



Yashwantrao
Chavan
Maharashtra
Open University

CMP508

Web Technologies

WEB Technologies

Yashwantrao Chavan Maharashtra Open University

Dnyangangotri, Near Gangapur Dam

Nashik-422222

Yashwantrao Chavan Maharashtra Open University

Vice-Chancellor: Dr. R. Krishnakumar

SCHOOL OF COMPUTER SCIENCE: SCHOOL COUNCIL

Dr. Ramchandra Tiwari Director School of Computer Science Y.C.M.Open University Nashik	Shri. Pramod Khandare Assistant Professor School of Computer Science Y.C.M.Open University, Nashik	Prof. M.S. Karyakate Associate Professor Computer Department VIIT, Pune
Dr. Manoj Killedar Director, School of Architecture, Science & Technology, Y.C. M. Open University, Nashik	Shri. Surendra Patole Assistant Professor School of Commerce & Management Y.C.M. Open University,Nashik	Prof. M.N. Shelar H.O.D., Computer Department K.K. Wagh College Pimpalgaon (B)
Dr. R.V. Vadnere Director School of Continuing Education Y.C.M.Open University, Nashik	Shri. Madhav Palshikar Associate Professor School of Computer Science Y.C.M. Open University, Nashik	Shri. Mahesh Paradkar IBM Pune
Prof. S.S. Sane Head of Department Computer Department K.K.Wagh College of Engineering Nashik	Dr. Bharati Gawali Associate Professor Dept. Of Computer Science Dr. Babasaheb Ambedkar Marathwada University, Aurangabad	Mrs. Shubhangi Desle Assistant Professor Student Services Division Y.C.M. Open University, Nashik

Writer/s**Editor****Co-ordinator**

Dr. Pramod Khandare
Assistant Professor
School of Computer Science
Y.C.M.Open University,
Nashik 422 222

Production

WEB TECHNOLOGIES CMP508

1. Introduction to the Web 3 Counseling Sessions

History and Evolution
Web development cycle Web publishing
Web contents
Dynamic Web contents

2. Languages and technologies for browsers 3 Counseling Sessions

HTML, DHTML, XHTML, ASP, JavaScript Features and Applications

3. Introduction to HTML 3 Counseling Sessions

HTML Fundamentals HTML Browsers
HTML tags, Elements and Attributes Structure of HTML code
 Head
 Body
Lists
 Ordered List
 Unordered List
 Definition List
 Nesting List
Block Level Tags
 Block formatting, Heading, Paragraph, Comments, Text alignment,
Font size
Text Level Tags
 Bold, Italic, Underlined, Strikethrough, Subscript, superscript
Inserting graphics, Scaling images Frameset
Forms
An introduction to DHTML, DOM

4. Cascading Style Sheets 3 Counseling Sessions

The usefulness of style sheets Creating style sheets
Common tasks with CSS Font Family
 Font Metrics
 Units
Properties
Classes and Pseudo classes CSS tags

5. Introduction to Client side Scripting 2 Counseling Sessions

What is Scripting Language ?
Client side and server side scripting

Types of scripting languages

6. JavaScript

5 Counseling Sessions

Introduction

Operators, Assignments and Comparisons, Reserved words Starting with JavaScript

Writing first JavaScript program

Putting Comments Functions

Statements in JavaScript Working with objects

Object Types and Object Instantiation

Date object, Math Object, String object, Event object, Frame object,

Screen object

Handling Events

Event handling attributes

Window Events, Form Events

Event Object

Event Simulation

7. XML

4Counseling Sessions

Introduction to XML,

Anatomy of an XML document

Creating XML Documents,

Creating XML DTDs, XML Schemas, XSL

8. Website Design Concepts

4 Counseling Sessions

How the website should be

Basic rules of Web Page design

Types of Website

Reference Books :

Web Technologies Achyut S. Godbole, AtulKahate Tata McGraw Hill

Web Tech. & Design C. Xavier New Age

Multimedia & Web Technology – Ramesh Bangia

HTML : The complete reference – Thomas A. Powel

JavaScript Bible – Danny Goodman

1

UNIT 1 INTRODUCTION TO WEB TECHNOLOGY

Unit Structure

History and Evolution
Web development cycle
Web publishing
Web contents
Dynamic Web contents

HISTORY AND EVOLUTION OF WEB

The *World Wide Web* (w.w.w) allows computer users to locate and view multimedia-based documents (i.e., documents with text, graphics, animations, audios or videos) on almost any subject. In 1990, *Tim Berners-Lee* of CERN (the European Laboratory for Particle Physics) developed the World Wide Web and several communication protocols that form the backbone of the Web.

The Internet's origins

In the late 1960s, one of the authors (HMD) was a graduate student at MIT. His research at MIT's Project Mac (now the Laboratory for Computer Science—the home of the World Wide Web Consortium) was funded by ARPA—the Advanced Research Projects Agency of the Department of Defense.

The protocols for communicating over the ARPAnet became known as *TCP—the Transmission Control Protocol*. TCP ensured that messages were properly routed from sender to receiver and that those messages arrived intact.

As the Internet evolved, organizations worldwide were implementing their own networks for both intra-organization (i.e., within the organization) and inter-organization (i.e., between organizations) communications. A wide variety of networking hardware and software appeared. One challenge was to get these different networks to communicate. ARPA accomplished this with the development of *IP—the Internetworking Protocol*, truly creating a “network of networks,” the current architecture of the Internet. The combined set of protocols is now commonly called *TCP/IP*.

Initially, Internet use was limited to universities and research institutions; then the military began using the Internet. Eventually, the government decided to allow access to the Internet for commercial purposes.

The formation of the W3C

In October 1994, Tim Berners-Lee founded an organization—called the *World Wide Web Consortium (W3C)*—devoted to developing nonproprietary, interoperable technologies for the World Wide Web. One of the W3C's primary goals is to make the Web universally accessible—regardless of disability, language or culture.

The W3C is also a standardization organization. Web technologies standardized by the W3C are called *Recommendations*. W3C Recommendations include the ExtensibleHyper-Text Markup Language (XHTML), Cascading Style Sheets (CSS), Hypertext Markup Language (HTML; now considered a “legacy” technology) and the Extensible Markup Language (XML).

A recommendation is not an actual software product, but a document that specifies a technology's role, syntax, rules, etc. Before becoming a W3C

The W3C homepage (www.w3.org) provides extensive resources on Internet and Web technologies. For each Internet technology with which the W3C is involved, the site provides a description of the technology and its benefits to Web designers, the history of the technology and the future goals of the W3C in developing the technology. This site also describes W3C's goals. The goals of the W3C are divided into the following categories:

User Interface Domain, Technology and Society Domain, Architecture Domain and Web Accessibility Initiatives.

Evolution of www:

Web 1.0 – The World Wide Web (1990 – 2000)

- Remain limited mostly to static websites.
- Mostly publishing / Brochure-ware. Limited to reading only for majority.
- Proprietary and closed access.
- Corporations mostly, no communities.
- HTTP, HTML

Web 2.0 – The Social Web (2000 – 2010)

- Publishing as well as Participation
- Social Media, Blogging, Wikis
- RSS – Syndicate site contents.
- Rich User Experience
- Tagging
- Keyword Search
- AJAX, JavaScript Frameworks (jQuery, Dojo, YUI, Ext Jsetc), XML, JSON

Web 3.0 – The Semantic Web (2010 – onward)

- Mostly Drag n Drop
- Highly mobile oriented
- Widgets
- Micro blogging
- Cloud and Grid Computing
- Open ID
- Semantic Search, Semantic Techniques like RDF, SWRL, OWL etc.

WEB DEVELOPMENT CYCLE

There are numerous steps in the web site design and development process. From gathering initial information, to the creation of your web site, and finally to maintenance to keep your web site up to date and current.

The basic steps of Web Development Cycle are:

- Information Gathering
- Planning
- Design
- Development
- Testing and Delivery
- Maintenance Phase

Phase One: Information Gathering

Just like Software Development ,the first step in designing a successful web site is to gather information. Many things need to be taken into consideration when the look and feel of your site is created.

The Client's business requirements and goals are understood

It is important that the web designer start off by asking a lot of questions to help them understand the business and the needs in a web site.

Certain things to consider are:

Purpose

What is the purpose of the site? Do you want to provide information, promote a service, sell a product... ?

Goals

What do you hope to accomplish by building this web site?

Target Audience

Is there a specific group of people that will help you reach your goals? It is helpful to picture the "ideal" person you want to visit your web site. Consider their demographics – this will help determine the best design style for the site.

Content

What kind of information will the target audience be looking for on the site? Are they looking for specific information, a particular product or service, online ordering...?

Phase Two: Planning

Using the information gathered from phase one, it is time to put together a plan for the web site. This is the point where a site map is developed.

The site map is a list of all main topic areas of the site, as well as sub-topics, if applicable. This serves as a guide as to what content will be on the site, and is essential

to developing a consistent, easy to understand navigational system. The end-user of the web site must be kept in mind when designing the site. A good user interface creates an easy to navigate web site.

During the planning phase, the web designer decides what technologies should be implemented. Elements such as interactive forms, e-commerce, flash, etc. are discussed when planning the web site.

Phase Three: Design

In this phase the look and feel of the web site is determined.

Target audience is one of the key factors taken into consideration. As part of the design phase, it is also important to incorporate elements such as the company logo or colors to help strengthen the identity of the company on the web site.

The web designer will create one or more prototype designs for the web site. The client selects what suits his/her needs the best.

In this phase, communication between the client and the web designer is very important to ensure that the final web site will be according to the client's requirements.

After the design is finalized the development begins.

Phase Four: Development

The developmental stage is the point where the web site itself is created. At this time, the web designer will take all of the individual graphic elements from the prototype and use them to create the actual, functional site.

This is typically done by first developing the home page, followed by a "shell" for the interior pages. The shell serves as a template for the content pages of your site, as it contains the main navigational structure for the web site. Once the shell has been created, the designer will take the client's content and distribute it throughout the site, in the appropriate areas.

Elements such as interactive contact forms, flash animations or ecommerce shopping carts are implemented and made functional during this phase, as well.

This entire time, the designer should continue to make the in-progress web site available to the client for viewing, so that he/she can suggest any additional changes or corrections.

On the technical front, a successful web site requires an understanding of front-end web development. This involves writing valid XHTML / CSS code that complies to current web standards, maximizing functionality, as well as accessibility for as large an audience as possible.

This is tested in the next phase...and Delivered

The final web site is tested. The testing includes complete functionality of forms or other scripts, as well compatibility issues (viewing differences between different web

browsers), ensuring that the web site is optimized to be viewed properly in the most recent browser versions.

A good web designer is one who is well versed in current standards for web site design and development. The basic technologies currently used are XHTML and CSS (Cascading Style Sheets). As part of testing, the designer should check to be sure that all of the code written for the web site validates. Valid code means that the site meets the current web development standards.

After the web site is finalized an FTP (File Transfer Protocol) program is used to upload the web site files to the server. Most web designers offer domain name registration and web hosting services as well. Once these accounts have been setup, and your web site uploaded to the server, the site should be put through one last run-through. This is just precautionary, to confirm that all files have been uploaded correctly, and that the site continues to be fully functional.

This marks the official launch of your site, as it is now viewable to the public. The development of your web site is not necessarily over, though. One way to bring repeat visitors to your site is to offer new content or products on a regular basis. Many designers offer maintenance packages at reduced rates, based on how often you anticipate making changes or additions to the web site.

If the client prefers to be more hands on, and update their own content, there is something called a CMS (Content Management System) that can be implemented to the web site. This is something that would be decided upon during the Planning stage. With a CMS, the designer will utilize online software to develop a database driven site for you.

A web site driven by a CMS gives the client the ability to edit the content areas of the web site on his own.

Other maintenance type items include SEO (Search Engine Optimization) and SES (Search Engine Submission). This is the optimization of the web site with elements such as title, description and keyword tags which help the web site achieve higher rankings in the search engines. The previously mentioned code validation is something that plays a vital role in SEO, as well.

THE PROCESS OF WEB PUBLISHING

Web publishing is the process of publishing original content on the Internet.

The process includes building and uploading websites, updating the associated webpages, and posting content to these webpages online. Web publishing comprises of personal, business, and community websites in addition to e-books and blogs.

The content meant for web publishing can include text, videos, digital images, artwork, and other forms of media.

Publishers must possess a web server, a web publishing software, and an Internet connection to carry out web publishing.

Web publishing is also known as online publishing.

WEB CONTENT

Web content is the textual, visual or aural content.. It may include, among other things: text, images, sounds, videos and animations. We categorize websites according to the content a website contains.

There are two basic kinds of web content:

Text: Text is simple. It is added on the webpage as text blocks or within images. The best written content is unique textual web content that is free from plagiarism. Web content added as text can also include good internal links that help readers gain access to more information.

- **Multimedia:** Another kind of web content is multimedia. Simply put, multimedia refers to any content which is not text; some examples include:
 - **Animations:** Animations can be added with the help of Flash, Ajax, GIF images as well as other animation tools.
 - **Images:** Images are considered the most popular option to incorporate multimedia to websites. Clip art, photos, or even drawings can be created by means of a scanner or a graphics editor. It is recommended to optimize the images so that the users can download them quickly.
 - **Audio:** Different types of audio files can be added as part of the web content so as to increase the desirability of the website.
 - **Video:** It is the most popular multimedia contents; however, when adding video files, the publishers should make sure that they efficiently on various browsers.

Web content management (WCM) is essential to run a website successfully. To manage web content, publishers should organize content in line with the requirements of the audience.

The page concept

Web content is dominated by the "page" concept.

A web site is made up of a number of web pages. Usually most of the web site has a home page.

A home page is a webpage that serves as the starting point of website. It is the default webpage that loads when you visit a web address that only contains a domain name. For example, visiting <https://in.yahoo.com/> will display the Yahoo home page.

There is no standard home page layout, but most home pages include a navigation bar that provides links to different sections within the website. Other common elements found on a home page include a search bar, information about the website, and recent news or updates. Some websites include information that changes every day.

STATIC AND DYNAMIC WEB CONTENT

A website can be of two types:

- Static Website
- Dynamic Website

Static website

Static website is the basic type of website that is easy to create. You don't need web programming and database design to create a static website. Its web pages are coded in HTML.

The codes are fixed for each page so the information contained in the page does not change and it looks like a printed page.

Advantages and disadvantages

Advantages

- No programming skills are required to create a static page.
- Inherently publicly cacheable (i.e. a cached copy can be shown to anyone).
- No particular hosting requirements are necessary.
- Can be viewed directly by a web browser without needing a web server or application server, for example directly from a CD-ROM or USB Drive.

Disadvantages

- Any personalization or interactivity has to run client-side (i.e. in the browser), which is restricting.
- Maintaining large numbers of static pages as files can be impractical without automated tools.

Application areas of Static Website:

Need of Static web pages arise in the following cases.

- Changes to web content is infrequent
- List of products / services offered is limited
- Simple e-mail based ordering system should suffice ,No advanced online ordering facility is required,
- Features like order tracking, verifying availability of stock, online credit card transactions, are not needed
- Web site not required to be connected to back-end system.

Dynamic website

Dynamic website is a collection of dynamic web pages whose content changes dynamically. It accesses content from a database or Content Management System (CMS). Therefore, when you alter or update the content of the database, the content of the website is also altered or updated.

Dynamic website uses client-side scripting or server-side scripting, or both to generate dynamic content.

Client side scripting generates content at the client computer on the basis of user input. The web browser downloads the web page from the server and processes the code within the page to render information to the user.

In server side scripting, the software runs on the server and processing is completed in the server then plain pages are sent to the user.

Application areas of Dynamic Website

Dynamic web page is required when following necessities arise:

Need to change main pages more frequently to encourage clients to return to site.

Long list of products / services offered that are also subject to up gradation

Introducing sales promotion schemes from time to time

Need for more sophisticated ordering system with a wide variety of functions

Tracking and offering personalized services to clients.

Facility to connect Web site to the existing back-end system

Static vs Dynamic website

Static Website	Dynamic Website
Prebuilt content is same every time the page is loaded.	Content is generated quickly and changes regularly.
It uses the HTML code for developing a website.	It uses the server side languages such as PHP, SERVLET, JSP, and ASP.NET etc. for developing a website.
It sends exactly the same response for every request.	It may generate different HTML for each of the request.
The content is only changes when someone publishes and updates the file (sends it to the web server).	The page contains "server-side" code it allows the server to generate the unique content when the page is loaded.
Flexibility is the main advantage of static website.	Content Management System (CMS) is the main advantage of dynamic website.



2

UNIT 2

LANGUAGE AND TECHNOLOGY FOR BROWSERS

Unit Structure

HTML
DHTML
XHTML
JSP
JavaScript
Features and Applications

2.1 HTML

HTML, which stands for **Hypertext Markup Language**, is the predominant markup language for web pages. It is written in the form of HTML elements consisting of "tags" surrounded by angle brackets within the web page content.

It allows images and objects to be embedded and can be used to create interactive forms. It provides a means to create structured documents by denoting structural semantics for text such as headings, paragraphs, lists, links, quotes and other items. It can embed scripts in languages such as JavaScript which affect the behavior of HTML web pages.

HTML can also be used to include Cascading Style Sheets (CSS) to define the appearance and layout of text and other material. The W3C, maintainer of both HTML and CSS standards, encourages the use of CSS over explicit presentational markup.

2.1.1 A brief history of HTML

HTML and SGML

HTML stands for Hyper-Text Markup Language. It is a coding language, which uses a method called markup, to create hyper-text. HTML is actually a simplified subset of a more general markup language called SGML, which stands for Standard Generalized Markup Language. .

The first version of the Netscape browser implemented HTML 1.0.

HTML 1.0 and 2.0

In 1992, Berners-Lee and the CERN team released the first draft HTML 1.0, which was finalized in 1993. This specification was so simple it could be printed on one side of a piece of paper, but even then it contained the basic idea that has become central in the recent evolution of HTML, which is the separation between logical structures and presentational elements. In 1994, HTML 2.0 was developed by the Internet Engineering Task Force's HTML Working Group. This group later was disbanded in favor of the World Wide Web Consortium (<http://www.w3.org>), which continues to develop HTML.

Browsers and HTML

Netscape was just one of a number of browsers available. Mosaic was still offered by NCSA, Lynx was available on Unix machines, and few other companies were creating browsers. One of them, Spyglass, was purchased by Microsoft, and became the basis for Internet Explorer. Each browser contains, in its heart, a *rendering engine*, which is the code that tells it how to take your HTML and turn it into something you can see on the screen.

W3C takes over: HTML 3.0 and HTML 3.2

The World Wide Web Consortium (W3C), which had taken over HTML development, attempted to create some standardization in HTML 3.0. In 1996 a consensus version, HTML 3.2, was issued. This added features like tables, and text flowing around images, to the official specification, while maintaining backwards compatibility with HTML 2.0.

HTML 4.0x

The W3C released the HTML 4.0 specification at the end of 1997, and followed with HTML 4.01 in 1999, which mostly corrected a few errors in the 4.0 specification.

XHTML 1.0

This is the successor to HTML. The "X" stands for Extensible. This is a reformulation of HTML 4.01 within XML (Extensible Markup Language), which is far more rigorous, and is intended to start moving the creation of Web pages away from HTML. This was released earlier this year, and is the most current standard for creating Web pages. This introduces some interesting changes in coding. For example, virtually all tags now have to be closed, including paragraph tags. Other tags, like the FONT tag, have been banished in favor of using Cascading Style Sheets to control all presentational elements.

HTML5

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

WHATWG was working with web forms and applications, and W3C was working with XHTML 2.0. In 2006, they decided to cooperate and create a new version of HTML.

Some rules for HTML5 were established:

- New features should be based on HTML, CSS, DOM, and JavaScript
- Reduce the need for external plugins (like Flash)
- Better error handling
- More markup to replace scripting
- HTML5 should be device independent
- The development process should be visible to the public

The HTML5 <!DOCTYPE>

In HTML5 there is only one <!doctype> declaration, and it is very simple:

```
<!DOCTYPE html>
```

Minimum HTML5 Document

Below is a simple HTML5 document, with the minimum of required tags:

```
<!DOCTYPE html>
<html>
<head>
<title>Title of the document</title>
</head>

<body>
The content of the document.....
</body>

</html>
```

DHTML

Dynamic HTML, or **DHTML**, is an umbrella term for a collection of technologies used together to create interactive and animated web sites by using a combination of a static markup language (such as HTML), a client-side scripting language (such as JavaScript), a presentation definition language (such as CSS), and the Document Object Model.

DHTML allows scripting languages to change variables in a web page's definition language, which in turn affects the look and function of otherwise "static" HTML page content, *after* the page has been fully loaded and during the viewing process. Thus the dynamic characteristic of DHTML is the way it functions while a page is viewed, not in its ability to generate a unique page with each page load.

By contrast, a dynamic web page is a broader concept — any web page generated differently for each user, load occurrence, or specific variable values. This includes pages created by client-side scripting, and ones created by server-side scripting (such as PHP, Perl, JSP or ASP.NET) where the web server generates content before sending it to the client.

There are four parts to DHTML

Document Object Model (DOM) Scripts

Cascading Style Sheets (CSS)

XHTML

DOM

Definition: Document Object Model; The DOM or DocumentObject Model is the API that binds JavaScript and other scripting languages together with HTML and other markup languages. It is what allows Dynamic HTML to be dynamic.

