follows, we must use semicolons:

```
<script language="JavaScript" type="text/JavaScript">
    var1 = 10; var2 = 20;
</script>
```

Note: It is a good programming practice to use semicolons.

3. Case Sensitivity:

JavaScript is a case-sensitive language. This means that the language keywords, variables, function names, and any other identifiers must always be typed with a consistent capitalization of letters. So the identifiers **Time** and **TIME** will convey different meanings in JavaScript.

NOTE – Care should be taken while writing variable and function names in JavaScript.

4. Comments in JavaScript:

JavaScript supports both C-style and C++-style comments:

- Any text between a // and the end of a line is treated as a comment and is ignored by JavaScript.
- Any text between the characters /* and */ is treated as a comment. This may span multiple lines.
- JavaScript also recognizes the HTML comment opening sequence <!--. JavaScript treats this as a single-line comment, just as it does the // comment.
- The HTML comment closing sequence --> is not recognized by JavaScript so it should be written as //-->.

Example

The following example shows how to use comments in JavaScript.

```
<script language="javascript" type="text/javascript">
  <!--
    // This is a comment. It is similar to comments in C++
    /*
    * This is a multiline comment in JavaScript
    * It is very similar to comments in C Programming
    */
  //-->
</script>
```

Using JavaScript in HTML Document:

Example 1:

```
<!DOCTYPE html>
<html>
<body>
    <h1>My First JavaScript</h1>
    <button type="button" onclick="document.getElementById('demo').innerHTML = Date()">
    Click me to display Date and Time. </button>
    <p id="demo"></p>
</body>
```

```
</html>
```

Example 2: JavaScript in Head Section

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

Example 3: JavaScript in External File

```
function sayHello() {
          alert("Hello World")
}
```

Save this file with name hello.js then after,

```
<html>
  <head>
    <script type="text/javascript" src = "hello.js"> </script>
   </head>
   <body>
    <input type="button" onclick="sayHello()" value="Say Hello" />
  </body>
</html>
```

PROGRAMMING FUNDAMENTALS:

1. 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 allows us to work with three primitive data types:

- **Numbers,** eg. 123, 120.50, etc.
- **Strings** of text e.g. "This text string" etc.
- **Boolean** e.g. true or false.

JavaScript also defines two trivial data types, **null** and **undefined,** each of which defines only a single value. In addition to these primitive data types, JavaScript supports a composite data type known as **object**.

Note: JavaScript does not make a distinction between integer values and floating-point values. All numbers in JavaScript are represented as floating-point values. JavaScript represents numbers using the 64-bit floating-point format defined by the IEEE 754 standard.

2. JavaScript Variables:

Like many other programming languages, JavaScript has variables. Variables can be thought of as named containers. We can place data into these containers and then refer to the data simply by naming the container.

Before we use a variable in a JavaScript program, we must declare it. Variables are declared with the **var** keyword as follows.

```
<script type="text/javascript">
    var money;
    var name;
</script>
```

We can also declare multiple variables with the same **var** keyword as follows:

```
<script type="text/javascript">
    var money, name;
</script>
```

Storing a value in a variable is called **variable initialization**. We can do variable initialization at the time of variable creation or at a later point in time when we need that variable.

For instance, we might create a variable named **money** and assign the value 2000.50 to it later. For another variable, we can assign a value at the time of initialization as follows.

```
<script type="text/javascript">
    var name = "Ali";
    var money;
    money = 2000.50;
</script>
```

Note: Use the **var** keyword only for declaration or initialization, once for the life of any variable name in a document. We should not re-declare same variable twice.

JavaScript is **untyped** language. This means that a JavaScript variable can hold a value of any data type. Unlike many other languages, we don't have to tell JavaScript during variable declaration what type of value the variable will hold. The value type of a variable can change during the execution of a program and JavaScript takes care of it automatically.

3. JavaScript Variable Scope:

The scope of a variable is the region of our program in which it is defined. JavaScript variables have only two scopes.

- **Global Variables**: A global variable has global scope which means it can be defined anywhere in our JavaScript code.
- **Local Variables**: A local variable will be visible only within a function where it is defined. Function parameters are always local to that function.

Within the body of a function, a local variable takes precedence over a global variable with the same name. If we declare a local variable or function parameter with the same name as a global variable, we effectively hide the global variable. Take a look into the following example.

```
<html>
  <body onload = checkscope();>
    <script type = "text/javascript">
        var myVar = "global"; // Declare a global variable
        function checkscope( ) {
          var myVar = "local";  // Declare a local variable
          document.write(myVar);
        }
    </script>
  </body>
</html>
```

4. JavaScript Variable Names:

While naming variables in JavaScript, keep the following rules in mind.

- We should not use any of the JavaScript reserved keywords as a variable name. These keywords are mentioned in the next section. For example, **break** or **boolean** variable names are not valid.
- JavaScript variable names should not start with a numeral (0-9). They must begin with a letter or an underscore character. For example, **123test** is an invalid variable name but **_123test** is a valid one.
- JavaScript variable names are case-sensitive. For example, **Name** and **name** are two different variables.

5. JavaScript Reserved Words:

A list of all the reserved words in JavaScript are given in the following table. They cannot be used as JavaScript variables, functions, methods, loop labels, or any object names.

abstract	class	enum	function	long
boolean	const	export	goto	native
break	continue	extends	if	new
byte	debugger	false	implements	null
case	default	final	import	package
catch	class	finally	instanceof	private
char	do	float	int	protected
delete	else	for	interface	Public
return	short	static	typeof	Var

void	while	super	switch	this
volatile	with	in	synchronized	Throw
throws	transient	true	try	double

6. Operators:

Let us take a simple expression **4 + 5 is equal to 9**. Here 4 and 5 are called **operands** and '+' is called the **operator**. JavaScript supports the following types of operators.

a. Arithmetic Operators:

JavaScript supports the following arithmetic operators. Assume variable A holds 10 and variable B holds 20, then:

S.N.	Operator and Description
1	**+ (Addition)** Adds two operands **Ex:** A + B will give 30
2	**- (Subtraction)** Subtracts the second operand from the first **Ex:** A - B will give -10
3	*** (Multiplication)** Multiply both operands **Ex:** A * B will give 200
4	**/ (Division)** Divide the numerator by the denominator **Ex:** B / A will give 2
5	**% (Modulus)** Outputs the remainder of an integer division **Ex:** B % A will give 0
6	**++ (Increment)** Increases an integer value by one **Ex:** A++ will give 11
7	**-- (Decrement)** Decreases an integer value by one **Ex:** A-- will give 9

Note: Addition operator (+) works for Numeric as well as Strings. e.g. "a" + 10 will give "a10".

Example:

```
<html>
  <body>
    <script type="text/javascript">
        var a = 33;
        var b = 10;
        var c = "Test";
        var linebreak = "<br />";
```

```
        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 = ");
        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>
```

Output

```
a + b = 43
a - b = 23
a / b = 3.3
a % b = 3
a + b + c = 43Test
++a = 35
--b = 8
```

b. <u>Comparison Operators:</u>

JavaScript supports the following comparison operators. Assume variable A holds 10 and variable B holds 20, then:

S.No.	Operator and Description
1	= = **(Equal)**

	Checks if the value of two operands are equal or not, if yes, then the condition becomes true. **Ex:** (A == B) is not true.
2	**!= (Not Equal)** Checks if the value of two operands are equal or not, if the values are not equal, then the condition becomes true. **Ex:** (A != B) is true.
3	**> (Greater than)** Checks if the value of the left operand is greater than the value of the right operand, if yes, then the condition becomes true. **Ex:** (A > B) is not true.
4	**< (Less than)** Checks if the value of the left operand is less than the value of the right operand, if yes, then the condition becomes true. **Ex:** (A < B) is true.
5	**>= (Greater than or Equal to)** Checks if the value of the left operand is greater than or equal to the value of the right operand, if yes, then the condition becomes true. **Ex:** (A >= B) is not true.
6	**<= (Less than or Equal to)** Checks if the value of the left operand is less than or equal to the value of the right operand, if yes, then the condition becomes true. **Ex:** (A <= B) is true.

Example:

```
<html>
  <body>
    <script type="text/javascript">
        var a = 10;
        var b = 20;
        var linebreak = "<br />";

        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) => ");
        result = (a > b);
        document.write(result);
```

Output

```
(a == b) => false
(a < b) => true
(a > b) => false
(a != b) => true
(a >= b) => false
a <= b) => true
```

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

c. Logical Operators:

JavaScript supports the following logical operators. Assume variable A holds 10 and variable B holds 20, then:

S.No.	Operator and Description
1	**&& (Logical AND)** If both the operands are non-zero, then the condition becomes true. **Ex:** (A && B) is true.
2	**|| (Logical OR)** If any of the two operands are non-zero, then the condition becomes true. **Ex:** (A || B) is true.
3	**! (Logical NOT)** Reverses the logical state of its operand. If a condition is true, then the Logical NOT operator will make it false. **Ex:** ! (A && B) is false.

Example:

```
<html>
  <body>
    <script type="text/javascript">
        var a = true;
        var b = false;
        var linebreak = "<br />";

        document.write("(a && b) => ");
```

Output

```
(a && b) => false
(a || b) => true
!(a && b) => true
```

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

d. Bitwise Operators:

JavaScript supports the following bitwise operators. Assume variable A holds 2 and variable B holds 3, then:

S.No.	Operator and Description
1	**& (Bitwise AND)** It performs a Boolean AND operation on each bit of its integer arguments. **Ex:** (A & B) is 2.
2	**| (BitWise OR)** It performs a Boolean OR operation on each bit of its integer arguments. **Ex:** (A | B) is 3.
3	**^ (Bitwise XOR)** It performs a Boolean exclusive OR operation on each bit of its integer arguments. Exclusive OR means that either operand one is true or operand two is true, but not both. **Ex:** (A ^ B) is 1.
4	**~ (Bitwise Not)** It is a unary operator and operates by reversing all the bits in the operand. **Ex:** (~B) is -4.
5	**<< (Left Shift)** It moves all the bits in its first operand to the left by the number of places specified in the second operand. New bits are filled with zeros. Shifting a value left by one position is equivalent to multiplying it by 2, shifting two positions is equivalent to multiplying by 4, and so on. **Ex:** (A << 1) is 4.
6	**>> (Right Shift)** Binary Right Shift Operator. The left operand's value is moved right by the number of bits specified by the right operand. **Ex:** (A >> 1) is 1.
7	**>>> (Right shift with Zero)**

	This operator is just like the >> operator, except that the bits shifted in on the left are always zero. **Ex:** (A >>> 1) is 1.

Example:

```
<html>
  <body>
    <script type="text/javascript">
        var a = 2; // Bit presentation 10
        var b = 3; // Bit presentation 11
        var linebreak = "<br />";

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

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

```
(a & b) => 2
(a | b) => 3
(a ^ b) => 1
(~b) => -4
(a << b) => 16
(a >> b) => 0
```

e. Assignment Operators:

JavaScript supports the following assignment operators:

S.No.	Operator and Description
1	**= (Simple Assignment)** Assigns values from the right side operand to the left side operand **Ex:** C = A + B will assign the value of A + B into C
2	**+= (Add and Assignment)** It adds the right operand to the left operand and assigns the result to the left operand. **Ex:** C += A is equivalent to C = C + A
3	**−= (Subtract and Assignment)** It subtracts the right operand from the left operand and assigns the result to the left operand. **Ex:** C -= A is equivalent to C = C - A
4	***= (Multiply and Assignment)** It multiplies the right operand with the left operand and assigns the result to the left operand. **Ex:** C *= A is equivalent to C = C * A
5	**/= (Divide and Assignment)** It divides the left operand with the right operand and assigns the result to the left operand. **Ex:** C /= A is equivalent to C = C / A
6	**%= (Modules and Assignment)** It takes modulus using two operands and assigns the result to the left operand. **Ex:** C %= A is equivalent to C = C % A

Note: Same logic applies to Bitwise operators so they will become like <<=, >>=, >>=, &=, |= and ^=.

Example:

```
<html>
  <body>
    <script type="text/javascript">
        var a = 33;
        var b = 10;
        var linebreak = "<br />";

        document.write("Value of a => (a = b) => ");
        result = (a = b);
        document.write(result);
        document.write(linebreak);

        document.write("Value of a => (a += b) => ");
        result = (a += b);
        document.write(result);
        document.write(linebreak);
```

Output

```
Value of a => (a = b) => 10
Value of a => (a += b) => 20
Value of a => (a -= b) => 10
Value of a => (a *= b) => 100
Value of a => (a /= b) => 10
Value of a => (a %= b) => 0
```

```
        document.write("Value of a => (a -= b) => ");
        result = (a -= b);
        document.write(result);
        document.write(linebreak);

        document.write("Value of a => (a *= b) => ");
        result = (a *= b);
        document.write(result);
        document.write(linebreak);

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

        document.write("Value of a => (a %= b) => ");
        result = (a %= b);
        document.write(result);
        document.write(linebreak);
    </script>
  </body>
</html>
```

f. 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.

S.No.	Operator and Description
1	**? : (Conditional)** If Condition is true? Then value X : Otherwise value Y

Example:

```
<html>
  <body>
    <script type="text/javascript">
        var a = 10;
        var b = 20;
        var linebreak = "<br />";

        document.write ("((a > b) ? 100 : 200) => ");
        result = (a > b) ? 100 : 200;
        document.write(result);
        document.write(linebreak);

        document.write ("((a < b) ? 100 : 200) => ");
```

Output

```
((a > b) ? 100 : 200) => 200
((a < b) ? 100 : 200) => 100
```

```
        result = (a < b) ? 100 : 200;
        document.write(result);
        document.write(linebreak);
    </script>
  </body>
</html>
```

g. typeof Operator:

The **typeof** operator is a unary operator that is placed before its single operand, which can be of any type. Its value is a string indicating the data type of the operand.

The *typeof* operator evaluates to "number", "string", or "boolean" if its operand is a number, string, or boolean value and returns true or false based on the evaluation.

Here is a list of the return values for the **typeof** Operator.

Type	String Returned by typeof
Number	"number"
String	"string"
Boolean	"boolean"
Object	"object"
Function	"function"
Undefined	"undefined"
Null	"object"

Example:

```
<html>
  <body>
    <script type="text/javascript">
        var a = 10;
        var b = "String";
        var linebreak = "<br />";

        result = (typeof b == "string" ? "B is String" : "B is Numeric");
        document.write("Result => ");
        document.write(result);
        document.write(linebreak);

        result = (typeof a == "string" ? "A is String" : "A is Numeric");
        document.write("Result => ");
        document.write(result);
        document.write(linebreak);
    </script>
  </body>
</html>
```

Output

```
Result => B is String
Result => A is Numeric
```

7. Control Flow Statements:

While writing a program, there may be a situation when you need to adopt one out of a given set of paths. In such cases, you need to use conditional statements that allow our program to make correct decisions and perform right actions.

JavaScript supports conditional statements which are used to perform different actions based on different conditions.

a. if statement:

The **if** statement is the fundamental control statement that allows JavaScript to make decisions and execute statements conditionally.

Syntax:

```
if (expression){
  Statement(s) to be executed if expression is true
}
```

Example:

```
<html>
  <body>
    <script type="text/javascript">
        var age = 20;
        if( age > 18 ){
          document.write("<b>Qualifies for driving</b>");
        }
    </script>
  </body>
</html>
```

b. if...else statement:

The **'if...else'** statement is the next form of control statement that allows JavaScript to execute statements in a more controlled way.

Syntax:

```
if (expression){
  Statement(s) to be executed if expression is true
}
else{
  Statement(s) to be executed if expression is false
}
```

Example:

```
<html>
  <body>
      <script type="text/javascript">
```

```
        var age = 15;
        if( age > 18 ){
          document.write("<b>Qualifies for driving</b>");
        }
        else{
          document.write("<b>Does not qualify for driving</b>");
        }
    </script>
  </body>
</html>
```

c. if...else if... statement:

The **if...else if...** statement is an advanced form of **if...else** that allows JavaScript to make a correct decision out of several conditions.

Syntax:

```
if (expression 1){
  Statement(s) to be executed if expression 1 is true
}
else if (expression 2){
  Statement(s) to be executed if expression 2 is true
}
else if (expression 3){
  Statement(s) to be executed if expression 3 is true
}
else{
  Statement(s) to be executed if no expression is true
}
```

Example:

```
<html>
  <body>
    <script type="text/javascript">
        var book = "maths";
        if( book == "history" ){
          document.write("<b>History Book</b>");
        }
        else if( book == "maths" ){
          document.write("<b>Maths Book</b>");
        }
        else if( book == "economics" ){
          document.write("<b>Economics Book</b>");
        }
        else{
          document.write("<b>Unknown Book</b>");
        }
```

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

d. Switch Statement:

The objective of a switch statement is to give an expression to evaluate several different statements to execute based on the value of the expression. The interpreter checks each case against the value of the expression until a match is found. If nothing matches, a default condition will be used.

Syntax:

```
switch (expression)
{
  case condition 1: statement(s)
  break;
    case condition 2: statement(s)
  break;
  ...
    case condition n: statement(s)
  break;
    default: statement(s)
}
```

Example:

```
<html>
  <body>
      <script type="text/javascript">
        var grade='A';
        document.write("Entering switch block<br />");
        switch (grade)
        {
          case 'A': document.write("Good job<br />");
          break;

          case 'B': document.write("Pretty good<br />");
          break;

          case 'C': document.write("Passed<br />");
          break;

          case 'D': document.write("Not so good<br />");
          break;

          case 'F': document.write("Failed<br />");
          break;
```

```
            default:  document.write("Unknown grade<br />")
          }
          document.write("Exiting switch block");
      </script>
    </body>
</html>
```

8. Loops:

While writing a program, we may encounter a situation where we need to perform an action over and over again. In such situations, we would need to write loop statements to reduce the number of lines. JavaScript supports all the necessary loops to ease down the pressure of programming.

a. The while Loop:

The most basic loop in JavaScript is the **while** loop which would be discussed in this chapter. The purpose of a **while** loop is to execute a statement or code block repeatedly as long as an **expression** is true. Once the expression becomes **false,** the loop terminates.

Syntax:

```
while (expression){
  Statement(s) to be executed if expression is true
}
```

Example:

```
<html>
  <body>
    <script type="text/javascript">
        var count = 0;
        document.write("Starting Loop ");
        while (count < 10){
          document.write("Current Count : " + count + "<br />");
          count++;
        }
        document.write("Loop stopped!");
    </script>
  </body>
</html>
```

b. The do...while Loop:

The **do...while** loop is similar to the **while** loop except that the condition check happens at the end of the loop. This means that the loop will always be executed at least once, even if the condition is **false**.

Syntax:

```
do{
```

```
  Statement(s) to be executed;
}
```

Example:

```
<html>
  <body>
    <script type="text/javascript">
        var count = 0;
        document.write("Starting Loop" + "<br />");
        do{
          document.write("Current Count : " + count + "<br />");
          count++;
        }
        while (count < 5);
        document.write ("Loop stopped!");
    </script>
  </body>
</html>
```

c. <u>For Loop:</u>

The '**for**' loop is the most compact form of looping. It includes the following three important parts:

- The **loop initialization** where we initialize our counter to a starting value. The initialization statement is executed before the loop begins.
- The **test statement** which will test if a given condition is true or not. If the condition is true, then the code given inside the loop will be executed, otherwise the control will come out of the loop.
- The **iteration statement** where we can increase or decrease our counter.

We can put all the three parts in a single line separated by semicolons.

Syntax:

```
for (initialization; test condition; iteration statement){
  Statement(s) to be executed if test condition is true
}
```

Example:

```
<html>
  <body>
    <script type="text/javascript">
        var count;
        document.write("Starting Loop" + "<br />");
        for(count = 0; count < 10; count++){
          document.write("Current Count : " + count );
          document.write("<br />");
```

```
        }
        document.write("Loop stopped!");
    </script>
  </body>
</html>
```

d. For... in Loop:

The **for...in** loop is used to loop through an object's properties. As we have not discussed Objects yet, we may not feel comfortable with this loop. But once we understand how objects behave in JavaScript, we will find this loop very useful.

Syntax

```
for (variablename in object){
  statement or block to execute
}
```

In each iteration, one property from **object** is assigned to **variablename** and this loop continues till all the properties of the object are exhausted.

Example:

```
<html>
  <body>
    <script type="text/javascript">
        var aProperty;
        document.write("Navigator Object Properties<br /> ");
        for (aProperty in navigator) {
          document.write(aProperty);
          document.write("<br />");
        }
        document.write ("Exiting from the loop!");
    </script>
  </body>
</html>
```

e. Loop Control:

JavaScript provides full control to handle loops and switch statements. There may be a situation when we need to come out of a loop without reaching its bottom. There may also be a situation when we want to skip a part of our code block and start the next iteration of the loop.

To handle all such situations, JavaScript provides ***break*** and ***continue*** statements. These statements are used to immediately come out of any loop or to start the next iteration of any loop respectively.

The break Statement:

The **break** statement, which was briefly introduced with the *switch* statement, is used to exit a loop early, breaking out of the enclosing curly braces.

Example:

```
<html>
  <body>
    <script type="text/javascript">
      var x = 1;
      document.write("Entering the loop<br /> ");
      while (x < 20)
      {
        if (x == 5){
          break; // breaks out of loop completely
        }
        x = x + 1;
        document.write( x + "<br />");
      }
      document.write("Exiting the loop!<br /> ");
    </script>
  </body>
</html>
```

- **The continue Statement:**

The **continue** statement tells the interpreter to immediately start the next iteration of the loop and skip the remaining code block. When a **continue** statement is encountered, the program flow moves to the loop check expression immediately and if the condition remains true, then it starts the next iteration, otherwise the control comes out of the loop.

Example:

```
<html>
  <body>
    <script type="text/javascript">
        var x = 1;
        document.write("Entering the loop<br /> ");
        while (x < 10)
        {
          x = x + 1;
          if (x == 5){
            continue; // skip rest of the loop body
          }
          document.write( x + "<br />");
        }
        document.write("Exiting the loop!<br /> ");
    </script>
  </body>
</html>
```

Using Labels to Control the Flow:

Starting from JavaScript 1.2, a label can be used with **break** and **continue** to control the flow more precisely. A **label** is simply an identifier followed by a colon (:) that is applied to a statement or a block of code.

Note: Line breaks are not allowed between the **'continue'** or **'break'** statement and its label name. Also, there should not be any other statement in between a label name and associated loop.

Example 1: Break

```
<html>
  <body>
    <script type="text/javascript">
        document.write("Entering the loop!<br /> ");
        outerloop: // This is the label name
        for (var i = 0; i < 5; i++)
        {
          document.write("Outerloop: " + i + "<br />");
          innerloop:
          for (var j = 0; j < 5; j++)
          {
            if (j > 3 ) break ; // Quit the innermost loop
            if (i == 2) break innerloop; // Do the same thing
            if (i == 4) break outerloop; // Quit the outer loop
            document.write("Innerloop: " + j + " <br />");
          }
        }

        document.write("Exiting the loop!<br /> ");
    </script>
  </body>
</html>
```

Example 2: Continue

```
<html>
  <body>
    <script type="text/javascript">
      document.write("Entering the loop!<br /> ");
      outerloop: // This is the label name
      for (var i = 0; i < 3; i++)
      {
        document.write("Outerloop: " + i + "<br />");
        for (var j = 0; j < 5; j++)
        {
          if (j == 3){
            continue outerloop;
```

```
            }
            document.write("Innerloop: " + j + "<br />");
          }
        }
        document.write("Exiting the loop!<br /> ");
      </script>
    </body>
</html>
```

9. Dialog Boxes/Popup 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, we 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>
    <p>Click the following button to see the result: </p>
    <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.

Example

```
<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>
    <p>Click the following button to see the result: </p>
    <form>
      <input type="button" value="Click Me" onclick="getConfirmation();" />
    </form>
  </body>
</html>
```

3. **Prompt Dialog Box:**

The prompt dialog box is very useful when we want to pop-up a text box to get user input. Thus, it enables us 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:

1. A label which we want to display in the text box
2. 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 FUNCTION:

A function is a group of reusable code which can be called anywhere in our program. This eliminates the need of writing the same code again and again. It helps programmers in writing modular codes. Functions allow a programmer to divide a big program into a number of small and manageable functions.

Like any other advanced programming language, JavaScript also supports all the features necessary to write modular code using functions. JavaScript allows us to write our own functions as well.

1. 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>
```

2. CALLING A FUNCTION:

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

Example:

```
<html>
  <head>
    <script type="text/javascript">
      function sayHello()
      {
        document.write ("Hello there!");
      }
    </script>
  </head>
  <body>
    <p>Click the following button to call the function</p>
    <form>
      <input type="button" onclick="sayHello()" value="Say Hello">
    </form>
    <p>Use different text in write method and then try...</p>
  </body>
</html>
```

3. Function Parameters:

Till now, we have seen functions without parameters. But 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.

Example:

```
<html>
  <head>
    <script type="text/javascript">
      function sayHello(name, age)
      {
        document.write (name + " is " + age + " years old.");
      }
    </script>
  </head>
  <body>
    <p>Click the following button to call the function</p>
    <form>
      <input type="button" onclick="sayHello('Zara', 7)" value="Say Hello">
    </form>
    <p>Use different parameters inside the function and then try...</p>
  </body>
</html>
```

4. The return Statement:

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

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

Example:

```
<html>
  <head>
    <script type="text/javascript">
      function concatenate(first, last)
      {
        var full;
        full = first + last;
        return full;
      }
      function secondFunction()
      {
        var result;
        result = concatenate('Zara', 'Ali');
        document.write (result );
      }
    </script>
  </head>
  <body>
    <p>Click the following button to call the function</p>
    <form>
      <input type="button" onclick="secondFunction()" value="Call Function">
    </form>
    <p>Use different parameters inside the function and then try...</p>
  </body>
</html>
```

5. Nested Function:

Prior to JavaScript 1.2, function definition was allowed only in top level global code, but JavaScript 1.2 allows function definitions to be nested within other functions as well. Still there is a restriction that function definitions may not appear within loops or conditionals. These restrictions on function definitions apply only to function declarations with the function statement.

As we'll discuss later in the next chapter, function literals (another feature introduced in JavaScript 1.2) may appear within any JavaScript expression, which means that they can appear within **if** and other statements.

Example:

```
<html>
  <head>
    <script type="text/javascript">
        function hypotenuse(a, b) {
          function square(x) { return x*x; }
          return Math.sqrt(square(a) + square(b));
        }
        function secondFunction(){
          var result;
          result = hypotenuse(1,2);
          document.write ( result );
        }
    </script>
  </head>
  <body>
    <p>Click the following button to call the function</p>
    <form>
      <input type="button" onclick="secondFunction()" value="Call Function">
    </form>
    <p>Use different parameters inside the function and then try...</p>
  </body>
</html>
```

6. Function () Constructor:

The function statement is not the only way to define a new function; we can define our function dynamically using Function() constructor along with the new operator.

Note: Constructor is a terminology from Object Oriented Programming. We may not feel comfortable for the first time, which is OK.

Syntax:

```
<script type="text/javascript">
    var variablename = new Function(Arg1, Arg2..., "Function Body");
</script>
```

The **Function()** constructor expects any number of string arguments. The last argument is the body of the function – it can contain arbitrary JavaScript statements, separated from each other by semicolons.

Notice that the **Function()** constructor is not passed any argument that specifies a name for the function it creates. The **unnamed** functions created with the **Function()** constructor are called **anonymous** functions.

Example:

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

```
    <script type="text/javascript">
        var func = new Function("x", "y", "return x*y;");
        function secondFunction(){
          var result;
          result = func(10,20);
          document.write ( result );
        }
    </script>
  </head>
  <body>
    <p>Click the following button to call the function</p>
    <form>
      <input type="button" onclick="secondFunction()" value="Call Function">
    </form>
    <p>Use different parameters inside the function and then try...</p>
  </body>
</html>
```

7. Calling Function with Timer:

In JavaScript the timer is a very important feature, it allows us to execute a JavaScript function after a specified period, thereby making it possible to add a new dimension, time, to our website. With the help of the timer, we can run a command at specified intervals, run loops repeatedly at a predefined time, and synchronize multiple events in a particular time span.

There are various methods for calling function with timer such as:

1. The setTimeout() method:

Executes code at a specified interval.

Syntax:

```
setTimeout(function, delayTime)
```

In the preceding syntax the setTimeout() method contains two parameters, function and delayTime. The function parameter specifies the method that the timer calls and the delayTime parameter specifies the number of milliseconds to wait before calling the method.

Example:

```
<!DOCTYPE HTML>
<html>
<head>
<script type = "text/JavaScript">
function timedMsg(){
    var message;
    message = setTimeout("alert('This is use of setTimeout Function')",3000);
}
</script>
</head>
```

```
<body>
<form>
<p>Click the button below and output will be seen in 3 Second</p>
<input type = "button" value = "Start" onclick = "timedMsg()"/>
</form>
</body>
</html>
```

2. The clearTimeout() method:

Deactivates or cancels the timer that is set using the setTime() method.

Syntax:

```
clearTimeout(timer)
```

In the preceding syntax, timer is a variable that is created using the setTimeout() method.

Example:

```
<!DOCTYPE HTML>
<html>
<head>
<script type = "text/JavaScript">
var message;
function setMessage(){
    message = setTimeout("alert('This is use of setTimeout Function')",3000);
}
function clearMessage(){
    clearTimeout(message);
    alert("This is use of clearTimeout Function");
}
</script>
</head>
<body>
<form>
<p>Click the button below and output will be seen in 3 Second</p>
<input type = "button" value = "Start" onclick = "setMessage()"/>
<input type = "button" value = "Stop" onclick = "clearMessage()"/>
</form>
</body>
</html>
```

3. The setInterval() Method:

The setInterval() method repeats a given function at every given time-interval.

Syntax:

```
setInterval(function, milliseconds);
```

The first parameter is the function to be executed. The second parameter indicates the length of the time-interval between each execution.

Example:

```
<!DOCTYPE HTML>

    <html>
    <head>
    <script type = "text/JavaScript">
    function timedMsg(){
        var message;
        message = setInterval("alert('This is the use of setInterval Function')",3000);
    }
    </script>
    </head>
    <body>
    <form>
    <p>Click the button below and output will be seen in 3 Second</p>
    <input type = "button" value = "Start" onclick = "timedMsg()"/>
    </form>
    </body>
    </html>
```

4. The clearInterval() Method:

The clearInterval() method stops the executions of the function specified in the setInterval() method.

Syntax:

clearInterval(*timerVariable*)

Example:

```
<!DOCTYPE HTML>
<html>
<head>
<script type = "text/JavaScript">
var message;
function setMessage(){
    message = setInterval("alert('This is use of setInterval Function')",3000);
}
function clearMessage(){
    clearInterval(message);
    alert("This is use of clearInterval Function");
}
</script>
</head>
<body>
```

```
<form>
<p>Click the button below and output will be seen in 3 Second</p>
<input type = "button" value = "Start" onclick = "setMessage()"/>
<input type = "button" value = "Stop" onclick = "clearMessage()"/>
</form>
</body>
</html>
```

EVENT AND EVENT HANDLER:

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.

Events are a part of the Document Object Model (DOM) Level 3 and every HTML element contains a set of events which can trigger JavaScript Code.

1. onclick Event Type:

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

Example:

```
<html>
  <head>
    <script type="text/javascript">
        function sayHello() {
          alert("Hello World")
        }
    </script>
      </head>
  <body>
    <p>Click the following button and see result</p>
    <form>
      <input type="button" onclick="sayHello()" value="Click Me" />
    </form>
  </body>
</html>
```

2. onsubmit Event type:

onsubmit is an event that occurs when we try to submit a form. We can put our form validation against this event type.

Example

The following example shows how to use onsubmit. Here we are calling a **validate()** function before submitting a form data to the webserver. If **validate()** function returns true, the form will be submitted, otherwise it will not submit the data.

```
<html>
  <head>
    <script type="text/javascript">
        function validation() {
          all validation goes here
          .........
          return either true or false
        }
    </script>
  </head>
    <form method="POST" action="t.cgi" onsubmit="return validate()">
      .......
      <input type="submit" value="Submit" />
    </form>
  </body>
</html>
```

3. onmouseover and onmouseout:

These two event types will help us to create nice effects with images or even with text as well. The onmouseover event triggers when we bring our mouse over any element and the onmouseout triggers when we move our mouse out from that element.

Example:

```
<html>
  <head>
    <script type="text/javascript">
        function over() {
          document.write ("Mouse Over");
        }

        function out() {
          document.write ("Mouse Out");
        }

    </script>
  </head>
```

```
  <body>
    <p>Bring your mouse inside the division to see the result:</p>
    <div onmouseover="over()" onmouseout="out()">
      <h2> This is inside the division </h2>
    </div>

  </body>
</html>
```

4. HTML 5 Standard Events:

The standard HTML 5 events are listed here for our reference. 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

5. Form Validation:

Form validation normally used to occur at the server, after the client had entered all the necessary data and then pressed the Submit button. If the data entered by a client was incorrect or was simply missing, the server would have to send all the data back to the client and request that the form be resubmitted with correct information. This was really a lengthy process which used to put a lot of burden on the server.

JavaScript provides a way to validate form's data on the client's computer before sending it to the web server. Form validation generally performs two functions.

- **Basic Validation:**

First of all, the form must be checked to make sure all the mandatory fields are filled in. It would require just a loop through each field in the form and check for data.

- **Data Format Validation:**

Secondly, the data that is entered must be checked for correct form and value. Your code must include appropriate logic to test correctness of data.

Example:

```
<html>
  <head>
    <title>Form Validation</title>
    <script type="text/javascript">
        // Form validation code will come here.
    </script>
  </head>
  <body>
    <form action="/cgi-bin/test.cgi" 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>
```

6. Basic Form Validation:

First let us see how to do a basic form validation. In the above form, we are calling **validate ()** to validate data when **onsubmit** event is occurring. The following code shows the implementation of this validate () function.

Example:

```
<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>
```

7. Data Format Validation:

Now we will see how we can validate our entered form data before submitting it to the web server.

The following example shows how to validate an entered email address. An email address must contain at least a '@' sign and a dot (.). Also, the '@' must not be the first character of the email address, and the last dot must at least be one character after the '@' sign.

Example:

```
<script type="text/javascript">
    function validateEmail()
    {
      var emailID = document.myForm.EMail.value;
      atpos = emailID.indexOf("@");
      dotpos = emailID.lastIndexOf(".");

      if (atpos < 1 || ( dotpos - atpos < 2 ))
      {
        alert("Please enter correct email ID")
        document.myForm.EMail.focus() ;
        return false;
      }
      return( true );
    }
</script>
```

JAVASCRIPT OBJECTS:

JavaScript is an Object Oriented Programming (OOP) language. A programming language can be called object-oriented if it provides four basic capabilities to developers:

- **Encapsulation**: the capability to store related information, whether data or methods, together in an object.
- **Aggregation**: the capability to store one object inside another object.
- **Inheritance**: the capability of a class to rely upon another class (or number of classes) for some of its properties and methods.
- **Polymorphism**: the capability to write one function or method that works in a variety of different ways.

Objects are composed of attributes. If an attribute contains a function, it is considered to be a method of the object, otherwise the attribute is considered a property.

1. OBJECT PROPERTIES:

Object properties can be any of the three primitive data types, or any of the abstract data types, such as another object. Object properties are usually variables that are used internally in the object's methods, but can also be globally visible variables that are used throughout the page.

Syntax:

```
objectName.objectProperty = propertyValue;
```

For example: The following code gets the document title using the **"title"** property of the **document** object.

```
var str = document.title;
```

2. OBJECT METHODS:

Methods are the functions that let the object do something or let something be done to it. There is a small difference between a function and a method; a function is a standalone unit of statements and a method is attached to an object and can be referenced by the **this** keyword.

Methods are useful for everything from displaying the contents of the object to the screen to performing complex mathematical operations on a group of local properties and parameters.

For example: Following is a simple example to show how to use the **write ()** method of document object to write any content on the document.

```
document.write("This is test");
```

3. USER-DEFINED OBJECTS

All user-defined objects and built-in objects are descendants of an object called **Object**.

The new Operator:

The **new** operator is used to create an instance of an object. To create an object, the **new** operator is followed by the constructor method.

In the following example, the constructor methods are Object(), Array(), and Date(). These constructors are built-in JavaScript functions.

```
var employee = new Object();
var books = new Array("C++", "Perl", "Java");
var day = new Date("August 15, 1947");
```

- **The Object() Constructor:**

A constructor is a function that creates and initializes an object. JavaScript provides a special constructor function called **Object()** to build the object. The return value of the **Object()** constructor is assigned to a variable.

The variable contains a reference to the new object. The properties assigned to the object are not variables and are not defined with the **var** keyword.

Example 1: How to create an object

```
<html>
  <head>
    <title>User-defined objects</title>
    <script type="text/javascript">
      var book = new Object();   // Create the object
      book.subject = "Perl"; // Assign properties to the object
      book.author  = "Mohtashim";
    </script>
  </head>
  <body>
    <script type="text/javascript">
      document.write("Book name is : " + book.subject + "<br>");
      document.write("Book author is : " + book.author + "<br>");
    </script>
  </body>
</html>
```

Example 2: how to create an object with a User-Defined Function. Here this keyword is used to refer to the object that has been passed to a function.

```
<html>
  <head>
  <title>User-defined objects</title>
    <script type="text/javascript">
      function book(title, author){
        this.title = title;
        this.author  = author;
      }
    </script>
  </head>
  <body>
    <script type="text/javascript">
```

```
    var myBook = new book("Perl", "Mohtashim");
    document.write("Book title is : " + myBook.title + "<br>");
    document.write("Book author is : " + myBook.author + "<br>");
  </script>
 </body>
</html>
```

4. Defining Methods for an Object:

The previous examples demonstrate how the constructor creates the object and assigns properties. But we need to complete the definition of an object by assigning methods to it.

Example

The following example shows how to add a function along with an object.

```
<html>
 <head>
 <title>User-defined objects</title>

  <script type="text/javascript">
    // Define a function which will work as a method
    function addPrice(amount){
      this.price = amount;
    }

    function book(title, author){
      this.title = title;
      this.author  = author;
      this.addPrice = addPrice; // Assign that method as property.
    }
  </script>

 </head>
 <body>

  <script type="text/javascript">
    var myBook = new book("Perl", "Mohtashim");
    myBook.addPrice(100);

    document.write("Book title is : " + myBook.title + "<br>");
    document.write("Book author is : " + myBook.author + "<br>");
    document.write("Book price is : " + myBook.price + "<br>");
  </script>

 </body>
</html>
```

5. The 'with' Keyword:

The **'with'** keyword is used as a kind of shorthand for referencing an object's properties or methods.

The object specified as an argument to **with** becomes the default object for the duration of the block that follows. The properties and methods for the object can be used without naming the object.

Syntax

```
with (object){
  properties used without the object name and dot
}
```

Example:

```
<html>
  <head>
  <title>User-defined objects</title>
    <script type="text/javascript">
      // Define a function which will work as a method
      function addPrice(amount){
        with(this){
          price = amount;
        }
      }

      function book(title, author){
        this.title = title;
        this.author  = author;
        this.price = 0;
        this.addPrice = addPrice; // Assign that method as property.
      }
    </script>
  </head>
  <body>
    <script type="text/javascript">
      var myBook = new book("Perl", "Mohtashim");
      myBook.addPrice(100);

      document.write("Book title is : " + myBook.title + "<br>");
      document.write("Book author is : " + myBook.author + "<br>");
      document.write("Book price is : " + myBook.price + "<br>");
    </script>
  </body>
</html>
```

WORKING WITH BROWSER OBJECT:

It is important to understand the differences between different browsers in order to handle each in the way it is expected. So it is important to know which browser our web page is running in.

To get information about the browser our webpage is currently running in, we use the built-in **navigator** object.

1. NAVIGATOR PROPERTIES:

There are several Navigator related properties that we can use in our Web page. The following is a list of the names and descriptions of each.

S.No.	Property & Description
1	**appCodeName** This property is a string that contains the code name of the browser, Netscape for Netscape and Microsoft Internet Explorer for Internet Explorer.
2	**appVersion** This property is a string that contains the version of the browser as well as other useful information such as its language and compatibility.
3	**language** This property contains the two-letter abbreviation for the language that is used by the browser. Netscape only.
4	**mimTypes[]** This property is an array that contains all MIME types supported by the client. Netscape only.
5	**platform[]** This property is a string that contains the platform for which the browser was compiled."Win32" for 32-bit Windows operating systems
6	**plugins[]** This property is an array containing all the plug-ins that have been installed on the client. Netscape only.
7	**userAgent[]** This property is a string that contains the code name and version of the browser. This value is sent to the originating server to identify the client.

2. NAVIGATOR METHODS:

There are several Navigator-specific methods. Here is a list of their names and descriptions.

S.No.	Description
1	**javaEnabled()** This method determines if JavaScript is enabled in the client. If JavaScript is enabled, this method returns true; otherwise, it returns false.
2	**plugings.refresh** This method makes newly installed plug-ins available and populates the plugins array with all new plug-in names. Netscape only.
3	**preference(name,value)**

	This method allows a signed script to get and set some Netscape preferences. If the second parameter is omitted, this method will return the value of the specified preference; otherwise, it sets the value. Netscape only.
4	**taintEnabled()** This method returns true if data tainting is enabled; false otherwise.