The DOM is what allows you to access any part of your Web page to change it with DHTML. Every part of a Web page is specified by the DOM and using its consistent naming conventions you can access them and change their properties.

Scripts

Scripts written in either JavaScript or ActiveX are the two most common scripting languages used to activate DHTML. You use a scripting language to control the objects specified in the DOM.

CascadingStyleSheets (CSS)

CSS is used in DHTML to control the look and feel of the Web page. Style sheets define the colors and fonts of text, the background colors and images, and the placement of objects on the page. Using scripting and the DOM, you can change the style of various elements

XHTML

XHTML or HTML 4.x is used to create the page itself and build the elements for the CSS and the DOM to work on. There is nothing special about XHTML for DHTML - but having valid XHTML is even more important, as there are more things working from it than just the browser.

Features of DHTML

There are four primary features of DHTML:

- Changing the tags and properties
- Real-time positioning
- Dynamic fonts (Netscape Communicator)
- Data binding (Internet Explorer)

Changing the tags and Properties

This is one of the most common uses of DHTML. It allows you to change the qualities of an HTML tag depending on an event outside of the browser (such as a mouse click, time, or date, and so on). You can use this to preload information onto a page, and not display it unless the reader clicks on a specific link.

Real-time postioning

When most people think of DHTML this is what they expect. Objects, images, and text moving around the Web page. This can allow you to play interactive games with your readers or animate portions of your screen.

Dynamic Fonts

This is a Netscape only feature. Netscape developed this to get around the problem designers had with not knowing what fonts would be on a reader's system. With dynamic fonts, the fonts are encoded and downloaded with the page, so that the page always looks how the designer intended it to.

Data binding

This is an IE only feature. Microsoft developed this to allow easier access to databases from Web sites. It is very similar to using a CGI to access a database, but uses an ActiveX control to function. This feature is very advanced and difficult to use for the beginning DHTML writer.

XHTML

XHTML (Extensible Hypertext Markup Language) is a family of XML markup languages that mirror or extend versions of the widely used Hypertext Markup Language (HTML), the language in which web pages are written.

The Main Changes

There are several main changes in XHTML from HTML:

- All tags must be in lower case
- All documents must have a doctype
- All documents must be properly formed All tags must be closed
- All attributes must be added properly The name attribute has changed
- Attributes cannot be shortened All tags must be properly nested

The Doctype

The first change which will appear on your page is the Doctype. When using HTML it is considered good practice to add a Doctype to the beginning of the page like this.

Although this was optional in HTML, XHTML requires you to add a Doctype. There are three available for use.

Strict - This is used mainly when the markup is very clean and there is no 'extra' markup to aid the presentation of the document. This is best used if you are using Cascading Style Sheets for presentation. `<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">`

Transitional - This should be used if you want to use presentational features of HTML in your page. <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

Frameset - This should be used if you want to have frames on your page. <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Frameset//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-frameset.dtd">

The doctype should be the very first line of your document and should be the only thing on that line. You don't need to worry about this confusing older browsers because the Doctype is actually a comment tag. It is used to find out the code which the page is written in, but only by browsers/validators which support it, so this will cause no problems.

Document Formation

After the Doctype line, the actual XHTML content can be placed. As with HTML, XHTML has <html><head><title> and <body> tags but, unlike with HTML, they must all be included in a valid XHTML document. The correct setup of your file is as follows:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML1.0
Transitional//EN""http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>

<title>PageTitle</title>
OTHER HEADDATA

</head>
<body>
CONTENT
</body>
</html>
```

It is important that your document follows this basic pattern. This example uses the transitional Doctype but you can use either of the others (although frames pages are not structured in the same way).

General Rules for converting HTML to XHTML

The first line in the HTML document may be the XML processing instruction:

```
<? xml version="1.0" encoding="iso-8859-1"?>
```

W3C recommends that this declaration be included in all XHTML documents, although it is absolutely required only when the character encoding of the document is

other than the default Unicode UTF-8 or UTF-16. I said necessary because there can be problems with older browsers which cannot identify this as a valid HTML tag.

The second line in the XHTML document should be the specification of the document type declaration (DTD) used. The document type declaration for transitional XHTML documents is:

```
<!DOCTYPE html
PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

The declarations for the strict XHTML DTD is:

```
<!DOCTYPE html
PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

The declarations for the frameset XHTML DTD is:

```
<!DOCTYPE html
PUBLIC "-//W3C//DTD XHTML 1.0 Frameset//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-frameset.dtd">
```

XML requires that there must be one and only one root element for a document. Hence, in XHTML, all tags should be enclosed within the `<html>` tag, ie., `<html>` should be the root element for the document.

The starting tag `<html>` should be modified to include namespace information. The modification is:

```
<html xmlns="http://www.w3.org/1999/xhtml" lang="EN">
```

Attribute `xmlns` is the XML namespace with which we associate the XHTML document. The value of the attribute `lang` is the code for the language of the document as specified in RFC1766.

XHTML tag elements should be in lower case. That means `<HTML>` and `<Body>` are wrong. They should be rewritten as `<html>` and `<body>` respectively.

All XHTML tags should have their end tags. In HTML it is common for paragraphs to have only the starting `<p>` tag. In XHTML this is not allowed. You need to end a paragraph with

the</p> tag. Example: <p>Hello is wrong; it should be written as<p>Hello</p>.

Empty XHTML tags should be ended with /> instead of >.

The commonly used empty tags in XHTML are:

<meta />: for meta information (contained in the head section).

<base />: used to specify the base URI and also the targetframe for hyperlinks (contained in the head section).

<basefont />: used to specify a base font for the document. Note that attribute 'size' is mandatory.

<param />: parameters for applets and objects.

<link />: to specify external stylesheets and other references.

: to include images. Attributes 'src' for the source URI and 'alt' for alternate text are mandatory.

: used for forced line break.

<hr />: for horizontal rules.

<area />: used inside image maps. Attribute 'alt' is mandatory.

<input />: used inside forms for input form elements like buttons, textboxes, textareas, checkboxes and radio buttons.

Example: <br clear="all"> is wrong; it should be rewritten as <brclear="all" />. <imgsrc="back.gif" alt="Back"> is wrong; it should be <imgsrc="back.gif" alt="Back" />

Proper nesting of tags is compulsory in XHTML. Example: <i>This is bold italics<i> is wrong. It should be rewritten as <i>This is bold italics</i>.

Rules for XHTML Attributes

All XHTML attribute names should be in lower case.

Example: Width="100" and WIDTH="100" are wrong;
only width="100" is correct.

Similarly onMouseOut="javascript:myFunction();" is wrong; it should be rewritten as onmouseout="javascript:myFunction();".

All attribute-value pairs should be quoted.
Example: width=100 is wrong; it should be
width="100" or width='100'.

HTML supports certain attributes which have no values.

Examples are noshade which appears in the <hrnoshade/>tag. XHTML does not allow such empty or compact attributes. The compact attributes generally found in HTML are compact, nowrap, ismap, declare, noshade, checked, disabled, readonly, multiple, selected, noresize and defer. They should always have a value. In XHTML this is done by giving the

attribute name itself as the value!

Example: noshade becomes noshade="noshade"

checked becomes checked="checked".

The name attribute is deprecated and will be removed in a future version of XHTML and the id attribute will take its place. So, for HTML tags that need the name attribute, an id attribute should also be specified with the same value as that for name.

Example: <frame name="myFrame" > becomes <frame name="myFrame" id="myFrame" >

All & (ampersand) characters in the source code have to be replaced with &, which is the equivalent character entity

code. This change should be done in all attribute values and

URLs.

Example: Bee&Nee will result in an error if you try to validate it;

It should be written as Bee&Nee.

Go is wrong; it should be coded as Go.

XHTML Tables

For <table> tag, attribute height is not supported in XHTML 1.0. Only the width is supported. The <td> tag does support the height attribute.

The <table>, <tr> and the <td> tag does not support the attribute background which is used to specify a background image for the table or the cell. Background images will have to be specified either using the style attribute or using external stylesheet. The attribute bgcolor for background color is however supported by these tags.

XHTML Images

The alt attribute is mandatory. This value of this attribute will be the text that has to be shown in older browsers, text-only browsers, and in place of the image when it is not available. Note that is an empty tag.

Example: <imgsrc="back.gif" alt="Back" />

XHTML and Javascript

The type attribute is mandatory for all <script> tags. This value of type is text/javascript for Javascript.

The use of external scripts is recommended.

Example:

```
<script type="text/javascript" language="javascript" src="functions.js"></script>
```

XHTML and Stylesheets

The type attribute is mandatory for <style> tag. The value of type is text/css for stylesheets.

The use of external stylesheets is recommended.

Example: <link rel="stylesheet" type="text/css" href="screen.css" />

Element Prohibitions in XHTML

The W3C recommendation also prohibits certain XHTML elements from containing some elements. Those are given below:

<a>cannot contain other<a>elements.<pre>cannot contain the, <object>, <big>, <small>, <sub>, or <sup>elements.
<button>cannot contain

the<input>, <select>, <textarea>, <label>, <button>,<form>, <fieldset>, <iframe>, or<isindex>elements.

<label>cannot contain other<label>elements.<form>cannot contain other<form>elements.

JAVASCRIPT

A scripting language developed by Netscape to enable Web authors to design interactive sites. Although it shares many of the features and structures of the full Java language, it was developed independently. JavaScript can interact with HTML source code, enabling Web authors to spice up their sites with dynamic content. JavaScript is endorsed by a number of software companies and is an open language that anyone can use without purchasing a license. It is supported by recent browsers from Netscape and Microsoft, though Internet Explorer supports only a subset, which Microsoft calls *Jscript*.

What can a JavaScript do?

JavaScript gives HTML designers a programming tool - HTML authors are normally not programmers, but JavaScript is a scripting language with a very simple syntax! Almost anyone can put small "snippets" of code into their HTML pages

JavaScript can put dynamic text into an HTML page - A JavaScript statement like this: `document.write("<h1>" + name + "</h1>")` can write a variable text into an HTML page

JavaScript can react to events - A JavaScript can be set to execute when something happens, like when a page has finished loading or when a user clicks on an HTML element

JavaScript can read and write HTML elements - A JavaScript can read and change the content of an HTML element

JavaScript can be used to validate data - A JavaScript can be used to validate form data before it is submitted to a server. This saves the server from extra processing

JavaScript can be used to detect the visitor's browser - A JavaScript can be used to detect the visitor's browser, and - depending on the browser - load another page specifically designed for that browser

JavaScript can be used to create cookies - A JavaScript can be used to store and retrieve information on the visitor's computer

FEATURES AND APPLICATION

There are literally hundreds of different technologies available to the webmaster. Making proper use of these technologies allows the creation of maintainable, efficient and useful web sites. For example, using SSI (server side includes) or CSS (cascading style sheets) a webmaster can change every page on his web site by editing one file.

A few of the more common technologies are listed below.

CGI

Common Gateway Interface is one of the older standards on the internet for moving data between a web page and a web server. CGI is by far and away the most commonly used method of handling things like guestbooks, email forms, message boards and so on. CGI is actually a standard for passing data back and forth and not a scripting language at all. In fact, CGI routines are commonly written in interpreted languages such as PERL or compiled languages like C.

CSS

Cascading Style Sheets to format your web pages anyway that you want. CSS is a language that describes the style of an HTML document. CSS describes how HTML elements should be displayed.

JavaScript

This is a scripting language which is interpreted and executed by the browser. It is very useful for getting tasks done on the client, such as moving pictures around the screen, creating very dynamic navigation systems and even games. JavaScript is generally preferable on internet sites because it is supported on more browsers than VBScript, which is the chief competitor.

JSP

Java Server Pages (JSP) is a server-side programming technology that enables the creation of dynamic, platform-independent method for building Web-based applications. JSP have access to the entire family of Java APIs, including the JDBC API to access enterprise databases.

ASP and ASP.NET

ASP stands for Active Server Pages. ASP is a development framework for building web pages. ASP supports many different development models:

Classic ASP

ASP.NET Web Forms

ASP.NET MVC

ASP.NET Web Pages

ASP.NET API

ASP.NET Core

The ASP Technology

ASP and ASP.NET are server side technologies.

Both technologies enable computer code to be executed by an Internet server.

When a browser requests an ASP or ASP.NET file, the ASP engine reads the file, executes any code in the file, and returns the result to the browser.

Office

The Microsoft Office suite includes a number of tools, including Word, Excel, Access and Powerpoint. Each of these tools has the ability to save in HTML format and has special commands for the internet. This is especially useful, for example, if you work in an office where people are trained in Excel and you don't want to retrain them to create web pages. On the other hand, if you are creating internet web sites (as opposed to intranet sites) you probably would be better off using web specific products to edit your web pages.

Perl

A great scripting language which makes use of the CGI standard to allow work to be done on the web server. PERL is very easy to learn (as programming languages go)

and straightforward to use. It is most useful for guestbooks, email forms and other similar, simple tasks. PERL's primary disadvantage is the overhead on the server is very high, as one process is created each time a routine is called, and the language is interpreted, which means the code is recompiled each time it is run. For complex tasks, a server-side scripting language such as PHP or ASP is much preferred.

PHP

This language is, like ASP, used to get work done on the server. PHP is similar in concept to ASP and can be used in similar circumstances. PHP is very efficient, allows access to databases using products such as MySQL, and can be used to create very dynamic web pages.

SSI

If your site is hosted on a typical Apache server, then you probably can use something called Server Side Includes. This is a way to get the web server to perform tasks before displaying a web page. One of the most common uses is to, well, include common text. This is great when you have, for example, a navigation system which is common to all of your pages. You can make one change in an SSI file and thus change your entire web site.

SSI is very common but has really been superseded by languages such as PHP. The overhead of SSI on the server is high as each page is scanned for SSI directives before passing it to the browser.

VBScript

Visual Basic Scripting was Microsoft's answer to JavaScript. VBScript is a good tool for any site which is intended to be only displayed by the Internet Explorer browser. In my opinion, VBScript should never be used on a web site - JavaScript is preferable due to a wider acceptance among browsers.



INTRODUCTION TO HTML

Unit Structure

HTML Fundamentals

HTML Browsers

HTML tags, Elements and Attributes

Structure of HTML code

- Head Body

- Ordered List

- Unordered List

- Definition List

- Nesting List

HTML FUNDAMENTALS

HTML – HyperText Markup Language – The Language of Web Pages on the World Wide Web.

HTML is a text formatting language.

Its collection of “**TAGS**”, that are used to make web documents that are displayed using browsers on internet.

HTML is platform independent language.

To display a document in web it is essential to mark-up the different elements (headings, paragraphs, tables, and so on) of the document with the HTML tags.

To view a mark-up document, user has to open the document in a browser.

Browser – A software program which is used to show web pages

A browser understands and interprets the HTML tags, identifies the structure of the document (which part are which) and makes decision about presentation (how the parts look) of the document.

We can also make documents look attractive using graphics, fonts size and color using HTML

User can make a link to the other document or the different section of the same document by creating Hypertext Links also known as Hyperlinks.

How to make HTML pages?

There are many different programs that you can use to create web documents.

HTML Editor – A word processor that has been specialized to make the writing of HTML documents more effortless.

HTML Editors enable users to create documents quickly and easily by pushing a few buttons. Instead of entering all of the HTML codes by hand. These programs will generate the HTML Source Code for you.

HTML Editors are excellent tools for experienced web developers; however, it is important that you learn and understand the HTML language so that you can edit code and fix “bugs” in your pages.

Editing HTML

We can use a plain text editor (like Notepad) to edit HTML.

We can also use dreamviewer or frontpage

When you save an HTML file, you can use either the **.htm** or the **.html** file extension.

HTML gives authors the means to:

Publish online documents with headings, text, tables, lists, photos, etc.

Retrieve online information via hypertext links, at the click of a button.

Design forms for conducting transactions with remote services, for use in searching for information, making reservations, ordering products, etc.

Include spread-sheets, video clips, sound clips, and other applications directly in their documents.

TAGS, ELEMENTS, ATTRIBUTE

HTML is set of instructions.

These instructions, along with the text to which the instructions apply, are called

HTML Elements

The HTML instructions are themselves called as **tags**, and look like **<element_name>** -- that is, element name surrounded by left and right angle brackets(<>).

HTML Tags

HTML markup tags are usually called HTML tags:

HTML tags are keywords surrounded by **angle brackets** like <html>

HTML tags normally **come in pairs** like and

The first tag in a pair is the **start tag**, the second tag is the **end tag**

Start and end tags are also called **opening tags** and **closing tags**

Tags are used to represent various elements of web page like Header, Footer, Title, Images etc. Tags are of two types:

Container Tags, Empty Tags.

Container Tags:

These tags are always paired with closures tags are called *container tags*.

These tags activate an effect and have a companion tag to close/discontinue the effect.

Tags which have both the opening and closing i.e. <TAG> and </TAG>

For example tag starts bold effect for text and its companion tag ends the bold effect.

Statement like: How

Will have word **How** in bold.

The <HTML>, <HEAD>, <TITLE> and <BODY><SCRIPT><A> etc. tags are all container tags.

Empty Tags:

Tags, which have only opening and no ending, are called *empty tags/ standalone tag*. The

<HR>, which is used to draw horizontal rule across the width of the document, and line break
 tags are empty tags.

When client request for a page from web server browser fetches. .

All web pages contain instructions for display called 'tags'.

Browsers read tags and display page according to tags on client

computer

HTML *Attributes*

HTML elements can have **attributes**.

Attributes provide **additional information** about an element about how the tag should appear or behave.

Attributes are always specified in **the start tag** .

An element's start tag may contain any number of space separated attribute/value pairs.

Attributes consist of a *name* and a *value* separated by an equals (=) sign (name/value pairs like: **name = "value"**).

For example, consider the tag BODY, which marks as the beginning (or end) of HTML body.

This tag can have several attributes, one of them is BGCOLOR, specific the background color of the document.

```
<BODY          bgcolor      =      "background_color"  background      =  
"background_image">.
```

Attribute values should always be enclosed in quotes.

Double style quotes are the most common, but single style quotes are also allowed.

Many attributes are available to HTML elements, some are common across most tags, and others can only be used on certain tags. Some of the more common attributes are:

Attribute	Description	Possible Values
Class	Used with <u>Cascading Style Sheets</u> (CSS)	(the name of a predefined class)
Style	Used with <u>Cascading Style Sheets</u> (CSS)	(You enter CSS code to specify how the way the HTML element is presented)
Title	Can be used to display a "tooltip" for your elements.	(You supply the text)

STRUCTURE OF HTML CODE

HTML documents are structured into two parts, the **HEAD**, and the **BODY**.

Both of these are contained within the HTML element – it simply denotes its HTML document

The head contains information about the document that is not generally displayed with the document, such as its **TITLE**.

The **BODY** contains the body of the text

Elements allowed inside the HEAD, such as TITLE, are not allowed inside the BODY, and vice versa.

Creating a Basic Starting Document

page1.html

```
<HTML>
<HEAD>
<TITLE>My First Page</TITLE>
</HEAD>
<BODY>
    Hello World !!!<br>
    I am learning Web Technology.
</BODY>
</HTML>
```

The HEAD of your document point to above window part.

The TITLE of your document appears in the very top line of the user's browser. If the user chooses to "Bookmark" your page or save as a "Favorite"; it is the TITLE that is added to the list.

The text in your TITLE should be as descriptive as possible because this is what many search engines, on the internet, use for indexing your site.

Setting Document Properties

Document properties are controlled by attributes of the BODY element. For example, there are color settings for the background color of the page, the document's text and different states of links.

BODY tag **(main content of document)**

The BODY tag specifies the main content of a document. You should put all content that is to appear in the web page between the <BODY> and </BODY> tags.

The BODY tag has attributes that let you specify characteristics for the document. You can specify the background color or an image to use as a tiled background for the window in which the document is displayed. You can specify the default text color, active link color, unvisited link color, and visited link color. You can specify actions to occur when the document finishes loading or is unloaded, and when the window in which the document is displayed receives or loses focus.

Syntax

```
<BODY  
  BACKGROUND="bgURL"  
  BGCOLOR="color"  
  TEXT="color"  
  LINK="color"  
  ALINK="color"  
  VLINK="color"  
  ONLOAD="loadJScript"  
  ONUNLOAD="unloadJScript"  
  ONBLUR="blurJScript"  
  ONFOCUS="focusJScript"  
  CLASS="styleClass"  
  ID="namedPlaceOrStyle"  
  LANG="ISO"  
  STYLE="style"  
>  
...  
</BODY>
```

BACKGROUND="bgURL" (Deprecated)

specifies an image to display in the background of the document. The URL value can be an absolute URL

for example, "http://www.webopedia.com/imagesvr_ce/2123/computer.jpg"

or a relative URL

for example, "images/image1.gif".

The image is tiled, which means it is repeated in a grid to fill the entire window or frame where the document is displayed.

BGCOLOR="color" (Deprecated)

sets the color of the background. See Color Palette for information about color values.

TEXT="color" (Deprecated)

sets the color of normal text (that is, text that is not in a link) in the document. See Color Palette for information about color values.

LINK="color" (Deprecated)

sets the default text color of unvisited links in the document. An unvisited link is a link that has not been clicked on (or followed)..

ALINK="color" (Deprecated)

specifies the color to which links briefly change when clicked. After flashing the ALINK color, visited links change to the VLINK color if it has been specified; otherwise they change to the browser's default visited link color.

VLINK="color" (Deprecated)

specifies the text color of visited (followed) links in a document.

ONLOAD="loadJScript"

specifies JavaScript code to execute when the document finishes loading.

ONUNLOAD="unloadJScript"

specifies JavaScript code to execute when the document is unloaded.

ONFOCUS="focusJScript"

specifies JavaScript code to execute when the window in which the document is displayed receives an onFocus event, indicating that the window has acquired focus.

ONBLUR="blurJScript"

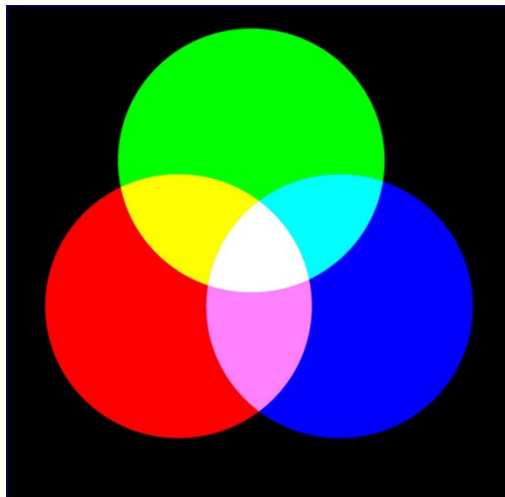
specifies JavaScript code to execute when the window in which the document is displayed receives an onBlur event, indicating that the window has lost focus.

Example

The following example sets the background color to light yellow, ordinary text to black, unvisited links to blue, visited links to green, and active links to red:

```
<BODY BGCOLOR="#FFFFAA" TEXT="black" LINK="blue" VLINK="green"
ALINK="red">
...
</BODY>
```

Main Colours



16 Basic Colors

Color Name	RGB Triplet	Hexadecimal	Color Name	RGB Triplet	Hexadecimal
Aqua	(0,255,255)	00FFFF	Navy	(0,0,128)	000080
Black	(0,0,0)	000000	Olive	(128,128,0)	808000
Blue	(0,0,255)	0000FF	Purple	(128,0,128)	800080
Fuchsia	(255,0,255)	FF00FF	Red	(255,0,0)	FF0000
Gray	(128,128,128)	808080	Silver	(192,192,192)	C0C0C0
Green	(0,128,0)	008000	Teal	(0,128,128)	008080
Lime	(0,255,0)	00FF00	White	(255,255,255)	FFFFFF
Maroon	(128,0,0)	800000	Yellow	(255,255,0)	FFFF00

Headings, <Hx></Hx>

Inside the BODY element, heading elements H1 through H6 are generally used for major divisions of the document. Headings are permitted to appear in any order, but you will obtain the best results when your documents are displayed in a browser if you follow these guidelines:

H1: should be used as the highest level of heading, H2 as the next highest, and so forth. You should not skip heading levels: e.g., an H3 should not appear after an H1, unless there is an H2 between them.

```
<HTML>
<HEAD>
<TITLE> Example Page</TITLE>
</HEAD>
<BODY>
<H1> Heading 1 </H1>
<H2> Heading 2 </H2>
<H3> Heading 3 </H3>
<H4> Heading 4 </H4>
<H5> Heading 5 </H5>
<H6> Heading 6 </H6>
</BODY>
</HTML>
```

Heading 1

Heading 2

Heading 3

Heading 4

Heading 5

Heading 6

Paragraphs, <P></P>

Paragraphs allow you to add text to a document in such a way that it will automatically adjust the end of line to suite the window size of the browser in which it is being displayed. Each line of text will stretch the entire length of the window.

```
<HTML>
<HEAD>
<TITLE> Example Page</TITLE>
</HEAD>
<BODY></H1> Heading 1 </H1>
<P> Paragraph 1, ....</P>
<H2> Heading 2 </H2>
<P> Paragraph 2, ....</P>
<H3> Heading 3 </H3>
<P> Paragraph 3, ....</P>
<H4> Heading 4 </H4>
<P> Paragraph 4, ....</P>
<H5> Heading 5 </H5>
<P> Paragraph 5, ....</P>
<H6> Heading 6</H6>
<P> Paragraph 6, ....</P>
</BODY>
</HTML>
```

Heading 1

Paragraph 1,....

Heading 2

Paragraph 2,....

Heading 3

Paragraph 3,....

Heading 4

Paragraph 4,....

Heading 5

Paragraph 5,....

Heading 6

Paragraph 6,....

Break,

Line breaks allow you to decide where the text will break on a line or continue to the end of the window.

A
 is an empty Element, meaning that it may contain attributes but it does not contain content.

The
 element does not have a closing tag but in XHTML it is written as
.

```
<HTML>
<HEAD>
<TITLE> Example Page</TITLE>
</HEAD>
<BODY>
<H1> Heading 1 </H1>
<P>Paragraph 1, <BR>
Line 2 <BR> Line 3 <BR>....
</P>
</BODY>
</HTML>
```

Heading 1

Paragraph 1,....

Line 2

Line 3

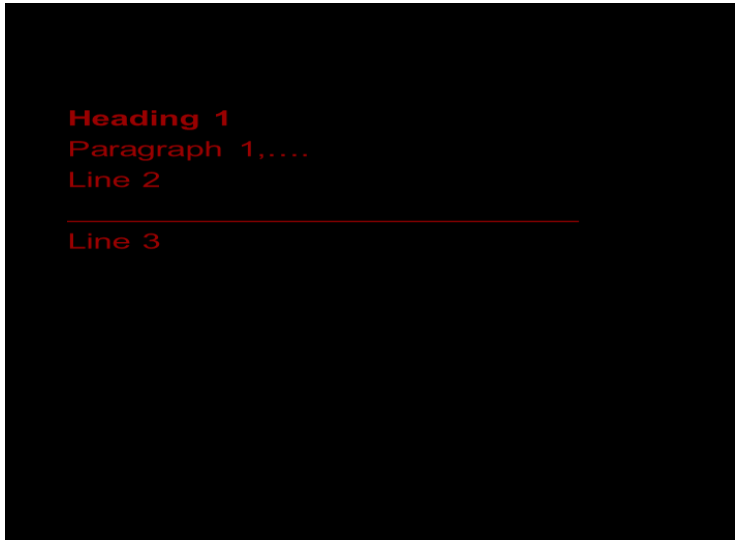
....

Horizontal Rule, <HR>

The <HR> element causes the browser to display a horizontal line (rule) in your document.

<HR> does not use a closing tag, </HR>. In XHTML it is written as **<hr/>**.

```
<HTML>
<HEAD>
<TITLE> Example Page</TITLE>
</HEAD>
<BODY>
<H1> Heading 1 </H1>
<P>Paragraph 1, <BR>
Line 2 <BR>
<HR>Line 3 <BR>
</P>
</BODY>
</HTML>
```



Heading 1
Paragraph 1,....
Line 2

Line 3

Character Formatting

In this chapter you will learn how to enhance your page with Bold, Italics, and other character formatting options.

Objectives

Upon completing this section, you should be able to

1. Change the color and size of your text.
2. Use Common Character Formatting Elements.
3. Align your text.
4. Add special characters.
5. Use other character formatting elements.

Bold, Italic and other Character Formatting Elements

 Two sizes bigger (*not supported in HTML5*) will still work in some browsers. CSS is preferred.

The size attribute can be set as an absolute value from 1 to 7 or as a relative value using the "+" or "-" sign. Normal text size is 3 (from -2 to +4).

** Bold **

<I> Italic </I>

<U> Underline </U>

Color = "#RRGGBB" The COLOR attribute of the FONT element. E.g., this text has color

<PRE> Preformatted </PRE> Text enclosed by PRE tags is displayed in a mono-spaced font. Spaces and line breaks are supported without additional elements or special characters.

** Emphasis ** Browsers usually display this as italics.

** STRONG ** Browsers display this as bold.

<TT> TELETYPE </TT> Text is displayed in a mono-spaced font. A typewriter text, e.g. fixed-width font.

<STRIKE> strike-through text </STRIKE>

<BIG>places text in a big font </BIG>

<SMALL> places text in a small font</SMALL>

_{places text in subscript position}

^{places text in superscript style position}

Alignment

Some elements have attributes for alignment (ALIGN) e.g. Headings, Paragraphs and Horizontal Rules.

The Three alignment values are : LEFT, RIGHT, CENTER.

<CENTER></CENTER> Will center elements.

Example:

```
<html>
<head>
<title> Text Formating</title>
</head>
<body>
<center>
This is normal text
<br><br>
<b> - This line is in Bold text - </b>
<br><br>
<strong> - Important text simillar to bold - </strong>
<br><br>
<i> - Italic text - </i><br><br>
<em> - Emphasized text - </em><br><br>
<mark> - Marked text - </mark><br><br>
<small> - Small text - </small><br><br>
<del> - Deleted text - </del><br><br>
<ins> - Inserted text - </ins><br><br>

<sub> - Subscript text - </sub><br><br>

<sup> - Superscript text - </sup><br><br>

H<sub>2</sub></sub>O <br><br>

23<sup>rd</sup></sup>August 2017 <br><br>

<code> - Source Code text - </code><br><br>

<tt> - Type Writter Text - </tt><br><br>

<font size="2"> Hello</font><br><br>

<font face="Algerian"> Hello </font><br><br>

<font color="red"> Hello </font><br><br>

<font color="teal" face="verdana" size="12" > All font attributes together </font>

<br><br>

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

List Elements

HTML supplies several list elements. Most list elements are composed of one or more (List Item) elements.

UL : Unordered List. Items in this list start with a list mark such as a bullet. Browsers will usually change the list mark in nested lists.

```
<UL>
<LI> List item ...</LI>
<LI> List item ...</LI>
</UL>
List item ...
List item ...
```

You have the choice of three bullet types: disc(default), circle, square.

These are controlled by the "TYPE" attribute for the element.

```
<UL TYPE="square">
<LI> List item ...</LI>
<LI> List item ...</LI>
<LI> List item ...</LI>
</UL>
```

- List item ...
- List item ...
- List item ...

OL: Ordered List. Items in this list are numbered automatically by the browser.

```
<OL>
<LI> List item ...</LI>
<LI> List item ...</LI>
<LI> List item ...</LI>
</OL>
```

1. List item ...
2. List item ...
3. List item ...

You have the choice of setting the TYPE Attribute to one of five numbering styles.

Arabic numbers	1,2,3,
Lower alpha	a, b, c,
Upper alpha	A, B, C,
Lower roman	i, ii, iii,
Upper roman	I, II, III,

You can specify a starting number for an ordered list.

```
<OL TYPE="I">
<LI> List item ...</LI>
<LI> List item ...</LI>
</OL>
<P> text ....</P>
<OL TYPE="I" START="3">
<LI> List item ...</LI>
</OL>
```

DL: Definition List. This kind of list is different from the others. Each item in a DL consists of one or more Definition Terms (DT elements), followed by one or more Definition Description (DD elements).

```
<DL>
<DT> HTML </DT>
<DD> Hyper Text Markup Language </DD>
<DT> DOM </DT>
<DD> A human's best friend!</DD>
</DL>
```

HTML

Hyper Text Markup Language

DOM

Document Object Model

Nesting Lists

You can nest lists by inserting a UL, OL, etc., inside a list item (LI).

Example

```
<UL TYPE="square">
<LI> List item ...</LI>
<LI> List item ...
<OL TYPE="I" START="3">
<LI> List item ...</LI>
<LI> List item ...</LI>
<LI> List item ...</LI>
<LI> List item ...</LI>
<LI> List item ...</LI>
</OL>
</LI>
<LI> List item ...</LI>
</UL>
```

Comments <!-- -->

Although HTML documents tend to be fairly legible, there are several advantages to adding comments to your HTML code. HTML uses the tag

<!-- to begin a comment and -->

to end a comment. Note that the comment can span multiple lines, but the browser will ignore anything between the comment tags.

INSERTING IMAGE

Tag type: Stadalone Function:

Images can be placed in a web page by using tag.

The gif format is considered superior to the jpeg format for its clarity and ability to maintain the originality of an image without lowering its quality.

Appearance:

Attributes: SRC=URL

ALT=string

ALIGN=left|right|top|middle|bottom

HEIGHT=n

WIDTH=n

BORDER=n

HSPACE=n

VSPACE=n

Example:

```

```

Image Maps

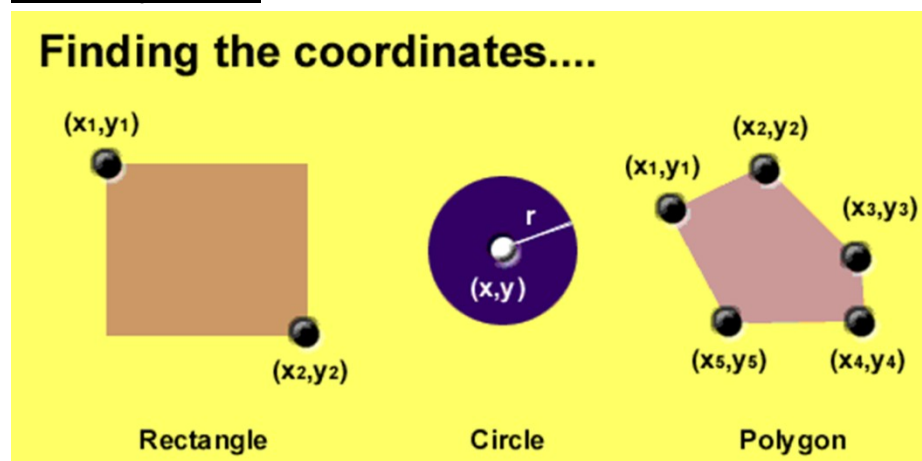
Image maps are images, usually in gif format that have been divided into regions; clicking in a region of the image cause the web surfer to be connected to a new URL. Image maps are graphical form of creating links between pages.

There are two type of image maps:

Client side and server side

Both types of image maps involve a listing of co-ordinates that define the mapping regions and which URLs those coordinates are associated with. This is known as the map file.

Area ShapesUsed



Types of Shapes

Rect : used for squares and ordered shapes.

Circle : used for circles.

Poly : used for unordered shapes.

Number of coordinations for each shape:

Rect : 4 numbers for two corners

Circle : 3 numbers for the center & R

Poly : depends on the number of corners of the shape(2 numbers for each corner)

Client-side image maps (USEMAP) use a map file that is part of the HTML document (in an element called MAP), and is linked to the image by the Web browser.

example

```
<IMG SRC="tomandjerry.jpg" Width=200 Height=200
border="5" USEMAP="#map1">
<MAP NAME="map1">
<AREA SHAPE="RECT" COORDS="0,0,90,90" HREF="first.html" ALT="see first...">
<AREA SHAPE="RECT" COORDS="100,100,160,160" HREF="second.html" ALT="see
second..." >
<AREA SHAPE="CIRCLE" COORDS="150,50,20" HREF="third.html"
ALT="see third..." >
</MAP>
```

HOW TO MAKE A Hyper LINK

Anchor tag

The tags used to produce links are the <A> and . The <A> tells where the link should start and the indicates where the link ends. Everything between these two will work as a link.

The example below shows how to make the word **here** work as a link to yahoo.

Click here to go to yahoo.

Internal Links : Links can also be created inside large documents to simplify navigation. Today's world wants to be able to get the information quickly. Internal links can help you meet these goals.

Select some text at a place in the document that you would like to create a link to, then add an anchor to link to like this:

```
<A NAME="bookmark_name"></A>
```

The Name attribute of an anchor element specifies a location in the document that we link to shortly. All NAME attributes in a document must be unique.

Next select the text that you would like to create as a link to the location created above.

```
<A HREF="#bookmark_name">Go To Book Mark</A>
```

FRAMESET AND FORMS

FRAMESET

The **FRAMESET** element is a *frame container* for dividing a window into rectangular subspaces called *frames*. In a Frameset document, the outermost **FRAMESET** element takes the place of **BODY** and immediately follows the HEAD.

The **FRAMESET** element contains one or more **FRAMESET** or **FRAME** elements, along with an optional **NOFRAMES** element to provide alternate content for browsers that do not support frames or have frames disabled. A meaningful **NOFRAMES** element should always be provided and should at the very least contain links to the main frame or frames.

Syntax	<FRAMESET>...</FRAMESET>
Attribute	ROWS = <u>MultiLengths</u> (row lengths)
Specifications	COLS = <u>MultiLengths</u> (column lengths) ONLOAD = <u>Script</u> (all frames have been loaded) ONUNLOAD = <u>Script</u> (all frames have been removed) <u>core attributes</u>
Contents	One or more <u>FRAMESET</u> and <u>FRAME</u> elements, as well as an optional <u>NOFRAMES</u>
Contained in	<u>HTML</u>

The **FRAMESET** element is a *frame container* for dividing a window into rectangular subspaces called *frames*. In a Frameset document, the outermost **FRAMESET** element takes the place of **BODY** and immediately follows the HEAD.

The **FRAMESET** element contains one or more **FRAMESET** or **FRAME** elements, along with an optional **NOFRAMES** element to provide alternate content for browsers that do not support frames or have frames disabled. A meaningful **NOFRAMES** element should always be provided and should at the very least contain links to the main frame or frames.

The **ROWS** and **COLS** attributes define the dimensions of each frame in the set. Each attribute takes a comma-separated list of lengths, specified in pixels, as a percentage, or as a relative length. A relative length is expressed as *i** where *i* is an integer. For example, a frameset defined with **ROWS = "3*,*"** (* is equivalent to 1*) will have its first row allotted three times the height of thesecond row.

The values specified for the **ROWS** attribute give the height of each row, from top to bottom. The **COLS** attribute gives the width of each column from left to right. If **ROWS** or **COLS** is omitted, the implied value for the attribute is **100%**. If both attributes are specified, a grid is defined and filled left-to-right then top-to-bottom.

<Frame>

Syntax	<FRAME>
Attribute	NAME = <i>CDATA</i> (<u>name of frame</u>)
Specifications	SRC = <i>URI</i> (<u>content of frame</u>)
	LONGDESC = <i>URI</i> (<u>long description of</u> frame)
	FRAMEBORDER = [1 0] (frame border)
	MARGINWIDTH = <i>Pixels</i> (<u>margin width</u>)
	MARGINHEIGHT = <i>Pixels</i> (<u>margin</u> height)
	NORESIZE (disallow frame resizing)
	SCROLLING = [yes no <i>auto</i>] (ability to scroll)
	<u>core attributes</u>
Contents	Empty
Contained in	<u>FRAMESET</u>

The **FRAME** element defines a *frame*--a rectangular subspace within a Frameset document. Each **FRAME** must be contained within a **FRAMESET** that defines the dimensions of the frame.

The **SRC** attribute provides the URI of the frame's content, which is typically an HTML document. If the frame's content is an image, video, or similar object, and if the object cannot be described adequately using the **TITLE** attribute of **FRAME**, then

authors should use the **LONGDESC** attribute to provide the URI of a full HTML description of the object.

For better accessibility to disabled users and better indexing with search engines, authors should not use an image or similar object as the content of a frame. Rather, the object should be embedded within an HTML document to allow the indexing of keywords and easier provision of alternate content.

The **NAME** attribute gives a name to the frame for use with the **TARGET** attribute of the **A**, **AREA**, **BASE**, **FORM**, and **LINK** elements. The **NAME** attribute value must begin with a character in the range A-Z or a-z.

The **NAME** should be human-readable and based on the content of the frame since non-windows browsers may use the **NAME** as a title for presenting a list of frames to the user. Foreexample, **NAME = left** would be inappropriate since it says nothing about the content while **NAME = nav** would be inappropriate since it is not very human-readable. More suitable would be **NAME =Content** and **NAME = Navigation**. The **TITLE** attribute can also be used to provide a slightly longer title for the frame, though this is not widely supported by current browsers.

The **FRAMEBORDER** attribute specifies whether or not the frame has a visible border. The default value, **1**, tells the browser to draw a border between the frame and all adjoining frames. The value **0** indicates that no border should be drawn, though borders from other frames will override this.

The **MARGINWIDTH** and **MARGINHEIGHT** attributes define the number of pixels to use as the left/right margins and top/bottom margins, respectively, within the frame. The value must be non-negative.

The boolean **NORESIZE** attribute prevents the user from resizing the frame. This attribute should never be used in a user-friendly Web site.

The **SCROLLING** attribute specifies whether scrollbars are provided for the frame. The default value, **auto**, generates scrollbars only when necessary. The value **yes** gives scrollbars at

all times, and the value **no** suppresses scrollbars--even when they are needed to see all the content.