3. Browser Detection

There is a simple JavaScript which can be used to find out the name of a browser and then accordingly an HTML page can be served to the user.

```
<html>
  <head>
    <title>Browser Detection Example</title>
  </head>
  <body>
    <script type="text/javascript">
        var userAgent   = navigator.userAgent;
        var opera       = (userAgent.indexOf('Opera') != -1);
        var ie          = (userAgent.indexOf('MSIE') != -1);
        var gecko       = (userAgent.indexOf('Gecko') != -1);
        var netscape    = (userAgent.indexOf('Mozilla') != -1);
        var version     = navigator.appVersion;

        if (opera){
          document.write("Opera based browser");
          // Keep your opera specific URL here.
        }

        else if (gecko){
          document.write("Mozilla based browser");
          // Keep your gecko specific URL here.
        }

        else if (ie){
          document.write("IE based browser");
          // Keep your IE specific URL here.
        }

        else if (netscape){
          document.write("Netscape based browser");
          // Keep your Netscape specific URL here.
        }

        else{
          document.write("Unknown browser");
        }
```

```
        // You can include version to along with any above condition.
        document.write("<br /> Browser version info : " + version );
    </script>
  </body>
</html>
```

DOCUMENT OBJECT MODEL OR DOM:

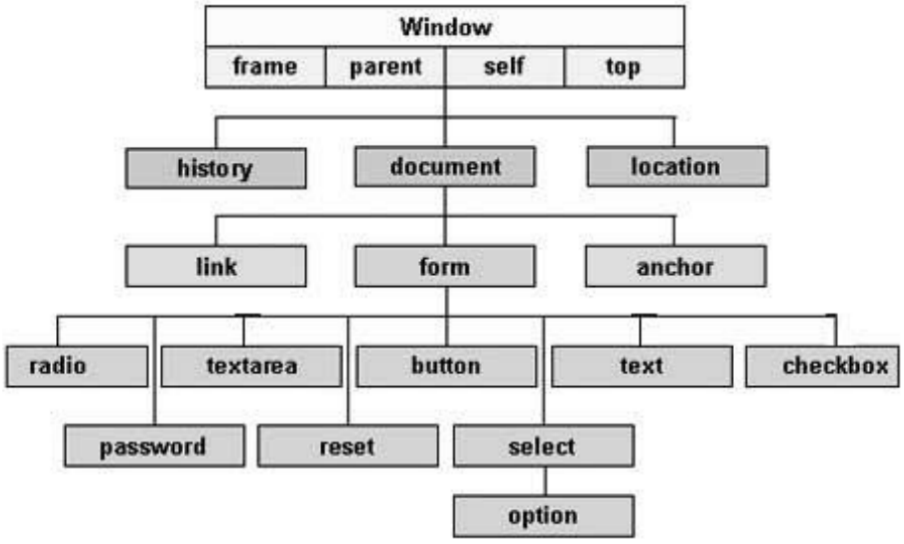
Every web page resides inside a browser window which can be considered as an object.

A Document object represents the HTML document that is displayed in that window. The Document object has various properties that refer to other objects which allow access to and modification of document content.

The way a document content is accessed and modified is called the **Document Object Model**, or **DOM**. The Objects are organized in a hierarchy. This hierarchical structure applies to the organization of objects in a Web document.

- **Window object**: Top of the hierarchy. It is the outmost element of the object hierarchy.
- **Document object**: Each HTML document that gets loaded into a window becomes a document object. The document contains the contents of the page.
- **Form object**: Everything enclosed in the <form>...</form> tags sets the form object.
- **Form control elements**: The form object contains all the elements defined for that object such as text fields, buttons, radio buttons, and checkboxes.

Here is a simple hierarchy of a few important objects:



There are several DOMs in existence. The following sections explain each of these DOMs in detail and describe how you can use them to access and modify document content.

- **The Legacy DOM**

This is the model which was introduced in early versions of JavaScript language. It is well supported by all browsers, but allows access only to certain key portions of documents, such as forms, form elements, and images.

- **The W3C DOM**

This document object model allows access and modification of all document content and is standardized by the World Wide Web Consortium (W3C). This model is supported by almost all the modern browsers.

- **The IE4 DOM**

This document object model was introduced in Version 4 of Microsoft's Internet Explorer browser. IE 5 and later versions include support for most basic W3C DOM features.

1. DOM COMPATIBILITY:

If we want to write a script with the flexibility to use either W3C DOM or IE 4 DOM depending on their availability, then we can use a capability-testing approach that first checks for the existence of a method or property to determine whether the browser has the capability you desire. For example:

```
if (document.getElementById) {
  // If the W3C method exists, use it
}
else if (document.all) {
  // If the all[] array exists, use it
}
else {
  // Otherwise use the legacy DOM
}
```

ERRORS & EXCEPTIONS HANDLING

1. TYPES OF ERRORS:

There are three types of errors in programming:

a. Syntax Errors

Syntax errors, also called **parsing errors,** 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">
    window.print(;
</script>
```

When a syntax error occurs in JavaScript, only the code contained within the same thread as the syntax error is affected and the rest of the code in other threads gets executed assuming nothing in them depends on the code containing the error.

b. 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.

c. 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 we make a mistake in the logic that drives our script and we do not get the result as expected.

We cannot catch those errors, because it depends on our business requirement what type of logic we want to put in our program.

2. The try...catch...finally Statement

The latest versions of JavaScript added exception handling capabilities. JavaScript implements the **try...catch...finally** construct as well as the **throw** operator to handle exceptions.

We can **catch** programmer-generated and **runtime** exceptions, but you cannot **catch** JavaScript syntax errors.

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.

Examples

Here is an example where we are trying to call a non-existing function which in turn is raising an exception.

```
<html>
  <head>
        <script type="text/javascript">
        function myFunc()
        {
          var a = 100;
          alert("Value of variable a is : " + a );
        }
    </script>
  </head>
  <body>
    <p>Click the following to see the result:</p>
    <form>
      <input type="button" value="Click Me" onclick="myFunc();" />
    </form>
  </body>
</html>
```

Now let us try to catch this exception using **try...catch** and display a user-friendly message. We can also suppress this message, if we want to hide this error from a user.

```
<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 );
          }
        }
    </script>
  </head>
```

```
  <body>
    <p>Click the following to see the result:</p>
    <form>
      <input type="button" value="Click Me" onclick="myFunc();" />
    </form>
  </body>
</html>
```

We can use **finally** block which will always execute unconditionally after the try/catch. Here is an example.

```
<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>
```

3. The throw Statement

We can use **throw** statement to raise our built-in exceptions or our customized exceptions. Later these exceptions can be captured and we can take an appropriate action.

Example

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

```
            var a = 100;
            var b = 0;

            try{
              if ( b == 0 ){
                throw( "Divide by zero error." );
              }

              else
              {
                var c = a / b;
              }
            }

            catch ( e ) {
              alert("Error: " + e );
            }
          }
        </script>
      </head>
      <body>
        <p>Click the following to see the result:</p>
        <form>
          <input type="button" value="Click Me" onclick="myFunc();" />
        </form>
      </body>
    </html>
```

4. The onerror() Method

The **onerror** event handler was the first feature to facilitate error handling in JavaScript. The **error** event is fired on the window object whenever an exception occurs on the page.

```
<html>
  <head>
    <script type="text/javascript">
        window.onerror = function () {
          alert("An error occurred.");
        }
    </script>
  </head>
  <body>
    <p>Click the following to see the result:</p>
    <form>
      <input type="button" value="Click Me" onclick="myFunc();" />
    </form>
  </body>
```

```
</html>
```

The **onerror** event handler provides three pieces of information to identify the exact nature of the error:

- **Error message**: The same message that the browser would display for the given error
- **URL**: The file in which the error occurred
- **Line number**: The line number in the given URL that caused the error

Here is the example to show how to extract this information.

Example

```
<html>
  <head>
    <script type="text/javascript">
        window.onerror = function (msg, url, line) {
          alert("Message : " + msg );
          alert("url : " + url );
          alert("Line number : " + line );
        }
    </script>
  </head>
  <body>
    <p>Click the following to see the result:</p>
    <form>
      <input type="button" value="Click Me" onclick="myFunc();" />
    </form>
  </body>
</html>
```

WE can display extracted information in whatever way we think it is better. We can use an **onerror** method, as shown below, to display an error message in case there is any problem in loading an image.

******

We can use **onerror** with many HTML tags to display appropriate messages in case of errors.

Chapter – 4

Style Sheet

Introduction to CSS:

CSS was invited by Håkon Wium Lie on October 10, 1994 and maintained through a group of people within the W3C called the CSS Working Group. **C**ascading **S**tyle **S**heets, fondly referred to as CSS, is a simple design language intended to simplify the process of making web pages presentable.

CSS handles the look and feel part of a web page. Using CSS, we can control the color of the text, the style of fonts, the spacing between paragraphs, how columns are sized and laid out, what background images or colors are used, layout designs, and variations in display for different devices and screen sizes as well as a variety of other effects.

CSS is easy to learn and understand but it provides powerful control over the presentation of an HTML document. Most commonly, CSS is combined with the markup languages HTML or XHTML.

Advantages:

1. **CSS Saves Time:**

We can write CSS once and then reuse same sheet in multiple HTML pages. We can define a style for each HTML element and apply it to as many Web pages as we want.

2. **Pages Load Faster:**

If we are using CSS, we 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.

3. **Easy Maintenance:**

To make a global change, simply change the style, and all elements in all the web pages will be updated automatically.

4. **Superior Styles To HTML**:

CSS has a much wider array of attributes than HTML, so we can give a far better look to our HTML page in comparison to HTML attributes.

5. **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.

6. **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.

7. **Offline Browsing**:

CSS can store web applications locally with the help of an offline cache. Using of this, we can view offline websites. The cache also ensures faster loading and better overall performance of the website.

8. **Platform Independence**:

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

CSS VERSIONS:

Cascading Style Sheets, level 1 (CSS1) was came out of W3C as a recommendation in December 1996. This version describes the CSS language as well as a simple visual formatting model for all the HTML tags.

CSS2 was became a W3C recommendation in May 1998 and builds on CSS1. This version adds support for media-specific style sheets e.g. printers and aural devices, downloadable fonts, element positioning and tables.

CSS3 was became a W3C recommendation in June 1999 and builds on older versions CSS. it has divided into documentations is called as Modules and here each module having new extension features defined in CSS2.

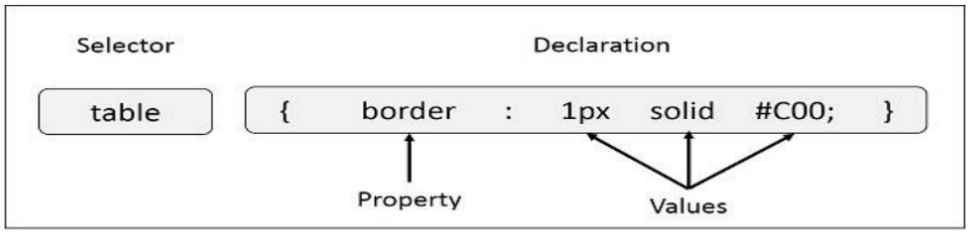
SYNTAX:

A CSS comprises of style rules that are interpreted by the browser and then applied to the corresponding elements in your document. A style rule is made of three parts:

- **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.

We can put CSS Style Rule Syntax as follows:

selector { property: value }



Example: We can define a table border as follows:

table{ border :1px solid #C00; }

Here table is a selector and border is a property and given value 1px solid #C00 is the value of that property. We can define selectors in various simple ways based on our comfort.

CSS COMMENTS:

Many times, we may need to put additional comments in our style sheet blocks. So, it is very easy to comment any part in style sheet. We can simply put our comments inside /*.....this is a comment in style sheet.....*/.

We can use /* ....*/ to comment multi-line blocks in similar way we do in C and C++ programming languages.

Example

```
<!DOCTYPE html>
<html>
  <head>
    <style>
      p {
        color: red;
        /* This is a single-line comment */
        text-align: center;
      }
      /* This is a multi-line comment */
    </style>
  </head>
  <body>
    <p>Hello World!</p>
  </body>
</html>
```

TYPES OF SELECTORS:

To select html elements in an efficient way and provide more control over selection of html elements, CSS provides various types of selectors. Selectors allow us to locate or select html elements on the page as much as generically and as much as specifically and apply styles on them.

1. The Type/Tag Selectors:

Type/tag selectors are the name of the html tags as shown in example below:

```
h1 {
  color: #36CFFF;
}
```

2. The ID Selectors:

We 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. We can make it a bit more particular. For example:

```
h1#black {
  color: #000000;
}
```

This rule renders the content in black for only <h1> elements with *id* attribute set to *black*.

The true power of *id* selectors is when they are used as the foundation for descendant selectors, For example:

```
#black h2 {
  color: #000000;
}
```

In this example all level 2 headings will be displayed in black color when those headings will lie with in tags having *id* attribute set to *black*.

3. The Class Selectors:

We 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. We can make it a bit more particular. For example:

```
h1.black {
  color: #000000;
}
```

This rule renders the content in black for only <h1> elements with class attribute set to *black*. We can apply more than one class selectors to given element. Consider the following example:

```
<p class="center bold">
  This para will be styled by the classes center and bold.
</p>
```

4. Contextual Selector:

Contextual selectors define styles that are only applied when certain tags are nested within other tags.

Contextual selectors are merely strings of two or more simple selectors separated by white space. These selectors can be assigned normal properties and, due to the rules of cascading order, they will take precedence over simple selectors. For example, the contextual selector in

```
p em {
    background: yellow;
}
```

a. The Descendant Selectors:

Suppose we want to apply a style rule to a particular element only when it lies inside a particular element. As given in the following example, style rule will apply to element only when it lies inside <ul> tag.

```
ul em {
  color: #000000;
}
```

b. The Child Selectors:

We have seen the descendant selectors. There is one more type of selector, which is very similar to descendants but have different functionality. Consider the following example

```
body > p {
  color: #000000;
}
```

This rule will render all the paragraphs in black if they are direct child of <body> element. Other paragraphs put inside other elements like or <td> would not have any effect of this rule.

c. General Sibling Selectors:

A selector that uses a general sibling selector matches elements based on sibling relationships. That is to say, the selected elements are beside each other in the HTML.

```
h2 ~ p {
  color:#009900;
}
```

This type of selector is declared using the tilde character (~). In this example, all paragraph elements (<p>) will be styled with the specified rules, but only if they are siblings of <h2> elements. There could be other elements in between the <h2> and <p>, and the styles would still apply.

Example:

```
<h2>Title</h2>
<p>Paragraph example.</p>
<p>Paragraph example.</p>
<p>Paragraph example.</p>
<div class="box">
  <p>Paragraph example.</p>
</div>
```

In this example, the styles will apply only to the first three paragraph elements. The last paragraph element is not a sibling of the <h2>element because it sits inside the element.

d. <u>Adjacent Sibling Selector:</u>

A selector that uses the adjacent sibling uses the plus symbol (+), and is almost the same as the general sibling selector. The difference is that the targeted element must be an immediate sibling, not just a general sibling.

```
p + p {
  color:red;
}
```

This example will apply the specified styles only to paragraph elements that immediately follow other paragraph elements. This means the first paragraph element on a page would not receive these styles. Also, if another element appeared between two paragraphs, the second paragraph of the two wouldn't have the styles applied.

So, if we apply this selector to the following HTML:

```
<h2>Title</h2>
<p>Paragraph example.</p>
<p>Paragraph example.</p>
<p>Paragraph example.</p>
<div class="box">
    <p>Paragraph example.</p>
    <p>Paragraph example.</p>
</div>
```

The styles will apply only to the second, third, and fifth paragraphs in this section of HTML.

5. The Attribute Selectors:

We can also apply styles to HTML elements with particular attributes. The style rule below will match all the input elements having a type attribute with a value of *text*.

```
input[type = "text"]{
  color: #009900;
}
```

The advantage to this method is that the <input type = "submit" /> element is unaffected, and the color applied only to the desired text fields.

There are following rules applied to attribute selector.

- **p[lang]** - Selects all paragraph elements with a *lang* attribute.
- **p[lang="fr"]** - Selects all paragraph elements whose *lang* attribute has a value of exactly "fr".
- **p[lang~="fr"]** - Selects all paragraph elements whose *lang* attribute contains the word "fr".
- **p[lang|="en"]** - Selects all paragraph elements whose *lang* attribute contains values that are exactly "en", or begin with "en-".

6. Multiple Style Rules:

We may need to define multiple style rules for a single element. We can define these rules to combine multiple properties and corresponding values into a single block as defined in the following example:

```
h1 {
  color: #36C;
  font-weight: normal;
  letter-spacing: .4em;
  margin-bottom: 1em;
  text-transform: lowercase;
}
```

Here all the property and value pairs are separated by a **semi colon (;)**. We can keep them in a single line or multiple lines. For better readability we keep them into separate lines.

7. Grouping Selectors:

We can apply a style to many selectors if we like. Just separate the selectors with a comma, as given in the following example

```
h1, h2, h3 {
  color: #36C;
  font-weight: normal;
  letter-spacing: .4em;
  margin-bottom: 1em;
  text-transform: lowercase;
}
```

This define style rule will be applicable to h1, h2 and h3 element as well. The order of the list is irrelevant. All the elements in the selector will have the corresponding declarations applied to them.

We can combine the various id selectors together as shown below

```
#content, #footer, #supplement {
  position: absolute;
  left: 510px;
  width: 200px;
}
```

8. The Universal Selectors:

Rather than selecting elements of a specific type, the universal selector quite simply matches the name of any element type:

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

This rule renders the content of every element in our document in black.

Pseudo Class:

CSS pseudo-classes are used to add special effects to some selectors. We do not need to use JavaScript or any other script to use those effects. A simple syntax of pseudo-classes is as follows:

selector:pseudo-class {property: value}

CSS classes can also be used with pseudo-classes:

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

The most commonly used pseudo-classes are as follows:

Value	Description
:link	Use this class to add special style to an unvisited link.
:visited	Use this class to add special style to a visited link.
:hover	Use this class to add special style to an element when you mouse over it.
:active	Use this class to add special style to an active element.
:focus	Use this class to add special style to an element while the element has focus.
:first-child	Use this class to add special style to an element that is the first child of some other element.
:lang	Use this class to specify a language to use in a specified element.

While defining pseudo-classes in a <style>...</style> block, following points should be noted:

- a:hover MUST come after a:link and a:visited in the CSS definition in order to be effective.
- a:active MUST come after a:hover in the CSS definition in order to be effective.
- Pseudo-class names are not case-sensitive.
- Pseudo-class are different from CSS classes but they can be combined.

Example:

```
<!DOCTYPE HTML>
<html>
  <head><title>Use of Pseudo-class</title>
    <style type="text/css" lang = "en">
            a:link {color:yellow;}
            a:visited {color:lime;}
            a:hover {color:blue; background-color:wheat; font-size:25px;}
```

```
            a:active {color:green;}
            a:focus {color:cyan;}
    </style>
  </head>
  <body>
    <a href=".\FirstProgram.html" target = "_blank">Click The Link</a>
  </body>
</html>
```

☑ **The :first-child pseudo-class:**

The *:first-child* pseudo-class matches a specified element that is the first child of another element and adds special style to that element that is the first child of some other element. To make :first-child work in IE <!DOCTYPE> must be declared at the top of document.

For example, to indent the first paragraph of all elements, you could use this definition:

```
<html>
  <head>
    <style type="text/css">
      div > p:first-child
      {
        text-indent: 25px;
      }
    </style>
  </head>
  <body>
    <div>
      <p>First paragraph in div. This paragraph will be indented</p>
      <p>Second paragraph in div. This paragraph will not be indented</p>
    </div>
    <p>But it will not match the paragraph in this HTML:</p>
    <div>
      <h3>Heading</h3>
      <p>The first paragraph inside the div. This paragraph will not be effected.</p>
    </div>
  </body>
</html>
```

THE :LANG PSEUDO-CLASS:

The language pseudo-class *:lang*, allows constructing selectors based on the language setting for specific tags.

This class is useful in documents that must appeal to multiple languages that have different conventions for certain language constructs. For example, the French language typically uses

angle brackets (< and >) for quoting purposes, while the English language uses quote marks (' and ').

In a document that needs to address this difference, we can use the :lang pseudo-class to change the quote marks appropriately. The following code changes the <blockquote> tag appropriately for the language being used –

```
<html>
  <head>
    <style type="text/css">
      /* Two levels of quotes for two languages*/
      :lang(en) { quotes: '"' '"'  "'" "'"; }
      :lang(fr) { quotes: "<<" ">>" "<" ">"; }
    </style>
  </head>
  <body>
    <p>...<q lang="fr">A quote in a paragraph</q>...</p>
  </body>
</html>
```

The :lang selectors will apply to all the elements in the document. However, not all elements make use of the quotes property, so the effect will be transparent for most elements.

PSEUDO ELEMENTS:

CSS pseudo-elements are used to add special effects to some selectors. We do not need to use JavaScript or any other script to use those effects. A simple syntax of pseudo-element is as follows:

```
selector:pseudo-element {property: value}
```

CSS classes can also be used with pseudo-elements:

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

The most commonly used pseudo-elements are as follows:

Value	Description
:first-line	Use this element to add special styles to the first line of the text in a selector.
:first-letter	Use this element to add special style to the first letter of the text in a selector.
:before	Use this element to insert some content before an element.
:after	Use this element to insert some content after an element.

1. The :First-Line Pseudo-Element:

The following example demonstrates how to use the *:first-line* element to add special effects to the first line of elements in the document.