Example1

```
<!DOCTYPE html>
<html>

<frameset cols="25%,*,25%">
<frame src="first.html">
<frame src="second.html">
<frame src="third.html">
</frameset>

</html>
```

Example2

```
<HEAD>
<FRAMESET ROWS="25%,50%,25%"
    <FRAME SRC=" ">
<FRAMESET COLS="25%,*">
    <FRAME SRC=" ">
    <FRAME SRC=" ">
    </FRAMESET>
    <FRAME SRC=" ">
</FRAMESET>
</HEAD>
```

Targets

- When you use links for use in a frames environment you will need to specify an additional attribute called TARGET.
- The TARGET attribute uses the NAME attribute of the FRAME element.
- There are **4** special target names that cannot be assigned by the NAME attribute of the FRAME tag.
 1. **TARGET=“_top”** : This loads the linked document into the full browser window with the URL specified by the HREF attribute. All frames disappear, leaving the new linked page to occupy the entire window. The back is turned on.
 2. **TARGET=“_blank”** : Opens an unnamed new browser window and loads the document specified in the URL attribute into the new window (and your old window stays open). The back is turned off. Other windows remains on.
 3. **TARGET=“_self”** : Loads the document in the same window where the anchor was {Clicked}. This is the **default** setting for linking elements.

4. TARGET=“_parent” : the _parent frame is a prior frameset that the current frameset was “spawned” from. If there isn’t one it is the browser window. The document is loaded into the area occupied by the columns or rows frameset containing the frame that contains the link. The back is turned on. All windows disappear.

FORMS

An HTML form is a section of a document containing normal content, markup, special elements called *controls* (checkboxes, radio buttons, menus, etc.), and labels on those controls. Users generally “complete” a form by modifying its controls (entering text, selecting menu items, etc.), before submitting the form to an agent for processing (e.g., to a Web server, to a mail server, etc.)

Controls

Users interact with forms through named *controls*.

A control's “*control name*” is given by its name attribute. The scope of the name attribute for a control within a FORM element is the FORM element.

Each control has both an initial value and a current value, both of which are character strings. Please consult the definition of each control for information about initial values and possible constraints on values imposed by the control. In general, a control's “*initial value*” may be specified with the control element's value attribute. However, the initial value of a TEXTAREA element is given by its contents, and the initial value of an OBJECT element in

a form is determined by the object implementation (i.e., it lies outside the scope of this specification).

A control's initial value does not change. Thus, when a form is reset, each control's current value is reset to its initial value. If a control does not have an initial value, the effect of a form reset on that control is undefined.

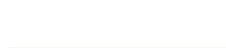
When a form is submitted for processing, some controls have their name paired with their current value and these pairs are submitted with the form. Those controls for which name/value pairs are submitted are called successful controls.

Text Fields

`<input type = "text" />` defines a one-line input field that a user can enter text into:

`<form>`

First name: `<input type = "text" name = "firstname" />``
`Last name: `<input type = "text" name = "lastname" />``</form>`



Note: The form itself is not visible. Also note that the default width of a text field is 20 characters.

`<input>`

This is the tag name for many of the form elements that are there to collect user input. type

The value of this attribute decides which of the input elements this one is. In this case, text is telling the browser that it's a single-line text-box.

name

size

This specifies the length of the box on your page. If the box is not wide enough for the information that is entered, it will scroll across to allow more letters, but you should tailor this to the type of information being asked for so that most people can see their whole response at once.

Password Field

`<input type = "password" />` defines a password field:

`<form>`

Password: `<input type = "password" name = "pwd" />``</form>`

Note: The characters in a password field are masked (shown as asterisks or circles).

These three elements give the reader a choice of options, and asks them to pick out one or more of them.

Radio Buttons

These small circular buttons information forms to ask the user their up a group of them, you can only select **one choice**

1. ☐ 2. ☐ 3. ☐

The code for a radio button is:

```
<input type = "radio" name = "choices" value = "choice1">
```

`<input type = "radio" />` defines a radio button. Radio buttons let a user select ONLY ONE one of a limited number of choices:

```
<form>
```

```
<input type = "radio" name = "gender" value = "male" /> Male<br /><input type = "radio" name = "gender" value = "female" /> Female </form>
```

How the HTML code above looks in a browser:

Male ☐

Female ☐

Check Boxes

Groups of check boxes are similar to radio buttons except they are not grouped, so **multiple boxes can be selected at the same time**. They are small squares that are marked with a tick when selected. Here's a few to play with:

1. ☐ 2. ☐ 3. ☐

The code for these boxes is the same as for the radio buttons, with just the value of the type attribute changed:

```
<input type = "checkbox" name = "checkbox1">
```

`<input type = "checkbox" />` defines a checkbox. Checkboxes let a user select ONE or MORE options of a limited number of choices.

```
<form>
```

```
<input type = "checkbox" name = "vehicle" value = "Bike" /> I have a bike<br/>
```

```
<input type = "checkbox" name = "vehicle" value = "Car" /> I have a car  
</form>
```

How the HTML code above looks in a browser:

I have a bike ☐

I have a car ☐

Notice that there is no value attribute for checkboxes, as they will either be checked or left blank. If you want a checkbox to be checked before the user gets to modify it, add the boolean checked attribute:

```
<input type = "checkbox" name = "newsletter" checked = "checked">
```

. This checked attribute can also be used on a radio button to set one of the radios as selected by default.

Drop-down Select Boxes

Using this control a user to select an option. They perform the same thing as radio buttons, it's just the way they look that's different.



```
<select name = "continent" size = "1">  
<option value = "Europe">Europe</option>  
<option value = "America"> America</option>  
<option value = "asia">asia</option>  
<option value = "africa">africa</option>  
</select>
```

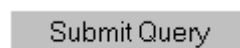
select boxes are like textareas — they have their own tag, but these elements hold further tags inside them too. Each choice you give your reader is denoted by an option. The name/value system remains from the tags above. The size attribute sets how many lines of options are displayed.

Submit Button

Every set of Form tags requires a Submit button. This is the element causes the browser to send the names and values of the other elements to the CGI Application specified by the ACTION attribute of the FORM element.

```
<INPUT TYPE="SUBMIT">
```

The browser will display



Submit has the following attributes:

- **TYPE:** submit.
- **NAME:** value used by the CGI script for processing.
- **VALUE:** determines the text label on the button, usually Submit Query.

A submit button is used to send form data to a server. The data is sent to the page specified in the form's action attribute. The file defined in the action attribute usually does something with the received input.

Reset Button It is a good idea to include one of these for each form where users are entering data. It allows the surfer to clear all the input in the form.

<INPUT TYPE="RESET">

Browser will display



Reset buttons have the following attributes:

TYPE: reset.

VALUE: determines the text label on the button, usually Reset.

```
<form name = "input" action = "html_form_action.asp" method = "get">
Username: <input type = "text"      />
<input type="Submit" />
</form>
```

If you type some characters in the text field above, and click the "Submit" button, the browser will send your input to a page called "html_form_action.asp". The page will show you the received input.

Table tag

A table can be inserted in a web page using table tag

The <TABLE></TABLE> element has four sub-elements:

1. Table Row <TR></TR>.
2. Table Header <TH></TH>.
3. Table Data <TD></TD>.
4. Caption <CAPTION></CAPTION>.

The table row elements usually contain table header elements or table data elements.

Example

```
<table border="1">
<tr>
<th> Column 1 header </th>
<th> Column 2 header </th>
</tr>
<tr>
<td> Row1, Col1 </td>
<td> Row1, Col2 </td>
</tr>
<tr>
<td> Row2, Col1 </td>
<td> Row2, Col2 </td>
</tr>
</table>
```


Tables Attributes

- **BGColor:** Some browsers support background colors in a table.
- **Width:** you can specify the table width as an absolute number of pixels or a percentage of the document width. You can set the width for the table cells as well.
- **Border:** You can choose a numerical value for the border width, which specifies the border in pixels.
- **CellSpacing:** Cell Spacing represents the space between cells and is specified in pixels.
- **CellPadding:** Cell Padding is the space between the cell border and the cell contents and is specified in pixels.
- **Align:** tables can have left, right, or center alignment.
- **Background:** Background Image, will be titled in IE3.0 and above.
- **BorderColor, BorderColorDark.**

A table caption allows you to specify a line of text that will appear centered above or below the table.

<TABLE BORDER=1 CELLPADDING=2>

<CAPTION ALIGN="BOTTOM"> Label For My Table </CAPTION>

The Caption element has one attribute ALIGN that can be either TOP (Above the table) or BOTTOM (below the table).

Table Data and Table Header Attributes

- **Colspan:** Specifies how many cell columns of the table this cell should span.
- **Rowspan:** Specifies how many cell rows of the table this cell should span.
- **Align:** cell data can have left, right, or center alignment.
- **Valign:** cell data can have top, middle, or bottom alignment.
- **Width:** you can specify the width as an absolute number of pixels or a percentage of the document width.
- **Height:** You can specify the height as an absolute number of pixels or a percentage of the document height.

Basic Table Code

Example:

```
<TABLE BORDER=1 width=50%>
<CAPTION><h1>Spare Parts </Caption>
<TR><TH>Stock Number</TH><TH>Description</TH><TH>List Price</TH></TR>
<TR><TD bgcolor=red>1234-AB</TD><TD>56mm Socket</TD><TD>45.00</TD></TR>
<TR><TD>3478-AC</TD><TD><font color=blue>78mm
Socket</font></TD><TD>47.50</TD></TR>
<TR><TD>3480-AB</TD><TD>80mm Socket</TD><TD>50.00</TD></TR>
</TABLE>
```

Example

```
<table border="1" cellpadding="2">
<tr><th> Column 1 Header</th><th> Column 2 Header</th></tr>
<tr><td colspan="2"> Row 1 Col 1</td></tr>
<tr><td rowspan="2">Row 2 Col 1</td>
<td> Row 2 Col2</td></tr>
<tr><td> Row 3 Col2</td></tr>
</table>
```

Special Things to Note

- **TH, TD and TR should always have end tags.**
Although the end tags are formally optional, many browsers will mess up the formatting of the table if you omit the end tags. In particular, you should ***always*** use end tags if you have a TABLE within a TABLE -- in this situation, the table parser gets hopelessly confused if you don't close your TH, TD and TR elements.
- **A default TABLE has no borders**
By default, tables are drawn without border lines. You need the BORDER attribute to draw the lines.
- **By default, a table is flush with the left margin**
TABLEs are plopped over on the left margin. If you want centered tables, You can either: place the table inside a DIV element with attribute ALIGN="center". Most current browsers also supports table alignment, using the ALIGN attribute. Allowed values are "left", "right", or "center", for example: <TABLE ALIGN="left">. The values "left" and "right" float the table to the left or right of the page, with text flow allowed around the table. This is entirely equivalent to IMG alignment

UNIT 4

CASCADING STYLE SHEETS

The usefulness of style sheets

Types of Style sheets

Creating style sheets

Common tasks with CSS

Font Family: Font Metrics, Units

Properties

Classes and Pseudo classes

CSS tags

WHAT IS CSS STYLE?

While Web site visitors demand more attractive, fast loading, and interesting sites, traditional formatting and page layout are no longer efficient enough to handle more complex design requirements. As a simple example, imagine a page with hundreds of lines and more than 50 paragraphs. Each paragraph is to be formatted by the traditional font face and size attributes. It would be a nightmare to make any changes. Therefore a structural cascading mechanism is urgently needed. To rescue this reusability crisis, W3C came up with an elegant solution called the Cascading Style Sheet (CSS). It is a structure that separates formatting features from the contents of a page.

Using the <style> element

The <style> element behaves like other HTML elements. It has a beginning and ending tag and everything in between is treated as a style definition. As such, everything between the <style> tags needs to follow style definition guidelines. A document's <style> section must appear inside the document's <head> section, although multiple <style> sections are permissible.

The <style> tag has the following, minimal format:

```
<style type="text/css">
```

```
... style definitions ...
```

```
</style>
```

CSS works by specifying the element you want to modify, and stating how you want it to be displayed by the Web browser. For example, a typical CSS may look like this:

```
<style>
h2 {color:red;font-family:arial;font-size:12pt}
</style>
```

This CSS defines the characteristics or style for the second-level headers (i.e., <h2>). In this case, the text within the element <h2> will be displayed using the Arial font, 14pt, and red color. More importantly, the style h2 can be cascaded over by subsequent CSS definitions.

CSS is the term used to broadly refer to several style methods of applying style elements to HTML pages. These are the inline style, internal (embedded) style, and external style sheets. A style is simply a set of formatting instructions that can be applied to a piece of text.

Styles define how to display HTML elements. The results are better font control, color management, margin control, and even the addition of special effects such as text shading. Multiple style definitions will cascade into one. This means that the first is overridden by the second, the second by the third, and so on.

Since the beginning of HTML usage for web page creation, people have realized the need to separate the way the page looks and the actual content it displays. Even the first versions of HTML have supported different ways to present text using **FONT**, **B** (bold) or **I** (italic) tags. Those HTML elements still exist today, but their capabilities are far below what Web pages should provide.

As we've already said, CSS enables you to separate the layout of the Web page from its content. This is important because you may want the content of your web page to change frequently (for example, a current events page) but not the design/layout, or vice versa. It is a standard of the World Wide Web Consortium (W3C), which is an international Web standards consortium. Practically, all the style and layout guidelines for a website are kept in CSS files that are separate from the HTML files which contain the data, text and content for a website. Simply put, when talking

about displaying Web pages in the browser, HTML answers the question "What?", while CSS answers "How?". When using CSS, you are defining how to display each element of the page. You can, for example, say to show all text in **DIV** elements in blue color, to have all links italic and bold, etc. With CSS you can also define classes, which tell the browser how to display all elements of that class. Maybe you're asking yourself, why bother with CSS? Isn't it much simpler and faster to define everything inside the HTML page? Using HTML tags and attributes, you can modify the style of each element on your page.

But what if you have a Web site with a larger number of pages, let's say 50? Imagine the process of setting the style for each element on your 50 pages. And then, if later on down the road you want to change the font style, you'll have to manually go through each file and change all the HTML elements. You can count on a very long, boring and tiring process! With CSS you can put all the information about displaying HTML elements in a separate page. Then you can simply connect this CSS file with all pages of your Web site, and – all the pages will follow the same guidelines. Change the CSS file, and you have indirectly changed all pages of your Web site. In addition, you get much greater design capabilities with CSS, as we will show in this guide.

Use of Style Sheet

Understanding the Style Sheet Cascade

The concept behind Cascading Style Sheets is essentially that multiple style definitions can trickle, or cascade, down through several layers to affect a document. Several layers of style definitions can apply to any document. Those layers are applied in the following order:

- The user agent settings (typically, the user is able to modify some of these settings)
- Any linked style sheets
- Any styles present in a <style> element
- Styles specified within a tag's style attribute

Each level of styles overrides the previous level where there are duplicate properties being defined. For example, consider the following two files:

mystyle.css

```
/* mystyle.css - Styles for the main site */
h1, h2, h3, h4 { color: blue; }
h1 { font-size: 18pt; }
h2 { font-size: 16pt; }
h3 { font-size: 14pt; }
h4 { font-size: 12pt; }
p { font-size: 10pt; }
```

index.html

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

```
<head>
<link rel="stylesheet" type="text/css"
href="mystyle.css" />
<style type="text/css">
h1 { color: Red; }
</style>
</head>
<body>
<h1>A Sample Heading</h1>
```

...

What color will the <h1> heading in index.html be? The external style specifies blue, but the style element specifies red. In this case, the internal style takes precedence and the <h1> text will appear in red.

How do I use CSS?

Let's get started with using style sheets. CSS data is actually plain text written in a specific way. Let's take a look at the contents of a sample CSS file:

It is actually completely readable – this style sheet defines that all content within the HTML **BODY** element will use font Verdana with size of 9 points and will align it to the right. But, if there's a **DIV** element, the text within that will be written in font Georgia. We're also using a class named "important" (classes use

notation, which we will cover later on). All elements of this class will have a set background color, a border and will use Franklin Gothic Book font. As you see, style definitions for a certain element or class are written inside curly braces (“{ }”) and each line ends with a semicolon “;”.

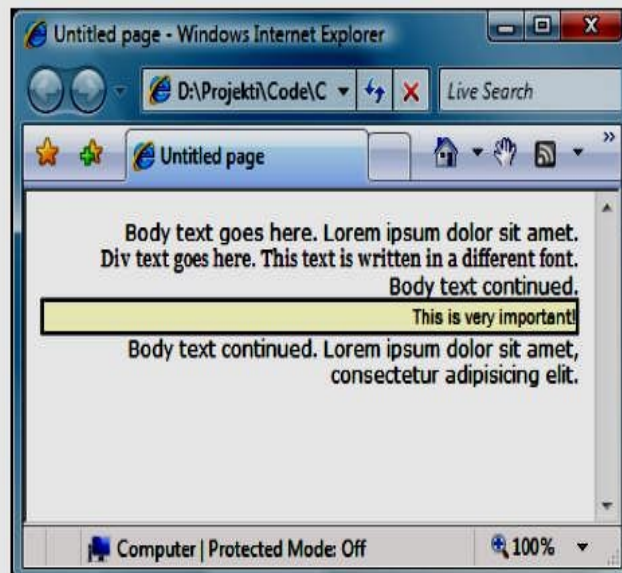
```
body
{
    font-family: Verdana;
    font-size: 9pt;
    text-align: right;
}
div
{
    font-family: Georgia;
}
.important
{
    background-color: #ffffde;
    border: thin black ridge;
    font-family: Franklin Gothic Book;
}
```

Now is the perfect time to explain the scoping of styles. All CSS definitions are inheritable – if you define a style for **BODY** element, it will be applied to all of its children, like **P**, **DIV**, or **SPAN** elements. But, if you define a style for **DIV** element, it will override all styles from its parent. So, in this case, the **DIV** element text would use font Georgia size 9 points and would be aligned to the right. As you see, **DIV** style definition for the font family has overridden **BODY** style definitions. This goes on – if you have a **DIV** element which is also of class "important", the class definition will override **DIV** style definitions. In this case, such **DIV** element would have a background color set, a border, it would use font Franklin Gothic Book size 9 points and be aligned to the right.

Here are the elements that would be affected by the sample CSS file.

```
<html>
...
<body>
Body text goes here. Lorem ipsum dolor sit amet.
<div>Div text goes here. This text is written in a different font.</div>
Body text continued.
<div class="important">This is very important!</div>
Body text continued. Lorem ipsum dolor sit amet, consectetur adipisicing
elit.
</body>
</html>
```

And, of course, the browser would show this content as follows.



Write it in notepad. Copy and paste the CSS sample above into this file and save this file as "style.css" into a folder on your computer. Now select File | New File and choose "HTML Page". Also save this HTML page into the same folder on your computer. Insert the following code into the HTML page.

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

This code should be put within the HTML page header, within **HEAD** element. As you see, *href* attribute defines which CSS file to use. Put this **LINK** element within all HTML pages you wish to apply styles to and you're done!

CSS data doesn't necessarily have to be in a separate file. You can define CSS styles inside of a HTML page. In this case, all CSS definitions have to be inside a **STYLE** element. This approach can be used to define the looks of elements that are specific to a certain page and will not be reused by other pages. Take a look at how that HTML page might look:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<html>
  <head>
    <title>My title</title>
    <style type="text/css">
      body
      {
        font-family: Verdana;
        font-size: 9pt;
        text-align: right;
      }
      div
      {
        font-family: Georgia;
      }
      .important
      {
        background-color: #ffffde;
```

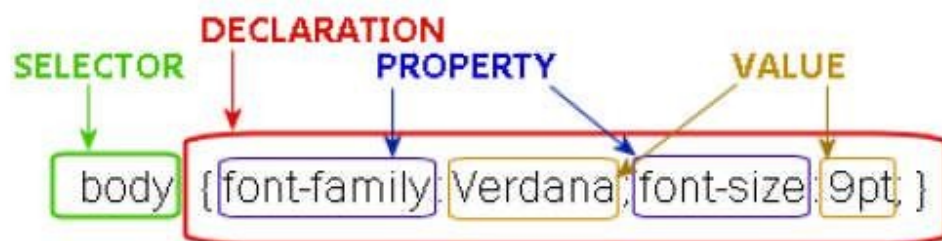
```
        border: thin black ridge;
        font-family: Franklin Gothic Book;
      }
    </style>
  </head>
  <body>
    <div class="important">My content</div>
  </body>
</html>
```

Notice that in this example you can see how to define an element of a specific class – just add *class* attribute and set its value. All classes within CSS style definitions are prefixed with a dot ("."). The third way to define a CSS style, in addition to the previously explained methods of a separate CSS file, and the **STYLE** element within the HTML page header, is inside of a specific HTML element. To do this, you need to use the *style* attribute. Take a look at the following example:

```
<span style="font-family: Tahoma; font-size: 12pt;">My text</span>
```

All the text inside of this **SPAN** element will be displayed using 12 point Tahoma font. And note – when applying styles directly to elements, as in this last example, these style definitions will override all element definitions and class definitions previously set in a separate CSS file or inside of HTML page header **STYLE** element.

CSS style definition syntax



To be able to write CSS files and definitions correctly, you need to remember few simple rules. Although CSS syntax is rather logical and easy to learn, there are 6 basic things you need to know. First, take a look at the structure of a style definition.

And here are 6 rules of style definitions:

1. Every CSS definition has to have a selector and a declaration. The declaration follows the selector and uses curly braces.
2. The declaration consists of one or more properties separated with a semicolon.
3. Every property has a name, colon and a value.

4. A property can have multiple values separated with a comma (e.g. "Verdana, Arial, Franklin Gothic Book").
5. Along with a value, can also be a unit of measure (e.g. "9pt", where "pt" stands for *points*). No space is allowed between the value and the unit.

When writing CSS code, you can use whitespaces as needed – new lines, spaces, whatever makes your code more readable.

CREATING STYLE SHEET

Styles can be defined within your HTML documents or in a separate, external style sheet. CSS is the term used to broadly refer to several style methods of applying style elements to HTML pages. These are the *inline style*, *internal (embedded) style*, and *external style sheets*. A style is simply a set of formatting instructions that can be applied to a piece of text. You can also use both methods within the same document. The following sections cover the various methods of defining styles.

Inline style

They are basically inline styles. You can add inline style to any "sensible" HTML elements by using the style attribute in the associated element. The browser will then use the inline style definitions to format only the contents of that element. The style attribute can contain any CSS property. Example [ex02-01.htm](#) shows how to define the style of a document body and how to change its default definitions

Example: ex02.html - Inline CSS Style

```
<html>
<head><title> Inline CSS Style - ex02.html</title></head>
<body style="font-family:Times;font-weight: bold; background:#000088" >
<div style="font-size:20pt;text-align:center;color:#00ffff"> Inline CSS Style </div>
<p style="font-family:arial;font-size:16pt;color:#ffff00; margin-left:20px;margin-right:20px">
With CSS, we can control the margins of an element.
This is a paragraph with margin-left:20px and margin-right:20px.
</p>
</body>
</html>
```

In this example, the style attribute is used within the <body> element (line 6). The default font and background color are now set to bold Times and color value #000088 (dark blue) respectively. All CSS properties have to be included inside the double quotation marks of the style attribute and are separated by semi-colons.

The division element <div> in line 8 has all the CSS properties from <body> with some additional definitions. A division is similar to a paragraph but without an additional line break. Next to this division, there is a paragraph element <p> (line 10). This paragraph changes the default font family to "arial" and adds some margin controls.

When an element has two or more of the same CSS definitions, the earlier ones will be overridden by the latest one. That is, the styles will be cascaded into one.

Notice how you can call for a font using the font's name as well as point size. In CSS, you can also use points (pt), pixels (px), percentage (%), inches (in), and centimeters (cm) to control sizing and positioning of an element. As a good design habit, always include the measurement units in your page

The embedded style element <style>

In addition to inline style, there are also internal (or embedded) and external styles. External style is a separate file for CSS properties. Internal styles are usually defined within the <style> element. A typical example is

```
<style type="text/css">
h2 {color:#00ffff;font-size:20pt;text-align:center}
h4 {margin-left:70%}
body {font-family:arial;font-size:14pt;color:#ffff00; background-
image:url("backgr01.jpg")}
</style>
```

The browser will then read the style definitions and format the document accordingly.

A browser normally ignores unknown elements. This means that an earlier browser that does not support styles will ignore the <style> element, but the content of <style> will still be displayed on the page. It is possible to prevent an earlier browser from displaying the content by hiding it in the HTML comment symbols.

Example: ex03.html - The Style Element <style>

```
<html>
  <head><title> The Style Element <style> I - ex03.html</title></head>
  <style type="text/css">
    h2 {color:#00ffff;font-size:20pt;text-align:center}
    h4 {margin-left:70%}
    p {font-family:arial;font-size:16pt;color:#ffff00; margin-left:20px;margin-right:20px}
    body {font-family:arial;font-size:14pt;color:#ffff00; background-image: url("backgr01.jpg")}
  </style>
</head>
<body>
  <h2>Internal CSS Style</h2>
  <h4>This area was created by CSS margin margin-left:70% and margin-right:20%</h4>
  <p>With CSS, you can control text font, color, dimension, position,
margin, background and much more ...</p>
</body>
</html>
```

As can be seen from this example, with CSS styles you have precise control over how you would like your text to be displayed. There are also a number of CSS elements that can take a URL. In CSS, the URL should be contained within round brackets, immediately preceded by the statement URL without an equals sign as illustrated in line 12.

Another useful aspect of the CSS style is the inline keyword class. This gives you ways of breaking down your style rules into very precise pieces to provide a lot of variety. You define a style class by putting a dot in front of a CSS definition. This class style can be used in almost any XHTML element with attribute class= and the unique class name.

Example [ex02-03.htm](#) defines two CSS classes. One of them is dedicated to defining a button on your browser window.

Example: ex02-03.html - The Style Element <style> II

```
<html>
<head><title> The Style Element <style> II - ex02-03.html</title></head>
<style type="text/css">
.txtSt {font-family:arial;color:#ffff00;font-size:20pt; font-weight:bold}
    .butSt {background-color:#aaffaa;font-family:arial;font-weight:bold;
        font-size:14pt;color:#008800;width:240px;height:30px} </style>
</head>
<body style="background:#000088;text-align:center">
<div class="txtSt">Internal CSS Style Example II</div><br/>
<input type="button" class="butSt" value="CSS Style Button" />
</body>
</html>
```

External CSS style sheets

An external style sheet is ideal when the style is applied to many pages. The style information is placed in a separate document with the file extension .css. With an external style sheet, you can change the look of an entire Web site by changing the corresponding style information file. Each page must link to the style sheet using the <link> element, which usually goes within the <head> section. For example,

```
<head>
<link rel="stylesheet" type="text/css" href="ex02-04.css"></head>
```

The browser will read the style definitions from the external CSS file [ex02-04.css](#) and format the document accordingly.

An external style sheet can be written in any text editor and should be saved with the file extension .css. You should also be sure either that this file is in the root directory

with the HTML files that you intend to process or that the link is coded appropriately. An example of a style sheet file is shown below.

The following is an example of an external style sheet at work

Example: ex02-04.html - External CSS Style

```
<html>
<head><title> External CSS Style - ex02-
04.htm</title></head>
<link rel="stylesheet" type="text/css" href="ex02-04.css">
</head>
<body>
<div style="text-align:center;color:#00ffff">
External CSS File</div><br /><br />
<div>
  This is a paragraph defined by the division element <div>with
margin-left:20% and margin-right:20%</div>
<hr>
<div>
  This is another paragraph defined by the division element and separated
by a horizontal line. All CSS properties are defined in the
external CSS
file: ex02-04.css
</div>
</body>
</html>
```

This page includes a link to an external style sheet called [ex02-04.css](#). This file defines all the default formatting features used inside the page. The corresponding external CSS style sheet [ex02-04.css](#) is given next:

Example: ex02-04.css - External CSS File For ex02-04.html

```
hr {color: sienna}
div {margin-left:20px; margin-right:20px; color:#ffff00} body {background-image:
url("backgr01.jpg");
font-family:arial; font-size:14pt;color:#ffff00; font-weight:bold}
```

Any page containing this link adopts the styles defined in the external style sheet [ex02-04.css](#). In this example, the horizontal rule line <hr> is changed to the color sienna. Additional margin control is added to the <div> element and the element <body> is given a different style definition. Bold "arial" and color value #ffff00 in a font size of 14 points are used as default attributes. A background image backgr01.jpg is also added to specify graphics as background images.

```
/* mystyles.css - Styles for the main site */
h1, h2, h3, h4 { color: blue; }
h1 { font-size: 18pt; }
h2 { font-size: 16pt; }
h3 { font-size: 14pt; }
h4 { font-size: 12pt; }
p { font-size: 10pt; }
```

Tip You can include comments in your styles to further annotate your definitions. Style comments begin with a /* and end with a */. Comments can span several lines, if necessary.

FONT FAMILY & PROPERTIES

FONT FAMILY

Working with Font Styling Attributes

There are several styling attributes to control such characteristics as font families, sizes, bolding, and spacing.

Naming font families using CSS

CSS provides a mechanism for rendering font families in a browser if those fonts are installed on a user's system. This is accomplished by creating either an inline style on an element such as a `td` or `span` element, or by creating a class rule selector within the style element. Either way, the syntax is the same, with a list of font family names, each separated by a comma, contained within a set of braces:

```
font-family {Arial, Helvetica, sans-serif;}
```

The browser will look first for the Arial font in the preceding example, then the Helvetica font, then the "default" sans-serif font, which is whatever sans-serif font the user's operating system defaults to. If you name a font family with spaces between characters, you need to enclose the name in quotes, as shown in bold in the following:

```
.myFontClass {font-family: 'Helvetica Narrow', sans-serif}
```

In practice, it may be a good idea to use quotes even when there are no spaces between characters, because some versions of Netscape 4 have trouble recognizing font names otherwise.

Using Class Selector and Inline Style to Name a Font Family

```
<html>
<head>
<title>Font sizes</title>
<style type="text/css">
<!--
.myFontClass {font-family: "Helvetica Narrow", sans-serif}

-->
</style>
</head>
<body>
<p>This is an <span style="font-family: 'Helvetica Narrow',
sans-serif">inline</span> style.</p>
<p>This uses a <span class="myFontClass">class
selector</span></p>
</body>
</html>
```

The first bolded line shows a class selector named `myFontClass`, which is called by a `span` element's class attribute (the last bolded code fragment). Figure 18-5 shows the results from rendering Listing 18-3 in the browser.

7.1.3 Working with font styles

In traditional HTML, you can choose whether you want your font to appear in Roman style (non-italic) font or italics by using or not using the `em` or `i` elements: Emphasizing a point with the `em element` or the `<i>i element</i>`.

The preceding code fragment results in the following in a browser: Emphasizing a point with the *em element* or the *i element*.

If you want to really be sure even the earliest of browsers recognize your italics, `em` is the way to go. More importantly, it's a better choice because aural browsers should **emphasize** the contents of this element to sight-impaired users of your Web site. For this reason, this is one of the rare exceptions to the rule of using CSS for styling over HTML elements. However, there's nothing wrong with using both. To use italics in CSS, simply include the following either inline or in a rule selector: `font-style: italic`

Note Be sure to call it "italic," not "italics" with an s. You can also use `font-style: oblique`, but older versions of Netscape will not recognize it.

Establishing font sizes

Managing font size can be tricky even with CSS, but most developers seem to agree that the most reliable unit of measurement in CSS is the pixel. To establish size using CSS, you simply name the property in your selector or inline style rule:

```
.twelve {font-size: 12px}  
H1 {font-size: xx-large}  
.xsmall {font-size: 25%}
```

In the preceding code fragment, three style rules are created, each with its own font size. The first creates a relative size using pixels as the unit of measure. Never spell out the word pixels in your style definition. Always use the form px. px is the most reliable unit of measure because it is based on the user's screen size, and the pixel resolution of his or her monitor. It also has virtually bug-free support across all browsers that support CSS.

Other relative sizes include the following:

em, for ems, is based on the em square of the base font size, so that 2em will render a font twice as large as your document's base font size. Support in Netscape 4 and IE3 is awful.

ex is based on the X height of the base font size, so that 2ex will render a font whose X character is twice as tall as the X character at the default, or base, font size. This is a meaningless unit in the real world, because support is either nonexistent or so poor as to make it worthless.

The next line in the preceding code fragment sets an absolute size called xx-large, although it isn't absolute among browsers, only the one browser your user is using to render the page. xx-large is part of a larger collection that includes the following possible values:

xx-small, x-small, small, medium, large, x-large, xx-large Other absolute sizes include the following:

pt for points. This is appropriate for pages that are used for printing, but is not a particularly reliable measure for managing screen-based fonts.

in (inches), cm (centimeters), mm (millimeters), and pc (picas) are all rarely used on the Web, because they're designed with print production in mind.

Finally, you can create a font size using a percentage by simply adding the % character next to the actual size. This will render the font x% of the base size. You can experiment with font sizes by modifying below code.

Creating Font Sizes with CSS and the Font Element's Size Attribute

```
<html>
<head>
<title>Font sizes</title>
<meta http-equiv="Content-Type" content="text/html;
charset=iso-8859-1">
<style type="text/css">
<!--
.12pixels {font-size: 12px;}
.13pixels {font-size: 13px;}
.14pixels {font-size: 14px;}
.15pixels {font-size: 15px;}
.16pixels {font-size: 16px;}
.17pixels {font-size: 17px;}
.18pixels {font-size: 18px;}
.sans-serif {font-family: Frutiger, Arial, Helvetica, sansserif;}
.sans-serif-b {font-family: Frutiger, Arial, Helvetica, sansserif;
font-weight: 900;}
-->
</style>
</head>
<body>
<table width="100%" border="0" cellspacing="0" cellpadding="5" style="border: #cccccc
thin solid"><tr align="left" valign="top" bgcolor="#D5D5D5" >
<td width="26%" valign="bottom" class="sans-serif-b">Font Size</td>
<td width="29%" valign="bottom" class="sans-serifb"> Font Size +</td>
<td width="17%" valign="bottom" class="sans-serifb"> Font Size -</td>
```

```

<td width="28%" valign="bottom" class="sans-serifb">
CSS</td>
</tr>
<tr align="left" valign="top">
<td><p><font size="1">Font Size = 1</font></p></td><td><font size="+1">Font Size =
+1</font></td><td><font size="-1">Font Size = -1</font></td><td class="12pixels">font-
size: 12px</td>
</tr>
<!-- Additional rows of all the font-sizes here - download actual code to view all rows --
></table>

<p>&nbsp;</p>
</body>
</html>

```

Bolding fonts by changing font weight

Font weight refers to the stroke width of a font. If a font has a very thin, or light, stroke width, it will have a weight of 100. If it has a thick, or heavy, stroke width, it will be 900. Everything else is inbetween. To denote font width, you use a numeric set of values from 100 to 900 in increments of 100: 100, 200, 300, 400, and so on. Or, you can use the keywords bold, normal, bolder or lighter to set a value, which will be relative to the font weight of the element containing the font. The keyword bold is equal to the numeric value

An example of using font-weight in style rules written for a style element might be as follows:

```

p {font-weight: normal}
p.bold {font-weight: 900}

```

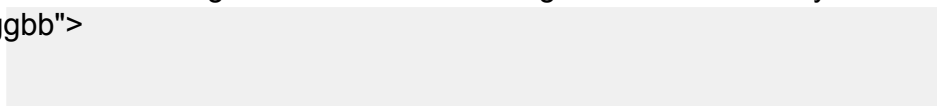
Making font wider or thinner using font stretch

This font property is supposed to allow you to make a font look fatter or thinner.

Background color and image

The CSS background element allows you to add a background color or image to your Web page. For example, you may like to use a dark color to set a background against light-colored paragraphs to create an effect of sidebars or offsetting text for emphasis.

The CSS element <background-color> takes the general format <b style="background-color:#rrggbb">



your body text here ...

The following example [ex02-10.htm](#) shows some simple background-color effects:

Example: ex02-10.html - Background Color

```
<html>
<head><title>Background Color ex02-10.htm</title></head>
<body style="background:#f0fff0">
<div style="font-family:arial,times,serif; font-size:20pt; font-weight:bold;text-align:center"> Background Color <br />Demo</div><br />
<div style="background-color:#00ffff;font-family:'Comic Sans MS',
times; font-size:20pt;color:#ff0000">
This text will appear in red in a small box with cyan
background on a larger honeydew background
</div><br />
</body>
</html>
```

Positioning a background image

You can further control the position at which a background image begins to tile on your Web page. This is all done by the CSS element background-position. It takes the general form

```
<body style="background-image:url (bg_image.gif) background-position: x y">
```

where x y represents the position of the image. Note that with the IE4 and NS4 browsers, tiling only happens down and to the right.

CLASSES AND CSS TAG

Classes and Pseudo Classes
CSS Tag.

CSS PSEUDO-CLASSES

Syntax

The syntax of pseudo-classes:

```
selector:pseudo-class {property:value;}
```

CSS classes can also be used with pseudo-classes:

```
selector.class:pseudo-class {property:value;}
```


8.1.1 Anchor Pseudo-classes

Links can be displayed in different ways in a CSS-supporting browser:

Example

```
a:link {color:#FF0000;} /* unvisited link */ a:visited {color:#00FF00;} /* visited link */  
a:hover {color:#FF00FF;} /* mouse over link */ a:active {color:#0000FF;} /* selected link  
*/
```

Note: a:hover MUST come after a:link and a:visited in the CSS definition in order to be effective!!

Note: a:active MUST come after a:hover in the CSS definition in order to be effective!!

Note: Pseudo-class names are not case-sensitive.

Pseudo-classes and CSS Classes

Pseudo-classes can be combined with CSS classes:

```
a.red:visited {color:#FF0000;}
```

```
<a class="red" href="css_syntax.asp">CSS Syntax</a>
```

If the link in the example above has been visited, it will be displayed in red.

CSS - The :first-child Pseudo-class

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

Note: For :first-child to work in IE a <!DOCTYPE> must be declared.

Match the first <p> element

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

Example

```
<html>
<head>
<style type="text/css">
p:first-child
{
color:blue;
}
</style>
</head>

<body>
<p>I am a strong man.</p>
<p>I am a strong woman.</p>
</body>
</html>
```

Match the first <i> element in all <p> elements

In the following example, the selector matches the first <i> element in all <p> elements:

Example

```
<html>
<head>
<style type="text/css">
i:first-child
{
font-weight:bold;
}
</style>
</head>

<body>
<p>I am a <i>strong</i> man. I am a <i>strong</i> man.</p>
<p>I am a <i>strong</i> man. I am a <i>strong</i> man.</p>
</body>
</html>
```

Match all <i> elements in all first child <p> elements

In the following example, the selector matches all <i> elements in <p> elements that are the first child of another element:

Example

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

CSS ID AND CLASS

The id and class Selectors

In addition to setting a style for a HTML element, CSS allows you to specify your own selectors called "id" and "class".

8.2.1 The id Selector

The id selector is used to specify a style for a single, unique element.

The id selector uses the id attribute of the HTML element, and is defined with a "#".

The style rule below will be applied to the element with id="para1":

Example

```
#para1
{
text-align:center;
color:red;
}
```

Do **NOT** start an ID name with a number! It will not work in Mozilla/Firefox.

The class Selector

The class selector is used to specify a style for a group of elements. Unlike the id selector, the class selector is most often used on several elements.

This allows you to set a particular style for any HTML elements with the same class.

The class selector uses the HTML class attribute, and is defined with a ".".

In the example below, all HTML elements with class="center" will be center-aligned:

Example

```
.center {text-align:center;}
```

You can also specify that only specific HTML elements should be affected by a class.

In the example below, all p elements with class="center" will be center-aligned:

Example

```
p.center {text-align:center;}
```

Example:

```
<html>
<head>
<style type="text/css">
.para
{
font-family:Arial;
font-size:13px;
color:Aqua;
}
</style>
</head>
<body>
<p class="para">Hello World</p>
</body>
</html>
```



Unit 5

Introduction to Client side Scripting

What is Scripting Language ?

Client side and server side scripting

Types of scripting languages

What is Scripting Language ?

A scripting language or script language is a programming language that supports the writing of scripts, programs written for a software environment that automate the execution of tasks which could alternatively be executed one-by-one by a human operator. Environments that can be automated through scripting include software applications, web pages within a web browser, the shells of operating systems (OS), and several general purpose and domain-specific languages such as those for embedded systems.

What is meant by script in HTML?

It defines how locally executable **scripts** may be used in a web page. A particular client-side application, such as a web browser, may support several **script** languages. **Script** code may be executed as the document loads or at a later time. **Script** code can be written directly in the **HTML** document inside: **SCRIPT** elements.

Client-Side versus Server-Side Scripting

There are two basic varieties of scripting, client-side and server-side. As their names imply, the main difference is where the scripts are actually executed.

Client-side scripting

Client-side scripts are run by the client software—that is, the user agent. As such, they impose no additional load on the server, but the client must support the scripting language being used. JavaScript is the most popular client-side scripting language, but Jscript and VBScript are also widely used. Client-side scripts are typically embedded in HTML documents and deployed to the client. As such, the client user can usually easily view the scripts. For security reasons, client-side scripts generally cannot read or write to the server or client file system.

Server-side scripting

Server-side scripts are run by the Web server. Typically, these scripts are referred to as CGI scripts, CGI being an acronym for Common Gateway Interface, the first interface for server-side Web scripting. Server-side scripts impose more load on the server, but generally don't influence the client—even output to the client is optional; the client may have no idea that the server is running a script. Perl, Python, PHP, and Java are all examples of server-side scripting languages. The script typically resides only on the server, but is called by code in the HTML document. Although server-side scripts cannot read or write to the client's file system, they usually have some access to the

server's file system. As such, it is important that the system administrator takes appropriate measures to secure server-side scripts and limit their access.

Unit 6

INTRODUCING JAVASCRIPT

Introduction

Operators, Assignments and
Comparisons, Reserved words

Starting with JavaScript: Writing first
JavaScript program, Putting Comments

Functions

- Statements in JavaScript
- Working with objects:
Object Types and Object
Instantiation, Date object,
Math Object, String object,
Event object, Frame object,
Screen object
- Handling Events: Event
handling attributes, Window
Events, Form Events, Event
Object, Event Simulation
- Events- Keyboard & Mouse
events

INTRODUCTION

It's important to understand the difference between Java and JavaScript. Java is a full programming language developed by **Sun Microsystems** with formal structures, etc. JavaScript is a *scripting* language developed by **Netscape** that is used to modify web pages. Most JavaScript must be written in the HTML document between **<SCRIPT>** tags. You open with a **<SCRIPT>** tag, write your JavaScript, and write a closing **</SCRIPT>** tag. Sometimes, as an attribute to script, you may add "**Language=JavaScript**" because there are other scripting languages as well as JavaScript that can be used in HTML. To understand the workings of JavaScript, it is essential to understand a few basic programming concepts.

JavaScript is object-oriented. An *Object* in JavaScript is a resource that has specific characteristics known as *properties* and provides several services known as *methods* and *events*.

Features of Javascript

- Javascript is object based scripting language based on c++. Scripting language is a light weight programming language which easy to learn and understand. It is generally used for small applicatons.
- Javascript was basically designed to add interactivity in HTML pages and is directly embedded into HTML.
- Javascript is free to use by anyone.
- Javascript is interpreted language ie is not precompiled before execution. As javascript is interpreted, it is platform independent.
- Java is a full fledged complex programming language developed by sun microsystem. Javascript is developed by netscape communications and is no sub language of java.
- Javascript is originally a scripting language developed by European Computer Manufacturer's association (ECMA)
- Javascript that runs at the client side (ie at the client's browser) is client side java script (CCJS) and javascript that runs at the server is serverside java script (SSJS)
- Javascript being object oriented, uses number of built in javascript as well as objects can be created.
- Every object has properties and methods. Property is value(s) associated with an object. Methods are actions associated with an object.
- Example:

```
<scripttype="text/javascript">  
    document.write("This message is written by JavaScript");  
</script>
```

in above example, 'document' is object and write() is a method of document object.

- Javascript runs in a web browser, and when a script written by a third party is executed on the browser, there is a risk of running a spyware or a virus program.

- Hence, each time javascript is loaded on the browser implements a security policy designed to minimise the risk of such unknown code.
- Security policy is set of rules governing what scripts can do under which circumstances.
- Modern javascript security is based upon Java. Scripts downloaded are isolated from the operating system and then executed. This is known as the 'sandbox' model. Some scripts are often stored randomly here and there. And hence, many times obtain more power than expected by design or by accident.
- Scripts in general are given limited access and more access is only given with the user consent. Taking a consent for every execution is not a practical solution.
- Scripts from 'trusted' source are many times excluded from this consent procedure.
- A policy called 'same origin' does not block scripts coming from the same origin as trusted scripts. This same origin check is performed on all methods of windows object, also on embedded and externally linked objects.

JAVASCRIPT OPERATORS

Category	Operator	Name/Description	Example	Result
Arithmetic	+	Addition	3+2	5
	-	Subtraction	3-2	1
	*	Multiplication	3*2	6
	/	Division	10/5	2
	%	Modulus	10%5	0
	++	Increment and then return value	X=3; ++X	4
		Return value and then increment	X=3; X++	3
	--	Decrement and then return value	X=3; --X	2
		Return value and then decrement	X=3; X--	3
Logical	&&	Logical “and” evaluates to true when both operands are true	3>2 && 5>3	False
		Logical “or” evaluates to true when either operand is true	3>1 2>5	True
	!	Logical “not” evaluates to true if the operand is false	3!=2	True
Comparison	==	Equal	5==9	False
	!=	Not equal	6!=4	True
	<	Less than	3<2	False
	<=	Less than or equal	5<=2	False
	>	Greater than	4>3	True
	>=	Greater than or equal	4>=4	True
String	+	Concatenation(join two strings together)	“A”+“BC”	ABC

Javascript Operators Example

```

JSOperatorEg.html - Notepad
File Edit Format View Help
<body>
<script type="text/javascript">
<!--
var two = 2
var ten = 10

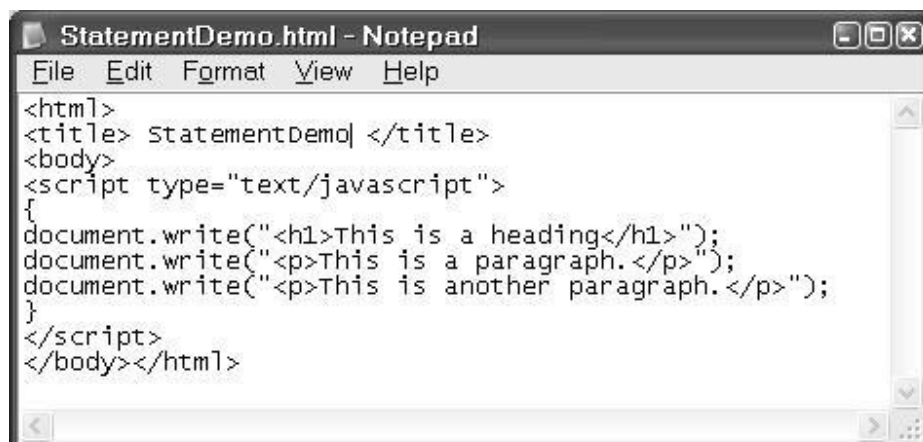
document.write("two plus ten = ")
var result = two + ten
document.write(result)
document.write("<br />")

document.write("ten * ten = ")
result = ten * ten
document.write(result)
document.write("<br />")

document.write("ten / two = ")
result = ten / two
document.write(result)
//
  
```


JAVASCRIPT STATEMENTS

- JavaScript is a sequence of statements to be executed by the browser. Browser executes the statements in the same order as they are written.
- JavaScript is case sensitive with all syntax, variable and function names.
- The semicolon at the end of line is optional (according to the JavaScript standard), and the browser is supposed to interpret the end of the line as the end of the statement. However, semicolon at the end of line is good programming practice. Also, it enables us to write multiple statements on the same line.
- JavaScript statements can be grouped together in blocks. Blocks start with a left curly bracket {, and end with a right curly bracket }. The purpose of a block is to make the sequence of statements execute together.
- A block is normally used to group the statements in a function or condition.
- A general example of block is –

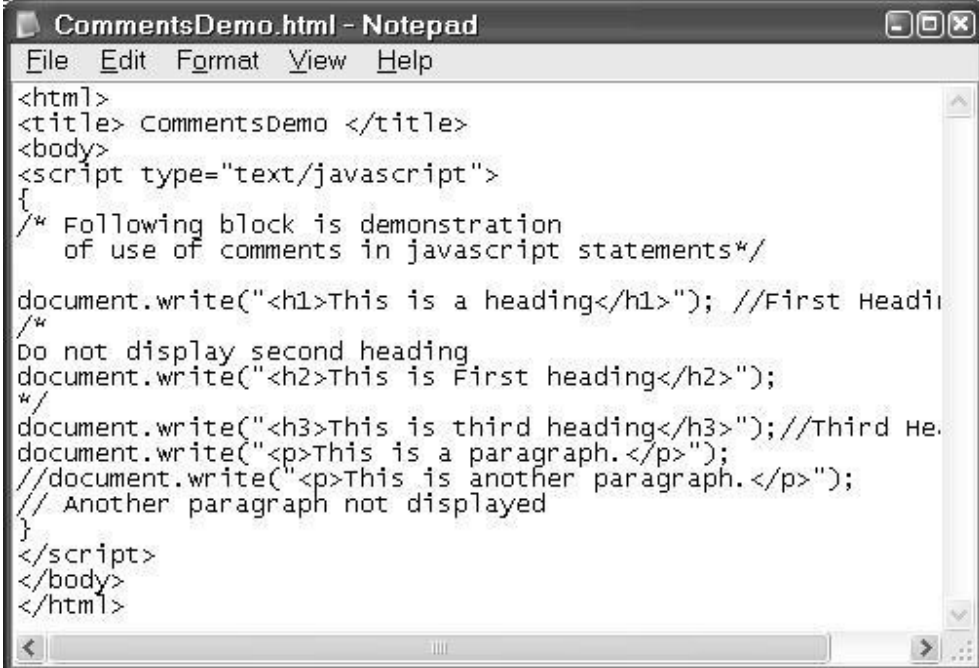


```
StatementDemo.html - Notepad
File Edit Format View Help
<html>
<title> statementDemo </title>
<body>
<script type="text/javascript">
{
document.write("<h1>This is a heading</h1>");
document.write("<p>This is a paragraph.</p>");
document.write("<p>This is another paragraph.</p>");
}
</script>
</body></html>
```

JavaScript Comments

- JavaScript comments can be used to make the code more readable.
- Comments can be added to explain the JavaScript.
- Comments can be added at end of a line.
- Single line comments start with //.
- Multi line comments start with /* and end with */.
- The comment is used to prevent the execution of a single code line or a code block. This can be suitable for debugging

Following example demonstrates use of comments in javascript code.



```
<html>
<title> CommentsDemo </title>
<body>
<script type="text/javascript">
{
/* Following block is demonstration
   of use of comments in javascript statements*/

document.write("<h1>This is a heading</h1>"); //First Heading
/*
Do not display second heading
document.write("<h2>This is First heading</h2>");
*/
document.write("<h3>This is third heading</h3>");//Third Heading
document.write("<p>This is a paragraph.</p>");
//document.write("<p>This is another paragraph.</p>");
// Another paragraph not displayed
}
</script>
</body>
</html>
```

Javascript Variables (Var statement)

- Variables are "containers" for storing information. This information can be values or expressions.
- A variable can have a short name, like x, or a more descriptive name, like MyName.
- Rules for JavaScript variable names:
 - Variable names are case sensitive (y and Y are two different variables)
 - Variable names must begin with a letter or the underscore character
- Creating variables in JavaScript is most often referred to as "declaring" variables.
- You declare JavaScript variables with the var keyword like var x;
- After the declaration shown above, the variables are empty (they have no values yet). However, you can also assign values to the variables when you declare them like var x = 10; After the execution of this statement, the variable x will hold the value 10
- A variable declared within a JavaScript function becomes **LOCAL** and can only be accessed within that function. (the variable has local scope).
- You can have local variables with the same name in different functions, because local variables are only recognized by the function in which they are declared.
- Local variables are destroyed when you exit the function.
- Variables declared outside a function become **GLOBAL**, and all scripts and functions on the web page can access it.
- Global variables are destroyed when you close the page.
- If you declare a variable, without using "var", the variable always becomes GLOBAL.
- If you assign values to variables that have not yet been declared, the variables will automatically be declared as global variables.
- All javascript operators can be used with variables

Example –

```
<html>
<title> VarDemo </title>
<body>
<script type="text/javascript">
{
/* Following block is demonstration
```

Conditional statements

Conditional statements are used to perform different actions based on different conditions. In JavaScript we have the following conditional statements:

1. **If statement** - This statement is used to execute some code only if a specified condition is true.

Syntax:

```
if(condition)

{

    code to be executed if condition is true

}
```

2. **If...else statement** - This statement is used to execute some code if the condition is true and another code if the condition is false

Syntax:

```
if (condition)

{

    code to be executed if condition is true

}

else

{

    code to be executed if condition is not true

}
```

3. **If...else if....else statement** - This statement is used to select one of many blocks of code to be executed

Syntax:

```
if (condition1)
```

```
{
```

```
    code to be executed if condition1 is true
```

```
}
```

```
else if (condition2)
```

```
{
```

```
    code to be executed if condition2 is true
```

```
}
```

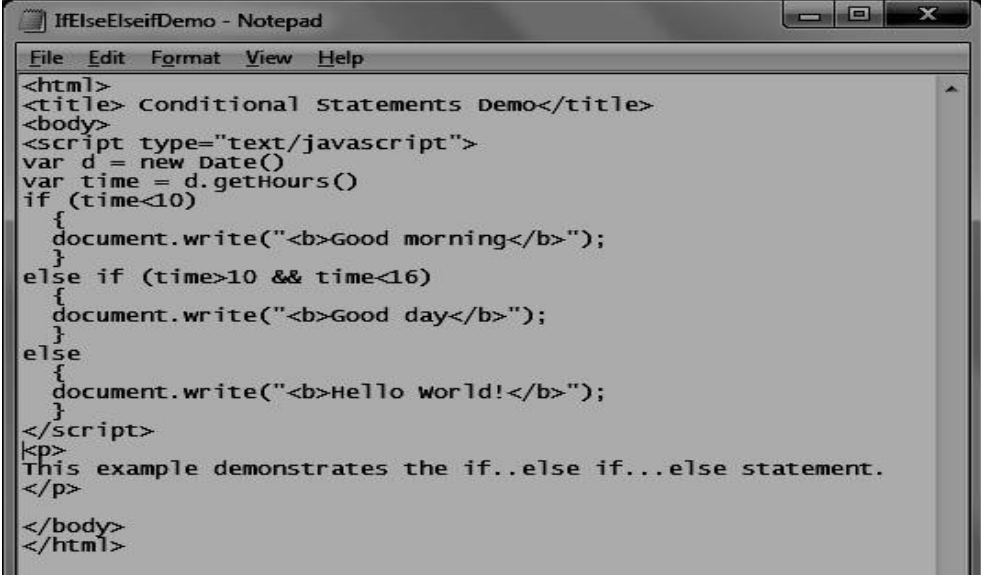
```
else
```

```
{
```

```
    code to be executed if neither condition1  
    nor condition2 is true
```

```
}
```

Example



```
File Edit Format View Help
<html>
<title> Conditional Statements Demo</title>
<body>
<script type="text/javascript">
var d = new Date()
var time = d.getHours()
if (time<10)
{
document.write("<b>Good morning</b>");
}
else if (time>10 && time<16)
{
document.write("<b>Good day</b>");
}
else
{
document.write("<b>Hello world!</b>");
}
</script>
<p>
This example demonstrates the if..else if...else statement.
</p>
</body>
</html>
```

4. **switch statement** -This statement is another way to select one of many blocks of code to be executed.

Syntax -

```
switch(n)
```

```
{
```

case 1:

execute code block 1
break;

case 2:

execute code block 2
break;

default:

code to be executed if n is different from case 1
and 2

}

Example:

```
<html>
<title> Conditional Statements Demo - switch statement</title>
<body>
<script type="text/javascript">
/*You will receive a different greeting based
on what day it is. Note that Sunday=0,
Monday=1, Tuesday=2, etc.*/

var d=new Date();
document.write("Today is " + d + ("<br\>"));
var theDay=d.getDay();

switch (theDay)
{
case 4:
```

Loop statements

Loops execute a block of code a specified number of times, or while a specified condition is true.

Often when you write code, you want the same block of code to run over and over again in a row. Instead of adding several almost equal lines in a script we can use loops to perform a task like this.

In JavaScript, there are two different kind of loops:

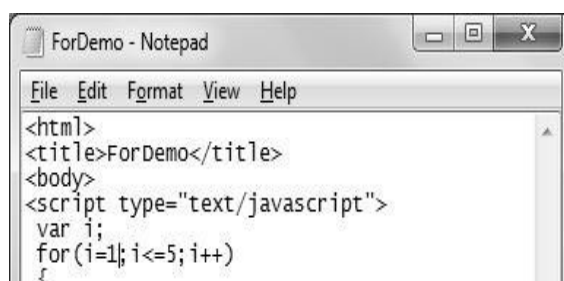
1. for - loops through a block of code a specified number of times. It can be used only when it is known in advance, how many times we have to run the loop.
2. while - loops through a block of code while a specified condition is true.

for Loop Syntax:

```
for (variable=startvalue;variable<=endvalue;variabl  
e=variable+increment)  
  
{  
  
code to be executed  
  
}
```

Example:

Javascript given below will print 5 numbers. Each time, value of the variable is incremented by 1.



While loop syntax

```
while (var<=endvalue)

{

    code to be executed

}
```

While loop can be used with any comparison operator.

Do While loop is a variation to the while loop. In this case block will be executed at least once, as the statements are executed before the

condition is tested. Syntax for do while is as follows –

```
do
```

```
{  
  
    code to be executed  
  
}  
  
while (var<=endvalue);
```

Consider the example where number is printed after incrementing it by 1. This is performed while the number is less than or equal to

. Script and the outputs with while and do while loop are as given below –

```
<html>  
  
<body>  
  
<script type="text/javascript">  
  
var i=6;  
  
do  
  
    {  
  
        document.write("The number is " + i);  
        document.write("<br />"); i++;  
  
    }  
  
while (i<=5);  
  
</script>  
  
</body>  
  
</html>
```

Changes in script with just while loop and corresponding output --

```
while (i<=5)

{

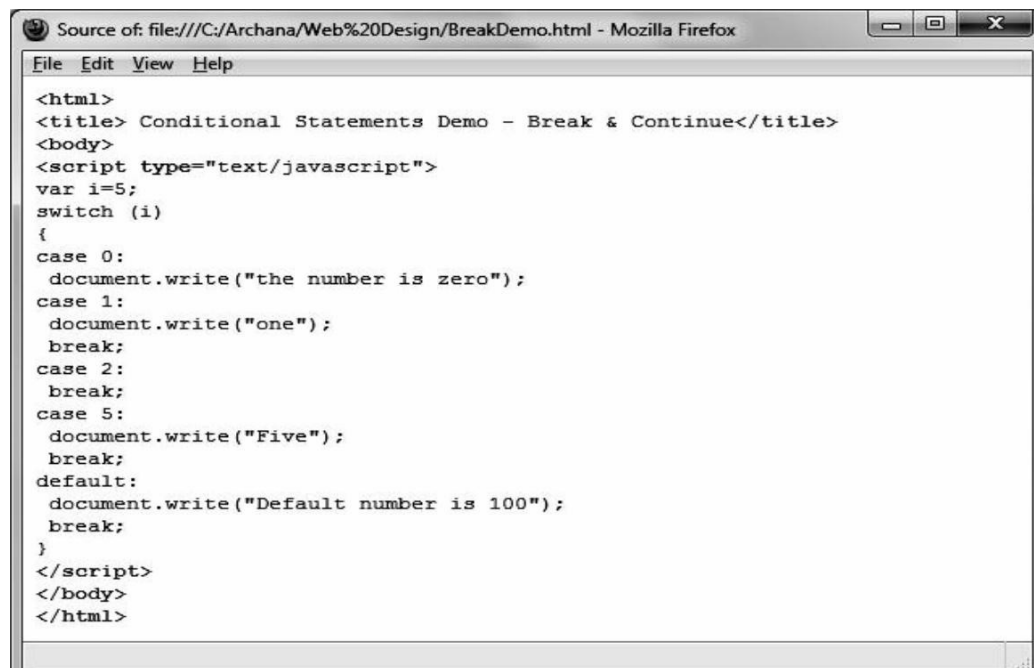
    document.write("The number is " + i);
    document.write("<br />");

    i++;
}

document.write("value of i is printed outside the
loop"); document.write("<br>" + i);
```

Break and Continue statement in javascript

- The break statement will break the loop and continue executing the code that follows after the loop (if any).
- The continue statement will break the current loop and continue with the next value.