```
<html>
```

```
<head>
  <style type="text/css">
    p:first-line { text-decoration: underline; }
    p.noline:first-line { text-decoration: none; }
  </style>
</head>
<body>
  <p class="noline"> This line would not have any underline because this belongs to  nline class.</p>
  <p>The first line of this paragraph will be underlined as defined in the CSS rule above. Rest of the lines in this paragraph will remain normal. This example shows how to use :first-line pseduo element to give effect to the first line of any HTML element.</p>
</body>
</html>
```

2. The :First-Letter Pseudo-Element:

The following example demonstrates how to use the *:first-letter* element to add special effects to the first letter of elements in the document.

```
<html>
<head>
  <style type="text/css">
    p:first-letter { font-size: 5em; }
    p.normal:first-letter { font-size: 10px; }
  </style>
</head>
<body>
  <p class="normal"> First character of this paragraph will be normal and will have font size 10 px;</p>
  <p>The first character of this paragraph will be 5em big as defined in the CSS rule above. Rest of the characters in this paragraph will remain normal. This example shows how to use :first-letter pseduo element to give effect to the first characters  of any HTML element.</p>
</body>
</html>
```

3. The :Before Pseudo-Element:

The following example demonstrates how to use the *:before* element to add some content before any element.

```
<html>
<head>
  <style type="text/css">
    p:before
    {
      content: url(/images/bullet.gif)
    }
```

```
    </style>
  </head>
  <body>
    <p> This line will be preceded by a bullet.</p>
    <p> This line will be preceded by a bullet.</p>
    <p> This line will be preceded by a bullet.</p>
  </body>
</html>
```

4. The :After Pseudo-Element:

The following example demonstrates how to use the *:after* element to add some content after any element.

```
<html>
  <head>
    <style type="text/css">
      p:after
      {
        content: url(/images/bullet.gif)
      }
    </style>
  </head>
  <body>
    <p> This line will be succeeded by a bullet.</p>
    <p> This line will be succeeded by a bullet.</p>
    <p> This line will be succeeded by a bullet.</p>
  </body>
</html>
```

MEASUREMENT UNITS:

CSS supports a number of measurements including absolute units such as inches, centimeters, points, and so on, as well as relative measures such as percentages and em units. We need these values while specifying various measurements in our Style rules example: **border = "1px solid red"**.

Unit	Description	Example
%	Defines a measurement as a percentage relative to another value, typically an enclosing element.	p { font-size: 16pt; line-height: 125%; }
cm	Defines a measurement in centimeters.	div { margin-bottom: 2cm; }
em	A relative measurement for the height of a font in em spaces. Because an em unit is equivalent to the size of a given font, if you assign a font to 12pt, each "em" unit would be 12pt; thus, 2em would be 24pt.	p { letter-spacing: 7em; }

ex	This value defines a measurement relative to a font's x-height. The x-height is determined by the height of the font's lowercase letter x.	p {font-size: 24pt; line-height: 3ex;}
in	Defines a measurement in inches.	p { word-spacing: .15in; }
mm	Defines a measurement in millimeters.	p { word-spacing: 15mm; }
pc	Defines a measurement in picas. A pica is equivalent to 12 points; thus, there are 6 picas per inch.	p { font-size: 20pc; }
pt	Defines a measurement in points. A point is defined as 1/72nd of an inch.	body { font-size: 18pt; }
px	Defines a measurement in screen pixels.	p { padding: 25px; }
vh	1% of viewport height.	h2 { font-size: 3.0vh; }
vw	1% of viewport width	h1 { font-size: 5.9vw; }
vmin	1vw or 1vh, whichever is smaller	p { font-size: 2vmin;}

APPLYING <DIV> TAG TO STYLE SHEET:

Div (short for division) divides the content into individual sections. Each section can then have its own formatting, as specified by the CSS. Div is a block-level container, meaning that there is a line feed after the tag.

For example, if we have the following CSS declaration:

```
.large {
  color: #00FF00;
  font-family:arial;
  font-size: 4pt;
}
```

The HTML code

```
<div class="large">
  This is a DIV sample.
</div>
```

APPLYING <SPAN> TAG TO STYLE SHEET:

Span is similar to div in that they both divide the content into individual sections. The difference is that span goes into a finer level, so we can span to format a single character if needed. There is no line feed after the tag.

For example, if we have the following CSS declaration:

```
.largefont {
  color: #0066FF;
  font-family:arial;
  font-size: 6px;
```

```
}
```

The HTML code

```
Span is not at the <span class="largefont">block level</span>.
```

LINKING STYLE SHEET:

There are four ways to associate styles with our HTML document. Most commonly used methods are inline CSS and External CSS.

1. Embedded CSS - The <style> Element:

We can put our CSS rules into an HTML document using the <style> element. This tag is placed inside <head>...</head> tags. Rules defined using this syntax will be applied to all the elements available in the document.

```
<!DOCTYPE html>
<html>
  <head>
    <style type = "text/css" media = "all">
      body {
        background-color: linen;
      }
      h1 {
        color: maroon;
        margin-left: 40px;
      }
    </style>
  </head>
  <body>
    <h1>This is a heading</h1>
    <p>This is a paragraph.</p>
  </body>
</html>
```

Attributes associated with <style> elements are:

Attribute	Value	Description
type	text/css	Specifies the style sheet language as a content-type (MIME type). This is required attribute.
media	screen tty tv projection handheld print braille aural	Specifies the device the document will be displayed on. Default value is *all*. This is an optional attribute.

	all	

2. Inline CSS - The style Attribute:

We can use *style* attribute of any HTML element to define style rules. These rules will be applied to that element only. Here is the generic syntax:

<element style = "...style rules....">

Attributes

Attribute	Value	Description
style	style rules	The value of *style* attribute is a combination of style declarations separated by semicolon (;).

Example

```
<html>
  <head>
  </head>
  <body>
    <h1 style = "color:#36C;"> This is inline CSS </h1>
  </body>
</html>
```

3. External CSS - The <link> Element:

The <link> element can be used to include an external stylesheet file in our HTML document.

An external style sheet is a separate text file with **.css** extension. We define all the Style rules within this text file and then we can include this file in any HTML document using <link> element.

Here is the generic syntax of including external CSS file

```
<head>
  <link type = "text/css" href = "..." media = "..." />
</head>
```

Attributes

Attribute	Value	Description
type	text/css	Specifies the style sheet language as a content-type (MIME type). This attribute is required.
href	URL	Specifies the style sheet file having Style rules. This attribute is a required.
media	screen tty tv projection handheld	Specifies the device the document will be displayed on. Default value is *all*. This is optional attribute.

	print braille aural all	

Example

Consider a simple style sheet file with a name *mystyle.css* having the following rules:

```
h1, h2, h3 {
  color: #36C;
  font-weight: normal;
  letter-spacing: .4em;
  margin-bottom: 1em;
  text-transform: lowercase;
}
```

Now we can include this file mystyle.css in any HTML document as follows:

```
<head>
  <link type = "text/css" href = "mystyle.css" media = " all" />
</head>
```

4. Imported CSS - @import Rule:

@import is used to import an external stylesheet in a manner similar to the <link> element. Here is the generic syntax of @import rule.

```
<head>
  <@import "URL";
</head>
```

Here URL is the URL of the style sheet file having style rules. We can use another syntax as well:

```
<head>
  <@import url("URL");
</head>
```

Example

```
<head>
  @import "mystyle.css";
</head>
```

CSS RULES OVERRIDING:

We have discussed four ways to include style sheet rules in a HTML document. Here is the rule to override any Style Sheet Rule.

- Any inline style sheet takes highest priority. So, it will override any rule defined in <style>...</style> tags or rules defined in any external style sheet file.
- Any rule defined in <style>...</style> tags will override rules defined in any external style sheet file.
- Any rule defined in external style sheet file takes lowest priority, and rules defined in this file will be applied only when above two rules are not applicable.

HANDLING OLD BROWSERS:

There are still many old browsers who do not support CSS. So, we should take care while writing our Embedded CSS in an HTML document. The following snippet shows how we can use comment tags to hide CSS from older browsers:

```
<style type="text/css">
  <!--
    body, td {
      color: blue;
    }
  -->
</style>
```

CREATING CSS FILE:

Open a text editing program. If we have Microsoft Windows PC then we have to open the program named Notepad (hold down the Windows Key on keyboard and press R, then type notepad and press enter). If we are using a Macintosh computer, launch the application named "TextEdit" (which can be found in Apps folder).

STEP 1: LET'S WRITE OUR FIRST BIT OF CSS:

Let's imagine we have a simple web page with a heading, and we want the heading to be orange and center aligned. Add the following code into your new blank text document:

```
h1 {
    color: orange;
    text-align: center;
}
```

STEP 2: SAVING THE CSS FILE:

Create a new folder on desktop (or another location) and name it **CSS-Test**. Now, back in to text editing program save the document as ***"style.css".***

STEP 3: CREATING HTML PAGE:

Our new CSS file is worthless if we don't apply it to a web page. Let's create a quick HTML page for this lesson. Create a new blank file in Notepad (or TextEdit) and add the following code:

```
<!DOCTYPE html>
```

```
<html>
    <head>
        <meta charset="utf-8">
        <title>CSS-Test</title>
    </head>
    <body>
        <h1>CSS-Test</h1>
        <div id="box-one">
        <p>This is box one.</p>
        </div>
        <div id="box-two">
        <p>This is box two.</p>
        </div>
    </body>
</html>
```

Step 4: Linking CSS File to HTML:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>CSS-Test</title>
        <link rel = "stylesheet" type = "text/css" href="link.css" media = "all">
    </head>
    <body>
        <h1>CSS-Test</h1>
        <div id="box-one">
        <p>This is box one.</p>
        </div>
        <div id="box-two">
        <p>This is box two.</p>
        </div>
    </body>
</html>
```

Step 5: Linking Multiple CSS Files to HTML:

```
<!DOCTYPE html>
    <html>
            <head>
                    <meta charset="utf-8">
                    <title>CSS-Test</title>
                    <!--External CSS-->
                    <link rel = "stylesheet" type = "text/css" href="link.css" media = "all">
                    <!--Internal CSS-->
                    <style type = "text/css">
                            #BoxOne{
```

```
                    background-color: gray;
                }

                #BoxTwo{
                    background-color: yellow;
                    padding: 10px;
                }
            </style>
        </head>
        <body>
            <h1>CSS-Test</h1>
            <div id = "BoxOne">
            <p>This is box one.</p>
            </div>
            <div id = "BoxTwo">
            <p>This is box two.</p>
            </div>
        </body>
    </html>
```

CSS – FONTS:

1. Set the Font Family:

Possible value could be any font family name.

```
<html>
  <head>
  </head>
  <body>
    <p style="font-family:georgia,garamond,serif;">
    This text is rendered in either georgia, garamond, or the default serif font
    depending on which font  you have at your system.
    </p>
  </body>
</html>
```

2. Set the Font Style:

Possible values are *normal, italic and oblique*.

```
<html>
  <head>
  </head>
  <body>
    <p style="font-style:italic;">
    This text will be rendered in italic style
    </p>
```

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

3. Set the Font Variant:

Possible values are normal and small-caps.

```
<html>
  <head>
  </head>
  <body>
    <p style="font-variant:small-caps;">
    This text will be rendered as small caps
    </p>
  </body>
</html>
```

4. Set the Font Weight:

The font-weight property provides the functionality to specify how bold a font is. Possible values could be *normal, bold, bolder, lighter, 100, 200, 300, 400, 500, 600, 700, 800, 900*.

```
<html>
  <head>
  </head>
  <body>
    <p style="font-weight:bold;">This font is bold.</p>
    <p style="font-weight:bolder;">This font is bolder.</p>
    <p style="font-weight:500;">This font is 500 weight.</p>
  </body>
</html>
```

5. Set the Font Size:

The font-size property is used to control the size of fonts. Possible values could be *xx-small, x-*small, small, medium, large, x-large, xx-large, smaller, larger, size in pixels or in %.

```
<html>
  <head>
  </head>
  <body>
    <p style="font-size:20px;">This font size is 20 pixels</p>
    <p style="font-size:small;">This font size is small</p>
    <p style="font-size:large;">This font size is large</p>
  </body>
</html>
```

6. Set the Font Size Adjust:

This property enables you to adjust the x-height to make fonts more legible. Possible value could be any number.

```
<html>
  <head>
  </head>
  <body>
    <p style="font-size-adjust:0.61;">
      This text is using a font-size-adjust value.
    </p>
  </body>
</html>
```

7. Set the Font Stretch:

This property relies on the user's computer to have an expanded or condensed version of the font being used. Possible values could be *normal, wider, narrower, ultra-condensed, extra-condensed, condensed, semi-condensed, semi-expanded, expanded, extra-expanded, ultra-expanded*.

```
<html>
  <head>
  </head>
  <body>
    <p style="font-stretch:ultra-expanded;">
      If this doesn't appear to work, it is likely that your computer doesn't have a
      condensed or expanded version of the font being used.
    </p>
  </body>
</html>
```

8. Shorthand Property:

We can use the *font* property to set all the font properties at once. For example

```
<html>
  <head>
  </head>
  <body>
    <p style="font:italic small-caps bold 15px georgia;">
    Applying all the properties on the text at once.
    </p>
  </body>
</html>
```

CSS – TEXT:

1. Set the Text Color:

Possible value could be any color name in any valid format.

```
<html>
  <head>
  </head>
  <body>
    <p style="color:red;">
    This text will be written in red.
    </p>
  </body>
</html>
```

2. Set the Text Direction:

Possible values are *ltr or rtl*.

```
<html>
  <head>
  </head>
  <body>
    <p style="direction:rtl;">
    This text will be renedered from right to left
    </p>
  </body>
</html>
```

3. Set the Space between Characters:

Possible values are *normal or a number specifying space.*.

```
<html>
  <head>
  </head>
  <body>
    <p style="letter-spacing:5px;">
    This text is having space between letters.
    </p>
  </body>
</html>
```

4. Set the Space between Words:

Possible values are *normal or a number specifying space*.

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

```
  <body>
    <p style="word-spacing:5px;">
    This text is having space between words.
    </p>
  </body>
</html>
```

5. Set the Text Indent:

Possible values are % or a number specifying indent space.

```
<html>
  <head>
  </head>
  <body>
    <p style="text-indent:1cm;">
    This text will have first line indented by 1cm and this line will remain at
    its actual position this is done by CSS text-indent property.
    </p>
  </body>
</html>
```

6. Set the Text Alignment:

Possible values are left, right, center, justify.

```
<html>
  <head>
  </head>
  <body>
    <p style="text-align:right;">
    This will be right aligned.
    </p>
    <p style="text-align:center;">
    This will be center aligned.
    </p>
    <p style="text-align:left;">
    This will be left aligned.
    </p>
  </body>
</html>
```

7. Decorating the Text:

Possible values are *none, underline, overline, line-through, blink*.

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

```
  <body>
    <p style="text-decoration:underline;">
    This will be underlined
    </p>

    <p style="text-decoration:line-through;">
    This will be striked through.
    </p>

    <p style="text-decoration:overline;">
    This will have a over line.
    </p>

    <p style="text-decoration:blink;">
    This text will have blinking effect
    </p>
  </body>
</html>
```

8. Set the Text Cases:

Possible values are *none, capitalize, uppercase, lowercase*.

```
<html>
  <head>
  </head>
  <body>
    <p style="text-transform:capitalize;">
    This will be capitalized
    </p>

    <p style="text-transform:uppercase;">
    This will be in uppercase
    </p>

    <p style="text-transform:lowercase;">
    This will be in lowercase
    </p>
    </body>
</html>
```

9. Set the White Space between Text:

Possible values are *normal, pre, nowrap*.

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

```
  <body>
    <p style="white-space:pre;">
    This text has a line break and the white-space pre setting tells the browser to honor
    it just like the HTML pre tag.</p>
  </body>
</html>
```

10. Set the Text Shadow:

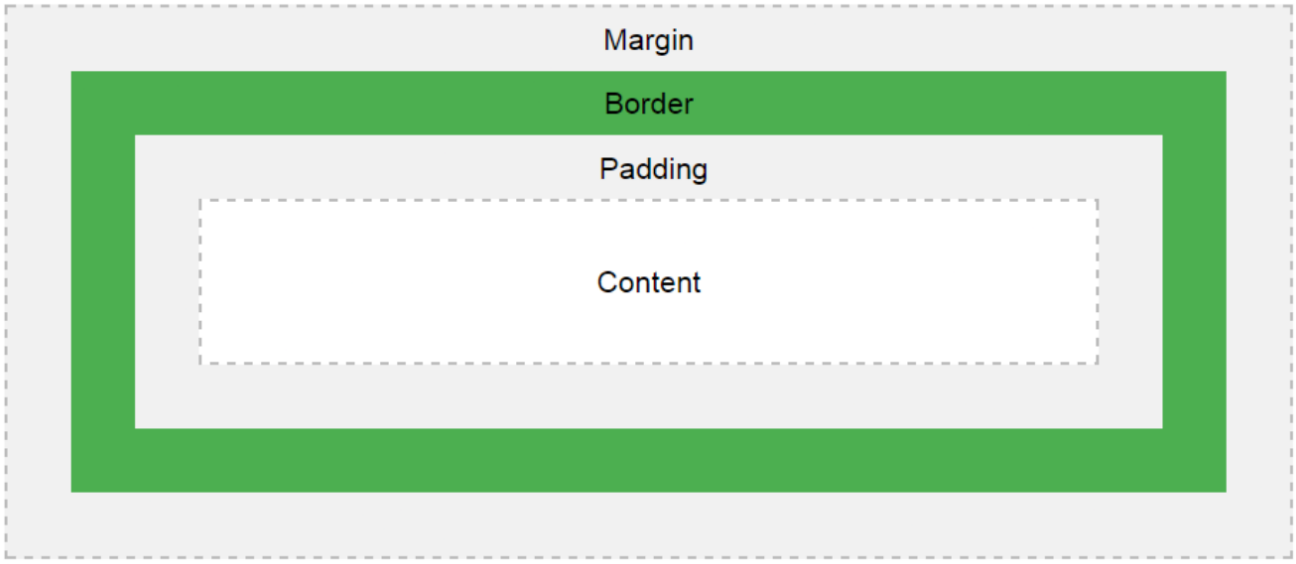
The following example demonstrates how to set the shadow around a text. This may not be supported by all the browsers.

```
<html>
  <head>
  </head>
  <body>
    <p style="text-shadow:4px 4px 8px blue;">
    If your browser supports the CSS text-shadow property, this text will have a blue shadow.
    </p>
  </body>
</html>
```

Introducing the Box Model:

In a document, each element is represented as a rectangular box. Determining the size, properties like its color, background, borders aspect and the position of these boxes is the goal of the rendering engine.

In CSS, each of these rectangular boxes is described using the standard *box model*. This model describes the space of the content taken by an element. Each box has four edges: the **margin edge**, **border edge**, **padding edge**, and **content edge**.



The **content area** is the area containing the real content of the element. It often has a background, a color or an image (in that order, an opaque image hiding the background color) and is located inside the *content edge*; its dimensions are the *content width*, or *content-box width*, and the *content height*, or *content-box height*.

If the CSS box-sizing property is set to default, the CSS properties width, min-width, max-width, height, min-height and max-height control the content size.

The **padding area** extends to the border surrounding the padding. When the content area has a background, color, or image set on it, this will extend into the padding, which is why you can think of the padding as extending the content. The padding is located inside the *padding edge*, and its dimensions are the *padding-box width* and the *padding-box height*.

The space between the padding and the content edge can be controlled using the padding-top, padding-right, padding-bottom, padding-left and the shorthand padding CSS properties.

The **border area** extends the padding area to the area containing the borders. It is the area inside the *border edge*, and its dimensions are the *border-box width* and the *border-box height*. This area depends on the size of the border that is defined by the border-width property or the shorthand border.

The **margin area** extends the border area with an empty area used to separate the element from its neighbors. It is the area inside the *margin edge*, and its dimensions are the *margin-box width* and the *margin-box height*.

The size of the margin area is controlled using the margin-top, margin-right, margin-bottom, margin-left and the shorthand margin CSS properties.

When margin collapsing happens, the margin area is not clearly defined since margins are shared between boxes.

LINKS:

- ☑ The **:link** signifies unvisited hyperlinks.
- ☑ The **:visited** signifies visited hyperlinks.
- ☑ The **:hover** signifies an element that currently has the user's mouse pointer hovering over it.
- ☑ The **:active** signifies an element on which the user is currently clicking.

Usually, all these properties are kept in the header part of the HTML document.

Remember a:hover MUST come after a:link and a:visited in the CSS definition in order to be effective. Also, a:active MUST come after a:hover in the CSS definition as follows:

```
<style type="text/css">
  a:link {color: #000000}
  a:visited {color: #006600}
  a:hover {color: #FFCC00}
  a:active {color: #FF00CC}
</style>
```

BACKGROUNDS:

We can set the following background properties of an element:

- ☑ The **background-color** property is used to set the background color of an element.