```
Source of: file:///C:/Archana/Web%20Design/BreakDemo.html - Mozilla Firefox
File Edit View Help
<html>
<title> Conditional Statements Demo - Break & Continue</title>
<body>
<script type="text/javascript">
var i=5;
switch (i)
{
case 0:
document.write("the number is zero");
case 1:
document.write("one");
break;
case 2:
break;
case 5:
document.write("Five");
break;
default:
document.write("Default number is 100");
break;
}
</script>
</body>
</html>
```


Function

- Function is a segment of program that performs a given task.
- A function contains code that will be executed by an event or by a call to the function.
- You may call a function from anywhere within a page (or even from other pages if the function is embedded in an external javascript file).
- Functions can be defined both in the <head> and in the <body> section of a document.

However, to assure that a function is read/loaded by the browser before it is called, it should be put functions in the <head> section.

Syntax of the function is as follows –

```
functionfunctionname(var1,var2,...,varX)

{

    some code

}
```


- Function always includes parenthesis after the name of function '()'
- Function calls are case sensitive as javascript is case sensitive.
- The return statement is used to specify the value that is returned from the function.

So, functions that are going to return a value must use the return statement.

- If a variable is declared using "var", within a function, the variable can only be accessed within that function.

- The variable is destroyed once function call is over. These variables are called local variables. Local variables can have the same name in different functions, because each is recognized only by the function in which it is declared.
- If a variable is declared outside a function, all the functions on your page can access it.
- The lifetime of these variables starts when they are declared, and ends when the page is closed.
- Following is an example of function. Function products computes and returns product of variables a and b.
- Basic advantage of using a function is reusability. Same task can be performed again and again simply by calling the function which performs the task.

Example:



```
<html>
<head>
<script type="text/javascript">
function product(a,b)
{
return a*b;
}
</script>
</head>

<body>
<script type="text/javascript">
document.write(product(4,3));
</script>

</body>
</html>
```

CORE JAVASCRIPT

Data Types are classified as primitive data types and composite data types.

Composite Data types -

- Numbers - are values that can be processed and calculated. The numbers can be either positive or negative. Javascript integer can have three base values – 10 (decimal), 8(octal) or 16(hexadecimal). Number can be integer or floating point number.
- Strings - are a series of letters and numbers enclosed in single or double quotation marks. Strings are used for text to be displayed or values to be passed along.
- Some characters that you may want in a string may not exist on the keyboard, or may be special characters that can't appear as themselves in a string.
- In order to put these characters in a string, you need to use an escape sequence to represent the character. An escape sequence is a character or numeric value representing a character that is preceded by a backslash (\) to indicate that it is a special character.

Some escaped characters are as follows:

Escape

Sequence

Character

\b	Backspace.
\t	Tab. Tabs behave erratically on the Web and are best avoided, but sometimes you need them.
\n	New line (\u000a). Inserts a line break at the point specified. It is a combination of the carriage return (\r) and the form feed (\f).
\"	Double quote.
\'	Single quote, or an apostrophe, such as in can't.
\\	The backslash, since by itself it is a command.

- Boolean (true/false) - lets you evaluate whether a condition meets or does not meet specified criteria.
- Null - is an empty value. null is not the same as 0 -- 0 is a real, calculable number, whereas null is the absence of any value. An empty string is distinct from null value.
- NAN – Some javascript functions return a special value called not a number.

Primitive Data Types –

Object –

- An *object* is a collection of named values, called the *properties* of that object. Functions associated with an object are referred to as the *methods* of that object.
- Properties and methods of objects are referred to with a dot(.) notation that starts with the name of the object and ends with the name of the property. For instance image.src.
- Normally in objects there are only two nodes, the object and the property, but sometimes the properties can have properties of their own, creating an object tree.
- For instance, document.form1.namefield.
- Objects in JavaScript can be treated as associative arrays. This means that image.src and image['src'] are equivalent.
- JavaScript has many predefined objects, such as a Date object and a Math object. These are used much as function libraries are used in a language like C.
- They contain a collection of useful methods that are predefined and ready for use in any JavaScript code you may write.

Date Object –

- The Date object is used to work with dates and times.
- Date objects are created with new Date().
- There are four ways of instantiating a date:

```
var d = new Date();
```

```
var d = new Date(milliseconds);
```

```
var d = new Date(dateString);
```

```
var d = new Date(year, month, day, hours, minutes,  
seconds, milliseconds);
```

- Some javascript predefined methods are –
 - **getDate()** Returns the day of the month (from 1-31)
 - **getDay()** Returns the day of the week (from 0-6)
 - **getFullYear()** Returns the year (four digits)
 - **getHours()** Returns the hour (from 0-23)
 - **getMilliseconds()** Returns the milliseconds (from 0-999)
 - **getMinutes()** Returns the minutes (from 0-59)
 - **getMonth()** Returns the month (from 0-11)
 - **getSeconds()** Returns the seconds (from 0-59)
 - **getTime()** Returns the number of milliseconds since midnight Jan 1, 1970
 - **getTimezoneOffset()** Returns the time difference between GMT and local time, in minutes

- **getYear()** Deprecated. Use the
- **getFullYear()** method instead
- **parse()** Parses a date string and returns the number of milliseconds since midnight of January 1, 1970
- **setDate()** Sets the day of the month (from 1-31)
- **setHours()** Sets the hour (from 0-23)
- **setMilliseconds()** Sets the milliseconds (from 0-999)
- **setMinutes()** Set the minutes (from 0-59)
- **setMonth()** Sets the month (from 0-11)
- **setSeconds()** Sets the seconds (from 0-59)
- **setTime()** Sets a date and time by adding or subtracting a specified number of milliseconds to/from midnight January 1, 1970
- **toString()** Converts a Date object to a string
- **toTimeString()** Converts the time portion of a Date object to a string
- **toUTCString()** Converts a Date object to a string, according to universal time
- **valueOf()** Returns the primitive value of a Date object

Math object –

The Math object allows you to perform mathematical tasks.

The Math object includes several mathematical constants and methods.

- `round()` – rounds the number to nearest integer value.
- `random()` - returns a random number between 0 and 1.
- `max()` - returns the number with the highest value of two specified numbers.
- `min()` - returns the number with the lowest value of two specified numbers.

Array –

- An *Array* is an ordered collection of data values.
- In JavaScript, an array is just an object that has an index to refer to its contents. In other words, the fields in the array are numbered, and you can refer to the number position of the field.
- The array index is included in square brackets immediately after the array name. In JavaScript, the array index starts with zero, so the first element in an array would be `arrayName[0]`, and the third would be `arrayName[2]`.
- JavaScript does not have multi-dimensional arrays, but you can nest them, which is to say, an array element can contain another array.
- You access them listing the array numbers starting for the outmost array and working inward. Therefore, the third element (position 2) of or inside the ninth element (position 8) would be `arrayName[8][2]`.

Document and associated objects

Document

Link

Area

Anchor

Image

Events and Event Handlers

General information about events

Defining event handlers

Event Types

- onAbort
- onBlur
- onChange
- onClick
- onDbClick
- onDragDrop
- onError
- onFocus
- onKeyDown
- onKeyPress
- onKeyUp
- onLoad
- onMouseDown
- onMouseMove
- onMouseOut
- onMouseOver
- onMouseUp
- onMove
- onReset
- onResize

- onSelect
- onSubmit
- onUnload

DOCUMENT AND ASSOCIATED OBJECTS

Document

- Each HTML document loaded into a browser window becomes a Document object.
- The Document object provides access to all HTML elements in a page, from within a script.
- The Document object is also part of the Window object, and can be accessed through the 'window.document' property.
- Document object is part of document object model.
- This model has a fixed hierarchy, where topmost object in the hierarchy is Browser itself.
- After browser window object and inside window, as shown below, comes the document object.
- Relation of document object to the window object can be depicted in the fig given below –



|-> Document

|-> Anchor

|-> Link

|-> Images

|-> Tags

|-> Form

|-> Text-box

|-> Text Area

|-> Radio Button

|-> Check Box

|-> Select

|-> Button

The above figure shows Document Object and Window objectHierarchy in Document Object

Document object has following properties and methods

Properties –

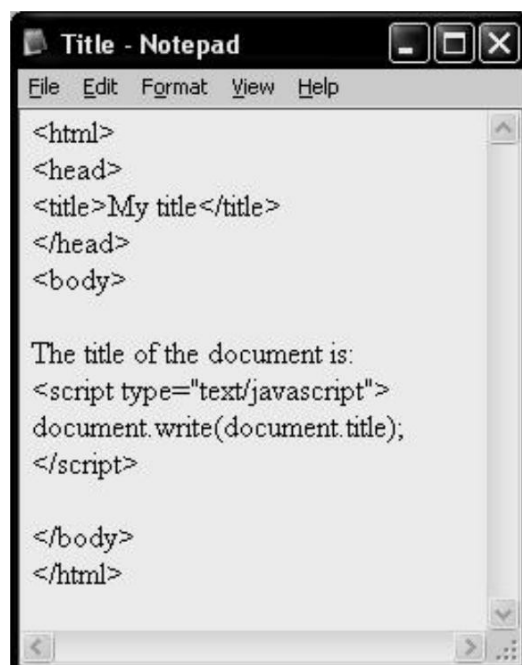
1. **cookie**Returns all name/value pairs of cookies in the document
2. **documentMode**Returns the mode used by the browser to renderthe document
3. **domain** Returns the domain name of the server that loaded thedocument
4. **lastModified**Returns the date and time the document was lastmodified
5. **readyState**Returns the (loading) status of the document
6. **referrer** Returns the URL of the document that loaded the currentdocument
7. **title** Sets or returns the title of the document
8. **URL** Returns the full URL of the document

Methods –

1. **close()** Closes the output stream previously opened withdocument.open()
2. **getElementById()** Accesses the first element with the specified id
3. **getElementsByName()** Accesses all elements with a specifiedname
4. **getElementsByTagName()** Accesses all elements with a specifiedtagname
5. **open()** Opens an output stream to collect the output fromdocument.write() or document.writeln()

6. **write()** Writes HTML expressions or JavaScript code to document
7. **writeln()** Same as write(), but adds a newline character after each statement Yes

Following examples demonstrates use of some properties and methods of document object.



```
<html>
<head>
<title>My title</title>
</head>
<body>

The title of the document is:
<script type="text/javascript">
document.write(document.title);
</script>

</body>
</html>
```

Script and output Demonstrating write method and title property of Document object.

EVENTS AND EVENT HANDLERS

What are events?

- Events are actions that can be detected by JavaScript.
- Every element on a web page has certain events which can trigger a JavaScript.
- For example, we can use the onClick event of a button element to indicate that a function will run when a user clicks on the button.

Examples of events:

1. A mouse click
2. A web page or an image loading
3. Mousing over a hot spot on the web page
4. Selecting an input field in an HTML form
Submitting an HTML form
5. A keystroke

Events are normally used in combination with functions, and the function will not be executed before the event occurs!

Defining event handlers

- They are JavaScript code that are not added inside the <script> tags, but rather, inside the html tags, that execute JavaScript when something happens, such as pressing a button, moving your mouse over a link, submitting a form etc.
- The basic syntax of these event handlers is:

name_of_handler="JavaScript code here"

- For example:

```
<a href="http://google.com"
onClick="alert('hello!')">Google</a>
```

- When events are associated with functions, the functions are written in the head section within the <script> tag and are called from the event handlers.

Example :

```
<html>

<body>

<h1      onclick="this.innerHTML='Welcome      to
EventHandlers'">Click      on      this      text</h1>

</body>

</html>
```

EVENT

Following is the list of events used by various javascript objects and when are these events triggered.

Attribute	The event occurs when...
onabort	Page is not finished loading
onblur	An element loses focus

onchange	The content of a field changes
onclick	Mouse clicks an object
ondblclick	Mouse double-clicks an object
ondragdrop	A user drops an object
onerror	An error occurs when loading a document or an image

onfocus	An element gets focus
onkeydown	A keyboard key is pressed
onkeypress	A keyboard key is pressed or held down
onkeyup	A keyboard key is released
onload	A page or image is finished loading
onmousedown	A mouse button is pressed
onmousemove	The mouse is moved
onmouseout	The mouse is moved off an element
onmouseover	The mouse is moved over an element
onmouseup	A mouse button is released
onmove	The position of top left corner of an object is moved.
onresize	A window or frame is resized
onreset	Reset button on the form is clicked.
onselect	Text is selected
onsubmit	Validate all form fields before submitting it.
onunload	The user exits the page

Following is an example which shows onmouseover event to give different messages for different parts of an imagemap.

```
eventDemo Notepad
File Edit Format View Help
<html>
<head>
<script type="text/javascript">
function writeText(text)
{
document.getElementById("desc").innerHTML+=text;
}
</script>
</head>

<body>


<map name="planetmap">
<area shape="rect" coords="0,0,82,126"
onmouseover="writeText('The Sun and the gas giant planets like Jupiter are by far the largest objects in
our Solar System.')"
href="sun.htm" target="_blank" alt="Sun" />

<area shape="circle" coords="90,58,3"
onmouseover="writeText('The planet Mercury is very difficult to study from the Earth because it is
always so close to the Sun.')"
href="mercur.htm" target="_blank" alt="Mercury" />

<area shape="circle" coords="121,58,8"
onmouseover="writeText('Until the 1960s, Venus was often considered a twin sister to the Earth
because Venus is the nearest planet to us, and because the two planets seem to share many
characteristics.')"
href="venus.htm" target="_blank" alt="Venus" />
</map>

<p id="desc">Mouse over the sun and the planets and see the different descriptions.</p>

</body>
</html>
```

**Script to demonstrate onmouseover event and its event
handler**

UNIT 7

XML

Introduction to XML

. Anatomy of an XML Document

Creating XML Documents

. XML DTDs

XML Schemas

XSL

INTRODUCTION TO XML

XML (extensible Markup Language) is a meta-language; that is, it is a language in which other languages are created. In XML, data is "marked up" with tags, similar to HTML tags. In fact, the latest version of HTML, called XHTML, is an XML-based language, which means that XHTML follows the syntax rules of XML.

XML was designed to describe data. XML is used to store data or information. This data might be intended to be read by people

or by machines. It can be highly structured data such as data typically stored in databases or spreadsheets, or loosely structured data, such as data stored in letters or manuals. XML tags are not predefined in XML. You must define your own tags. XML uses a DTD (Document Type Definition) to formally describe the data.

The main difference between XML and HTML

1. XML is not a replacement for HTML.

XML and HTML were designed with different goals:

- a. XML was designed to describe data and to focus on what data is.
 - b. HTML was designed to display data and to focus on how data looks.
2. HTML is about displaying information, XML is about describing information.

XML Benefits

When you write an HTML document, you see a nicely formatted page in a browser - instant satisfaction. When you write an XML document, you see an XML document (not the output)

1. **XML Holds Data, Nothing More**

XML does not really do much of anything. Rather, developers can create XML-based languages that store data in a structure way. Applications can then use this data to do any number of things.

2. **XML Separates Structure from Formatting**

One of the difficulties with HTML documents, word processor documents, spreadsheets, and other forms of documents is that they mix structure with formatting. This makes it difficult to manage content and design, because the two are mixed together.

In HTML, there is a <U> tag used for underlining text. It is also used for emphasis, or to mark a unit title. It would be

very difficult to write an application that searches through such a document for unit titles.

In XML, the book titles could be placed in <UNIT_TITLE> tags and the emphasized text could be place in tags.

3. XML Promotes Data Sharing

Applications that hold data using different structures must share data with one another. It can be very difficult for a developer to map the different data structures to each other. XML can solve this problem. Each application's data structure is mapped to an agreed-upon XML structure. Then all the applications share data in this XML format. Each application only has to know two structures, its own and the XML structure, to be able to share data with many other applications.

4. XML is Human-Readable

XML documents are (or can be) read by people as data stored in a database. It is not easy to browse through a database and read different segments of it as you would a text file. Given below is an XML document (person.xml):

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

```
<PERSON>
```

```
<NAME>
```

```
<FIRSTNAME>Raj</FIRSTNAME>
```

```
<LASTNAME>Mehra</LASTNAME>
```

```
</NAME>
```

```
<JOB>Software Engineer</JOB>
```

```
<GENDER>Male</GENDER>
```


</PERSON>

Code Explanation

Above XML is describing a person named Raj Mehra, who is a software engineer and is male.

5. **XML is Free**

XML doesn't cost anything to use. It can be written with a simple text editor or one of the many freely available XML authoring tools, such as XML Notepad. In addition, many web development tools, such as Dreamweaver and Visual Studio .NET have built-in XML support. There are also many free XML parsers, such as Microsoft's MSXML (downloadable from microsoft.com) and Xerces (downloadable at apache.org).

ANATOMY OF AN XML DOCUMENT

An XML document is made up of the following parts:

1. The Prolog (optional)

The prolog of an XML document can contain the following items.

1. The XML Declaration

The XML declaration, if it appears at all, must appear on the very first line of the document with no preceding white space. It looks like this:

```
<?XML          VERSION="1.0"          ENCODING="UTF-8"  
  
STANDALONE="yes"?>
```

This declares that the document is an XML document. The version attribute is required, but the encoding and standalone attributes are not. If the XML document uses any markup declarations that set defaults for attributes or declare entities then standalone must be set to no.

2. Processing Instructions

Processing instructions are used to pass parameters to an application. These parameters tell the application how to process the XML document. For example, the following processing instruction tells the application that it should transform the XML document using the XSL stylesheet artist.xsl

```
<?XML-stylesheet href="artist.xsl"
```

```
type="text/xsl"?>
```

As shown above, processing instructions begin with and <? end with ?>.

3. Comments

Comments can appear throughout an XML document. Like in HTML, they begin with <!-- and end with -->.

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

4. A Document Type Declaration

The Document Type Declaration (or DOCTYPE Declaration) has three roles.

It specifies the name of the document element.

It may point to an external Document Type Definition (DTD).

It may contain an internal DTD.

The DOCTYPE Declaration shown below simply states that the document element of the XML document is artists.

```
<!DOCTYPE ARTISTS>
```

If a DOCTYPE Declaration points to an external DTD, it must either specify that the DTD is on the same system as the XML document itself using SYSTEM keyword or that it is in some public location using PUBLIC keyword. It then points to the location of the DTD using a relative Uniform Resource Indicator (URI) or an absolute URI.

Syntax: <!--DTD is on the same system as the XML document-->

```
<!DOCTYPE ARTISTS SYSTEM "dtds/artists.dtd">
```

Syntax: <!--DTD is publicly available-->

```
<!DOCTYPE ARTISTS PUBLIC "-  
//freespace//DTD artists 1.0//EN"  
"http://www.freespace.com/artists/DTD/artists.  
dtd" >
```

In the second declaration above, public identifiers are divided into three parts:

1. An Organization (E.g., Freespace)
2. A Name for the DTD (E.g., Artists 1.0)
3. A Language (E.g., EN for English)

Prolog (optional)

Document Type Definition
(DTD)

XML Declaration

Comment

Processing Instructions

White Space

```
<?XML
VERSION="1.0"
ENCODING="UTF-8"
STANDALONE="no"?>

<!DOCTYPE
DOCUMENT SYSTEM
"tts.dtd">

<!-- Here is a comment -->

<?XML-stylesheet
TYPE="text/css"

HREF="myStyles.css"?>
```

2. The Document Element (usually containing nested elements)

- Document / Root Element

Every XML document must have at least one element, called the document/root element. The document element usually contains other elements, which contain other elements, and so on. Elements are denoted with tags. Consider again person.xml which we discussed earlier.

```
<?XML
VERSION="1.0"?><PER
SON>

<NAME>

  <FIRSTNAME>Raj</FIRSTNAME>

  <LASTNAME>Mehra</LASTNAME>

</NAME>

<JOB>Singer</JOB>

<GENDER>Male</GENDER>

</PERSON>
```

Code Explanation

The document / root element is PERSON. It contains three elements: NAME, JOB and GENDER. Further, the NAME element contains two elements of its own: FIRSTNAME and LASTNAME. As you can see, XML elements are denoted with tags, just as in HTML. Elements that are nested within another element are said to be children of that element.

- Empty Elements

In XML all elements might not contain other elements or text. E.g., in HTML, there is element / tag used to display

an image. It does not contain any text or elements / tags within it, so it is called an empty element. In XML, empty elements must be closed, but they do not require a separate close tag. Instead, they can be closed with a forward slash at the end of the open tag as shown below:

```
<IMG SRC="images/raj.jpg"/>
```

The above code is identical in function to the code below: ``

Elements & Content (required)

Root Element Opening Tag `<TTS><TT>`

Child Elements and Content `<name>XML </NAME>`

```
<URL>http://www.myserver.com/xml/tt</URL>
</TT>
```

```
<TT><NAME>HTML </NAME>
```

```
<URL>http://www.myserver.com/html/tt</URL>
</TT>
```

```
</TTS>
```

Root Element Closing Tag

- Attributes

XML elements can be further defined with attributes, which appear inside of the element's open tag as shown below:

```
<NAME TITLE="SoftwareEngineer">
```

```
<FIRSTNAME>Raj</FIRSTNAME>
```

```
<LASTNAME>Mehra</LASTNAME>
```

```
</NAME>
```

Here TITLE is an attribute of NAME element.

- CDATA

Sometimes it is necessary to include sections in an XML document that should not be parsed by the XML parser. These sections might contain content that will confuse the XML parser, perhaps because it contains content that appears to be XML, but is not meant to be interpreted as XML. Such content must be nested in CDATA sections. The syntax for CDATA sections is shown below:

```
<![CDATA[
```

```
This section will not get parsed by the XML parser.]]>
```

- White Space

In XML data, there are only four white space characters. They are:

1. Tab
2. Line-feed
3. Carriage-return

4. Single space

There are several important rules to remember with regards to white space in XML:

1. White space within the content of an element is important; that is, the XML processor will pass these characters to the application or user agent.
2. White space in attributes is normalized; that is, neighboring white spaces are reduced to a single space.
3. White space in between elements is ignored.

XML Syntax Rules

XML has relatively straightforward, but very strict, syntax rules. A document that follows these syntax rules is said to be well-formed.

1. There must be one and only one document element.
2. Every open tag must be closed.
3. If an element is empty, it still must be closed.
 - o Poorly-formed: `<TAG>`
 - o Well-formed: `<TAG></TAG>`
 - o Also well-formed: `<TAG />`
4. Elements must be properly nested:
 - o Poorly-formed: `<A>`
 - o Well-formed: `<A>`
5. Tag and attribute names are case sensitive.
6. Attribute values must be enclosed in single or double quotes.

- **Special Characters**

There are five special characters that cannot be included in XML documents. These characters are replaced with predefined entity references as shown in the table below:

Character	Entity Reference
<	<
>	>
&	&
"	"
'	'

3. Comments or Processing Instructions (optional)

Comments can appear throughout an XML document. Like in HTML, they begin with <!-- and end with -->.

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

Processing instructions are used to pass parameters to an application. These parameters tell the application how to process the XML document. For example, the following processing instruction tells the application that it should transform the XML document using the XSL stylesheet artist.xsl

```
<?XML-stylesheet HREF="artist.xsl"
```

```
TYPE="text/xsl"?>
```

As shown above, processing instructions begin with and <? end

with ?>.

CREATING XML DOCUMENTS

The following is relatively simple XML file describing the Artists:

contacts.xml

```
<?xml version="1.0" encoding="utf-8" ?>
<contacts>
<contact>
<name>Fred Flintstone </name>
<telephone>1234567823</telephone>
<mobile>567975977</mobile>
</contact>
<contact>
<name>Casper Ghost</name>
<telephone>5467823</telephone>
<mobile>98797785</mobile>
</contact>
<contact>
<name>Jughead</name>
<telephone>46565577</telephone>
<mobile>785454232</mobile>
</contact>
</contacts>
```

Code Explanation

In above document root element is contacts. Contacts element has one child element contact. Contact also has one child element name, telephone and mobile.

9.4. XML DTDs

A Document Type Definition (DTD) is a type of schema. The purpose of DTDs is to provide a framework for validating XML documents. By defining a structure that XML documents must conform to, DTDs allow different organizations to create shareable data files.

Well-formed vs. Valid

1. A well-formed XML document is one that follows the syntax rules described in "XML Syntax Rules".
2. A valid XML document is one that conforms to a specified structure.
3. For an XML document to be validated, it must be checked against a schema (document that defines the structure for a class of XML documents).
4. XML documents that are not intended to conform to a schema can be well-formed, but they cannot be valid.

Creating DTDs

DTDs are simple text files that can be created with any basic text editor. A DTD outlines what elements can be in an XML document and the attributes and sub-elements that they can take. Let's start by taking a look at a complete DTD and then dissecting it.

contacts.dtd

```
<!ELEMENT contacts (contact+)>
```

```
<!ELEMENT contact (NAME)>
```

```
<!ELEMENT contact (telephone)>
```

```
<!ELEMENT FIRSTNAME  
(#PCDATA)><!ELEMENT LASTNAME  
(#PCDATA)>
```

The Document Element

When creating a DTD, the first step is to define the document element.