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

- ☑ The **background-image** property is used to set the background image of an element.

```
<style>
    body  {
      background-image: url("css.jpg");
      background-color: #cccccc;
    }
</style>
```

- ☑ The **background-repeat** property is used to control the repetition of an image in the background.

```
<style>
    body {
      background-image: url("css.jpg");
      background-repeat: repeat;
    }
 </style>
```

- ☑ The **background-position** property is used to control the position of an image in the background.

```
<style>
    body {
      background-image: url("css.jpg");
      background-position:100px 200px;
    }
 </style>
```

- ☑ The **background-attachment** property is used to control the scrolling of an image in the background.

```
<style>
    body  {
      background-image: url('/css/images/css.jpg');
      background-repeat: no-repeat;
      background-attachment: fixed;
    }
</style>
```

☑ The **background** property is used as a shorthand to specify a number of other background properties.

```
<p style="background:url(/images/pattern1.gif) repeat fixed;">
  This parapgraph has fixed repeated background image.
</p>
```

LIST:

Lists are very helpful in conveying a set of either numbered or bullet points. We have the following five CSS properties, which can be used to control lists:

1. The list-style-type Property:

The *list-style-type* property allows us to control the shape or style of bullet point (also known as a marker) in the case of unordered lists and the style of numbering characters in ordered lists.

Here are the values which can be used for an unordered list:

Value	Description
none	NA
disc (default)	A filled-in circle
circle	An empty circle
square	A filled-in square

Here are the values, which can be used for an ordered list:

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

Example:

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

  <body>
    <ul style="list-style-type:circle;">
      <li>Maths</li>
      <li>Social Science</li>
```

```
    <li>Physics</li>
  </ul>

  <ul style="list-style-type:square;">
    <li>Maths</li>
    <li>Social Science</li>
    <li>Physics</li>
  </ul>

  <ol style="list-style-type:decimal;">
    <li>Maths</li>
    <li>Social Science</li>
    <li>Physics</li>
  </ol>

  <ol style="list-style-type:lower-alpha;">
    <li>Maths</li>
    <li>Social Science</li>
    <li>Physics</li>
  </ol>

  <ol style="list-style-type:lower-roman;">
    <li>Maths</li>
    <li>Social Science</li>
    <li>Physics</li>
  </ol>
 </body>
</html>
```

2. The list-style-position Property:

The *list-style-position* property indicates whether the marker should appear inside or outside of the box containing the bullet points. It can have one the two values:

Value	Description
none	NA
inside	If the text goes onto a second line, the text will wrap underneath the marker. It will also appear indented to where the text would have started if the list had a value of outside.
outside	If the text goes onto a second line, the text will be aligned with the start of the first line (to the right of the bullet).

Example:

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

```
<body>
  <ul style="list-style-type:circle; list-style-position:outside;">
    <li>Maths</li>
    <li>Social Science</li>
    <li>Physics</li>
  </ul>

  <ul style="list-style-type:square;list-style-position:inside;">
    <li>Maths</li>
    <li>Social Science</li>
    <li>Physics</li>
  </ul>

  <ol style="list-style-type:decimal;list-style-position:outside;">
    <li>Maths</li>
    <li>Social Science</li>
    <li>Physics</li>
  </ol>

  <ol style="list-style-type:lower-alpha;list-style-position:inside;">
    <li>Maths</li>
    <li>Social Science</li>
    <li>Physics</li>
  </ol>
 </body>
</html>
```

3. The list-style-image Property:

The *list-style-image* allows us to specify an image so that we can use our own bullet style. The syntax is similar to the background-image property with the letters url starting the value of the property followed by the URL in brackets. If it does not find the given image then default bullets are used.

Example:

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

  <body>
    <ul>
      <li style="list-style-image: url(/images/bullet.gif);">Maths</li>
      <li>Social Science</li>
      <li>Physics</li>
    </ul>
```

```
    <ol>
      <li style="list-style-image: url(/images/bullet.gif);">Maths</li>
      <li>Social Science</li>
      <li>Physics</li>
    </ol>
  </body>
</html>
```

4. The list-style Property:

The *list-style* allows us to specify all the list properties into a single expression. These properties can appear in any order.

Example:

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

  <body>
    <ul style="list-style: inside square;">
      <li>Maths</li>
      <li>Social Science</li>
      <li>Physics</li>
    </ul>

    <ol style="list-style: outside upper-alpha;">
      <li>Maths</li>
      <li>Social Science</li>
      <li>Physics</li>
    </ol>
  </body>
</html>
```

5. The marker-offset Property:

The *marker-offset* property allows us to specify the distance between the marker and the text relating to that marker. Its value should be a length as shown in the following example:

Unfortunately, this property is not supported in IE 6 or Netscape 7.

Example:

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

  <body>
    <ul style="list-style: inside square; marker-offset:2em;">
      <li>Maths</li>
```

```
    <li>Social Science</li>
    <li>Physics</li>
  </ul>

  <ol style="list-style: outside upper-alpha; marker-offset:2cm;">
    <li>Maths</li>
    <li>Social Science</li>
    <li>Physics</li>
  </ol>
 </body>
</html>
```

TABLES:

1. The border-collapse Property:

This property can have two values *collapse* and *separate*. The following example uses both the values:

```
<html>
 <head>
  <style type="text/css">
   table.one {border-collapse:collapse;}
   table.two {border-collapse:separate;}
   td.a {
    border-style:dotted;
    border-width:3px;
    border-color:#000000;
    padding: 10px;
   }
   td.b {
    border-style:solid;
    border-width:3px;
    border-color:#333333;
    padding:10px;
   }
  </style>

 </head>
 <body>

  <table class="one">
   <caption>Collapse Border Example</caption>
   <tr><td class="a"> Cell A Collapse Example</td></tr>
   <tr><td class="b"> Cell B Collapse Example</td></tr>
  </table>
  <br />
```

Collapse Border Example

Cell A Collapse Example

Cell B Collapse Example

Separate Border Example

Cell A Separate Example

Cell B Separate Example

```
    <table class="two">
      <caption>Separate Border Example</caption>
      <tr><td class="a"> Cell A Separate Example</td></tr>
      <tr><td class="b"> Cell B Separate Example</td></tr>
    </table>
  </body>
</html>
```

2. The border-spacing Property:

The border-spacing property specifies the distance that separates adjacent cells'. borders. It can take either one or two values; these should be units of length.

If we provide one value, it will applies to both vertical and horizontal borders. Or we can specify two values, in which case, the first refers to the horizontal spacing and the second to the vertical spacing.

Example:

```
<html>
  <head>

    <style type="text/css">
      table.one {
        border-collapse:separate;
        width:400px;
        border-spacing:10px;
      }
      table.two {
        border-collapse:separate;
        width:400px;
        border-spacing:10px 50px;
      }
    </style>

  </head>
  <body>

    <table class="one" border="1">
      <caption>Separate Border Example with border-spacing</caption>
      <tr><td> Cell A Collapse Example</td></tr>
      <tr><td> Cell B Collapse Example</td></tr>
    </table>
    <br />

    <table class="two" border="1">
      <caption>Separate Border Example with border-spacing</caption>
```

Separate Border Example with border-spacing

Cell A Collapse Example
Cell B Collapse Example

Separate Border Example with border-spacing

Cell A Separate Example
Cell B Separate Example

```
    <tr><td> Cell A Separate Example</td></tr>
    <tr><td> Cell B Separate Example</td></tr>
  </table>

 </body>
</html>
```

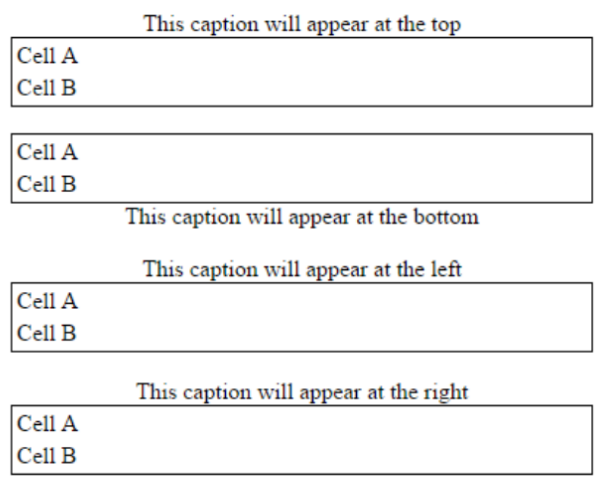
3. The caption-side Property:

The caption-side property allows us to specify where the content of a <caption> element should be placed in relationship to the table. The table that follows lists the possible values.

This property can have one of the four values *top, bottom, left* or *right*. The following example uses each value.

Example:

```
<html>
 <head>

  <style type="text/css">
   caption.top {caption-side:top}
   caption.bottom {caption-side:bottom}
   caption.left {caption-side:left}
   caption.right {caption-side:right}
  </style>

 </head>
 <body>

  <table style="width:400px; border:1px solid black;">
   <caption class="top">
   This caption will appear at the top
   </caption>
   <tr><td > Cell A</td></tr>
   <tr><td > Cell B</td></tr>
  </table>
  <br />

  <table style="width:400px; border:1px solid black;">
   <caption class="bottom">
   This caption will appear at the bottom
   </caption>
   <tr><td > Cell A</td></tr>
   <tr><td > Cell B</td></tr>
  </table>
  <br />
```



```
    <table style="width:400px; border:1px solid black;">
      <caption class="left">
      This caption will appear at the left
      </caption>
      <tr><td > Cell A</td></tr>
      <tr><td > Cell B</td></tr>
    </table>
    <br />

    <table style="width:400px; border:1px solid black;">
      <caption class="right">
      This caption will appear at the right
      </caption>
      <tr><td > Cell A</td></tr>
      <tr><td > Cell B</td></tr>
    </table>

  </body>
</html>
```

4. The empty-cells Property:

The empty-cells property indicates whether a cell without any content should have a border displayed. This property can have one of the three values - *show, hide* or *inherit*. Here is the empty-cells property used to hide borders of empty cells in the <table> element.

```
<html>
  <head>

    <style type="text/css">
      table.empty{
        width:350px;
        border-collapse:separate;
        empty-cells:hide;
      }
      td.empty{
        padding:5px;
        border-style:solid;
        border-width:1px;
        border-color:#999999;
      }
    </style>

  </head>
  <body>

    <table class="empty">
```

	Title one	Title two
Row Title	value	value
Row Title	value	

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

    <tr>
      <th>Row Title</th>
      <td class="empty">value</td>
      <td class="empty">value</td>
    </tr>

    <tr>
      <th>Row Title</th>
      <td class="empty">value</td>
      <td class="empty"></td>
    </tr>
    </table>

  </body>
</html>
```

5. The table-layout Property:

The table-layout property is supposed to help us to control how a browser should render or lay out a table. This property can have one of the three values: *fixed, auto* or *inherit*. The following example shows the difference between these properties.

```
<html>
  <head>

    <style type="text/css">
      table.auto {
        table-layout: auto
      }
      table.fixed{
        table-layout: fixed
      }
    </style>

  </head>
  <body>

    <table class="auto" border="1" width="100%">
    <tr>
      <td width="20%">10000000000000000000000000000</td>
      <td width="40%">10000000</td>
```

10000000000000000	10000000	100

100000000000	10000000	100

```
      <td width="40%">100</td>
    </tr>
    </table>
    <br />

    <table class="fixed" border="1" width="100%">
    <tr>
      <td width="20%">10000000000000000000000000000</td>
      <td width="40%">10000000</td>
      <td width="40%">100</td>
    </tr>
    </table>

  </body>
</html>
```

OUTLINES:

Outlines are very similar to borders, but there are few major differences as well:

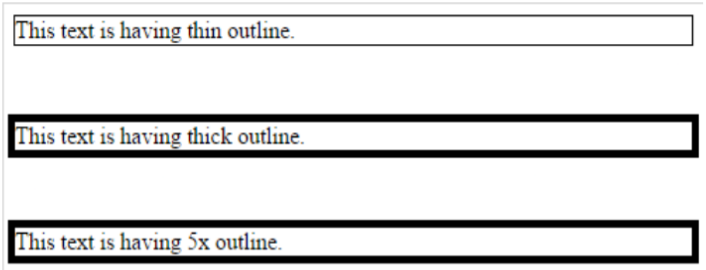
- ☑ An outline does not take up space.
- ☑ Outlines do not have to be rectangular.
- ☑ Outline is always the same on all sides; we cannot specify different values for different sides of an element.

We can set the following outline properties using CSS:

1. The outline-width Property:

The *outline-width* property specifies the width of the outline to be added to the box. Its value should be a length or one of the values *thin, medium, or thick,* just like the border-width attribute.

A width of zero pixels means no outline.



Example:

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

  <body>
    <p style="outline-width:thin; outline-style:solid;">
    This text is having thin outline.
    </p>
    <br />

    <p style="outline-width:thick; outline-style:solid;">
    This text is having thick outline.
    </p>
```

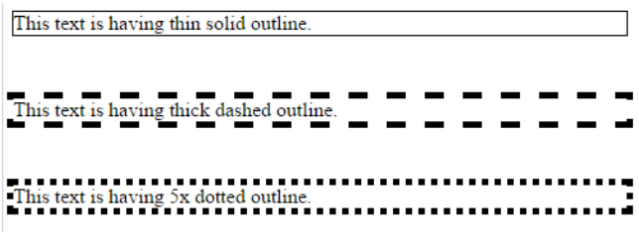
```
    <br />

    <p style="outline-width:5px; outline-style:solid;">
    This text is having 5x outline.
    </p>
  </body>
</html>
```

2. The outline-style Property:

The *outline-style* property specifies the style for the line (solid, dotted, or dashed) that goes around an element. It can take one of the following values:

- ☑ **none:** No border. (Equivalent of outline-width:0;)
- ☑ **solid:** Outline is a single solid line.
- ☑ **dotted:** Outline is a series of dots.
- ☑ **dashed:** Outline is a series of short lines.
- ☑ **double:** Outline is two solid lines.
- ☑ **groove:** Outline looks as though it is carved into the page.
- ☑ **ridge:** Outline looks the opposite of groove.
- ☑ **inset:** Outline makes the box look like it is embedded in the page.
- ☑ **outset:** Outline makes the box look like it is coming out of the canvas.
- ☑ **hidden:** Same as none.

Example:

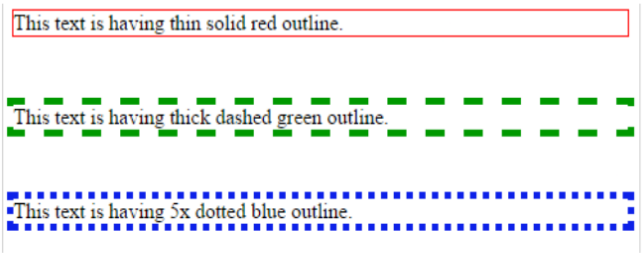



```
<html>
  <head>
  </head>

  <body>
    <p style="outline-width:thin; outline-style:solid;">
    This text is having thin solid  outline.
    </p>
    <br />

    <p style="outline-width:thick; outline-style:dashed;">
    This text is having thick dashed outline.
    </p>
    <br />

    <p style="outline-width:5px;outline-style:dotted;">
    This text is having 5x dotted outline.
    </p>
  </body>

</html>
```

3. The outline-color Property:

The *outline-color* property allows you to specify the color of the outline. Its value should either be a color name, a hex color, or an RGB value, as with the color and border-color properties.

Example:

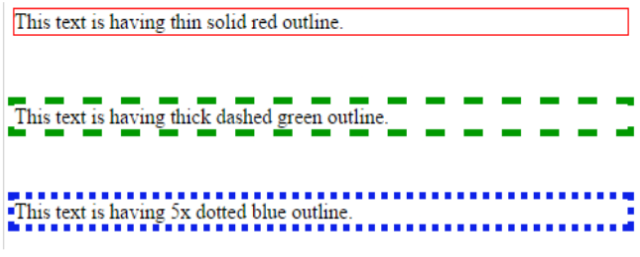



```
<html>
  <head>
  </head>

  <body>
    <p style="outline-width:thin; outline-style:solid;outline-color:red">
    This text is having thin solid red  outline.
    </p>
    <br />

    <p style="outline-width:thick; outline-style:dashed;outline-color:#009900">
    This text is having thick dashed green outline.
    </p>
    <br />

    <p style="outline-width:5px;outline-style:dotted;outline-color:rgb(13,33,232)">
    This text is having 5x dotted blue outline.
    </p>
  </body>

</html>
```

4. The outline Property:

The *outline* property is a shorthand property that allows us to specify values for any of the three properties discussed previously in any order but in a single statement.

Example:




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

  <body>
    <p style="outline:thin solid red;">
    This text is having thin solid red outline.
    </p>
    <br />

    <p style="outline:thick dashed #009900;">
    This text is having thick dashed green outline.
    </p>
```

```
    <br />

    <p style="outline:5px dotted rgb(13,33,232);">
    This text is having 5x dotted blue outline.
    </p>
  </body>

</html>
```

POSITIONING:

CSS helps us to position our HTML element. We can put any HTML element at whatever location we like. We can specify whether we want the element positioned relative to its natural position in the page or absolute based on its parent element.

Now, we will see all the CSS positioning related properties with examples:

1. Relative Positioning:

Relative positioning changes the position of the HTML element relative to where it normally appears. So "left:20" adds 20 pixels to the element's LEFT position.

We can use two values *top* and *left* along with the *position* property to move an HTML element anywhere in the HTML document.

- ☑ Move Left - Use a negative value for *left*.
- ☑ Move Right - Use a positive value for *left*.
- ☑ Move Up - Use a negative value for *top*.
- ☑ Move Down - Use a positive value for *top*.

Example:

```
<html>
  <head>
  </head>
  <body>
    <div style="position:relative;left:80px;top:2px;background-color:yellow;">
    This div has relative positioning.
    </div>
  </body>
</html>
```

2. Absolute Positioning:

An element with **position: absolute** is positioned at the specified coordinates relative to our screen top-left corner. We can use two values *top* and *left* along with the *position* property to move an HTML element anywhere in the HTML document.

- ☑ Move Left - Use a negative value for left.
- ☑ Move Right - Use a positive value for left.

- ☑ Move Up - Use a negative value for top.
- ☑ Move Down - Use a positive value for top.

Example:

```
<html>
  <head>
  </head>
  <body>
    <div style="position:absolute; left:80px; top:20px; background-color:yellow;">
    This div has absolute positioning.
    </div>
  </body>
</html>
```

3. Fixed Positioning:

Fixed positioning allows us to fix the position of an element to a particular spot on the page, regardless of scrolling. Specified coordinates will be relative to the browser window. We can use two values *top* and *left* along with the *position* property to move an HTML element anywhere in the HTML document.

- ☑ Move Left - Use a negative value for left.
- ☑ Move Right - Use a positive value for left.
- ☑ Move Up - Use a negative value for top.
- ☑ Move Down - Use a positive value for top.

Example:

```
<html>
  <head>
  </head>
  <body>
    <div style="position:fixed; left:80px; top:20px; background-color:yellow;">
    This div has fixed positioning.
    </div>
  </body>
</html>
```

LAYOUT:

CSS provides plenty of controls for positioning elements in a document. Since CSS is *the wave of the future,* why not use CSS instead of tables for page layout purposes?

The following list collects a few pros and cons of both the use of table and CSS for layout:

- ☑ Most browsers support tables, while CSS support is being slowly adopted.

- ☑ Tables are more forgiving when the browser window size changes morphing their content and wrapping to accommodate the changes accordingly. CSS positioning tends to be exact and fairly inflexible.
- ☑ Tables are much easier to learn and manipulate than CSS rules.

But each of these arguments can be reversed:

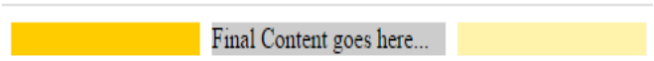
- ☑ CSS is pivotal to the future of Web documents and will be supported by most browsers.
- ☑ CSS is more exact than tables, allowing our document to be viewed as we intended, regardless of the browser window.
- ☑ Keeping track of nested tables can be a real pain. CSS rules tend to be well organized, easily read, and easily changed.

CSS also provides *table-layout* property to make our tables load much faster. Following is an example:

```
<table style="table-layout:fixed; width:600px;">
  <tr height="30">
    <td width="150">CSS table layout cell 1</td>
    <td width="200">CSS table layout cell 2</td>
    <td width="250">CSS table layout cell 3</td>
  </tr>
</table>
```

We notice the benefits more on large tables. With traditional HTML, the browser had to calculate every cell before finally rendering the table. When we set the table-layout algorithm to *fixed*, however, it only needs to look at the first row before rendering the whole table. It means our table will need to have fixed column widths and row heights.

Sample Column Layout:




```
<!DOCTYPE html>
    <html>
    <head>
  <style style="text/css">
  body {
    margin:9px 9px 0 9px;
    padding:0;
    background:#FFF;
  }

  #level0 {background:#FC0;}

  #level1 {
    margin-left:143px;
    padding-left:9px;
    background:#FFF;
  }

  #level2 {background:#FFF3AC;}
```

```
  #level3 {
    margin-right:143px;
    padding-right:9px;
    background:#FFF;
  }

  #main {background:#CCC;}
</style>
</head>
<body>
  <div id="level0">
    <div id="level1">
      <div id="level2">
        <div id="level3">
          <div id="main">
            Final Content goes here...
          </div>
        </div>
      </div>
    </div>
  </div>
</body>
</html>
```

BROWSER SAFE COLORS:

Here is the list of 216 colors which are supposed to be most safe and computer independent colors. These colors vary from hexa code 000000 to FFFFFF. These colors are safe to use because they ensure that all computers would display the colors correctly when running a 256 color palette:

000000	000033	000066	0000z99	0000CC	0000FF
003300	003333	003366	003399	0033CC	0033FF
006600	006633	006666	006699	0066CC	0066FF
009900	009933	009966	009999	0099CC	0099FF
00CC00	00CC33	00CC66	00CC99	00CCCC	00CCFF
00FF00	00FF33	00FF66	00FF99	00FFCC	00FFFF
330000	330033	330066	330099	3300CC	3300FF

333300	333333	333366	333399	3333CC	3333FF
336600	336633	336666	336699	3366CC	3366FF
339900	339933	339966	339999	3399CC	3399FF
33CC00	33CC33	33CC66	33CC99	33CCCC	33CCFF
33FF00	33FF33	33FF66	33FF99	33FFCC	33FFFF
660000	660033	660066	660099	6600CC	6600FF
663300	663333	663366	663399	6633CC	6633FF
666600	666633	666666	666699	6666CC	6666FF
669900	669933	669966	669999	6699CC	6699FF
66CC00	66CC33	66CC66	66CC99	66CCCC	66CCFF
66FF00	66FF33	66FF66	66FF99	66FFCC	66FFFF
990000	990033	990066	990099	9900CC	9900FF
993300	993333	993366	993399	9933CC	9933FF
996600	996633	996666	996699	9966CC	9966FF
999900	999933	999966	999999	9999CC	9999FF
99CC00	99CC33	99CC66	99CC99	99CCCC	99CCFF
99FF00	99FF33	99FF66	99FF99	99FFCC	99FFFF
CC0000	CC0033	CC0066	CC0099	CC00CC	CC00FF
CC3300	CC3333	CC3366	CC3399	CC33CC	CC33FF
CC6600	CC6633	CC6666	CC6699	CC66CC	CC66FF
CC9900	CC9933	CC9966	CC9999	CC99CC	CC99FF
CCCC00	CCCC33	CCCC66	CCCC99	CCCCCC	CCCCFF
CCFF00	CCFF33	CCFF66	CCFF99	CCFFCC	CCFFFF
FF0000	FF0033	FF0066	FF0099	FF00CC	FF00FF
FF3300	FF3333	FF3366	FF3399	FF33CC	FF33FF

FF6600	FF6633	FF6666	FF6699	FF66CC	FF66FF
FF9900	FF9933	FF9966	FF9999	FF99CC	FF99FF
FFCC00	FFCC33	FFCC66	FFCC99	FFCCCC	FFCCFF
FFFF00	FFFF33	FFFF66	FFFF99	FFFFCC	FFFFFF

CHAPTER – 5

EXTENSIBLE MARKUP LANGUAGE (XML)

INTRODUCTION:

XML stands for **E**xtensible **M**arkup **L**anguage. It is a text-based markup language derived from Standard Generalized Markup Language (SGML).

XML tags identify the data and are used to store and organize the data, rather than specifying how to display it like HTML tags, which are used to display the data. XML is not going to replace HTML in the near future, but it introduces new possibilities by adopting many successful features of HTML.

There are three important characteristics of XML that make it useful in a variety of systems and solutions:

- ☑ **XML is extensible:** XML allows us to create our own self-descriptive tags, or language, that suits our application.
- ☑ **XML carries the data, does not present it:** XML allows us to store the data irrespective of how it will be presented.
- ☑ **XML is a public standard:** XML was developed by an organization called the World Wide Web Consortium (W3C) and is available as an open standard.

SYNTAX:

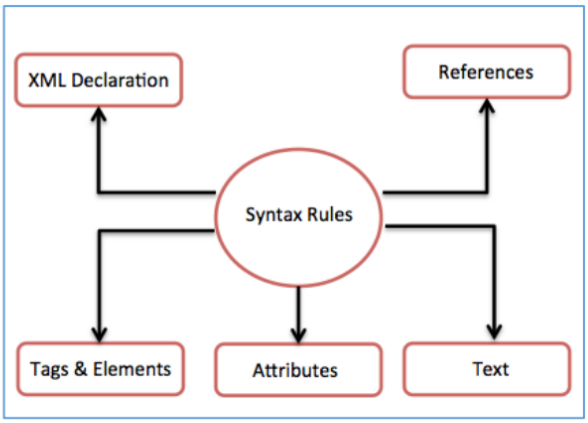
Following is a complete XML document:

```
<?xml version="1.0"?>
    <contact-info>
    <name>Ramesh Pandey</name>
    <company>Insoft</company>
    <phone>(+977) 9806587733 </phone>
</contact-info>
```

We can notice there are two kinds of information in the above example:

- ➢ markup, like <contact-info> and
- ➢ The text, or the character data, Insoft and (+977) 9806587733.

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



Let us see each component of the above diagram in detail:

XML DECLARATION:

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

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

Where *version* is the XML version and *encoding* specifies the character encoding used in the document.

SYNTAX 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.
- The XML declaration strictly needs be the first statement in the XML document.
- An HTTP protocol can override the value of *encoding* that you put in the XML declaration.

TAGS AND ELEMENTS:

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>
```

SYNTAX RULES FOR TAGS AND ELEMENTS

1. **Element Syntax:**

Each XML-element needs to be closed either with start or with end elements as shown below:

```
<element>....</element>
```

2. **Nesting Of Elements:**

An XML-element can contain multiple XML-elements as its children, but the children elements must not overlap. i.e., an end tag of an element must have the same name as that of the most recent unmatched start tag.

Following example shows incorrect nested tags:

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

Following example shows correct nested tags:

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

```
<contact-info>
<company>TutorialsPoint</company>
<contact-info>
```

3. 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>
```

4. 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>**.

ATTRIBUTES:

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

```
<a href="http://www.tutorialspoint.com/">Tutorialspoint!</a>
```

Here *href* is the attribute name and *http://www.tutorialspoint.com/* is attribute value.

SYNTAX RULES FOR XML ATTRIBUTES:

- Attribute names in XML (unlike HTML) are case sensitive. That is, *HREF* and *href* are considered two different XML attributes.
- Same attribute cannot have two values in a syntax. The following example shows incorrect syntax because the attribute *b* is specified twice: ....
- Attribute names are defined without quotation marks, whereas attribute values must always appear in quotation marks. Following example demonstrates incorrect xml syntax: ....
- In the above syntax, the attribute value is not defined in quotation marks.

XML REFERENCES:

References usually allow us 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:

1. **Entity References:**

An entity reference contains a name between the start and the end delimiters. For example **&** where *amp* is *name*. The *name* refers to a predefined string of text and/or markup.

2. **Character References:**

These contain references, such as **A** 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".

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.
- Some characters are reserved by the XML syntax itself. Hence, they cannot be used directly. To use them, some replacement-entities are used, which are listed below:

Not Allowed Character	Replacement-Entity	Character Description
<	<	less than
>	>	greater than
&	&	ampersand
'	'	apostrophe
"	"	quotation mark

XML USAGE:

- ☑ XML can work behind the scene to simplify the creation of HTML documents for large web sites.
- ☑ XML can be used to exchange the information between organizations and systems.
- ☑ XML can be used for offloading and reloading of databases.
- ☑ XML can be used to store and arrange the data, which can customize your data handling needs.
- ☑ XML can easily be merged with style sheets to create almost any desired output.
- ☑ Virtually, any type of data can be expressed as an XML document.

XML SPECIFICATION:

- ☑ **XML:**
  Defines the syntax of XML.
- ☑ **XLL (Extensible, Linking Language):**
  Defines a standard way to represent links between resources.

- ☑ **XSL (Extensible Style Language):**
  Will define a standard style sheet language for XML.
- ☑ **XUA (XML User Agent):**
  Will help standardize XML User Agents (like browser).

DOCUMENTS:

An XML *document* is a basic unit of XML information composed of elements and other markup in an orderly package. An XML *document* can contains wide variety of data. For example, database of numbers, numbers representing molecular structure or a mathematical equation.

Example:

```
<?xml version="1.0"?>
<contact-info>
  <name>Tanmay Patil</name>
  <company>TutorialsPoint</company>
  <phone>(011) 123-4567</phone>
</contact-info>
```

The following image depicts the parts of XML document.

Document Prolog Section:

The **document prolog** comes at the top of the document, before the root element. This section contains:

- XML declaration
- Document type declaration

Document Elements Section:

Document Elements are the building blocks of XML. These divide the document into a hierarchy of sections, each serving a specific purpose. You can separate a document into multiple sections so that they can be rendered differently, or used by a search engine. The elements can be containers, with a combination of text and other elements.

STRUCTURE OF XML:

According to the specification, XML documents have both logical and physical structure. A document is built up from storage units called entities. They can contain parsed or unparsed

data. Parsed entities contain characters that formed either character data or markups. Markups are used to encode the logical and physical structure of the document. Both structures are subject of limitations, well-formalness and validity.

☑ Logical Structure:

The document logical structure consists of declarations, elements, comments, processing instructions and character references. The UML diagram in figure below illustrates this:

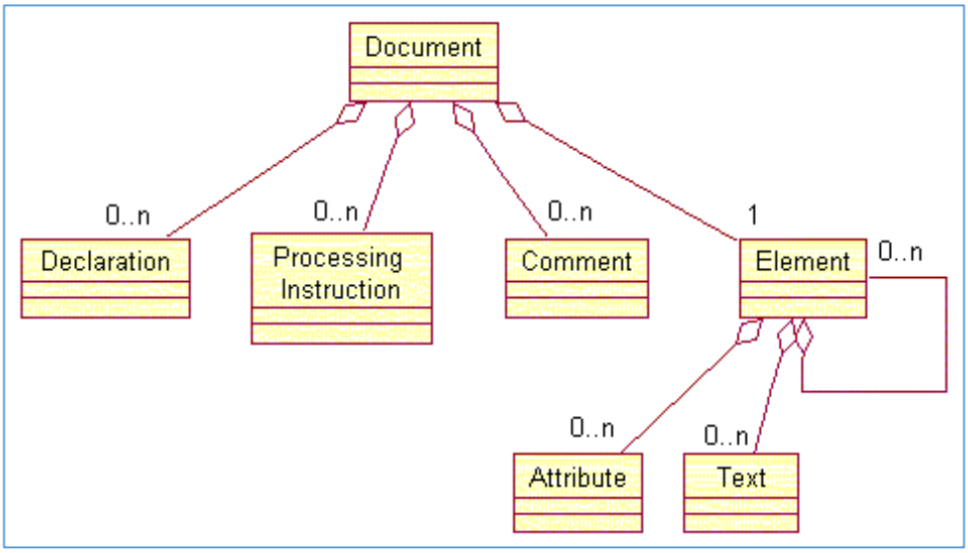


Fig: XML Document Logical Structure

Every well-formed document contains one or more elements that form a tree hierarchy. Consequently, there is exactly one element at the highest level of the hierarchy that serves as a root for the tree.

The allowed locations of processing instructions, comments and declarations regarding to the element hierarchy are given by the grammar rules in the specification.

The document logical structure can vary in the level of details and structure. This depends on the needs of the concrete application. For instance text nodes can be further decomposed into characters. Also, declarations can be represented either as a monolithic item or as a list of single declarations.

At the moment the specifications in W3C uses several logical models. The most important are:

- **DOM (Document Object Model)**, which provides logical model for XML and HTML documents and a set of interfaces to it. DOM has 3 variations: leve1, 2 and 3. They provide different level of details on the document content.
- **XML Information Set** defines abstract data model of the document information content. Its purpose is to serve as a common set of terms used by the other specifications;
- **Models** used by XPath and XPointer specifications. These models treat an XML document as a tree of nodes. There are mappings from them to XML Information Set.

XML processor takes the responsibility to provide interface to the logical structure. Furthermore, applications can extend it and provide new features. However, before doing that

certain recognition of document physical structure is required followed by reading and processing of declared entities. This is a process that depends on the parser and therefore the resulted logical structure exposed to the application can vary.

☑ PHYSICAL STRUCTURE:

This section describes the entity concept and presents three classifications of entities as they are defined in the specification. For each entity type examples show the syntax for entity declaration and usage. Finally, the section gives some remarks about the process of entity expansion.

Entities:

Every XML document is composed of storage units called entities. An entity has a name and content. The name is used to form a reference to the entity. There are two exceptions of entities without names, the document entity and the part of the DTD that is not contained in the document (so called external subset).

An entity can contain references to other entities. There is a special entity called document entity or root that serves as a main storage unit. XML processors always start document processing from that unit, which can contain the whole document.

Entities can take different forms. They can form a separate text file with XML data identified by an URI and obtained and processed by the parser. Processing usually results in inclusion of the text in the place of reference. Or they can be files that contain any kind of resources including non-textual objects. In this case the entity content is not retrieved and processed by the parser. The application is informed about the resource and can take some actions. Also, entities can be defined as named strings inside another entity and referred to from several places.

Entity Types:

We have three classifications of entities:

1. ***Parsed And Unparsed Entities:***

According to the specification parsed entities contain text that is intended to be processed by the parser and is considered as an integral part of the document. Unparsed entities are resources that can be of any type including text objects.

The main difference between parsed and unparsed entities is in the treatment taken by the parser. The XML parser never processes unparsed entities. Instead, their presence is reported to the application.

The following are two declarations of parsed entities:

```
<!ENTITY full_name “XML Technology in E_Commerce” > ---------- (1)
<!ENTITY short_name “XTEC” > ---------- (2)
```

In this examples the strings **full_name** and **short_name** are entity names and the text in quotations is the entity value. Here two entities take the form of named strings.

These entities can be used in the document by using references to them:

```
<courses> ---------- (3)
    <course>
```

```
        <title>&full_name;</title>
        <abrev>&short_name;</abrev>
    </course>
</courses>
```

The entity reference is made up by the entity name delimited with & and ; characters. In the example above if the name of the course is used several times across the document, it can be declared only once as an entity value and a reference to it can be used multiple times. The value will be included by replacing the reference. This process is performed by the parser and is called expansion of the entity reference. If a change of the course name is required, it is localized at a single place, the entity value. This mechanism makes the document content more manageable and reduces the potential for errors.

In the example above the entities take the form of a string declared in some storage unit, usually a text file. There is no separate storage unit for them. It is possible to declare an entity whose value is contained outside of the storage object that contains the declaration, e.g. in a separate file. This type of entity is called external. In (4) and (5) an example of external entity declaration is shown:

<!ENTITY course_objectives SYSTEM "http://trese.cs.utwente.nl/courses/xtec/objectives.xml" > ---------- (4)

This declaration uses system identifier, which is an URI after the SYSTEM keyword that can be used to retrieve the entity resource. There is another variant based on a public identifier:

<!ENTITY course_objectives PUBLIC "-//Twente University//XTEC Objectives//EN" "http://trese.cs.utwente.nl/courses/xtec/objectives.xml" > ---------- (5)

The mechanism of public identifiers is inherited from SGML. It defines a set of names within an organization but the names may be invisible outside.

The exact syntax rules for these two forms are in the XML specification.

Entity reference syntax to such kind of entities is the same, but the treatment by the XML parser is different. There can be an expansion of the reference, but this depends on the type of the parser. Detailed discussion about parser types (validating and non-validating) is contained in the specification. Generally, non-validating parsers are not required to process external parsed entities.

The usage of external entities allows for modularization of XML documents and independent authoring by multiple authors.

Unparsed entities are declared in the following way:

<!ENTITY logo SYSTEM "http://trese.cs.utwente.nl/courses/xtec/logo.gif" NDATA gif> ------ (6)

The form with public identifier is also possible.

This example entity declares an image resource in GIF format. The entity is recognized as unparsed by the NDATA keyword followed by a notation name, in this case "gif". XML processor must inform the application about the entity identifiers and notation name. Notation carries out information about the format of an unparsed entity. Notations are declared like that:

<!NOTATION gif SYSTEM “GIF” > ---------- (7)

According to the validity constraints notation names must be declared.

XML specification doesn’t specify notation semantics. It is assumed that the application will handle the entity on the base of the notation name, but it also can take another specific handling.

Unparsed entities are only used by name. It can be the value of attributes with type ENTITY or ENTITIES.

2. ***General And Parameter Entities:***

The distinction between these types lies in the scope of their usage. Parameter entities are always parsed and used only in the DTD part of the XML document, whereas the general entities are used in the document content. The entities of these types are declared and used differently.

Assume that in the DTD we have several elements that share a common set of attributes. XML specification does not allow single attribute definition that can be referred to multiple times in the context of different elements. Attribute definition is always attached to an element declaration. Consequently, if several elements share a common attribute the attribute definition will be repeated for each element. The mechanism of parameter entities provides a work around for the problem. Instead of repeating the definition of attributes we can declare a parameter entity like this:

```
<!ENTITY % common_attr  “id ID ---------- (8)
#REQUIRED meta CDATA
#IMPLIED time CDATA
#IMPLIED”>
```

and use it for more than one element:

```
<!ELEMENT el1 ……..> ---------- (9)
<!ATTLIST    el1 %common_attr;>
<!ELEMENT el2 ……..>
<!ATTLIST    el2 %common_attr;>
……
……
```

Parameter entity declaration includes % character before the name and an entity reference uses % and ; as delimiters. Since the context of both types is different, we can have two entities with the same name, but one as general and one as parameter:

```
<!ENTITY % common_attr  “id ID ---------- (10)
#REQUIRED meta CDATA
#IMPLIED time CDATA
#IMPLIED”>
<!ENTITY common_attr “Example of general and parameter entities with the same names”>
```

3. ***Internal and External Entities:***

For internal entities there is no separate physical storage object. In the above examples internal entities are (1), (2), (8) and (10). Internal entities are always parsed.

If an entity is not internal it is an external entity like (4), (5) and (6). It can be noticed that the presence of SYSTEM denotes an entity as external. NDATA keyword marks an external entity as unparsed.

ELEMENTS:

XML elements can be defined as building blocks of an XML. Elements can behave as containers to hold text, elements, attributes, media objects or all of these.

Each XML document contains one or more elements, the scope of which are either delimited by start and end tags, or for empty elements, by an empty-element tag.

Syntax

```
<element-name attribute1 attribute2>
     ....content
</element-name>
```

Where

- ☑ element-name is the name of the element. The name its case in the start and end tags must match.
- ☑ attribute1, attribute2 are attributes of the element separated by white spaces. An attribute defines a property of the element. It associates a name with a value, which is a string of characters. An attribute is written as:
  name = "value"
  name is followed by an = sign and a string value inside double(" ") or single(' ') quotes.

EMPTY ELEMENT:

An empty element (element with no content) has following syntax:

```
<name attribute1 attribute2.../>
```

Example:

```
<?xml version="1.0"?>
<contact-info>
  <address category="residence">
    <name>Tanmay Patil</name>
    <company>TutorialsPoint</company>
    <phone>(011) 123-4567</phone>
  <address/>
</contact-info>
```

NAMING RULES:

XML elements must follow these naming rules:

- Element names are case-sensitive i.e. Address is not same as address.
- Element names must start with a letter (alphabets) or underscore (_)
- Element names cannot start with the letters xml (or XML, or Xml, etc)

- Element names can contain letters, digits, hyphens, underscores, and periods (.)
- Element names cannot contain spaces.

Any name can be used, no words are reserved (except xml).

Best Naming Practices:

- Create descriptive names, like this: <person>, <firstname>, <lastname>.
- Create short and simple names, like this: <book_title> not like this: <the_title_of_the_book>.
- Avoid "-". If you name something "first-name", some software may think you want to subtract "name" from "first".
- Avoid ".". If you name something "first.name", some software may think that "name" is a property of the object "first".
- Avoid ":". Colons are reserved for namespaces (more later).

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

Naming Styles:

There are no naming styles defined for XML elements. But here are some commonly used:

Style	Example	Description
Lower case	<firstname>	All letters lower case
Upper case	<FIRSTNAME>	All letters upper case
Underscore	<first_name>	Underscore separates words
Pascal case	<FirstName>	Uppercase first letter in each word
Camel case	<firstName>	Uppercase first letter in each word except the first

If we choose a naming style, it is good to be consistent! XML documents often have a corresponding database. A common practice is to use the naming rules of the database for the XML elements.

Element Content Models:

To declare the syntax of an element in a DTD, we use the <!ELEMENT> element like this: <!ELEMENT*name content_model*>. In this syntax, *name* is the name of the element we're declaring and *content_model* is the content model of the element. A *content model* indicates what content the element is allowed to have, for example, we can allow child elements or text data, or we can make the element empty by using the EMPTY keyword, or we can allow any content by using the ANY keyword.