```
<!ELEMENT contacts (contact+)>
```

The element declaration above states that the contacts element must contain one or more contact elements.

Child Elements

When defining child elements in DTDs, you can specify how many times those elements can appear by adding a modifier after the element name. If no modifier is added, the element must appear once and only once. The other options are shown in the table below:

Modifier	Description
?	Zero or One Times.
+	One or More Times.
*	Zero or More Times.

Other Elements

The other elements are declared in the same way as the document element - with the `<!ELEMENT>` declaration. The ARTISTS DTD declares three additional elements.

Choice of Elements

It is also possible to indicate that one of several elements may appear as a child element. E.g., the declaration below indicates that an IMG element may have a child element NAME or a child element ID, but not both.

```
<!ELEMENT IMG (NAME|ID)>
```

Empty Elements

Empty elements are declared as follows.

```
<!ELEMENT IMG EMPTY>
```

Mixed Content

Sometimes elements can have elements and text mixed. E.g., the following declaration is for a BODY element that may contain text in addition to any number of LINK and IMG elements.

```
<!ELEMENT BODY (#PCDATA | LINK | IMG)*>
```

Location of Modifier

The location of modifiers in a declaration is important. If the modifier is outside of a set of parentheses, it applies to the group; whereas, if the modifier is immediately next to an element name, it applies only to that element.

The following examples illustrate:

`<!ELEMENT BODY (LINK* | IMG*)>`

the BODY element can have any number of child LINK and IMG elements, but they must come in pairs, with the LINK element preceding the IMG element

`<!ELEMENT BODY (LINK, IMG)*>`

the BODY element can have any number of child LINK and IMG elements, but they must come in pairs, with the LINK element preceding the IMG element

`<!ELEMENT body (link*, img*)>`

the BODY element can have any number of child LINK elements followed by any number of child IMG elements

Using Parentheses for Complex Declarations

Element declarations can be more complex than the examples above. E.g., you can specify that a PERSON element either contains a single NAME element or a FIRSTNAME and LASTNAME element. To group elements, put them in parentheses as shown below:

`<!ELEMENT PERSON (NAME | (FIRSTNAME, LASTNAME))>`

Declaring Attributes

Attributes are declared using the `<!ATTLIST>` declaration. The syntax is shown below:

`<!ATTLISTElementName`

AttributeNameAttributeType State
DefaultValue?AttributeNameAttributeType
State DefaultValue?>

- ElementName is the name of the element taking the attributes.
- AttributeName is the name of the attribute.
- AttributeType is the type of data that the attribute value may hold. Although there are many types, the most common are CDATA (unparsed character data) and ID (a unique identifier). A list of options can also be given for the attribute type.
- DefaultValue is the value of the attribute if it is not included in the element.
- State can be one of three values: #REQUIRED, #FIXED (set value), and #IMPLIED (optional).

Validating an XML Document with a DTD

The DOCTYPE declaration in an XML document specifies the DTD to which it should conform. In the code sample below, the DOCTYPE declaration indicates the file should be validated against artists.dtd in the same directory. Add below line in Artists.xml after declaration:

```
<!DOCTYPEcontacts SYSTEM "contacts.dtd">
```

Below is the example how to work with internal and external DTD:

Internal DTD

```
<?XML VERSION="1.0"?>
```

```
<!DOCTYPE NOTE [
```

```
<!ELEMENT NOTE
```


(TO, FROM, HEADING, BODY)>

<!ELEMENT (#PCDATA)>
TO

<!ELEMENT (#PCDATA)>
FROM

<!ELEMENT HEADING (#PCDATA)>

<!ELEMENT (#PCDATA)>
BODY

<NOTE>

<TO>Amar</TO>

<FROM>Ajit</FROM>

<HEADING>Reminder</HEADING>

<BODY>Don't forget to read this</BODY>

</NOTE>

External DTD

This is the same XML document with an external DTD:

```
<?XML VERSION="1.0"?>

<!DOCTYPE NOTE SYSTEM "note.dtd">

<NOTE>

  <TO>Amar</TO>

  <FROM>Ajit</FROM>

  <HEADING>Reminder</HEADING>

  <BODY>Don't forget to read this</BODY>

</NOTE>
```

This is a copy of the file "note.dtd" containing the Document Type Definition:

```
<?XML VERSION="1.0"?>

<!ELEMENT NOTE (TO, FROM, HEADING, BODY)>

<!ELEMENT TO (#PCDATA)>

<!ELEMENT FROM (#PCDATA)>

<!ELEMENT HEADING (#PCDATA)>

<!ELEMENT BODY (#PCDATA)>
```

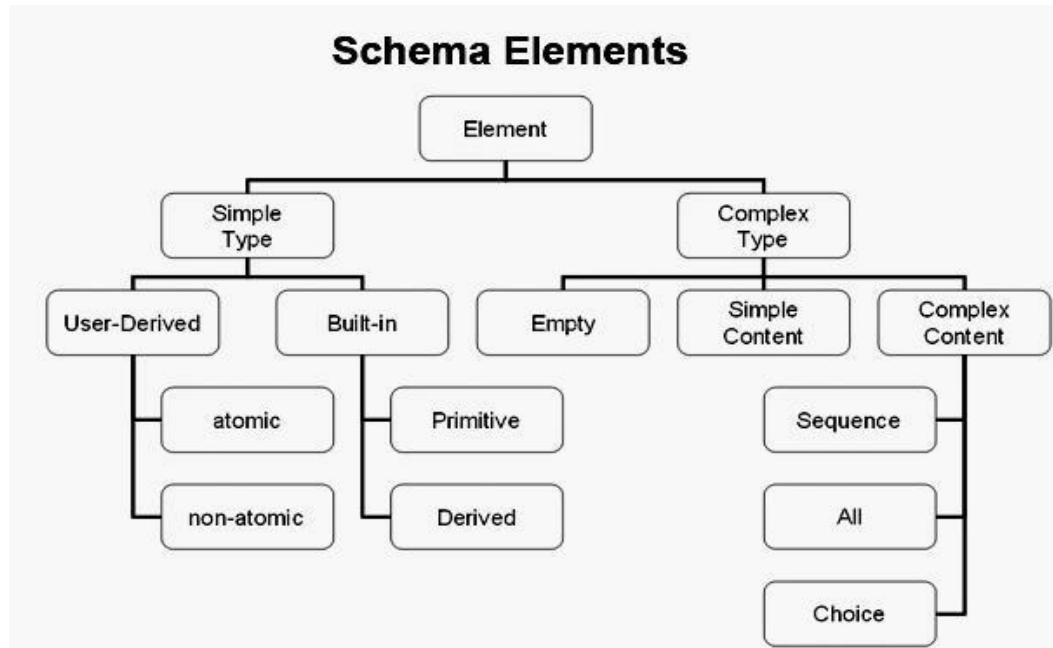
9.5. XML SCHEMAS

It is an XML-based language used to create XML-based languages and data models. It defines element and attribute names for a class of XML documents. It specifies the structure that those documents must adhere to and the type of content that each element can hold.

Why need XML Schema / Limitations of DTDs

- DTDs do not have built-in data types.
- DTDs do not support user-derived data types.
- 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.

Schema Elements



Schema authors can define their own types or use the built-in types. The following is a high-level overview of Schema types. Elements can be of simple type or complex type.

1) Simple Type

- a) These elements can only contain text. They cannot have child elements or attributes.
- b) All the built-in types are simple types (E.g., XS:STRING).
- c) Schema authors can derive simple types by restricting another simple type. E.g., an email type could be derived by limiting a string to a specific pattern (that includes '@', '.', '_', etc.)
- d) Simple types can be atomic (E.g., strings and integers) or non-atomic (E.g., lists).

2) Complex Type

- a) These elements can contain child elements and attributes as well as text.
- b) By default, complex-type elements have child elements.
- c) These elements can only contain text. But they are different from simple type elements in that they have attributes.
- d) These elements can be empty, but they have may have attributes.
- e) These elements may have mixed content - a combination of text and child elements.

<p>Simple XML Schema – Student.xsd</p> <pre> <?XML VERSION="1.0" ENCODING="UTF-8"?> <XS:SCHEMA XMLNS:XS="http://www.w3.org/2001 / XMLSchema"> <XS:ELEMENT NAME="Student"> <XS:COMPLEXTYPE> <XS:SEQUENCE> <XS:ELEMENT NAME="FirstName" TYPE="xs:string" /> <XS:ELEMENT NAME="LastName" TYPE="xs:string" /> </XS:SEQUENCE> </XS:COMPLEXTYPE> </XS:ELEMENT> </XS:SCHEMA> </pre>	<p>The code below shows a valid XML instance of this XML schema</p> <p>– student1.xml</p> <pre> <?XML VERSION="1.0"?> <STUDENT XMLNS:XSI="http://www.w3.org/ 2001/XMLSchema-instance" XSI:NONAMESPACESCHEMAL OCATION="Student.xsd"> <FIRSTNAME>Sumit</FIRSTN A ME> <LASTNAME>Tiwari</LASTNA ME> </STUDENT> </pre>
<p>Code Explanation:</p> <p>An XML schema is an XML document and must be well formed (i.e. follow all</p>	<p>Code Explanation:</p> <p>This is a simple XML document. Its document element is</p>

the syntax rules of XML document).	STUDENT, which contains two
XML schemas also have to follow the	child elements: FIRSTNAME
rules defined in the "Schema of	and
schemas," which defines, among	LASTNAME, just as the
other	associated XML schema
things, the structure of an element	requires.
and	The XMLNS:XSI attribute of the
attribute names in an XML schema.	document element indicates that
It is a common practice to use the XS	this XML document is an
qualifier to identify Schema elements	instance
and types.	of an XML schema. The
The document element of XML	document
schemas	is tied to a specific XML schema
is XS:SCHEMA. It takes the attribute	with the
XMLNS:XS with the value of	XSI:NONAMESPACESCHEMAL
http://www.w3.org/2001/XMLSchema	LOCATION attribute.
,	
indicating that the document should	
follow the rules of XML Schema.	
In this XML schema, we see a	
XS:ELEMENT element within the	
XS:SCHEMA element. XS:ELEMENT	
is used to define an element. In this	
case	
it defines the element STUDENT as	
a	
complex type element, which	
contains a	
sequence of two elements:	
FIRSTNAME and LASTNAME, both	
of which are of the simple string type.	

XSL

HTML pages use predefined tags, and the meaning of these tags is well understood: <P> means a paragraph and <H1> means a header, and the browser knows how to display these pages.

With XML we can use any tags we want and the meaning of these tags are not automatically understood by the browser: <TABLE> could mean a HTML table or maybe a piece of furniture. Because of the nature of XML, there is no standard way to display an XML document.

In order to display XML documents, it is necessary to have a mechanism to describe how the document should be displayed. One of these mechanisms is Cascading Style Sheets (CSS), but XSL (eXtensibleStylesheet Language) is the preferred style sheet language of XML, and XSL is far more sophisticated than the CSS used by HTML. XSL consists of two parts:

- ☐ a method for transforming XML documents (XSLT)
- ☐ a method for formatting XML documents (XSL-FO)

XSL is a language that can transform XML into HTML, a language that can filter and sort XML data and a language that can format XML data, based on the data value, like displaying negative numbers in red.

XSLT

An XSLT looks at an XML document as a collection of nodes of the following types:

- a. Root node
- b. Element nodes

- c. Attribute nodes
- d. Text nodes
- e. Processing instruction nodes
- f. Comment nodes

An XSLT document contains one or more templates, which are created with the

<XSL:TEMPLATE /> tag. The XSLT processor reads through the XML document starting at the root, progressing from top to bottom.

Example – artists.xsl	Code Explanation
<pre> <?XML VERSION="1.0"?> <XSL:stylesheet VERSION="1.0" XMLNS:XSL="http://www.w3.org/1999/ XSL/Transform"> <XSL:output METHOD="html"/> <XSL:template MATCH="child::ARTIST"> </pre>	<pre> Document begins with an XML declaration. As with all XML documents, the XML declaration is optional. The second line is the document element of the XSLT. It states that this document is a version 1.0 </pre>

</BODY>

</HTML>

</XSL:TEMPLATE>

</XSL:STYLESHEET>

<HTML>

<HEAD>

<TITLE>

<XSL:VALUE-OF

SELECT="descendant::FIRSTNAME"
/><XSL:TEXT></XSL:TEXT><XSL:
VALUE-OF

SELECT="descendant::LASTNAME"
/></TITLE>

</HEAD>

<BODY>

<TABLE BORDER="1"
WIDTH="200"><TR><TD>

<XSL:VALUE-OF

SELECT="descendant::FIRSTNAME"
/>

<XSL:TEXT></XSL:TEXT>
<XSL:VALUE-OF

SELECT="descendant::LASTNAME"
/>

</TD>

</TR>

</TABLE>

XSLT document.

```
<XSL:STYLESHEET
```

```
VERSION="1.0"
```

```
XMLNS:XSL="http://www.w3.or
```

```
g/1999/XSL/Transform">
```

The third line indicates that the resulting output will be HTML.

```
<XSL:OUTPUT METHOD="html"/>
```

The fourth line is an open `<XSL:TEMPLATE>` element. The `MATCH` attribute of this tag takes an XPath, which indicates that this template applies to the `ARTIST` node of the XML document. Because `ARTIST` is the document element of the source document, this template will only run once.

There are then a few lines of HTML tags followed by two `<XSL:VALUE-OF />` elements separated by one `<XSL:TEXT>` element. The `<XSL:VALUE-OF />` tag has a `SELECT` attribute, which takes an XPath pointing to a specific element or group of elements within the XML document. In this case, the two `<XSL:VALUE-OF />` tags point to `FIRSTNAME` and `LASTNAME` elements, indicating that they should be output in the title of the HTML page. The `<XSL:TEXT>` element is used to create a space between the `FIRSTNAME` and the `LASTNAME` elements.

```
<XSL:VALUE-OF
```

```
SELECT="descendant::FIRSTNA
```

```
ME" />
```

```
<XSL:TEXT></XSL:TEXT>
```

```
<XSL:VALUE-OF
```

```
SELECT="descendant::LASTNAM
```

```
E" />
```

There are then some more HTML tags followed by the same XSLT tags, re-outputting the `FIRSTNAME` and `LASTNAME` of the Beatle in the body of the HTML page.

After creating artists.xsl place it in the same directory as of artists.xml.

Also insert the following line in artists.xml after declaration:

```
<?XML-stylesheet HREF="artists.xsl" TYPE="text/xsl"?>
```

Save artists.xml and open it in browser. You will be able to find the output in tabular format as shown below:

Raj Mehra
Ajay Verma
Kapil Sharma
Rajiv malani

UNIT 8

WEBDESIGN CONCEPT

How the website should be

Basic rules of Web Page design Types of Website

What Is Good Web Design?

Before you read about the process of building Web pages, this section helps you define your goal clearly. What, exactly, is good Web design? Some people discuss what *isn't* good Web design (www.webpagesthatsuck.com), but this really doesn't demonstrate how to create good Web sites. Others like to discuss aesthetics and layout. Looks aren't everything. Function is important, too, and some people even claim that the answer to what constitutes good Web design is purely a matter of function. If it isn't usable, then it isn't reasonable—but function without motivating form is boring. Consider whether economically successful or trendy Web pages are well designed. Characterizing good Web design is not easy, especially because it depends largely on your target audience. Most Web discussions lose sight of the big picture, placing too much emphasis on how pages look, and not enough emphasis on their content, purpose, functionality, or the user's experience. Web design is not just graphic design. Web design

includes graphic design. Other important aspects of the Web design process may include such areas as the following:

- _ Artistic style, color theory, typography, and other visual concerns
- _ Information design, which specifies how information should be organized and linked
- _ Hypertext theory _ Technical writing _ System design _ Programming
- _ Network and server design
- _ Business issues and project management

Many disciplines are part of Web design. The first requirement, however, is a clear understanding of the site's ultimate purpose. The goal of a Web designer is to produce a

usable and appealing visual design for a software system, in the form of a Web site that helps a user fulfill some goal. In other words, the goal is to develop a site that can be delivered to the user in a satisfactory manner, be interpreted correctly by the user, and induce the desired outcome. Web design should be concerned not only with the aesthetic qualities of a Web site, but also with the user's overall experience in the context of a specific task or problem. The focus is on how something can be done, not just on how it looks. It is easy to throw out expressions like "perception is reality" or "content is king" as arguments for or against focusing on the visual nature of the Web. However, the reality is a balance between these extreme points of view. If you skimp on graphics, the site may seem boring. If you provide a wonderful interface, but skimp on content, the user may leave to find a site with more information. If you forget to debug, you may send the user away, facing error dialog boxes. Remember: experience is important. Always consider what feeling the user will take away after visiting your site. A sense of accomplishment? Frustration? Understanding? Disgust? The best approach to Web design is a holistic one, in which content, presentation, and interactivity work in harmony. So, how can you make a Web site that is both functional and visually appealing, without exceeding the constraints of the Internet and Web technologies?

10 Rules of Web Design

Follow these guidelines to create great websites

1. Simple is beautiful.
Cramming too much into each page creates confusion. Visitors get frustrated when they have to scan through rows of links and images to find what they are looking for. By keeping your pages simple, your website will be easier to use.
2. Design is paramount.
When you meet someone for the first time, you want to make a good first impression. The same should be true for your website. The overall look and feel of your site is the first thing your visitors will notice.
3. Navigation should be intuitive.
There are few things more frustrating than not being able to find what you want on a website. Pages should be well-organized with a top-down design so that visitors can easily browse through the different sections of your site.
4. Consistency is key.
Visitors shouldn't feel like they are visiting a new website each time they open a new page on your site. Consistent design across the pages within your site makes navigation a much easier task.
5. Colors are crucial.
Color selection can make or break a website. Most of us have visited websites that are simply painful to look at. When choosing colors, use a consistent palette of

colors that don't clash and make sure there is a strong contrast between the text and the background.

6. Make your website responsive.

People will access your website using a wide variety of devices – from smartphones to desktop computers. Therefore, it is important that your website displays correctly on different screen sizes. CSS media queries are a great way to implement responsive web design.

7. Develop for multiple browsers.

Browsers are supposed to render webpages the same way, but they don't. Therefore, make sure to check your website in multiple browsers to make sure everything appears correctly. It is best to catch problems ahead of time instead of relying on complaints from your visitors.

8. Check your website for errors.

As any experienced editor will tell you, a great piece of work can be tarnished by a small error. If you're a webmaster, check your websites on a regular basis for typos, broken links, and images that do not load correctly.

9. Write your own code.

Whether it's HTML or PHP, nothing beats writing your code from scratch. If you build your site from templates and pre-written scripts, you will be clueless when something goes wrong. When you code your own pages, you have full control over how they look and act.

10. Don't forget the content.

Even if your site is beautifully designed, it is only an empty shell without content. A good website has both great design and great content. Therefore, make sure your pages have unique, original content that makes them worth visiting.

Types of websites

There are three website types:

Content (information)

E-Commerce (online sales)

Interaction (Blogs, Bulletin Boards, Chat Rooms, and gaming sites).

Website types are implemented as dynamic or static:

Dynamic websites have frequently changing content or interact with the visitor. Dynamic websites typically use server side programming to generate HTML code as requested.

Static websites are written in pure HTML perhaps with a bit of JavaScript and only change when manually updated.

It's common to see combinations of the three types as well as combinations of dynamic and static. It's important to understand what they are and what works for you!

Content or information websites may be dynamic or static and the implementation depends upon how frequently the website information changes. News sites and search engines are dynamic database driven websites to allow rapid information update. Many corporate websites are static but that is changing rapidly.

E-commerce sites are almost always dynamic allowing for frequent product changes, pricing changes, sales and inventory updates. Simple e-commerce transactions like membership applications and online payment may be interactive while the main website is still static.

Interaction sites (Blogs, Bulletin Boards, Chat Rooms, and gaming sites) are dynamic.

Websites can be a combination of Content, E-Commerce and Interactive as well as a combination of dynamic and static. It's common to see a combination of dynamic and static implementations and combination of types. Because of this, more website owners are moving toward dynamic pages.

Pictures and graphics are always good to liven up a website. You should have at least some because the phrase "one picture is worth a thousand words" is as true now as when it was coined.



Reference:

HTML Black Book

https://sharpened.com/web_design_rules