```
<!DOCTYPE document [
<!ELEMENT document (employee)*>
  .
  .
  .
]>
```

This <!ELEMENT> element not only declares the <document> element, but it also says that the <document> element may contain <employee> elements. When we declare an element in this way, we also specify what contents that element can legally contain; the syntax for doing that is a little involved. The following sections dissect that syntax, taking a look at how to specify the content model of elements, starting with the least restrictive content model of all—ANY, which allows any content at all.

1. **Handling Any Content:**

If we give an element the content model ANY, that element can contain any content, which means any elements and/or any character data. What this really means is that we're turning off validation for this element because the contents of elements with the content model ANY are not even checked. Here's how to specify the content model ANY for an element named <document>:

```
<!DOCTYPE document [
<!ELEMENT document ANY>
   .
   .
   .
]>
```

As far as the XML validator is concerned, this just turns off validation for the <document> element. It's usually not a good idea to turn off validation, but we might want to turn off validation for specific elements, for example, if we want to debug a DTD that's not working. It's usually far preferable to actually list the contents we want to allow in an element, such as any possible child elements the element can contain.

2. **Specifying Child Elements:**

We can specify what child elements an element can contain in that element's content model. For example, we can specify that an element can contain another element by explicitly listing the name of the contained element in parentheses, like this:

```
<!DOCTYPE document [
<!ELEMENT document (employee)*>
   .
   .
   .
]>
```

This specifies that a <document> element can contain <employee> elements. The * here means that a <document> element can contain any number (including zero) <employee> elements. (We'll talk about what other possibilities besides * are available in a few pages.) With this line in a DTD, we can now start placing an <employee> element or elements inside a <document> element, this way:

```
<?xml version = "1.0" standalone="yes"?>
<!DOCTYPE document [
```

```
<!ELEMENT document (employee)*>
]>
<document>
 <employee>
  .
  .
  .
 </employee>
</document>
```

Note, however, that this is no longer a valid XML document because we haven't specified the syntax for individual <employee> elements. Because <employee> elements can contain <name>, <hiredate>, and <projects> elements, *in that order*, we can specify a content model for<employee> elements this way:

```
<?xml version = "1.0" standalone="yes"?>
<!DOCTYPE document [
<!ELEMENT document (employee)*>
<!ELEMENT employee (name, hiredate, projects)>
<!ELEMENT name (lastname, firstname)>
<document>
 <employee>
  <name>
   <lastname>Kelly</lastname>
   <firstname>Grace</firstname>
  </name>
  <hiredate>October 15, 2005</hiredate>
  <projects>
   <project>
    <product>Printer</product>
    <id>111</id>
    <price>$111.00</price>
   </project>
   <project>
    <product>Laptop</product>
    <id>222</id>
    <price>$989.00</price>
   </project>
  </projects>
 </employee>
</document>
```

Listing multiple elements in a content model this way is called creating a *sequence*. We use commas to separate the elements we want to have appear, and then the elements have to appear in that sequence in our XML document. For example, if we declare this sequence in the DTD:

```
<!ELEMENT employee (name, hiredate, projects)>
```

then inside an <employee> element, the <name> element must come first, followed by the <hiredate>element, followed by the <projects> element, like this:

```
<employee>
 <name>
  <lastname>Kelly</lastname>
  <firstname>Grace</firstname>
 </name>
 <hiredate>October 15, 2005</hiredate>
 <projects>
  <project>
   <product>Printer</product>
   <id>111</id>
   <price>$111.00</price>
  </project>
  <project>
   <product>Laptop</product>
   <id>222</id>
   <price>$989.00</price>
  </project>
 </projects>
</employee>
```

This example introduces a whole new set of elements <name>, <hiredate>, <lastname>, and so on that don't contain other elements at all—they contain text. So how can we specify that an element contains text? Read on.

3. Handling Text Content:

In the preceding section's example, the <name>, <hiredate>, and <lastname> elements contain text data. In DTDs, non-markup text is considered parsed character data (in other words, text that has already been parsed, which means the XML processor shouldn't touch that text because it doesn't contain markup). In a DTD, we refer to parsed character data as #PCDATA. Note that this is the only way to refer to text data in a DTD we can't say anything about the actual format of the text, although that might be important if we're dealing with numbers. In fact, this lack of precision is one of the reasons that XML schemas were introduced.

Here's how to give the text-containing elements in the PCDATA content model example:

```
<?xml version = "1.0" standalone="yes"?>
<!DOCTYPE document [
<!ELEMENT document (employee)*>
<!ELEMENT employee (name, hiredate, projects)>
<!ELEMENT name (lastname, firstname)>
<!ELEMENT lastname (#PCDATA)>
<!ELEMENT firstname (#PCDATA)>
<!ELEMENT hiredate (#PCDATA)>
```

```
<!ELEMENT projects (project)*>
<!ELEMENT project (product,id,price)>
<!ELEMENT product (#PCDATA)>
<!ELEMENT id (#PCDATA)>
<!ELEMENT price (#PCDATA)>
]>
<document>
 <employee>
  <name>
   <lastname>Kelly</lastname>
   <firstname>Grace</firstname>
  </name>
  <hiredate>October 15, 2005</hiredate>
  <projects>
   <project>
    <product>Printer</product>
    <id>111</id>
    <price>$111.00</price>
   </project>
   <project>
    <product>Laptop</product>
    <id>222</id>
    <price>$989.00</price>
   </project>
  </projects>
 </employee>
</document>
```

4. Specifying Multiple Child Elements:

There are a number of options for declaring an element that can contain child elements. We can declare the element to contain a single child element:

```
<!ELEMENT document (employee)>
```

We can declare the element to contain a list of child elements, in order:

```
<!ELEMENT document (employee, contractor, partner)>
```

We can also use symbols with special meanings in DTDs, such as *, which means "zero or more of," as in this example, where we're allowing zero or more <employee> elements in a <document> element:

```
<!ELEMENT document (employee)*>
```

There are a number of other ways of specifying multiple children by using symbols. (This syntax is actually borrowed from regular expression handling in the Perl language, so if we know that language, we have a leg up here.) Here are the possibilities:

- **x+**—Means x can appear one or more times.
- **x***—Means x can appear zero or more times.
- **x?**—Means x can appear once or not at all.
- **x, y**—Means x followed by y.
- **x | y**—Means x *or* y—but not both.

The following sections take a look at these options.

5. **Allowing One or More Children:**

We might want to specify that a <document> element can contain between 200 and 250 <employee>elements, and if we do, we're out of luck with DTDs because DTD syntax doesn't give us that kind of precision. On the other hand, we still do have some control here; for example, we can specify that a <document> element must contain one or more <employee> elements if we use a + symbol, like this:

```
<!ELEMENT document (employee)+>
```

Here, the XML processor is being told that a <document> element has to contain at least one <employee> element.

6. **Allowing Zero or More Children:**

By using a DTD, we can use the * symbol to specify that we want an element to contain any number of child elements that is, zero or more child elements. We saw this in action earlier, when we specified that the <document> element may contain <employee> elements in the above example:

```
<!ELEMENT document (employee)*>
```

7. **Allowing Zero or One Child:**

When using a DTD, we can use ? to specify zero or one child elements. Using ? indicates that a particular child element *may* be present once in the element we're declaring, but it need not be. For example, here's how to indicate that a <document> element may contain zero or one <employee> elements:

```
<!ELEMENT document (employee)?>
```

ELEMENT OCCURRENCE INDICATORS:

There are seven indicators:

☑ ORDER INDICATORS:

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

1. **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 we 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).

2. **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>
```

3. **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.

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.

2. **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:

A working example:

An XML file called "Myfamily.xml":

```
<?xml version="1.0" encoding="UTF-8"?>
<persons xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="family.xsd">
<person>
<full_name>Hege Refsnes</full_name>
 <child_name>Cecilie</child_name>
</person>
```

```
<person>
<full_name>Tove Refsnes</full_name>
<child_name>Hege</child_name>
<child_name>Stale</child_name>
<child_name>Jim</child_name>
<child_name>Borge</child_name>
</person>

<person>
<full_name>Stale Refsnes</full_name>
</person>
</persons>
```

The XML file above contains a root element named "persons". Inside this root element we have defined three "person" elements. Each "person" element must contain a "full_name" element and it can contain up to five "child_name" elements.

Here is the schema file "family.xsd":

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs=http://www.w3.org/2001/XMLSchema elementFormDefault="qualified">
<xs:element name="persons">
<xs:complexType>
<xs:sequence>
<xs:element name="person" maxOccurs="unbounded">
<xs:complexType>
<xs:sequence>
<xs:element name="full_name" type="xs:string"/>
<xs:element name="child_name" type="xs:string" minOccurs="0" maxOccurs="5"/>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:schema>
```

☑ Group Indicators:

Group indicators are used to define related sets of elements.

1. Element Groups:

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

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

```
</xs:group>
```

We 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 we have defined a group, we 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>
```

2. 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 we have defined an attribute group, we 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>
```

CDATA:

The term CDATA means, Character Data. CDATA are defined as blocks of text that are not parsed by the parser, but are otherwise recognized as markup.

The predefined entities such as <, >, and & require typing and are generally difficult to read in the markup. In such cases, CDATA section can be used. By using CDATA section, you are commanding the parser that the particular section of the document contains no markup and should be treated as regular text.

Syntax

```
<![CDATA[
  characters with markup
]]>
```

The above syntax is composed of three sections:

- **CDATA Start section -** CDATA begins with the nine-character delimiter **<![CDATA[**
- **CDATA End section -** CDATA section ends with]**]>** delimiter.
- **CData section -** Characters between these two enclosures are interpreted as characters, and not as markup. This section may contain markup characters (<, >, and &), but they are ignored by the XML processor.

Example

The following markup code shows example of CDATA. Here, each character written inside the CDATA section is ignored by the parser.

```
<script>
<![CDATA[
  <message> Welcome to TutorialsPoint </message>
]] >
</script >
```

In the above syntax, everything between <message> and </message> is treated as character data and not as markup.

CDATA Rules:

The given rules are required to be followed for XML CDATA:

- CDATA cannot contain the string "]]>" anywhere in the XML document.
- Nesting is not allowed in CDATA section.

DTD (DOCUMENT TYPE DECLARATION):

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. An XML DTD can be either specified inside the document, or it can be kept in a separate document and then liked separately.

Syntax

```
<!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.

Syntax

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

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

Example

```
<?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>Tanmay Patil</name>
  <company>TutorialsPoint</company>
  <phone>(011) 123-4567</phone>
</address>
```

Let us go through the above code:

Start Declaration: Begin the XML declaration with following statement

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

DTD: Immediately after the XML header, the *document type declaration* follows, commonly referred to as the DOCTYPE:

```
<!DOCTYPE address [
```

The DOCTYPE declaration has an exclamation mark (!) at the start of the element name. The DOCTYPE informs the parser that a DTD is associated with this XML document.

DTD Body: The DOCTYPE declaration is followed by body of the DTD, where we declare elements, attributes, entities, and notations:

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

Several elements are declared here that make up the vocabulary of the <name> document. <!ELEMENT name (#PCDATA)> defines the element *name* to be of type "#PCDATA". Here #PCDATA means parse-able text data.

End Declaration: Finally, the declaration section of the DTD is closed using a closing bracket and a closing angle bracket (]>). This effectively ends the definition, and thereafter, the XML document follows immediately.

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.

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.

Syntax:

```
<!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>Tanmay Patil</name>
  <company>TutorialsPoint</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)>
```

Types:

We can refer to an external DTD by using either **system identifiers** or **public identifiers**.

1. **System Identifiers:**

A system identifier enables us to specify the location of an external file containing DTD declarations. Syntax is as follows:

```
<!DOCTYPE name SYSTEM "address.dtd" [...]>
```

As we can see, it contains keyword SYSTEM and a URI reference pointing to the location of the document.

2. **Public Identifiers:**

Public identifiers provide a mechanism to locate DTD resources and are written as below:

```
<!DOCTYPE name PUBLIC "-//Beginning XML//DTD Address Example//EN">
```

As we can see, it begins with keyword PUBLIC, followed by a specialized identifier. Public identifiers are used to identify an entry in a catalog. Public identifiers can follow any format, however, a commonly used format is called *Formal Public Identifiers, or FPIs.*

LIMITATION OF DTD:

- DTDs do not have built-in datatypes.
- DTDs do not support user-derived datatypes.
- DTDs allow only limited control over cardinality (the number of occurrences of an element within its parent).
- DTDs do not support Namespaces or any simple way of reusing or importing other schemas.

DOCUMENT TYPE DEFINITION (DTD):

A specification for a SGML or HTML document that specifies structural elements and markup definitions that can be used to create documents that describe content. The DTD can put constraints on the occurrence and content of elements and other details of the document structure

SCHEMA:

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.

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 legitimate format that an XML document can take.

Definition Types:

We can define XML schema elements in following ways:

1. **Simple Type:**

Simple type element is used only in the context of the text. Some of predefined simple types are: xs:integer, xs:boolean, xs:string, xs:date. For example:

```
<xs:element name="phone_number" type="xs:int" />
```

2. **Complex Type:**

A complex type is a container for other element definitions. This allows us to specify which child elements an element can contain and to provide some structure within your XML documents. For example:

```
<xs:element name="Address">
  <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>
```

In the above example, *Address* element consists of child elements. This is a container for other <xs:element> definitions, that allows to build a simple hierarchy of elements in the XML document.

3. **Global Types:**

With global type, we can define a single type in our document, which can be used by all other references. For example, suppose we want to generalize the *person* and *company* for different addresses of the company. In such case, we can define a general type as below:

```
<xs:element name="AddressType">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="name" type="xs:string" />
```

```
            <xs:element name="company" type="xs:string" />
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

Now let us use this type in our example as below:

```
<xs:element name="Address1">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="address" type="AddressType" />
          <xs:element name="phone1" type="xs:int" />
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="Address2">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="address" type="AddressType" />
          <xs:element name="phone2" type="xs:int" />
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Instead of having to define the name and the company twice (once for *Address1* and once for *Address2*), we now have a single definition. This makes maintenance simpler, i.e., if you decide to add "Postcode" elements to the address, you need to add them at just one place.

Attributes:

Attributes in XSD provide extra information within an element. Attributes have *name* and *type* property as shown below:

```
<xs:attribute name="x" type="y"/>
```

XSL (Extensible Style Sheet Language):

XSL is a language for expressing style sheets. An XSL style sheet is, like with CSS, a file that describes how to display an XML document of a given type. XSL shares the functionality and is compatible with CSS2 (although it uses a different syntax). It also adds:

- A transformation language for XML documents: XSLT. Originally intended to perform complex styling operations, like the generation of tables of contents and indexes, it is now used as a general purpose XML processing language. XSLT is thus widely used for purposes other than XSL, like generating HTML web pages from XML data.

- Advanced styling features, expressed by an XML document type which defines a set of elements called Formatting Objects, and attributes (in part borrowed from CSS2 properties and adding more complex ones.

ASSOCIATING CSS STYLE SHEET WITH XML:

- XML has emerged as a "universal" data format in a variety of application areas.
- Style sheets are an essential step in XML deployment to define the presentation of XML documents.
- The association consists of inserting the XML processing instruction at the top of the document, before the root element of the XML document and after the XML prolog.
- The processing instruction has two required attributes type and href which respectively specify the type of stylesheet (Internet Media Type text/css) and its address (path).
  <?xml-stylesheet type="text/css" href="foo.css"?>

Example:

First create a file .xml

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/css" href="style.css"?>
<PU>
    <Centers>
    <Center>
            <Code>001</Code>
            <Location id="Pkr">Pokhara</Location>
            <Phone>344333</Phone>
            <Email>pkr@webcom.com</Email>
    </Center>
    <Center>
            <Code>003</Code>
            <Location id="Ktm">Kathmandu</Location>
            <Phone>3444433</Phone>
            <Email>ktm@webcom.com</Email>
    </Center>
    <Center>
            <Code>004</Code>
            <Location id="Tnu">Tanahu</Location>
            <Phone>342223</Phone>
            <Email>tanahu@webcom.com</Email>
    </Center>
    </Centers>
</PU>
```

Then create style.css file:

```
PU {
```

```
    font-family: verdana;
    color:brown;
  }
Center {
    background:yellow;
    display:block;
    margin:5px;
  }
Location {
    font-size:large;
    display:block;
  }
Email {
    font-size:small;
    display:block;
  }
```

XML PROCESSORS:

1. DOM:

The Document Object Model (DOM) is an application programming interface (API) for HTML and XML documents. It defines the logical structure of documents and the way a document is accessed and manipulated.

XML DOM is a standard object model for XML. XML documents have a hierarchy of informational units called nodes; DOM is a standard programming interface of describing those nodes and the relationships between them.

As XML DOM also provides an API that allows a developer to add, edit, move or remove nodes at any point on the tree in order to create an application.

Example:

The following example (sample.htm) parses an XML document ("address.xml") into an XML DOM object and then extracts some information from it with JavaScript:

```
<!DOCTYPE html>
<html>
  <body>
    <h1>TutorialsPoint DOM example </h1>
    <div>
      <b>Name:</b> <span id="name"></span><br>
      <b>Company:</b> <span id="company"></span><br>
      <b>Phone:</b> <span id="phone"></span>
    </div>
    <script>
```

```
        if (window.XMLHttpRequest)
        {// code for IE7+, Firefox, Chrome, Opera, Safari
          xmlhttp = new XMLHttpRequest();
        }
        else
        {// code for IE6, IE5
          xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");
        }
        xmlhttp.open("GET","/xml/address.xml",false);
        xmlhttp.send();
        xmlDoc=xmlhttp.responseXML;

        document.getElementById("name").innerHTML=
        xmlDoc.getElementsByTagName("name")[0].childNodes[0].nodeValue;
        document.getElementById("company").innerHTML=
        xmlDoc.getElementsByTagName("company")[0].childNodes[0].nodeValue;
        document.getElementById("phone").innerHTML=
        xmlDoc.getElementsByTagName("phone")[0].childNodes[0].nodeValue;
      </script>
    </body>
  </html>
```

Contents of **address.xml** are as below:

```
<?xml version="1.0"?>
<contact-info>
  <name>Tanmay Patil</name>
  <company>TutorialsPoint</company>
  <phone>(011) 123-4567</phone>
</contact-info>
```

Now let us keep these two files **sample.htm** and **address.xml** in the same directory **/xml** and execute the **sample.htm** file by opening it in any browser.

Advantages

- XML DOM is language and platform independent.
- XML DOM is travesible - 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.

Disadvantages

- 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.

2. SAX (Simple API for XML):

SAX (Simple API for XML) is an event-driven online algorithm for parsing XML documents, with an API developed by the XML-DEV mailing list. 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.

A SAX parser only needs to report each parsing event as it happens, and normally discards almost all of that information once reported (it does, however, keep some things, for example a list of all elements that have not been closed yet, in order to catch later errors such as end-tags in the wrong order).

Advantages

- It is simple and memory efficient.
- It is very fast and works for huge documents.

Disadvantages

- It is event-based so its API is less intuitive.
- Clients never know the full information because the data is broken into pieces.

Chapter – 6

Introduction to JQuery

Introduction:

jQuery is a fast and concise JavaScript Library created by John Resig in 2006 with a nice motto **Write less, do more**. jQuery simplifies HTML document traversing, event handling, animating, and Ajax interactions for rapid web development. jQuery is a JavaScript toolkit designed to simplify various tasks by writing less code.

Here is the list of important core features supported by jQuery:

- ☑ **DOM manipulation**: The jQuery made it easy to select DOM elements, traverse them and modifying their content by using cross-browser open source selector engine called **Sizzle**.
- ☑ **Event handling**: The jQuery offers an elegant way to capture a wide variety of events, such as a user clicking on a link, without the need to clutter the HTML code itself with event handlers.
- ☑ **AJAX Support**: The jQuery helps you a lot to develop a responsive and feature-rich site using AJAX technology.
- ☑ **Animations**: The jQuery comes with plenty of built-in animation effects which you can use in your websites.
- ☑ **Lightweight**: The jQuery is very lightweight library - about 19KB in size (Minified and gzipped).
- ☑ **Cross Browser Support**: The jQuery has cross-browser support, and works well in IE 6.0+, FF 2.0+, Safari 3.0+, Chrome and Opera 9.0+
- ☑ **Latest Technology**: The jQuery supports CSS3 selectors and basic XPath syntax.

How to use JQuery?

There are two ways to use jQuery.

- ☑ **Local Installation**: We can download jQuery library on our local machine and include it in our HTML code.
- ☑ **CDN Based Version**: We can include jQuery library into our HTML code directly from Content Delivery Network (CDN).

Why JQuery?

- ☑ There are lots of other JavaScript frameworks out there, but jQuery seems to be the most popular, and also the most extendable.
- ☑ Many of the biggest companies on the Web use jQuery, such as: Google, Microsoft, IBM, Netflix
- ☑ The jQuery team knows all about cross-browser issues, and they have written this knowledge into the jQuery library. jQuery will run exactly the same in all major browsers, including Internet Explorer 6.

Syntax:

The jQuery syntax is tailor-made for selecting HTML elements and performing some action on the element(s).

$(selector).action()

A $ sign to define/access jQuery
A (selector) to "query (or find)" HTML elements
A jQuery action() to be performed on the element(s)

Examples:

$(this).hide() - hides the current element.
$("p").hide() - hides all <p> elements.
$(".test").hide() - hides all elements with class="test".
$("#test").hide() - hides the element with id="test".

Example:

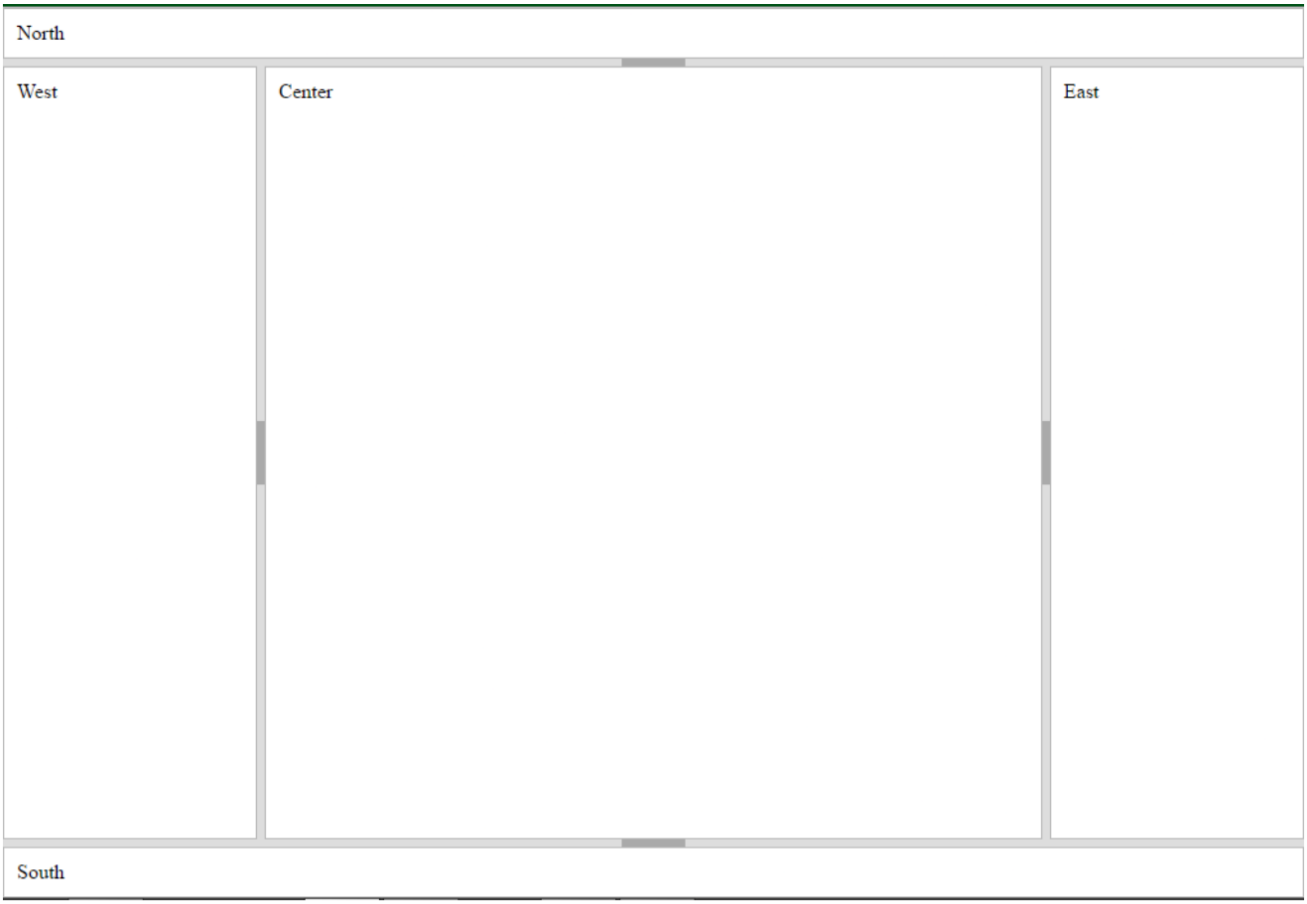
```
<!DOCTYPE html>
<html>
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/1.12.2/jquery.min.js"></script>
<script>
$(document).ready(function(){
  $("p").click(function(){
    $(this).hide();
  });
});
</script>
</head>
<body>

<p>If you click on me, I will disappear.</p>
<p>Click me away!</p>
<p>Click me too!</p>

</body>
</html>
```

PAGE LAYOUT USING JQUERY:

We can directly download the page layout and apply in out code. Demo page layout using jQuery is shown below:

```
<!DOCTYPE html>
<html>
<head>
```

```
    <title>Basic Layout Example</title>

    <script type="text/javascript" src="js/jquery.js"></script>
    <script type="text/javascript" src="../source/stable/jquery.layout.js"></script>

    <script type="text/javascript">
    $(document).ready(function(){
            $("body").layout({ applyDemoStyles: true });
    });
    </script>

</head>
<body>

<div class="ui-layout-center">Center</div>
<div class="ui-layout-north">North</div>
<div class="ui-layout-south">South</div>
<div class="ui-layout-east">East</div>
<div class="ui-layout-west">West</div>

</body>
</html>
```



Chapter – 7

Page Layout and Design Issues

Page Layout:

In graphic design, page layout is the process of placing and arranging text, images and graphics on a software page to produce documents such as newsletters, brochures and books or to attract readership to a website. The goal is to produce eye-catching pages that grab the attention of the reader. Often this involves using a set of design rules and specific colors, the specific style of a publication or website to adhere to a visual brand.

1. Understanding Site Audience:

A website audience is a group of users visiting our site that we have cookied with our tracking tag. An audience might include everyone who has visited our site, just users who have registered, or users who have abandoned their shopping carts.

Essentially, it is a group of people we have identified that they had visited our site. We can choose to create multiple website audiences. Some users may overlap into multiple audiences depending on our settings we created.

2. Page Size and Screen Resolution:

Our pages should work at any resolution. Jakob Nielsen is on record as recommending:

a. **Optimize for 1024×768,** which was the most widely used screen size for a long time. Of course, the general guideline is to optimize for our target audience's most common resolution, so size will change in the future. It might even be a different size now, if, say, we are designing an intranet for a company that gives all employees new big monitors.
b. **Do not design solely for a specific monitor size** unless we have been specifically commissioned to do so because screen sizes vary among users. Window size variability is even greater, since users don't always maximize their browsers (especially if they have large screens).
c. **Use a liquid layout that stretches to the current user's window size** (that is, avoid frozen layouts that are always the same size).

3. Designing Pages:

Web design itself refers to the process of creating a web page's appearance and to the choice of a right color scheme, page layout, fonts and more. Every single web page in a website has different content, but all the pages are using a similar graphic design. Often sites will use website templates, which contain all the basic elements of web design, the website's CSS style, buttons, backgrounds, borders and various graphic elements like hover images, bullets and header banners. When the website template is applied to the website, all the pages assume its appearance, using the same styles, background and other graphical elements.

But most of the pages in a website have their own design elements. This is needed because every page has to present a different content, and the basic website style is not fully applicable for all the web pages. For example, designing a contact form will sometimes require a custom layout

and design elements with which to handle the form's fields, buttons, drop-down menus, etc. All these styles, layouts, images, etc. are often not applicable for the other website pages. So, the contact page uses the basic website template for the menus, backgrounds, header images, etc. but also has its own web page design, including the contact form's design itself and all the other elements specific for that page.

4. Coding Our Design:

Coding is basically the computer language used to develop apps, websites and software. Without it, we'd have none of the major technology we've come to rely on such as Facebook, our smartphones, the browser we choose to view our favorite blogs or even the blogs themselves. It all runs on code.

5. Developing for Mobile Devices:

Various devices have different capabilities and screen resolutions, which makes it difficult to build a single page layout suitable for all devices. Individual page templates allow to define alternative page layouts for specific device profiles.

Pages based on these templates automatically use the appropriate layout according to the device profile detected for each visitor.

Design Issues:

The designing industry is highly competitive, and to get ahead in the race, designers need to keenly focus on being technical as well as strategic. Here are a few common issues that designers have to face during web design and development.

1. Website Accessibility:

The Web is basically designed to work for all people, irrespective of the culture, language, location, or physical or mental ability. However, one of the major challenges a web designer faces is to enhance the accessibility of websites. A good designer should ensure that the website is not only accessible across the world but also its various features are fully functional as well.

2. Compatibility With Browsers:

With the introduction of different browsers, designers are constantly facing the challenge of building a website which is compatible with almost all the major browsers. After designing a website, it should be tested on all browsers to ensure that the website is completely functional.

3. Navigational Structure:

Navigational structure is one of the vital aspects of any website, as the usability of the website is based on an excellent navigational structure. Hence, in order to avoid any such issues, designers have to ensure that they provide a proper navigational structure to the users.

4. Positioning Of Content:

Another prominent aspect of a website is that the users should find it readable. While designing the structure of the website, the designer should place the content in such a manner that it enhances easy reading. In addition, use suitable colors when it comes to font.

5. Challenges In Creating A Responsive Website

The process of creating a responsive website is a major challenge for designers as it involves a wide array of devices, code frameworks, scripts, and of course, the constant need to work in an innovative way with clients to effectively manage the process.

6. Typography:

Typography is the art and technique of arranging type to make written language legible, readable, and appealing when displayed. The arrangement of type involves selecting typefaces, point sizes, line lengths, line-spacing (leading), and letter-spacing (tracking), and adjusting the space between pairs of letters (kerning). The term typography is also applied to the style, arrangement, and appearance of the letters, numbers, and symbols created by the process. Type design is a closely related craft, sometimes considered part of typography; most typographers do not design typefaces, and some type designers do not consider themselves typographers. Typography also may be used as a decorative device, unrelated to communication of information.

Steps to follow for Website Design:

1. Requirement Analysis:

Before designing any website first of all we should analysis all the necessary thing that are required while creating effective website.

a. Customer Requirement Analysis:

If we are designing any website for our customer, then we must collect the entire requirement from our customer, what type of website they wanted. Like; they wanted *a highly graphics, a highly animated, or database oriented or e-commerce enabled or simpler one.*

b. Designer Requirement Analysis:

If we are designing a website then we should analysis all the required tools, data, content etc. Before starting any website to design we must be aware of what we are going to design. Also sure that we have all tools and data that customer wanted on website. And we have to estimate the total time to finalize the website.

c. Technical Requirement Analysis:

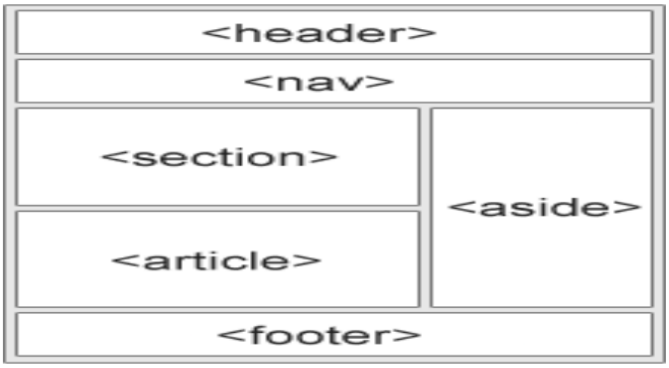
Before starting design of any site, analyze the technical part of the website. Where we should aware of the web space, Web server platform, Web Server, Database server, Bandwidth assigned for site, domain name of website.

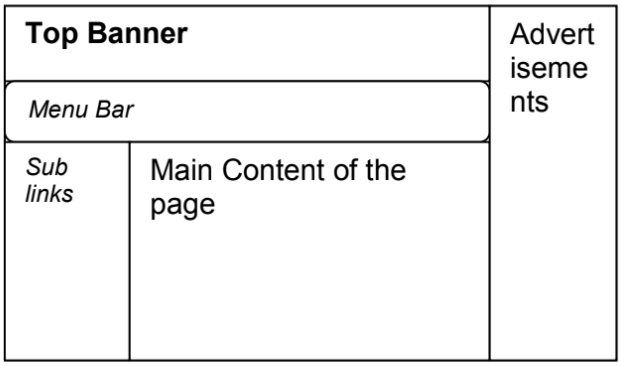
2. Website Design:

After completing analyzing part we first design the website. In this section we should design the whole website module by module.

a. Web Layout Design:

Before starting the website design we should design the Layout of Front Page and Inner page. Also design any extra other special pages on our website. We must identify the size of each part of our layout, how much pixel we want to specify.





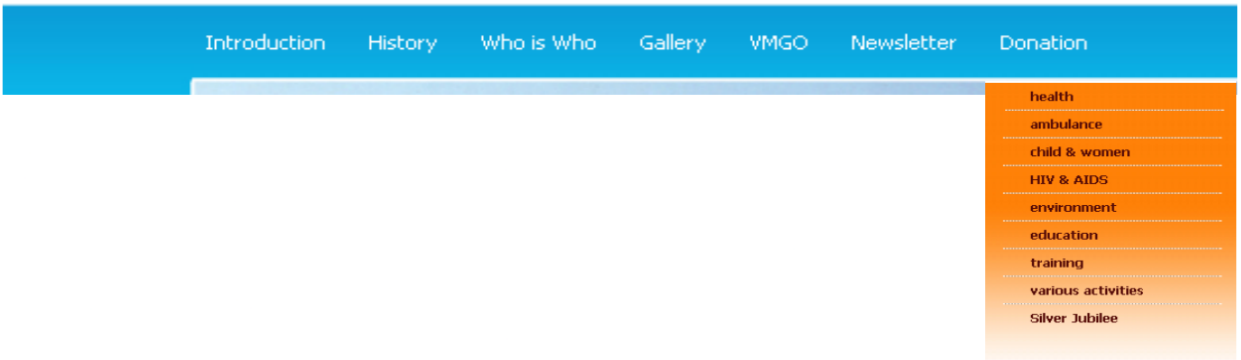
- <header> - Defines a header for a document or a section
- <nav> - Defines a container for navigation links
- <section> - Defines a section in a document
- <article> - Defines an independent self-contained article
- <aside> - Defines content aside from the content (like a sidebar)
- <footer> - Defines a footer for a document or a section
- <details> - Defines additional details
- <summary> - Defines a heading for the <details> element

b. Banner Design:

Banner is the main attraction of a website so we should design attractive web banner for our website. To design a banner we can use any one of the graphic editing software. Like: Adobe Photoshop, Corel Draw, etc.

c. Menus Design:

After designing the Banner we must design the menu/links that we need on our website. Menu/link can be created simple text link, image link or JavaScript enabled dynamic animated menus.



d. Content of Page:

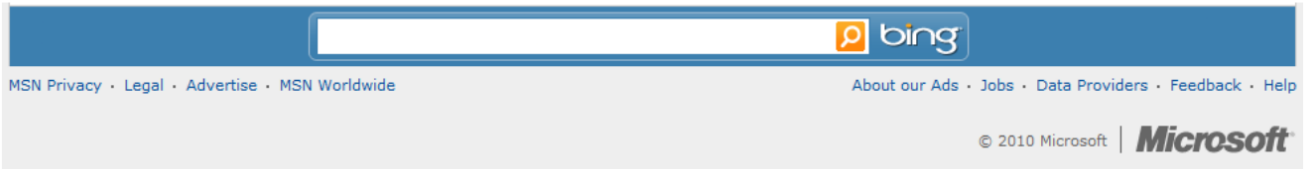
Now, when we have designed layout, banner and menus then we should start to manage the content of each page of our website. Collect all the content which we want to include into the pages from the primary or secondary sources.

e. Current Events:

If our website is of an organization then we should include the all recent events and must display on highlighted form, where we can include GIF images, JavaScript Animated, Video Files or Flash Animated.

f. Footer:

Footer is the information about the organization/person that is holding the ownership of website. Where we can include the full address and contact about an organization.



3. Resource Collection:

Resource collection is the major part of our website design, where we have to collect all the content that we want to include into website like; information about the organization, pictures related to different event of an organization, graphics we want to place on our website and other media files. But be sure that we must have all the rights to use that resources otherwise we have to take permissions to use that resources from the related source.

4. Finalize/managing the site:

After creating most of pages of our website, we have to finalize all pages, links of our site, where place all the content in a proper manner, where we can categorize the content and place in the different web folders. Like; we can place all web pages of one category into one web folder and images used on that category into another image folder. To manage the website in a proper manner we can use third party tools like; Microsoft FrontPage, Adobe GoLive, Macromedia Dreamweaver, etc.

5. Testing:

After finalizing the web designing we should test our website for error free, which makes our website better.

- Layout: Check website on browser is it finalized according to the Layout.
- Links: Check all the links whether it is working or not placed on website.

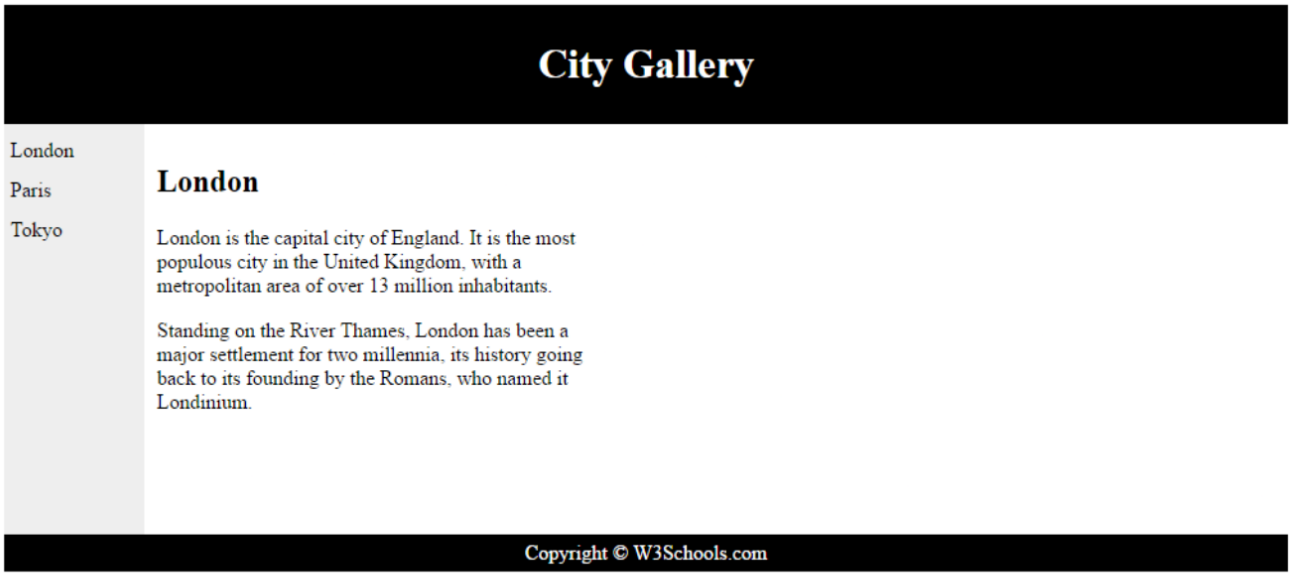
6. Upload to the server:

After completing the site upload all webpages, media files on web server. We can use various upload manager to manage our webserver. Like: Macromedia Dreamweaver's file manager, FileZila, etc.

Example:

```
<!DOCTYPE html>
<html><head>
<style>
#header {
   background-color:black;
   color:white;
   text-align:center;
   padding:5px;
}
#nav {
   line-height:30px;
   background-color:#eeeeee;
   height:300px;
   width:100px;
   float:left;
   padding:5px;
}
#section {
```

```
    width:350px;
    float:left;
    padding:10px;
}
#footer {
    background-color:black;
    color:white;
    clear:both;
    text-align:center;
   padding:5px;
}
</style>
</head>
<body>

<div id="header">
<h1>City Gallery</h1>
</div>
<div id="nav">
London<br>
Paris<br>
Tokyo
</div>

<div id="section">
<h2>London</h2>
<p>London is the capital city of England. It is the most populous city in the United Kingdom,
with a metropolitan area of over 13 million inhabitants.</p>
<p>Standing on the River Thames, London has been a major settlement for two millennia, its history going back to its founding by the Romans, who named it Londinium.</p>
</div>

<div id="footer">
Copyright © W3Schools.com
</div>
</body>
</html>
```



HTML RESPONSIVE WEB DESIGN:

Responsive Web Design makes our web page look good on all devices (desktops, tablets, and phones). Responsive Web Design is about using CSS and HTML to resize, hide, shrink, enlarge, or move the content to make it look good on any screen: