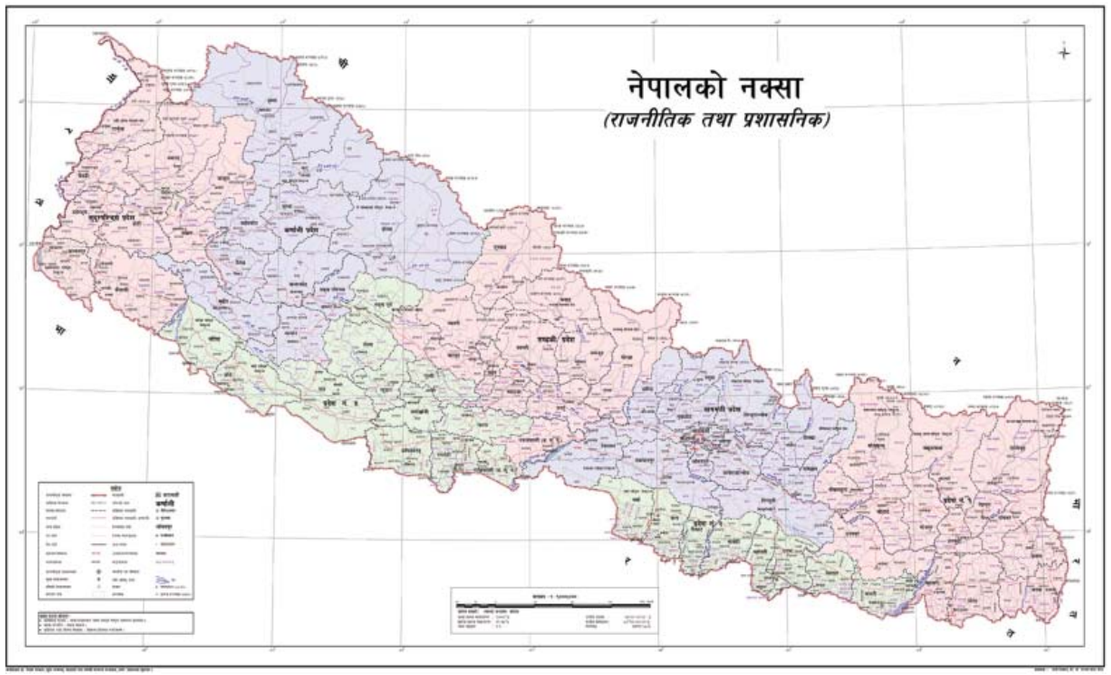



Government of Nepal
Ministry of Education, Science and Technology
Curriculum Development Centre
Sanothimi, Bhaktapur
Phone : 5639122/6634373/6635046/6630088
Website : www.moecdc.gov.np

# Web Page Designing

Feedback Copy

# Technical and Vocational Stream
# Learning Resource Material

# Web Page Designing
# (Grade 9)

## Secondary Level
## Computer Engineering

Government of Nepal
Ministry of Education, Science and Technology
**Curriculum Development Centre**
Sanothimi, Bhaktapur

**Publisher :** Government of Nepal
Ministry of Education, Science and Technology
**Curriculum Development Centre**
Sanothimi, Bhaktapur



**Layout by Khados Sunuwar**

# Preface

The curriculum and curricular materials have been developed and revised on a regular basis with the aim of making education objective-oriented, practical, relevant and job oriented. It is necessary to instill the feelings of nationalism, national integrity and democratic spirit in students and equip them with morality, discipline and self-reliance, creativity and thoughtfulness. It is essential to develop in them the linguistic and mathematical skills, knowledge of science, information and communication technology, environment, health and population and life skills. it is also necessary to bring in them the feeling of preserving and promoting arts and aesthetics, humanistic norms, values and ideals. It has become the need of the present time to make them aware of respect for ethnicity, gender, disabilities, languages, religions, cultures, regional diversity, human rights and social values so as to make them capable of playing the role of responsible citizens with applied technical and vocational knowledge and skills. This Learning Resource Material for Computer Engineering has been developed in line with the Secondary Level Computer Engineering Curriculum with an aim to facilitate the students in their study and learning on the subject by incorporating the recommendations and feedback obtained from various schools, workshops and seminars, interaction programs attended by teachers, students and parents.

In bringing out the learning resource material in this form, the contribution of the Director General of CDC Dr. Lekhnath Poudel, Dr. Tankanath Sharma, Dr. Romakanta Pandey, Bishnuraj Bhandari, Sankar Kumar Yadav, Anup Bhuju, Trimandir Prajapati is highly acknowledged. The book is written by Satyaram Suwal and Tej Prasad Dhamala and the subject matter of the book was edited by Badrinath Timalsina and Khilanath Dhamala. CDC extends sincere thanks to all those who have contributed in developing this book in this form.

This book is a supplimentary learning resource material for students and teachrs. In addition they have to make use of other relevnt materials to ensure all the learning outcomes set in the curriculum. The teachers, students and all other stakeholders are expected to make constructive comments and suggestions to make it a more useful learning resource material.

2076 BS

Ministry of Education, Science and Technology
**Curriculum Development Centre**

# Table of Contents

# UNIT-1

# Internet /Web basics

## Learning Outcomes

To get knowledge of history of internet

- Familiar with protocol, web browsers and search engine
- Knowledge about web 2.0

## Lesson 1

### History of Internet

The internet has its root in the ARPANET system of the Advanced Research Project Agency of the U.S Department of Defense. ARPANET was the first WAN and had only four sites in 1960 A.D. The internet evolved from basic ideas of ARPANET for interconnecting computers. Initially, research organizations and universities used it to share and exchange information. In 1989, the U.S government lifted restrictions on the use of the internet, and allowed its usage for commercial purposes as well. Since then, the internet has grown rapidly to become the world's largest network. It now interconnects more than 30,000 networks, allowing more than 10 million computers, and more than 50 million computer users in more than 150 countries around the world to communicate with each other. The internet continues to grow at a rapid pace.

### How does Internet work?

In internet, most computers are not connected directly to the internet. Rather they are connected to smaller networks, which in turn connected through gateways to the internet backbone. Two new terms you have encountered just now – gateway and backbone.

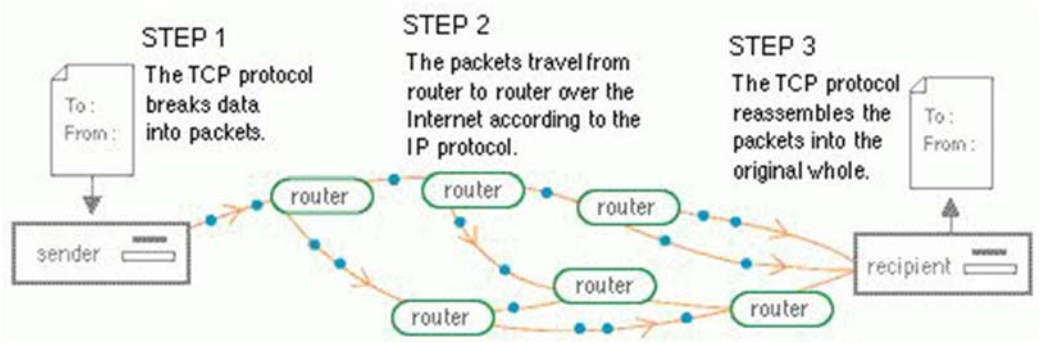



Figure No. 1 How data traves over the Net.

Let us now see how Internet works:

- At the source computer, the message or the file/document to be sent to another computer is firstly divided into very small parts called Packets. A packet generally contains upto 1500 characters.
- Each packet is given a number serial wise e.g., 1,2,3.
- All these packets are then sent to the address of destination computer.
- The destination computer receives the packet in random manner. (it may even receive packet 10 before packet 1 arrives). If a packet is garbled or lost, it is demanded again.
- The packets are reassembled in the order of their number and the original massage/file/document is obtained.

Let us below see what all helps internet in doing so.

**Internet Functioning**

The reason behind the internet works is that every computer connected to it uses the same set of rules for communication. The sets of such rules is called protocol ?

The communication protocol used by internet is TCP/IP. The TCP (Transmission Control Protocol ) part is responsible for dividing the file/massage into packets on the source computer. It (TCP) is also responsible for reassembling the received packets at the destination or recipient computer.

The IP (Internet Protocol) part is responsible for handling the address of destination computer so that each packet is routed (send) to its proper destination. At this point,

an interesting question may arise ? Ok. Let's know – what is the full form of TCP/IP? Well the clues are nearby.

**Uses of Internet**

Today individuals, companies and institutions use the internet in many ways as mentioned below:

1. Businesses use the internet to provide access to complex databases, such as financial databases.
2. Companies carry out electronic commerce (commerce on internet) including advertising, selling, buying, and distributing products and people to telecommunicate, or work from a distance.
3. Businesses and institutions use the internet for voice and video conferencing and other forms of communication that enable people to telecommunication, or work from a distance.
4. The use of electronic mail (E-mail) over the internet has greatly speeded up the communication between companies, among co-workers and between other individuals.
5. Media and entertainment companies use the internet to broadcast audio and video, including live radio and television programs. They also offer online chat groups, in which people carry on discussions using written text, online news and weather programs.
6. Scientists and scholars use the internet to communicate with colleagues, to perform research, to distribute lecture notes and course materials to students, and to publish papers and articles.
7. Individuals use the internet for communication, entertainment, finding information, and to buy and sell goods and services.
8. Now a days people enjoy in social networking and news sites through internet.

**Advantages of the Internet**

- Greater access to information reduces research times.
- Useful communication links.
- Global reach enables one to connect to anyone on the internet.
- Easy communication between people.

- Publishing documents on the internet saves paper.
- A valuable resource for companies to advertise and conduct business.

**Disadvantages of the Internet**

- Cyber frauds may take place involving Credit/Debit card numbers and details.
- Unsuitable and undesirable content available at internet are sometimes used by notorious people such as terrorists.
- Computer viruses get downloaded and spread across machines connected to networks and have detrimental effects.
- Much of the information isn't checked and may be incorrect or irrelevant.
- Massages sent across the internet can be easily intercepted and are open to be abused by others.
- Too much time spent on the internet could result in a lack of face to face interaction between people and the there may be in people loss of social skills.
- People use internet as a tool in cyber crime.

**World Wide Web (WWW)**

The Web, or World Wide Web, is basically a system of internet servers that support specially formatted documents The documents are formatted in a markup language called HTML (HyperText Markup Language) that supports links to other documents, as well as graphics, audio, and video files. This means you can jump from one document to another simply by clicking on hot spots. Not all internet servers are part of the World Wide Web.

Tim Berners-Lee's vision of a global hyperlinked information system became a possibility by the second half of the 1980s. By 1985, the global internet began to proliferate in Europe and in the Domain Name System (which the Uniform Resource Locator is built upon) came into being. In 1988 the first direct IP connection between Europe and North America was made and Berners-Lee began to openly discuss the possibility of a web-like system at CERN.

## LESSON 2

### Web Browsers and Web Server

The World Wide Web is based upon clients and servers. A WWW client is called a Web browser or simply a browser, and a WWW server is called a Web server or sometimes just a server.

### Web Browser

A web browser (commonly referred to as a browser) is a software application for retrieving, presenting, and traversing information resources on the World Wide Web. An information resource is identified by a Uniform Resource Identifier (URI/URL) and may be a web page, image, video or other piece of content.[1] Hyperlinks present in resources enable users easily to navigate their browsers to related resources.

Although browsers are primarily intended to use the World Wide Web, they can also be used to access information provided by web servers in private networks or files in file systems. The major web browsers are Firefox, Internet Explorer/Microsoft Edge, Google Chrome, Opera, and Safari.

### Mobile Browsers

Also, there are a number of browsers that are designed to access the Web using a mobile device. A mobile browser, also called a microbrowser, is optimized to display Web content on smaller mobile device screens and to perform efficiently on these computing devices, which have far less computing power and memory capacity as desktop or laptop. Mobile browsers are typically "stripped down" versions of Web browsers and offer fewer features in order to run well on mobile devices.

### Web Server

Web Documents that you view on the Internet are stored on different Web servers. Web servers are computers on which Web documents reside.

A Web server is a program that runs on a computer connected to the Internet. The Web server watches the Internet connection and waits for requests from the web browser. When it receives a request, it finds the documents, generates the information if needed and sends it back to the browser that requested it.

**Web Sites, Web Addresses and Web Pages**

In this section, we are going to discuss briefly about web sites, web addresses and web pages.

**Web site and Web page**

A location on a web server is called a Web Site. The documents residing on web sites are called Web pages. The web pages use a protocol HTTP.

There are many associated terms which should be discussed for better understanding. Let us discuss them:

1. **HOME PAGE.** This is default page and also known as index page. It is the top-level web page of a web site. When a web site is opened, its home page is displayed.
2. **WEB PORTAL.** It is a web site, that presents information from diverse sources. In other words, a web portal has hyperlinks to many other web sites. By clicking upon these links, the corresponding web sites can be opened. Thus, you can say that web portals provide a single point of access to a variety of content and core services. www. Yahoo.com is an example of a web portal. Other examples are www. Kantipur.com, www. Khoj.com, etc.
3. **Web Address and URL** A location on a net server is called a web site. Each web site has a unique address called URL (Uniform Resource Locater) e.g, the web site of Microsoft has an address or URL. called http://microsoft.com.

Let us explore it further. The Internet structure of the World Wide Web is built on a set of rules called Hypertext Transfer Protocol (HTTP) and a page description language called Hypertext Markup Language (HTML). HTTP uses Internet addresses in a special format called a Uniform Resource Locator and URLs look like this:

**Type:// address/path**

Where **type:** specifies the type of server in which the file is located, **address** is the address of server, and path tells the location of tile on the server. For example, in the following URL

http://encycle.msn.com/getnfo/styles.asp

http: specifies the type of server, encycle.msn.com is the address of the server and getinfo/style.asp is the part of the file styles.asp. The other examples of URLs are

ftp://ftp.prenhall.com, http://www.yahoo.com, news://alt.tennis etc.

## LESSON 3

### SEARCH ENGINES

We are surfing internet by visiting sites known to us. What if we want to view particular information, but we do not know which site would provide us that information. In that case, we need to search for that information on the web. And this can be done through search engines, a program used for searching information on the internet.

**Table: Some popular search engines**

| Search Engine/Directory | URL |
|---|---|
| Google | https//www.goolge .com/ |
| Askjeevs | https//www.askjeevs.com/ |
| HotBot | https//www.hotbot.com/ |
| Infoseek | https//www.infoseek .com/ |
| Lycos | https//www.lycos.com/ |
| NetGuide live | https//www.netguide .com/ |
| All the Web | https//www.alltheweb.com/ |
| Yahoo! | https//www.yahoo .com/ |
| Ask | https//www.ask.com |
| Bing | https//www.bing.com |
| Altavista | https//www.altavista.com |

There are many search engines on the web. The searching process is similar on these all search engines. Lists some popular search engines on the web. We are going to learn the information searching process in one very popular search engine-google! (www.google.com).

To search for web pages pertaining to specific information, all you have to do is:

i. Go to the homepage of the search engine.
ii. Type the information to be searched for, before these steps one more step:- type 'www.google.com' in address bar of web browser. in the box provided for it.
iii. Now click at Search button next to it. Within a few seconds, the search engine will search for that information and display the links to the web pages, which are linked to your desired information in some way.

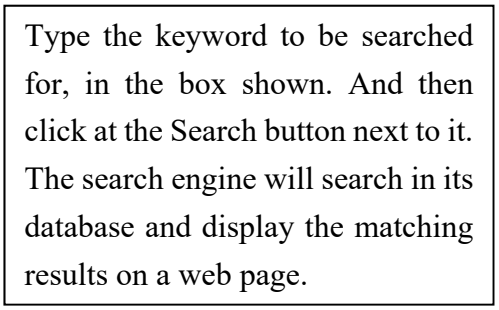


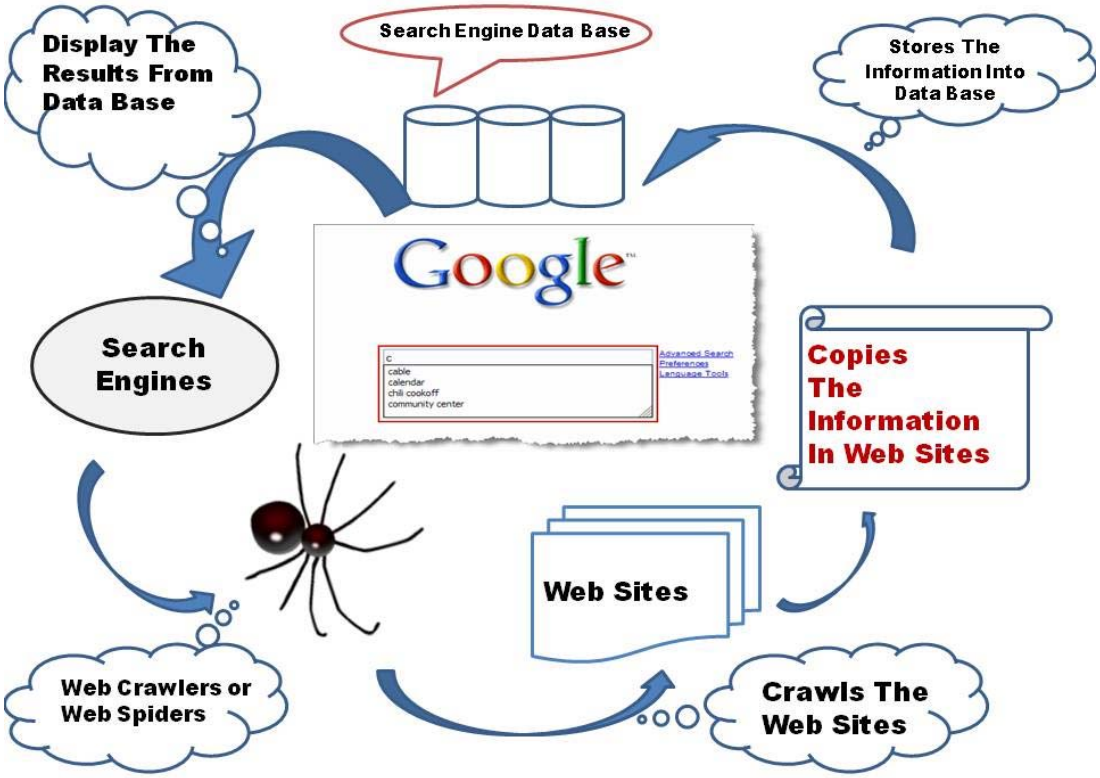

**Fig.** Show this process of searching information.

**How Search Engine Works?**

In a search engine, you can type keyword(s) to search for and a search engine searches them, on the web and provides you the details. A search engine works with the help of following three elements.

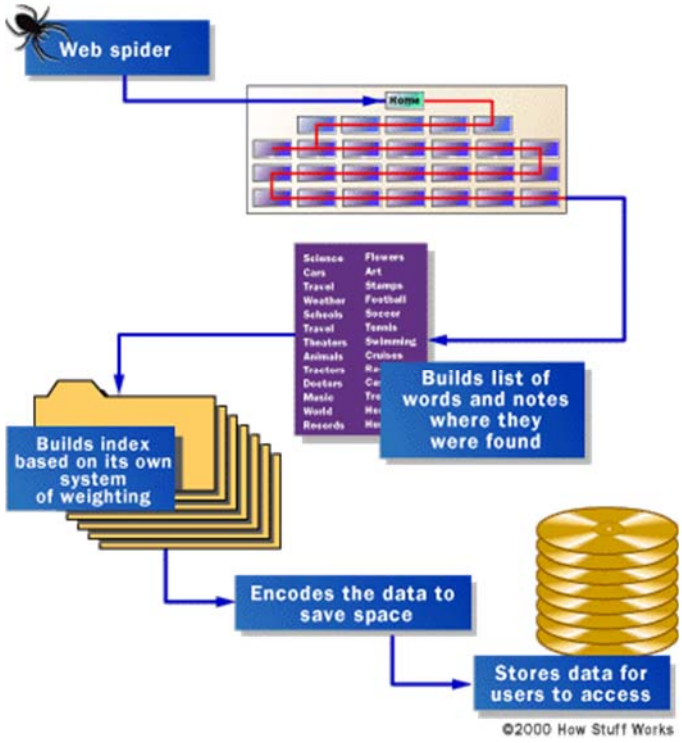
1) Spiders or Webcrawler or Bots or Agents.- The search engines use a software called spider or web crawler or Robot or bot or agent which comb the internet looking for documents and their web addresses. The spiders or webcrawlers perform the methodical searches needed to find information.

   The Bots or Spiders are given direction by the search engine and they crawl from one server to another, compiling the huge lists of URLs (based upon the directions) given by search engine.

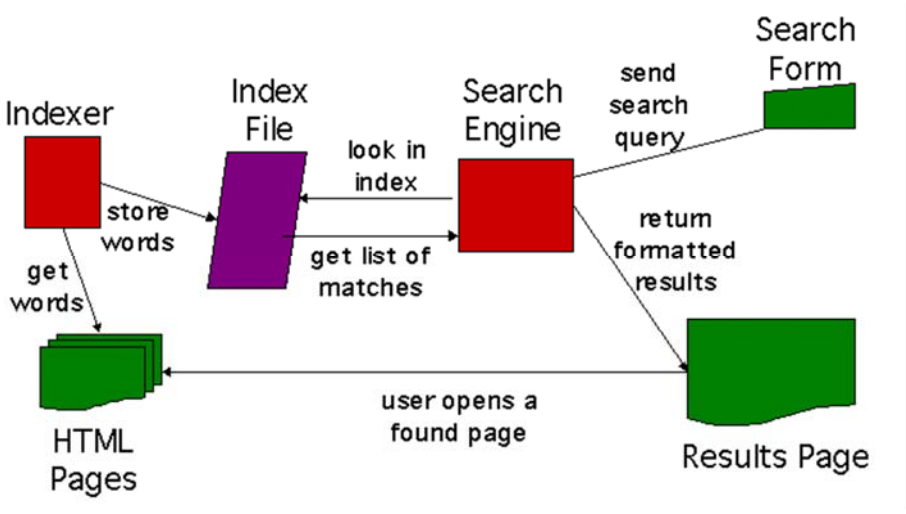



**2)** Indexing Software and Data-Base:- The lists of documents and web addresses collected by bots are sent to the indexing software. The indexing software extracts information from the documents and web addresses, prepares on Index of it and stores in a database.

The king of information indexed depends upon the particular search engine. Some index every word in the document: other index the document title only.




**3) Search Algorithm.:-** When you perform a search by entering keywords, the search engine software searches its database (in which indexing software stores its entries) using a particular search method called search algorithm. And then it displays the matching documents or web addresses. The working process of search engine are shown in figures.



**Some Searching Tips**

To effectively search for desired information, you must learn the art of framing search queries. In the following lines, we are going to discuss some tips for the same:

1. Do not ask question. For example, if you want to find out some information about Home Minister of Nepal and if you type
   Who is the Home Minister of Nepal?

In the search box, then you may end up getting lakhs of matches but most being useless.

This is because the search engine searches for each word in the search query separately. That is, in this case, your search will search for words “who”, “home”, “minister”, “Nepal” separately and then display all the results. Common words like “a”, “and”, “is”, “the” etc. are dropped.

2. Surround your query in quotes or put some punctuation marks(;-) if you want it to be treated as single phrase rather than a series of individual search terms. For example, if you want to search for the book titled 'God of Small thing' then you may write it as
   i. “god of small things”
   ii. Or as god-of-small-things
   iii. Or as god ; of ; small ; things
3. Use wildcard * for pattern matching. For instance, if you want to search for certain things starting with dir, then you may write “dir” in the search box and your search engine will search for many words starting with dir e.g. dire, direct, direction, directive, director etc.
4. The lowercase word matches are not case-sensitive but uppercase words are case-sensitive. For example, if you write program it will match with program or Programs or PRogram, …… and PROGRAM. But if you write program
   Then it will match PROgram only .
5. To specify that, a word or phrase must appear in matched documents, put a plus sing (+) immediately before it. For example, the following query + boston + university
   Will match with web pages having both the words boston and university.

6. To specify that a word or phrase must not appear in matched documents, put a minus sign (-) immediately before it , eg. + asia-china

## LESSON 4

### Web 1.0

Shopping cart applications, that most ecommerce website owners employ in some shape or form, basically fall under the category of Web 1.0. "The overall goal is to present products to potential customers, much as a catalog or a brochure does — only, with a website, you can also provide a method for anyone in the world to purchase products." The web provided a vector for exposure, and removed the geographical restrictions associated with a brick-and-mortar business.

### Web 2.0

Web 2.0 is the current state of online technology as it compares to the early days of the Web, characterized by greater user interactivity and collaboration, more pervasive network connectivity and enhanced communication channels.


One of the most significant differences between Web 2.0 and the traditional World Wide Web (WWW, retroactively referred to as Web 1.0) is greater collaboration among internet users, content providers and enterprises. Originally, data was posted on web sites, and users simply viewed or downloaded the content. Increasingly, users have more input into the nature and scope of web content and in some cases exert real-time control over it.

The social nature of Web 2.0 is another major difference between it and the original, static web. Increasingly, websites enable community-based input, interaction, ontent-sharing and collaboration. Types of social media sites and applications include forums, microblogging, social networking, social ookmarking, social curation, and

wikis.

**Web 3.0**

A web service is a software system designed to support computer-to-computer interaction over the internet. Web services are not new and usually take the form of an Application Programming Interface (API). The popular photography-sharing website Flickr provides a web service whereby developers can programmatically interface with Flickr to search for images. Currently, thousands of web services are available. However, in the context of Web 3.0, they take center stage. By combining a semantic markup and web services, the Web 3.0 promises the potential for applications that can speak to each other directly, and for broader searches for information through simpler interfaces.

**The History of Web 2.0**

The foundational components of Web 2.0 are the advances enabled by Ajax and other applications such as RSS and Eclipse and the user empowerment that they support.

Darcy DiNucci, an information architecture consultant, coined the term “Web 2.0 In her 1999 article, "Fragmented Future”:

“The Web we know now, which loads into a browser window in essentially static screenfuls, is only an embryo of the Web to come. The first glimmerings of Web 2.0 are beginning to appear, and we are just starting to see how that embryo might develop. The Web will be understood not as screenfuls of text and graphics but as a transport mechanism, the ether through which interactivity happens.”

Tim O'Reilly is generally credited with popularizing the term, following a conference dealing with next-generation Web concepts and issues held by O'Reilly Media and MediaLive International in 2004. O'Reilly Media has subsequently been energetic about trying to copyright "Web 2.0" and holds an annual conference of the same name.

**Instruction for Teacher:**

1) Use the Charts or Multimedia projector as far as possible to explain the content.
2) Demonstrate how internet and search engine work by using internet.
3) Assign the homework at the end of every class from the chapter so far covered during that class.
4) Conduct group discussions after finishing every lesson.

**References and Resources**


1) www.wikipedia.com
2) Information Technology class x Dhanapat Rai & co . Sumitra Arora.
3) http://www.webopedia.com/TERM/B/browser.html
4) http://www.practicalecommerce.com/articles/464-Basic-Definitions-Web-1-0-Web-2-0-Web-3-0

# UNIT-2

# An Introduction to HTML

## Learning Outcomes:

- Introduced with HTML
- HTML attributes
- HTML styles, tables,list,forms, media
- Introduction of HTML 5
- HTML 5 Canvas
- HTML 5 Drag and Drop
- HTML 5 elements
- HTML 5 attributes

### LESSON 1: AN INTRODUCTION TO HTML

**Html and its capabilities**

Before we actually start learning and writing HTML codes, we must know what it is and what it can do along with its limitations.

HTML is a document layout and hyperlink- specification language i.e, a language used to design the layout of a document and to specify the hyperlinks.

HTML tells the browser how to display the contents of a hypertext documents i.e, a document including text, images and other supported media. The language also tells how to make a document interactive through special hyperlinks.

Though HTML is a language that supports multimedia and new page layout features yet it has some limitations.

HTML is not a word processing tool; it is not a desktop publishing solution or even a programming language. It is just a page-layout and hyperlink markup specification language.

**WRITING HTML DOCUMENTS**

HTML is made up of elements or tags and attributes, which work together to identify document parts and tell browsers how to display them.

All HTML tags are contained within angle brackets (< >) e.g., <HEAD> is a tag. Similarly <H1> is a tag. Please note that you can type HTML in capital letters as well as in small letters. A browser treats both of them in the same manner. However, throughout this chapter and the coming chapters of HTML, we shall be using capitals for HTML tags to make them stand out from the rest of the text.

HTML Document Structure

An HTML document consists of text, which comprises the content of the document, and tags, which define the structure and appearance of the document, The basic structure of an HTML document is simple with the entire document bound by a pair of <HTML> and </HTML> tags.

Every HTML document should follow this general form:

```
<HTML>
<HEAD>
<TITLE> Title of page is written here </TITLE>
</HEAD>
<BODY>
The HTML tags that define your page go here
</BODY>
</HTML>
```

The <HEAD>……..</HEAD> tags make the header of the document and <BODY>……..</BODY> tags make the body of the HTML document. Header- that appears on top of the window and body- that appears in windows.

The <HTML> tag

The <HTML> tag identifies the document as an HTML document. An HTML document begins with <HTML> and ends with</HTML>. Here <HTML> starts the HTML tag and </HTML> ends the HTML tag.

```
<HTML>
……. HTML document lines here
```

</HTML>

The <HEAD> tag

The <HEAD> tag contains information about the document, including its title, scripts used, style definitions, and document descriptions. The <HEAD> tag is entered between <HTML> tags. For example;

<HTML>
<HEAD>
Header information comes here
</HEAD>
</HTML>

The <TITLE> tag

This tag contains the document title. The title specified inside <TITLE> tag appears on the browser's title bar. It'll be visible clearly to you when you start doing examples (especially example) given below along with their outputs.

The <TITLE> tag is entered between the opening and closing <HEAD> tags. E.g.,

<HTML>
<HEAD>
<TITLE>
Document Title here
</TITLE>
</HEAD>
</HTML>

The <BODY> tag

The <BODY> tag encloses all the tags, attributes and information to be displayed in the web page. The <BODY> tag is entered below the closing </HEAD> tag and above the closing <HTML> tag as shown below:

<HTML>
<HEAD>
<TITLE>
Document
</TITLE>

</HEAD>
<BODY>
All the tags, attributes, and information in the document body go here

</BODY>
</HTML>

**HTML Writing Tools**

As for making and testing your documents, the easiest way to do this is to create and view them on your personal computer. This may be done using am HTML editor like FrontPage Express, but I prefer to use a text processor like gedit and web browser like Mozilla FireFox or Iceweasal. The following lines explain how can you create HTML document and view them in a browser.

**Creating an HTML Document**

To create an HTML document using a text editor, follow the steps given below. Please note that you can use any text editor that lets you create plain text files (e.g Notepad, notepad ++) etc. For illustration purposes, we are using the default text editor (gedit) of Boss Linux Distribution. [recall that Boss Linux distribution is Indianite GNU/ Linux distribution]

1. Open text editor by clicking at
   Application Accessories Text Editor → →
2. Now the gedit Window appears. Type the HTML code here
3. To save this code, click at File → save command. Then in the save dialog box, after selecting the desired folder, give the desired file name along with extension. HTM or .HTML and then click on Save button. We have saved out HTML document shown in under name try.html.

Viewing HTML Document in a Browser

To view an HTML document in a browser, follow these steps:

1. Open the Browser window. To open the Iceweasal Web Browser That comes preloaded with Boss Linux, we clicked at commands
   Application → Internet → Iceweasal Web Browser.
2. Once the browser window is open. In the address bar, type the HTML document's name along with its path. For instance. If you want to view

try.HTML, which is stored under my documents folder on C drive, you need to type c:\My documents\try.html in the address bar and then hit Enter key.

3. [ Alternatively you can click at File → Open file command in browser window. Then in the Open dialog, firstly choose the place (i.e., folder location) from left pane and select desired .htm or .html file from right pane.
   And you'll be viewing your desired web page.

You can follow the same process for creating and viewing web pages in other Oss and browsers. In the following figure. We have opened html files in Iceweasal web browser (available on Linux desktops) for viewing purposes.

**Fig. 5.2** illustrates how to open an HTML file in a web browser and view rendered webpage.

## LESSON 2: CONTAINER AND EMPTY ELEMENTS

In HTML, there are two types of elements – one that require a starting as well as ending tags, and another that require just a starting tag and not an ending tags. These two types are known as container and empty elements following lines exemplify these two types of tags.

**Container elements.**

This type of HTML elements require pair tags i.e, a starting as well as an ending tag e.g., <title>……</title>, <head>…….</head>.

Notice that an ending tag is similar to that of a starting tag except that it begins with a slash(/) symbol some examples of container elements

<HTML>……..</HTML>
<HEAD>…….</HEAD>
<TITLE>……</TITLE>
<B>……..</B>
<CENTER>……</CENTER>

**Empty Elements**

This type of HTML elements require just a starting tag and not an ending tag e.g., <BR>, <BASE>. Empty elements just carry out their specific job e.g <HR> inserts a horizontal rule and <BR> breaks a line. Some examples of empty elements are;

<BR>
<BASE>
<BASEFONT>
<HR>
<IMG>
<LINK>

**Head Tag**

The Head tag is used to define the document header. The <head> tag contains information about the document, including its title, scripts used style definitions and document descriptions. To add <head> tag, we enter it between <HTML> tags.

For instance
<html>
<head>
</head>
</html>

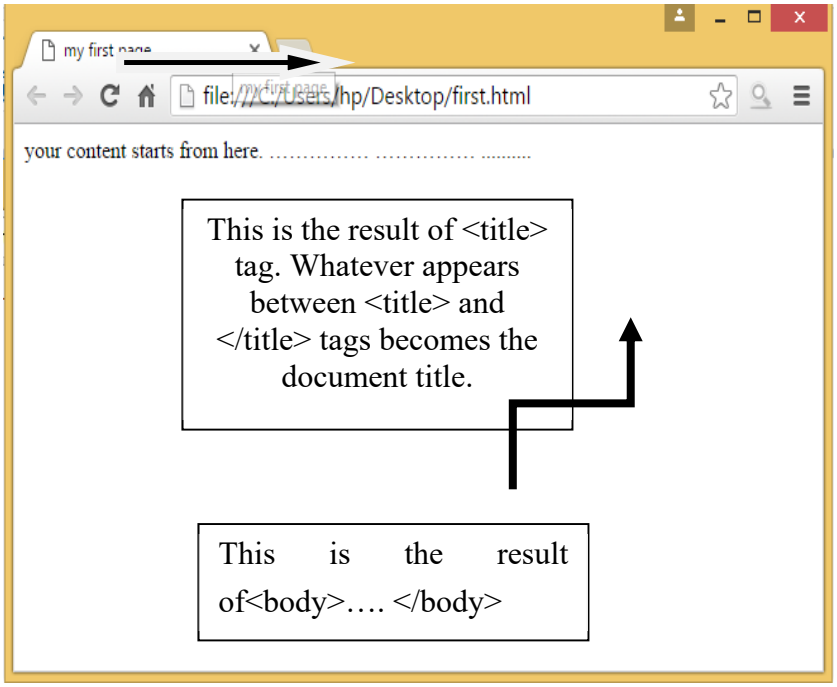
## Title tag

The title element contains your document title and identifies its content in a global context. The title is typically displayed in the title bar at the top of the browser window, but not inside the window itself (see the output of example) . The title is also what is displayed on someone's hotlist or bookmark list, so choose something descriptive, unique, and relatively short. A title is also used to identify your page for search engines.

Example
<HTML>
<head>
<title> my first page</title>
</head>
<body>
……
……
</body>
</html>

## The Body tag

The second and largest – part of our HTML document is the body. <body>……. </body> , which contains the content of your document ( displayed within the text area of your browser window). The body tag defines the document's body. It contains all the contents of an HTML document , such as text, images , lists tables, hyperlinks etc.




## LESSON 3

### HTML Document Structure

A typical HTML document will have the following structure: Document declaration tag. Document header related tags

### Document body related tags

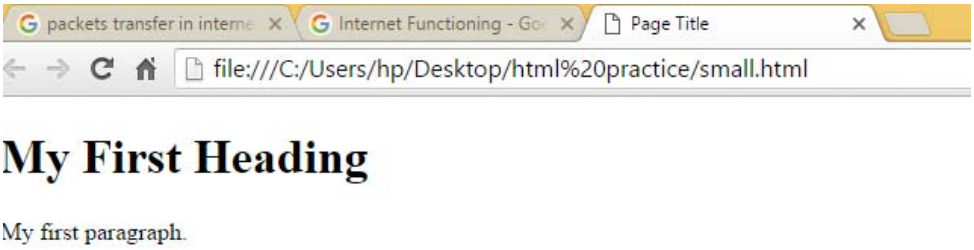
We will study all the header and body tags in subsequent chapters, but for now let's see what is document declaration tag.

**The Declaration :** The declaration tag is used by the web browser to understand the version of the HTML used in the document. Current version of HTML is 5 and it makes use of the following declaration:

There are many other declaration types which can be used in HTML document depending on what version of HTML is being used. We will see more details on this while discussing tag along with other HTML tags.

HTML Example
A small HTML document:

```
<!DOCTYPE html>
<html>
<head>
<title>PageTitle</title>
</head>
<body>
<h1>My First Heading</h1>
<p>My first paragraph.</p>
</body>
</html>
```




**Explanation**

- The DOCTYPE declaration defines the document type to be HTML
- The text between <html> and </html> describes an HTML document
- The text between <head> and </head> provides information about the document.
- The text between <title> and </title> provides a title for the document.

- The text between <body> and </body> describes the visible page content
- The text between <h1> and </h1> describes a heading
- The text between  and  describes a paragraph

Using this description, a web browser can display a document with a heading and a paragraph.

HTML Page Structure
Below is a visualization of an HTML page structure:

```
<html>
<head>
<title>Page title</title>
</head>
<body>
<h1>This is a heading</h1>
<p>This is a paragraph.</p>
<p>This is another paragraph.</p>
</body>
</html>
```

Only the <body> area (the gray area) is displayed by the browser.
The <!DOCTYPE> Declaration
The <!DOCTYPE> declaration helps the browser to display a web page correctly.
There are different document types on the web.
To display a document correctly, the browser must know both type and version.
The doctype declaration is not case sensitive. All cases are acceptable:

```
<!DOCTYPE html>
<!DOCTYPE HTML>
<!doctype html>
<!Doctype Html>
```

**Common Declarations**
HTML5

```
<!DOCTYPE html>
```

HTML 4.01

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

**Capitalization**

Original versions of HTML were case-insensitive and, in fact, very forgiving. This means all of the following examples would be considered the same by the browser:

- <html>
- <HTML>
- <HTml>

That said, HTML4/XHTML is *case-sensitive* and requires all tags to be lowercase. Of the three preceding examples, the browser might properly interpret only the first.

To make it really confusing, HTML5 returns to being case-insensitive. Given the differences between the various versions of HTML currently in use, I recommend using all-lowercase tags.

**NOTE**

A few characters are reserved and given special meaning in HTML. For example, the brackets (< and >) are used to signify HTML tags, while the ampersand (&) is used to begin these entities. If you need to use a bracket within the content of your HTML page, such as when a greater-than symbol is needed, in the case of 3 > 2, you should use the corresponding character entity (>) to do so.

| Character | Numbered | Entity Named Entity |
|---|---|---|
| | &#34 | ; " |
| & | & | & |
| (nonbreaking space) |   |   |
| © | © | © |
| ® | ® | ® |
| é | é | é |
| < | < | < |
| > | > | > |

**Add Comments to an HTML File :** Sometimes you might not want your web site visitors to see personal comments or notes you've added to your web pages. These notes might be directions to another person or reminders to yourself. <**!**-- Remember to update this page after the new product becomes available -->

After the opening bracket, an exclamation mark and two hyphens signify the beginning of a comment. A space should appear after the opening comment code, as well as before the closing comment code.

Comments are not restricted in size but can cover many lines at a time. The end comment code (-->) doesn't need to be on the same line as the beginning comment code. If you forget to close your comment tag, the rest of the page will not appear in your browser. If this happens, don't be alarmed. Simply go back to the code and close that comment. The rest of the page will become visible when you save the file and reload it in the browser.

## LESSON 4

### Background color, text color, link color

By default browsers display text in black on a grey ( or white) background. However, you can change both elements if you want. Some HTML authors select a background color and coordinate it with a change in the color of the text.

You change the color or text ( by text attribute), links ( by link attribute), visited links (by vlink attribute), and active links ( by alink attribute) , active links are the liks that are currently being clicked on) using further attributes of the <body> tag. For example:

<body bgcolor= "#000000" text = "#FFFFFF" link = "#9690cc">

This will create a window with a black background , white text and silvery hyperlinks. Similarly consider the following.

<body bgcolor= "teal" text= "magenta" link = "yellow" vlink= "lime" alink= "red">

It gives you

- The background color is teal
- Text color is magenta
- Links that have not been visited recently are made yellow
- Recently visited links are made lime green
- Links blink briefly red when someone clicks them

**Colors and their RGB values**

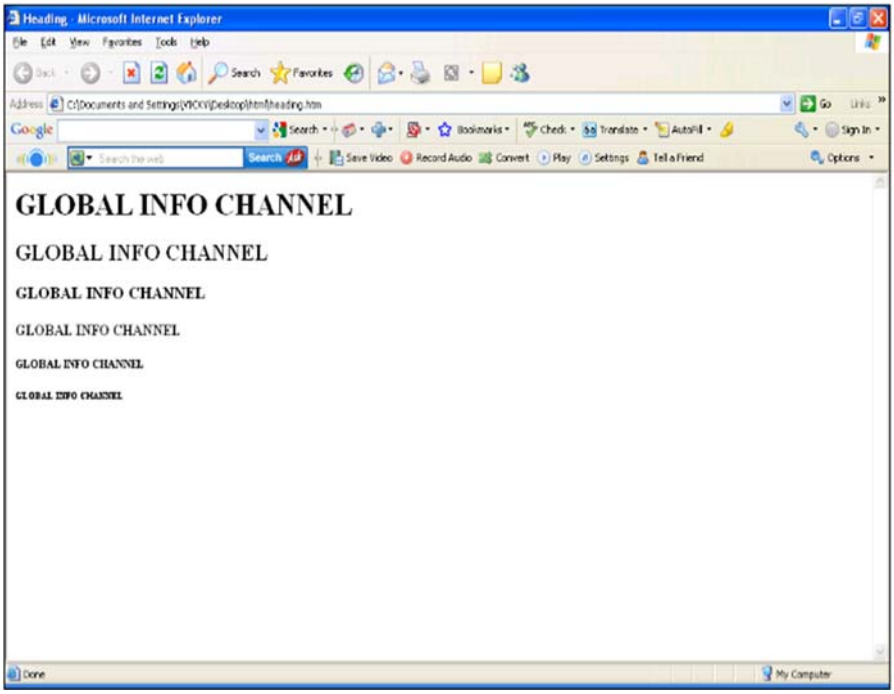
| Color | HTML / CSS Name | Hex Code #RRGGBB | Decimal Code (R,G,B) |
|---|---|---|---|
| | Black | #000000 | (0,0,0) |
| | White | #FFFFFF | (255,255,255) |
| | Red | #FF0000 | (255,0,0) |
| | Lime | #00FF00 | (0,255,0) |
| | Blue | #0000FF | (0,0,255) |
| | Yellow | #FFFF00 | (255,255,0) |
| | Cyan / Aqua | #00FFFF | (0,255,255) |
| | Magenta / Fuchsia | #FF00FF | (255,0,255) |
| | Silver | #C0C0C0 | (192,192,192) |
| | Gray | #808080 | (128,128,128) |
| | Maroon | #800000 | (128,0,0) |
| | Olive | #808000 | (128,128,0) |
| | Green | #008000 | (0,128,0) |
| | Purple | #800080 | (128,0,128) |
| | Teal | #008080 | (0,128,128) |
| | Navy | #000080 | (0,0,128) |

**Heading Element:**

- There are six heading elements (<H1>,<H2>,<H3>,<H4>, <H5>,<H6>).
- All the six heading elements are container tag and requires a closing tag.
- <h1> will print the largest heading.
- <h6> will print the smallest heading.
- Headings are defined with the <h1> to <h6> tags.
- <h1> defines the most important heading. <h6> defines the least important heading.

Examples:

```
<html>
<head><title>heading</title></head>
<body>
<h1> GLOBAL INFO CHANNEL</h1>
<h2> GLOBAL INFO CHANNEL</h2>
<h3> GLOBAL INFO CHANNEL</h3>
<h4> GLOBAL INFO CHANNEL</h4>
<h5> GLOBAL INFO CHANNEL</h5>
<h6> GLOBAL INFO CHANNEL</h6>
</body>
</html>
```



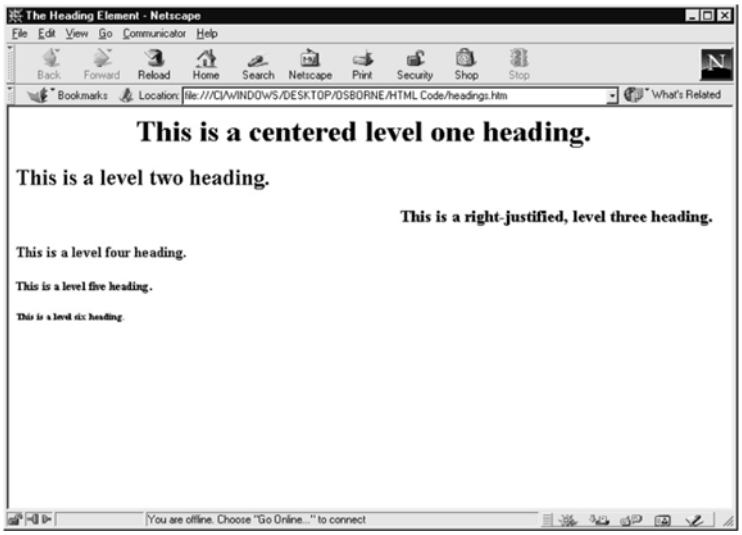

To experiment further, try adding the align attribute to some of the headings and then check your results. Change the level one heading to read:

```
<h1 align="center">This is a centered level one heading.</h1>
```

Now, modify the level three line to read this way:

```
<h3 align="right">This is a right-justified, level three heading.</h3>
```

Save the file and display it in your browser. It should look like figure below.




**Headings Are Important**

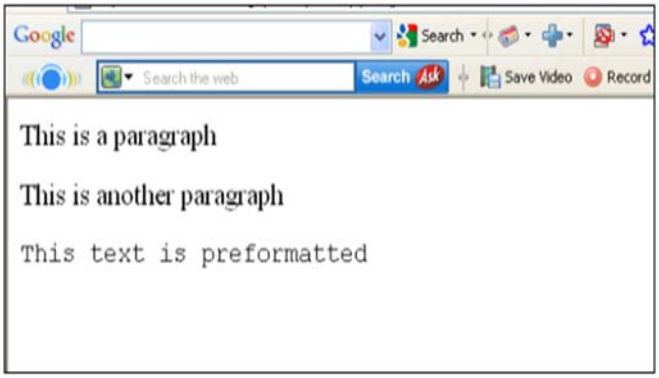
- Use HTML headings for headings only. Don't use headings to make text BIG or bold.
- Search engines use your headings to index the structure and content of your web pages.
- Users skim your pages by its headings. It is important to use headings to show the document structure.
- <h1> headings should be main headings, followed by <h2> headings, then the less important <h3>, and so on.

## LESSON 5

**HTML Paragraph Tag**

- HTML documents are divided into paragraphs.
- Paragraphs are defined with the ** ** tag i.e.

This is a paragraph
This is another paragraph
<pre>This text is preformatted</pre>



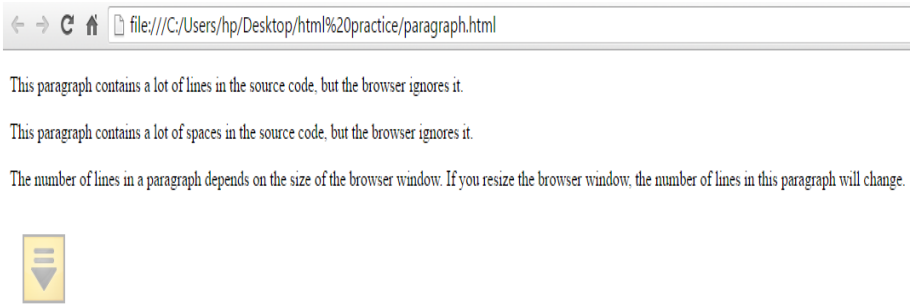

Examples

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

<p>
This paragraph
contains a lot of lines
in the source code,
but the browser
ignores it.
</p>

<p>
This paragraph
contains a lot of spaces
in the source code,
but the browser
ignores it.
</p>
<p>
```

The number of lines in a paragraph depends on the size of the browser window. If you resize the browser window, the number of lines in this paragraph will change.

```
</p>
</body>
```

</html>



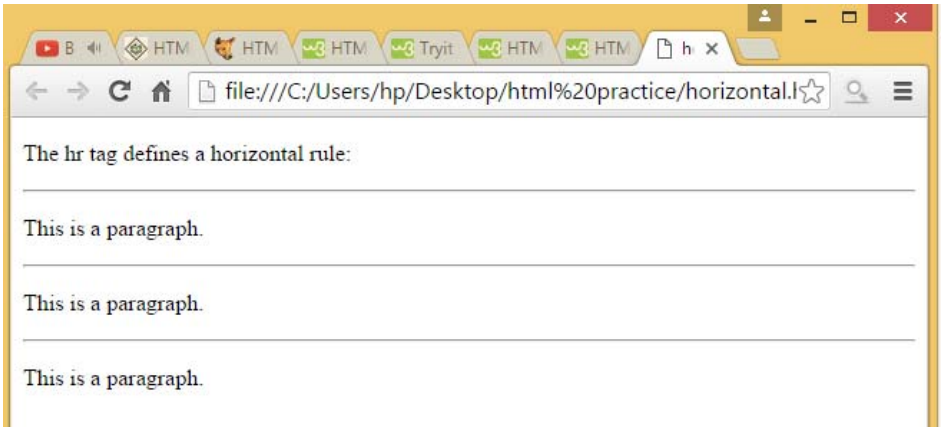

## LESSON 6

### HTML Horizontal Rule

The **<hr>** tag creates a horizontal line in an HTML page.

The <hr> element can be used to separate content:

```
<p>This is a paragraph.</p>
<hr>
<p>This is a paragraph.</p>
<hr>
<p>This is a paragraph.</p>
```




### Line Break Tag <br>

Whenever you use the <br /> element, anything following it starts from the next line. This tag is an example of an empty element, where you do not need opening and closing tags, as there is nothing to go in between them.

The <br /> tag has a space between the characters br and the forward slash. If you omit this space, older browsers will have trouble rendering the line break, while if you miss the forward slash character and just use <br> it is not valid in XHTML

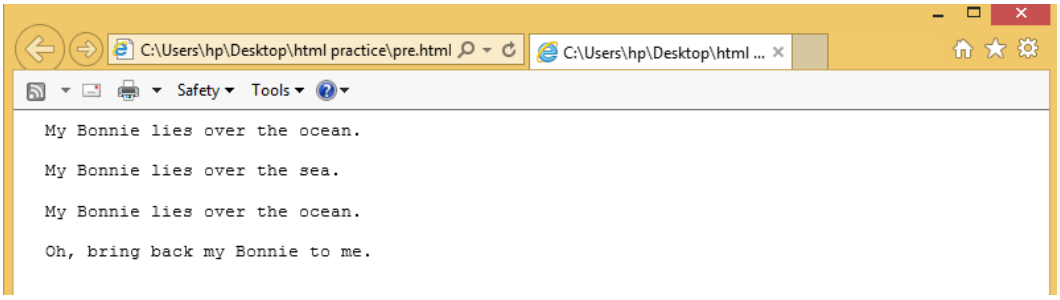
This is<br>a para<br>graph with line breaks

**The HTML <pre> Element**

The HTML <pre> element defines preformatted text.

The text inside a <pre> element is displayed in a fixed-width font (usually Courier), and it preserves both spaces and line breaks:

```
<pre>
  My Bonnie lies over the ocean.
  My Bonnie lies over the sea.
  My Bonnie lies over the ocean.
  Oh, bring back my Bonnie to me.
</pre>
```




## LESSON 7

**Control Text with Character Elements**

The character elements give you some basic controls over how text will display on a Web browser. There's nothing fancy here just meat-and-potatoes text control. As you work with these elements, you will notice that some of them are named for how a character actually will look when it appears on a browser (for example, the bold <b> element for bold text). These are called *physical* elements because they describe the physical appearance of the characters. Other text elements derive their names from the intended function or purpose of the text (for example, the  element also

will display bold typeface, but the element's name stands for strongly emphasized text). When a character element is named this way, it is called a *logical* element because it describes the logical function of the text.

**Display Italicized Text**

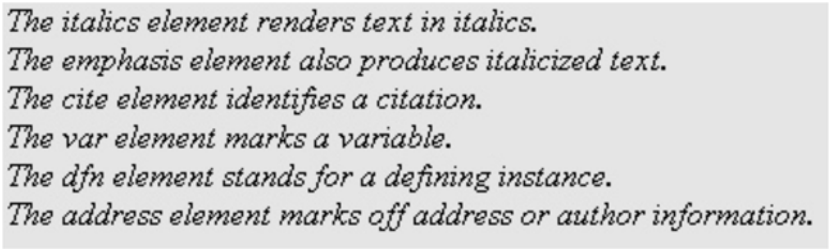
If you want to display text in italic, you have no less than six different elements to choose from. Interestingly, each has its own distinct purpose:

_ ** ** The *italics* element is a physical element for displaying italicized text.

- ** ** The *emphasis* element is a logical element for emphasizing important portions of a document. It generally displays italicized text.
- **<cite> </cite>** The *citation* element identifies a portion of your document as a reference to an outside source.
- **<var> </var>** This logical element indicates a *variable*, as might be used in computer code.
- **<dfn> </dfn>** This logical element identifies a portion of text as a *defining instance* of a term. It also generally displays in italic.
- **<address> </address>** You might use this logical element (which also renders your text in italics) to set apart your address or personal information at the bottom of a web page. This element also generally adds a line break before and after the address.

To see each of these elements in action, use your template to create a new HTML document, and save it as text.htm. Then add the following lines in the <body> </body> portion of the page to get the results shown in the following illustration:

```
<i>The italics element renders text in italics.</i><br />
<em>The emphasis element also produces italicized text.</em><br />
<cite>The cite element identifies a citation.</cite><br />
<var>The var element marks a variable.</var><br />
<dfn>The dfn element stands for a defining instance.</dfn><br />
<address>The address element marks off address or author
information.</address><br />
```

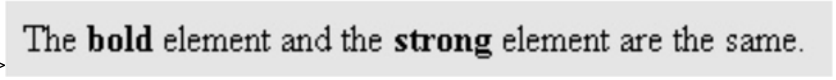



**Display Bold Text**

Two different elements allow you to display boldface text:

- ** ** The *bold* element is a physical element that allows you to render boldface text.
- ** ** The *strong* element also displays in bold. Strictly speaking, this logical element indicates a heavier or stronger emphasis than  does, but there is usually no difference in how the two look in a browser. Try this line for a side-by-side comparison, like the one shown in the following illustration:

The bold element and the strong element are the same.<br />



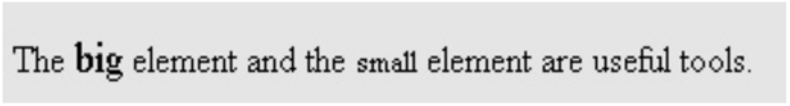

**Display Big and Small Text**

If you want a portion of text to appear slightly larger or smaller than the surrounding characters, you can use the following elements:

- **<big> </big>** This element displays text one font size larger than the surrounding text (for more on font sizes, see Unit 3).
- **<small> </small>** This element reduces text by one font size.

To see the difference, put this line in your page. The following illustration shows what you should see on your screen:

The <big>big</big> element and the <small>small</small> element are useful tools.<br>



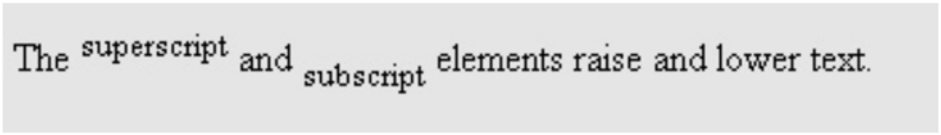
**Create Superscripts and Subscripts**

Sometimes you might have a need to add a superscript to your text. Maybe you'll be documenting a source and want to insert a footnote reference at the end of a quotation. Or perhaps you're a science nut and want to describe the molecular structure of water or carbon dioxide. Then you'll need to know how to do a subscript.

- **<sup> </sup>** This element creates a superscript.
- **<sub> </sub>** This element forces text to display as a subscript.

To see these elements in action, insert the following line into your page. It will create a line with both superscripts and subscripts, as in the following illustration:

The <sup>superscript</sup> and <sub>subscript</sub> elements raise and lower text.<br />

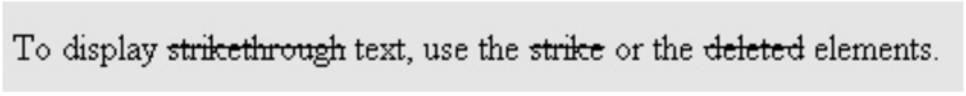



**Display Strikethroughs**

If you want your text to display with a horizontal line drawn through it, you again have more than one element to choose from:

- _**<s> </s>** This *physical* element will display text as "strikethrough."
- **<strike> </strike>** The *strike* element is another physical element for producing strikethrough text.
- **<del> </del>** The *deleted* element is a logical element that indicates (by a strikethrough) that the text has been deleted but left in the page.Try this sample line to see how these three elements compare. Your screen should resemble the following illustration.

To display <s>strikethrough</s> text, use the <strike>strike </strike> or the <del>deleted</del> elements.<br />



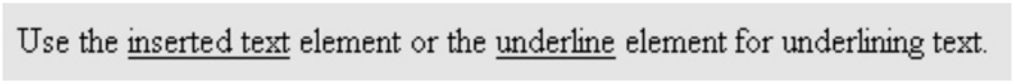

**Display Underlined Text**

If you want text to be displayed underlined, you again have more than one option:

_ **<u> </u>** The underline element will render text with a line underneath.
_ <ins> </ins> The inserted text element is a logical element that indicates

text that has been inserted since the document was written.Although either one of these elements will produce underlined text, the <ins>element is supported only by Internet Explorer 4 and higher. If you want your text underlined, you are better off using the <u> element. The following illustration reflects the results you will get with these elements:

Use the <ins>inserted text</ins> element or the
<u>underline</u> element for underlining text.




**In summary**

| | |
|---|---|
| <b> | Defines bold text |
| <big> | Defines big text |
|  | Defines emphasized text |
| <i> | Defines italic text |
| <small> | Defines small text |
|  | Defines strong text |
| <sub> | Defines subscripted text |
| <super> | Defines superscripted text |
| <ins> | Defines inserted text |
| <del> | Defines deleted text |
| <tt> | Defines teletype text |
| <u> | Defines underline text |
| <strike> | Defines strike text |

**Create a Scrolling Marquee**

A very simple way to add animation to a web page is with the <marquee> </marquee> element. Any text enclosed inside this element will scroll across the page, much the

same way as a bar of text will move across your TV screen when the station wants to pass on some news without breaking into a program.

The <marquee> element's greatest disadvantage is that it is proprietary (not part of the HTML specification) developed by Internet Explorer. Only IE browsers support this element.

**Use <marquee> to Make Text Scroll**

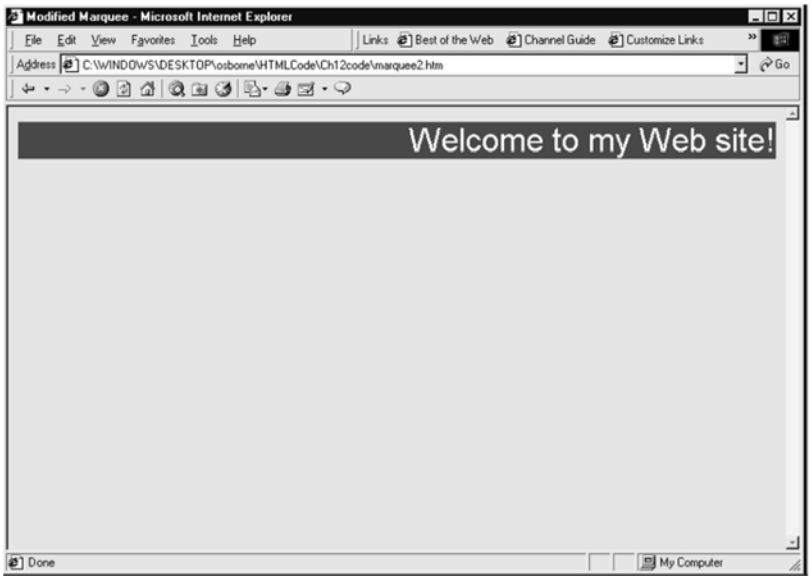
Create a blank HTML page by opening template.htm and saving it as marquee.htm; then add the following line in the <body> </body> portion of the page: <marquee>Welcome to my Web Site!</marquee> . _ behavior=" " This attribute enables you to tell the browser how to scroll your text. "Scroll" (the default value) allows the text to scroll endlessly. "Slide" will bring the text in from either the left or right side of the screen and slide it to the opposite margin. "Alternate" will keep the text moving between the left and right margins (as if it were bouncing of both sides.

**How to Do Everything with HTML**

- **loop=" "** You can use this attribute to determine how many times you want the text to scroll, slide, or alternate. "Infinite" (the default value) keeps the text moving indefinitely. A numerical value will cause the action to be performed for that number of times. For instance, loop="10" will scroll the message ten times.
- **direction=" "** With this attribute you can choose the direction in which your text will move. The values are "left" and "right."
- **scrollamount=" "** If the motion of the scrolling text seems too jerky, you can use scrollamount to smooth it out. Add a numerical value to represent the number of pixels you want the text to move each time. For example, scrollamount="2" will move the text a distance of two pixels each time it moves.
- **scrolldelay=" "** The scrolldelay attribute also can be used to control the movement of the text, this time by delaying each separate movement for a specified period of time. The values for this attribute must be in milliseconds;

thus, a delay of 100 milliseconds would be equivalent to one tenth of a second. Some other attributes that can be used to modify the size and positioning of the marquee are height=" " (height of the marquee in pixels), width=" " (width of the marquee in pixels), hspace=" " (adds space on the sides of the marquee), vspace=" " (adds space above and below the marquee), and bgcolor=" " (enables you to specify a background color for the marquee). To modify the appearance and behavior of the simple marquee created in the previous example, try the following steps:

1. Set the marquee's background color to red:
   <marquee bgcolor="#ff0000">
   #ff0000 is the hexadecimal code for the color red. For more on color and hex codes, see Unit 4.
2. Change the direction so the text scrolls from left to right:
   <marquee bgcolor="#ff0000" direction="right">
3. Let's slow down the motion of the text so it's not so hard of your visitors' eyes:
   <marquee bgcolor="#ff0000" direction="right"
   scrolldelay="100" scrollamount="2">
4. To reduce the annoyance factor, set the text to scroll across the window and stop at the opposite side:
   <marquee bgcolor="#ff0000" direction="right"
   scrolldelay="100" scrollamount="2"
   behavior="slide">
5. Just for fun, let's throw in a little style with CSS:
   <marquee bgcolor="#ff0000" direction="right"
   scrolldelay="100" scrollamount="2"
   behavior="slide" style="color: white;
   font-family: arial; font-size: 24pt;">
6. Save the page as marquee2.htm and view it in Internet Explorer.
   Your finished marquee should resemble this:




**Get Precise Control with the style Attribute**

Cascading Style Sheets are somewhat of a misnomer for what you are using here. The term "style sheet" implies something separate from the page, and you will see later why the term "cascading" is used. What you will work with here is called an *inline style*. It works about the same way as what you have learned already with HTML. You learned the three most important terms in HTML: elements, attributes, and values. An attribute goes inside an element and describes what you want to modify. A value is attached to the attribute and gives the details of how you are going to modify the element.

Inline style sheets are similar, but add one more term to the mix: *property.* Property is a term you will become very familiar with as you work with CSS. It functions like an attribute; that is, it tells the browser what aspect of an element you are modifying. For example, some properties you might use in adjusting a font are font-family, font-size, margin-left, margin-right, color, and so forth. You will see that you have a much larger selection of properties to work with in CSS, and you can do many more things with them. Using inline styles is similar to using any other attribute inside an element. However, there are a few important differences in syntax:

- Property-value combinations must be enclosed in the quotation marks that follow the style attribute:

- Properties and values must be separated by a colon:

- Property-value combinations must be separated by a semicolon:

For example, to choose an Arial font and display it in green, your tag would look like this:

This is a green Arial font

To demonstrate the versatility of CSS, try telling your browser to display a paragraph in a 24pt, Verdana font, and have the font display in navy with a left margin of 1.3 inches.

1. Once you've chosen a paragraph to modify, remove any size, face, or color attributes from the opening  tag.
2. Insert the style= attribute within the tag. This tells the browser that you are using a style sheet:

3. To set the font, choose the font-family property:

4. Now specify the font style you want to display:

5. Tell the browser to display a 17pt font. Be sure to separate each property-value combination from the next one by a semicolon:

6. Now choose a left margin of 1.3 inches:
   <p style="font-family: verdana;
   font-size: 17pt; margin-left: 1.3in; ">
7. Finally, set the font color to navy:
   <p style="font-family: verdana;
   font-size: 17pt; margin-left: 1.3in;
   color: navy;">
8. Save your page and display your newly styled paragraph in your browser.
   Quite a difference!

## LESSON 8

### Organize Your Material with Lists

HTML lists are a great way to organize material on your web site. Whether you want to list the ingredients of your favorite recipe, create a page of links with their descriptions, or offer step-by-step instructions for washing a dog, you can use list elements to put your material in order.

### Create a Bulleted List

You need two elements to create a bulleted (or unordered) list:

_ <ul> </ul> The *unordered list* element enables you to create the list.
_ <li> </li> You specify individual items on the list with the *list item* element.

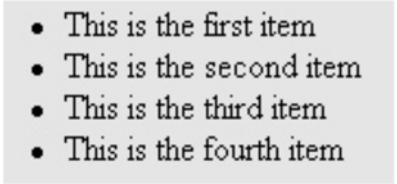
Use your template to create a new HTML document. Save it in your HTML reference directory as ulist.htm. Then, in between the <body> </body> tags:

1. Type an opening *unordered list* tag: <ul>.
2. Next, add a set of *list item* tags: <li> </li>.
3. For each new list item, add another set of <li> </li> tags.
4. When the list is complete, type a closing </ul> tag.

The code for a simple unordered list might look something like what you see here:

```
<html>
<head><title>Unordered Lists</title><head>
<li>This is the second item</li>
<li>This is the third item</li>
<li>This is the fourth item</li>
</ul>
</body>
</html>
```

When you display your page in a browser, it will produce a bulleted list, like the one in the illustration here:



You'll notice when you display the list that your browser supplies a solid disc as the bullet for each item. If you would prefer a circle or a square, you can specify it by using the type=" " attribute. To specify the type of bullet for the entire list, put the attribute inside the opening <ul> tag:

- To specify a square: **<ul type="square">**
- To specify a circle: **<ul type="circle">**
- To specify a disc: **<ul type="disc">**


You also can control the type of bullet for each separate item by putting the attribute inside the <li> tag. For example, the following code will supply a different bullet for each item, as reflected in the illustration that follows:

```
<ul>
<li type="square">This supplies a square.</li>
<li type="circle">This supplies a circle.</li>
<li type="disc">This supplies a disc.</li>
</ul>
```

- This supplies a square.
- This supplies a circle.
- This supplies a disc.

**Create a Multi-Level List**

To create a list with multiple levels, you simply nest one or more unordered lists inside each other. Save ulist.htm with the new name: ulist2.htm. Now make a multi-level list from it:

1. Inside the first set of <li> tags, type <ul>.
2. Add a line that says <li>This is a sub point.</li>.
3. Add another line that reads: <li>This is another sub point.</li>.
4. Close out the sub list with </ul>.

Your list should look like this:

```
<html>
<head><title>Unordered Lists</title><head>
<body>
<ul>
<li>This is the first item
<ul>
<li>This is a sub point</li>
<li>This is another sub point</li>
</ul></li>
<li>This is the second item
<ul>
<li>This is a sub point</li>
<li>This is another sub point</li>
</ul></li>
<li>This is the third item
<ul>
<li>This is a sub point</li>
<li>This is another sub point</li>
</ul></li>
</ul>
</body>
</html>
```

If you want to add another level of sub points, you can do it by nesting another complete list inside each <li> </li> element where you want the next level.

**Create Ordered Lists**

What if you want to display an outline with numbers and letters delineating the various points? Or perhaps you want to create a list of instructions in numbered sequence. Lists that arrange items in sequence by number or letter are called *ordered lists* in HTML and are quite similar in their structure to unordered lists.

To create an ordered list, you need the following:

- **<ol> </ol>** The ordered list element
- **<li> </li>** The list item element

Use <ol> to Create a Numbered List

You can create a simple numbered list by enclosing a series of list items inside the ordered list element, as in the following sample page. Use your template to open a new HTML page and save it as olist.htm. Then type in this code:

```
<html>
<head><title>Ordered Lists</title></head>
<body>
<ol>
<li>Ordered lists display items with numbers.</li>
<li>But HTML doesn't sort the items.</li>
<li>It only numbers them.</li>
<li>You have to do the arranging yourself.</li>
</ol>
</body>
</html>
```

After you save your page, display it in your browser. You should see a numbered

1. Ordered lists display items with numbers.
2. But HTML doesn't sort the items.
3. It only numbers them.
4. You have to do the arranging yourself.

When you display this in a browser, you will notice that the list is numbered in sequence. The actual format depends on the browser you use, but normally it is simply

a numbered list. One important difference with numbered lists is that if you nest the lists to create multiple levels, the browser uses the same numbering system throughout; it does not automatically change to a different one with each level. You must use the type attribute to specify any changes you want.

**Use the type Attribute to Specify Numbers or Letters**

Just as you can instruct the browser to use different types of bullets in an unordered list, you can tell the browser what types of letters or numbers to use. This is very useful if you want to produce an outline in HTML, as you have a nice range of choices available. You specify numbers or letters with the type=" " attribute, just as you did for unordered lists. To produce this:

**<ul type="I">** Capitalized Roman numerals
**<ul type="i">** Lowercase Roman numerals
**<ul type="1">** Numbers (default)
**<ul type="A">** Capital letters
**<ul type="a">** Lowercase letters

As in unordered lists, if you place the type attribute inside the <ol> tag, you can specify your preferences for the entire list. If you place the attribute inside a <li> tag, it will change only that particular list item.

**Use the start Attribute to Choose a Starting Number**

What if you want to begin a list at a point other than *1* or *A*? All you need to do is include the start=" " attribute at the point where you want to change the number or letter. The browser will start the list at the number you choose and continue numbering from that point. For example, if you want to start a list at the number 23, you might do it this way:

```
<ol type="1" start="23">
```

Create a new HTML document and save it as olist2. htm. Now, try typing the following code and displaying it in your browser to see what a list like this might look like if you did it with Roman numerals, starting at number 10:

```
<html>
<head><title>Ordered Lists</title></head>
```

```
<body>
<ol type="I" start="10">
<li>This is the tenth list item.</li>
<li>This is the eleventh list item.</li>
<li>This is the twelfth list item.</li>
<li>This is the thirteenth list item.</li>
</ol>
</body>
</html>
```

When you display this in your browser, the list will begin with the Roman numeral "X," as in the following:

X. This is the tenth list item.
XI. This is the eleventh list item.
XII. This is the twelfth list item.
XIII. This is the thirteenth list item.

## LESSON 9

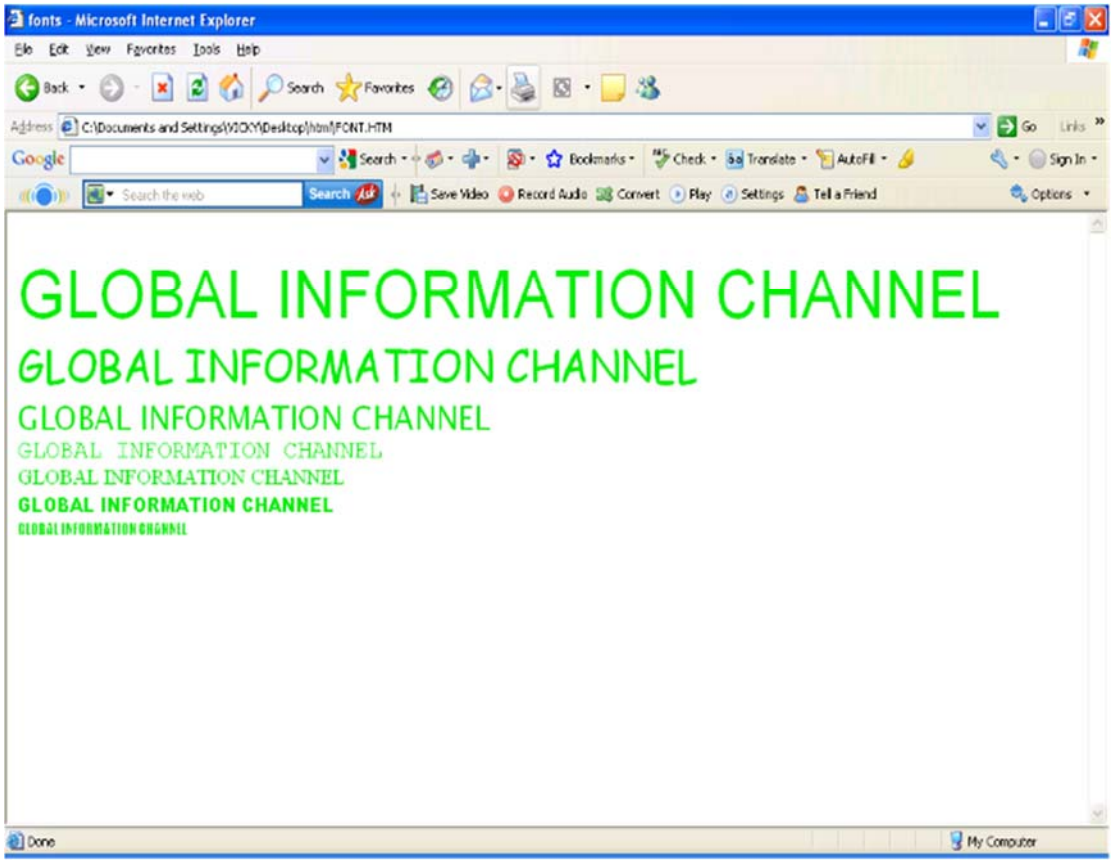
**Font Tag**

- This element is used to format the **size, typeface** and **color** of the enclosed text.
- The commonly used fonts for web pages are Arial, Comic Sans MS , Lucida Sans Unicode, Arial Black, Courier New, Times New Roman, Arial Narrow, Impact, Verdana.
- The size attribute in font tag takes values from **1 to 7**.

```
<html>
<head><title> fonts</title></head>
<body>
<br><font color="green" size="7" face="Arial"> GLOBAL INFORMATION
CHANNEL </font>
<br><font color="green" size="6" face="Comic Sans MS "> GLOBAL
INFORMATION CHANNEL </font>
<br><font color="green" size="5" face="Lucida Sans Unicode"> GLOBAL
```

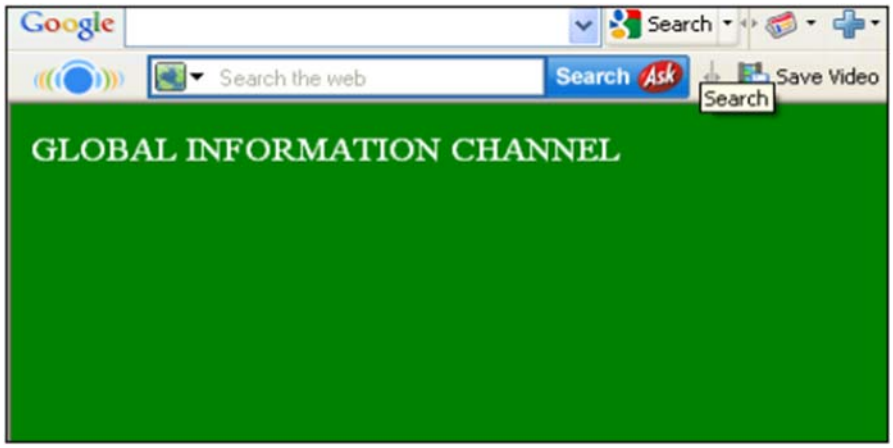
```
INFORMATION CHANNEL </font>
<br><font color="green" size="4" face="Courier New"> GLOBAL
INFORMATION CHANNEL </font>
<br><font color="green" size="3" face="Times New Roman"> GLOBAL
INFORMATION CHANNEL </font>
<br><font color="green" size="2" face="Arial Black"> GLOBAL
INFORMATION CHANNEL </font>
<br><font color="green" size="1" face="Impact"> GLOBAL
INFORMATION CHANNEL </font>
</body>
</html>
```

**Result of above tags**




**Background & Text Color Tag**

The attribute bgcolor is used for changing the back ground color of the page.




<body bgcolor="Green" >

Text is use to change the color of the enclosed text.

<body text="White">

Text Alignment Tag

It is use to alignment of the text.

Left alignment <align="left">

Right alignment <align="right">

Center alignment <align="center">

## LESSON 10

### Hyperlink Tag, Image Tag and Table

A hyperlink is a reference (an address) to a resource on the web.

Hyperlinks can point to any resource on the web: an HTML page, an image, a sound file, a movie, etc. The HTML anchor element <a>, is used to define both hyperlinks and anchors.<a href="url">Link text</a> The **href attribute** defines the link address. <a href="http://www.globalinfochannel/">Visit globalinfochannel!</a>

Example

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

<p><a href="http://www.w3schools.com/html/">Visit our HTML
tutorial</a></p>

</body>
</html>
```

## Image Tag

To display an image on a page, you need to use the src attribute. src stands for "source". The value of the src attribute is the URL of the image you want to display on your page.

It is a empty tag.

Example

```
<IMG SRC ="url">
<IMG SRC="picture.gif“>
<IMG SRC="picture.gif“ HEIGHT="30" WIDTH="50">
<html><body>
<p><img src="file:///C:/WINDOWS/tulups.jpg"
align="left" width="48" height="48"> </p>
<p><img src ="file:///C:/WINDOWS/tulups.jpg"
align="right" width="48" height="48"></p>
</body></html>
```

```
<HTML>
<<body background="file:///C:/WINDOWS/Soap%20Bubbles.bmp"
text="white">
<br><br><br>
<h2> Background Image!</h2>
</BODY></HTML>
```

**HTML Table Tag**

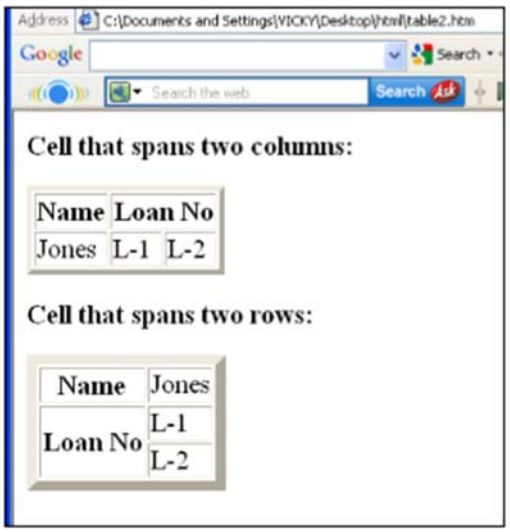
| | |
|---|---|
| <table> | used to create table |
| <tr> | table is divided into rows |
| <td> | each row is divided into data cells |
| <th> | Headings in a table |
| <Caption> | caption to the table |
| <colgroup> | Defines groups of table columns |
| <col> | Defines the attribute values for one or more columns in a table |
| <thead> | Defines a table head |
| <tbody> | Defines a table body |
| <tfoot> | Defines a table footer |
| <Cellspacing> | amount of space between table cells. |
| <Cellpadding> | space around the edges of each cell |
| <Colspan> | No of column working with will span |
| <rowspan> | No of rows working with will span |
| <Border> | attribute takes a number |

Example

```
<html>
<body>
<h3>Table without border</h3>
<table>
<tr> <td>MILK</td>
<td>TEA</td>
<td>COFFEE</td> </tr>
<tr> <td>400</td>
<td>500</td>
<td>600</td> </tr>
</table>
</body>
</html>
```


**Table Code with Colspan & Rowspan**

```
<html><body>
<h4>Cell that spans two
columns:</h4>
<table border="4">
<tr> <th>Name</th>
<th colspan="2">Loan No</th>
</tr>
<tr> <td>Jones</td>
<td>L-1</td>
<td>L-2</td> </tr> </table>
<h4>Cell that spans two
rows:</h4>
<table border="8">
<tr> <th>Name</th>
<td>Jones</td></tr><tr>
<th rowspan="2">Loan No</th>
<td>L-1</td></tr><tr>
<td>L-2</td></tr></table>
</body></html>
```



**Cellpadding,Image & Backcolor Code**

```
<html><body>
<h3>Without cellpadding:</h3>
<table border="2" bgcolor="green">
<tr> <td>Jones</td>
<td>Smith</td></tr>
<tr> <td>Hayes</td>
<td>Jackson</td></tr></table>
<h4>With cellpadding:</h4>
<table border="8"
cellpadding="10"
```


```
background="file:///C:/WINDOWS/FeatherTexture.bmp">
<tr> <td>Jones</td>
<td>Smith</td></tr>
<tr> <td>Hayes</td>
<td>Jackson</td></tr></table>
</body></html>
```

## LESSON 11

**Frames and Forms**

The different parts that the screen screen is divided into are called Frames Frames can be created by using the <FRAMESET> and </FRAMESET>

TWO Attribute:

- Cols
- Rows

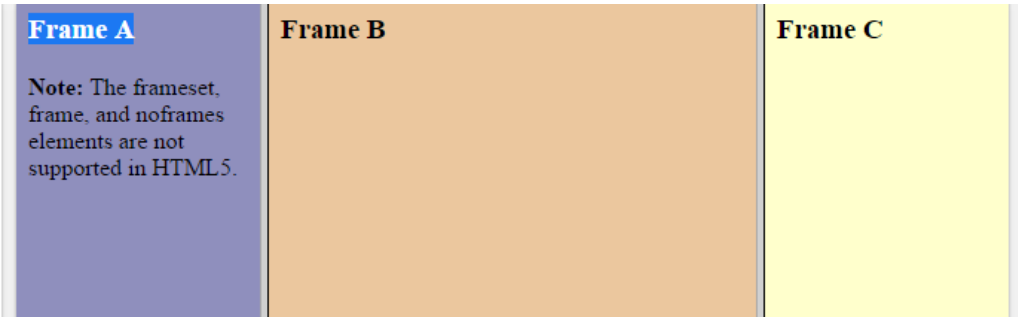
Example: File name – Main.html

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

```
<frameset cols="25%,*,25%">
  <frame src="frame_a.htm">
  <frame src="frame_b.htm">
  <frame src="frame_c.htm">
</frameset>

</html>
```

**<NOFRAMES>**

Tag is used to specify alternate text for browsers that do not support frames




**HTML Form**

A form is an area that can contain form elements. Form elements are elements that allow the user to enter information in a form. like text fields, textarea fields, drop-down menus, radio buttons and checkboxes etc A form is defined with the **<form>** tag.

The syntax:-

A form is defined with the <form> tag.

```
<form>
<input>
</input>
</form>
```

Input

The most used form tag is the <input> tag. The type of input is specified with the type attribute.

Text Fields

Text fields are used when you want the user to type letters, numbers, etc. in a form.

Example:

```
<HTML><BODY>
<FORM> First name:
<input type="text" name="X"> <br>
Last name: <input type="text" name="Y">
</BODY></FORM>
</HTML>
```

**Radio Buttons**

Radio Buttons are used to select one of a limited number of choices.

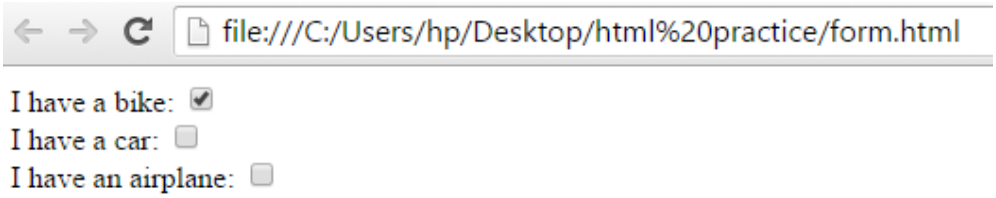
```
<FORM>
<input type="radio" name="s" value="male"> Male <br>
<input type="radio" name="se" value="female"> Female </form>
```

Checkboxes

Checkboxes are used when you want the user to select one or more options of a limited number of choices**.**

```
<HTML>
<BODY>
<FORM> I have a bike:
<input type="checkbox" name="v1" value="Bike"> <br>
I have a car:
<input type="checkbox" name="v2" value="Car"> <br>
I have an airplane: <input type="checkbox" name="v3" value="Airplane">
</form>
<BODY>
</HTML>
```

Output



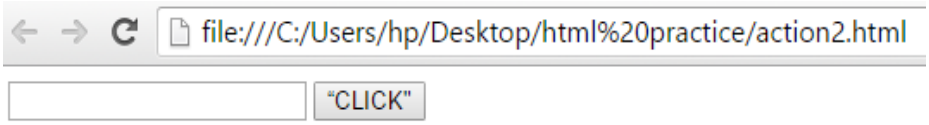

**The Form's Action Attribute and the Submit Button**

When the user clicks on the "Submit" button, the content of the form is sent to another

file. The form's action attribute defines the name of the file to send the content to. The file defined in the action attribute usually does something with the received input.

Example:

```
<HTML>
<BODY>
<input type="text" name="user">
 <input type="submit" value="CLICK">
</BODY>
</HTML>
 </form>
```



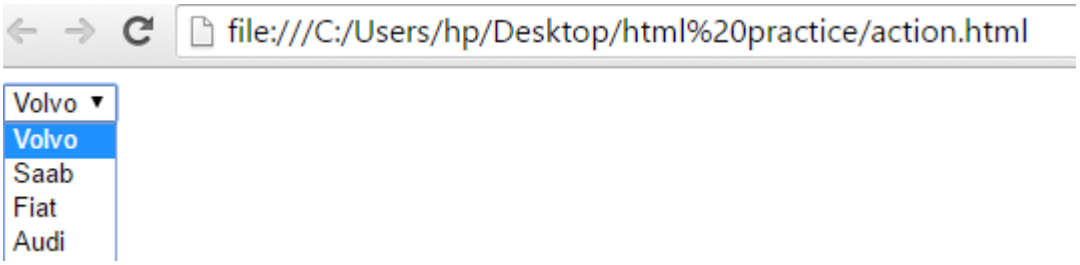

**Simple drop down box**

A drop-down box is a selectable list.

Example

```
<html>
<body>
<form action="">
<select name="cars">
<option value="vo">Volvo</option>
<option value="sa">Saab</option>
<option value="fi">Fiat</option>
<option value="audi">Audi</option>
</select>
</form>
</body>
</html>
```

Output




**MULTI LINE Text area**

(a multi-line text input control). A user can write text in the text-area. In a text-area you can write an unlimited number of characters.

```
<HTML>
<BODY>
<TEXTAREA ROWS="6" COLS="40" NAME="x">
PLEASE ENTER YOUR COMMENTS HERE.
</TEXTAREA>
</BODY>
</HTML>
```

**Text Boxes**

Text boxes are probably the most frequently used controls in forms

Example:

```
<HTML>
<BODY>
Enter your name :<input Type-"Text" Name="N">
Your favorite web site:<BR>
<INPUT TYPE="TEXT" NAME="url" size="50"  value="http://">
</BODY>
</HTML>
```

**Password controls**

Example:

```
<HTML>
<BODY>
<FORM>
Enter your password:<input Type="password" Name="pas">
</FORM>
</BODY>
</HTML>
```

**Menu controls**

Menu controls are like the drop-down lists used in windows dialog boxes. Such menu controls can be created using the <SELECT> and the </SELECT> Tags.

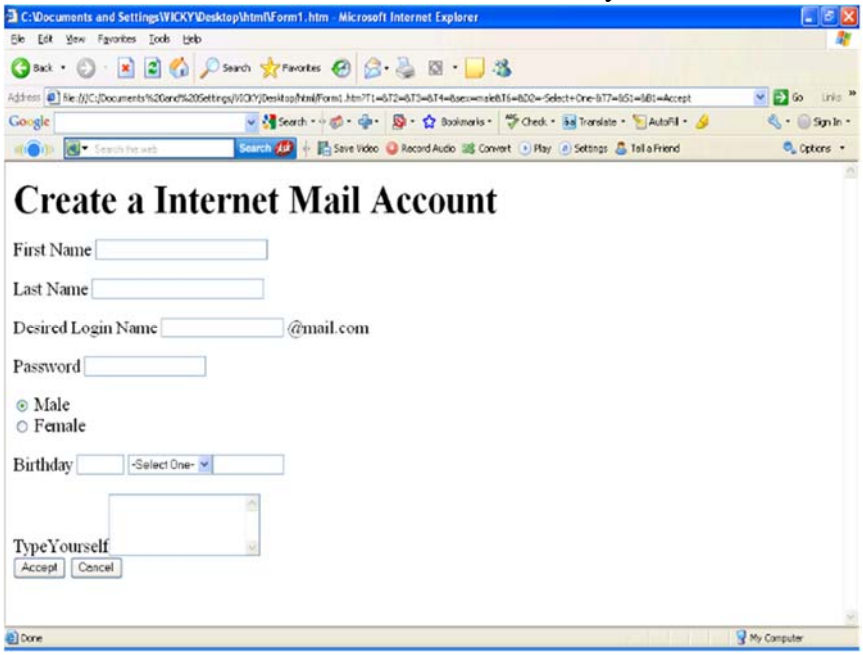
```
<HTML>
<BODY>
<SELECT NAME="VEG" SIZE="5">
<OPTION> C & C++
<OPTION> ORACLE
<OPTION>JAVA
<OPTION>MS OFFICE
<OPTION>DOS
</SELECT>
</BODY>
</HTML>
```

**Form Tags**

| | |
|---|---|
| <form> | Defines a form for user input |
| <input> | used to create an input field |
| <text> | Creates a single line text entry field |
| <textarea> | Defines a text-area (a multi-line text input control) |
| <password> | Creates a single line text entry field. And the characters entered are shown as asterisks (*) |
| <label> | Defines a label to a control |
| <option> | Creates a Radio Button. |
| <select> | Defines a selectable list (a drop-down box) |
| <button> | Defines a push button |
| <value> | attribute of the option element. |
| <checkbox> | select or unselect a checkbox |
| <dropdown box> | A drop-down box is a selectable list |

**Example**

```
<html><body><form>
<h1>Create a Internet Mail Account</h1>
<p>First Name <input type="text" name="T1" size="30"></p>
<p>Last Name <input type="text" name="T2" size="30"></p>
<p>Desired Login Name <input type="text" name="T3" size="20">
@mail.com</p>
<p>Password <input type="password" name="T4" size="20"></p>
<input type="radio" checked="checked" name="sex" value="male" />
Male</br>
<input type="radio" name="sex" value="female" /> Female
<p>Birthday <input type="text" name="T6" size="05">
<select size="1" name="D2">
<option>-Select One-</option>
<option>January</option>
<option>February</option>
<option>March</option> </select>
<input type="text" name="T7" size="10"></p>
TypeYourself<textarea rows="4" name="S1" cols="20"></textarea>
<br><input type="submit" value="Accept" name="B1"> <input type="reset“
value="Cancel" name="B2"></br> </form></body></html>
```



## LESSON 12

### Introduction to HTML5

HTML5 is the fifth revision and newest version of the HTML standard. It offers new features that provide not only rich media support, but also enhance support for creating web applications that can interact with the user, his/her local data, and servers, more easily and effectively than was possible previously.

Some HTML5 features remain unsupported by some browsers. However, Gecko, and by extension, Firefox, has very good support for HTML5, and work continues toward supporting more of its features. Gecko began supporting some HTML5 features in version 1.8.1. You can find a list of all of the HTML5 features that Gecko currently supports on the main HTML5 page. For detailed information about multiple browsers' support of HTML5 features, refer to the CanIUse website.

### Declaring that the document contains HTML5 mark-up with the HTML5 doctype

The doctype for HTML5 is very simple. To indicate that your HTML content uses HTML5, simply use: <!DOCTYPE html>

Doing so will cause even browsers that don't presently support HTML5 to enter into standards mode, which means that they'll interpret the long-established parts of HTML in an HTML5-compliant way while ignoring the new features of HTML5 they don't support.This is much simpler than the former doctypes, and shorter, making it easier to remember and reducing the amount of bytes that must be downloaded.

### Declaring the character set with the <meta charset>

The first thing done on a page is usually indicating the character set that is used. In previous versions of HTML, it was done using a very complex <meta> element. Now, it is very simple:

```
<meta charset="UTF-8">
```

Place this right after your <head>, as some browsers restart the parsing of an HTML document if the declared charset is different from what they had anticipated. Also, if you are not currently using UTF-8, it's recommended that you switch to it in your Web

pages, as it simplifies character handling in documents using different scripts.

Note that HTML5 restricts the valid charset to that compatible with ASCII and using at least 8 bits. This was done to tighten security and prevent some types of attacks.

HTML5 Example:

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Title of the document</title>
</head>

<body>
Content of the document.....
</body>

</html>
```

New HTML5 Elements

The most interesting new elements are:

New **semantic** elements like <header>, <footer>, <article>, and <section>.

New form **control attributes** like number, date, time, calendar, and range.

New **graphic** elements: <svg> and <canvas>.

New **multimedia** elements: <audio> and <video>.

**Elements Removed in HTML5**

The following HTML4 elements have been removed from HTML5:

| Element | Use instead |
|---|---|
| <acronym> | <abbr> |
| <applet> | <object> |
| <basefont> | CSS |
| <big> | CSS |

| | |
|---|---|
|  | CSS |
| <dir> | <ul> |
| <font> | CSS |
| <frame> | |
| <frameset> | |
| <noframes> | |
| <strike> | CSS |
| <tt> | CSS |

**New Semantic/Structural Elements**

HTML5 offers new elements for better document structure:

| Tag | Description |
|---|---|
| <article> | Defines an article in the document |
| <aside> | Defines content aside from the page content |
| <bdi> | Defines a part of text that might be formatted in a different direction from other text |
| <details> | Defines additional details that the user can view or hide |
| <dialog> | Defines a dialog box or window |
| <figcaption> | Defines a caption for a <figure> element |
| <figure> | Defines self-contained content, like illustrations, diagrams, photos, code listings, etc. |
| <footer> | Defines a footer for the document or a section |
| <header> | Defines a header for the document or a section |

| | |
|---|---|
| <main> | Defines the main content of a document |
| <mark> | Defines marked or highlighted text |
| <menuitem> | Defines a command/menu item that the user can invoke from a popup menu |
| <meter> | Defines a scalar measurement within a known range (a gauge) |
| <nav> | Defines navigation links in the document |
| <progress> | Defines the progress of a task |
| <rp> | Defines what to show in browsers that do not support ruby annotations |
| <rt> | Defines an explanation/pronunciation of characters (for East Asian typography) |
| <ruby> | Defines a ruby annotation (for East Asian typography) |
| <section> | Defines a section in the document |
| <summary> | Defines a visible heading for a <details> element |
| <time> | Defines a date/time |
| <wbr> | Defines a possible line-break |

**New Form Elements**

| Tag | Description |
|---|---|
| <datalist> | Defines pre-defined options for input controls |
| <keygen> | Defines a key-pair generator field (for forms) |
| <output> | Defines the result of a calculation |

## HTML5 Audio

HTML5 provides a standard for playing audio files. Before HTML5, there was no standard for playing audio files on a web page.Before HTML5, audio files could only be played with a plug-in (like flash).The HTML5 <audio> element specifies a standard way to embed audio in a web page.

To play an audio file in HTML, use the **<audio>** element:

```
<audio controls>
  <source src="horse.ogg" type="audio/ogg">
  <source src="horse.mp3" type="audio/mpeg">
Your browser does not support the audio element.
</audio>
```

The **controls** attribute adds audio controls, like play, pause, and volume.Text between the <audio> and </audio> tags will display in browsers that do not support the <audio> element. Multiple **<source>** elements can link to different audio files. The browser will use the first recognized format.

File format Media type
MP3 audio/mpeg
Ogg audio/ogg
Wav audio/wav

### HTML5 Video

Before HTML5, there was no standard for showing videos on a web page.Before HTML5, videos could only be played with a plug-in (like flash).The HTML5 <video> element specifies a standard way to embed a video in a web page.

To show a video in HTML, use the <video> element:

```
<video width="320" height="240" controls>
  <source src="movie.mp4" type="video/mp4">
  <source src="movie.ogg" type="video/ogg">
Your browser does not support the video tag.
</video>
```

The controls attribute adds video controls, like play, pause, and volume. It is a good idea to always include width and height attributes. If height and width are not set, the

browser does not know the size of the video. The effect will be that the page will change (or flicker) while the video loads. Text between the <video> and </video> tags will only display in browsers that do not support the <video> element. Multiple <source> elements can link to different video files. The browser will use the first recognized format.

To start a video automatically use the autoplay attribute:

```
<video width="320" height="240" autoplay>
  <source src="movie.mp4" type="video/mp4">
  <source src="movie.ogg" type="video/ogg">
Your browser does not support the video tag.
</video>
```

**HTML5 Canvas**

The HTML <canvas> element is used to draw graphics on a web page.The graphic to the left is created with <canvas>. It shows four elements: a red rectangle, a gradient rectangle, a multicolor rectangle, and a multicolor text. The HTML <canvas> element is used to draw graphics, on the fly, via scripting (usually JavaScript). The <canvas> element is only a container for graphics. You must use a script to actually draw the graphics. Canvas has several methods for drawing paths, boxes, circles, text, and adding images.

Canvas Examples

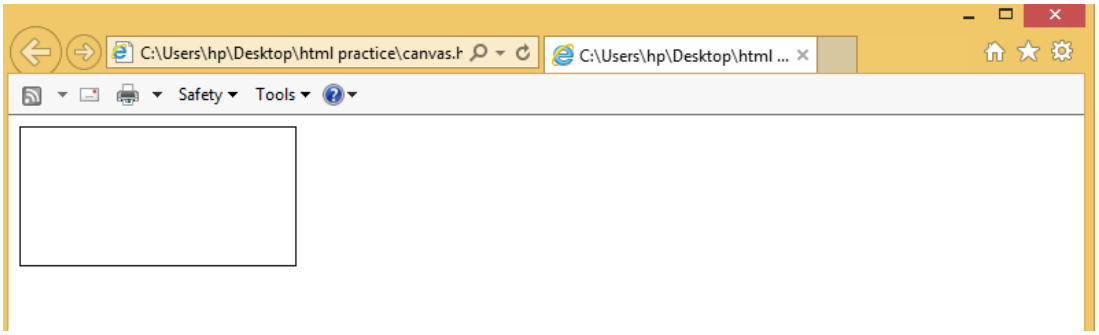
A canvas is a rectangular area on an HTML page. By default, a canvas has no border and no content.

The markup looks like this:

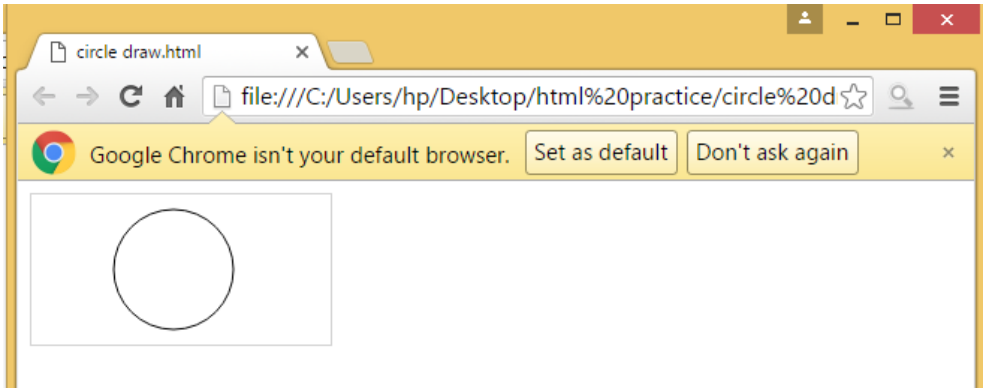
```
<canvas id="myCanvas" width="200" height="100"></canvas>
```

Canvas Example

```
<canvas id="myCanvas" width="200" height="100" style="border:1px solid
#000000;">
</canvas>
```




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

<canvas id="myCanvas" width="200" height="100" style="border:1px solid
#d3d3d3;">
Your browser does not support the HTML5 canvas tag.</canvas>

<script>
var c = document.getElementById("myCanvas");
var ctx = c.getContext("2d");
ctx.beginPath();
ctx.arc(95,50,40,0,2*Math.PI);
ctx.stroke();
</script>

</body>
</html>
```



**Drag and Drop**

Drag and drop is a very common feature. It is when you "grab" an object and drag it to a different location.

In HTML5, drag and drop is part of the standard, and any element can be draggable.

Example

```
<!DOCTYPE HTML>
<html>
<head>
<style>
#div1 {width:350px;height:70px;padding:10px;border:1px solid #aaaaaa;}
</style>
<script>
function allowDrop(ev) {
   ev.preventDefault();
}

function drag(ev) {
   ev.dataTransfer.setData("text", ev.target.id);
}

function drop(ev) {
   ev.preventDefault();
   var data = ev.dataTransfer.getData("text");
   ev.target.appendChild(document.getElementById(data));
}
</script>
</head>
<body>

<p>Drag the Class 9 engineering :</p>

<div id="div1" ondrop="drop(event)"
ondragover="allowDrop(event)"></div>
```

```
<br>
<img id="drag1" src="img_logo.gif" draggable="true"
ondragstart="drag(event)" width="336" height="69">

</body>
</html>
```

It might seem complicated, but lets go through all the different parts of a drag and drop event.

- First of all: To make an element draggable, set the draggable attribute to true:
  `<img draggable="true">`
- Then, specify what should happen when the element is dragged. In the example above, the ondragstart attribute calls a function, drag(event), that specifies what data to be dragged. The dataTransfer.setData() method sets the data type and the value of the dragged data:

  ```
  function drag(ev) {
  ev.dataTransfer.setData("text", ev.target.id);
  }
  ```

  In this case, the data type is "text" and the value is the id of the draggable element ("drag1").
- The ondragover event specifies where the dragged data can be dropped. By default, data/elements cannot be dropped in other elements. To allow a drop, we must prevent the default handling of the element. This is done by calling the event.preventDefault() method for the ondragover event:
  *event*.preventDefault()
- When the dragged data is dropped, a drop event occurs. In the example above, the ondrop attribute calls a function, drop(event):

```
function drop(ev) {
ev.preventDefault();
var data = ev.dataTransfer.getData("text");
 ev.target.appendChild(document.getElementById(data));
}
```

**Code explained:**

- Call preventDefault() to prevent the browser default handling of the data (default is open as link on drop)
- Get the dragged data with the dataTransfer.getData() method. This method will return any data that was set to the same type in the setData() method
- The dragged data is the id of the dragged element ("drag1")
- Append the dragged element into the drop element

You can easily support dragging and dropping of objects on a web page by setting up event handlers for the ondragstart, ondragover, and ondrop events, as in

**Example**

```
Example  Dragging and dropping objects
<!DOCTYPE HTML>
<html>
<head>
<title>Drag and Drop</title>
<script src='OSC.js'></script>
<style>
#dest {
background:lightblue;
border :1px solid #444;
width :320px;
height :100px;
padding :10px;
}
</style>
</head>
<body>
<div id='dest' ondrop='drop(event)' ondragover='allow(event)'></div><br>
Drag the image below into the above element<br><br>
<img id='source1' src='image1.png' draggable='true'
ondragstart='drag(event)'>
<img id='source2' src='image2.png' draggable='true'
ondragstart='drag(event)'>
```

```
<img id='source3' src='image3.png' draggable='true'
ondragstart='drag(event)'>
<script>
function allow(event)
{
event.preventDefault()
}
function drag(event)
{
event.dataTransfer.setData('image/png', event.target.id)
}
function drop(event)
{
event.preventDefault()
var data=event.dataTransfer.getData('image/png')
event.target.appendChild(O(data))
}
</script>
</body>
</html
```

**HTML Global Attributes**

HTML attributes give elements meaning and context. The global attributes below can be used on any HTML element.

| Attribute | Description |
|---|---|
| accesskey | Specifies a shortcut key to activate/focus an element |
| class | Specifies one or more classnames for an element (refers to a class in a style sheet) |
| contenteditable | Specifies whether the content of an element is editable or not |
| contextmenu | Specifies a context menu for an element. The |

| | |
|---|---|
| | context menu appears when a user right-clicks on the element |
| data-* | Used to store custom data private to the page or application |
| dir | Specifies the text direction for the content in an element |
| draggable | Specifies whether an element is draggable or not |
| dropzone | Specifies whether the dragged data is copied, moved, or linked, when dropped |
| hidden | Specifies that an element is not yet, or is no longer, relevant |
| id | Specifies a unique id for an element |
| lang | Specifies the language of the element's content |
| spellcheck | Specifies whether the element is to have its spelling and grammar checked or not |
| style | Specifies an inline CSS style for an element |
| tabindex | Specifies the tabbing order of an element |
| title | Specifies extra information about an element |
| translate | Specifies whether the content of an element should be translated or not |

## LESSON 14

**HTML Lab work**

| | |
|---|---|
| 1. | Write a HTML document for following list-<br>1. Color<br>Red<br>Green<br>Blue<br>2. Fruits<br>i. Apple<br>ii. Banana<br>iii. Mango |
| 2. | Create a HTML doc for changing background, insert image and change font of text. |
| 3. | Write a HTML doc to create a table and also rows span and columns span. |
| 4. | Create an HTML doc<br>(a) Hyperlink<br>(b)Marquee |
| 5. | Write an HTML document to create frame. |
| 6. | Write an HTML document to create from using radio button, check box and image button. |
| 7. | Write an HTML document to create three different paragraphs with different color. |
| 8. | Write an HTML document to print the Following.<br>The family has following facility<br>1. Own house<br>2400 square feet living<br>Separate bungalow<br>Car shed available<br>2. Car<br>Maruti omini Car<br>Registration Number TN 728435<br>1994 Model<br>3. Farm |

<table>
<tr><td></td><td>Coconut Groves<br>35 acres<br>Mango groves</td></tr>
<tr><td>9.</td><td>Create an HTML document for following.<br>Order list can be defined.<br>1. Keyboard<br>2. Mouse<br>3. Monitor<br>Order List can have verify of type<br>a. Lower case letter<br>b. Upper case letter<br>iii. Lower case Roman Number<br>iv. Upper case Roman Number</td></tr>
<tr><td>10.</td><td>Create a table with the following data:-
<table>
<tr><th>Community</th><th>Male</th><th>Female</th></tr>
<tr><td>OBC</td><td>35</td><td>40</td></tr>
<tr><td>SC/ST</td><td>23</td><td>12</td></tr>
<tr><td>GEN</td><td>12</td><td>10</td></tr>
</table></td></tr>
</table>

1. Write an HTML document for the following list-

1. Color
   - Red
   - Green
   - Blue
2. Fruits

i Apple

ii Banana

iii Mango

**Source code:-**

```
 <html>
<body>
<font size=”5”>
<ol>
<li>Color</li>
<ul>
<li>Red</li>
<li>Green</li>
<li>Blue</li>
</ul>

<li>Fruits</li>
<ul>
<li type="i">Apple</li>
<li type="i">Banana</li>
<li type="i">Mango</li>
</ul>
</ol>
</font>
</body>
</html>
```

Output:-

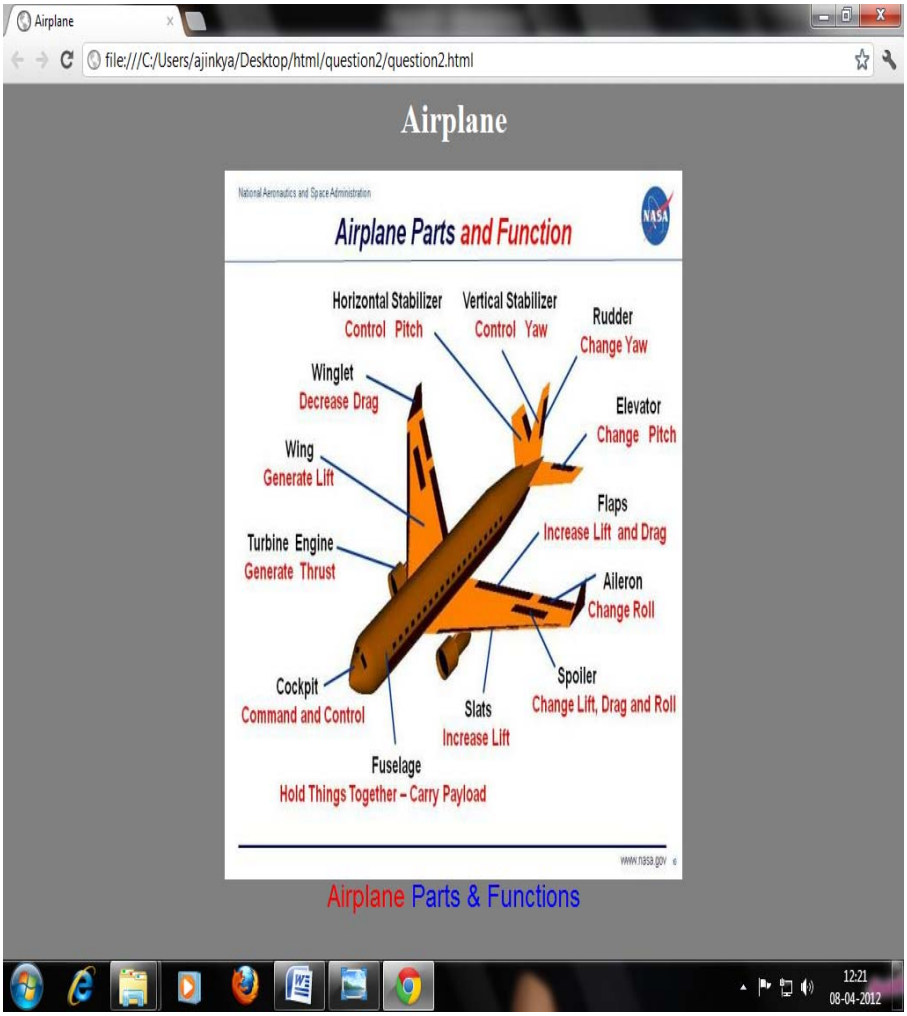
2. **Create an H.T.M.L document for changing background, insert image, change font of text.**

Source code:-

```
<html>

<body>
<head>

<title>Airplane</title>

</head>
<body bgcolor="grey" text="white">
```

```
<h1><center>Airplane</center></h1>
<p><center>

<img src="airplane.jpg" width=520 height=540 /></br>
<font size="5" face="arial" color="red"> Airplane</font>
<font size="5" face="arial" color="blue">Parts & Functions</font>
</center>

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

Output:-



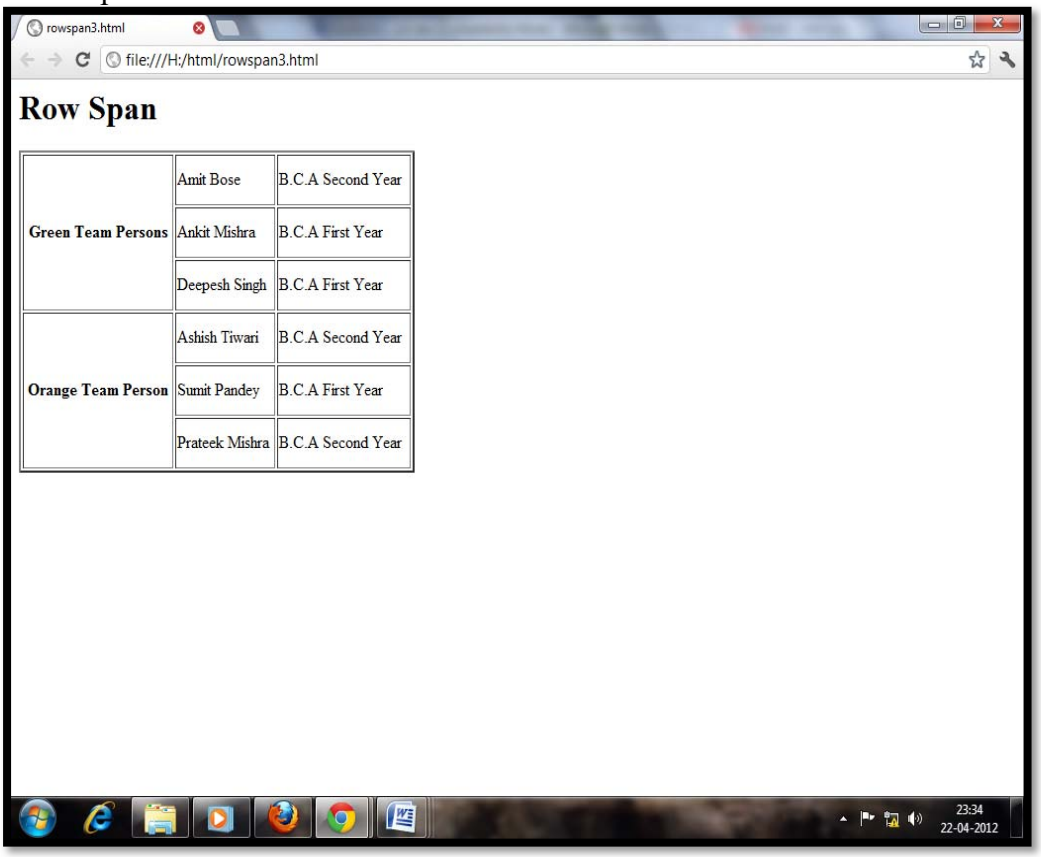
**3. Write an HTML doc to create a table and also rows span and columns span.**

- Source code of table with rows span:-

```
<html>
<body>

<h1>Row Span</h1>

<table border=”2” width=”400” height=”300”>
<tr>
   <th rowspan=3>Green Team Persons</th>
   <td>Amit Bose</td>
    <td>B.C.A Second Year</td>
</tr>

<tr>
   <td>Ankit Mishra</td>
   <td>B.C.A First Year</td>
</tr>

<tr>
   <td>Deepesh Singh</td>
   <td>B.C.A First Year</td>
</tr>

<tr>
   <th rowspan=3 >Orange Team Person</th>
   <td>Ashish Tiwari</td>
   <td>B.C.A Second Year</td>
</tr>

<tr>
   <td>Sumit Pandey</td>
   <td>B.C.A First Year</td>
</tr>
```

```
<tr>
   <td>Prateek Mishra</td>
   <td>B.C.A Second Year</td>
</tr>

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

Output:-

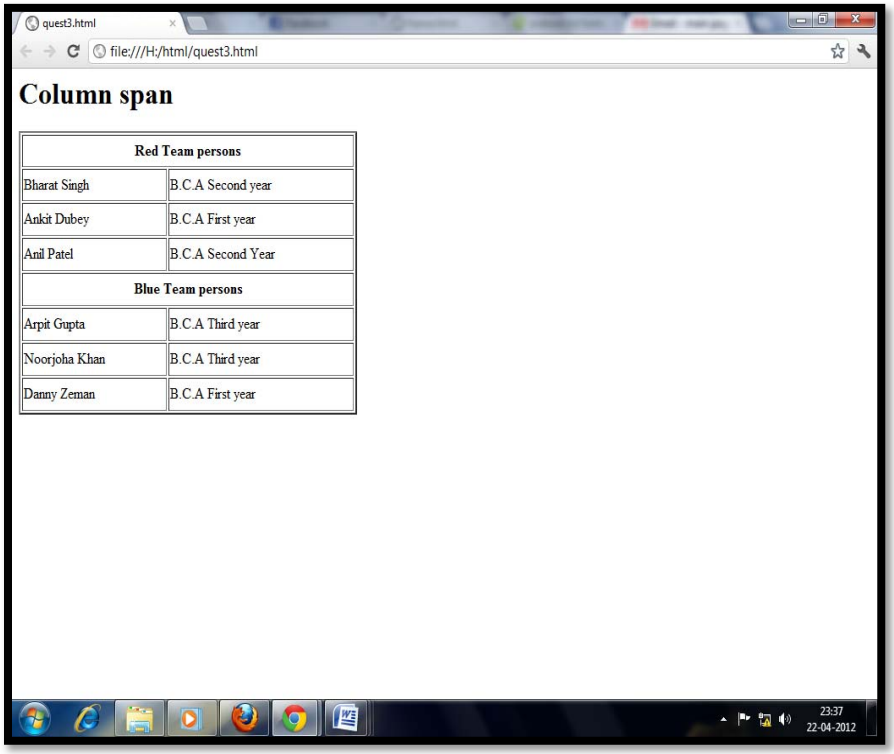


- Source code of Table with columns span

```
<html>
<body>
<h1>Column span</h1>
<table border=”2” width=”400” height=”300”>
        <tr>
        <th colspan=2>Red Team persons</th>
        </tr>

        <tr>
        <td>Bharat Singh</td>
        <td>B.C.A Second year</td>
        </tr>

        <tr>
        <td>Ankit Dubey</td>
        <td>B.C.A First year</td>
        </tr>

        <tr>
        <td>Anil Patel</td>
        <td>B.C.A Second Year</td>
        </tr>

        <tr>
        <th colspan=2>Blue Team persons</th>
        </tr>

        <tr>
        <td>Arpit Gupta</td>
```

```
<td>B.C.A Third year</td>
</tr>

<tr>
<td>Noorjoha Khan</td>
<td>B.C.A Third year</td>
</tr>

<tr>
<td>Danny Zeman</td>
<td>B.C.A First year</td>
</tr>
</table>
</body>
</html>
```

Output:-



4. **Create a HTML document**

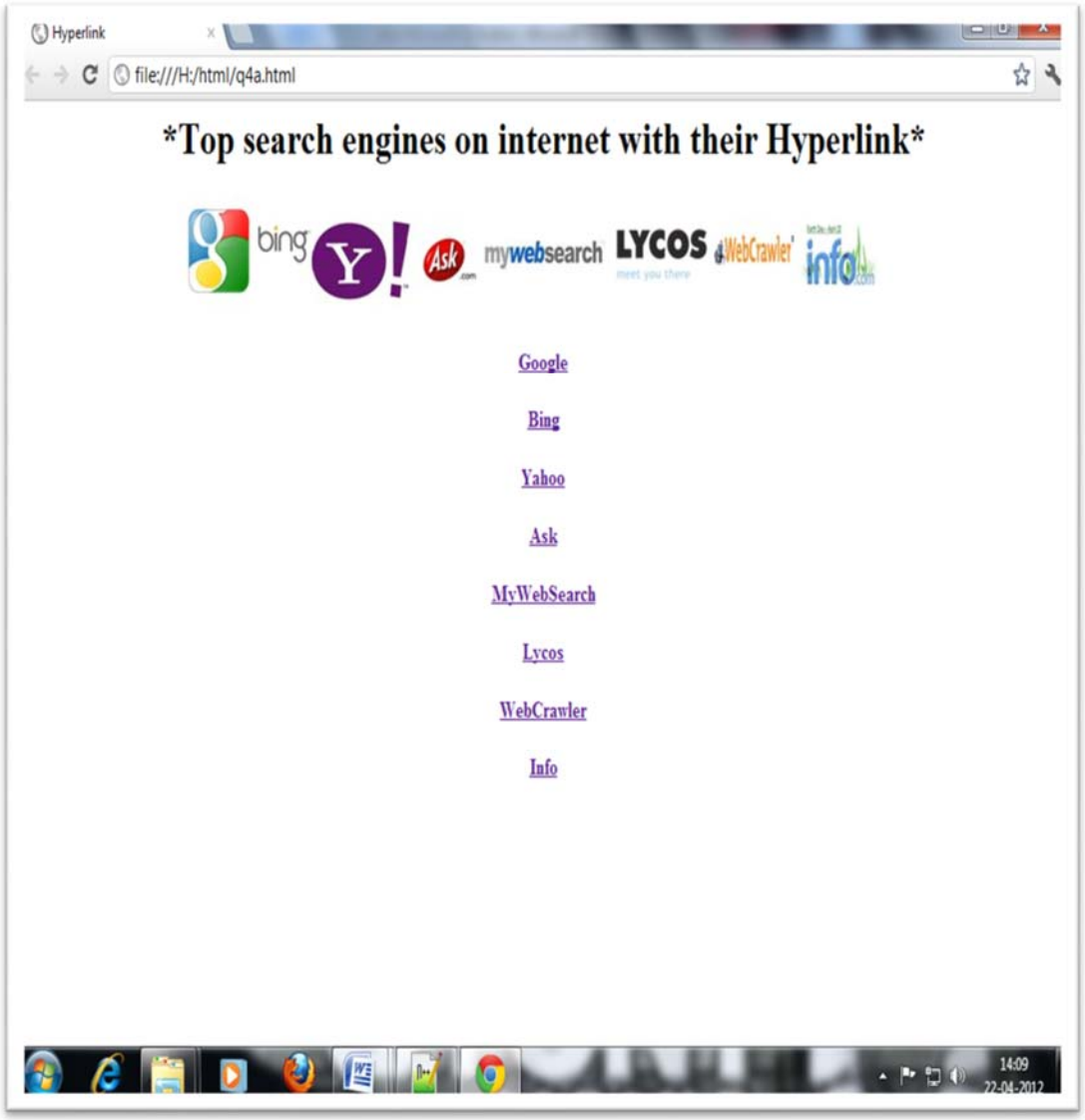
a) Hyperlink

b) Marquee

Source code of Hyperlink HTML document:

```
<html>
<head>
<title>Hyperlink</title>
</head>
<body>
<h1><center>*Top search engines on internet with their
Hyperlink*</center></h1>
<center><img src="icon.jpg"  width=695 height=87 /></center>
<h4 align=center><a href="http://www.google.com">Google</a></h4>
<h4 align=center><a href="http://www.bing.com">Bing</a></h4>
<h4 align=center><a href="http://in.yahoo.com">Yahoo</a></h4>
<h4 align=center><a href="http://www.ask.com">Ask</a></h4>
<h4
align=center><ahref="http://www.mywebsearch.com/">MyWebSearch</a></
h4>
<h4 align=center><a href="http://www.lycos.com/">Lycos</a></h4>
<h4 align=center><a
href="http://www.webcrawler.com/">WebCrawler</a></h4>
<h4 align=center><a href="http://www.info.com/">Info</a></h4>
</body>
</html>
```

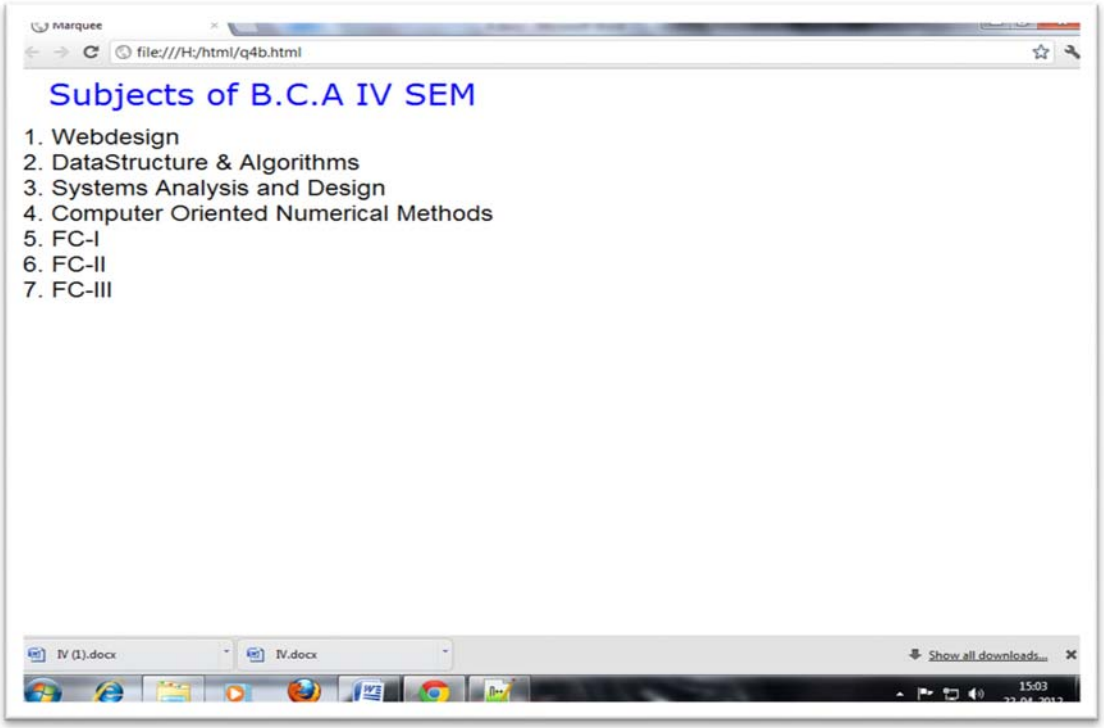
**Output:-**


Hyperlink
file:///H:/html/q4a.html
*Top search engines on internet with their Hyperlink*
Google
Bing
Yahoo
Ask
MyWebSearch
Lycos
WebCrawler
Info
14:09
22-04-2012

a) Source code of Marquee HTML document:-

```
<html>
<head>
<title>Marquee</title>
</head>
<body>
<p>
<font size="6" face="verdana" color="blue">
<marquee>Subjects of B.C.A IV SEM</marquee>
</font>
<p/>
<p>
<font size="5" face="arial" color="black">
1. Web design</br>
2. DataStructure & Algorithms</br>
3. Systems Analysis and Design</br>
4. Computer Oriented Numerical Methods</br>
5. FC-I</br>
6. FC-II</br>
7. FC-III</br>
</font>
</p>
</body>
</html>
```

**Output:-**




**5. Write an HTML document to create frame.**

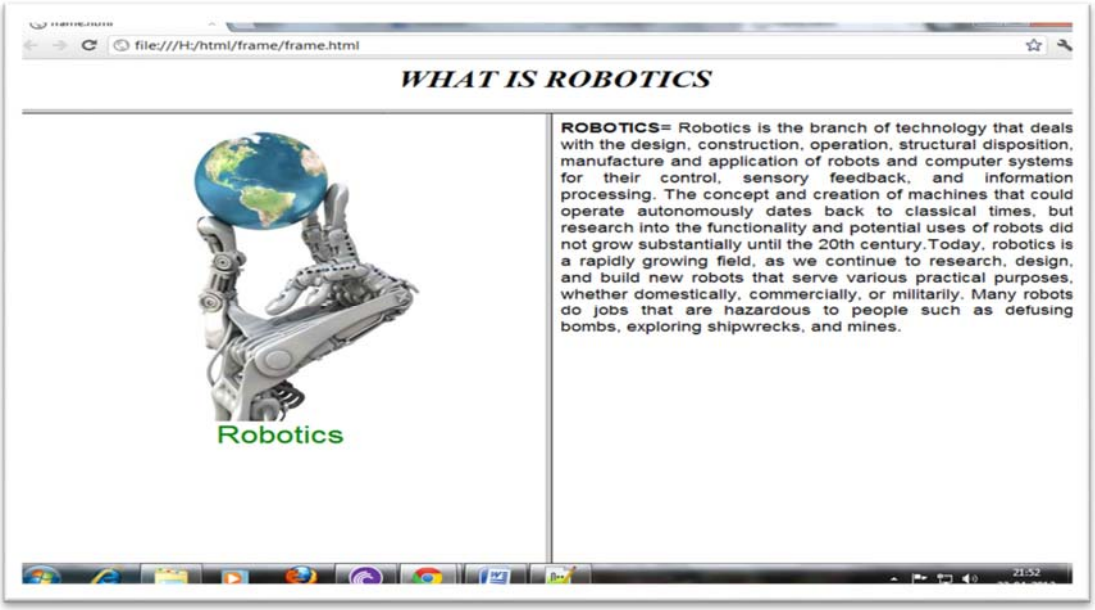
**Source code:-**

```
<html>
<frameset rows="10%, 90%" width=500 height=500>
<frameset>
 <frame name="frame1" src="frame_a.html">
</frameset>

 <frameset cols="50%, 50%">
 <frame name="frame2" src="frame_b.html">
 <frame name="frame3" src="frame_c.html">
</frameset>
</frameset>
</html>
```

**Description:-**

In each frame for the appearance we have to add a invidual page hyperlink as given above in source code. We have mention "frame_a.html" in first frame, "frame_b.html" in second frame and "frame_b.html" in third frame.

**Output:-**



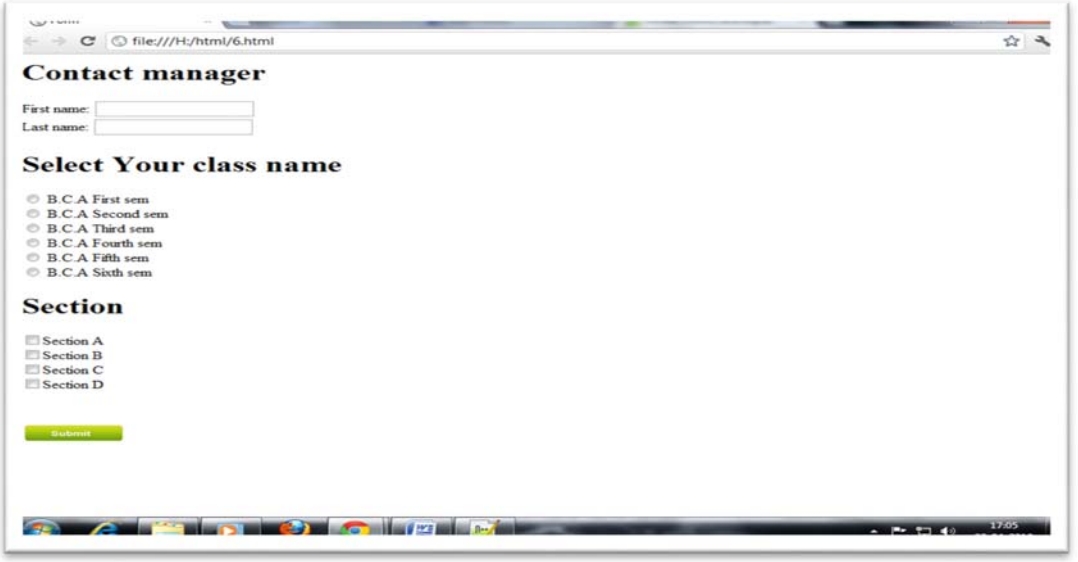

**6. Write a HTML document to create form using radio button, check box, Image button.**

Source code:-

```
<html>

<head>
<title>Form</title>
</head>

<body>

<h1>Contact manager</h1>

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

```
</form>

<h1> Select Your class name </h1>

<form>
<input type="radio" name="class" value="B.C.A First sem"  /> B.C.A First sem  </br>
<input type="radio" name="class" value="B.C.A Second sem" /> B.C.A Second sem </br>
<input type="radio" name="class" value="B.C.A Third sem"  /> B.C.A Third sem  </br>
<input type="radio" name="class" value="B.C.A Fourth sem" /> B.C.A Fourth sem </br>
<input type="radio" name="class" value="B.C.A Fifth sem"  /> B.C.A Fifth sem  </br>
<input type="radio" name="class" value="B.C.A Sixth sem"  /> B.C.A Sixth sem  </br>
</form>

<h1>Section</h1>

<form>
<input type="checkbox" name="Section" value="A" />Section A</br>
<input type="checkbox" name="Section" value="B" />Section B</br>
<input type="checkbox" name="Section" value="C" />Section C</br>
<input type="checkbox" name="Section" value="D" />Section D</br>
</form>

<form>
<input type="image" src="s.jpg" width="100"  height="90" alt="submit">
</form>
</body>
</html>
```

**Output:-**



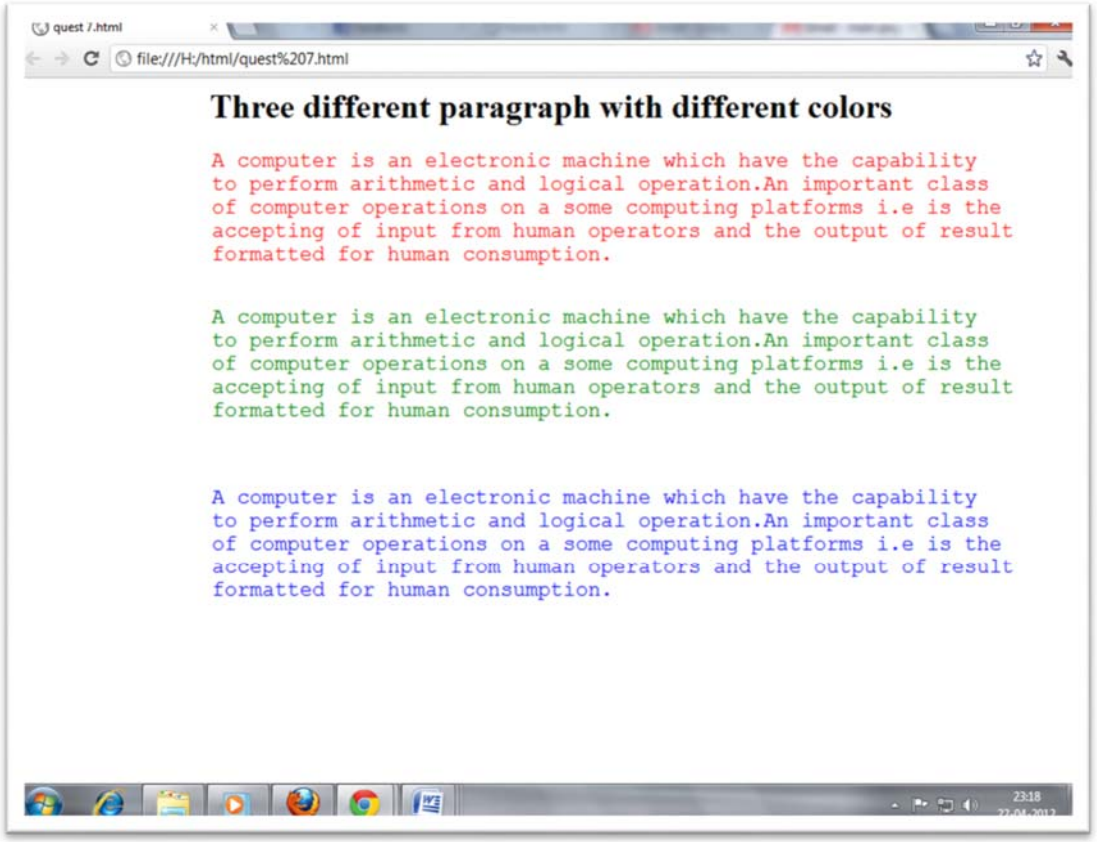

**7. Write a HTML document to create three different paragraph with three different colors?**

Source code

```
<html>
<body>

<h1> <center>Three different paragraph with different colors</center> </h1>

<p>
<font size=”5” color="red">
        <pre>
            A computer is an electronic machine which have the capability
            to perform arithmetic and logical operation.An important class
            of computer operations on a some computing platforms i.e is the
            accepting of input from human operators and the output of result
            formatted for human consumption.
        </pre>
</font>
</p>
```

```
<p>
<font size=”5” color="green">
        <pre>
            A computer is an electronic machine which have the capability
           to perform arithmetic and logical operation.An important class
           of computer operations on a some computing platforms i.e is the
           accepting of input from human operators and the output of result
           formatted for human consumption.
        </pre>
</font>
</p>

<p>
<font size=”5” color="blue">
        <pre>
           A computer is an electronic machine which have the capability
           to perform arithmetic and logical operation.An important class
           of computer operations on a some computing platforms i.e is the
           accepting of input from human operators and the output of result
           formatted for human consumption.
        </pre>
</font>
</p>

</body>
</html>
```

**Output:-**




**8. Write an HTML document to print the following:-**

The family has following facility:-

1. Own house

2400 square feet living

Separate bungalow

Car shed available

2. Car

Maruti Omni car

Registration Number TN 728435

1994 model

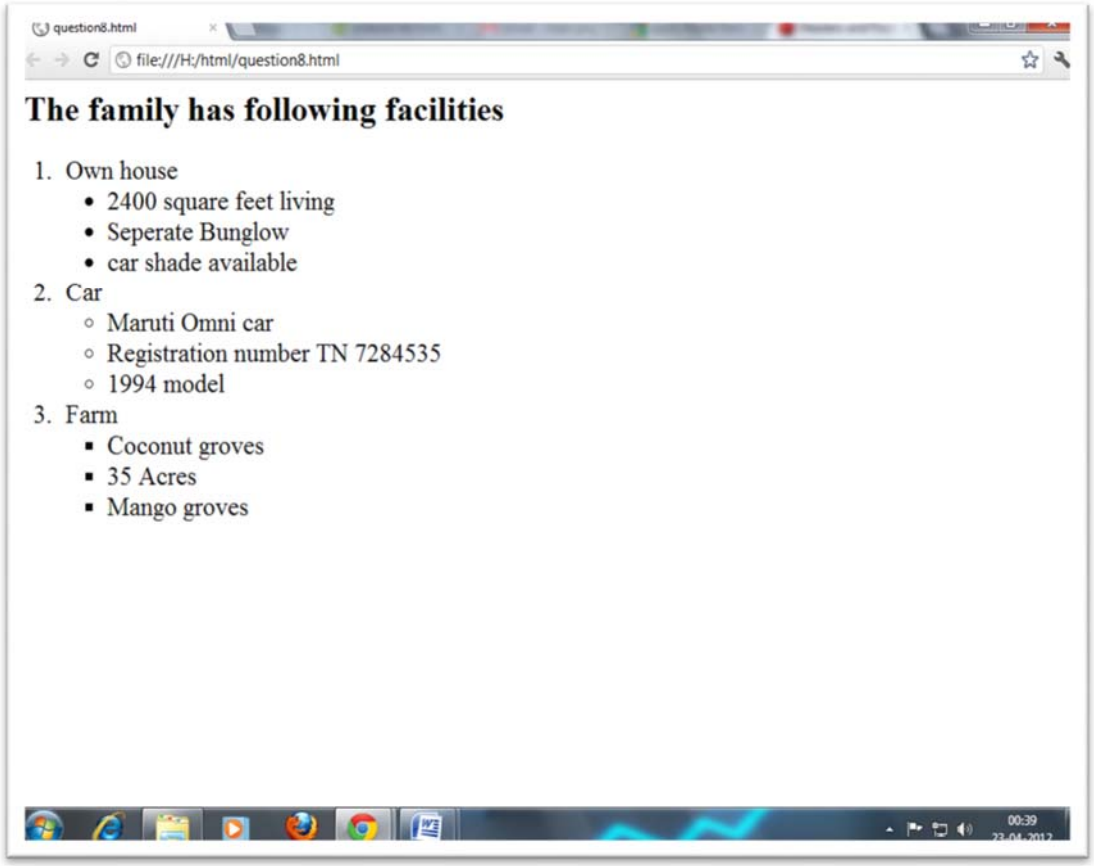
3. Farm

Coconut Groves
35 Acres
Mango Groves

Source code:-

```
<html>
<body>

<h1> The family has following facilities </h1>
<font size="5">
<ol>
<li>Own house</li>
<ul>
<li type=disc> 2400 square feet living</li>
<li type=disc> Seperate Bunglow</li>
<li type=disc> car shade available</li>
</ul>

<li>Car</li>
<ul>
<li type=circle> Maruti Omni car</li>
<li type=circle> Registration number TN 7284535</li>
<li type=circle> 1994 model</li>
</ul>

<li>Farm</li>
<ul>
<li type=square> Coconut groves</li>
<li type=square> 35 Acres</li>
<li type=square> Mango groves</li>
</ul>
</ol>
</font>
</body>
</html>
```

**Output:-**




**9. Create a HTML document for following:-**

Ordered list can be define:-

Keyboard
Mouse
Monitor

Ordered list can have verify of type:-

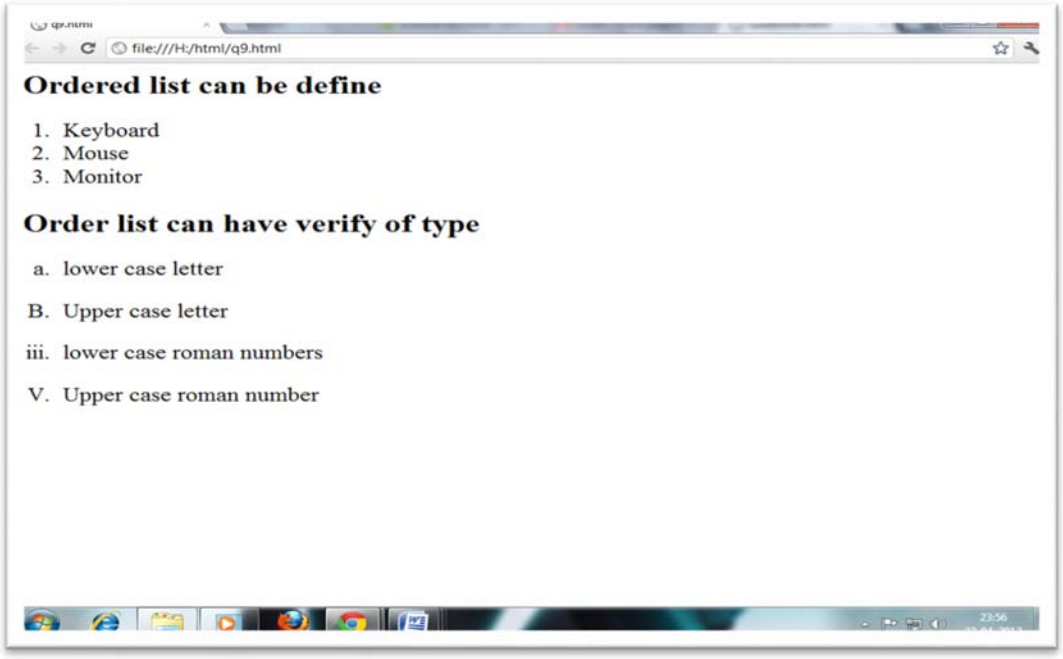
Lower case letter
Upper case letter
Lower case Roman Numbers
Upper case Roman Numbers

Source code:-

```
<html>

<body>

    <h1> Ordered list can be define </h1>
      <font size="5">
        <ol>
           <li>Keyboard</li>
           <li>Mouse</li>
           <li>Monitor</li>
        </ol>
       </font>

   <h1> Order list can have verify of type </h1>
     <font size="5">
        <ol type='a'>
        <li> lower case letter</li>
        </ol>
       </font>

     <font size="5">
        <ol type='A' start='2'>
        <li>Upper case letter</li>
        </ol>
    </font>

   <font size="5">
        <ol type='i' start='3' >
        <li>lower case roman numbers</li>
        </ol>
  </font>
```

```
  <font size="5">
      <ol type='I' start='5' >
      <li>Upper case roman number</li>
      </ol>
  </font>
</body>

</html>
```

Output:-

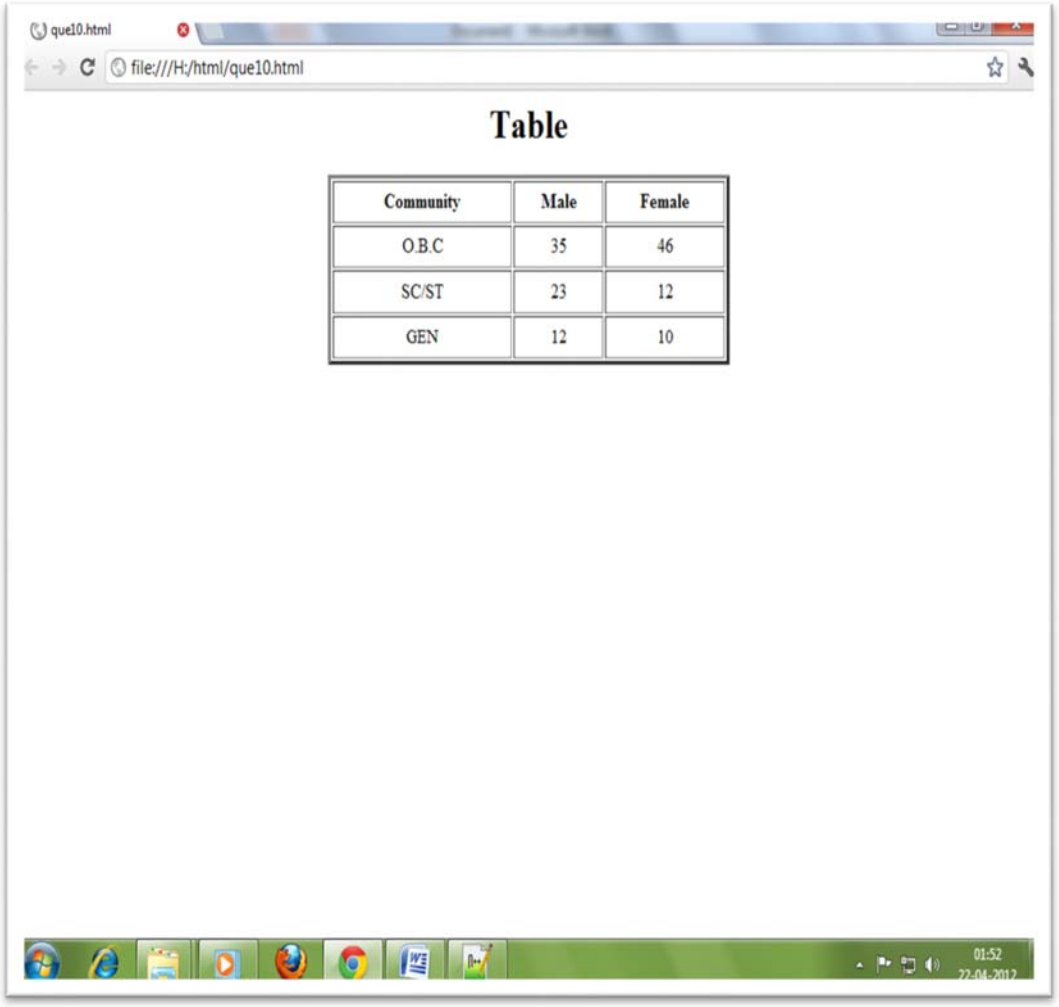



**10. Create a table with following data:-**

| Community | Male | Female |
|---|---|---|
| O.B.C | 35 | 40 |
| SC/ST | 23 | 12 |
| GEN | 12 | 10 |

Source code:-

```
<html>
<body>
<h1><center> Table </center></h1>
<center>
<table border="3" width=400 height=150>
<tr align=center>
<th>Community</th>
<th>Male</th>
<th>Female</th>
</tr>

<tr align=center>
<td>O.B.C</td>
<td>35</td>
<td>46</td>
</tr>
<tr align=center>
<td>SC/ST</td>
<td>23</td>
<td>12</td>
</tr>

<tr align=center>
<td>GEN</td>
<td>12</td>
<td>10</td>
</tr>
</center>
</body>
</html>
```




**11. Make up three image links and put them in a borderless table. Construct the table so that there is just a little space between the images.**

12. **Create form to fin information student.**

13. **Make a simple page and put two text inputs and a Submit button into it. Ask for the user's name and address...**

Top of Form

Name:

Address:

Submit

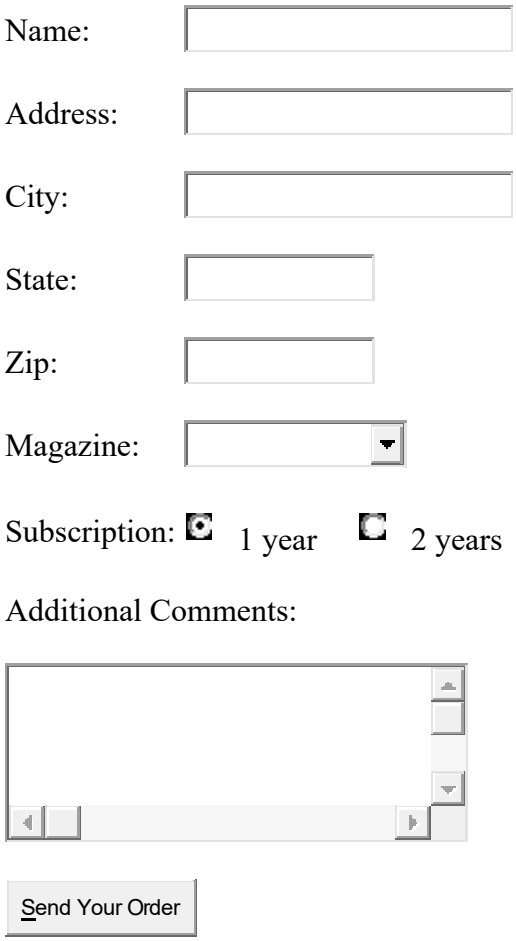
14. **Ask for a few more things… first name, last name, address, city, state, zip. Arrange things neatly in a borderless table so everything lines up and looks nice and neat...**

Top of Form

| | |
|---|---|
| Name: | |
| Address: | |
| City: | |
| State: | |
| Zip: | |

Send your Order

**15 Create following form**




**Instruction for Teacher:**

1) Use the Charts or Multimedia projector as far as possible to explain the content.
2) Assign the homework at the end of every class from the chapter so far covered during that class.
3) Conduct group discussions after finishing one lesson.
4) Do practical work of HTML

**References and Resources**

1) W3schools.com
2) Wikipedia.com
3) Information Technology class x Dhanapat Rai & co . Sumitra Arora.
4) https://developer.mozilla.org/en

# UNIT 3
# Introduction to CSS

## Learning Outcomes:

After Completion of this unit, Student will able to:

- Application and Implementation of CSS.
- Design Simple static pages using CSS.
- Manage CSS file and code in web based project

### LESION 1

### 3.1 Introduction to CSS


CSS stands for Cascading Style Sheets. CSS describes how HTML elements are to be displayed on screen, paper, or in other media. Cascading Style Sheets (CSS) is a language that allows you to define how you want your html document to look. This concerns features such as typeface, background, link colors, margins, and placement of objects on a page. Display instructions for these features are called “style rules” in CSS.

**LOGO of CSS3**

In the early days of HTML, one could define the various parts of a document (such as headings, body text, bold, etc.), but it was up to the browser to interpret what that was supposed to look like. Internet Explorer would display an <h1> heading tag with a particular font, size, color, etc., and Netscape would choose a different combination, and so on. What this meant was that you could make your page look great in one browser, then look at it in another browser and everything would change. Then Netscape and IE started to add special display features to their browsers that developers would use, but those features simply would not work across all browsers. Companies were losing the point about the web being a place where people could view a document across all kinds of computers and browsers without having major differences.

There were certain display features, such as margins, that were not possible to create unless you created strange workarounds with the available tags, such as tables. These workarounds resulted in problems such as longer download times and very messy coding. Then, whenever there was a change to a website, such as your boss saying, "Let's change our website color scheme", this meant editing many, many pages.

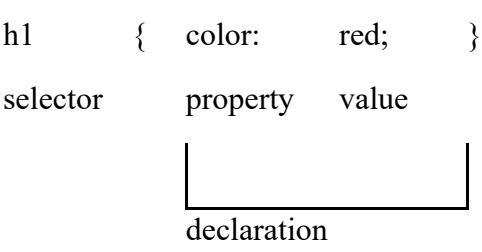
The W3C decided to tackle these problems by developing the CSS standard and urging all browser companies to comply with the standards. For the first few years companies resisted this call to order and there was a problem with browser compatibility with style sheet features. For example a style sheet command might work in Netscape 5.0 but not Netscape 4.0 and not in any version of Internet Explorer. These days this scenario is rapidly changing as companies are complying with the style sheet standards.

Why use them?

- Style sheets make managing the look of multiple web pages MUCH easier by separating CONTENT from DISPLAY information.
- They make web pages faster to download, very important when you are trying to reach people with older computers and modems.
- The W3C is slowly phasing out old ways of coding HTML in its standards, for example the use of the <font> tag. Professional standards are moving toward style sheets.

**CSS Syntax**

A CSS rule-set consists of a selector and a declaration block:




The selector points to the HTML element you want to style. The declaration block contains one or more declarations separated by semicolons. Each declaration includes a CSS property name and a value, separated by a colon. A CSS declaration always

ends with a semicolon, and declaration blocks are surrounded by curly braces. In the following example all <p> elements will be center-aligned, with a red text color:

**Example**

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

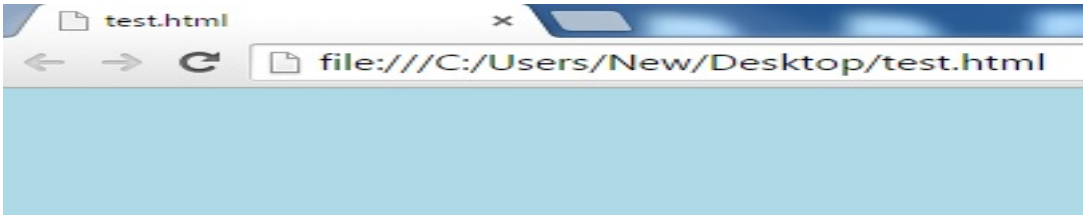
With CSS, a color is most often specified by:
a valid color name - like "red"
a HEX value - like "#ff0000"
an RGB value - like "rgb(255,0,0)"

### 3.1.1 CSS background

The **background-color** property specifies the background color of an element. The background color of a page is set like this:

```
body {
   background-color: lightblue;
}
```

Color name can be given as in the above different format given in the previous section.




### 3.1.2 CSS Boarder

**Border - Shorthand Property**

The border property is a shorthand property for the following individual border properties:

- border-width
- border-style (required)
- border-color

**Border Style**

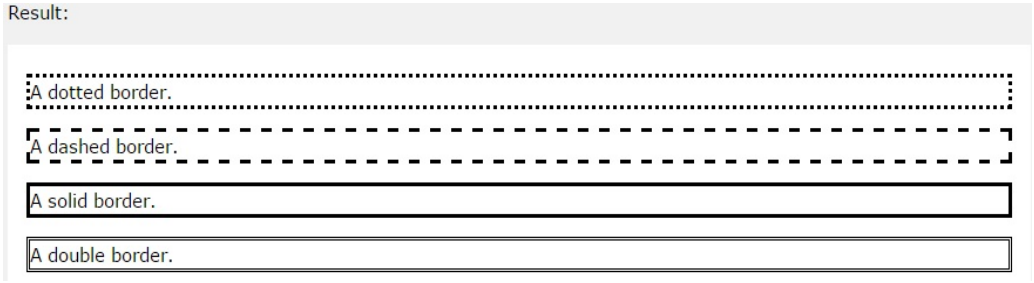
The border-style property specifies what kind of border to display.

The following values are allowed:

- dotted - Defines a dotted border
- dashed - Defines a dashed border
- solid - Defines a solid border
- double - Defines a double border
- groove - Defines a 3D grooved border. The effect depends on the border-color value
- ridge - Defines a 3D ridged border. The effect depends on the border-color value
- inset - Defines a 3D inset border. The effect depends on the border-color value
- outset - Defines a 3D outset border. The effect depends on the border-color value
- none - Defines no border
- hidden - Defines a hidden border

The **border-style** property can have from one to four values (for the top border, right border, bottom border, and the left border).

```
p.dotted {border-style: dotted;}
p.dashed {border-style: dashed;}
```




**Border Width**

The border-width property specifies the width of the four borders. The width can be set as a specific size (in px, pt, cm, em, etc) or by using one of the three pre-defined values: thin, medium, or thick. The border-width property can have from one to four

values (for the top border, right border, bottom border, and the left border).

```
p.one {
   border-style: solid;
   border-width: 5px;
}
```


**Border Color**

The border-color property is used to set the color of the four borders. The color can be set in different format.

The border-color property can have from one to four values (for the top border, right border, bottom border, and the left border). If border-color is not set, it inherits the color of the element.

```
p.one {
   border-style: solid;
   border-color: red;
}
```


**Border - Individual Sides**

It is possible to specify a different border for each side. In CSS, there is also a property for specifying each of the borders (top, right, bottom, and left):

```
p {
   border-top-style: dotted;
   border-right-style: solid;
   border-bottom-style: dotted;
   border-left-style: solid;
}
```

**All CSS Border Properties**

| Property | Description |
|---|---|
| border | Sets all the border properties in one declaration |
| border-bottom | Sets all the bottom border properties in one declaration |
| border-bottom-color | Sets the color of the bottom border |
| border-bottom-style | Sets the style of the bottom border |
| border-bottom-width | Sets the width of the bottom border |
| border-color | Sets the color of the four borders |
| border-left | Sets all the left border properties in one declaration |
| border-left-color | Sets the color of the left border |
| border-left-style | Sets the style of the left border |
| border-left-width | Sets the width of the left border |
| border-right | Sets all the right border properties in one declaration |
| border-right-color | Sets the color of the right border |
| border-right-style | Sets the style of the right border |
| border-right-width | Sets the width of the right border |
| border-style | Sets the style of the four borders |
| border-top | Sets all the top border properties in one declaration |
| border-top-color | Sets the color of the top border |
| border-top-style | Sets the style of the top border |
| border-top-width | Sets the width of the top border |
| border-width | Sets the width of the four borders |

### 3.1.3 CSS Margins

The CSS margin properties are used to generate space around elements. With CSS, you have full control over the margins. There are CSS properties for setting the margin for each side of an element (top, right, bottom, and left).

**Margin - Individual Sides**

CSS has properties for specifying the margin for each side of an element:

- margin-top
- margin-right
- margin-bottom
- margin-left

All the margin properties can have the following values:

- auto - the browser calculates the margin
- length - specifies a margin in px, pt, cm, etc.
- % - specifies a margin in % of the width of the containing element
- inherit - specifies that the margin should be inherited from the parent element

The following example sets different margins for all four sides of a <p> element:

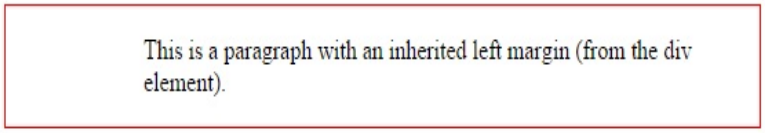
```
p {
  margin-top: 100px;
  margin-bottom: 100px;
  margin-right: 150px;
  margin-left: 80px;
}
```

This paragraph has a top and bottom margin of 100px, a left margin of 80px, and a right margin of 150px.

The following example lets the left margin be inherited from the parent element:

```
div.container {
  border: 1px solid red;
  margin-left: 100px;
}
```

```
p.one {
   margin-left: inherit;
}
```




```
p {
   margin: 100px 150px 100px 80px;
}
```

If the margin property has four values:

margin: 25px 50px 75px 100px;
top margin is 25px
right margin is 50px
bottom margin is 75px
left margin is 100px

If the margin property has three values:

margin: 25px 50px 75px;
top margin is 25px
right and left margins are 50px
bottom margin is 75px

If the margin property has two values:

margin: 25px 50px;
top and bottom margins are 25px
right and left margins are 50px

If the margin property has one value:

margin: 25px;
all four margins are 25px

You can set the margin property to auto to horizontally center the element within its container. The element will then take up the specified width, and the remaining space will be split equally between the left and right margins:

```
div {
   width: 300px;
   margin: auto;
   border: 1px solid red;
}
```

**All CSS Margin Properties**

| Property | Description |
|---|---|
| margin | A shorthand property for setting the margin properties in one declaration |
| margin-bottom | Sets the bottom margin of an element |
| margin-left | Sets the left margin of an element |
| margin-right | Sets the right margin of an element |
| margin-top | Sets the top margin of an element |

### 3.1.4 CSS Padding

The CSS padding properties are used to generate space around content. The padding properties set the size of the white space between the element content and the element border.

The padding clears an area around the content (inside the border) of an element.With CSS, you have full control over the padding. There are CSS properties for setting the padding for each side of an element (top, right, bottom, and left).

Padding - Individual Sides

CSS has properties for specifying the padding for each side of an element:

- padding-top
- padding-right
- padding-bottom
- padding-left

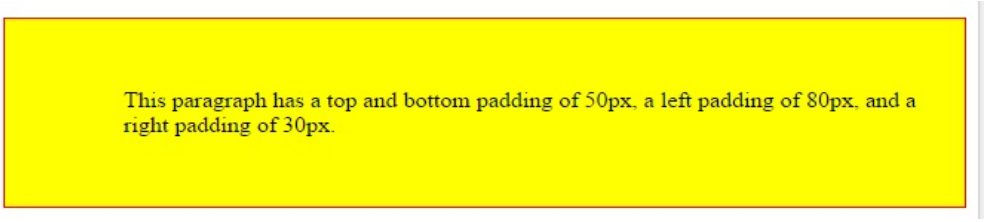
All the padding properties can have the following values:

- length - specifies a padding in px, pt, cm, etc.

% - specifies a padding in % of the width of the containing element inherit - specifies that the padding should be inherited from the parent element The following example

sets different padding for all four sides of a <p> element:

```
p {
  padding-top: 50px;
  padding-right: 30px;
  padding-bottom: 50px;
  padding-left: 80px;
}
```




```
p {
  padding: 50px 30px 50px 80px;
}
```

**All CSS Padding Properties**

| Property | Description |
|---|---|
| padding | A shorthand property for setting all the padding properties in one declaration |
| padding-bottom | Sets the bottom padding of an element |
| padding-left | Sets the left padding of an element |
| padding-right | Sets the right padding of an element |
| padding-top | Sets the top padding of an element |

### 3.1.4 CSS Dimension Properties (Height and Width)

The CSS dimension properties allow you to control the height and width of an element.

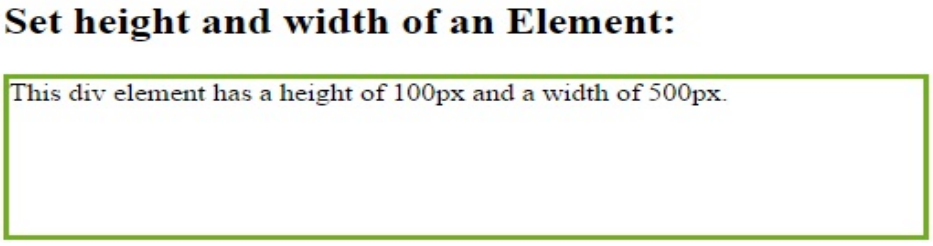
**Setting height and width**

The height and width properties are used to set the height and width of an element. The height and width can be set to auto (this is default. Means that the browser calculates the height and width), or be specified inlength values, like px, cm, etc., or in percent (%) of the containing block.

The height and width properties do not include padding, borders, or margins; they set the height/width of the area inside the padding, border, and margin of the element.

The following example shows a  element with a height of 100 pixels and a width of 500 pixels:

```
div {
  width: 500px;
  height: 100px;
  border: 3px solid #73AD21;
}
```




**Setting max-width**

The max-width property is used to set the maximum width of an element. The max-width can be specified in length values, like px, cm, etc., or in percent (%) of the containing block, or set to none (this is default. Means that there is no maximum width). The problem with the  above occurs when the browser window is smaller than the width of the element (500px). The browser then adds a horizontal scrollbar to the page. Using max-width instead, in this situation, will improve the browser's handling of small windows.

**All CSS Dimension Properties**

| Property | Description |
|---|---|
| height | Sets the height of an element |
| max-height | Sets the maximum height of an element |
| max-width | Sets the maximum width of an element |
| min-height | Sets the minimum height of an element |
| min-width | Sets the minimum width of an element |
| width | Sets the width of an element |

### 3.1.6 Text Formatting

This text is styled with some of the text formatting properties. The heading uses the text-align, text-transform, and color properties. The paragraph is indented, aligned, and the space between characters is specified.

**Text Color**

The color property is used to set the color of the text. If you define the color property, you must also define the background-color property.

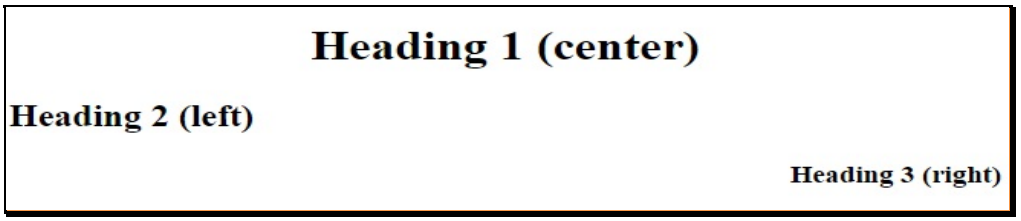
With CSS, a color is most often specified by:

- a color name - like "red"
- a HEX value - like "#ff0000"
- an RGB value - like "rgb(255,0,0)"

```
body {
    color: blue;
}

h1 {
    color: green;
}
```

**Text Alignment**

The text-align property is used to set the horizontal alignment of a text. A text can be left or right aligned, centered, or justified. The following example shows center aligned, and left and right aligned text (left alignment is default if text direction is left-to-right, and right alignment is default if text direction is right-to-left):




```
h1 {
   text-align: center;
}

h2 {
   text-align: left;
}

h3 {
   text-align: right;
}
```

When the text-align property is set to "justify", each line is stretched so that every line has equal width, and the left and right margins are straight (like in magazines and newspapers):

```
div {
   text-align: justify;
}
```

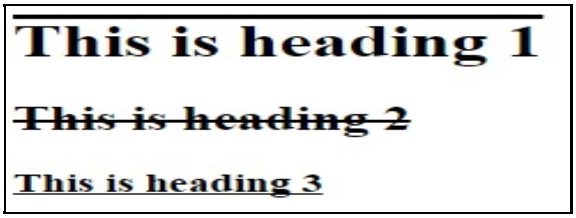
**Text Decoration**

The text-decoration property is used to set or remove decorations from text. The value text-decoration: none; is often used to remove underlines from links:

```
a {
   text-decoration: none;
}
```

The other text-decoration values are used to decorate text:

```
h1 {
   text-decoration: overline;
}

h2 {
   text-decoration: line-through;
}

h3 {
   text-decoration: underline;
}
```




It is not recommended to underline text that is not a link, as this often confuses the reader.

**Text Transformation**

The text-transform property is used to specify uppercase and lowercase letters in a text. It can be used to turn everything into uppercase or lowercase letters, or capitalize the first letter of each word:

```
p.uppercase {
   text-transform: uppercase;
}
```

```
p.lowercase {
   text-transform: lowercase;
}
p.capitalize {
   text-transform: capitalize;
}
```

Text Indentation

The text-indent property is used to specify the indentation of the first line of a text:

```
p {
   text-indent: 50px;
}
```

**Letter Spacing**

The letter-spacing property is used to specify the space between the characters in a text. The following example demonstrates how to increase or decrease the space between characters:

```
h1 {
   letter-spacing: 3px;
}

h2 {
   letter-spacing: -3px;
}
```

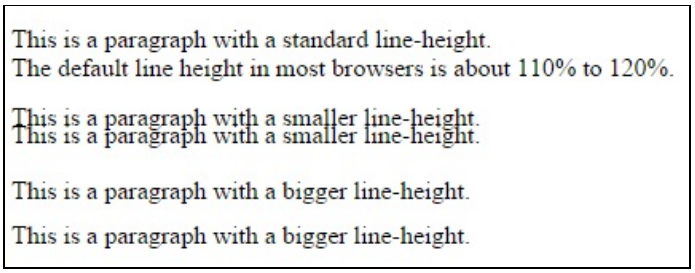
**THIS IS HEADING 1**

**THIS IS HEADING 2**

**Line Height**

The line-height property is used to specify the space between lines:

```
p.small {
   line-height: 0.8;
}
p.big {
   line-height: 1.8;
}
```




**Text Direction**

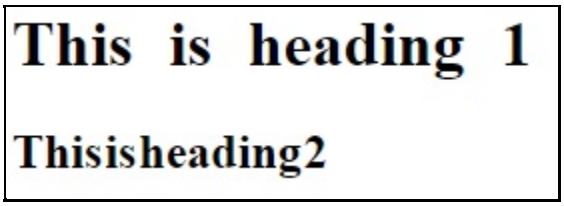
The direction property is used to change the text direction of an element:

```
div {
   direction: rtl;// here rlt means “right to left”
}
```

| Value | Description |
|---|---|
| ltr | The writing direction is left-to-right. This is default |
| rtl | The writing direction is right-to-left |
| initial | Sets this property to its default value. |
| inherit | Inherits this property from its parent element. |

**Word Spacing**

The word-spacing property is used to specify the space between the words in a text. The following example demonstrates how to increase or decrease the space between words:




```
h1 {
  word-spacing: 10px;
}

h2 {
   word-spacing: -5px;
}
```

**All CSS Text Properties**

| Property | Description |
|---|---|
| color | Sets the color of text |
| direction | Specifies the text direction/writing direction |
| letter-spacing | Increases or decreases the space between characters in a text |
| line-height | Sets the line height |
| text-align | Specifies the horizontal alignment of text |
| text-decoration | Specifies the decoration added to text |
| text-indent | Specifies the indentation of the first line in a text-block |
| text-shadow | Specifies the shadow effect added to text |
| text-transform | Controls the capitalization of text |
| unicode-bidi | Used together with the direction property to set or return whether the text should be overridden to support multiple languages in the same document |
| vertical-align | Sets the vertical alignment of an element |
| white-space | Specifies how white-space inside an element is handled |
| word-spacing | Increases or decreases the space between words in a text |

### 3.1.7 CSS Fonts

The CSS font properties define the font family, boldness, size, and the style of a text.

In CSS, there are two types of font family names:

i. generic family - a group of font families with a similar look (like "Serif" or "Monospace")
ii. font family - a specific font family (like "Times New Roman" or "Arial")

| Generic family | Font family | Description |
|---|---|---|
| Serif | Times New Roman<br>Georgia | Serif fonts have small lines at the ends on some characters |
| Sans-serif | Arial<br>Verdana | "Sans" means without - these fonts do not have the lines at the ends of characters |
| Monospace | Courier New<br>Lucida Console | All monospace characters have the same width |

On computer screens, sans-serif fonts are considered easier to read than serif fonts.

**Font Family**

The font family of a text is set with the font-family property. The font-family property should hold several font names as a "fallback" system. If the browser does not support the first font, it tries the next font, and so on.

Start with the font you want, and end with a generic family, to let the browser pick a similar font in the generic family, if no other fonts are available.

Note: If the name of a font family is more than one word, it must be in quotation marks, like: "Times New Roman".

More than one font family is specified in a comma-separated list:

```
p {
  font-family: "Times New Roman", Times, serif;
}
```

**Font Style**

The font-style property is mostly used to specify italic text.

This property has three values:

normal - The text is shown normally
italic - The text is shown in italics
oblique - The text is "leaning" (oblique is very similar to italic, but less supported)

```
p.normal {
   font-style: normal;
}

p.italic {
   font-style: italic;
}

p.oblique {
   font-style: oblique;
}
```

This is a paragraph in normal style.

*This is a paragraph in italic style.*

*This is a paragraph in oblique style.*

**3.1.8 Font Size**

The font-size property sets the size of the text. Being able to manage the text size is important in web design. However, you should not use font size adjustments to make paragraphs look like headings, or headings look like paragraphs.

Always use the proper HTML tags, like <h1> - <h6> for headings and <p> for paragraphs.

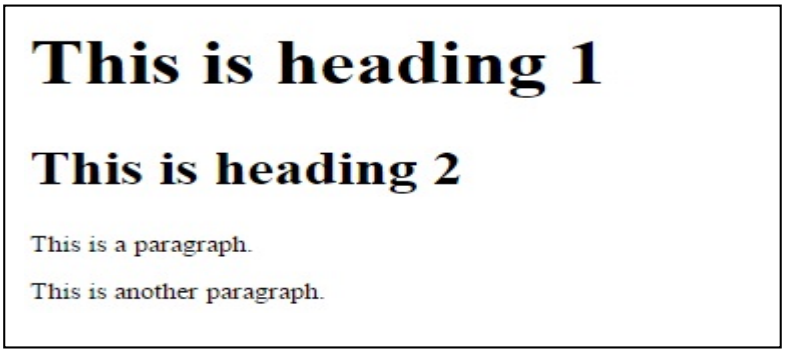
The font-size value can be an absolute, or relative size.
Absolute size:
Sets the text to a specified size
Does not allow a user to change the text size in all browsers (bad for accessibility reasons)
Absolute size is useful when the physical size of the output is known
Relative size:
Sets the size relative to surrounding elements

Allows a user to change the text size in browsers

NOTE: If you do not specify a font size, the default size for normal text, like paragraphs, is 16px (16px=1em).

**Set Font Size With Pixels**

Setting the text size with pixels gives you full control over the text size:




```
h1 {
  font-size: 40px;
}

h2 {
  font-size: 30px;
}

p {
  font-size: 14px;
}
```

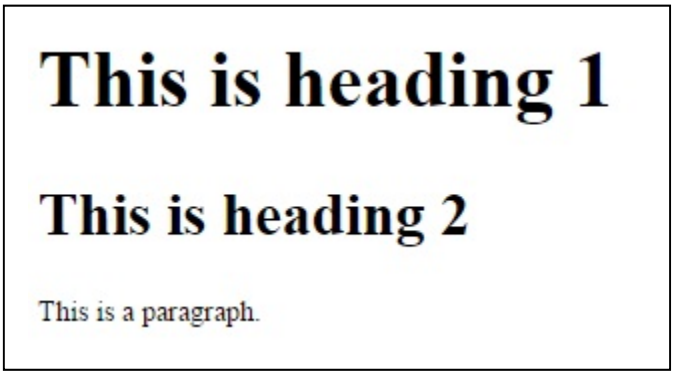
If you use pixels, you can still use the zoom tool to resize the entire page.

**Set Font Size With Em**

To allow users to resize the text (in the browser menu), many developers use em instead of pixels.

The em size unit is recommended by the W3C. 1em is equal to the current font size. The default text size in browsers is 16px. So, the default size of 1em is 16px. The size can be calculated from pixels to em using this formula: pixels/16=em

```
h1 {
   font-size: 2.5em; /* 40px/16=2.5em */
}

h2 {
   font-size: 1.875em; /* 30px/16=1.875em */
}

p {
   font-size: 0.875em; /* 14px/16=0.875em */
}
```



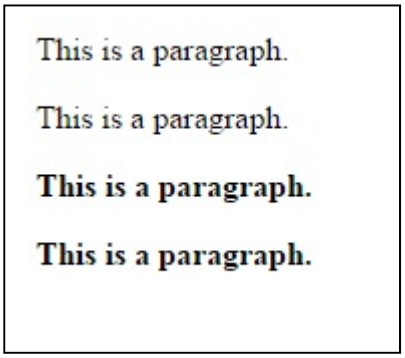

In the example above, the text size in em is the same as the previous example in pixels. However, with the em size, it is possible to adjust the text size in all browsers. Unfortunately, there is still a problem with older versions of IE. The text becomes larger than it should when made larger, and smaller than it should when made smaller.

**Use a Combination of Percent and Em**

The solution that works in all browsers, is to set a default font-size in percent for the <body> element:

```
body {
   font-size: 100%;
}

h1 {
   font-size: 2.5em;
```

```
}

h2 {
   font-size: 1.875em;
}

p {
   font-size: 0.875em;
}
```




Font Weight

The font-weight property specifies the weight of a font:

```
p.normal {
   font-weight: normal;
}

p.thick {
   font-weight: bold;
}
```

**Font Variant**

The font-variant property specifies whether or not a text should be displayed in a small-caps font. In a small-caps font, all lowercase letters are converted to uppercase letters. However, the converted uppercase letters appears in a smaller font size than the original uppercase letters in the text.

```
p.normal {
   font-variant: normal;
}

p.small {
   font-variant: small-caps;
}
```

**All CSS Font Properties**

| Property | Description |
|---|---|
| font | Sets all the font properties in one declaration |
| font-family | Specifies the font family for text |
| font-size | Specifies the font size of text |
| font-style | Specifies the font style for text |
| font-variant | Specifies whether or not a text should be displayed in a small-caps font |
| font-weight | Specifies the weight of a font |

### 3.1.9 CSS Position

The position property specifies the type of positioning method used for an element (static, relative, fixed or absolute).

**The position Property**

The position property specifies the type of positioning method used for an element. There are four different position values:

- static
- relative
- fixed
- absolute

Elements are then positioned using the top, bottom, left, and right properties. However, these properties will not work unless the position property is set first. They also work differently depending on the position value.

**position: static;**

HTML elements are positioned static by default.
Static positioned elements are not affected by the top, bottom, left, and right properties.
An element with position: static; is not positioned in any special way; it is always

positioned according to the normal flow of the page.

```
div.static {
  position: static;
  border: 3px solid #73AD21;
}
```


**position: relative;**

An element with position: relative; is positioned relative to its normal position.

Setting the top, right, bottom, and left properties of a relatively-positioned element will cause it to be adjusted away from its normal position. Other content will not be adjusted to fit into any gap left by the element.

```
div.relative {
  position: relative;
  left: 30px;
  border: 3px solid #73AD21;
}
```


**position: fixed;**

An element with position: fixed; is positioned relative to the viewport, which means it always stays in the same place even if the page is scrolled. The top, right, bottom, and left properties are used to position the element.

A fixed element does not leave a gap in the page where it would normally have been located.

Notice the fixed element in the lower-right corner of the page. Here is the CSS that is used:

```
div.fixed {
  position: fixed;
  bottom: 0;
  right: 0;
```

```
    width: 300px;
    border: 3px solid #73AD21;
}
```

**position: absolute;**

An element with position: absolute; is positioned relative to the nearest positioned ancestor (instead of positioned relative to the viewport, like fixed).

However; if an absolute positioned element has no positioned ancestors, it uses the document body, and moves along with page scrolling.

Note: A "positioned" element is one whose position is anything except static.

```
div.relative {
    position: relative;
    width: 400px;
    height: 200px;
    border: 3px solid #73AD21;
}

div.absolute {
    position: absolute;
    top: 80px;
    right: 0;
    width: 200px;
    height: 100px;
    border: 3px solid #73AD21;
}
```

**Overlapping Elements**

When elements are positioned, they can overlap other elements. The z-index property specifies the stack order of an element (which element should be placed in front of, or behind, the others).

An element can have a positive or negative stack order. Because the image has a z-index of -1, it will be placed behind the text.

```
img {
```

```
    position: absolute;
    left: 0px;
    top: 0px;
    z-index: -1;
}
```

An element with greater stack order is always in front of an element with a lower stack order.

If two positioned elements overlap without a z-index specified, the element positioned last in the HTML code will be shown on top.

All CSS Positioning Properties

| Property | Description |
|---|---|
| bottom | Sets the bottom margin edge for a positioned box |
| clip | Clips an absolutely positioned element |
| cursor | Specifies the type of cursor to be displayed |
| left | Sets the left margin edge for a positioned box |
| overflow | Specifies what happens if content overflows an element's box |
| overflow-x | Specifies what to do with the left/right edges of the content if it overflows the element's content area |
| overflow-y | Specifies what to do with the top/bottom edges of the content if it overflows the element's content area |
| position | Specifies the type of positioning for an element |
| right | Sets the right margin edge for a positioned box |
| top | Sets the top margin edge for a positioned box |
| z-index | Sets the stack order of an element |

### 3.1.10 CSS Float

The float property specifies whether or not an element should float. The clear property is used to control the behavior of floating elements.

In its simplest use, the float property can be used to wrap text around images. The following example specifies that an image should float to the right in a text:

```
img {
  float: right;
  margin: 0 0 10px 10px;
}
```

**The clear Property**

The clear property is used to control the behavior of floating elements. Elements after a floating element will flow around it. To avoid this, use the clear property. The clear property specifies on which sides of an element floating elements are not allowed to float:

```
div {
  clear: left;
}
```

**The clearfix Hack - overflow: auto;**

If an element is taller than the element containing it, and it is floated, it will overflow outside of its container. Then we can add overflow: auto; to the containing element to fix this problem:

```
.clearfix {
  overflow: auto;
}
```

Web Layout Example

It is common to do entire web layouts using the float property:

```
div {
  border: 3px solid blue;
}

.clearfix {
  overflow: auto;
}

nav {
  float: left;
  width: 200px;
```

```
    border: 3px solid #73AD21;
}

section {
    margin-left: 206px;
    border: 3px solid red;
}
```

**All CSS Float Properties**

| Property | Description |
|---|---|
| clear | Specifies on which sides of an element where floating elements are not allowed to float |
| float | Specifies whether or not an element should float |
| overflow | Specifies what happens if content overflows an element's box |
| overflow-x | Specifies what to do with the left/right edges of the content if it overflows the element's content area |
| overflow-y | Specifies what to do with the top/bottom edges of the content if it overflows the element's content area |

### 3.1.11 CSS Image Opacity

Creating transparent images with CSS is easy. The CSS opacity property is a part of the CSS3 recommendation.

**Creating a Transparent Image**

The CSS3 property for transparency is opacity. First we will show you how to create a transparent image with CSS.

```
img {
    opacity: 0.4;
    filter: alpha(opacity=40); /* For IE8 and earlier */
}
```

The opacity property can take a value from 0.0 - 1.0. The lower value, the more transparent. IE8 and earlier use filter:alpha(opacity=x). The x can take a value from 0 - 100. A lower value makes the element more transparent.

```
Image Transparency - Hover Effectimg {
   opacity: 0.4;
   filter: alpha(opacity=40); /* For IE8 and earlier */
}

img:hover {
   opacity: 1.0;
   filter: alpha(opacity=100); /* For IE8 and earlier */
}
```

In addition, we have added what should happen when a user hovers over one of the images. In this case we want the image to NOT be transparent when the user hovers over it. The CSS for this is opacity:1;. When the mouse pointer moves away from the image, the image will be transparent again.

**Text in Transparent Box**

```
<html>
<head>
<style>
div.background {
   background: url(klematis.jpg) repeat;
   border: 2px solid black;
}
div.transbox {
   margin: 30px;
   background-color: #ffffff;
   border: 1px solid black;
   opacity: 0.6;
   filter: alpha(opacity=60); /* For IE8 and earlier */
}
div.transbox p {
   margin: 5%;
```

```
    font-weight: bold;
    color: #000000;
}
</style>
</head>
<body>
<div class="background">
  <div class="transbox">
    <p>This is some text that is placed in the transparent box.</p>
  </div>
</div>
</body>
</html>
```

Positioning Text In an Image

How to position text over an image:

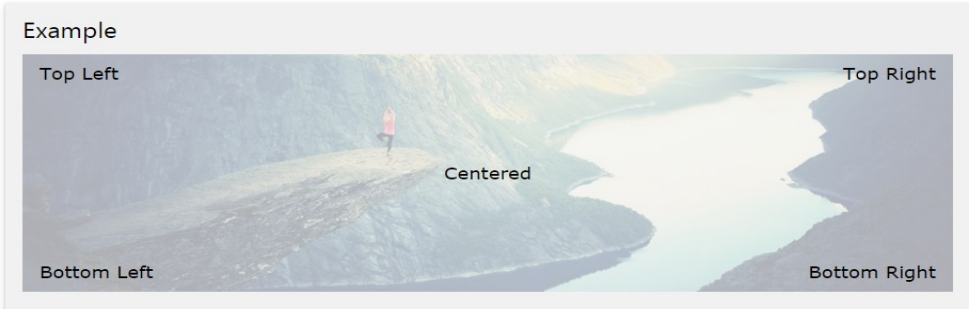



**Image Attributes Selectors**

Style HTML Elements with Specific Attributes

It is possible to style HTML elements that have specific attributes or attribute values.

### 3.1.12 CSS [attribute] Selector

The [attribute] selector is used to select elements with a specified attribute.

The following example selects all <a> elements with a target attribute:

```
a[target] {
  background-color: yellow;
}
```

CSS [attribute="value"] Selector

The [attribute="value"] selector is used to select elements with a specified attribute and value.

The following example selects all <a> elements with a target="_blank" attribute:

```
a[target="_blank"] {
  background-color: yellow;
}
```

CSS [attribute~="value"] Selector

The [attribute~="value"] selector is used to select elements with an attribute value containing a specified word.

The following example selects all elements with a title attribute that contains a space-separated list of words, one of which is "flower":

```
[title~="flower"] {
  border: 5px solid yellow;
}
```

The example above will match elements with title="flower", title="summer flower", and title="flower new", but not title="my-flower" or title="flowers".

**CSS [attribute|="value"] Selector**

The [attribute|="value"] selector is used to select elements with the specified attribute starting with the specified value.

The following example selects all elements with a class attribute value that begins with "top":

Note: The value has to be a whole word, either alone, like class="top", or followed by a hyphen( - ), like class="top-text"!

```
[class|="top"] {
  background: yellow;
}
```

**CSS [attribute^="value"] Selector**

The [attribute^="value"] selector is used to select elements whose attribute value begins with a specified value.

The following example selects all elements with a class attribute value that begins with "top":

```
[class^="top"] {
  background: yellow;
}
```

**CSS [attribute$="value"] Selector**

The [attribute$="value"] selector is used to select elements whose attribute value ends with a specified value.

The following example selects all elements with a class attribute value that ends with "test":

Note: The value does not have to be a whole word!

```
[class$="test"] {
  background: yellow;
}
```

**CSS [attribute*="value"] Selector**

The [attribute*="value"] selector is used to select elements whose attribute value contains a specified value.

The following example selects all elements with a class attribute value that contains "te":

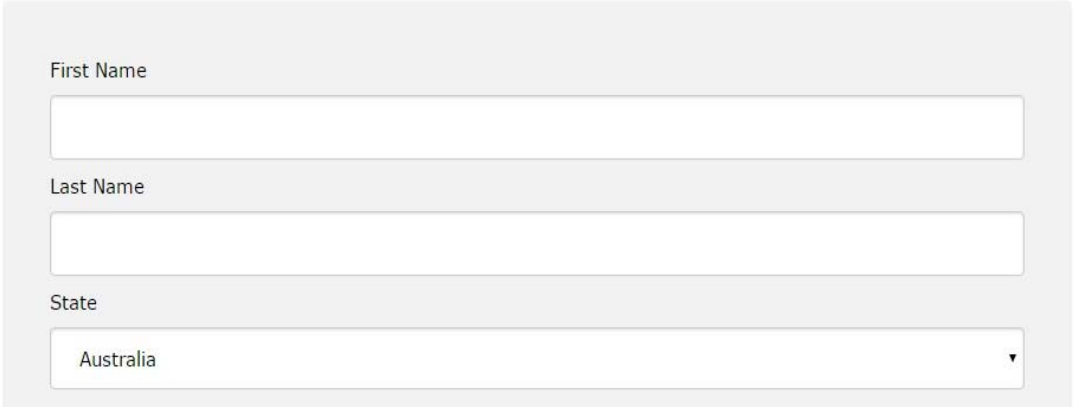
```
[class*="te"] {
  background: yellow;
}
```

**Styling Forms**

The attribute selectors can be useful for styling forms without class or ID:

```
input[type="text"] {
    width: 150px;
    display: block;
    margin-bottom: 10px;
    background-color: yellow;
}

input[type="button"] {
    width: 120px;
    margin-left: 35px;
    display: block;
}
```

**Example of Form**




**Three Ways to Insert CSS**

There are three ways of inserting a style sheet:

i. External style sheet
ii. Internal style sheet
iii. Inline style

## 3.2 External Style Sheet

With an external style sheet, you can change the look of an entire website by changing just one file.

Each page must include a reference to the external style sheet file inside the <link> element. The <link> element goes inside the <head> section:

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

An external style sheet can be written in any text editor. The file should not contain any html tags. The style sheet file must be saved with a .css extension.

Here is how the "myStyle.css" looks:

```
body {
   background-color: lightblue;
}

h1 {
   color: navy;
   margin-left: 20px;
}
```

Do not add a space between the property value and the unit (such as margin-left:20 px;). The correct way is:

```
margin-left:20px;
```

## 3.3 Internal Style Sheet

An internal style sheet may be used if one single page has a unique style. Internal styles are defined within the <style> element, inside the <head> section of an HTML page:

```
<head>
<style>
body {
   background-color: linen;
}
```

```
h1 {
   color: maroon;
   margin-left: 40px;
}
</style>
</head>
```

## 3.4 Inline Styles

An inline style may be used to apply a unique style for a single element. To use inline styles, add the style attribute to the relevant element. The style attribute can contain any CSS property.

The example below shows how to change the color and the left margin of a <h1> element:

```
<h1 style="color:blue;margin-left:30px;">This is a heading.</h1>
```

An inline style loses many of the advantages of a style sheet (by mixing content with presentation). Use this method sparingly.

**Multiple Style Sheets**

If some properties have been defined for the same selector (element) in different style sheets, the value from the last read style sheet will be used.

**Example**

Assume that an external style sheet has the following style for the <h1> element:

```
h1 {
   color: navy;
}
```

then, assume that an internal style sheet also has the following style for the <h1> element:

```
h1 {
   color: orange;
}
```

If the internal style is defined after the link to the external style sheet, the <h1> elements will be "orange":

**Example**

```
<head>
<link rel="stylesheet" type="text/css" href="mystyle.css">
<style>
h1 {
   color: orange;
}
</style>
</head>
```

However, if the internal style is defined before the link to the external style sheet, the <h1> elements will be "navy":

```
<head>
<style>
h1 {
   color: orange;
}
</style>
<link rel="stylesheet" type="text/css" href="mystyle.css">
</head>
```

**Cascading Order**

What style will be used when there is more than one style specified for an HTML element. Generally speaking we can say that all the styles will "cascade" into a new "virtual" style sheet by the following rules, where number one has the highest priority:

1) Inline style (inside an HTML element)
2) External and internal style sheets (in the head section)
3) Browser default

So, an inline style (inside a specific HTML element) has the highest priority, which means that it will override a style defined inside the <head> tag, or in an external style sheet, or a browser default value.

## 3.5 CSS Selectors

CSS selectors are used to "find" (or select) HTML elements based on their element name, id, class, attribute, and more.

### 3.5.1 The element Selector

The element selector selects elements based on the element name. You can select all <p> elements on a page like this (in this case, all <p> elements will be center-aligned, with a red text color):

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

### 3.5.2 The id Selector

The id selector uses the id attribute of an HTML element to select a specific element. The id of an element should be unique within a page, so the id selector is used to select one unique element.

To select an element with a specific id, write a hash (#) character, followed by the id of the element.

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

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

An id name cannot start with a number.

### The class Selector

The class selector selects elements with a specific class attribute. To select elements with a specific class, write a period (.) character, followed by the name of the class.

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

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

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

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

HTML elements can also refer to more than one class.

In the example below, the <p> element will be styled according to class="center" and to class="large":

```
<p class="center large">This paragraph refers to two classes.</p>
```

A class name cannot start with a number.

**Grouping Selectors**

If you have elements with the same style definitions, like this:

```
h1 {
   text-align: center;
   color: red;
}

h2 {
   text-align: center;
   color: red;
}

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

It will be better to group the selectors, to minimize the code. To group selectors, separate each selector with a comma. In the example below we have grouped the selectors from the code above:

```
h1, h2, p {
```

```
    text-align: center;
    color: red;
}
```

**CSS Comments**

Comments are used to explain the code, and may help when you edit the source code at a later date.

Comments are ignored by browsers. A CSS comment starts with /* and ends with */. Comments can also span multiple lines:

```
p {
    color: red;
    /* This is a single-line comment */
    text-align: center;
}

/* This is
a multi-line
comment */
```

## 3.7 Introduction to CSS3

CSS3 is the latest standard for CSS. CSS3 is completely backwards-compatible with earlier versions of CSS. This section teaches you about the new features in CSS3!

## 7.8 CSS3 Modules

CSS3 has been split into "modules". It contains the "old CSS specification" (which has been split into smaller pieces). In addition, new modules are added. Some of the most important CSS3 modules are:

Selectors represent the most used structure, implemented by almost all browsers. Selectors are powerful structures analyzing the hierarchy of a document and make selections according to the relationships between elements. By the time this article was written, the module is at the stage of Candidate Recommendation.

Box Model: This CSS module defines the box model for the elements considering the differences between vertical and horizontal writing.

Backgrounds and Borders module contains specific functions dealing with borders and backgrounds such as shapes, rounding borders and extending background images.

**CSS3 Background Properties**

| Property | Description |
|---|---|
| background | A shorthand property for setting all the background properties in one declaration |
| background-clip | Specifies the painting area of the background |
| background-image | Specifies one or more background images for an element |
| background-origin | Specifies where the background image(s) is/are positioned |
| background-size | Specifies the size of the background image(s) |

Image Values and Replaced Content: "CSS Replaced Content" is a new module dealing with replacing the content of an element with a CSS generated one.

**Text Effects:** The CSS Text module proposes new values and properties to control the text with the purpose of internationalization of how these properties are defined. CSS Color module implements new ways to approach the CSS colors. CSS Fonts module implements new values and properties of CSS fonts, such as the @font-face directive.

The following table lists the new CSS3 text properties:

| Property | Description |
|---|---|
| text-align-last | Specifies how to align the last line of a text |
| text-emphasis | A shorthand for setting text-emphasis-style and text-emphasis-color in one declaration |
| text-justify | Specifies how justified text should be aligned and spaced |
| text-overflow | Specifies how overflowed content that is not displayed should be signaled to the user |
| word-break | Specifies line breaking rules for non-CJK scripts |
| word-wrap | Allows long words to be able to be broken and wrap onto the next line |

**2D Transformations module** applies to CSS several concepts already featured in Scalable Vector Graphic SVG such as scalling, rotation and transformation of the elements.

**3D Transformations Module** adds new specifications for 3D transformations of the elements.

## CSS3 Transform Properties

The following table lists all the 2D transform properties:

| Property | Description |
|---|---|
| transform | Applies a 2D or 3D transformation to an element |
| transform-origin | Allows you to change the position on transformed elements |

### 2D Transform Methods

| Function | Description |
|---|---|
| Matrix (n,n,n,n,n,n) | Defines a 2D transformation, using a matrix of six values |
| translate(x,y) | Defines a 2D translation, moving the element along the X- and the Y-axis |
| translateX(n) | Defines a 2D translation, moving the element along the X-axis |
| translateY(n) | Defines a 2D translation, moving the element along the Y-axis |
| scale(x,y) | Defines a 2D scale transformation, changing the elements width and height |
| scaleX(n) | Defines a 2D scale transformation, changing the element's width |
| scaleY(n) | Defines a 2D scale transformation, changing the element's height |
| rotate(angle) | Defines a 2D rotation, the angle is specified in the parameter |
| skew(x-angle,y-angle) | Defines a 2D skew transformation along the X- and the Y-axis |
| skewX(angle) | Defines a 2D skew transformation along the X-axis |

| | |
|---|---|
| skewY(angle) | Defines a 2D skew transformation along the Y-axis |

The following table lists all the 3D transform properties:

| Property | Description |
|---|---|
| transform | Applies a 2D or 3D transformation to an element |
| transform-origin | Allows you to change the position on transformed elements |
| transform-style | Specifies how nested elements are rendered in 3D space |
| perspective | Specifies the perspective on how 3D elements are viewed |
| perspective-origin | Specifies the bottom position of 3D elements |
| backface-visibility | Defines whether or not an element should be visible when not facing the screen |

CSS Animations module deals with new properties controlling the stages of an animation (i.e. a sequence). Implementation status. CSS Transitions module introduces new concepts among the elements such as transitions and delay in transitions. (i.e. after receiving focus, an element loses it).

The following table lists the @keyframes rule and all the animation properties:

| Property | Description |
|---|---|
| @keyframes | Specifies the animation code |
| animation | A shorthand property for setting all the animation properties |
| animation-delay | Specifies a delay for the start of an animation |
| animation-direction | Specifies whether an animation should play in reverse direction or alternate cycles |
| animation-duration | Specifies how many seconds or milliseconds an animation takes to complete one cycle |
| animation-fill-mode | Specifies a style for the element when the animation is not playing (when it is finished, or when it has a delay) |
| animation-iteration-count | Specifies the number of times an animation should be played |
| animation-name | Specifies the name of the @keyframes animation |
| animation-play-state | Specifies whether the animation is running or paused |

| | |
|---|---|
| animation-timing-function | Specifies the speed curve of the animation |

Multiple Column Layout module, the layout of inline elements is more precisely defined. The Multi-column Layout module deals with new options to manage the multi column layout.

The following table lists all the multi-columns properties:

| Property | Description |
|---|---|
| column-count | Specifies the number of columns an element should be divided into |
| column-fill | Specifies how to fill columns |
| column-gap | Specifies the gap between the columns |
| column-rule | A shorthand property for setting all the column-rule-* properties |
| column-rule-color | Specifies the color of the rule between columns |
| column-rule-style | Specifies the style of the rule between columns |
| column-rule-width | Specifies the width of the rule between columns |
| column-span | Specifies how many columns an element should span across |
| column-width | Specifies a suggested, optimal width for the columns |
| columns | A shorthand property for setting column-width and column-count |

CSS User Interface module was extended with additional methods to assign new styles to a web document and is actually at the status of Candidate Recommendation.

The following table lists all the user interface properties:

| Property | Description |
|---|---|
| box-sizing | Allows you to include the padding and border in an element's total width and height |
| nav-down | Specifies where to navigate when using the arrow-down navigation key |
| nav-index | Specifies the tabbing order for an element |
| nav-left | Specifies where to navigate when using the arrow-left navigation key |
| nav-right | Specifies where to navigate when using the arrow-right navigation key |
| nav-up | Specifies where to navigate when using the arrow-up navigation key |
| outline-offset | Adds space between an outline and the edge or border of an element |
| resize | Specifies whether or not an element is resizable by the user |

**CSS3 Rounded Corner**

With CSS3, you can give any element "rounded corners", by using the border-radius property.

**Here are three examples:**

1. Rounded corners for an element with a specified background color:
2. Rounded corners for an element with a border:
3. Rounded corners for an element with a background image:

```
#rcorners1 {
    border-radius: 25px;
    background: #73AD21;
    padding: 20px;
    width: 200px;
    height: 150px;
}
```

```
#rcorners2 {
    border-radius: 25px;
    border: 2px solid #73AD21;
    padding: 20px;
    width: 200px;
    height: 150px;
}

#rcorners3 {
    border-radius: 25px;
    background: url(paper.gif);
    background-position: left top;
    background-repeat: repeat;
    padding: 20px;
    width: 200px;
    height: 150px;
}
```

The border-radius property is actually a shorthand property for the border-top-left-radius, border-top-right-radius, border-bottom-right-radius and border-bottom-left-radius properties.

**CSS3 border-radius - Specify Each Corner**

If you specify only one value for the border-radius property, this radius will be applied to all 4 corners.

However, you can specify each corner separately if you wish. Here are the rules:

1. Four values: first value applies to top-left, second value applies to top-right, third value applies to bottom-right, and fourth value applies to bottom-left corner
2. Three values: first value applies to top-left, second value applies to top-right and bottom-left, and third value applies to bottom-right
3. Two values: first value applies to top-left and bottom-right corner, and the second value applies to top-right and bottom-left corner
4. One value: all four corners are rounded equally

**Here are three examples:**

1. Four values - border-radius: 15px 50px 30px 5px:
2. Three values - border-radius: 15px 50px 30px:
3. Two values - border-radius: 15px 50px:

Example

```
#rcorners4 {
  border-radius: 15px 50px 30px 5px;
  background: #73AD21;
  padding: 20px;
  width: 200px;
  height: 150px;
}

#rcorners5 {
  border-radius: 15px 50px 30px;
  background: #73AD21;
  padding: 20px;
  width: 200px;
  height: 150px;
}

#rcorners6 {
  border-radius: 15px 50px;
  background: #73AD21;
  padding: 20px;
  width: 200px;
  height: 150px;
}
```

You could also create elliptical corners:

```
Example
#rcorners7 {
  border-radius: 50px/15px;
  background: #73AD21;
```

```
    padding: 20px;
    width: 200px;
    height: 150px;
}

#rcorners8 {
    border-radius: 15px/50px;
    background: #73AD21;
    padding: 20px;
    width: 200px;
    height: 150px;
}

#rcorners9 {
    border-radius: 50%;
    background: #73AD21;
    padding: 20px;
    width: 200px;
    height: 150px;
}
```

**CSS3 Rounded Corners Properties**

| Property | Description |
|---|---|
| border-radius | A shorthand property for setting all the four border-*-*-radius properties |
| Border-top-left-radius | Defines the shape of the border of the top-left corner |
| Border-top-right-radius | Defines the shape of the border of the top-right corner |
| Border-bottom-right-radius | Defines the shape of the border of the bottom-right corner |
| border-bottom-left-radius | Defines the shape of the border of the bottom-left corner |

**CSS3 Gradients**

CSS3 gradients let you display smooth transitions between two or more specified colors. Earlier, you had to use images for these effects. However, by using CSS3 gradients you can reduce download time and bandwidth usage. In addition, elements with gradients look better when zoomed, because the gradient is generated by the browser.

**CSS3 defines two types of gradients:**

- Linear Gradients (goes down/up/left/right/diagonally)
- Radial Gradients (defined by their center)

**CSS3 Linear Gradients**

To create a linear gradient you must define at least two color stops. Color stops are the colors you want to render smooth transitions among. You can also set a starting point and a direction (or an angle) along with the gradient effect.

**Syntax**

background: linear-gradient(direction, color-stop1, color-stop2, ...);

Linear Gradient - Top to Bottom (this is default)

The following example shows a linear gradient that starts at the top. It starts red, transitioning to yellow

```
#grad {
   background: red; /* For browsers that do not support gradients */
   background: -webkit-linear-gradient(red, yellow); /* For Safari 5.1 to 6.0 */
   background: -o-linear-gradient(red, yellow); /* For Opera 11.1 to 12.0 */
   background: -moz-linear-gradient(red, yellow); /* For Firefox 3.6 to 15 */
   background: linear-gradient(red, yellow); /* Standard syntax */
}
```

**Linear Gradient - Left to Right**

The following example shows a linear gradient that starts from the left. It starts red, transitioning to yellow:

```
#grad {
  background: red; /* For browsers that do not support gradients */
  background: -webkit-linear-gradient(left, red , yellow); /* For Safari 5.1 to 6.0 */
  background: -o-linear-gradient(right, red, yellow); /* For Opera 11.1 to 12.0 */
  background: -moz-linear-gradient(right, red, yellow); /* For Firefox 3.6 to 15 */
  background: linear-gradient(to right, red , yellow); /* Standard syntax */
}
```

**Linear Gradient - Diagonal**

You can make a gradient diagonally by specifying both the horizontal and vertical starting positions.

The following example shows a linear gradient that starts at top left (and goes to bottom right). It starts red, transitioning to yellow:

```
#grad {
  background: red; /* For browsers that do not support gradients */
  background: -webkit-linear-gradient(left top, red, yellow); /* For Safari 5.1 to 6.0 */
  background: -o-linear-gradient(bottom right, red, yellow); /* For Opera 11.1 to 12.0 */
  background: -moz-linear-gradient(bottom right, red, yellow); /* For Firefox 3.6 to 15 */
  background: linear-gradient(to bottom right, red, yellow); /* Standard syntax */
}
```

**Using Angles**

If you want more control over the direction of the gradient, you can define an angle, instead of the predefined directions (to bottom, to top, to right, to left, to bottom right, etc.).

**Syntax**

background: linear-gradient(angle, color-stop1, color-stop2);
The angle is specified as an angle between a horizontal line and the gradient line.
The following example shows how to use angles on linear gradients:

```
#grad {
   background: red; /* For browsers that do not support gradients */
  background: -webkit-linear-gradient(-90deg, red, yellow); /* For Safari 5.1 to 6.0 */
  background: -o-linear-gradient(-90deg, red, yellow); /* For Opera 11.1 to 12.0 */
  background: -moz-linear-gradient(-90deg, red, yellow); /* For Firefox 3.6 to 15 */
   background: linear-gradient(-90deg, red, yellow); /* Standard syntax */
}
```

**Using Multiple Color Stops**

The following example shows a linear gradient (from top to bottom) with multiple color stops:

```
#grad {
   background: red; /* For browsers that do not support gradients */
  background: -webkit-linear-gradient(red, yellow, green); /* For Safari 5.1 to 6.0 */
  background: -o-linear-gradient(red, yellow, green); /* For Opera 11.1 to 12.0 */
  background: -moz-linear-gradient(red, yellow, green); /* For Firefox 3.6 to 15 */
   background: linear-gradient(red, yellow, green); /* Standard syntax */
}
```

The following example shows how to create a linear gradient (from left to right) with the color of the rainbow and some text:


**Gradient Background**

```
#grad {
        background: red; /* For browsers that do not support gradients */
         /* For Safari 5.1 to 6.0 */
         background: -webkit-linear-
       gradient(left,red,orange,yellow,green,blue,indigo,violet);
```

```
  /* For Opera 11.1 to 12.0 */
  background: -o-linear-
gradient(left,red,orange,yellow,green,blue,indigo,violet);
  /* For Fx 3.6 to 15 */
  background: -moz-linear-
gradient(left,red,orange,yellow,green,blue,indigo,violet);
  /* Standard syntax */
  background: linear-gradient(to right,
red,orange,yellow,green,blue,indigo,violet);
}
```

**Using Transparency**

CSS3 gradients also support transparency, which can be used to create fading effects.

To add transparency, we use the rgba() function to define the color stops. The last parameter in the rgba() function can be a value from 0 to 1, and it defines the transparency of the color: 0 indicates full transparency, 1 indicates full color (no transparency).

The following example shows a linear gradient that starts from the left. It starts fully transparent, transitioning to full color red:

```
#grad {
  background: red; /* For browsers that do not support gradients */
  background: -webkit-linear-
gradient(left,rgba(255,0,0,0),rgba(255,0,0,1)); /*Safari 5.1-6*/
  background: -o-linear-
gradient(right,rgba(255,0,0,0),rgba(255,0,0,1)); /*Opera 11.1-12*/
  background: -moz-linear-
gradient(right,rgba(255,0,0,0),rgba(255,0,0,1)); /*Fx 3.6-15*/
  background: linear-gradient(to right, rgba(255,0,0,0),
rgba(255,0,0,1)); /*Standard*/
}
```

**Repeating a linear-gradient**

The repeating-linear-gradient() function is used to repeat linear gradients:

```
#grad {
  background: red; /* For browsers that do not support gradients */
  /* Safari 5.1 to 6.0 */
  background: -webkit-repeating-linear-gradient(red, yellow 10%, green
```

```
20%);
  /* Opera 11.1 to 12.0 */
  background: -o-repeating-linear-gradient(red, yellow 10%, green 20%);
  /* Firefox 3.6 to 15 */
  background: -moz-repeating-linear-gradient(red, yellow 10%, green 20%);
  /* Standard syntax */
  background: repeating-linear-gradient(red, yellow 10%, green 20%);
}
```

The repeating-linear-gradient() function is used to repeat linear gradients:

**CSS3 Radial Gradients**

A radial gradient is defined by its center. To create a radial gradient you must also define at least two color stops.

**Syntax**

background: radial-gradient(shape size at position, start-color, ... last-color);

By default, shape is ellipse, size is farthest-corner, and position is center. Radial Gradient - Evenly Spaced Color Stops (this is default) The following example shows a radial gradient with evenly spaced color stops:

```
#grad {
  background: red; /* For browsers that do not support gradients */
  background: -webkit-radial-gradient(red, yellow, green); /* Safari 5.1 to 6.0
*/
  background: -o-radial-gradient(red, yellow, green); /* For Opera 11.6 to 12.0
*/
  background: -moz-radial-gradient(red, yellow, green); /* For Firefox 3.6 to
15 */
  background: radial-gradient(red, yellow, green); /* Standard syntax */
}
```

**Radial Gradient - Differently Spaced Color Stops**

```
#grad {
  background: red; /* For browsers that do not support gradients */
  background: -webkit-radial-gradient(red 5%, yellow 15%, green 60%); /*
Safari 5.1-6.0 */
  background: -o-radial-gradient(red 5%, yellow 15%, green 60%); /* For
Opera 11.6-12.0 */
  background: -moz-radial-gradient(red 5%, yellow 15%, green 60%); /* For
Firefox 3.6-15 */
  background: radial-gradient(red 5%, yellow 15%, green 60%); /* Standard
syntax */
}
```

**Set Shape**

The shape parameter defines the shape. It can take the value circle or ellipse. The default value is ellipse.

The following example shows a radial gradient with the shape of a circle:

```
#grad {
  background: red; /* For browsers that do not support gradients */
  background: -webkit-radial-gradient(circle, red, yellow, green); /* Safari */
  background: -o-radial-gradient(circle, red, yellow, green); /* Opera 11.6 to
12.0 */
  background: -moz-radial-gradient(circle, red, yellow, green); /* Firefox 3.6
to 15 */
  background: radial-gradient(circle, red, yellow, green); /* Standard syntax
*/
}
```

**Use of Different Size Keywords**

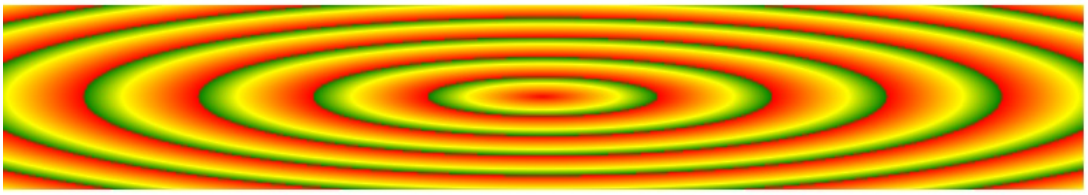
The size parameter defines the size of the gradient. It can take four values:
closest-side
farthest-side
closest-corner

farthest-corner

**A radial gradient with different size keywords:**

```
#grad1 {
  background: red; /* For browsers that do not support gradients */
  /* Safari 5.1 to 6.0 */
  background: -webkit-radial-gradient(60% 55%, closest-side, red, yellow,
black);
  /* For Opera 11.6 to 12.0 */
  background: -o-radial-gradient(60% 55%, closest-side, red, yellow, black);
  /* For Firefox 3.6 to 15 */
  background: -moz-radial-gradient(60% 55%, closest-side, red, yellow,
black);
  /* Standard syntax */
  background: radial-gradient(closest-side at 60% 55%, red, yellow, black);
}

#grad2 {
  /* Safari 5.1 to 6.0 */
  background: -webkit-radial-gradient(60% 55%, farthest-side, red, yellow,
black);
  /* Opera 11.6 to 12.0 */
  background: -o-radial-gradient(60% 55%, farthest-side, red, yellow, black);
  /* For Firefox 3.6 to 15 */
  background: -moz-radial-gradient(60% 55%, farthest-side, red, yellow,
black);
  /* Standard syntax */
  background: radial-gradient(farthest-side at 60% 55%, red, yellow, black);
}
```

The repeating-radial-gradient() function is used to repeat radial gradients:

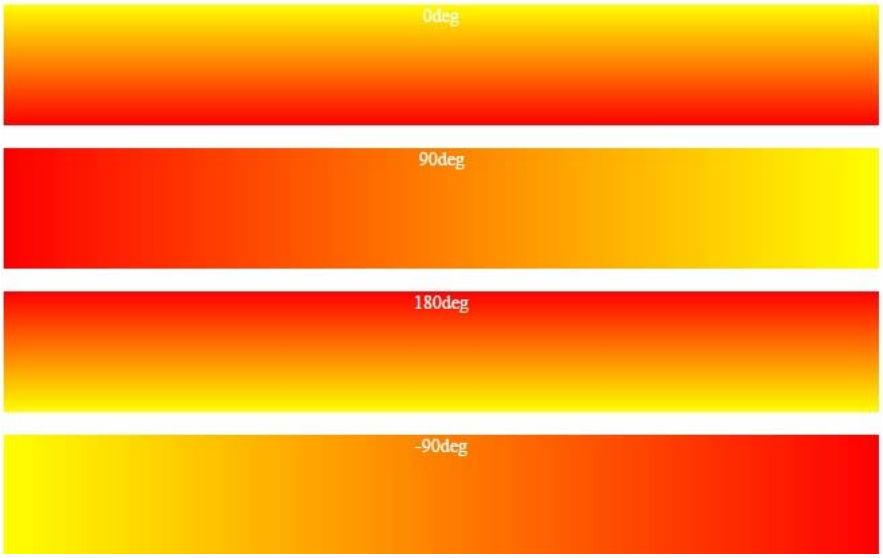


**Repeating a radial-gradient**

The repeating-radial-gradient() function is used to repeat radial gradients:

A repeating radial gradient:

```
#grad {
    background: red; /* For browsers that do not support gradients */
    /* For Safari 5.1 to 6.0 */
    background: -webkit-repeating-radial-gradient(red, yellow 10%, green
  15%);
    /* For Opera 11.6 to 12.0 */
    background: -o-repeating-radial-gradient(red, yellow 10%, green 15%);
    /* For Firefox 3.6 to 15 */
    background: -moz-repeating-radial-gradient(red, yellow 10%, green 15%);
    /* Standard syntax */
    background: repeating-radial-gradient(red, yellow 10%, green 15%);
  }
```




**CSS3 Shadow**

**C3 Shadow Effects**

With CSS3 you can add shadow to text and to elements. In this chapter you will learn about the following properties:

text-shadow

box-shadow

**CSS3 Text Shadow**

The CSS3 text-shadow property applies shadow to text. In its simplest use, you only specify the horizontal shadow (2px) and the vertical shadow (2px):

**Example**

```
h1 {
   text-shadow: 2px 2px;
}
```


Next, add a color to the shadow:

**Example**

```
h1 {
   text-shadow: 2px 2px red;
}
```


Then, add a blur effect to the shadow:

**Example**

```
h1 {
   text-shadow: 2px 2px 5px red;
}
```


The following example shows a white text with black shadow:

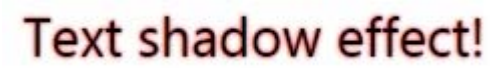
```
Example
h1 {
```

```
    color: white;
    text-shadow: 2px 2px 4px #000000;
}
```


The following example shows a red neon glow shadow:

**Example**

```
h1 {
    text-shadow: 0 0 3px #FF0000;
}
```



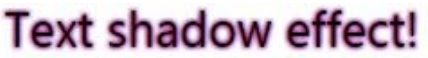

**Multiple Shadows**

To add more than one shadow to the text, you can add a comma-separated list of shadows.

The following example shows a red and blue neon glow shadow:

Example

```
h1 {
    text-shadow: 0 0 3px #FF0000, 0 0 5px #0000FF;
}
```




The following example shows a white text with black, blue, and darkblue shadow:

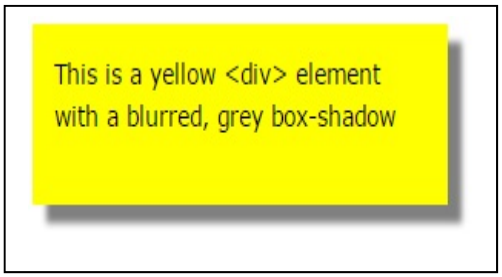
**Example**

```
h1 {
    color: white;
    text-shadow: 1px 1px 2px black, 0 0 25px blue, 0 0 5px darkblue;
}
```


**CSS3 box-shadow Property**

The CSS3 box-shadow property applies shadow to elements. In its simplest use, you only specify the horizontal shadow and the vertical shadow: This is a yellow  element with a black box-shadow Example

```
div {
   box-shadow: 10px 10px;
}
```




Next, add a color to the shadow:
This is a yellow  element with a grey box-shadow

```
div {
   box-shadow: 10px 10px grey;
}
```

Next, add a blur effect to the shadow:
This is a yellow  element with a blurred, grey box-shadow

```
div {
   box-shadow: 10px 10px 5px grey;
}
```

You can also add shadows to the ::before and ::after pseudo-classes, to create an interesting effect:

```
#boxshadow {
   position: relative;
   box-shadow: 1px 2px 4px rgba(0, 0, 0, .5);
   padding: 10px;
   background: white;
}
#boxshadow img {
   width: 100%;
   border: 1px solid #8a4419;
   border-style: inset;
}
#boxshadow::after {
   content: '';
```

```
    position: absolute;
    z-index: -1; /* hide shadow behind image */
    box-shadow: 0 15px 20px rgba(0, 0, 0, 0.3);
    width: 70%;
    left: 15%; /* one half of the remaining 30% */
    height: 100px;
    bottom: 0;
  }
```

**Cards**

An example of using the box-shadow property to create paper-like cards:

```
Examplediv.card {
    width: 250px;
    box-shadow: 0 4px 8px 0 rgba(0, 0, 0, 0.2), 0 6px 20px 0 rgba(0, 0, 0,
  0.19);
    text-align: center;
}
```

**CSS3 Shadow Properties**

The following table lists the CSS3 shadow properties:

| Property | Description |
|---|---|
| box-shadow | Adds one or more shadows to an element |
| text-shadow | Adds one or more shadows to a text |

**Terminologies**

CSS Cascading Style Sheet

Instruction for teachers

1. Give the practical examples found at their home and surrounding.
2. Use the Charts as far as possible to explain the content.
3. Assign the homework at the end of every class from the chapter so far covered during that class.
4. Conduct group discussions after finishing one lesson.

**References:**

http://www.w3schools.com/

PRACTICAL

TITLE

Implementation and Application of Cascading Style Sheet

OBJECTIVE

To learn about the use and implementation of CSS in real field.

LAB TASKS

- Cascading style sheet:
- Inline css
- Internal css
- External css
- Css background
- Css margin
- Css padding
- Css heights/widths
- Css text
- Css fonts
- Css max widths
- Css float
- Css position
- Css3 rounded corner
- Css3 boarder images
- Css3 backgrounds
- Css3 colors
- Css3 gradients
- Css3 shadow
- Css3 text
- Css3 buttons
- Css3 2D transforms
- Css3 3D transforms

# UNIT 4

# Javascript

## Learning Outcomes

- Introduced with Javascript programming.
- Use of Operators, conditional statement and Looping in Javascript
- Function and OOP concept
- Familiar with Document Object,window object, Status bar, Dialog Boxes and Date Object.
- Use of Events

### LESSON 1

**Introduction**

Javascript is an easy-to-use programming language that can be embedded in the header of your web pages. It can enhance the dynamics and interactive features of your page by allowing you to perform calculations, check forms, write interactive games, add special effects, customize graphics selections, create security passwords and more.

What is JavaScript ?

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

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

The ECMA-262 Specification defined a standard version of the core JavaScript language.

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

**Client-side JavaScript**

Client-side JavaScript is the most common form of the language. The script should be included in or referenced by an HTML document for the code to be interpreted by the browser.

It means that a web page need not be a static HTML, but can include programs that interact with the user, control the browser, and dynamically create HTML content.

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

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

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

**Advantages of JavaScript**

The merits of using JavaScript are −

- **Less server interaction** − You can validate user input before sending the page off to the server. This saves server traffic, which means less load on your server.
- **Immediate feedback to the visitors** − They don't have to wait for a page reload to see if they have forgotten to enter something.
- **Increased interactivity** − You can create interfaces that react when the user hovers over them with a mouse or activates them via the keyboard.

- **Richer interfaces** − You can use JavaScript to include such items as drag-and-drop components and sliders to give a Rich Interface to your site visitors.

**Limitations of JavaScript**

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

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

Once again, JavaScript is a lightweight, interpreted programming language that allows you to build interactivity into otherwise static HTML pages.

**JavaScript Development Tools**

One of major strengths of JavaScript is that it does not require expensive development tools. You can start with a simple text editor such as Notepad. Since it is an interpreted language inside the context of a web browser, you don't even need to buy a compiler.

To make our life simpler, various vendors have come up with very nice JavaScript editing tools. Some of them are listed here −

- **Microsoft FrontPage** − Microsoft has developed a popular HTML editor called FrontPage. FrontPage also provides web developers with a number of JavaScript tools to assist in the creation of interactive websites.
- **Macromedia Dreamweaver MX** − Macromedia Dreamweaver MX is a very popular HTML and JavaScript editor in the professional web development crowd. It provides several handy prebuilt JavaScript components, integrates well with databases, and conforms to new standards such as XHTML and XML.
- **Macromedia HomeSite 5** − HomeSite 5 is a well-liked HTML and JavaScript editor from Macromedia that can be used to manage personal websites effectively.

**Where is JavaScript Today ?**

The ECMAScript Edition 5 standard will be the first update to be released in over four years. JavaScript 2.0 conforms to Edition 5 of the ECMAScript standard, and the difference between the two is extremely minor.

The specification for JavaScript 2.0 can be found on the following site: http://www.ecmascript.org/

Today, Netscape's JavaScript and Microsoft's JScript conform to the ECMAScript standard, although both the languages still support the features that are not a part of the standard.

**Difference between JavaScript and Java?**

Actually, the 2 languages have almost nothing in common except for the name. Although Java is technically an interpreted programming language, it is coded in a similar fashion to C++, with separate header and class files, compiled together prior to execution. It is powerful enough to write major applications and insert them in a web page as a special object called an "applet." Java has been generating a lot of excitment because of its unique ability to run the same program on IBM, Mac, and Unix computers. Java is not considered an easy-to-use language for non-programmers.

Javascript is much simpler to use than Java. With Javascript, if I want check a form for errors, I just type an if-then statement at the top of my page. No compiling, no applets, just a simple sequence.

**Object Oriented Programming**

OOP is a programming technique (note: not a language structure - you don't even need an object-oriented language to program in an object-oriented fashion) designed to simplify complicated programming concepts. In essence, object-oriented programming revolves around the idea of user- and system-defined chunks of data, and controlled means of accessing and modifying those chunks.

Object-oriented programming consists of Objects, Methods and Properties. An object is basically a black box which stores some information. It may have a way for you to read that information and a way for you to write to, or change, that information. It may also have other less obvious ways of interacting with the

information.

Some of the information in the object may actually be directly accessible; other information may require you to use a method to access it - perhaps because the way the information is stored internally is of no use to you, or because only certain things can be written into that information space and the object needs to check that you're not going outside those limits.

The directly accessible bits of information in the object are its properties. The difference between data accessed via properties and data accessed via methods is that with properties, you see exactly what you're doing to the object; with methods, unless you created the object yourself, you just see the effects of what you're doing.

Other Javascript pages you read will probably refer frequently to objects, events, methods, and properties. This tutorial will teach by example, without focusing too heavily on OOP vocabulary. However, you will need a basic understanding of these terms to use other JavaScript references.

**Objects and Properties**

Your web page document is an object. Any table, form, button, image, or link on your page is also an object. Each object has certain properties (information about the object). For example, the background color of your document is written document.bgcolor. You would change the color of your page to red by writing the line: document.bgcolor="red"

The contents (or value) of a textbox named "password" in a form named "entryform" is document.entryform.password.value.

**Methods**

Most objects have a certain collection of things that they can do. Different objects can do different things, just as a door can open and close, while a light can turn on and off. A new document is opened with the method document.open() You can write "Hello World" into a document by typing document.write("Hello World") . open() and write() are both methods of the object: document.

**Events**

Events are how we trigger our functions to run. The easiest example is a button, whose definition includes the words onClick="run_my_function()". The onClick event, as its name implies, will run the function when the user clicks on the button. Other events include OnMouseOver, OnMouseOut, OnFocus, OnBlur, OnLoad, and OnUnload.

JavaScript can be implemented using JavaScript statements that are placed within the <script>... </script> HTML tags in a web page.

You can place the <script> tags, containing your JavaScript, anywhere within you web page, but it is normally recommended that you should keep it within the <head> tags.

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

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

The script tag takes two important attributes −

- **Language** − This attribute specifies what scripting language you are using. Typically, its value will be javascript. Although recent versions of HTML (and XHTML, its successor) have phased out the use of this attribute.
- **Type** − This attribute is what is now recommended to indicate the scripting language in use and its value should be set to "text/javascript".

So JavaScript segment will look like −

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

</script>
```

## LESSON 2

### First JavaScript Script

Let us take a sample example to print out "Hello World". We added an optional HTML comment that surrounds our JavaScript code. This is to save our code from a

browser that does not support JavaScript. The comment ends with a "//-->". Here "//" signifies a comment in JavaScript, so we add that to prevent a browser from reading the end of the HTML comment as a piece of JavaScript code. Next, we call a function document.write which writes a string into our HTML document.

This function can be used to write text, HTML, or both. Take a look at the following code.

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

This code will produce the following result −

Hello World!

**Whitespace and Line Breaks**

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

**Semicolons**

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

```
<script language="javascript" type="text/javascript">
  <!--
    var1 = 10
    var2 = 20
  //-->
</script>
```

But when formatted in a single line as follows, you must use semicolons −

```
<script language="javascript" type="text/javascript">
  <!--
```

```
    var1 = 10; var2 = 20;
  //-->
</script>
```

**Note** − It is a good programming practice to use semicolons.

**Case Sensitivity**

JavaScript is a case-sensitive language. This means that the language keywords, variables, function names, and any other identifiers must always be typed with a consistent capitalization of letters.

So the identifiers Time and TIME will convey different meanings in JavaScript.

> *NOTE − Care should be taken while writing variable and function names in JavaScript.*

**Comments in JavaScript**

JavaScript supports both C-style and C++-style comments, Thus −

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

**Example**

The following example shows how to use comments in JavaScript.

```
<script language="javascript" type="text/javascript">
  <!--

    // This is a comment. It is similar to comments in C++

    /*
    * This is a multiline comment in JavaScript
    * It is very similar to comments in C Programming
    */

  //-->
</script>
```

All the modern browsers come with built-in support for JavaScript. Frequently, you may need to enable or disable this support manually. This chapter explains the procedure of enabling and disabling JavaScript support in your browsers: Internet Explorer, Firefox, chrome, and Opera.

**JavaScript in Internet Explorer**

Here are simple steps to turn on or turn off JavaScript in your Internet Explorer −

- Follow Tools → Internet Options from the menu.
- Select **Security** tab from the dialog box.
- Click the **Custom Level** button.
- Scroll down till you find **Scripting option.**
- Select *Enable* radio button under **Active scripting**.
- Finally click OK and come out

To disable JavaScript support in your Internet Explorer, you need to select **Disable** radio button under **Active scripting**.

**JavaScript in Firefox**

Here are the steps to turn on or turn off JavaScript in Firefox −

- Open a new tab → type **about: config** in the address bar.
- Then you will find the warning dialog. Select **I'll be careful, I promise!**
- Then you will find the list of **configure options** in the browser.
- In the search bar, type **javascript.enabled**.
- There you will find the option to enable or disable javascript by right-clicking on the value of that option → **select toggle**.

If javascript.enabled is true; it converts to false upon clicking **toogle**. If javascript is disabled; it gets enabled upon clicking toggle.

**JavaScript in Chrome**

Here are the steps to turn on or turn off JavaScript in Chrome −

- Click the Chrome menu at the top right hand corner of your browser.
- Select **Settings**.
- Click **Show advanced settings** at the end of the page.
- Under the **Privacy** section, click the Content settings button.

- In the "Javascript" section, select "Do not allow any site to run JavaScript" or "Allow all sites to run JavaScript (recommended)".

**JavaScript in Opera**

Here are the steps to turn on or turn off JavaScript in Opera −

- Follow **Tools → Preferences** from the menu.
- Select **Advanced** option from the dialog box.
- Select **Content** from the listed items.
- Select **Enable JavaScript** checkbox.
- Finally click OK and come out.

To disable JavaScript support in your Opera, you should not select the **Enable JavaScript checkbox**.

**Warning for Non-JavaScript Browsers**

If you have to do something important using JavaScript, then you can display a warning message to the user using **<noscript>** tags.

You can add a **noscript** block immediately after the script block as follows −

```
<html>
  <body>

    <script language="javascript" type="text/javascript">
      <!--
        document.write("Hello World!")
      //-->
    </script>

    <noscript>
      Sorry...JavaScript is needed to go ahead.
    </noscript>

  </body>
</html>
```

Now, if the user's browser does not support JavaScript or JavaScript is not enabled, then the message from </noscript> will be displayed on the screen.

There is a flexibility given to include JavaScript code anywhere in an HTML document. However the most preferred ways to include JavaScript in an HTML file are as follows −

- Script in <head>...</head> section.
- Script in <body>...</body> section.
- Script in <body>...</body> and <head>...</head> sections.
- Script in an external file and then include in <head>...</head> section.

In the following section, we will see how we can place JavaScript in an HTML file in different ways.

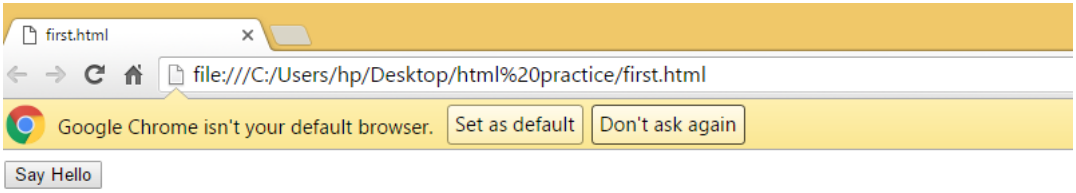
JavaScript in <head>...</head> section

If you want to have a script run on some event, such as when a user clicks somewhere, then you will place that script in the head as follows −

```
<html>

  <head>

    <script type="text/javascript">
      <!--
         function sayHello() {
           alert("Hello World")
         }
      //-->
    </script>

  </head>

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

</html>
```

This code will produce the following results –

Click on the button(say hello)

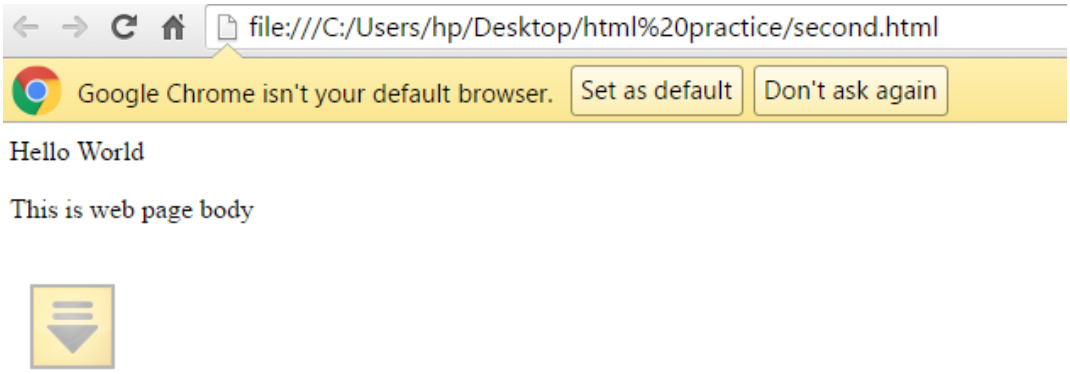



JavaScript in <body>...</body> section

If you need a script to run as the page loads so that the script generates content in the page, then the script goes in the <body> portion of the document. In this case, you would not have any function defined using JavaScript. Take a look at the following code.

```
<html>

  <head>
  </head>

  <body>

    <script type="text/javascript">
      <!--
         document.write("Hello World")
      //-->
    </script>

    <p>This is web page body </p>

  </body>
</html>
```

This code will produce the following results −

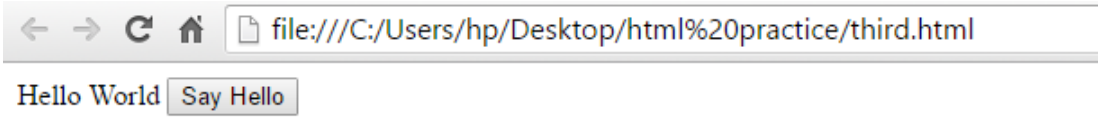



JavaScript in <body> and <head> Sections

You can put your JavaScript code in <head> and <body> section altogether as follows−

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

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

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

  </body>
</html>
```

This code will produce the following result −




## LESSON2

### Including JavaScript Files

As you begin to work more extensively with JavaScript, you will be likely to find that there are cases where you are reusing identical JavaScript code on multiple pages of a site. You are not restricted to be maintaining identical code in multiple HTML files. The **script** tag provides a mechanism to allow you to store JavaScript in an external file and then include it into your HTML files. Here is an example to show how you can include an external JavaScript file in your HTML code using **script** tag and its **src** attribute.

```
<html>
<head>
<script type="text/javascript" src="filename.js" ></script>
</head>
<body>
.....
</body>
</html>
```

To use JavaScript from an external file source, you need to write all your JavaScript source code in a simple text file with the extension ".js" and then include that file as shown above.

For example, you can keep the following content in **filename.js** file and then you can use **sayHello** function in your HTML file after including the filename.js file.

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

**JavaScript Data types**

One of the most fundamental characteristics of a programming language is the set of data types it supports. These are the type of values that can be represented and manipulated in a programming language.

JavaScript allows you to work with three primitive data types:

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

JavaScript also defines two trivial data types, **null** and **undefined**, each of which defines only a single value. In addition to these primitive data types, JavaScript supports a composite data type known as **object**. We will cover objects in detail in a separate chapter.

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

**JavaScript Variables**

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

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

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

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

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

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

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

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

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

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

## LESSON 3

### Naming Rules of variables

- A variable may include only the letters a–z, A–Z, 0–9, the $ symbol, and the

underscore(_).

- No other characters, such as spaces or punctuation, are allowed in a variable name.
- The first character of a variable name can be only a–z, A–Z, $, or _ (no numbers).
- Names are case-sensitive. Count, count, and COUNT are all different variables.
- There is no set limit on variable name lengths.

And yes, you're right, that is a $ there in that list. It *is* allowed by JavaScript and *may* be the first character of a variable or function name. Although I don't recommend keeping the $ symbols, it means that you can port a lot of PHP code more quickly to JavaScript that way.

**JavaScript Reserved Words**

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

| | | | |
|---|---|---|---|
| abstract | else | Instanceof | switch |
| boolean | enum | int | synchronized |
| break | export | interface | this |
| byte | extends | long | throw |
| case | false | native | throws |
| catch | final | new | transient |
| char | finally | null | true |
| class | float | package | try |
| const | for | private | typeof |
| continue | function | protected | var |
| debugger | goto | public | void |
| default | if | return | volatile |
| delete | implements | short | while |
| do | import | static | with |
| double | in | super | |

**Comments in JavaScript**

JavaScript supports both C-style and C++-style comments, Thus −

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

**String Manipulation**

JavaScript strings are used for storing and manipulating text.A JavaScript string simply stores a series of characters like "John Doe".A string can be any text inside quotes. You can use single or double quotes:

Example

```
var carname = "Volvo XC60";
var carname = 'Volvo XC60';
```

You can use quotes inside a string, as long as they don't match the quotes surrounding the string:

```
var answer = "It's alright";
var answer = "He is called 'Johnny'";
var answer = 'He is called "Johnny"';
```

**String Length**

The length of a string is found in the built in property length:

Example

```
var txt = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
var sln = txt.length;
```

**Special Characters**

Because strings must be written within quotes, JavaScript will misunderstand this string:

var y = "We are the so-called "Vikings" from the north." The string will be chopped to "We are the so-called ".The solution to avoid this problem, is to use the \ **escape character**.The backslash escape character turns special characters into string characters:

**Example**

var x = 'It\'s alright';

var y = "We are the so-called \"Vikings\" from the north."

The escape character (\) can also be used to insert other special characters in a string.

This is the list of special characters that can be added to a text string with the backslash sign:

| ode | Outputs |
|---|---|
| \' | single quote |
| \" | double quote |
| \\ | backslash |
| \n | new line |
| \r | carriage return |
| \t | tab |
| \b | backspace |
| \f | form feed |

**Breaking Long Code Lines**

For best readability, programmers often like to avoid code lines longer than 80 characters.

If a JavaScript statement does not fit on one line, the best place to break it is after an operator:

Example

```
document.getElementById("demo").innerHTML =
"Hello Dolly.";
```

You can also break up a code line **within a text string** with a single backslash:

Example

```
document.getElementById("demo").innerHTML = "Hello \
Dolly!";
```

**Strings Can be Objects**

Normally, JavaScript strings are primitive values, created from literals: **var firstName = "John"**

But strings can also be defined as objects with the keyword new: **var firstName = new String("John")**

Example

```
var x = "John";
var y = new String("John");

// typeof x will return string
// typeof y will return object
```

**String Methods and Properties**

Primitive values, like "John Doe", cannot have properties or methods (because they are not objects).

But with JavaScript, methods and properties are also available to primitive values, because JavaScript treats primitive values as objects when executing methods and properties.

**String Length**

The **length** property returns the length of a string:

Example

```
var txt = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
var sln = txt.length;
```

**Finding a String in a String**

The **indexOf()** method returns the index of (the position of) the **first** occurrence of a specified text in a string:

**Example**

```
var str = "Please locate where 'locate' occurs!";
var pos = str.indexOf("locate");
```

The **lastIndexOf()** method returns the index of the **last** occurrence of a specified text in a string:

**Example**

```
var str = "Please locate where 'locate' occurs!";
var pos = str.lastIndexOf("locate");
```

Both the indexOf(), and the lastIndexOf() methods return -1 if the text is not found.

**Searching for a String in a String**

The **search()** method searches a string for a specified value and returns the position of the match:

**Example**

```
var str = "Please locate where 'locate' occurs!";
var pos = str.search("locate");
```

**Extracting String Parts**

There are 3 methods for extracting a part of a string:

- slice(start, end)
- substring(start, end)
- substr(start, length)

**The slice() Method**

**slice()** extracts a part of a string and returns the extracted part in a new string.

The method takes 2 parameters: the starting index (position), and the ending index (position).

This example slices out a portion of a string from position 7 to position 13:

**Example**

```
var str = "Apple, Banana, Kiwi";
var res = str.slice(7,13);
```

The result of res will be:

Banana

If a parameter is negative, the position is counted from the end of the string.

This example slices out a portion of a string from position -12 to position -6:

Example

```
var str = "Apple, Banana, Kiwi";
var res = str.slice(-12,-6);
```

The result of res will be:

If you omit the second parameter, the method will slice out the rest of the string:

Example

```
var res = str.slice(7);
```

or, counting from the end:

Example

```
var res = str.slice(-12);
```

**The substring() Method**

**substring()** is similar to slice().

The difference is that substring() cannot accept negative indexes.

Example

```
var str = "Apple, Banana, Kiwi";
var res = str.substring(7,13);
```

The result of res will be:

Banana

If you omit the second parameter, substring() will slice out the rest of the string.

### The substr() Method

**substr()** is similar to slice().

The difference is that the second parameter specifies the **length** of the extracted part.

**Example**

```
var str = "Apple, Banana, Kiwi";
var res = str.substr(7,6);
```

The result of res will be:

Banana

If the first parameter is negative, the position counts from the end of the string.

The second parameter can not be negative, because it defines the length.

If you omit the second parameter, substr() will slice out the rest of the string.

### Replacing String Content

The **replace()** method replaces a specified value with another value in a string:

**Example**

```
str = "Please visit Microsoft!";
var n = str.replace("Microsoft","W3Schools");
```

The replace() method can also take a regular expression as the search value.

By default, the replace() function replaces only the first match. To replace all matches, use a regular expression with a g flag (for global match):

Example

```
str = "Please visit Microsoft!";
var n = str.replace(/Microsoft/g,"W3Schools");
```

### Converting to Upper and Lower Case

A string is converted to upper case with **toUpperCase()**:

**Example**

```
var text1 = "Hello World!";       // String
var text2 = text1.toUpperCase();  // text2 is text1 converted to upper
```

A string is converted to lower case with **toLowerCase()**:

Example

```
var text1 = "Hello World!";       // String
var text2 = text1.toLowerCase();  // text2 is text1 converted to lower
```

**The concat() Method**

**concat()** joins two or more strings:

Example

```
var text1 = "Hello";
var text2 = "World";
text3 = text1.concat(" ",text2);
```

The **concat()** method can be used instead of the plus operator. These two lines do the same:

```
var text = "Hello" + " " + "World!";
var text = "Hello".concat(" ","World!");
```

**Extracting String Characters**

There are 2 **safe** methods for extracting string characters:

- charAt(position)
- charCodeAt(position)

**The charAt() Method**

The **charAt()** method returns the character at a specified index (position) in a string:

Example

```
var str = "HELLO WORLD";
str.charAt(0);          // returns H
```

**The charCodeAt() Method**

The **charCodeAt()** method returns the unicode of the character at a specified index in a string:

Example

```
var str = "HELLO WORLD";
str.charCodeAt(0);        // returns 72
```

Accessing a String as an Array is Unsafe You might have seen code like this, accessing a string as an array:

```
var str = "HELLO WORLD";
str[0];              // returns H
```

This is **unsafe** and **unpredictable:**

- It does not work in all browsers (not in IE5, IE6, IE7)
- It makes strings look like arrays (but they are not)
- str[0] = "H" does not give an error (but does not work)

If you want to read a string as an array, convert it to an array first.

Converting a String to an Array

A string can be converted to an array with the **split()** method:

```
Example
var txt = "a,b,c,d,e";   // String
txt.split(",");          // Split on commas
txt.split(" ");          // Split on spaces
txt.split("|");          // Split on pipe
```

If the separator is omitted, the returned array will contain the whole string in index [0].If the separator is "", the returned array will be an array of single characters:

```
Example
var txt = "Hello";       // String
txt.split("");           // Split in characters
```

## LESSON 4

### An Operator of Javascript

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

- Arithmetic Operators
- Comparison Operators

- Logical (or Relational) Operators
- Assignment Operators
- Conditional (or ternary) Operators

Let's have a look at all the operators one by one.

**Arithmetic Operators**

JavaScript supports the following arithmetic operators:

Assume variable A holds 10 and variable B holds 20, then:

**OPERATORS**

| S. No. | Operator and Description |
|---|---|
| 1 | + (Addition)<br>Adds two operands<br>Ex: A + B will give 30 |
| 2 | - (Subtraction)<br>Subtracts the second operand from the first<br>Ex: A - B will give -10 |
| 3 | * (Multiplication)<br>Multiply both operands<br>Ex: A * B will give 200 |
| 4 | / (Division)<br>Divide the numerator<br>by the denominator<br>Ex: B / A will give 2 |
| 5 | % (Modulus)<br>Outputs the remainder of an integer division<br>Ex: B % A will give 0 |
| 6 | ++ (Increment)<br>Increases an integer value by one<br>Ex: A++ will give 11 |
| 7 | -- (Decrement)<br>Decreases an integer value by one<br>Ex: A-- will give 9 |

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

**Example**

The following code shows how to use arithmetic operators in JavaScript.

```
<html>
<body>
<script type="text/javascript">
<!--
var a = 33;
var b = 10;
var c = "Test";
var linebreak = "<br />";
document.write("a + b = ");
result = a + b;
document.write(result);
document.write(linebreak);
document.write("a - b = ");
result = a - b;
document.write(result);
document.write(linebreak);
document.write("a / b = ");
result = a / b;
document.write(result);
document.write(linebreak);
document.write("a % b = ");
result = a % b;
document.write(result);
document.write(linebreak);
document.write("a + b + c = ");
result = a + b + c;
document.write(result);
document.write(linebreak);
a = a++;
document.write("a++ = ");
result = a++;
document.write(result);
document.write(linebreak);
```

```
b = b--;
document.write("b-- = ");
result = b--;
document.write(result);
document.write(linebreak);
//-->
</script>
<p>Set the variables to different values and then try...</p>
</body>
</html>
```

Output

```
a + b = 43
a - b = 23
 a / b = 3.3
 a % b = 3
a + b + c = 43
Test a++ = 33
b-- = 10
Set the variables to different values and then try...
```

**Comparison Operators**

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

| S.No. | Operator and Description |
|---|---|
| 1 | == (Equal)<br>Checks if the value of two operands are equal or not, if yes, then the condition becomes true.<br>Ex: (A == B) is not true. |
| 2 | != (Not Equal)<br>Checks if the value of two operands are equal or not, if the values are not equal, then the condition becomes true.<br>Ex: (A != B) is true. |
| 3 | > (Greater than)<br>Checks if the value of the left operand is greater than the value of |
| the right operand, if yes, then the condition becomes true.<br>Ex: (A > B) is not true. | |

| | |
|---|---|
| 4 | < (Less than)<br>Checks if the value of the left operand is less than the value of the right operand, if yes, then the condition becomes true.<br>Ex: (A < B) is true. |
| 5 | >= (Greater than or Equal to)<br>Checks if the value of the left operand is greater than or equal to the value of the right operand, if yes, then the condition becomes true.<br>Ex: (A >= B) is not true. |
| 6 | <= (Less than or Equal to)<br>Checks if the value of the left operand is less than or equal to the value of the right operand, if yes, then the condition becomes true.<br>Ex: (A <= B) is true. |

Example

The following code shows how to use comparison operators in JavaScript.

```
<html>
<body>
<script type="text/javascript">
<!--
var a = 10;
var b = 20;
var linebreak = "<br />";
document.write("(a == b) => ");
result = (a == b);
document.write(result);
document.write(linebreak);
document.write("(a < b) => ");
result = (a < b);
document.write(result);
document.write(linebreak);
document.write("(a > b) => ");
result = (a > b);
document.write(result);
document.write(linebreak);
document.write("(a != b) => ");
result = (a != b);
```

```
document.write(result);
document.write(linebreak);
document.write("(a >= b) => ");
result = (a >= b);
document.write(result);
document.write(linebreak);
document.write("(a <= b) => ");
result = (a <= b);
document.write(result);
document.write(linebreak);
//-->
</script>
<p>Set the variables to different values and different operators and then
try...</p>
</body>
</html>
```

Output

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

Set the variables to different values and different operators and then try...

**Logical Operators**

JavaScript supports the following logical operators:

Assume variable A holds 10 and variable B holds 20, then:

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

**Example**

Try the following code to learn how to implement Logical Operators in JavaScript.

```
<html>
<body>
<script type="text/javascript">
<!--
var a = true;
var b = false;
var linebreak = "<br />";
document.write("(a && b) => ");
result = (a && b);
document.write(result);
document.write(linebreak);
document.write("(a || b) => ");
result = (a || b);
document.write(result);
```

```
document.write(linebreak);
document.write("!(a && b) => ");
result = (!(a && b));
document.write(result);
document.write(linebreak);
//-->
</script>
<p>Set the variables to different values and different operators and then
try...</p>
</body>
</html>
```

**Output**

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

Set the variables to different values and different operators and then try...

Assignment Operators

JavaScript supports the following assignment operators:

| S.No | Operator and Description |
|---|---|
| 1 | = (Simple Assignment )<br>Assigns values from the right side operand to the left side operand<br>Ex: C = A + B will assign the value of A + B into C |
| 2 | += (Add and Assignment)<br>It adds the right operand to the left operand and assigns the result to the left operand.<br>Ex: C += A is equivalent to C = C + A |
| 3 | -= (Subtract and Assignment)<br>It subtracts the right operand from the left operand and assigns the result to the left operand.<br>Ex: C -= A is equivalent to C = C - A |
| 4 | *= (Multiply and Assignment)<br>It multiplies the right operand with the left operand and assigns the result to the left operand.<br>Ex: C *= A is equivalent to C = C * A |

| | |
|---|---|
| 5 | /= (Divide and Assignment)<br>It divides the left operand with the right operand and assigns the result to the left operand.<br>Ex: C /= A is equivalent to C = C / A |
| 6 | %= (Modules and Assignment)<br>It takes modulus using two operands and assigns the result to the left operand.<br>Ex: C %= A is equivalent to C = C % A |

**Example**

Try the following code to implement assignment operator in JavaScript.

```
<html>
<body>
<script type="text/javascript">
<!--
var a = 33;
var b = 10;
var linebreak = "<br />";
document.write("Value of a => (a = b) => ");
result = (a = b);
document.write(result);
document.write(linebreak);
document.write("Value of a => (a += b) => ");
result = (a += b);
document.write(result);
document.write(linebreak);
document.write("Value of a => (a -= b) => ");
result = (a -= b);
document.write(result);
document.write(linebreak);
document.write("Value of a => (a *= b) => ");
result = (a *= b);
document.write(result);
document.write(linebreak);
document.write("Value of a => (a /= b) => ");
result = (a /= b);
```

```
document.write(result);
document.write(linebreak);
document.write("Value of a => (a %= b) => ");
result = (a %= b);
document.write(result);
document.write(linebreak);
//-->
</script>
<p>Set the variables to different values and different operators and then
try...</p>
</body>
</html>
```

Output

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

Conditional Operator (? :)

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

| S.No | Operator and Description |
|---|---|
| 1 | ? : (Conditional )<br>If Condition is true? Then value X : Otherwise value Y |

Example

Try the following code to understand how the Conditional Operator works in JavaScript.

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

```
<!--
var a = 10;
var b = 20;
var linebreak = "<br />";
document.write ("((a > b) ? 100 : 200) => ");
result = (a > b) ? 100 : 200;
document.write(result);
document.write(linebreak);
document.write ("((a < b) ? 100 : 200) => ");
result = (a < b) ? 100 : 200;
document.write(result);
document.write(linebreak);
//-->
</script>
<p>Set the variables to different values and different operators and then
try...</p>
</body>
</html>
```

Output

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

Set the variables to different values and different operators and then typeof Operator

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

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

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

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

| | |
|---|---|
| Null | "object" |

**Example**

The following code shows how to implement **typeof** operator.

```
<html>
<body>
<script type="text/javascript">
<!--
var a = 10;
var b = "String";
var linebreak = "<br />";
result = (typeof b == "string" ? "B is String" : "B is Numeric");
document.write("Result => ");
document.write(result);
document.write(linebreak);
result = (typeof a == "string" ? "A is String" : "A is Numeric");
document.write("Result => ");
document.write(result);
document.write(linebreak);
//-->
</script>
<p>Set the variables to different values and different operators and then
try...</p>
</body>
</html>
```

Output

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

**JavaScript Operator Precedence Values**

| able 1 operator precedence and associativity in JavaScript | | | |
|---|---|---|---|
| Operator | Operator Use | Operator Associativity | Operator Precedence |
| () | Method/function call, grouping | Left to right | Highest — 1 |
| [] | Array access | Left to right | 1 |
| . | Object property access | Left to right | 1 |
| ++ | Increment | Right to left | 2 |
| -- | Decrement | Right to left | 2 |
| - | Negation | Right to left | 2 |
| ! | Logical NOT | Right to left | 2 |
| ~ | Bitwise NOT | Right to left | 2 |
| delete | Removes array value or object property | Right to left | 2 |
| new | Creates an object | Right to left | 2 |
| typeof | Returns data type | Right to left | 2 |
| void | Specifies no value to return | Right to left | 2 |
| / | Division | Left to right | 3 |
| * | Multiplication | Left to right | 3 |
| % | Modulus | Left to right | 3 |
| + | Plus | Left to right | 4 |
| + | String Concatenation | Left to right | 4 |
| - | Subtraction | Left to right | 4 |
| >> | Bitwise right-shift | Left to right | 5 |
| << | Bitwise left-shift | Left to right | 5 |
| >, >= | Greater than, greater than or equal to | Left to right | 6 |
| <, <= | Less than, less than or equal to | Left to right | 6 |
| == | Equality | Left to right | 7 |
| != | Inequality | Left to right | 7 |

| | | | |
|---|---|---|---|
| === | Identity operator — equal to (and same data type) | Left to right | 7 |
| !== | Non-identity operator — not equal to (or don't have the same data type) | Left to right | 7 |
| & | Bitwise AND | Left to right | 8 |
| ^ | Bitwise XOR | Left to right | 9 |
| \| | Bitwise OR | Left to right | 10 |
| && | Logical AND | Left to right | 11 |
| \|\| | Logical OR | Left to right | 12 |
| ?: | Conditional branch | Left to right | 13 |
| = | Assignment | Right to left | 14 |
| *=, /=, %=, +=,, -=, <<=, >>=, >>>=, &=, ^=, \|= | Assignment according to the preceding operator | Right to left | 14 |
| , | Multiple evaluation | Left to right | Lowest: 15 |

Consider the following example:

5 + 6 - 11 + 8 * 5 + 4 = ?

In this example, we are using three instances of the addition/plus (+) operator. Without a predefined operator order precedence, we'll likely all have different answers. Fortunately, we can use the precedence and associativity of JavaScript's operators information shown in table 1 to avoid any conflicting results. According to this table, the multiplication operator (*) has higher precedence than plus and subtraction operators. Also, the table indicates the plus operator and the subtraction operator has the same level of precedence and their associativity indicates that we evaluate them left to right.

So to evaluate this expression, we'll first multiply 8 * 5 which will equal 40. After this operation is performed, our arithmetic expression becomes

5 + 6 - 11 + 40 + 4 = ?

We evaluate this expression left to right starting with adding 5 + 6. This yields 11.

Next we subtract 11 from 11 and this yields 0. So, we are left with 0 + 40 + 4 which equals 44. The following shows sequence of operations used to obtain the final result:

## LESSON 5

### Control Statement

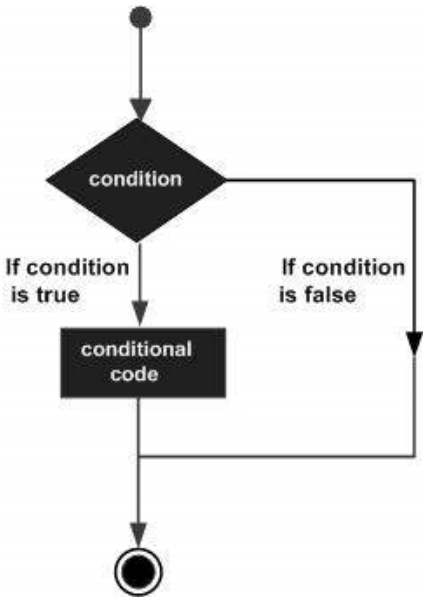
IF-ELSE

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

JavaScript supports conditional statements which are used to perform different actions based on different conditions. Here we will explain the **if..else** statement.

FlowChartof if-else

The following flow chart shows how the if-else statement works.




JavaScript supports the following forms of **if..else** statement −

- if statement
- if...else statement
- if...else if... statement.

**if statement**

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

## Syntax

The syntax for a basic if statement is as follows −

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

Here, a JavaScript expression is evaluated. If the resulting value is true, the given statement(s) are executed. If the expression is false, then no statement would be not executed. Most of the times, you will use comparison operators while making decisions

Example

Try the following example to understand how the **if** statement works.

```
<html>
<body>
<script type="text/javascript">
<!--
var age = 20;
if( age > 18 ){
document.write("<b>Qualifies for driving</b>");
}
//-->
</script>
<p>Set the variable to different value and then try...</p>
</body>
</html>
```

Output

Qualifies for driving

Set the variable to different value and then try...

if...else Statement

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

Syntax

The syntax of an **if-else** statement is as follows:

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

Here, JavaScript expression is evaluated. If the resulting value is true, the given statement(s) in the 'if' block, are executed. If the expression is false, then the given statement(s) in the else block are executed.

**Example**

Try the following code to learn how to implement an if-else statement in JavaScript.

```
<html>
<body>
<script type="text/javascript">
<!--
var age = 15;
if( age > 18 ){
document.write("<b>Qualifies for driving</b>");
}else{
document.write("<b>Does not qualify for driving</b>");
}
//-->
</script>
<p>Set the variable to different value and then try...</p>
</body>
</html>
```

**Output**

Does not qualify for driving

Set the variable to different value and then try...

if...else if... Statement

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

**Syntax**

The syntax of an if-else-if statement is as follows:

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

There is nothing special about this code. It is just a series of **if** statements, where each **if** is a part of the **else** clause of the previous statement. Statement(s) are executed based on the true condition, if none of the conditions is true, then the **else** block is executed.

**Example**

Try the following code to learn how to implement an if-else-if statement in JavaScript.

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

```
}
//-->
</script>
<p>Set the variable to different value and then try...</p>
</body>
</html>
```

Output

**Maths Book**

Set the variable to different value and then try...

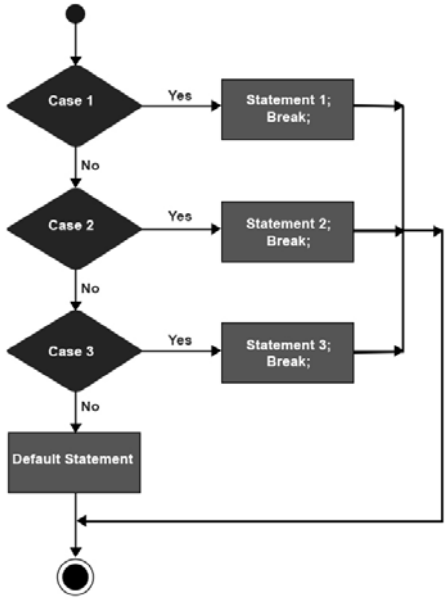
## SWITCH-CASE

You can use multiple **if...else…if** statements, as in the previous chapter, to perform a multiway branch. However, this is not always the best solution, especially when all of the branches depend on the value of a single variable.

Starting with JavaScript 1.2, you can use a **switch** statement which handles exactly this situation, and it does so more efficiently than repeated **if...else if** statements.

### FlowChart

The following flow chart explains a switch-case statement works.




**Syntax**

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

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

The **break** statements indicate the end of a particular case. If they were omitted, the interpreter would continue executing each statement in each of the following cases.

We will explain **break** statement in **Loop Control** chapter.

Example

Try the following example to implement switch-case statement.

```
<html>
<body>
<script type="text/javascript">
<!--
var grade='A';
document.write("Entering switch block<br />");
switch (grade)
{
break;
case 'B': document.write("Pretty good<br />");
break;
case 'C': document.write("Passed<br />");
break;
case 'D': document.write("Not so good<br />");
break;
case 'F': document.write("Failed<br />");
break;
default: document.write("Unknown grade<br />")
}
document.write("Exiting switch block");
//-->
</script>
<p>Set the variable to different value and then try...</p>
</body>
</html>
```

Output

Entering switch block Good job Exiting switch block

Set the variable to different value and then try...

Break statements play a major role in switch-case statements. Try the following code that uses switch-case statement without any break statement.

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

```
<script type="text/javascript">
<!--
var grade='A';
document.write("Entering switch block<br />");
switch (grade)
{
case 'A': document.write("Good job<br />");
case 'B': document.write("Pretty good<br />");
case 'C': document.write("Passed<br />");
case 'D': document.write("Not so good<br />");
case 'F': document.write("Failed<br />");
default: document.write("Unknown grade<br />")
}
document.write("Exiting switch block");
//-->
</script>
<p>Set the variable to different value and then try...</p>
</body>
</html>
```

**Output**

Entering switch block Good job Pretty good Passed Not so good Failed Unknown grade Exiting switch block

Set the variable to different value and then try...

## Another example

```
switch (new Date().getDay()) {
   case 0:
      day = "Sunday";
      break;
   case 1:
      day = "Monday";
      break;
   case 2:
      day = "Tuesday";
      break;
   case 3:
```

```
        day = "Wednesday";
        break;
    case 4:
        day = "Thursday";
        break;
    case 5:
        day = "Friday";
        break;
    case 6:
        day = "Saturday";
}
document.getElementById("demo").innerHTML = "Today is " + day;
</script>

</body>
</html>
```

The result of day will be:

Tuesday

**The default Keyword**

The **default** keyword specifies the code to run if there is no case match:

Example

The getDay() method returns the weekday as a number between 0 and 6.

If today is neither Saturday (6) nor Sunday (0), write a default message:

```
switch (new Date().getDay()) {
    case 6:
        text = "Today is Saturday";
        break;
    case 0:
        text = "Today is Sunday";
        break;
    default:
        text = "Looking forward to the Weekend";
}
```

The result of text will be:

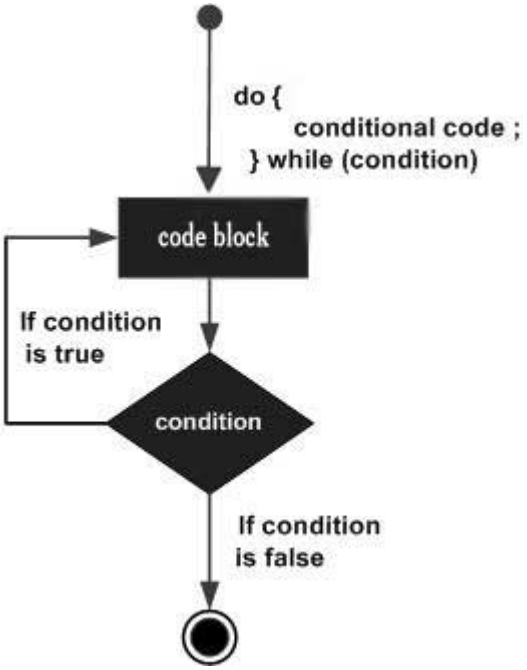
Looking forward to the Weekend

**Do...whileLoop**

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

**Flow Chart**

The flow chart of a **do-while** loop would be as follows:




**Syntax**

The syntax for **do-while** loop in JavaScript is as follows:

```
do{
Statement(s) to be executed;
} while (expression);
```

**Note:** Don't miss the semicolon used at the end of the **do...while** loop.

Example 1

Try the following example to learn how to implement a **do-while** loop in JavaScript.

```
<html>
<body>
<script type="text/javascript">
<!--
var count = 0;
document.write("Starting Loop" + "<br />");
```

```
do{
document.write("Current Count : " + count + "<br />");
count++;
}while (count < 5);
document.write ("Loop stopped!");
//-->
</script>
<p>Set the variable to different value and then try...</p>
</body>
</html>
```

Output

```
Starting Loop
Current Count : 0 Current Count : 1 Current Count : 2 Current Count : 3
Current Count : 4
Loop Stopped!

//-->
</script>
<p>Set the variable to different value and then try...</p>
</body>
</html>
```

## WHILE LOOP

While writing a program, you may encounter a situation where you need to perform an action over and over again. In such situations, you would need to write loop statements to reduce the number of lines.

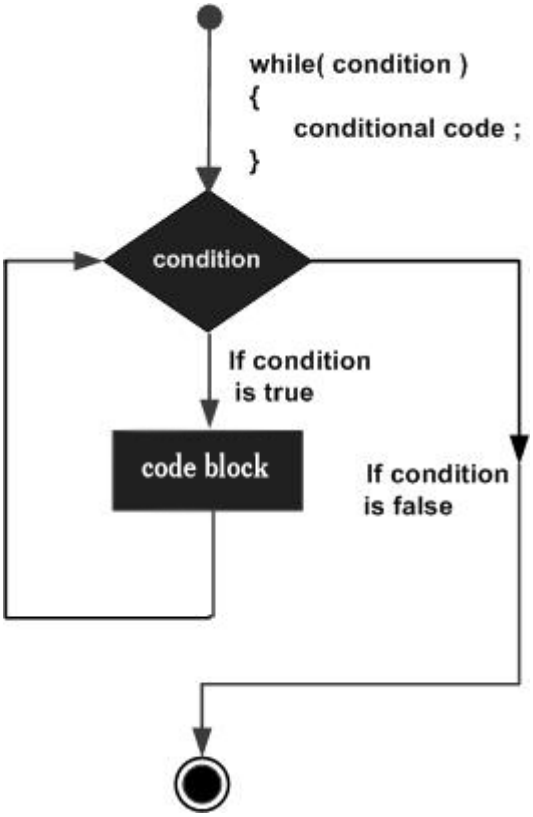
JavaScript supports all the necessary loops to ease down the pressure of programming.

**The while Loop**

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

**Flow Chart**

The flow chart of **while loop** looks as follows:




Syntax

The syntax of **while loop** in JavaScript is as follows:

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

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

<p>Click the button to loop through a block of code as long as i is less than 10.</p>

<button onclick="myFunction()">Try it</button>
<p id="demo"></p>
```

```
<script>
function myFunction() {
   var text = "";
   var i = 0;
   while (i < 10) {
      text += "<br>The number is " + i;
      i++;
   }
   document.getElementById("demo").innerHTML = text;
}
</script>

</body>
</html>
```

Output

Click the button to loop through a block of code as long as i is less than 10.

Try it

The number is 0
The number is 1
The number is 2
The number is 3
The number is 4
The number is 5
The number is 6
The number is 7
The number is 8
The number is 9

**Example 2**

Try the following example to implement while loop.

```
<html>
<body>
<script type="text/javascript">
<!--
var count = 0;
document.write("Starting Loop ");
while (count < 10){
document.write("Current Count : " + count + "<br />");
count++;
}
document.write("Loop stopped!");
//-->
</script>
<p>Set the variable to different value and then try...</p>
</body>
</html>
```

Output

```
Starting Loop Current Count : 0
Current Count : 1
 Current Count : 2
 Current Count : 3
 Current Count : 4
Current Count : 5
 Current Count : 6
Current Count : 7
Current Count : 8
Current Count : 9
 Loop stopped!
Set the variable to different value and then try...
```

**FOR LOOP**

The for Loop

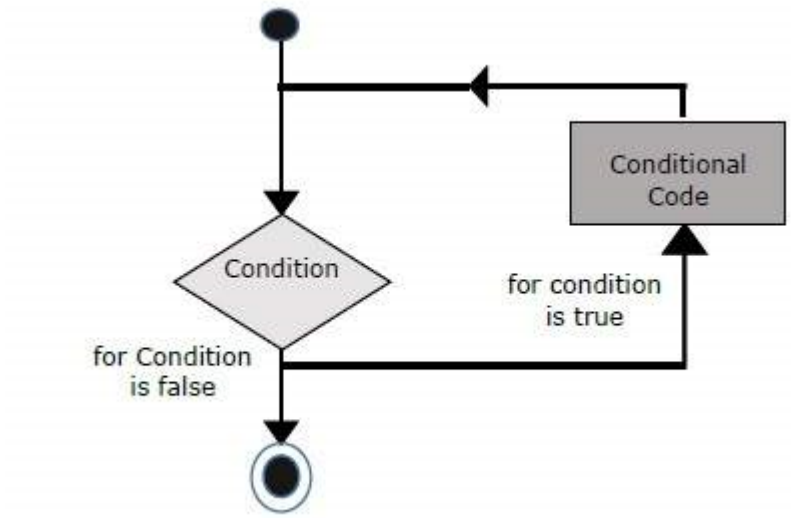
The '**for**' loop is the most compact form of looping. It includes the following three important parts:

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

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

Flow Chart

The flow chart of a **for** loop in JavaScript would be as follows:




Syntax

The syntax of **for** loop is JavaScript is as follows:

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

}

Example

Try the following example to learn how a **for** loop works in JavaScript.

```
<html>
<body>
<script type="text/javascript">
<!--
var count;
document.write("Starting Loop" + "<br />");
for(count = 0; count < 10; count++){
document.write("Current Count : " + count );
document.write("<br />");
}
document.write("Loop stopped!");
//-->
</script>
<p>Set the variable to different value and then try...</p>
</body>
</html>
```

Output

```
Starting Loop Current Count : 0
 Current Count : 1
Current Count : 2
Current Count : 3
Current Count : 4
Current Count : 5
Current Count : 6
 Current Count : 7
 Current Count : 8
 Current Count : 9
Loop stopped!
Set the variable to different value and then try...
```

Example

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

<p id="demo"></p>

<script>
var cars = ["BMW", "Volvo", "Saab", "Ford"];
var text = "";
var i;
for (i = 0; i < cars.length; i++) {
   text += cars[i] + "<br>";
}

document.getElementById("demo").innerHTML = text;
</script>


</body>
</html>
```

Output

BMW
Volvo
Saab
Ford

**FOR-IN LOOP**

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

Syntax

The syntax of 'for..in' loop is:

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

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

Example

Try the following example to implement ‘for-in’ loop. It prints the web browser’s Navigator object.

```
<html>
<body>
<script type="text/javascript">
<!--
var aProperty;
document.write("Navigator Object Properties<br /> ");
for (aProperty in navigator)
{
document.write(aProperty);
document.write("<br />");
}
document.write ("Exiting from the loop!");
//-->
</script>
<p>Set the variable to different object and then try...</p>
</body>
</html>
```

Output

```
Navigator Object Properties
 serviceWorker
 webkitPersistentStorage
webkitTemporaryStorage
 geolocation
doNotTrack
onLine
languages
language
 userAgent
 product
 platform
```

appVersion
appName
appCodeName
hardwareConcurrency
maxTouchPoints
vendorSub
vendor
productSub
cookieEnabled
mimeTypes
plugins
javaEnabled
getStorageUpdates
getGamepads
webkitGetUserMedia
vibrate
getBattery
sendBeacon
registerProtocolHandler
unregisterProtocolHandler
Exiting from the loop!
Set the variable to different object and then try...

**LOOP CONTROL**

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

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

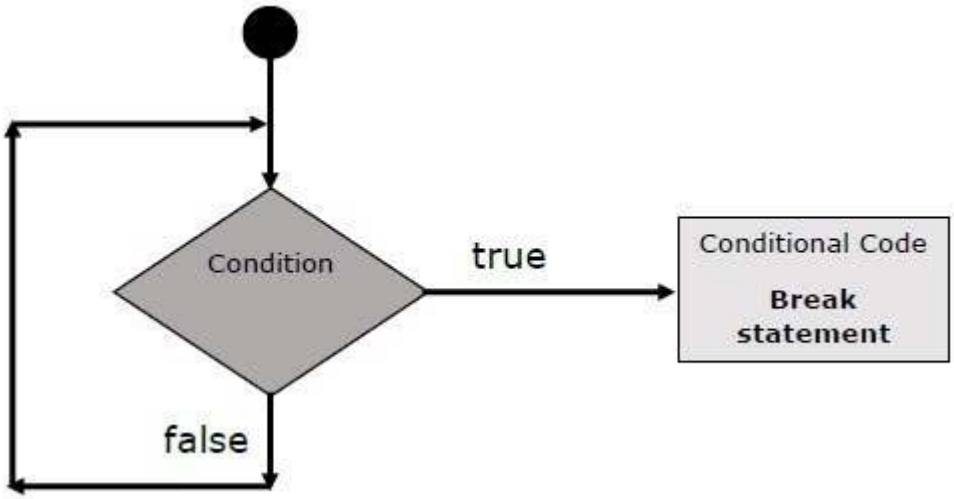
**The break Statement**

The **break** statement, which was briefly introduced with the *switch* statement, is used

to exit a loop early, breaking out of the enclosing curly braces.

**Flow Chart**

The flow chart of a break statement would look as follows:




Example

The following example illustrates the use of a **break** statement with a while loop. Notice how the loop breaks out early once **x** reaches 5 and reaches to **document.write (..)** statement just below to the closing curly brace:

```
<html>

<body>

<script type="text/javascript">

<!--

var x = 1;

document.write("Entering the loop<br /> ");

while (x < 20)

{

if (x == 5){
```

```
break; // breaks out of loop completely
}
x = x + 1;
document.write( x + "<br />");
}
document.write("Exiting the loop!<br /> ");
//-->
</script>
<p>Set the variable to different value and then try...</p>
</body>
</html>
```

Output

```
Entering the loop
2
3
4
5 Exiting the loop!
Set the variable to different value and then try...
```

**The continue Statement**

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

**Example**

This example illustrates the use of a **continue** statement with a while loop.

Notice how the **continue** statement is used to skip printing when the index held in variable **x** reaches 5.

```
<html>
<body>
<script type="text/javascript">
<!--
var x = 1;
document.write("Entering the loop<br /> ");
while (x < 10)
{
x = x + 1;
if (x == 5){
continue; // skill rest of the loop body
}
document.write( x + "<br />");
}
document.write("Exiting the loop!<br /> ");
//-->
</script>
<p>Set the variable to different value and then try...</p>
</body>
</html>
```

```
Entering the loop
2
3
4
6
7
8
9
10
Exiting the loop!
```

## LESSON 6

## FUNCTIONS

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

Like any other advanced programming language, JavaScript also supports all the features necessary to write modular code using functions. You must have seen functions like **alert()** and **write()** in the earlier chapters. We were using these functions again and again, but they had been written in core JavaScript only once.

JavaScript allows us to write our own functions as well. This section explains how to write your own functions in JavaScript.

### Function Definition

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

Syntax

The basic syntax is shown here.

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

Example

Try the following example. It defines a function called sayHello that takes no parameters:

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

```
<!--
function sayHello()
{
alert("Hello there");
}
//-->
</script>
```

**Calling a Function**

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

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

Output

Click the following button to call the function


Top of Form

Bottom of Form

Use different

**Function Parameters**

Till now, we have seen functions without parameters. But there is a facility to pass different parameters while calling a function. These passed parameters can be captured inside the function and any manipulation can be done over those parameters. A function can take multiple parameters separated by comma.

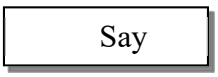
Example

Try the following example. We have modified our **sayHello** function here. Now it takes two parameters.

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

Output

Click the following button to call the function




Use different parameters inside the function and then try...

ThereturnStatement

A JavaScript function can have an optional **return** statement. This is required if you

want to return a value from a function. This statement should be the last statement in a function.

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

**Example**

Try the following example. It defines a function that takes two parameters and concatenates them before returning the resultant in the calling program.

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

Output

Click the following button to call the function

Call

Use different parameters inside the function and then try...

There is a lot to learn about JavaScript functions, however we have covered the most important concepts in this tutorial.

**Nested Functions**

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

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

**Example**

Try the following example to learn how to implement nested functions.

```
<html>
<head>
<script type="text/javascript">
<!--
function hypotenuse(a, b) {
function square(x) { return x*x; }
return Math.sqrt(square(a) + square(b));
}
function secondFunction(){
var result;
result = hypotenuse(1,2);
document.write ( result );
}
//-->
```

```
</script>
</head>
<body>
<p>Click the following button to call the function</p>
<form>
<input type="button" onclick="secondFunction()" value="Call Function">
</form>
<p>Use different parameters inside the function and then try...</p>
</body>
</html>
```

Output

Click the following button to call the function


## Lesson 7

### Object Oriented Programming

OOP is a programming technique (note: not a language structure - you don't even need an object-oriented language to program in an object-oriented fashion) designed to simplify complicated programming concepts. In essence, object-oriented programming revolves around the idea of user- and system-defined chunks of data, and controlled means of accessing and modifying those chunks.

Object-oriented programming consists of Objects, Methods and Properties. An object is basically a black box which stores some information. It may have a way for you to read that information and a way for you to write to, or change, that information. It may also have other less obvious ways of interacting with the information.

Some of the information in the object may actually be directly accessible; other information may require you to use a method to access it - perhaps because the way the information is stored internally is of no use to you, or because only certain things can be written into that information space and the object needs to check that you're not going outside those limits.

The directly accessible bits of information in the object are its properties. The difference between data accessed via properties and data accessed via methods is that with properties, you see exactly what you're doing to the object; with methods, unless you created the object yourself, you just see the effects of what you're doing.

Other Javascript pages you read will probably refer frequently to objects, events, methods, and properties. This tutorial will teach by example, without focusing too heavily on OOP vocabulary. However, you will need a basic understanding of these terms to use other JavaScript references.

Objects and Properties

Your web page document is an object. Any table, form, button, image, or link on your page is also an object. Each object has certain properties (information about the object). For example, the background color of your document is written **document.bgcolor**. You would change the color of your page to red by writing the line: document.bgcolor="red"

The contents (or value) of a textbox named "password" in a form named "entryform" is **document.entryform.password.value**.

**Methods**

Most objects have a certain collection of things that they can **do**. Different objects can do different things, just as a door can open and close, while a light can turn on and off. A new document is opened with the method **document.open()** You can write "Hello World" into a document by typing **document.write("Hello World")** . **open()** and **write()** are both *methods* of the object: document.

**Events**

Events are how we trigger our functions to run. The easiest example is a button, whose definition includes the words **onClick="run_my_function()"**. The onClick event, as its name implies, will run the function when the user clicks on the button. Other events include OnMouseOver, OnMouseOut, OnFocus, OnBlur, OnLoad, and OnUnload.

JavaScript can be implemented using JavaScript statements that are placed within the **<script>... </script>** HTML tags in a web page.

You can place the **<script>** tags, containing your JavaScript, anywhere within you web page, but it is normally recommended that you should keep it within the **<head>** tags.

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

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

**OBJECTS**

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

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

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

**Object Properties**

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

The syntax for adding a property to an object is:

objectName.objectProperty = propertyValue;

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

var str = document.title;

**Object Methods**

Methods are the functions that let the object do something or let something be done to it. There is a small difference between a function and a method – at a function is a standalone unit of statements and a method is attached to an object and can be referenced by the **this** keyword. Methods are useful for everything from displaying the contents of the object to the screen to performing complex mathematical operations on a group of local properties and parameters.

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

document.write ("This is test");

**User-Defined Objects**

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

**The new Operator**

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

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

var employee = new Object();

var books = new Array("C++", "Perl", "Java");

var day = new Date("August 15, 1947");

**TheObject()Constructor**

A constructor is a function that creates and initializes an object. JavaScript provides

a special constructor function called **Object()** to build the object. The return value of the **Object()** constructor is assigned to a variable.

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

Example 1
Try the following example; it demonstrates how to create an Object.

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

Output

```
Book name is : Perl
Book author is : Mohtashim
```

Example 2

This example demonstrates how to create an object with a User-Defined Function. Here **this** keyword is used to refer to the object that has been passed to a function.

```
<html>
<head>
<title>User-defined objects</title>
<script type="text/javascript">
```

```
function book(title, author){
this.title = title;
this.author = author;
}
</script>
</head>
<body>
<script type="text/javascript">
var myBook = new book("Perl", "Mohtashim");
document.write("Book title is : " + myBook.title + "<br>");
document.write("Book author is : " + myBook.author + "<br>");
</script>
</body>
</html>
```

Output

Book title is : Perl

Book author is : Mohtashim

**Defining Methods for an Object**

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

Example

Try the following example; it shows how to add a function along with an object.

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

```
this.author = author;
this.addPrice = addPrice; // Assign that method as property.
}
</script>
</head>
<body>
<script type="text/javascript">
var myBook = new book("Perl", "Mohtashim");
myBook.addPrice(100);
document.write("Book title is : " + myBook.title + "<br>");
document.write("Book author is : " + myBook.author + "<br>");
document.write("Book price is : " + myBook.price + "<br>");
</script>
</body>
</html>
```

Output

Book title is : Perl
Book author is : Mohtashim
Book price is : 100

The‘with’Keyword

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

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

Syntax

The syntax for with object is as follows:

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

Example

Try the following example.

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

```
<title>User-defined objects</title>
<script type="text/javascript">
// Define a function which will work as a method
function addPrice(amount){
with(this){
price = amount;
}
}
function book(title, author){
this.title = title;
this.author = author;
this.price = 0;
this.addPrice = addPrice; // Assign that method as property.
}
</script>
</head>
<body>
<script type="text/javascript">
var myBook = new book("Perl", "Mohtashim");
myBook.addPrice(100);
document.write("Book title is : " + myBook.title + "<br>");
document.write("Book author is : " + myBook.author + "<br>");
document.write("Book price is : " + myBook.price + "<br>");
</script>
</body>
</html>
```

Output

```
Book title is : Perl
Book author is : Mohtashim
Book price is : 100
```

**ARRAYS**

The **Array** object lets you store multiple values in a single variable. It stores a fixed-size sequential collection of elements of the same type. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type.

**Syntax**

Use the following syntax to create an **Array** Object.

var fruits = new Array( "apple", "orange", "mango" );

The **Array** parameter is a list of strings or integers. When you specify a single numeric parameter with the Array constructor, you specify the initial length of the array. The maximum length allowed for an array is 4,294,967,295.

You can create array by simply assigning values as follows:

var fruits = [ "apple", "orange", "mango" ];

You will use ordinal numbers to access and to set values inside an array as

fruits[0] is the first element
fruits[1] is the second element
fruits[2] is the third element
Array Properties

Here is a list of the properties of the Array object along with their description.

| Property | Description |
|---|---|
| constructor | Returns a reference to the array function that created the object. |
| index | The property represents the zero-based index of the match in the string |
| input | This property is only present in arrays created by regular expression matches. |
| length | Reflects the number of elements in an array. |
| prototype | The prototype property allows you to add properties and methods to an object. |

In the following sections, we will have a few examples to illustrate the usage of Array properties.

**constructor**

Javascript array **constructor** property returns a reference to the array function that created the instance's prototype.

Syntax
Its syntax is as follows:
array.constructor
Return Value
Returns the function that created this object's instance.

Example
Try the following example.

```
<html>
<head>
<title>JavaScript Array constructor Property</title>
</head>
<body>
<script type="text/javascript">
var arr = new Array( 10, 20, 30 );
document.write("arr.constructor is:" + arr.constructor);
</script>
</body>
</html>
```

Output

```
arr.constructor is:function Array() { [native code] }
```

length

Javascript array **length** property returns an unsigned, 32-bit integer that specifies the number of elements in an array

Syntax
Its syntax is as follows:
array.length
Return Value

Returns the length of an array.

Example

Try the following example.

```
<html>
<head>
<title>JavaScript Array length Property</title>
</head>
<body>
<script type="text/javascript">
var arr = new Array( 10, 20, 30 );
document.write("arr.length is:" + arr.length);
</script>
</body>
</html>
```

Output

```
arr.length is:3
```

**Array Methods**

Here is a list of the methods of the Array object along with their description.

| Method | Description |
|---|---|
| concat() | Returns a new array comprised of this array joined with other array(s) and/or value(s). |
| every() | Returns true if every element in this array satisfies the provided testing function. |
| filter() | Creates a new array with all of the elements of this array for which the provided filtering function returns true. |
| forEach() | Calls a function for each element in the array. |
| indexOf() | Returns the first (least) index of an element within the array equal to the specified value, or -1 if none is found. |

| | |
|---|---|
| join() | Joins all elements of an array into a string. |
| lastIndexOf() | Returns the last (greatest) index of an element within the array equal to the specified value, or -1 if none is found. |
| map() | Creates a new array with the results of calling a provided function on every element in this array. |
| pop() | Removes the last elem |
| push() | Adds one or more elements to the end of an array and returns the new length of the array. |
| reduce() | Apply a function simultaneously against two values of the array (from left-to-right) as to reduce it to a single value. |
| reduceRight() | Apply a function simultaneously against two values of the array (from right-to-left) as to reduce it to a single value. |
| reverse() | Reverses the order of the elements of an array -- the first becomes the last, and the last becomes the first. |
| shift() | Removes the first element from an array and returns that element. |
| slice() | Extracts a section of an array and returns a new array. |
| some() | Returns true if at least one element in this array satisfies the provided testing function. |
| toSource() | Represents the source code of an object |
| sort() | Sorts the elements of an array. |
| splice() | Adds and/or removes elements from an array. |
| toString() | Returns a string representing the array and its elements. |
| unshift() | Adds one or more elements to the front of an array and returns the new length of the array. |

In the following sections, we will have a few examples to demonstrate the usage of Array methods

concat()

Javascript array **concat()** method returns a new array comprised of this array joined with two or more arrays.

Syntax
The syntax of concat() method is as follows:
array.concat(value1, value2, ..., valueN);
Parameter Details

**valueN** : Arrays and/or values to concatenate to the resulting array.
Return Value
Returns the length of the array.
Example
Try the following example.

```
<html>
<head>
<title>JavaScript Array concat Method</title>
</head>
<body>
<script type="text/javascript">
var alpha = ["a", "b", "c"];
var numeric = [1, 2, 3];
var alphaNumeric = alpha.concat(numeric);
document.write("alphaNumeric : " + alphaNumeric );
</script>
</body>
</html>
```

Output
alphaNumeric : a,b,c,1,2,3

sort ()

Javascript array **sort()** method sorts the elements of an array.

Syntax

Its syntax is as follows:

```
array.sort( compareFunction );
```

**Parameter Details**

**compareFunction:** Specifies a function that defines the sort order. If omitted, the array is sorted lexicographically.

Return Value

Returns a sorted array.

Example

Try the following example.

```
<html>
<head>
<title>JavaScript Array sort Method</title>
</head>
<body>
<script type="text/javascript">
var arr = new Array("orange", "mango", "banana", "sugar");
var sorted = arr.sort();
document.write("Returned string is : " + sorted );
</script>
</body>
</html>
```

Output

```
Returned string is : banana,mango,orange,sugar
```

**DATE**

The Date object is a datatype built into the JavaScript language. Date objects are created with the **new Date()** as shown below.

Once a Date object is created, a number of methods allow you to operate on it. Most methods simply allow you to get and set the year, month, day, hour, minute, second,

and millisecond fields of the object, using either local time or UTC (universal, or GMT) time.

The ECMAScript standard requires the Date object to be able to represent any date and time, to millisecond precision, within 100 million days before or after 1/1/1970. This is a range of plus or minus 273,785 years, so JavaScript can represent date and time till the year 275755.

**Syntax**

You can use any of the following syntaxes to create a Date object using Date() constructor.

new Date( )

new Date(milliseconds)

new Date(datestring)

new Date(year,month,date[,hour,minute,second,millisecond ])

**Note:** Parameters in the brackets are always optional.

Here is a description of the parameters:

- **No Argument:** With no arguments, the Date() constructor creates a Date object set to the current date and time.
- **milliseconds:** When one numeric argument is passed, it is taken as the internal numeric representation of the date in milliseconds, as returned by the getTime() method. For example, passing the argument 5000 creates a date that represents five seconds past midnight on 1/1/70.
- **datestring:** When one string argument is passed, it is a string representation of a date, in the format accepted by the **Date.parse()** method.
- **7 agruments:** To use the last form of the constructor shown above. Here is a description of each argument:
- o **year:** Integer value representing the year. For compatibility (in order to avoid the Y2K problem), you should always specify the year in full; use 1998, rather than 98.
- o **month:** Integer value representing the month, beginning with 0 for January to 11 for December.
- **date:** Integer value representing the day of the month.

- **hour:** Integer value representing the hour of the day (24-hour scale).
- **minute:** Integer value representing the minute segment of a time reading.
- **second:** Integer value representing the second segment of a time reading.
- **millisecond:** Integer value representing the millisecond segment of a time reading

**Date Properties**

Here is a list of the properties of the Date object along with their description.

| Property | Description |
|---|---|
| constructor | Specifies the function that creates an object's prototype. |
| prototype | The prototype property allows you to add properties and methods to an object. |

In the following sections, we will have a few examples to demonstrate the usage of different Date properties.

**constructor**

Javascript date **constructor** property returns a reference to the array function that created the instance's prototype.
Syntax
Its syntax is as follows:
date.constructor
Return Value
Returns the function that created this object's instance.
Example
Try the following example.

```
<html>
<head>
<title>JavaScript Date constructor Property</title>
</head>
<body>
<script type="text/javascript">
```

```
var dt = new Date();
document.write("dt.constructor is : " + dt.constructor);
</script>
</body>
</html>
```

Output

```
dt.constructor is : function Date() { [native code] }
```

**Prototype**

The prototype property allows you to add properties and methods to any object (Number, Boolean, String, Date, etc.).

**Note:** Prototype is a global property which is available with almost all the objects.

Syntax

Its syntax is as follows:

```
object.prototype.name = value
```

Example

Try the following example.

```
<html>
<head>
<title>User-defined objects</title>
<script type="text/javascript">
function book(title, author){
this.title = title;
this.author = author;
}
</script>
</head>
<body>
<script type="text/javascript">
var myBook = new book("Perl", "Mohtashim");
book.prototype.price = null;
myBook.price = 100;
document.write("Book title is : " + myBook.title + "<br>");
document.write("Book author is : " + myBook.author + "<br>");
```

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

Output

```
Book title is : Perl
Book author is : Mohtashim
 Book price is : 100
```

**Date Methods**

Here is a list of the methods used with **Date** and their description.

| Method | Description |
|---|---|
| Date() | Returns today's date and time |
| getDate() | Returns the day of the month for the specified date according to local time. |
| getDay() | Returns the day of the week for the specified date according to local time. |
| getFullYear() | Returns the year of the specified date according to local time. |
| getHours() | Returns the hour in the specified date according to local time. |
| getMilliseconds() | Returns the milliseconds in the specified date according to local time. |
| getMinutes() | Returns the minutes in the specified date according to local time. |
| getMonth() | Returns the month in the specified date according to local time. |
| getSeconds() | Returns the seconds in the specified date according to local time. |
| getTime() | Returns the numeric value of the specified date as the number of milliseconds since January 1, 1970, 00:00:00 UTC. |
| getTimezoneOffset() | Returns the time-zone offset in minutes for the current locale. |
| getUTCDate() | Returns the day (date) of the month in the specified date according to universal time. |

| | |
|---|---|
| getUTCDay() | Returns the day of the week in the specified date according to universal time. |
| getUTCFullYear() | Returns the year in the specified date according to universal time. |
| getUTCHours() | Returns the hours in the specified date according to universal time. |
| getUTCMilliseconds() | Returns the milliseconds in the specified date |
| getUTCMinutes() | Returns the minutes in the specified date according to universal time. |
| getUTCMonth() | Returns the month in the specified date according to universal time. |
| getUTCSeconds() | Returns the seconds in the specified date according to universal time. |
| getYear() | Deprecated - Returns the year in the specified date according to local time. Use getFullYear instead. |
| setDate() | Sets the day of the month for a specified date according to local time. |
| setFullYear() | Sets the full year for a specified date according to local time. |
| setHours() | Sets the hours for a specified date according to local time. |
| setMilliseconds() | Sets the milliseconds for a specified date according to local time. |
| setMinutes() | Sets the minutes for a specified date according to local time. |
| setMonth() | Sets the month for a specified date according to local time. |
| setSeconds() | Sets the seconds for a specified date according to local time. |
| setTime() | Sets the Date object to the time represented by a number of milliseconds since January 1, 1970, 00:00:00 UTC. |
| setUTCDate() | Sets the day of the month for a specified date according to universal time. |
| setUTCFullYear() | Sets the full year for a specified date according to universal time. |
| setUTCHours() | Sets the hour for a specified date according to |

| | |
|---|---|
| | universal time. |
| setUTCMilliseconds() | Sets the milliseconds for a specified date according to universal time. |
| setUTCMinutes() | Sets the minutes for a specified date according to universal time |
| | |
| setUTCMonth() | Sets the month for a specified date according to universal time. |
| setUTCSeconds() | Sets the seconds for a specified date according to universal time. |
| setYear() | Deprecated - Sets the year for a specified date according to local time. Use setFullYear instead. |
| toDateString() | Returns the "date" portion of the Date as a human-readable string. |
| toGMTString() | Deprecated - Converts a date to a string, using the Internet GMT conventions. Use toUTCString instead. |
| toLocaleDateString() | Returns the "date" portion of the Date as a string, using the current locale's conventions. |
| toLocaleFormat() | Converts a date to a string, using a format string. |
| toLocaleString() | Converts a date to a string, using the current locale's conventions. |
| toLocaleTimeString() | Returns the "time" portion of the Date as a string, using the current locale's conventions. |
| toSource() | Returns a string representing the source for an equivalent Date object; you can use this value to create a new object. |
| toString() | Returns a string representing the specified Date object. |
| toTimeString() | Returns the "time" portion of the Date as a human-readable string. |
| toUTCString() | Converts a date to a string, using the universal time convention. |
| valueOf() | Returns the primitive value of a Date object. |

In the following sections, we will have a few examples to demonstrate the usage of Date methods.

**Date()**

Javascript **Date()** method returns today's date and time and does not need any object to be called.

Syntax

Its syntax is as follows:

Date()

Return Value

Returns today's date and time.

Example

Try the following example.

```
<html>
<head>
<title>JavaScript Date Method</title>
</head>
<body>
<script type="text/javascript">
var dt = Date();
document.write("Date and Time : " + dt );
</script>
</body>
</html>
```

Output

Date and Time : Wed Mar 25 2015 15:00:57 GMT+0530 (India Standard Time)

getDate()

Javascript date **getDate()** method returns the day of the month for the specified date according to local time. The value returned by **getDate** is an integer between 1 and 31.

Syntax

Its syntax is as follows:

Date.getDate()

Return Value

Returns today's date and time.

Example

Try the following example.

```
<html>
<head>
<title>JavaScript getDate Method</title>
</head>
<body>
<script type="text/javascript">
var dt = new Date("December 25, 1995 23:15:00");
document.write("getDate() : " + dt.getDate() );
</script>
</body>
</html>
```

Output

```
getDate() : 25
```

getDay()

Javascript date **getDay()** method returns the day of the week for the specified date according to local time. The value returned by **getDay** is an integer corresponding to the day of the week: 0 for Sunday, 1 for Monday, 2 for Tuesday, and so on.

Syntax

Its syntax is as follows:

```
Date.getDay()
```

Return Value

Returns the day of the week for the specified date according to local time.

Example

Try the following example.

```
<html>
<head>
<title>JavaScript getDay Method</title>
</head>
<body>
<script type="text/javascript">
var dt = new Date("December 25, 1995 23:15:00");
```

```
document.write("getDay() : " + dt.getDay() );
</script>
</body>
</html>
```

Output

getDay() : 1

**getFullYear()**

Javascript date **getFullYear()** method returns the year of the specified date according to local time. The value returned by **getFullYear** is an absolute number. For dates between the years 1000 and 9999, **getFullYear** returns a four-digit number, for example, 2008.

Syntax

Its syntax is as follows:

Date.getFullYear()

Return Value

Returns the year of the specified date according to local time.

Example

Try the following example.

```
<html>
<head>
<title>JavaScript getFullYear Method</title>
</head>
<body>
<script type="text/javascript">
var dt = new Date("December 25, 1995 23:15:00");
document.write("getFullYear() : " + dt.getFullYear() );
</script>
</body>
</html>
```

Output

getFullYear() : 1995

getHours()

Javascript Date **getHours()** method returns the hour in the specified date according to local time. The value returned by **getHours** is an integer between 0 and 23.

**Syntax**

Its syntax is as follows:

Date.getHours()

Return Value

Returns the hour in the specified date according to local time.

Example

Try the following example.

```
<html>
<head>
<title>JavaScript getHours Method</title>
</head>
<body>
<script type="text/javascript">
var dt = new Date("December 25, 1995 23:15:00");
document.write("getHours() : " + dt.getHours() );
</script>
</body>
</html>
```

Output

getHours() : 23

**getMilliseconds()**

Javascript date **getMilliseconds()** method returns the milliseconds in the specified date according to local time. The value returned by **getMilliseconds** is a number between 0 and 999.

**Syntax**

Its syntax is as follows:

Date.getMilliseconds ()

Return Value

Returns the milliseconds in the specified date according to local time.

Example

Try the following example.

```
<html>
<head>
<title>JavaScript getMilliseconds Method</title>
</head>
<body>
<script type="text/javascript">
var dt = new Date();
document.write("getMilliseconds() : " + dt.getMilliseconds() );
</script>
</body>
</html>
```

**Output**

getMilliseconds() : 641

**getMinutes ()**

Javascript date **getMinutes()** method returns the minutes in the specified date according to local time. The value returned by **getMinutes** is an integer between 0 and 59.

Syntax

Its syntax is as follows:

Date.getMinutes ()

Return Value

Returns the minutes in the specified date according to local time.

Example

Try the following example.

```
<html>
<head>
<title>JavaScript getMinutes Method</title>
</head>
<body>
<script type="text/javascript">
var dt = new Date( "December 25, 1995 23:15:00" );
document.write("getMinutes() : " + dt.getMinutes() );
```

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

Output

getMinutes() : 15

**getMonth ()**

Javascript date **getMonth()** method returns the month in the specified date according to local time. The value returned by **getMonth** is an integer between 0 and 11. 0 corresponds to January, 1 to February, and so on.

Syntax

Its syntax is as follows:

Date.getMonth ()

Return Value

Returns the Month in the specified date according to local time.

Example

Try the following example.

```
<html>
<head>
<title>JavaScript getMonth Method</title>
</head>
<body>
<script type="text/javascript">
var dt = new Date( "December 25, 1995 23:15:00" );
document.write("getMonth() : " + dt.getMonth() );
</script>
</body>
</html>
```

**Output**

getMonth() : 11

**getSeconds ()**

Javascript date **getSeconds()** method returns the seconds in the specified date according to local time. The value returned by **getSeconds** is an integer between 0 and 59.

Syntax

Its syntax is as follows:

Date.getSeconds ()

Return Value

Returns the seconds in the specified date according to local time.

**Example**

Try the following example.

```
<html>
<head>
<title>JavaScript getSeconds Method</title>
</head>
<body>
<script type="text/javascript">
var dt = new Date( "December 25, 1995 23:15:20" );
document.write("getSeconds() : " + dt.getSeconds() );
</script>
</body>
</html>
```

**Output**

getSeconds () : 20

**getTime ()**

Javascript date **getTime()** method returns the numeric value corresponding to the time for the specified date according to universal time. The value returned

by the **getTime** method is the number of milliseconds since 1 January 1970 00:00:00.

You can use this method to help assign a date and time to another Date object.

## Syntax

Its syntax is as follows:

Date.getTime ()

## Return Value

Returns the numeric value corresponding to the time for the specified date according to universal time.

## Example

Try the following example.

```
<html>
<head>
<title>JavaScript getTime Method</title>
</head>
<body>
<script type="text/javascript">
var dt = new Date( "December 25, 1995 23:15:20" );
document.write("getTime() : " + dt.getTime() );
</script>
</body>
</html>
```

**Output**

getTime() : 819913520000

**getUTCFullYear ()**

Javascript date **getUTCFullYear()** method returns the year in the specified date according to universal time. The value returned by **getUTCFullYear** is an absolute number that is compliant with year-2000, for example, 2008.

**Syntax**

Its syntax is as follows:

Date.getUTCFullYear ()

Return Value

Returns the year in the specified date according to universal time.

**Example**

Try the following example.

```
<html>
<head>
<title>JavaScript getUTCFullYear Method</title>
</head>

<body>
```

```
<script type="text/javascript">
var dt = new Date( "December 25, 1995 23:15:20" );
document.write("getUTCFullYear() : " + dt.getUTCFullYear() );
</script>
</body>
</html>
```

**Output**

getUTCFullYear() : 1995

setDate ()

Javascript date **setDate()** method sets the day of the month for a specified date according to local time.

Syntax

Its syntax is as follows:

Date.setDate( dayValue )

Parameter Detail

**dayValue** : An integer from 1 to 31, representing the day of the month.

**Example**

Try the following example.

```
<html>
<head>
<title>JavaScript setDate Method</title>
</head>
<body>
<script type="text/javascript">
var dt = new Date( "Aug 28, 2008 23:30:00" );
dt.setDate( 24 );
document.write( dt );
</script>
</body>
</html>
```

**Output**

Sun Aug 24 2008 23:30:00 GMT+0530 (India Standard Time)

toLocaleDateString ()

Javascript date **toLocaleDateString()** method converts a date to a string, returning the "date" portion using the operating system's locale's conventions.

**Syntax**

Its syntax is as follows:

Date.toGMTString()

Return Value

Returns a date to a string, using Internet GMT conventioins.

**Example**

Try the following example.

```
<html>
<head>
<title>JavaScript toGMTString Method</title>
</head>
<body>
<script type="text/javascript">
var dt = new Date(1993, 6, 28, 14, 39, 7);
document.write( "Formated Date : " + dt.toGMTString() );
</script>
</body>
</html>
```

Output

Formated Date : Wed, 28 Jul 1993 09:09:07 GMT

## Lesson 8

### Document Object(DOM )

Every web page resides inside a browser window which can be considered as an object. A Document object represents the HTML document that is displayed in that window. The Document object has various properties that refer to other objects which allow access to and modification of document content.

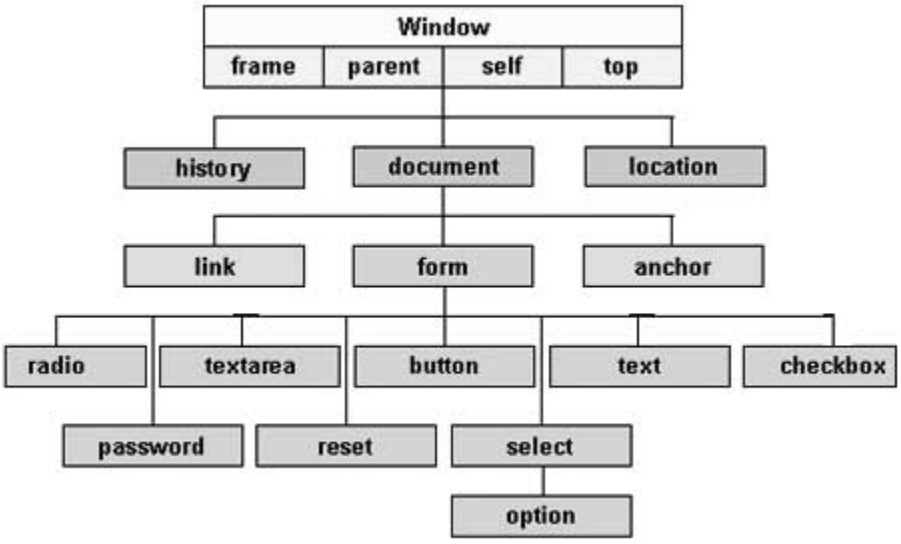
The way a document content is accessed and modified is called the **Document Object Model**, or **DOM**. The Objects are organized in a hierarchy. This hierarchical structure applies to the organization of objects in a Web document.

- **Window object:** Top of the hierarchy. It is the outmost element of the object hierarchy.
- **Document object:** Each HTML document that gets loaded into a window

becomes a document object. The document contains the contents of the page.

- **Form object:** Everything enclosed in the <form>...</form> tags sets the form object.
- **Form control elements:** The form object contains all the elements defined for that object such as text fields, buttons, radio buttons, and checkboxes.

Here is a simple hierarchy of a few important objects:




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

- **The Legacy DOM**: This is the model which was introduced in early versions of JavaScript language. It is well supported by all browsers, but allows access only to certain key portions of documents, such as forms, form elements, and images.
- **The W3C DOM**: This document object model allows access and modification of all document content and is standardized by the World Wide Web Consortium (W3C). This model is supported by almost all the modern browsers.
- **The IE4 DOM**: This document object model was introduced in Version 4 of

Microsoft's Internet Explorer browser. IE 5 and later versions include support for most basic W3C DOM features.

**The Legacy DOM**

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

This model provides several read-only properties, such as title, URL, and lastModified provide information about the document as a whole. Apart from that, there are various methods provided by this model which can be used to set and get document property values.

**Cument Properties in Legacy DOM**

Here is a list of the document properties which can be accessed using Legacy DOM.

| S.No | Property and Description |
|---|---|
| 1 | alinkColor<br>Deprecated - A string that specifies the color of activated links.<br>Ex: document.alinkColor |
| 2 | anchors[ ]<br>An array of Anchor objects, one for each anchor that appears in the document |
| 3 | applets[ ]<br>An array of Applet objects, one for each applet that appears in the document<br>Ex: document.applets[0], document.applets[1] and so on |
| 4 | bgColor<br>Deprecated - A string that specifies the background color of the document.<br>Ex: document.bgColor |
| 5 | Cookie<br>A string valued property with special behavior that allows the cookies associated with this document to be queried and set.<br>Ex: document.cookie |
| 6 | Domain<br>A string that specifies the Internet domain the document is from. Used for security purpose.<br>Ex: document.domain |

| | |
|---|---|
| 7 | embeds[ ]<br>An array of objects that represent data embedded in the document with the <embed> tag. A synonym for plugins []. Some plugins and ActiveX controls can be controlled with JavaScript code.<br>Ex: document.embeds[0], document.embeds[1] and so on |
| 8 | fgColor<br>A string that specifies the default text color for the document Ex: document.fgColor |
| 9 | forms[ ]<br>An array of Form objects, one for each HTML form that appears in the document.<br>Ex: document.forms[0], document.forms[1] and so on |
| 10 | images[ ]<br>An array of Image objects, one for each image that is embedded in the document with the HTML <img> tag.<br>Ex: document.images[0], document.images[1] and so on |
| 11 | lastModified<br>A read-only string that specifies the date of the most recent change to the document<br>Ex: document.lastModified |
| 12 | linkColor<br>Deprecated - A string that specifies the color of unvisited links<br>Ex: document.linkColor |
| 13 | links[ ]<br>It is a document link array.<br>Ex: document.links[0], document.links[1] and so on |
| 14 | Location<br>The URL of the document. Deprecated in favor of the URL property.<br>Ex: document.location |
| 15 | plugins[ ]<br>A synonym for the embeds[ ] |
| 16 | Referrer<br>A read-only string that contains the URL of the document, if any, from which the current document was linked.<br>Ex: document.referrer |
| 17 | Title<br>The text contents of the <title> tag. |

| | |
|---|---|
| | Ex: document.title |
| 18 | URL<br>A read-only string that specifies the URL of the document.<br>Ex: document.URL |
| 19 | vlinkColor<br>Deprecated - A string that specifies the color of visited links.<br>Ex: document.vlinkColor |

**Document Methods in Legacy DOM**

Here is a list of methods supported by Legacy DOM.

| S.No | Property and Description |
|---|---|
| 1 | clear( )<br>Deprecated - Erases the contents of the document and returns nothing.<br>Ex: document.clear( ) |
| 2 | close( )<br>Closes a document stream opened with the open( ) method and returns nothing. |
| 3 | open( )<br>Deletes existing document content and opens a stream to which new document contents may be written. Returns nothing.<br>Ex: document.open( ) |
| 4 | write( value, ...)<br>Inserts the specified string or strings into the document currently being parsed or appends to document opened with open( ). Returns nothing.<br>Ex: document.write( value, ...) |
| 5 | writeln( value, ...)<br>Identical to write( ), except that it appends a newline character to the output. Returns nothing.<br>Ex: document.writeln( value, ...) |

**Example**

We can locate any HTML element within any HTML document using HTML DOM. For instance, if a web document contains a **form** element, then using JavaScript, we can refer to it as **document.forms[0]**. If your Web document includes two **form** elements, the first form is referred to as document.forms[0] and the second as document.forms[1].

Using the hierarchy and properties given above, we can access the first form element using **document.forms[0].elements[0]** and so on.

Here is an example to access document properties using Legacy DOM method.

```
<html>
<head>
<title> Document Title </title>
<script type="text/javascript">
<!--
function myFunc()
{
var ret = document.title;
alert("Document Title : " + ret );
var ret = document.URL;
alert("Document URL : " + ret );
var ret = document.forms[0];
alert("Document First Form : " + ret );
var ret = document.forms[0].elements[1];
alert("Second element : " + ret );
}
//-->
</script>
</head>
<body>
<h1 id="title">This is main title</h1>
<p>Click the following to see the result:</p>
<form name="FirstForm">
<input type="button" value="Click Me" onclick="myFunc();" />
<input type="button" value="Cancel">
</form>
<form name="SecondForm">
<input type="button" value="Don't ClickMe"/>
</form>
</body>
</html>
```

Output

**This is main title**

Click the following to see the result:


**NOTE:** This example returns objects for forms and elements and we would have to access their values by using those object properties which are not discussed in this tutorial.

### The W3C DOM

This document object model allows access and modification of all document content and is standardized by the World Wide Web Consortium (W3C). This model is supported by almost all the modern browsers.

The W3C DOM standardizes most of the features of the legacy DOM and adds new ones as well. In addition to supporting forms[ ], images[ ], and other array properties of the Document object, it defines methods that allow scripts to access and manipulate any document element and not just special-purpose elements like forms and images.

### Document Properties in W3C DOM

This model supports all the properties available in Legacy DOM. Additionally, here is a list of document properties which can be accessed using W3C DOM.

| S.No | Property and Description |
|---|---|
| 1 | Body<br>A reference to the Element object that represents the <body> tag of this document.<br>Ex: document.body |
| 2 | defaultView<br>It is a read-only property and represents the window in which the document is displayed.<br>Ex: document.defaultView |

| | |
|---|---|
| 3 | documentElement<br>A read-only reference to the <html> tag of the document.<br>Ex: document.documentElement8/31/2008 |
| 4 | Implementation<br>It is a read-only property and represents the DOMImplementation object that represents the implementation that created this document.<br>Ex: document.implementation |

**Document Methods in W3C DOM**

This model supports all the methods available in Legacy DOM. Additionally, here is a list of methods supported by W3C DOM.

| S.No | Property and Description |
|---|---|
| 1 | createAttribute( name)<br>Returns a newly-created Attr node with the specified name.<br>Ex: document.createAttribute( name) |
| 2 | createComment( text)<br>Creates and returns a new Comment node containing the specified text.<br>Ex: document.createComment( text) |
| 3 | createDocumentFragment( )<br>Creates and returns an empty DocumentFragment node.<br>Ex: document.createDocumentFragment( ) |
| 4 | createElement( tagName)<br>Creates and returns a new Element node with the specified tag name.<br>Ex: document.createElement( tagName) |
| 5 | createTextNode( text)<br>Creates and returns a new Text node that contains the specified text.<br>Ex: document.createTextNode( text) |
| 6 | getElementById( id)<br>Returns the Element of this document that has the specified value for its id attribute, or null if no such Element exists in the document.<br>Ex: document.getElementById( id) |
| 7 | getElementsByName( name)<br>Returns an array of nodes of all elements in the document that have a specified value for their name attribute. If no such elements are found, returns a zero-length array.<br>Ex: document.getElementsByName( name) |

| | |
|---|---|
| 8 | getElementsByTagName( tagname)<br>Returns an array of all Element nodes in this document that have the specified tag name. The Element nodes appear in the returned array in the same order they appear in the document source.<br>Ex: document.getElementsByTagName( tagname) |
| 9 | importNode( importedNode, deep)<br>Creates and returns a copy of a node from some other document that is suitable for insertion into this document. If the deep argument is true, it recursively copies the children of the node too. Supported in DOM Version 2<br>Ex: document.importNode( importedNode, deep) |

**Example**

This is very easy to manipulate ( Accessing and Setting ) document element using W3C DOM. You can use any of the methods like **getElementById**, **getElementsByName**, or **getElementsByTagName**.

Here is an example to access document properties using W3C DOM method.

```
<html>
<head>
<title> Document Title </title>
<script type="text/javascript">
<!--
function myFunc()
{
var ret = document.getElementsByTagName("title");
alert("Document Title : " + ret[0].text );
var ret = document.getElementById("heading");
alert("Document URL : " + ret.innerHTML );
}
//-->
</script>
</head>
<body>
```

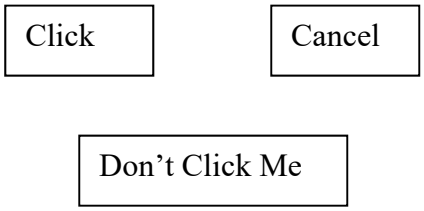
```
<h1 id="heading">This is main title</h1>
<p>Click the following to see the result:</p>
<form id="form1" name="FirstForm">
<input type="button" value="Click Me" onclick="myFunc();" />
<input type="button" value="Cancel">
</form>
<form d="form2" name="SecondForm">
<input type="button" value="Don't ClickMe"/>
</form>
</body>
</html>
```

NOTE: This example returns objects for forms and elements and we would have to access their values by using those object properties which are not discussed in this tutorial.

Output

**This is main title**

Click the following to see the result:




**The Window Object**

The **window** object is supported by all browsers. It represents the browser's window.

All global JavaScript objects, functions, and variables automatically become members of the window object.

Global variables are properties of the window object.

Global functions are methods of the window object.

Even the document object (of the HTML DOM) is a property of the window object:

```
window.document.getElementById("header");
```

is the same as:

document.getElementById("header");

**Window Size**

Two properties can be used to determine the size of the browser window.

Both properties return the sizes in pixels:

- window.innerHeight - the inner height of the browser window (in pixels)
- window.innerWidth - the inner width of the browser window (in pixels)

For Internet Explorer 8, 7, 6, 5:

- document.documentElement.clientHeight
- document.documentElement.clientWidth

or

- document.body.clientHeight
- document.body.clientWidth

A practical JavaScript solution (covering all browsers):

**Example**

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

<p id="demo"></p>

<script>
var w = window.innerWidth
|| document.documentElement.clientWidth
|| document.body.clientWidth;

var h = window.innerHeight
|| document.documentElement.clientHeight
|| document.body.clientHeight;

var x = document.getElementById("demo");
x.innerHTML = "Browser inner window width: " + w + ", height: " + h + ".";
</script>

</body>
</html>
```

Output
Browser inner window width: 668, height: 518.

**Other Window Methods**

Some other methods:

- window.open() - open a new window
- window.close() - close the current window
- window.moveTo() -move the current window
- window.resizeTo() -resize the current window

**JavaScript** Window Screen

The window.screen object contains information about the user's screen.

**Window Screen**

The **window.screen** object can be written without the window prefix.

**Properties:**

- screen.width
- screen.height
- screen.availWidth
- screen.availHeight
- screen.colorDepth
- screen.pixelDepth

**Window Screen Width**

The screen.width property returns the width of the visitor's screen in pixels.

Example
Display the width of the screen in pixels:

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

<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML =
"Screen width is " + screen.width;
</script>

</body>
</html>
```

Output
Screen width is 1366

**Window Screen Height**

The screen.height property returns the height of the visitor's screen in pixels.

Example
Display the height of the screen in pixels:

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

<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML =
"Screen height is " + screen.height;
</script>

</body>
</html>
```

Output
Screen height is 768

**Window Screen Available Width**

The screen.availWidth property returns the width of the visitor's screen, in pixels, minus interface features like the Windows Taskbar.

Example
Display the available width of the screen in pixels:

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

<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML =
"Available screen width is " + screen.availWidth;
</script>


</body>
</html>
```

Output
Available screen width is 1366

**Window Screen Color Depth**

The screen.colorDepth property returns the number of bits used to display one color.

All modern computers use 24 bit or 32 bit hardware for color resolution:

- 24 bits = 16,777,216 different "True Colors"
- 32 bits = 4,294,967,296 different "Deep Colors"

Older computers used 16 bits: 65,536 different "High Colors" resolution.

Very old computers, and old cell phones used 8 bits: 256 different "VGA colors".

**Example**

Display the color depth of the screen in bits:

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

<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML =
"Screen color depth is " + screen.colorDepth;
</script>

</body>
</html>
```

Output

Screen color depth is 24

**Window Screen Pixel Depth**

The screen.pixelDepth property returns the pixel depth of the screen.

**Example**

Display the pixel depth of the screen in bits:

```
<!DOCTYPE html>
```

```
<html>
<body>
<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML =
"Screen pixel depth is " + screen.pixelDepth;
</script>

</body>
</html>
```

Output

Screen pixel depth is 24

The window.location object can be used to get the current page address (URL) and to redirect the broser to a new page.

**Window Location**

The **window.location** object can be written without the window prefix.

Some examples:

- window.location.href returns the href (URL) of the current page
- window.location.hostname returns the domain name of the web host
- window.location.pathname returns the path and filename of the current page
- window.location.protocol returns the web protocol used (http:// or https://)
- window.location.assign loads a new document

**Window Location Href**

The **window.location.href** property returns the URL of the current page.

Example

Display the href (URL) of the current page:

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

```
<body>

<p>Display the entire URL of the current page.</p>

<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML =
"Page location is: " + window.location.href;
</script>

</body>
</html>
```

Output

Display the entire URL of the current page.

Page location is:

http://www.w3schools.com/js/tryit.asp?filename=tryjs_loc_href

## Window status Property

### Definition and Usage

The status property sets the text in the status bar at the bottom of the browser, or returns the previously set text.

**Note:** Do not confuse this property with the defaultStatus property, which specifies the default text displayed in the status bar.

**Note:** The status property does not work in the default configuration of IE, Firefox, Chrome, Safari or Opera 15 and newer. To allow scripts to change the text of the status, the user must set the dom.disable_window_status_change preference to false in the about:config screen. (or in Firefox: "Tools - Options - Content -Enable JavaScript / Advanced - Allow scripts to change status bar text").

**Example**

Set the text in the status bar:

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

<p>Look at the text in the statusbar.</p>
<p><b>Note:</b> This property does not work in default configuration of IE, Firefox, Chrome, Safari or Opera 15+.</p>

<script>
window.status = "Some text in the status bar!!";
</script>

</body>
</html>
```

Output

Look at the text in the statusbar.

***Note:*** *This property does not work in default configuration of IE, Firefox, Chrome, Safari or Opera 15+.*

**Syntax**

**window.status**

Enable Status Bar in browser.

Most (newer) major browsers disable status bar messages by default. If your status bar doesn't change when you hover over the link, it's probably because of this.

If you really want to see this example, you can enable status bar messages by changing your browser settings.

**For example, in Firefox:**

1) Go to Tools > Options
2) Click the *Content* tab
3) Ensure that the *JavaScript* option is checked
4) Click *Advanced* (next to the *Enable JavaScript* option)
5) Check the Change status bar text option

6) Click *OK* to save this screen
7) Click *OK* again

**In Internet Explorer:**

1) Go to Tools > Internet Options
2) Click the *Security* tab
3) Ensure that the *Internet* option is selected/highlighted
4) Click *Custom Level...* (this launches the security settings for the Internet zone)
5) Scroll down until you see *Allow status bar updates via script* (under the *Scripting* option). Click *Enable*
6) Click *OK* to save this screen
7) Click *OK* again

*Note that, because this is the default setting in most browsers, there's a good chance that most of your users won't see your status bar message.*

## DIALOG BOX

JavaScript supports three important types of dialog boxes. These dialog boxes can be used to raise and alert, or to get confirmation on any input or to have a kind of input from the users. Here we will discuss each dialog box one by one.

### Alert Dialog Box

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

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

**Example**

```
<html>
<head>
<script type="text/javascript">
<!--
```

```
function Warn() {
alert ("This is a warning message!");
document.write ("This is a warning message!");
}
//-->
</script>
</head>
<body>
<p>Click the following button to see the result: </p>
<form>
<input type="button" value="Click Me" onclick="Warn();" />
</form>
</body>
</html>
```

Output

Click the following button to see the result:


**Confirmation Dialog Box**

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

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

Example

```
<html>
<head>
<script type="text/javascript">
<!--
function getConfirmation(){
var retVal = confirm("Do you want to continue ?");
if( retVal == true ){
document.write ("User wants to continue!");
```

```
return true;
}else{
Document.write ("User does not want to continue!");
return false;
}
}
//-->
</script>
</head>
<body>
<p>Click the following button to see the result: </p>
<form>
<input type="button" value="Click Me" onclick="getConfirmation();" />
</form>
</body>
</html>
```

Output

Click the following button to see the result:


## LESSON 9

### Prompt Dialog Box

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

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

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

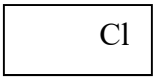
**Example**

The following example shows how to use a prompt dialog box:

```
<html>
<head>
<script type="text/javascript">
<!--
function getValue(){
var retVal = prompt("Enter your name : ", "your name here");
document.write("You have entered : " + retVal);
}
</script>
</head>
<body>
<p>Click the following button to see the result: </p>
<form>
<input type="button" value="Click Me" onclick="getValue();" />
</form>
</body>
</html>
```

Output

Click the following button to see the result:




**Definition and Usage**

The open() method opens a new browser window.

Use the close() method to close the window.

Example

Open "www.facebook.com" in a new browser window:

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

<p>Click the button to open a new browser window.</p>

<button onclick="myFunction()">Try it</button>
```

```
<script>
function myFunction() {
   window.open("http://www.facebook.com");
}
</script>

</body>
</html>
```

Click the button to open a new browser window.

Try it

**Syntax**

window.open(URL,name,specs,replace)

**Parameter Values**

| Parameter | Description |
|---|---|
| URL | Optional. Specifies the URL of the page to open. If no URL is specified, a new window with about:blank is opened |
| name | Optional. Specifies the target attribute or the name of the window. The following values are supported:<br>_blank - URL is loaded into a new window. This is default<br>_parent - URL is loaded into the parent frame<br>_self - URL replaces the current page<br>_top - URL replaces any framesets that may be loaded<br>name - The name of the window (Note: the name does not specify the title of the new window) |
| specs | Optional. A comma-separated list of items, no whitespaces. The following values are supported:<br>channelmode=yes\|no\|1\|0 — Whether or not to display the window in theater mode. Default is no. IE only<br>directories=yes\|no\|1\|0 — Obsolete. Whether or not to add directory buttons. Default is yes. IE only<br>fullscreen=yes\|n — Whether or not to display the browser in full- |

| | | |
|---|---|---|
| | o\|1\|0 | screen mode. Default is no. A window in full-screen mode must also be in theater mode. IE only |
| | height=pixels | The height of the window. Min. value is 100 |
| | left=pixels | The left position of the window. Negative values not allowed |
| | location=yes\|no\|1\|0 | Whether or not to display the address field. Opera only |
| | menubar=yes\|no\|1\|0 | Whether or not to display the menu bar |
| | resizable=yes\|no\|1\|0 | Whether or not the window is resizable. IE only |
| | scrollbars=yes\|no\|1\|0 | Whether or not to display scroll bars. IE, Firefox & Opera only |
| | status=yes\|no\|1\|0 | Whether or not to add a status bar |
| | titlebar=yes\|no\|1\|0 | Whether or not to display the title bar. Ignored unless the calling application is an HTML Application or a trusted dialog box |
| | toolbar=yes\|no\|1\|0 | Whether or not to display the browser toolbar. IE and Firefox only |
| | top=pixels | The top position of the window. Negative values not allowed |
| | width=pixels | The width of the window. Min. value is 100 |
| replace | Optional. Specifies whether the URL creates a new entry or replaces the current entry in the history list. The following values are supported:<br>true - URL replaces the current document in the history list<br>false - URL creates a new entry in the history list | |

**Example**

Open an about:blank page in a new window:

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

<p>Click the button to open an about:blank page in a new browser window that is 200px wide and 100px tall.</p>

<button onclick="myFunction()">Try it</button>

<script>
function myFunction() {
   var myWindow = window.open("", "", "width=200,height=100");
}
</script>

</body>
</html>
```

Try it Click the button to open an about:blank page in a new browser window that is 200px wide and 100px tall.

**Example**

Open a new window called "MsgWindow", and write some text into it:

```
var myWindow =
window.open("", "MsgWindow", "width=200,height=100");
myWindow.document.write("<p>This is 'MsgWindow'. I am 200px wide and 100px tall!</p>");
```

**Example**

Replace the current window with a new window:

```
var myWindow = window.open("", "_self");
myWindow.document.write("<p>I replaced the current window.</p>");
```

**Example**

Open a new window and control its appearance:

```
window.open("http://www.w3schools.com", "_blank","toolbar=yes,scrollbars
=yes,resizable=yes,top=500,left=500,width=400,height=400");
```

**Example**

Open multiple windows:

```
window.open("http://www.google.com/");
window.open("http://www.w3schools.com/");
```

Example

Open a new window. Use close() to close the new window:

```
function openWin() {
   myWindow =
window.open("", "myWindow", "width=200,height=100");   // Opens a new
window
}

function closeWin() {
   myWindow.close();   // Closes the new window
}
```

## LESSON 10

### EVENTS

What is an Event?

JavaScript's interaction with HTML is handled through events that occur when the user or the browser manipulates a page.

When the page loads, it is called an event. When the user clicks a button, that click too is an event. Other examples include events like pressing any key, closing a window, resizing a window, etc.

Developers can use these events to execute JavaScript coded responses, which cause buttons to close windows, messages to be displayed to users, data to be validated, and virtually any other type of response imaginable.

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

Please go through this small tutorial for a better understanding HTML Event Reference. Here we will see a few examples to understand the relation between Event and JavaScript.

onclick Event Type

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

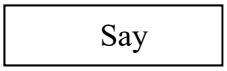
**Example**

Try the following example.

```
<html>
<head>
<script type="text/javascript">
<!--
function sayHello() {
document.write ("Hello World")
}
//-->
</script>
</head>
<body>
<p> Click the following button and see result</p>
<input type="button" onclick="sayHello()" value="Say Hello" />
</body>
</html>
```

Output

Click the following button and see result




**onsubmitEvent Type**

**onsubmit** is an event that occurs when you try to submit a form. You can put your form validation against this event type.

**Example**

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

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

**onmouseover and onmouseout**

These two event types will help you create nice effects with images or even with text as well. The **onmouseover** event triggers when you bring your mouse over any element and the **onmouseout** triggers when you move your mouse out from that element. Try the following example.

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

```
<!--
function over() {
document.write ("Mouse Over");
}
function out() {
document.write ("Mouse Out");
}
//-->
</script>
</head>
<body>
<p>Bring your mouse inside the division to see the result:</p>
<div onmouseover="over()" onmouseout="out()">
<h2> This is inside the division </h2>
</div>
</body>
</html>
```

Output

Bring your mouse inside the division to see the result:

This is inside the division

**HTML 5Standard Events**

The standard HTML 5 events are listed here for your reference. Here script indicates a Javascript function to be executed against that event.

| Attribute | Value | Description |
|---|---|---|
| Offline | script | Triggers when the document goes offline |
| Onabort | script | Triggers on an abort event |
| onafterprint | script | Triggers after the document is printed |
| onbeforeonload | script | Triggers before the document loads |
| onbeforeprint | script | Triggers before the document is printed |
| onblur | script | Triggers when the window loses focus |

| | | |
|---|---|---|
| oncanplay | script | Triggers when media can start play, but might has to stop for buffering |
| oncanplaythrough | script | Triggers when media can be played to the end, without stopping for buffering |
| onchange | script | Triggers when an element changes |
| onclick | script | Triggers on a mouse click |
| oncontextmenu | script | Triggers when a context menu is triggered |
| ondblclick | script | Triggers on a mouse double-click |
| ondrag | script | Triggers when an element is dragged |
| ondragend | script | Triggers at the end of a drag operation |
| ondragenter | script | Triggers when an element has been dragged to a valid drop target |
| ondragleave | script | Triggers when an element leaves a valid drop target |
| ondragover | script | Triggers when an element is being dragged over a valid drop target |
| ondragstart | script | Triggers at the start of a drag operation |
| ondrop | script | Triggers when dragged element is being dropped |
| ondurationchange | script | Triggers when the length of the media is changed |
| onemptied | script | Triggers when a media resource element suddenly becomes empty. |
| onended | script | Triggers when media has reach the end |
| onerror | script | Triggers when an error occur |
| onfocus | script | Triggers when the window gets focus |
| onformchange | script | Triggers when a form changes |
| onforminput | script | Triggers when a form gets user input |
| onhaschange | script | Triggers when the document has change |
| oninput | script | Triggers when an element gets user input |

| | | |
|---|---|---|
| oninvalid | script | Triggers when an element is invalid |
| onkeydown | script | Triggers when a key is pressed |
| onkeypress | script | Triggers when a key is pressed and released |
| onkeyup | script | Triggers when a key is released |
| onload | script | Triggers when the document loads |
| onloadeddata | script | Triggers when media data is loaded |
| onloadedmetadata | script | Triggers when the duration and other media data of a media element is loaded |
| onloadstart | script | Triggers when the browser starts to load the media data |
| onmessage | script | Triggers when the message is triggered |
| onmousedown | script | Triggers when a mouse button is pressed |
| onmousemove | script | Triggers when the mouse pointer moves |
| onmouseout | script | Triggers when the mouse pointer moves out of an element |
| onmouseover | script | Triggers when the mouse pointer moves over an element |
| onmouseup | script | Triggers when a mouse button is released |
| onmousewheel | script | Triggers when the mouse wheel is being rotated |
| onoffline | script | Triggers when the document goes offline |
| onoine | script | Triggers when the document comes online |
| ononline | script | Triggers when the document comes online |
| onpagehide | script | Triggers when the window is hidden |
| onpageshow | script | Triggers when the window becomes visible |
| onpause | script | Triggers when media data is paused |
| onplay | script | Triggers when media data is going to start playing |
| onplaying | script | Triggers when media data has start playing |

| | | |
|---|---|---|
| onpopstate | script | Triggers when the window's history changes |
| onprogress | script | Triggers when the browser is fetching the media data |
| onratechange | script | Triggers when the media data's playing rate has changed |
| onreadystatechange | script | Triggers when the ready-state changes |
| onredo | script | Triggers when the document performs a redo |
| onresize | script | Triggers when the window is resized |
| onscroll | script | Triggers when an element's scrollbar is being scrolled |
| onseeked | script | Triggers when a media element's seeking attribute is no longer true, and the seeking has ended |
| onseeking | script | Triggers when a media element's seeking attribute is true, and the seeking has begun |
| onselect | script | Triggers when an element is selected |
| onstalled | script | Triggers when there is an error in fetching media data |
| onstorage | script | Triggers when a document loads |
| onsubmit | script | Triggers when a form is submitted |
| onsuspend | script | Triggers when the browser has been fetching media data, but stopped before the entire media file was fetched |
| ontimeupdate | script | Triggers when media changes its playing position |
| onundo | script | Triggers when a document performs an undo |
| onunload | script | Triggers when the user leaves the document |
| onvolumechange | script | Triggers when media changes the volume, also when volume is set to "mute" |
| onwaiting | script | Triggers when media has stopped playing, but is expected to resume |

**onload Event**

The onload event occurs when an object has been loaded.

onload is most often used within the <body> element to execute a script once a web page has completely loaded all content (including images, script files, CSS files, etc.).

The onload event can be used to check the visitor's browser type and browser version, and load the proper version of the web page based on the information.

The onload event can also be used to deal with cookies (see "More Examples" below).

**Example**

Execute a JavaScript immediately after a page has been loaded:

```
<!DOCTYPE html>
<html>
<body onload="myFunction()">

<h1>Hello World!</h1>

<script>
function myFunction() {
   alert("Page is loaded");
}
</script>

</body>
</html>
```

Output

**Hello World!**

**onunload Event**

**Definition and Usage**

The onunload event occurs once a page has unloaded (or the browser window has been closed).

onunload occurs when the user navigates away from the page (by clicking on a link, submitting a form, closing the browser window, etc.).

Note: The onunload event is also triggered when a user reloads the page (and the onload event).

**Example**

Execute a JavaScript when a user unloads the document:

```
<!DOCTYPE html>
<html>
<body onunload="myFunction()">

<h1>Welcome to my Home Page</h1>

<p>Close this window or press F5 to reload the page.</p>
<p><strong>Note:</strong> Due to different browser settings, this event may not always work as expected.</p>

<script>
function myFunction() {
   alert("Thank you for visiting W3Schools!");
}
</script>

</body>
</html>
```

Output

**Welcome to my Home Page**

Close this window or press F5 to reload the page.

***Note:*** *Due to different browser settings, this event may not always work as expected.*

**Practical Javascript**

1. Write a JavaScript program that accept two integers and display the larger.

```
var num1, num2;
num1 = window.prompt("Input the First integer", "0");
num2 = window.prompt("Input the second integer", "0");

if(parseInt(num1, 10) > parseInt(num2, 10))
  {
  console.log("The larger of "+ num1+ " and "+ num2+ " is "+ num1+ ".");
  }
else
  if(parseInt(num2, 10) > parseInt(num1, 10))
  {
  console.log("The larger of "+ num1+" and "+ num2+ " is "+ num2+ ".");
  }
else
  {
   console.log("The values "+ num1+ " and "+num2+ " are equal.");
  }
```

2. Write a JavaScript conditional statement to find the sign of product of three numbers. Display an alert box with the specified sign.

```
var x=3;
var y=-7;
var z=2;
if (x>0 && y>0 && z>0)
{
      alert("The sign is +");
}
else if (x<0 && y<0 && z>0)
       {
         console.log("The sign is +");
       }
       else if (x>0 && y<0 && z<0)
       {
```

```
  console.log("The sign is +");
}
else if (x<0 && y>0 && z<0)
{
  console.log("The sign is +");
}
else
{
  console.log("The sign is -");
}
```

3. Write a JavaScript for loop that will iterate from 0 to 15. For each iteration, it will check if the current number is odd or even, and display a message to the screen.

Go to the editor

Sample Output :

"0 is even"

"1 is odd"

"2 is even"

```
for (var x=0; x<=15; x++) {
    if (x === 0) {
        console.log(x +  " is even");
    }
    else if (x % 2 === 0) {
        console.log(x + " is even");
    }
    else {
        console.log(x + " is odd");
    }
}
```

4. Write a JavaScript program which iterates the integers from 1 to 100. But for multiples of three print "Fizz" instead of the number and for the multiples of five print "Buzz". For numbers which are multiples of both three and five print "FizzBuzz".

```
for ( var i = 1; i <= 100; i++ )
{
 if ( i%3 === 0 && i%5 === 0 )
 {
  console.log( i + " FizzBuzz" );
 }
 else if ( i%3 === 0 )
 {
  console.log(i+ " Fizz" );
 }
 else if ( i%5 === 0 )
 {
  console.log(i+ " Buzz" );
 }
 else
 {
  console.log(i);
 }
}
```

5. Repeat all of above program by using while , and do….. while loop.
6. Use Break statement in above problem.
7. WAP to print all numbers from 1 to 100 which are divisible by 3 and 5 by using javascript.
8. WAP to enter random number in array and display them in asending order.
8. WAP to display all the prime numbers from 5 to 100
9. Print the contents of the current window.

Soln:

```
<!DOCTYPE html>
<html>
<head>
<meta charset=utf-8 />
<title>Print the current page.</title>
</head>
<script>
```

```
function print_current_page()
{
window.print();
}
</script>
<body>
<p></p>
<p>Click the button to print the current page.</p>
<button onclick="print_current_page()">Print this page</button>
</body>
</html>
```

**Explanation :**

window.print() : The window object represents a window containing a DOM document; the document property points to the DOM document loaded in that window, window.print() is used to open the Print Dialog to print the current document.

9. Write a JavaScript program to find the area of a triangle where lengths of the three of its sides are 5, 6, 7

```
var side1 = 5;
var side2 = 6;
var side3 = 7;
var perimeter = (side1 + side2 + side3)/2;
var area =  Math.sqrt(perimeter*((perimeter-side1)*(perimeter-
side2)*(perimeter-side3)));
console.log(area);
```

**Explanation** :

Calculate the area of a triangle of three given sides : In geometry, Heron's formula named after Hero of Alexandria, gives the area of a triangle by requiring no arbitrary choice of side as base or vertex as origin, contrary to other formulas for the area of a triangle, such as half the base times the height or half the norm of a cross product of two sides.

S= (a+b+c)/2

A= root(s(s-a)(s-b)(s-c)

10 Here is a sample html file with a submit button. Now modify the style of the paragraph text through javascript code

```
<!DOCTYPE html>
<html><br><head>
<meta charset=utf-8 />
<title>JS DOM paragraph style</title>
</head>
<body>
<p id ='text'>JavaScript Exercises - w3resource</p>
<div>
onclick="js_style()">Style</button>
</div>
</body>
</html>
<button id="jsstyle"
```

11 WAP which change the font, font size, and color of the paragraph text by clicking the buttons

Solution

```
!DOCTYPE html>
<html>
<head>
<meta charset=utf-8 />
<title>JS DOM paragraph style</title>
<script>
function js_style()
{
//font styles added by JS:
 text.style.fontSize = "14pt";
 text.style.fontFamily = "Comic Sans MS";
 text.style.color = "green";
}
</script>
</head>
<body>
```

```
<p id ='text'>JavaScript Exercises - w3resource</p>
<div>
<button id="jsstyle"
onclick="js_style()">Style</button>
</div>
</body>
</html>
```

12 Write a JavaScript program to set the background color of a paragraph

Soln

```
<!DOCTYPE html>
<html>
<head>
<meta charset=utf-8 />
<title>JS Bin</title>
</head>
<script>
function set_background() {
 docBody = document.getElementsByTagName("body")[0];
  //Get all the p elements that are descendants of the body
  myBodyElements = docBody.getElementsByTagName("p");
// get the first p elements
  myp1 = myBodyElements[0];
  myp1.style.background = "rgb(255,0,0)";
// get the second p elements
  myp2 = myBodyElements[1];
  myp2.style.background = "rgb(255,255,0)";
}
</script>
<body>
<input type="button" value="Click to set paragraph background color" onclic
k="set_background()">
<p>w3resource JavaScript Exercises</p>
<p>w3resource PHP Exercises</p>
</body>
</html>
```

13. Write a program to display a an alert box when click on a button.

Solution

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

<p>Click the button to display an alert box:</p>

<button onclick="myFunction()">Try it</button>

<script>
function myFunction() {
   alert("I am an alert box!");
}
</script>

</body>
</html>
```

14. WAP to create confirmation dialogue box.

Hence

```
bootbox.confirm("Are
you sure?",
function(result) {
```

| | |
|---|---|
| | Example.show("Confirm result: "+result); |
| | }); |

15. WAP to display prompt dialogue Box.

Hence

```
bootbox.prompt("What
is your name?",
function(result) {
                                    if (result === null) {
                                      Example.show("Prompt dismissed");
                                    } else {
```

```
            Example.show("Hi <b>"+result+"</b>");
          }
        });
```

16. WAP to display customs dialogue box by using javascript

Hence

```
bootbox.dialog({
                message: "I am a custom dialog",
                title: "Custom title",
                buttons: {
                  success: {
                    label: "Success!",
                    className: "btn-success",
                    callback: function() {
                      Example.show("great success");
                    }
                  },
                  danger: {
                    label: "Danger!",
                    className: "btn-danger",
                    callback: function() {
                      Example.show("uh oh, look out!");
                    }
                  },
                  main: {
                    label: "Click ME!",
                    className: "btn-primary",
                    callback: function() {
                      Example.show("Primary button");
                    }
                  }
                }
              });
```

**References:**

http://www.webteacher.com/javascript/
http://www.homeandlearn.co.uk/JS/javascript.html
http://www.tutorialspoint.com/javascript/javascript_overview.htm
http://www.w3schools.com
http://bootboxjs.com/examples.html

**Instruction for Teacher**

1. Use the Charts or Multimedia projector as far as possible to explain the content.
2. Assign the homework at the end of every class from the chapter so far covered during that class.
3. Conduct group discussions after finishing one lesson.
4. Do practical work of Javascript.
5. Make a lab sheet of different lessons for lab work

# UNIT 5

# XML( Extensible Markup Language)

## Introduction

XML stands for Extensible Markup Language, which became a W3C Recommendation on 10. February 1998. XML is a markup language which is like HTML. XML and HTML both use tags. But there are some differences between them: HTML was designed for how to display data. And XML was designed for how to store data. HTML tags are predefined. But XML tags are not predefined. You must define your own tags Before you learn XML, you should have a basic understanding of HTML. XML can be used to simplify data storage and sharing. With XML, data is separated from HTML. So you can create HTML layouts for displaying data. When the data changes, you don't have to recreate your HTML file. With XML, data can also be easily exchanged between computer and database systems, even they are incompatible in any other ways. Because XML data is stored in text format, this makes it easier to export data from a system to an XML file, and then import it into another system.

## Origins of XML

As the Web exploded in popularity and people all over the world began learning about HTML, they fairly quickly started running into the limitations outlined above. Heavy-metal SGML (Standard Generalized Markup Language) wonks, who had been working with SGML for years in relative obscurity, suddenly found that everyday people had some understanding of the concept of markup (that is, HTML). SGML experts began to consider the possibility of using SGML on the Web directly, instead of using just one application of it (again, HTML). At the same time, they knew that SGML, while powerful, was simply too complex for most people to use.

In the summer of 1996, Jon Bosak (currently online information technology architect at Sun Microsystems) convinced the W3C to let him form a committee on using SGML on the Web. He created a high-powered team of muckety-mucks from the SGML world. By November of that year, these folks had created the beginnings of

a simplified form of SGML that incorporated tried-and-true features of SGML but with reduced complexity. This was, and is, XML.

In March 1997, Bosak released his landmark paper, "XML, Java and the Future of the Web" (see Resources). Now, two years later (a very long time in the life of the Web), Bosak's short paper is still a good, if dated, introduction to why using XML is such an excellent idea. SGML was created for general document structuring, and HTML was created as an application of SGML for Web documents. XML is a simplification of SGML for general Web use.

## XML versus other technologies

**Xml** is based on SGML but different from HTML. Also, it offers certain benefits over EDI when we consider data sharing. So, let us compare XML with these technologies to derermine how dirrerent/ unique it is.

XML vs. SGML

**SGML** was the first generic markup language but it was complex and required overheads to use it. XML is based on SGML. It includes only the parts of SGML required for web publishing. XML did not include complex and optional features of SGML. XML retains the primary benefits that SGML offered, i.e XML, like SGML, offers following benefits;

a) XML is generalized markup language i.e., it allows creation of new tags/ tag-sets,
b) Documents describe the data they contain.
c) Documents can be validated.

## XML vs. HTML

XML and HTML are two different types of markup languages. While XML allows creation of new markup languages, HTML is a predefined markup language. XML and HTML can be thought of two different markup languages but complementary to one another. For example, you may use XML to structure and describe data and **HTML to format and display the data.**

| HTML | XML |
|---|---|
| HTML document formats and display web pages' data | XML documents carry data along with their description. |
| HTML tags are predefined. (predefined markup language) | XML tags are not predefined. You can create and define new tags as per your needs (meta language) |
| HTML tags may not have closing tag. | XML tags must have a closing tag. |
| HTML tags are not case-sensitive. | XML tars are case –sensitive. |
| HTML documents are directly viewable in a browser. | XML documents can be viewed only if proper stylesheet file is also available along with XML. |

## Advantages of XML

Apart from being an open source tool, and an easy to read format, XML has found wide usage, due to the following reasons. Most XML documents are written in plain text, makes it easier for developers, you can write it using a plain text editor or an IDE.

- Data identification is easier in XML documents, making it usable for all kinds of applications, so a search program can get the data, while an address book application can read only the address data.
- XML can be used to represent small amount of data as well as larger data sets making it more scalable.
- You can be using your customized stylesheet to display the XML data, or even using the standard XSL or XSLT.
- Documents can be added inline in an XML document, making it easier to interlink multiple documents.
- The tight definition of tags in XML makes it easier to process and parse, also the fact that every tag must be closed. So less chance of encountering a single tag like in HTML.
- Hierarchical structure of the elements makes it easier for you to drill down to the actual information. For e.g., if you are seeking the location of a specific department of an organization, you could just go down to the department element, and the location element in it.

**FEATURES OF XML**

Now that you have a brief idea about XML, let us quickly surf through the most common features of XML.

1. **XML was designed to carry data, not display data**

   XML was designed to carry data not display it. As it clarifies the fact that XML is not a replacement for HTML. XML were designed with different goals.

   XML was designed to describe data and to focus on what data is whereas HTML was designed to display data and to focus on how data looks.

2. **XML does not Do anything**

   XML was not designed to Do anything. Maybe it is a little hard to understand, but XML does not do anything. XML was created to structure, store and to send information.

   To make it behave in a particular way, additional files such as CSS files or XSL files are needed.

3. **XML is designed to be self-descriptive**

   XML tags are not predefined: rather these are to be created by the author to describe the content in appropriate manner. Therefore, each XML document is unique in itself and is able to describe its contents without any prior knowledge of tags etc.

4. **XML is free and extensible**

   XML tags are not predefined, rather, it is the author of the document who has to create or invent the new tags as per his/her needs. In other words, one can use XML to create own tag set e.g., to define questions in a question paper, your teacher may create a tag for defining questions and their corresponding answer.

A simple is being shown below:

```
<QuesionPaper>
<Question category=”MCQ”>
<title> Who is the author of Harry Potter series of books </title>
<optionA>A). Ruskin Bond </optionA>
<optionB>B). J.K. Rowling </optionB>
<optionC>C). Chetan Bhagat </optionC>
<optionD>D). Michael Alva </optionD>
<answer> B</answer>
</Question>
<Question category=”fillBlank”>
<title> The highest library of the world is situated in _____, _____, </title>
<answer> Shanghai, China </answer>
</Question>
</QuestionPaper>
```

The tags in the example above (like **<Question> and <answer>)** are not defined in any XML standard. These tags are “invented” by the author of this XML document.

5. **XML is platform independent**

   XML can run on all platforms. It’s created using standard text files. These files are compatible with all platforms such as Windows, Macintosh, and UNIX etc.

6. **XML can separate data form HTML**

   XML stores and describes data, which can later be formatted and presented in desired way. With XML, your data is for stored outside your HTML is for displaying whereas XML is for storing and describing data.

7. **XML can be used to create new Languages**

   XML is a meta language, which means it can be used to create new languages. A number of languages as per the needs or requirements of the specific domain have been created.

Some of them are:

**ADML** (Architecture Description Markup Language)[1],

- **BrainML**(Brain Markup Language) [2],
- **Chess GML**(Chess Game Markup Language) [3],
- **CML** (Chemical Markup Language) [4],
- **RecipeML** (Recipe Markup Language) [5],
- **GML** (Geography Markup Language) [6],
- **InkML** (Ink Markup Language) [7],
- **MathML** (Mathematical Markup Language) [8],
- **WML** ( Wireless Markup Language) [9], and many more.

The information box titled **Two Interesting Markup Languages based on XML** decribes two such languages.

**8. XML is a W3C recommendation**

XML is supported and recommended by World Wide Web Consortium (W3C). W3C forms working group to develop specifications for XML. The recommendations for XML are available at the URL http://www.w3.org/TR/xml/.

Now that you have fair idea about what XML is and what its features are, let us start learning to work with XML and create our own XML documents.

Structure of XML-file/document

Every XML document is a structured document. Every XML file or document has both

- A logical structure, and
- A physical structure

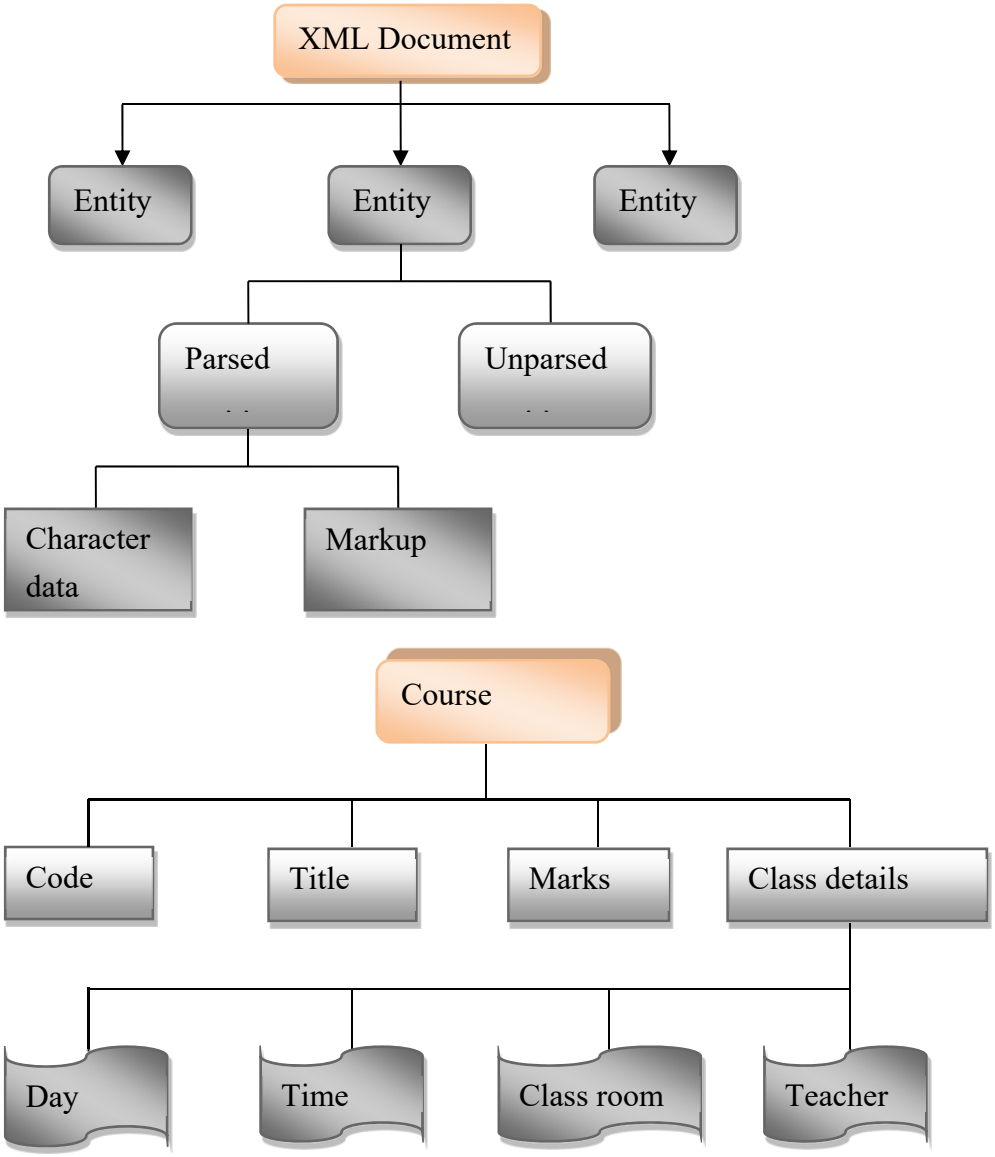
The logical structure basically tells about

- What all elements are to be included in the document
- The order of elements

The physical structure contains the actual data i.e., the actual content. The storage units in physical structure terms are called **entities.** Entities can either be contained inside document i.e., **internal entities** or can also exist outside the document i.e.,

**external entities**.

Before we go any further, let us have a look at following figure that shows the difference between logical and physical structure graphically through examples. Don't bother if some terms are not clear to you at this moment. All these would become very much clear to you in the due course.




Physical structure consist of one or more entities Logical structure consists of elements and their order

a) Physical structure

b) Logical structure

In terms of Physical Structure of an XML document:

- An entity is a storage unit.
- A Parsed Entity is processed by XML software such as XML parser.
- AN Unparsed Entity contains some related information and is not processed by XML-parser.

**Tags**

*Tags* are used to specify a name for a given piece of information. A tag consists of opening and closing angular brackets (<>) that enclose the name of the tag.

Example

<EMP_NAME>Hari bahadur </EMP_NAME>

Elements

Elements are represented using tags. An XML document **must always** have a *root element*. General format:

<element> … </element>

Empty element:

<empty-Element />

Example

<Authorname>John Smith</Authorname>


**Exmple**

<bookstore>

<book>

<category>PHP Books</category>

<link>http://www.itechcollege.com/books/PHP-n_150.html</link>

</book>

<book>

<category>Javascript Books</category>

<link>http://www.itechcollege.com/books/Javascript-n_146.html</link>

</book>
</bookstore>
In the example above, <bookstore> and <book> have element contents, because they contain other elements.

**Features of XML elements**

- XML Elements are Extensible
- XML documents can be extended to carry more information
- XML Elements have Relationships
- Elements are related as parents and children
- Elements havecContent
- Elements can have different content types: element content, mixed content, simple content, or empty content and attributes
- XML elements must follow the naming rules

**Attributes**

Attributes located in the start tag of elements. It Provides additional information about elements It often provide information that is not a part of data, it must be enclosed in quotes. Metadata (data about data) should be stored as attributes, and that data itself should be stored as elements

**Cooments**

*Comments* are statements used to explain the XML code.

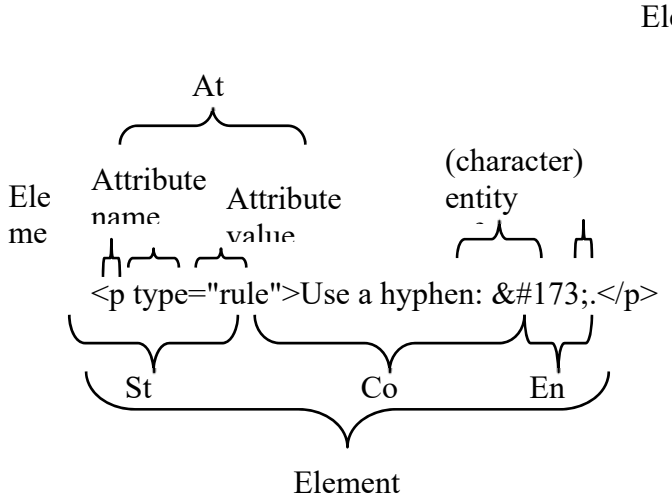
Example
<!--PRODUCTDATA is the root element-->
The text contained within a comment entry cannot have two consecutive hyphens
<!--PRODUCTDATA is the –-root element-->
Take a look at these examples:
<customer gender="male">
<firstname>John</firstname>
<lastname>Smith</lastname>
</customer>

<customer>

<gender>male</gender>
<firstname>John</firstname>
<lastname>Smith</lastname>
</customer>

In the first example gender is an attribute. In the last, gender is an element. Both examples contain the same information.

**Anatomy of an element**




**Example**

```
<bookstore>
  <book category="children">
    <title>Harry Potter</title>
    <author>J K. Rowling</author>
    <year>2005</year>
    <price>29.99</price>
  </book>
  <book category="web">
    <title>Learning XML</title>
    <author>Erik T. Ray</author>
    <year>2003</year>
   <price>39.95</price>
```

```
  </book>
</bookstore>
```

In the example above:

<title>, <author>, <year>, and <price> have text content because they contain text (like 29.99).

<bookstore> and <book> have element contents, because they contain elements.

<book> has an attribute (category="children").

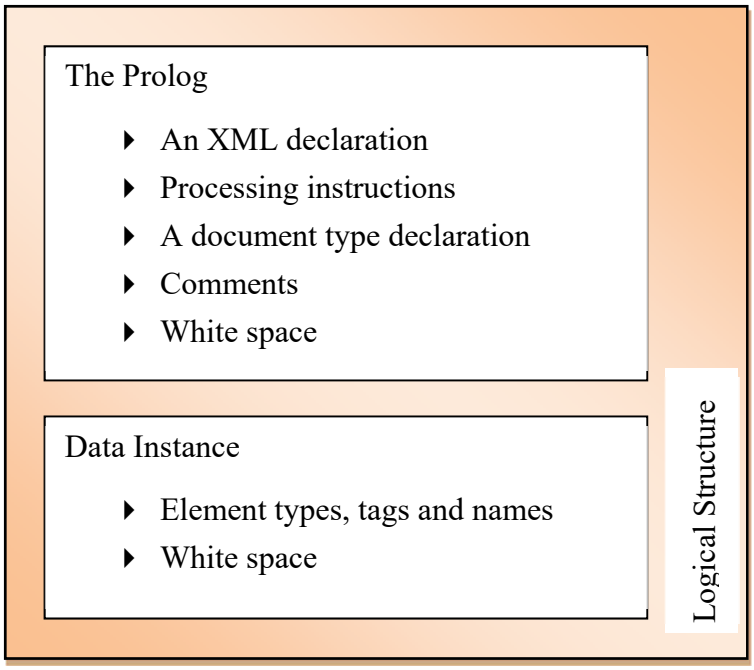
**Components of XML – File**

Let us mow learn about what all components are written in XML-file.

An XML document is made up of many things, such as declarations, elements (i.e., parts that markup a section), processing instructions, and comments. Here, We shall be talking about the basic declarations and elements of an XML-document.

Basically an XML-file contains (see figure)

- A prolog (optional), and
- A data instance



Let us talk about theses structure elements one by one. Following sections talk about these elements.

**Logical structure of XML document**

The logical structure of an XML-document consists of mainly:

- The prolog, and
- The Data instance

Above shown components comprise the basic logical structure. Each of these components further contain other components. Following lines will make it more clear where we are discussing the same.

**The prolog**

The prolog is a preface or introduction to the XML document. The prolog should be the first logical component. But one thing that you must know that prolog is optional.

The prolog can have up to five types of components (all optional):

**i) XML Declaration**

the xml declaration is a declaration that identifies following through pseudo-attributes:

a) the xml version
b) encoding
c) stand-alone or not

The XML declaration begins with a special tag<?xml……?>. It can have three psedo attributes as explained below:

**a) version pseudo-attribute specifies the xml version**

<?xml version= "1.0" ? >

the version pseudo – attribute refers to the xml version. As per this setting, this document refers to xml recommendation1.0.

**b) encoding pseudo-attribute(optional) specifies the character-set**

<? Xml version = "1.0" encoding= "UTF-8" ? >

The encoding pseudo-attribute specifies the character set. This pseudo- attribute is optional. If you skip it, the default character-set taken is UTF-8.

d) Standalone pseudo-attribute(optional) specificies whether document refers/needs external entities

<? Xml version=" 1.0" encoding= "UTF-8" standalone="no" ?>

The standalone pseudo-attribute tells whether the XML document requires external entities(e.g. a separate file explaining DTD or other presentation details) or not. If it requires then set standalone's value to "no" if it does not need to refer to any external entity, set it to "yes". The default value(in case you skip it ) is "yes".

A complete XML declaration will look like:

<?xml version= "1.0" encoding= "UTF-8" standalone= "no" ? >

**ii) Processing instructions**

The processing instruction (PI) are the instructions that pas information to the application. A processing instruction can take the following form:

<? Piname pseudo-attributes>

One most common pi is xml-stylesheet processing instruction that identifies a stylesheet for the XML-document using a cascading style-sheet(you can read about cascading stylesheet in previous chapter.)

<? Xml-stylesheet type="text/css" href= "main.css" ?>

Another example of xml stylesheet processing instruction could be :

<? Xml-stylesheet type= "text/xsl" href= "newmain.xsl" ?>

As you can makeout, this time it is referring to an xsl type text sytle-sheet.

**iii) Document type declaration**

The document type declaration consists of Markup code(i.e. tags etc) that indicates the grammar rules for the particular class of the document. The grammar rules are called Document Type Definition (DTD).

A DTD begins with !DOCTYPE declaration. A sample DTD is :

<! DOCTYPE animals system= "widanim.dtd">

A DTD is optional but if it appears, it must follow XML declaration and proceed that root element or DOCUMENT element.

Document type definition is a detailed topic, which we can not cover here as it is beyond the scope of the book.

**iv) comments**

the comments are not processed by XML parsers. The comments are inserted for purposes like:

a) to add notes about document structure

b) to break a document into sections.

Comments begin with< ! …… and end with ……> just like HTML comments.

Examples of comments are:

<! … library catalog last updated on 01-01-2015…..>

<!—test pattern =FIT-X 2015-->

**V) White space**

White space refers to spaces, tabs, carriage returns and blank lines. You can insert whitespaces to enhance readability of your document. XML parsers ignore additional white spaces.

**The Data Instance**

The data instance part of XML document follows the prolog and consists of one or more elements i.e., it contains the real data. The basic building blocks of data instance are the elements. Elements are means to define individual data items. An element begins with a start tag and ends with and end tag. The elements inside the data instance are the XML document's data containers. Let us discuss about various facets of elements in XML.

**Root/Document Element**

One thing that you must know is that, in data instance, a single element contains all the data of XML document; this element however can have other elements nested in it. IN other words, XML documents must contain one element that is parent of all other elements. This parent element is known as root element or document element.

Consider the following two data instances:

i) <gem>

<stone>

See, this is the parent element of all other elements; hence it qualifies to become the **root element**

<name> Ruby </name>

<type> Burma </type>

<weight> 1.14 carat </weight>

</stone>


:

:

</gem>

ii) <employee-list>

<employee>

<name> sandhya</name>

<empid> 2800 </empid>

<designation> PGT </designation>

</employee>

<employee>

<name> Joseph </name>

<empid> 3406 </empid>

<designation>TGT </designation>

</employee>

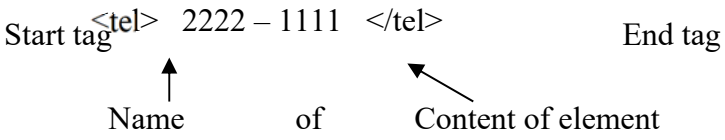
</employee-list>

**Element Types**

As you know by new that data instance is made up of elements contained inside one parent element called root or document element. The elements that are nested inside the root element belong to one of the following three element-categories:

- Start tag
- End tags
- Declared empty element tag

An XML element has a name and content e.g.,




- The **content** of an element is delimited by special markups called the **start tag** and **end tag**, just like HTML's tags.
- The **start tag** is the name of the element written inside angle brackets(< >) (e.g., <tel> above
- The **end tag** has the same name as that of its start tag but along with a preceeding slash before the name (e.g., </tel> above)

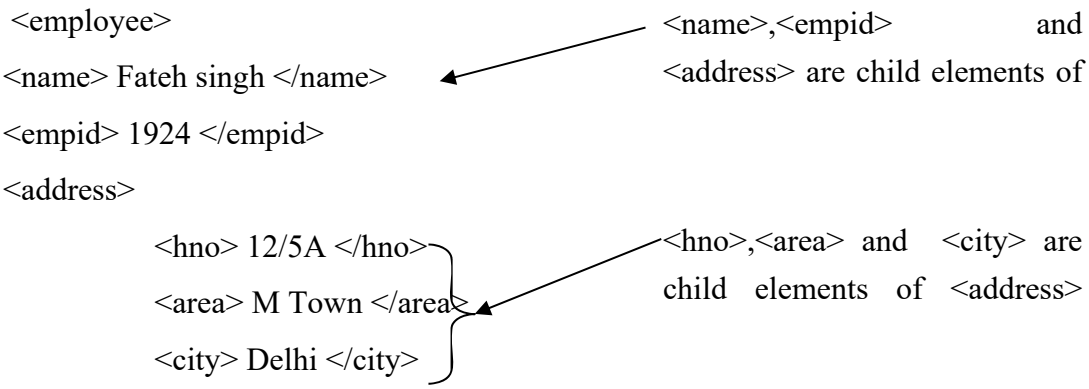
A tag hat does not have any end tag, (just like <BR> of HTML) cannot be written just as you do on HTML> But such a tag is written with name followed by a slash inside angle brackets e.g., <empty/> .

**Child Elements**

A tag can have other tags nested inside it.

The elements nested inside other elements are the child elements.

Consider the following code:



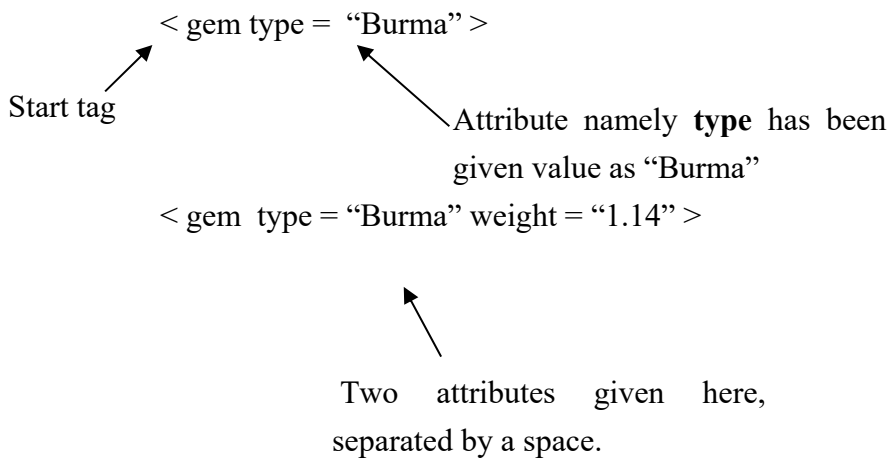
**NAMING RULES IN XML**

Names in XML (element names, attributes names etc.) must follow certain rules. These are:

1) Names in XML must start either with a letter or underscore character.
2) Rest of the name can consist of letters, digits, underscore characters, the dot (.) or a hyphen (-).
3) Spaces are not allowed in names.
4) XML is case sensitive i.e., <Tag> is not same as <tag> or <TAG>. (So a start tag as <Tag> must end with </Tag>. </tag> or </TAG> would not be considered as end tags of </Tag>).
5) Names cannot start with a string "xml". It is reserved for the XML specification.
6) By convention HTML elements in XML are written in uppercase, and XML elements are written in lowercase.

**Attributes**

Attributes are the property settings of the element. Attributes are defined through **name-value pairs** along with the start tag. Consider the following code :




As you would have noticed by now that the content that we represented as child elements has now been represented as attributes in the above code. Thus you can

represent an information either through child elements or through attributes But it you have to display the information or manipulate it then you must write them in elements.

Attribute values cannot be displayed or manipulated; they are good for representing additional information for the code-readers.

Also, you need to ensure that in a data instance, elements must have unique names and in an elements, attributes must have unique names.

Now that you have a fair idea about XML, don't you think some example problems should be discussed? Well, your wish is our command; here we go.

**Clients**

| Client ID | Name | Company | Phone | City |
|---|---|---|---|---|
| 001 | Jane Smith | ATnT | 9808512340 | Pokhara |
| 002 | Jameel Sheikh | NIIT | 9849054321 | Kathmandu |
| 003 | Raghav Seth | Wipro | 9841657532 | Banepa |
| : | : | : | : | : |

**Solution.**

```
<?xmal version = "1.o" encoding = "UTF-8" standalone =
"yes"?>
<clients>
<client>
<clientId> C01 </clientId>
<name> Jane Smith </name>
<company> ATnT </company>
<phone> 9808512340 </phone>
<city> Pokhara </city>
</client>
<client>
```

```
        <clientId> C02 </clientId>
        <name> Jameel Sheikh </name>
        <company> NIIT </company>
        <phone> 9849054321</phone>
        <city> Kathmandu </city>
</client>
<client>
        <clientId> C03 </clientId>
        <name> Raghav Seth </name>
        <company> Wipro </company>
        <phone>9841657532 </phone>
        <city> Banepa </city>
</client>
</clients>
```

**XML PRACTICAL LEARNING**

Before you start reading this section, make sure that you have gone through the information box on cascading styleheets as we'll be using that information also.

In order to work with XML practically, you need to proceed in following order:

I. Prepare an XML document as per the problem- statement.
II. Prepare a style-sheet file for the XML document.
III. Link the XML document with the stylesheet through <?xml-stylesheet> processing – instruction of prolog of the XML document.

Let us perform all this practically, through examples. Example 1.1 performs step i) and iii) and Example 1.2 performs the step ii).

Example 1 Create am XML document to represent various computer parts. Each part has its item-name, manufacture, model and cost. Choose data of your choice.

Solution.

```
        <?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/css" href="cd_catalog.css"?>
<CATALOG>
```

```
 <CD>
  <TITLE>Empire Burlesque</TITLE>
  <ARTIST>Bob Dylan</ARTIST>
  <COUNTRY>USA</COUNTRY>
  <COMPANY>Columbia</COMPANY>
  <PRICE>10.90</PRICE>
  <YEAR>1985</YEAR>
 </CD>
 <CD>
  <TITLE>Hide your heart</TITLE>
  <ARTIST>Bonnie Tyler</ARTIST>
  <COUNTRY>UK</COUNTRY>
  <COMPANY>CBS Records</COMPANY>
  <PRICE>9.90</PRICE>
  <YEAR>1988</YEAR>
 </CD>

</CATALOG>
```

EXAMPLES Create a css style-sheet for the XML document created in example 1.1.

Solution.

```
CATALOG {
  background-color: #ffffff;
  width: 100%;
}
CD {
  display: block;
  margin-bottom: 30pt;
  margin-left: 0;
}
TITLE {
  display: block;
```

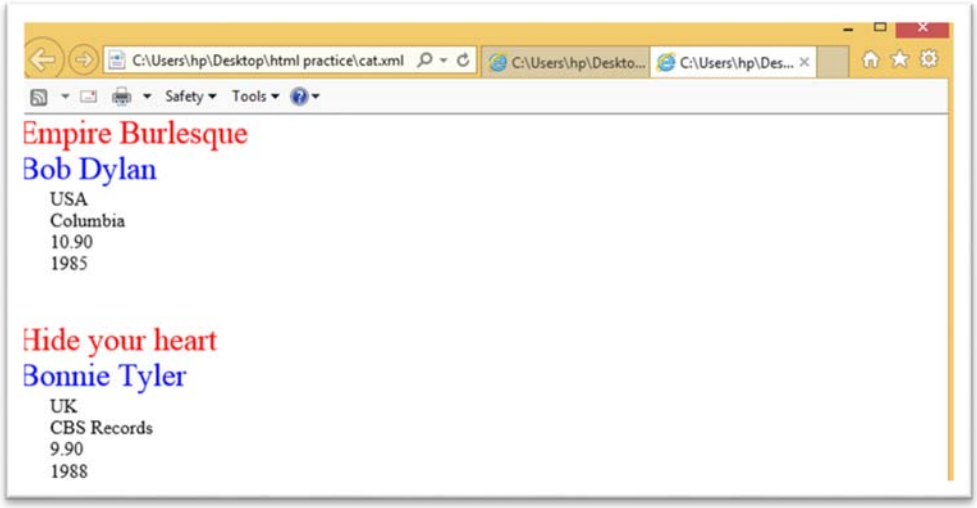
```
    color: #ff0000;
    font-size: 20pt;
}
ARTIST {
    display: block;
    color: #0000ff;
    font-size: 20pt;
}
COUNTRY, PRICE, YEAR, COMPANY {
    display: block;
    color: #000000;
    margin-left: 20pt;
}
```

The above file is stored as “parts.css”.

**Viewing the XML Document in Browser**

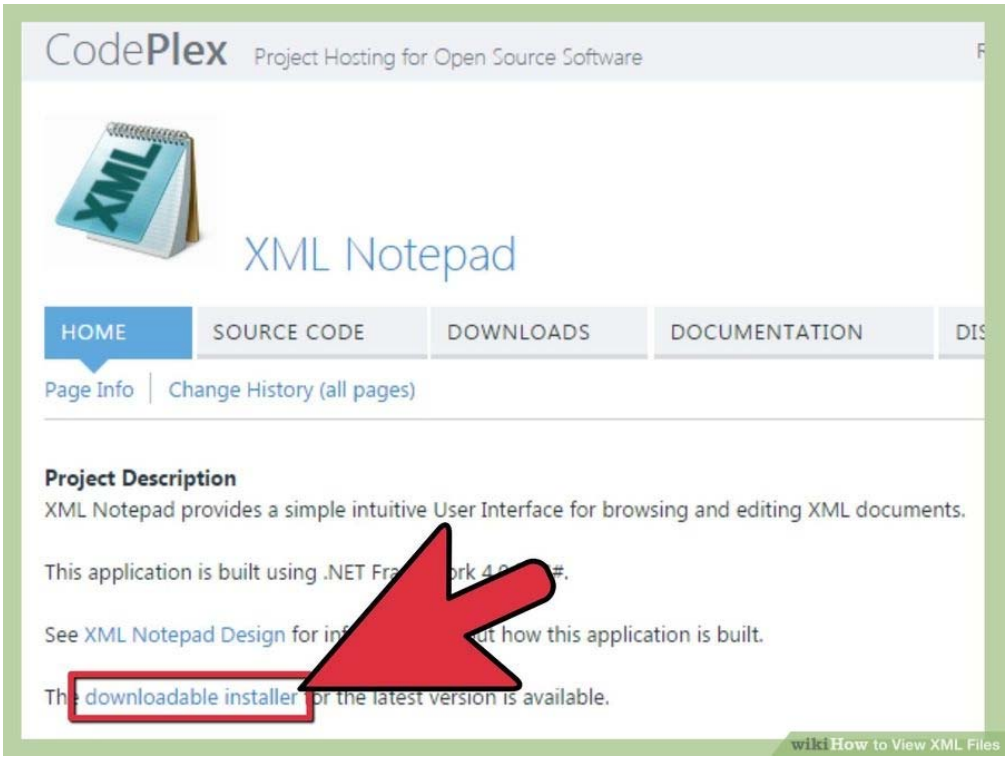
To view an XML document in a browser, you need to do the same that you do to open an HTML document. If the style-sheet file whose name is mentioned in the XML document is bot present along with the XML document, then your browser may either show you XML code or unformatted data depending upon its version.

However, if the style-sheet file is available along with the XML document, then the XML document will get formatted as per the stylesheet and shown. Example when viewed in a browser looked like the one shown on the above



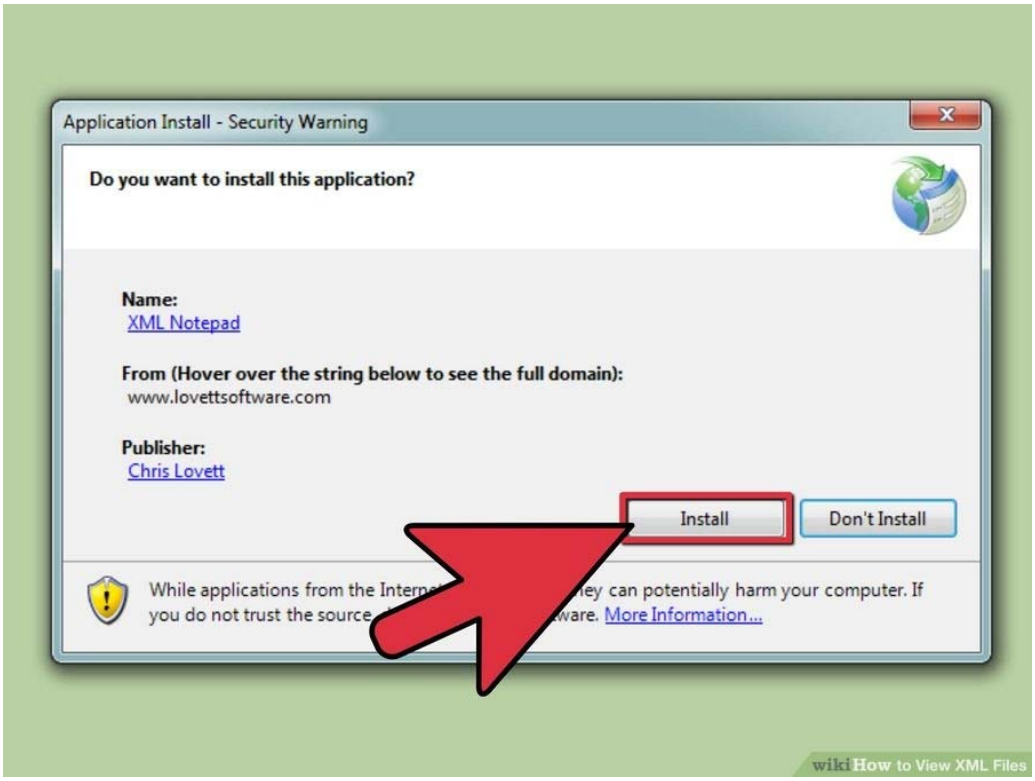

**View and Edit XML Files on Window**

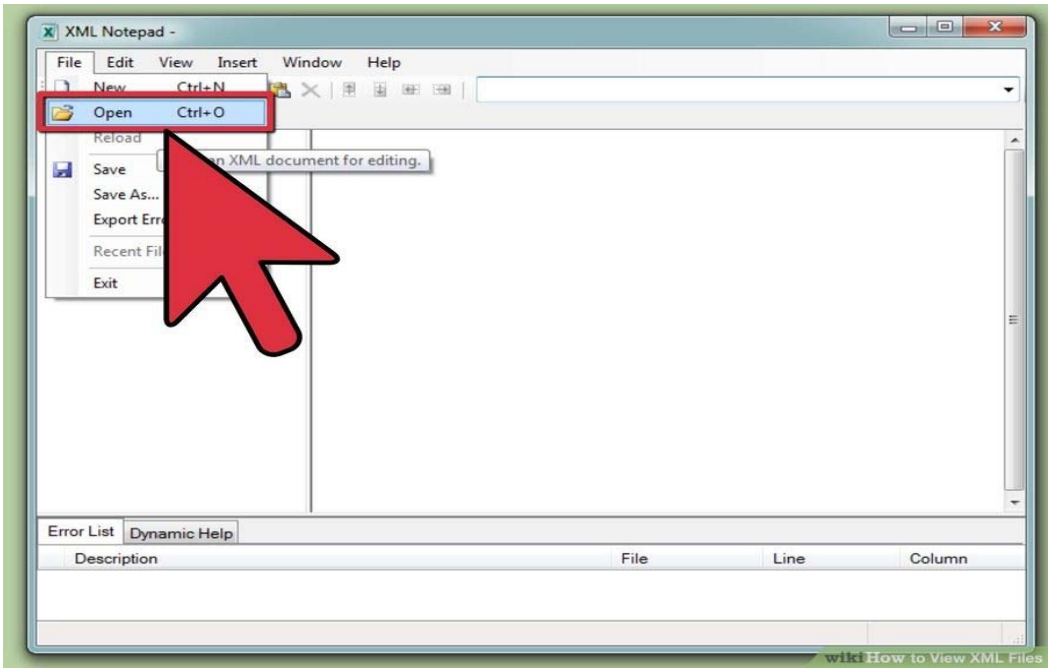
**Goto https://xmlnotepad.codeplex.com/ and follow the following steps**



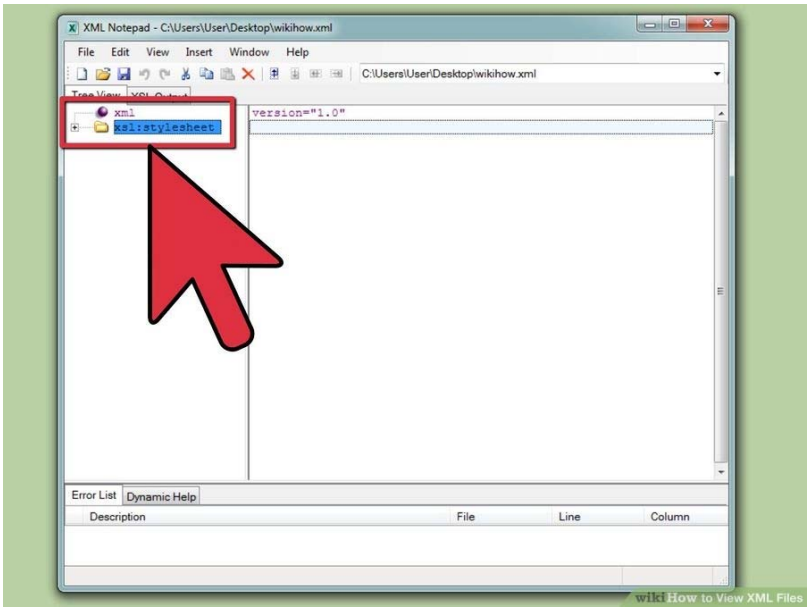
1. **Download XML Notepad.** XML Notepad is an XML viewer and editor published by Microsoft. Go to https://xmlnotepad.codeplex.com and download the downloadable or standalone installer. The links are located in the Project Description.

- If you're on Mac OS X, TextWrangler is a free text editor that color format the tags in an XML file to make it easier to edit.[1]

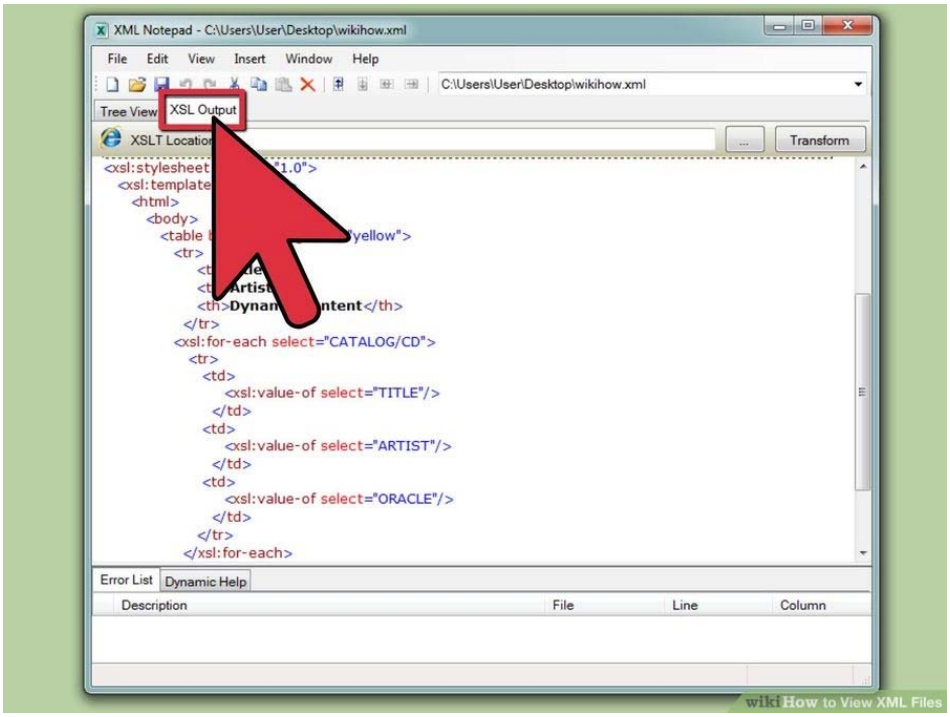


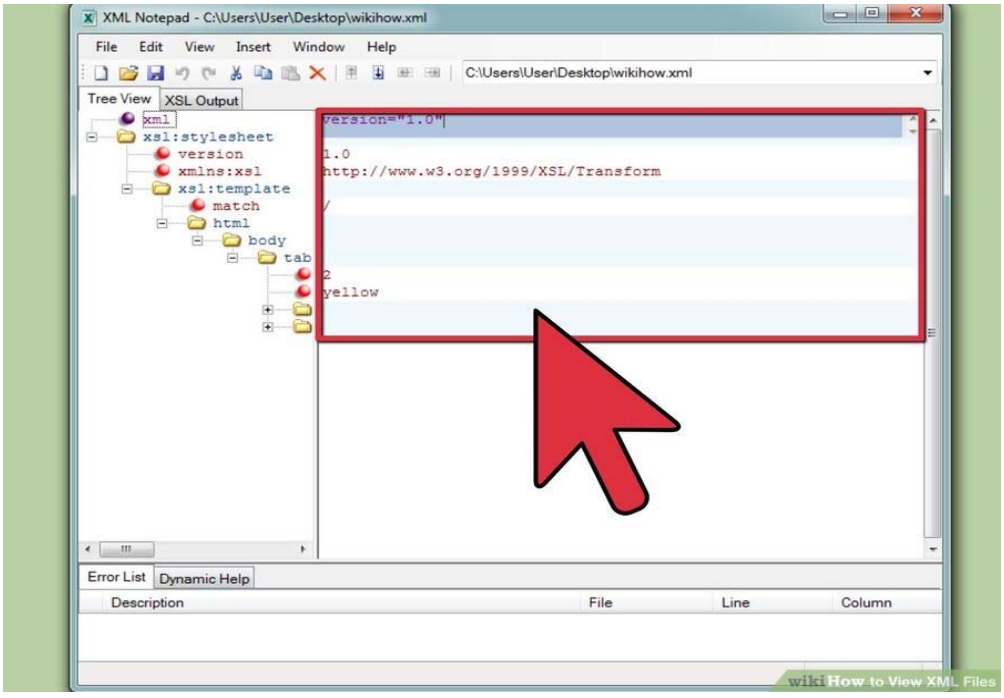


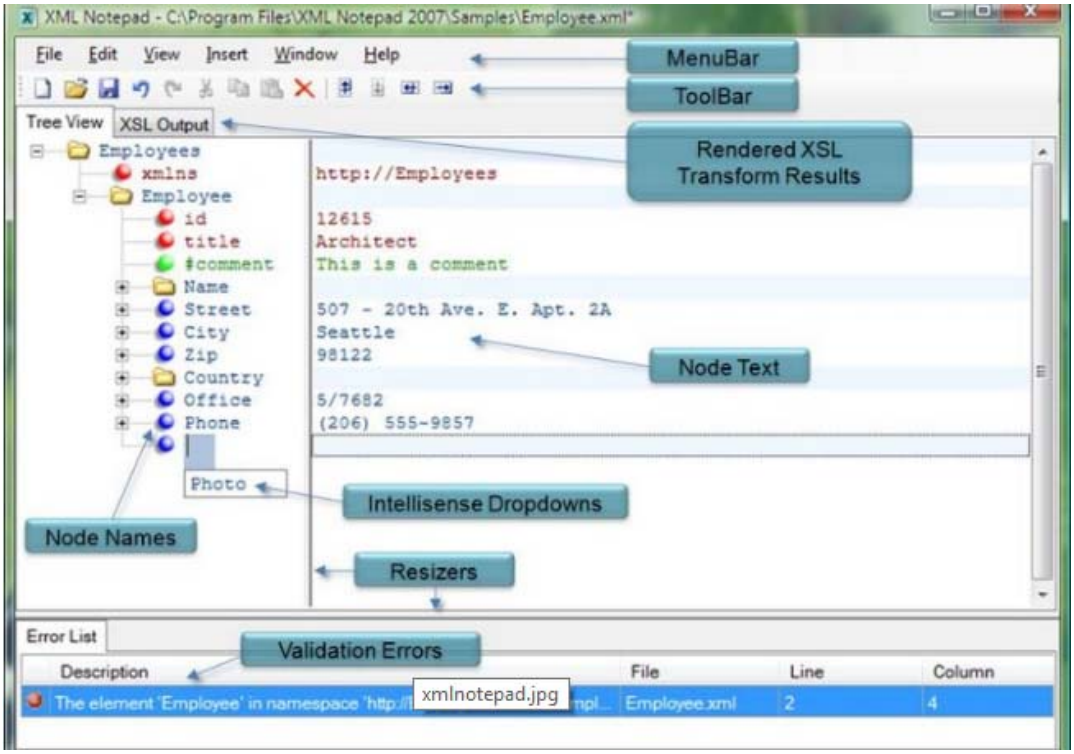

3. **Open an XML file with Notepad XML.** Open Notepad XML. Click the File menu, and then click Open. Find an XML file on your computer, and then click Open.




4. **View the XML in Tree View.** Tree View separates out the XML tags into the left sidebar. Click the Tree View tab. Click a tag to show the text content inside it.


XML Notepad - C:\Users\User\Desktop\wikihow.xml
File Edit View Insert Window Help
C:\Users\User\Desktop\wikihow.xml
Tree View
XSL Output
XSLT Location
Transform
<xsl:stylesheet
<xsl:template
<html>
<body>
<table
<tr>
<th>Artist
<th>Dynamic Content</th>
</tr>
<xsl:for-each select="CATALOG/CD">
<tr>
<td>
<xsl:value-of select="TITLE"/>
</td>
<td>
<xsl:value-of select="ARTIST"/>
</td>
<td>
<xsl:value-of select="ORACLE"/>
</td>
</tr>
</xsl:for-each>
Error List
Dynamic Help
Description
File
Line
Column
wikiHow to View XML Files


XML Notepad - C:\Users\User\Desktop\wikihow.xml
File Edit View Insert Window Help
C:\Users\User\Desktop\wikihow.xml
Tree View
XSL Output
xml
xsl:stylesheet
version
xmlns:xsl
xsl:template
match
html
body
tab
version="1.0"
1.0
http://www.w3.org/1999/XSL/Transform
/
2
yellow
Error List
Dynamic Help
Description
File
Line
Column
wikiHow to View XML Files




5 **View the XML in XSL Output.** The XSL Output view shows you the XML like you would see it in a browser.

6 **Edit the XML file.** Edit the XML file like you would a text file. XML Notepad has many other features for working with XML files.

Detail view of XML notepad

**XML Standards Reference**

Extensible Markup Language (XML) is a simple flexible text format that can be used as the basis for creating new markup languages that can be used in document publishing and data exchange. XML is based on a series recommendations published by the working groups of the World Wide Web Consortium (W3C) and is therefore well suited (though not limited) to use in Web-based applications.

At Microsoft, our engineers contribute to the efforts of the W3C working groups who

define and set standards for XML and other Web protocols. Wherever possible, we strive to achieve full compliance with the XML standards once they are stable and published as recommendations by the W3C.

Among the XML standards Microsoft currently provides developer support for are the following:

- The XML Schema definition language (XSD), a current W3C standard for using XML to create XML Schemas. XML Schemas can be used to validate other XML documents.
- Extensible Stylesheet Language Transformations (XSLT) 1.0, a current W3C XML style sheet language standard. XSLT is recommended for transforming XML documents.
- The XML Path Language (XPath) 1.0, a current W3C XML standard used by XSLT and other XML programming vocabularies to query and filter data stored in XML documents.

**Instruction for teachers**

1. Give the practical examples found at their home and surrounding.
2. Use the Charts as far as possible to explain the content.
3. Assign the homework at the end of every class from the chapter so far covered during that class.
4. Conduct group discussions after finishing one lesson.

**References and Resources:**


- https://msdn.microsoft.com/en-us/library/ms256177(v=vs.110).aspx
- Information Technology class x Dhanapat Rai & co . Sumitra Arora.
- www.w3schools.com

# UNIT 6
# GUI Based HTML Editor

A HTML editor also named as web content editor is a software where user can easily design a website and have a flexibility for create, view and update content. A HTML editor is for editing HTML, PHP, CSS, the source code of a webpage.

Generally, In the world of web development, there are two types of HTML editing tools. One is simply a text based HTML editor also called html text editor, where every piece of code is typed manually. The another is a WYSIWYG (wizzy-wig or What You See Is What You Get) HTML editor,also called GUI based editor, with which the web page is developed using a visual platform. Some visual editors offer the ability to view and edit the code as well. Although the HTML markup of a web page can be written with any text editor but specialized HTML editors can offer convenience and added functionality with Graphical User Interface(GUI) so it is called GUI Based HTML Editor(WYSUWYG). GUI based HTML editors provide the features that are not available with plain text editors.

There are many advantages in using the html editors for creating WebPages and developing websites. Some of listed below:

- Website browsing & navigation becomes easy and user does not have to remember all the html tags for creating the web pages.
- Using advanced editor is very much easy to manage web project in proper and in standard way.
- Some html editors come along with form validations and have a compatibility to work with multiple browsers without affecting the look and feel of the website.
- Most html editors also do come with SSI (Server Side Includes), which means the web pages are generated on the server side and one can customize many of the files and features on the website.
- Html editors come very handy when creating tables, borders around images, changing the background color to reflect the concerned business style within less time.

**Introduction to different types of Editor**

Learning HTML is a great way to make use of Internet. Once a little understanding is achieved for a web structure then html editors come in handy for live publishing and popular websites html editors or WYSIWYG editors comes with a tool bar to make life even simpler where a Microsoft word like interface is given for as to designing, making changes and embedding images.

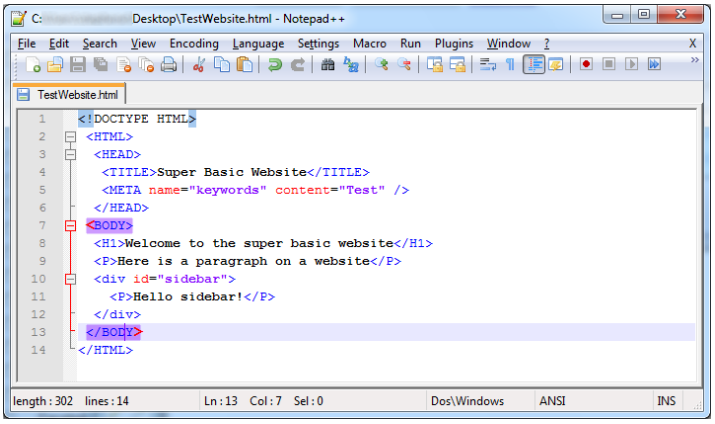
**Text Based HTML Editors are:**

Wordpad or Notepad++

WordPad is a good choice for writing papers or creating documents that you want to print. It is also great for making lists, since it supports bullets. WordPad is also used as a html editor. Html file can be create, opened and update but it is lack of formating and every tag have to be remember while programming in WordPad editor.

Notepad++, earlier version is known as notepad only, is the most basic text editor, which allows you to open and create text files. While creating several paragraphs of text with Notepad++, using line breaks (by pressing the Enter key), the program does not give you text formatting options. For example, you cannot change the font size or make the text bold.

Both programs are text editors that are included with the Windows operating system. Notepad++ is similar to WordPad, but gives more formatting options. Notepad++ is useful for program formatting and has feature for suggestion for the syntax and tags so that programmer doesn't have to remember all the tags.

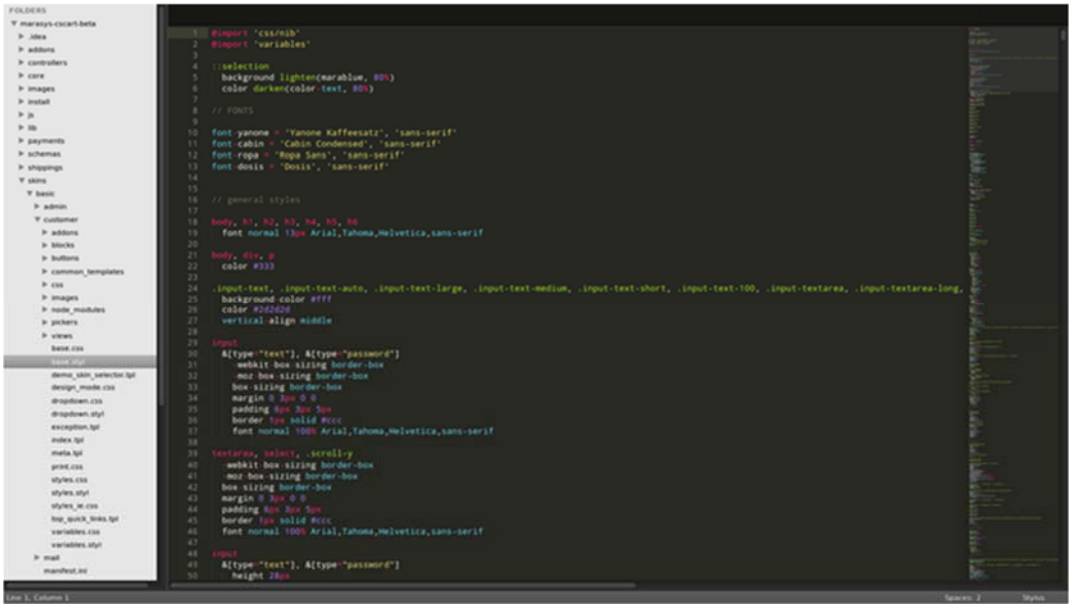


**Sublime**

Sublime Text is a sophisticated text editor for code, markup and prose. with slick user interface, extraordinary features and amazing performance. Sublime Text is a cross-platform source code editor with a Python application programming interface(API). It natively supports many programming languages and markup languages and maintained under free-software licenses.

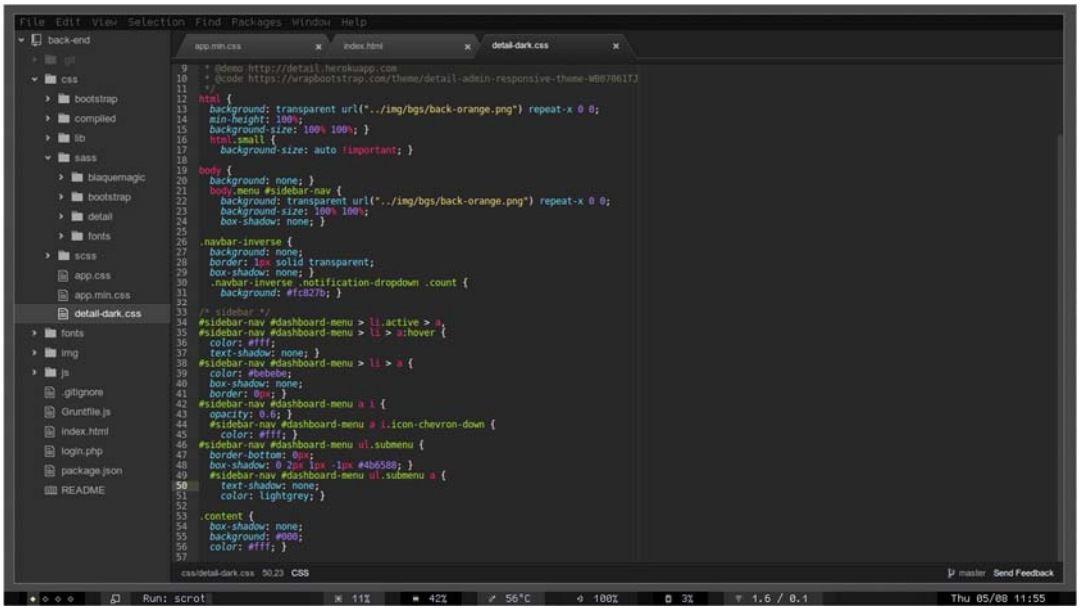
Sublime also auto-completes variables created by the user. This feature allows users to run code for certain languages from within the editor, which eliminates the need to switch out to the command line and back again. This function can also be set to build the code automatically every time the file is saved. Customizable key bindings, a navigational tool which allows users to assign hotkeys to their choice of options in both the menus and the toolbar. Spell check function corrects as code type.

Complete folder or project can be opened and edited side by side so that programming is very easy in this editor.



**Atom**

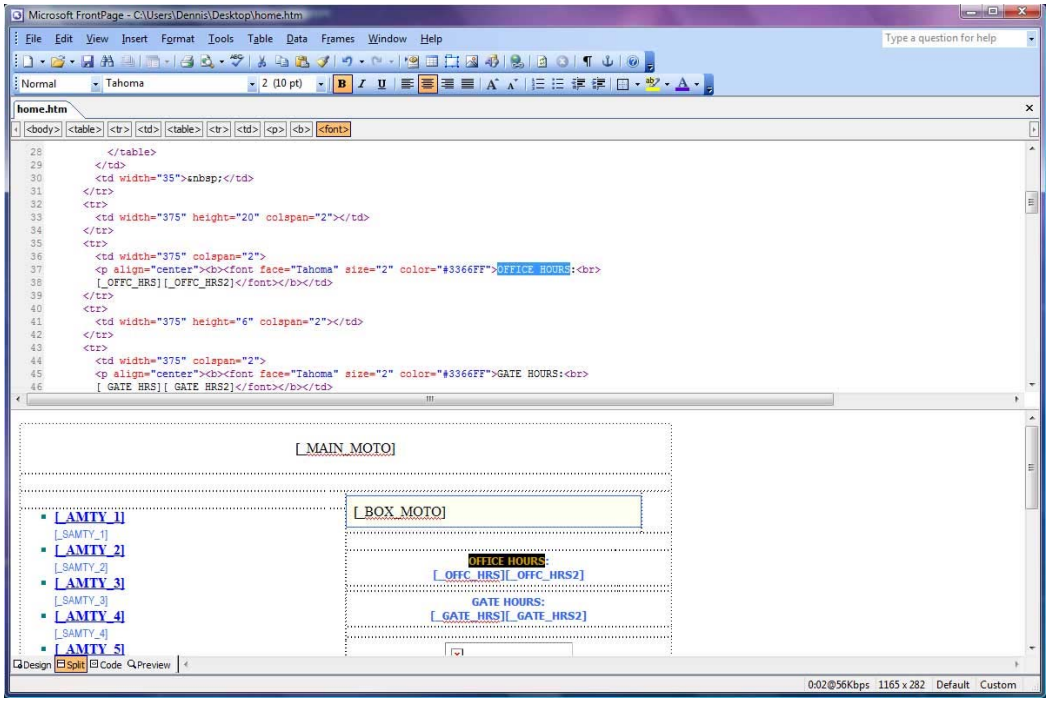
Atom is a text editor that's modern, approachable, yet hackable to the core—a tool anybody can customize to do anything but also use productively without ever touching a config file. Atom works across operating systems. Smart auto completion Atom helps to write code faster with a smart, flexible auto complete. File system browser Easily browse and open a single file, a whole project, or multiple projects in one window. Multiple panes Split your Atom interface into multiple panes to compare and edit code across files. Find and replace Find, preview, and replace text as typed in a file or across all projects.




**Front page from Microsoft**

Microsoft FrontPage (full name Microsoft Office FrontPage) is a discontinued WYSIWYG HTML editor and Web site administration tool from Microsoft for the Microsoft Windows line of operating systems. Microsoft FrontPage has since been replaced by Microsoft Expression Web and SharePoint Designer, which were first released in December 2006 alongside Microsoft Office 2007.

FrontPage 2003 consists of a Split View option to allow the user to code in Code View and preview in Design View without the hassle of switching from the Design and Code View tabs for each review.

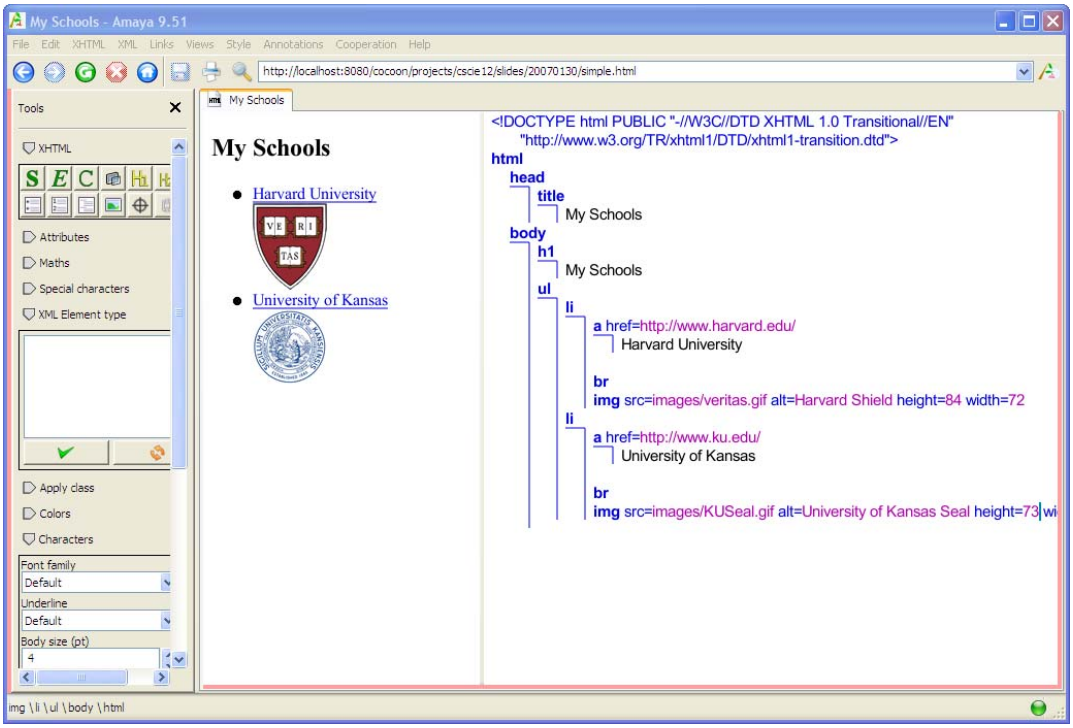



## Graphical User Interface Based HTML Editors:

### Dreamweaver from Adobe

Adobe Dreamweaver is a proprietary web development tool developed by Adobe Systems. Dreamweaver was created by Macromedia in 1997. DreamWeaver subsequent to version 8.0 have been more compliant with W3C standards. Recent versions have improved support for Web technologies such as CSS, HTML,PHP, JavaScript, and various server-side scripting languages.

Adobe Dreamweaver is a web design and development application that combines a visual design surface known as Live View and a code editor with standard features such as syntax highlighting, code completion, and code collapsing as well as more sophisticated features such as real-time syntax checking and code introspection for generating code hints to assist the user in writing code. Combined with an array of site management tools, Dreamweaver lets its users design, code and manage websites as well as mobile content. Dreamweaver is positioned as a versatile web design and development tool that enables visualization of web content while coding.




### Amaya

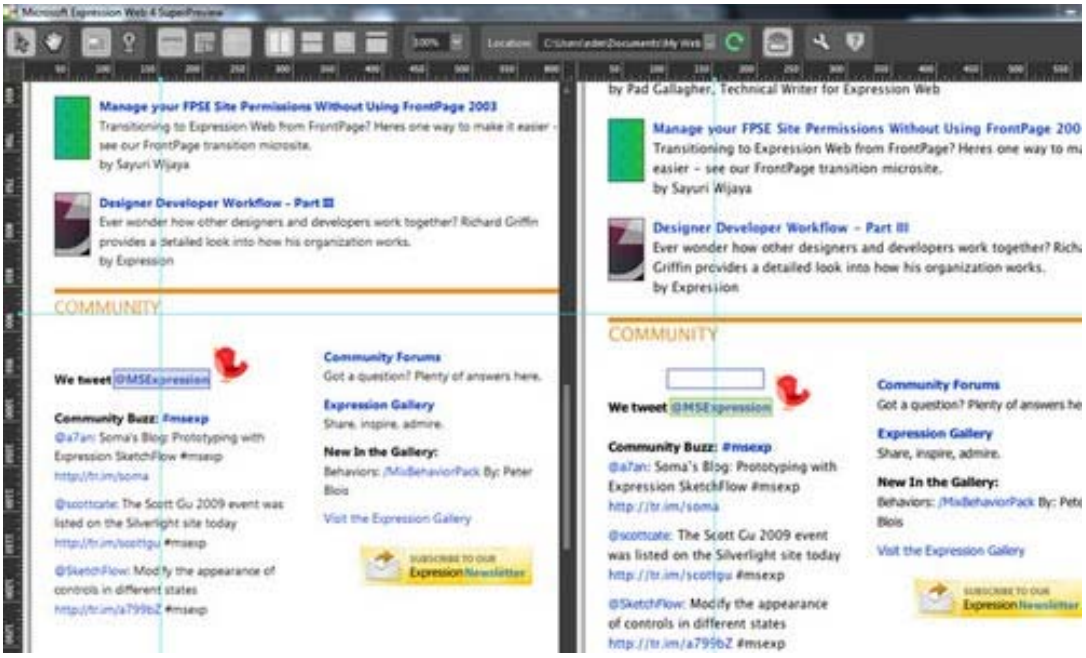
Amaya is a Web editor, i.e. a tool used to create and update documents directly on the Web. Browsing features are seamlessly integrated with the editing and remote access features in a uniform environment. Work on Amaya started at W3C in 1996 to provide a framework that can integrate as many W3C technologies as possible. It is used to demonstrate these technologies in action while taking advantage of their combination in a single, consistent environment.

Amaya started as an HTML + CSS style sheets editor and is an open source software project hosted by W3C. All are invited to contribute in many forms (documentation, translation, writing code, fixing bugs, porting to other platforms). The Amaya software is written in C and is available for Windows, Unix platforms and MacOS X.

### Microsoft Expression Web

Expression Web is Microsoft's current offering in the WYSIWYG arena (the popular but much maligned FrontPage was retired in 2003). For those of you who associate

Microsoft with poor web standards compliance, take comfort knowing that Expression Web has a totally separate engine from Internet Explorer and is compliant with a wide range of current web standards. It shares a lot of features with the other WYSIWYG editors featured here, like highlighting code errors and non-compliant code, a built-in CSS editor, and more, it also stands out for features like search engine optimization—offering you tips and ideas to optimize your sites for better crawling and search engine ranking.




Creating a page/view page in Text Editor

**Creating a Page**

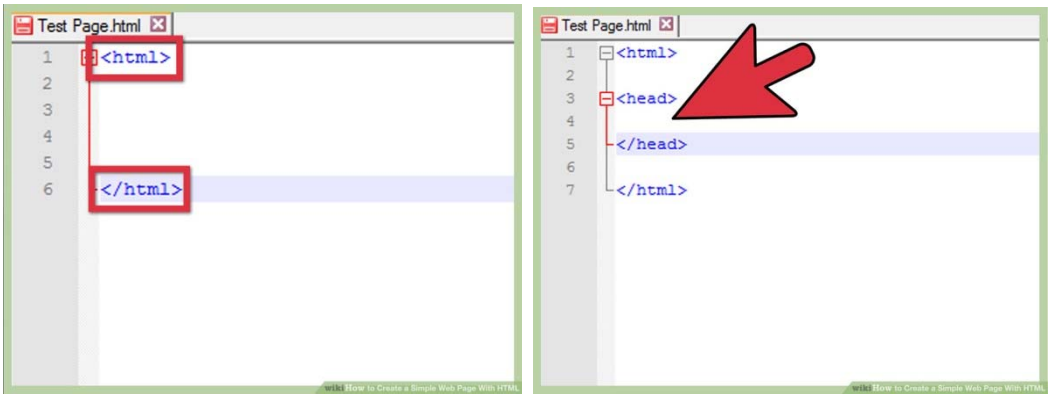
Now lets starts to create simple html page in simple text editor.

Go to Start, then "Programs" and then "Accessories." Click "(any text editor)". It is a lot easier if you use Notepad++ (you can download it for free on the internet).

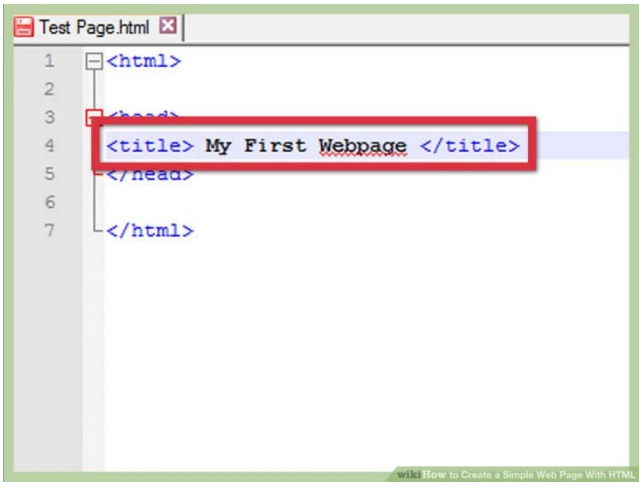
While creating page, standard order of html code should be write in a standard way.

To tell the browser what language you are using type <html> at first. It is the first tag you write that tells the computer you're starting a web-page. It will also be closed last, so at the end of the document, close it off by typing this: <>. This ends the web page.

Add the heading of the page as shown. Title tag is included inside heading tag.




Give your page a title. A title is important because it gives your users an idea what the page is about. Also, when users bookmark your site, that title is all they will see in their bookmark list. The title for HTML code is <title>. Close it off at the end of your title by writing </title> The title is going to show on the tab don't expect it to be the title of the actual website.




Work on the body of the page. Type <body> to open the body tag. Then close the body tag by typing </body>. The bulk of the information for your web-page goes between <body> and </body>.

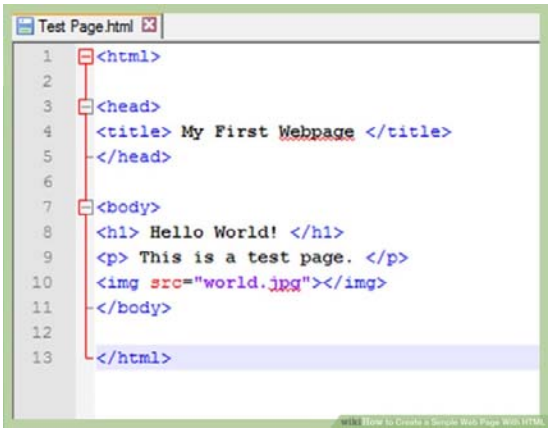
- To give your web-page a background color, you can add a style to the body. Instead of simply writing , write <body style="background-color:silver">. You can try a different color or even a hex code. The words in the quotation

marks are known as "attributes." They must be surrounded by quotation marks!

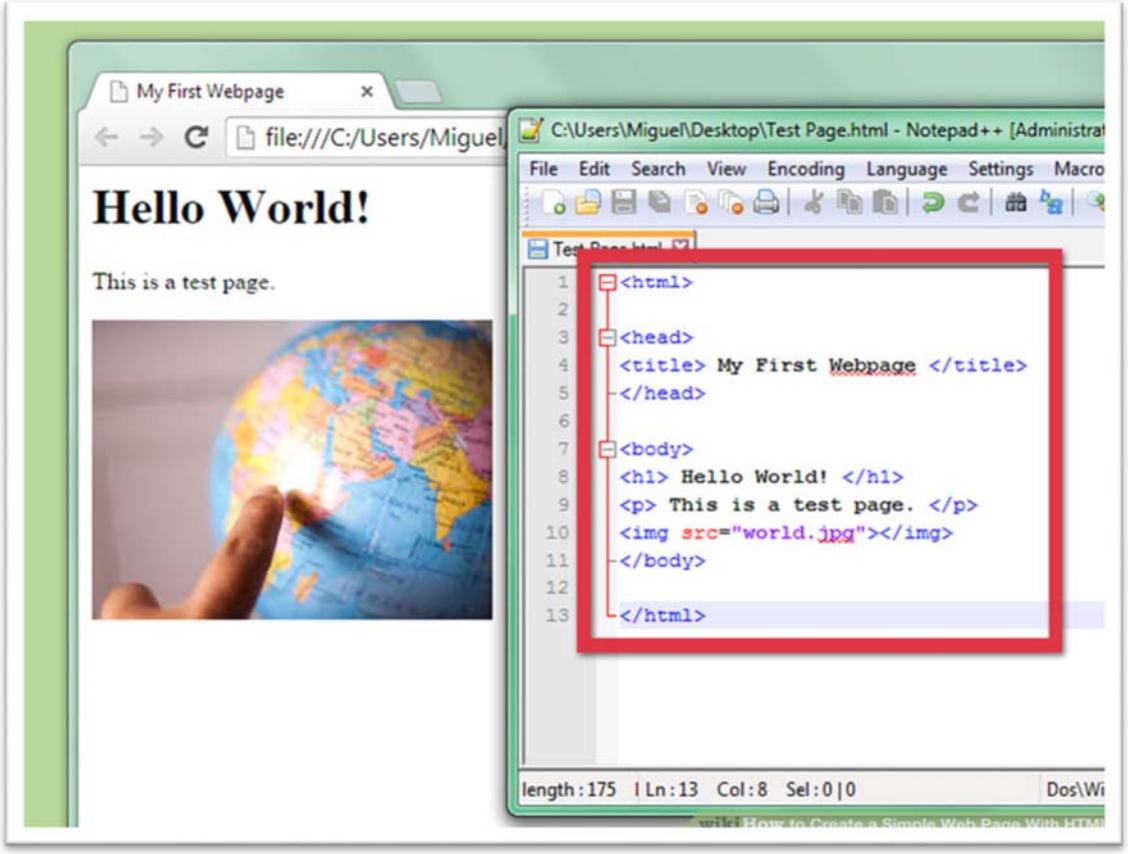
Write some text between the body tags.

- To make the text go to the next line (like pressing "Enter" on your keyboard), write <br>.
- Want to add a marquee, otherwise known as a word that moves across a screen. Simply type <marquee>TEXT GOES HERE</marquee>.

Add some pictures.If you want to put a picture from the Internet onto your web page, the HTML code for pictures is <img src="URL"> . The closing tag is: . However, the closing tag is optional.


Save your work. Go to "save as", put a filename with an .html extension (such as "test page.html") and choose "all files" or "text" under file type. It won't work if both are not done. Go find the page wherever you saved it, double click it, and your default web browser should open up your very own web-page.




It's not enough "just" to type text in the editor, you also need to give instructions to the computer. To do this in HTML, you use tags.

**Tags**

HTML pages are filled with so-called tags. Although these tags cannot be seen on the screen by those visiting your site, they allow the computer to understand what it has to display. Tags are easy to spot. They are surrounded by "angle brackets", in other words symbols < and >, like this: <tag>

**Tags in pairs**

They are opened, contain text, and are closed later. Here's what they look like:

<title>This is a paragraph</title>

**Orphan tags**

These are tags that are most often used to insert an element at a given point (for example, an image). There is no need to define the start and the end of the image. You just want to tell the computer to "Insert an image here".

An orphan tag is written like this:

```
<image />
```

Note that the ending / is not compulsory. You could just type<image>. To avoid confusing them with the first type of tag, however, web masters recommend that this / (slash) should be added at the end of orphan tags. So you'll see me adding a / to orphan tags.

**Attributes**

Attributes are a bit like tag options. They supplement them to provide additional information. The attribute is placed after the name of the opening tag and is most often a value, like this:

```
<tag attribute="value">
```

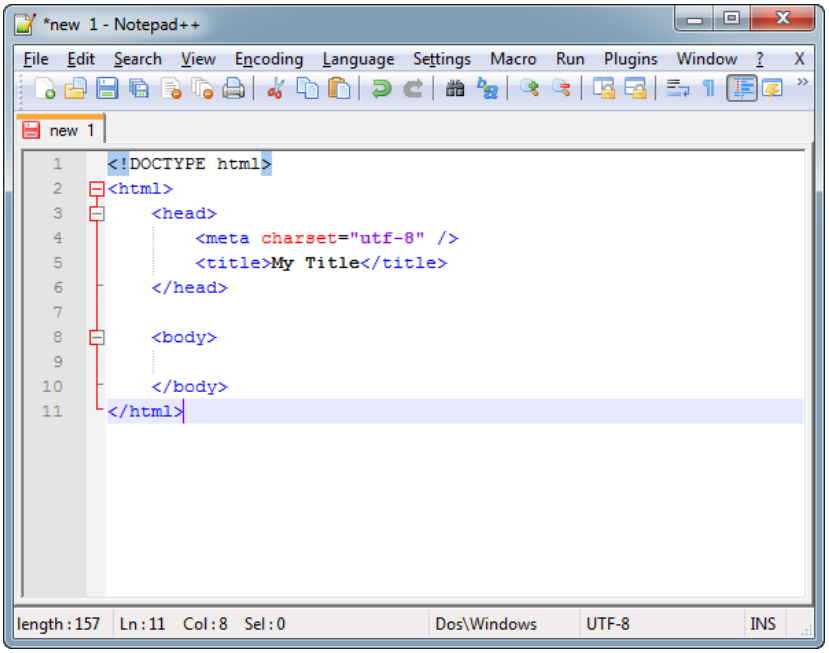
Basic structure of an HTML page

Let's go back to our text editor (in my case Notepad++). Just type or copy and paste the source code below into Notepad++. This code corresponds to the basis of a web page in HTML:

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

Spaces added at the start of some lines to "shift" the tags. Although this is not compulsory and has no effect on how the page is displayed, it makes the source code more readable. This is called indentation. In your editor, just press the Tab key to get the same result.

The result when copied into Notepad++ is shown in the next figure.




**DOCTYPE**

<!DOCTYPE html>

The very first line is called the doctype. It is essential as it is it indicates that it is effectively an HTML web page. It's not really a tag like the others (it starts with an exclamation mark), and can be considered to be an exception that proves the rule.

To the delight of webmasters, it was decided to simplify it in HTML5. When you see a short doctype tag (<!DOCTYPE html>), this means that the page is written in HTML5.(Although is is not used in the above example.)

The </html> tag

<html>

</html>

This is the code's main tag. It includes the whole content of your page. As you can see, the</html>closing tag is located right at the end of the code!

The header<head>and the body<body>

A web page consists of two parts:

The header<html>: this section provides some general information on the page such as its title, the encoding (for managing special characters), etc. This section is usually fairly short. The information contained in the header is not displayed on the page and is simply general information intended for the computer. It is, however, very important!

The title <title>of the page: This is the title of your page, which is probably the most important item! Every page must have a title that describes what it contains. It is advisable to keep the title fairly short (less than 100 characters in general).The title is not displayed in your page but at the top of it (usually in the browser tab). Search engines such as AltaVista, Infoseek, and Webcrawler will use the <TITLE> content to display the title of the page when they return information about it. It makes your page look bad if they instead display "No Title".

```
<html>
    <head>
        <title>This is title</title>
    </head>
</html>
```

The body<body>: this is where the main part of the page is located. Everything we type here will be displayed on the screen. Most of our code will be typed within the body. You'll notice that the tags are opened and closed in a specific order. For example, the<html>tag is the first one that is opened and is also the last one that is closed (at the very end of the code, with</html>). The tags must be closed in the reverse order they are opened. For example:

- <html><body></body></html>: correct. A tag which is opened inside another tag must also be closed inside it.
- <html><body></html></body>: incorrect, the tags are intertwined.

**Comments**

In this chapter, we learned how to create our first real HTML page. A comment in HTML is a text that is simply a memo. It is not displayed, is not read by the computer and does not change the display of the page. It is used by you and those who read your

page's source code. You can use comments to leave indications on how your page works. Comments will help you remember how your page works if you return to your source code after not seeing it for some time. No kidding, this happens to all webmasters.

**Inserting a comment**

A comment is an HTML tag with a very special form:

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

You can put it wherever you want in your source code: it has no effect on your page, but you can use it to help you find your way round source code (especially if it's long).

```
<!DOCTYPE html>
<html>
<head>
<!-- Page header -->
<title>Title</title>
</head>
<body>
<!-- Page body -->
</body>
</html>
```

Everyone can see your comments... and all your HTML code!

Everyone can see the HTML code of your page once it has been placed on line on the Web. Simply right click on the page and select "View page source" (the exact text may vary according to your browser).

**Formatting Pages**

Formatting pages means to manage the indentation of the html code in the pages.

Let's look at some examples of coding techniques to make it clear why you should indent(format pages) your HTML.

Wrong method 1: Everything on one line

```
<p>Lorem ipsum dolor sit ame  laboris nisi ut aliquip ex ea commodo consequat.</p>
```

**Problems:**

- Hard to figure out where code ends & text/content begins
- Hard to see where the is, even with wrapping turned on

Wrong method 2: 3 lines, but no indentation

```
<p>
Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt
ut labore et dolore magna aliqua.
Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.
</p>
```

Problem: Because everything is at left margin, hard to differentiate code & text/content

Best method: indent text/content

```
<p>
   Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor
   exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.
</p>
```

Advantages:

- Start ( ) & end ( ) are at same level, so it's easy to see that you've closed code & also see where code starts & ends
- Easy to differentiate between code & text/content

Examples of code indentation

Here are some examples of how Web Sanity likes to indent our code.

```
<p>
   this is paragraph1.
</p>
<p>
   this is paragraph2.
   <br>This is line break.</br>
</p>
```

```
<blockquote>
   <p>
      this is paragraph inside block-quote.
   </p>
</blockquote>

<table>
   <tr>
      <td>
         first row first column.
      </td>
      <td>
         first row second column.
      </td>
   </tr>
</table>
```

**Exceptions**

<h1>Title of the page</h1>

Note: We don't indent<h1>& other headers because they're short, but if you want to indent<h#>, that would be fine.

```
<head>
   <title>Document title</title>
</head>
```

Note: Again, we don't indent<title>& most other items inside <head> because they're short.

```
<ul>
   <li>Lorem</li>
   <li>Ipsum</li>
   <li>Dolor</li>
</ul>
```

Note: We don't indent<li>because most of the time the contents are short, because we're more concerned about the<ul>&</ul>, & because it can really lengthen the overall code. That said, if you want to indent<li>, feel free.

**Inline elements**

You never indent inline elements. Treat them like text/content. Examples:

```
<p>
   Lorem ipsum <strong>dolor sit amet</strong>, consectetur adipisicing elit,
   sed do <em>eiusmod tempor</em> incididunt ut labore et <code>dolore
   magna aliqua</code>. Ut enim ad minim veniam, <font size="5">quis
   nostrud</font> exercitation ullamco <a
href="http://www.demonoid.me">laboris
   nisi ut aliquip</a> ex ea commodo consequat.
</p>
<blockquote>
   <p>
      Lorem ipsum dolor sit amet, <a
href="http://www.avclub.com">consectetur
      adipisicing elit</a>, sed do <strong>eiusmod tempor</strong> incididunt
      ut labore et <em>dolore</em> magna aliqua.
   </p>
</blockquote>
What about the basic structure of a webpage?
<html>
<head>
   <title>Document title</title>
</head>
<body>
   <h1>Page title</h1>
   <p>
      Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed
   </p>
</body>
</html>
```

So why aren't<head>&<body>indented? A couple of reasons:

- You know that<head>&<body>are the children of <html>; in fact, they're always the only children of<html>, so there's no need to remind yourself of this via nesting
- You save yourself one level on indentation; when you start nesting HTML inside HTML inside HTML inside HTML, this can add up.

**Text editor should make it easy to indent your code**

You could press tab or the space bar every single time you want to indent your code, but you shouldn't have to do that. Fortunately, good text editors help you out when it comes to indenting.

**Auto-indent**

Sublime, for instance, has a setting in its Preferences for "Auto-indent", which it defines as follows: "When this option is selected, pressing the Return key in new windows automatically inserts spaces or tabs to indent the new line to the same level as the previous line."

Any good text editor should have a setting like this somewhere. The trick is finding it in the editor's Preferences or Options.

**Spaces or tabs? (Spaces!)**

Should you use spaces or tabs for indenting? A debate has raged on this topic for decades, with different developers insisting that theirs is the right choice. Your text editor should let you choose what gets inserted when you press the TAB key: a tab or spaces. Different text editors call that setting different things.

- Notepad++: Preferences > Language Menu/Tab Settings > Replace by space

**How many spaces?**

The next question is, how many spaces get inserted when you press TAB? Basically, people either choose 2, 4, or 8 spaces. Your text editor should let you choose how many spaces get inserted when you press TAB. Different text editors set that in different ways. Here are some examples:

- Notepad++: Preferences > Language Menu/Tab Settings > Tab size

**Indenting several lines at once**

What if you want to indent more than one line of code? Like 5? Or 25? You could manually move the cursor to the start of each line and press TAB the requisite number of times, but that would quickly grow tedious.

Fortunately, every good text editor lets you select all the lines you wish to indent and

then indent them all as a group. This is a fantastic time saver!

Here's how various text editors enable you to indent lines at a group:

- Notepad++: Highlight the lines & press TAB. To backdent the lines, press Shift-TAB.

## Linking HTML pages with CSS page

This short tutorial is meant for people who want to start using CSS and have never written different a CSS style sheet before.

It does not explain much of CSS. It just explains how to create an HTML and CSS file and how to make them work together. After that, you can read any of a number of other tutorials to add more features to the HTML and CSS files. Or you can switch to using a dedicated HTML or CSS editor, that helps you set up complex sites.

**STEP 1: Writing the HTML**

For this tutorial, it is suggested you use only the very simplest of tools. E.g., Notepad++ (under Windows), atom (on the linux) will do fine. But for your very first CSS style sheet, it is good not to be distracted by too many advanced features.

Step1 is to open your text editor and start creating simple HTML file.

For example it can be started with an empty editor window and type the following:

```
<!DOCTYPE html >
<html>
<head>
  <title>My first styled page</title>
</head>
<body>
<!-- Site navigation menu -->
<ul class="navbar">
  <li><a href="index.html">Home page</a>
  <li><a href="musings.html">Musings</a>
  <li><a href="town.html">My town</a>
  <li><a href="links.html">Links</a>
</ul>
<!-- Main content -->
<h1>My first styled page</h1>
<p>
        Welcome to my styled page!
</p>
<p>
        It lacks images, but at least it has style.
</p>
        And it has links, even if they don't go
        anywhere&hellip;
<p>
        There should be more here, but I don't know
        what yet.
</p>
```

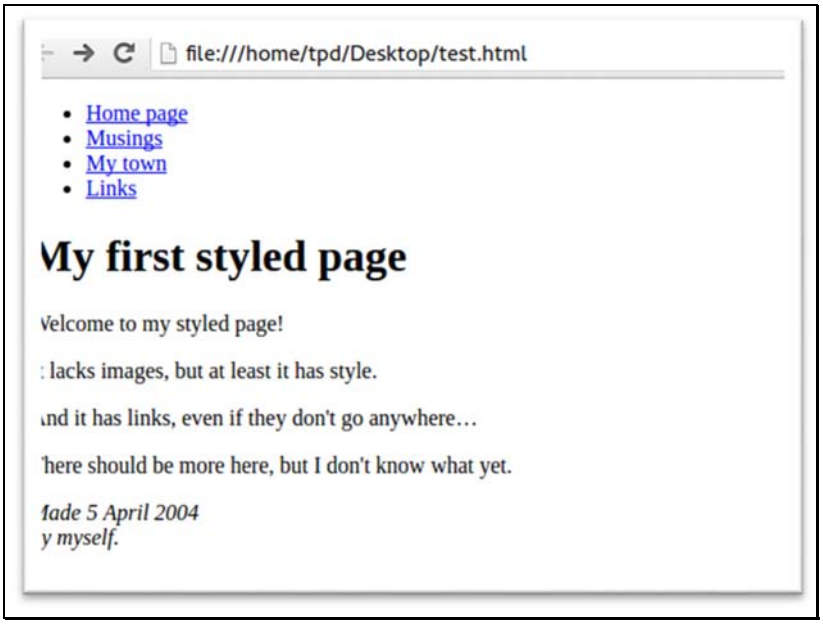
```
<!-- Sign and date the page, it's only polite! -->
<address>Made 5 April 2004<br>
  by myself.</address>
</body>
</html>
```

The editor showing the HTML source.

Now select “Save As…” from the File menu, navigate to a directory/folder where you want to put it (the Desktop is fine) and save the file as “mypage.html”. Don't close the editor yet, we will need it again.

**Viewing the HTML:** Open the file in a browser. You can do that as follows: find the file with your file manager (Windows Explorer, Chrome, Firefox) and click or double click the “mypage.html” file. It should open in your default Web browser. (If it does not, open your browser and drag the file to it.)

As you can see, the page looks rather boring…




**Adding more text formatting option:**

You probably see some black text on a white background, but it depends on how the browser is configured. So one easy thing we can do to make the page more stylish is to add some colors. (Leave the browser open, we will use it again later.)

We will start with a style sheet embedded inside the HTML file. Later, we will put the HTML and the CSS in separate files. Separate files is good, since it makes it easier to use the same style sheet for multiple HTML files: you only have to write the style sheet once. But for this step, we just keep everything in one file.

We need to add a <style> element to the HTML file(Outside <body>tag). The style sheet will be inside that element. So go back to the editor window and add the following five lines in the head part of the HTML file.

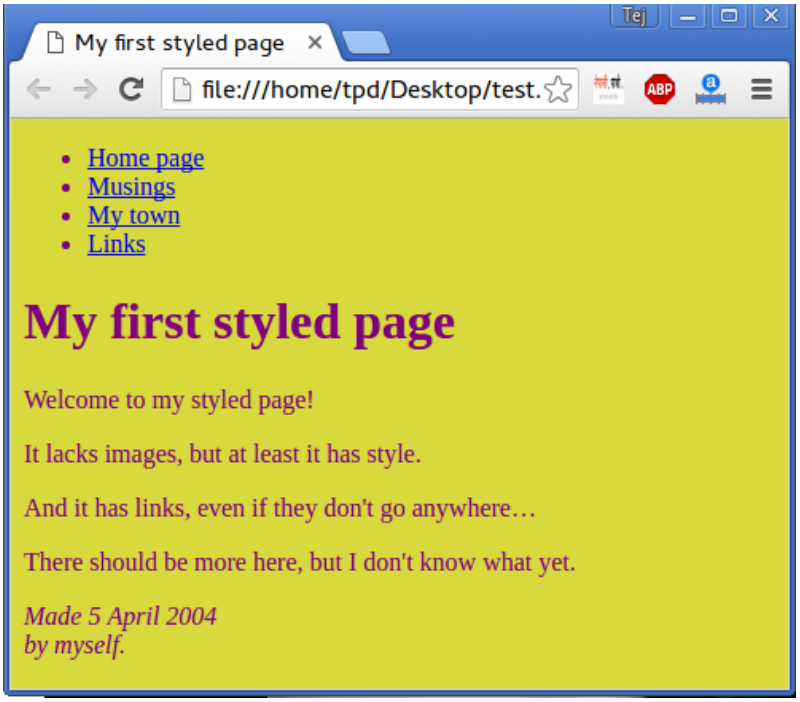
```
<style type="text/css">
body {
  color: purple;
  background-color: #d8da3d }
</style>
```

The first line says that this is a style sheet and that it is written in CSS (“text/css”).

The second line says that we add style to the “body” element. The third line sets the color of the text to purple and the next line sets the background to a sort of greenish yellow.

Style sheets in CSS are made up of rules which is already been studied in earlier chapter.

Now save this file (use “Save” from the File menu) and go back to the browser window. If you press the “Reload” button, the display should change from the “boring” page to a colored (but still rather boring) page. Apart from the list of links at the top, the text should now be purple against a greenish yellow background.




How one browser shows the page now that some colors have been added.

Colors can be specified in CSS in several ways. This example shows two of them: by name (“purple”) and by hexadecimal code (“#d8da3d”)

**STEP 3: ADDING FONTS**

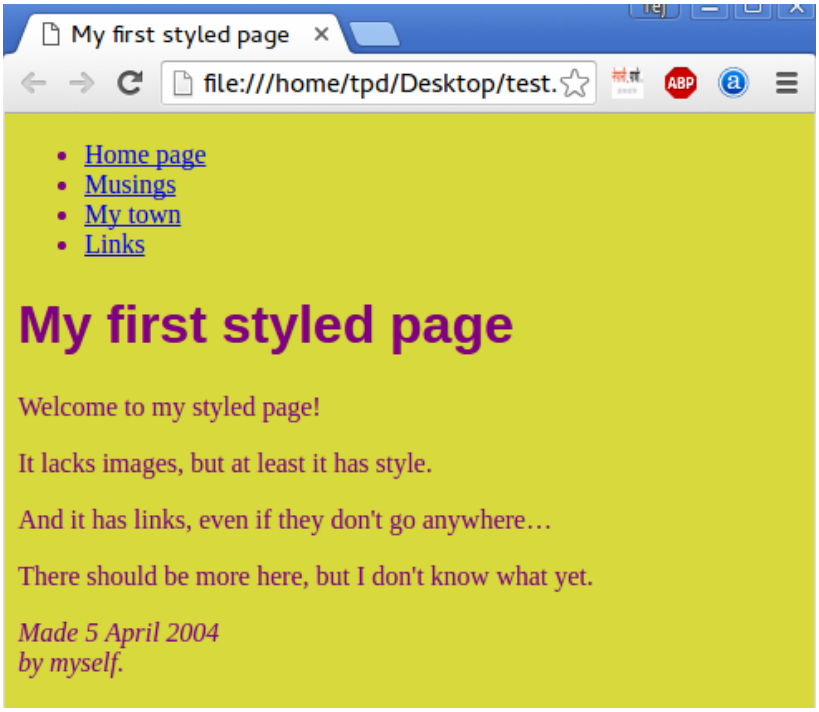
Another thing that is easy to do is to make some distinction in the fonts for the various elements of the page. So let's set the text in the “Georgia” font, except for the h1

heading, which we'll give “Helvetica.”

In the text editor add the following lines:

```
<!DOCTYPE html PUBLIC >
<html>
<head>
 <title>My first styled page</title>
 <style type="text/css">
 body {
   font-family: Georgia, "Times New Roman",
        Times, serif;
   color: purple;
   background-color: #d8da3d }
 h1 {
   font-family: Helvetica, Geneva, Arial,
        SunSans-Regular, sans-serif }
 </style>
</head>
<body>
```

If you save the file again and press “Reload” in the browser, there should now be different fonts for the heading and the other text.




Now the main text has a different font from the heading.

**STEP 4: ADDING A NAVIGATION BAR**

The list at the top of the HTML page is meant to become a navigation menu. Many Web sites have some sort of menu along the top or on the side of the page and this page should have one as well. We will put it on the left side, because that is a little more interesting than at the top.The menu is already in the HTML page. It is the <ul> list at the top. The links in it don't work, since our “Web site” so far consists of only one page, but that doesn't matter now. On a real Web site, there should not be any broken links, of course.

So we need to move the list to the left and move the rest of the text a little to the right, to make room for it. The CSS properties we use for that are 'padding-left' (to move the body text) and 'position', 'left' and 'top' (to move the menu).

There are other ways to do it. If you look for “column” or “layout” on the Learning CSS page, you will find several ready-to-run templates. But this one is OK for our purposes.

In the editor window, add the following lines to the HTML file:

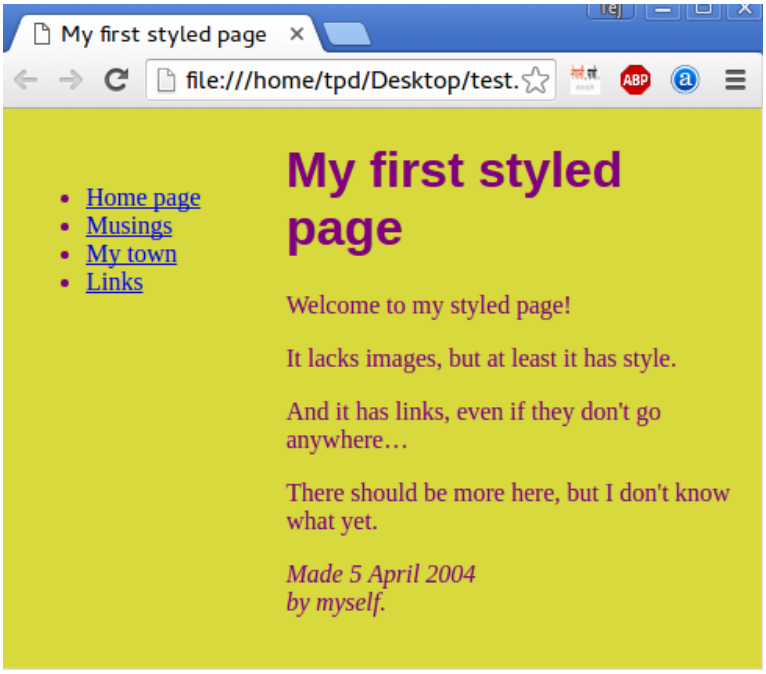
```
<!DOCTYPE html>
<html>
<head>
 <title>My first styled page</title>
 <style type="text/css">
 body {
  padding-left: 11em;
  font-family: Georgia, "Times New Roman",
       Times, serif;
  color: purple;
  background-color: #d8da3d }
 ul.navbar {
  position: absolute;
  top: 2em;
  left: 1em;
  width: 9em }
 h1 {
  font-family: Helvetica, Geneva, Arial,
       SunSans-Regular, sans-serif }
 </style>
</head>
<body>
```

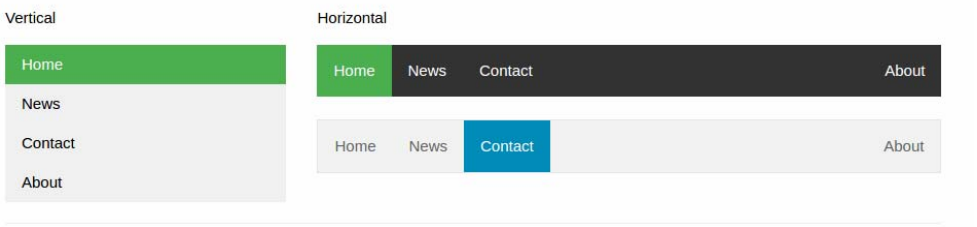
If you save the file again and reload it in the browser, you should now have the list of links to the left of the main text. That already looks much more interesting, doesn't it?

The main text has been moved over to the right and the list of links is now to the left of it, instead of above.

The 'position: absolute' says that the ul element is positioned independently of any text that comes before or after it in the document and the 'left' and 'top' indicate what that position is. In this case, 2em from the top and 1em from the left side of the window.

'2em' means 2 times the size of the current font. E.g., if the menu is displayed with a font of 12




points, then '2em' is 24 points. The 'em' is a very useful unit in CSS, since it can adapt automatically to the font that the reader happens to use. Most browsers have a menu for increasing or decreasing the font size: you can try it and see that the menu increases in size as the font increases, which would not have been the case, if we had used a size in pixels instead.

## PUTTING THE STYLE SHEET IN A SEPARATE FILE

We now have an HTML file with an embedded style sheet. But if our site grows we probably want many pages to share the same style. There is a better method than copying the style sheet into every page: if we put the style sheet in a separate file, all pages can point to it.

To make a style sheet file, we need to create another empty text file. You can choose “New” from the File menu in the editor, to create an empty window. (If you are using

TextEdit, don't forget to make it plain text again, using the Format menu.)

Then cut and paste everything that is inside the <style> element from the HTML file into the new window. Don't copy the <style> and </style> themselves. They belong to HTML, not to CSS. In the new editor window, you should now have the complete style sheet:

```
body {
  padding-left: 11em;
  font-family: Georgia, "Times New Roman",
       Times, serif;
  color: purple;
  background-color: #d8da3d }
ul.navbar {
  list-style-type: none;
  padding: 0;
  margin: 0;
  position: absolute;
  top: 2em;
  left: 1em;
  width: 9em }
h1 {
  font-family: Helvetica, Geneva, Arial,
       SunSans-Regular, sans-serif }
ul.navbar li {
  background: white;
  margin: 0.5em 0;
  padding: 0.3em;
  border-right: 1em solid black }
ul.navbar a {
  text-decoration: none }
a:link {
  color: blue }
a:visited {
  color: purple }
address {
```

```
margin-top: 1em;
padding-top: 1em;
border-top: thin dotted }
```

Choose “Save As…” from the File menu, make sure that you are in the same directory/folder as the mypage.html file, and save the style sheet as “mystyle.css”.(CSS File should be save as in .css) file extension.

Now go back to the window with the HTML code. Remove everything from the <style> tag up to and including the </style> tag and replace it with a <link> element, as follows:

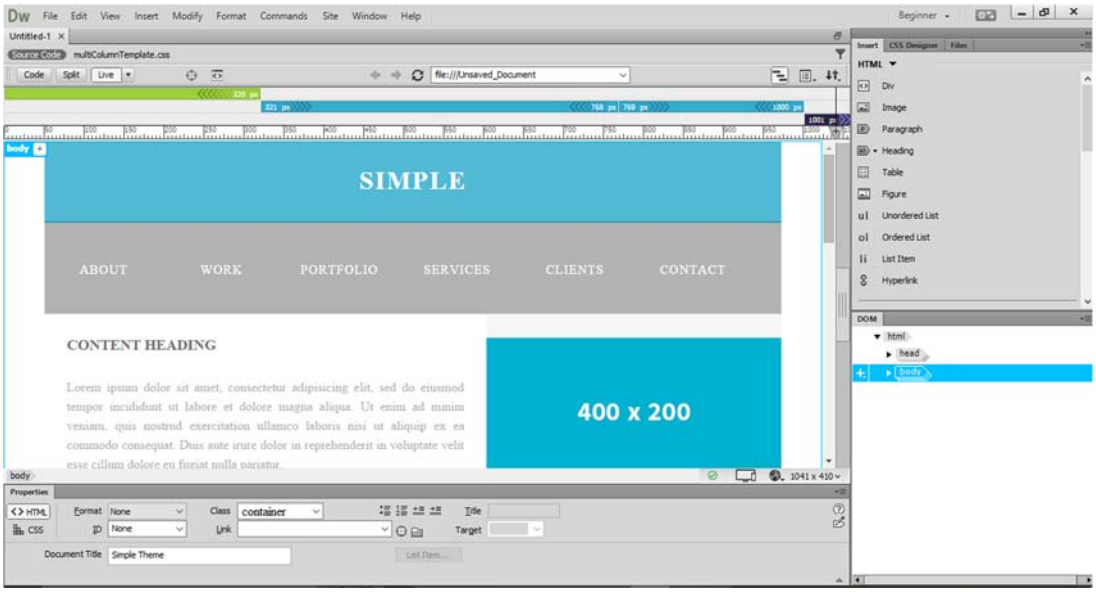
```
<!DOCTYPE html >
<html>
<head>
  <title>My first styled page</title>
  <link rel="stylesheet" href="mystyle.css">
</head>
<body>
</body>
        …….
</html>
```

This will tell the browser that the style sheet is found in the file called “mystyle.css” and since no directory is mentioned, the browser will look in the same directory where it found the HTML file.

If you save the HTML file and reload it in the browser, you should see no change: the page is still styled the same way, but now the style comes from an external file.

**Managing the website in the editor**

Total project(website) can be managed using editor. Different editor can be used to manage the web project but GUI based editor is described to manage website. Dreamweaver is one of the best GUI based Editor to manage project by importing project folder in editor. So that each file can be edited side by side. folder structure can be seen on the editor. We can see different sub folder, html, css and images files. So it will be easy to edit multiple folder in the same window.




All the files can be opened in different tabs in the same windows so that coding is very easy.

Multiple files can be opened using editor in multiple tabs. Although this feature is available in latest and non traditional editor only. We can edit css file and html file and we can see the changes live in the same window.

While making a website project, Structure of the files and folder plays important role. For example all the css file should be saved in the css folder and html files in different folder like wise all images that are used in the website should be placed in the other folder. So that the link is given while calling these files and images.

HTML files: Html files contains the different pages that can be seen in website projects. Clicking on different link provided in the website, different pages can be seen so all those pages are managed in the html files.(If we used different framework then this cant be work).

**CSS files:** All the css files are managed in side the css folder. And path of particular used css files is given to the html file.

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

The <link> tag defines a link between a document and an external resource.

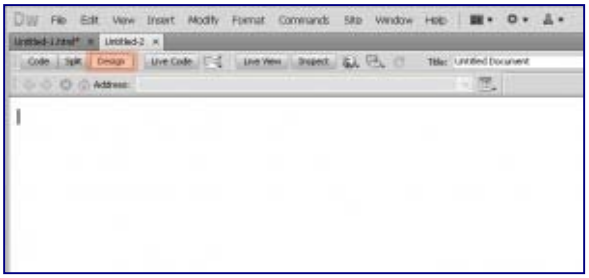
The <link> tag is used to link to external style sheets. “rel” is required to and Specifies the relationship between the current document and the linked document. “type” specifies the type of the document that is linked. And in “href” we have to mention the name of the css file and its path.

Images: Images folder is made so that all the images are placed in the same folder. And is called in different places according to the need.

<img src="html5.gif" style="width:128px;height:128px;">

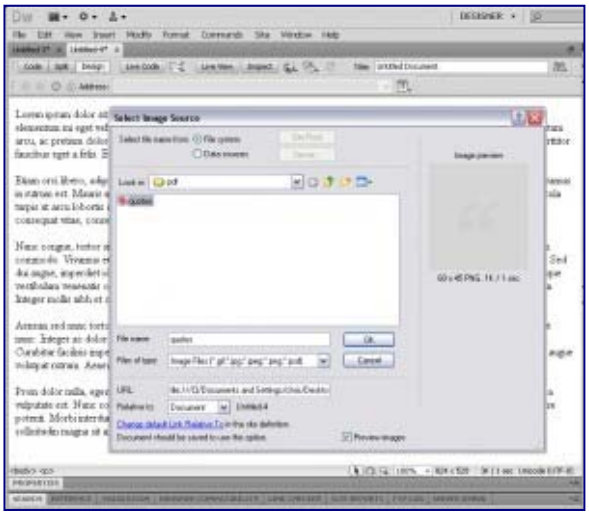
This tutorial shows how to add images in HTML editor.Design Mode

1. Make sure you are in Design mode. Click Design button in the top left corner of your screen



2. Put cursor where you want to insert an image.


3. From the top menu select Insert > Image
4. In the appeared window select the image from your computer and click OK.

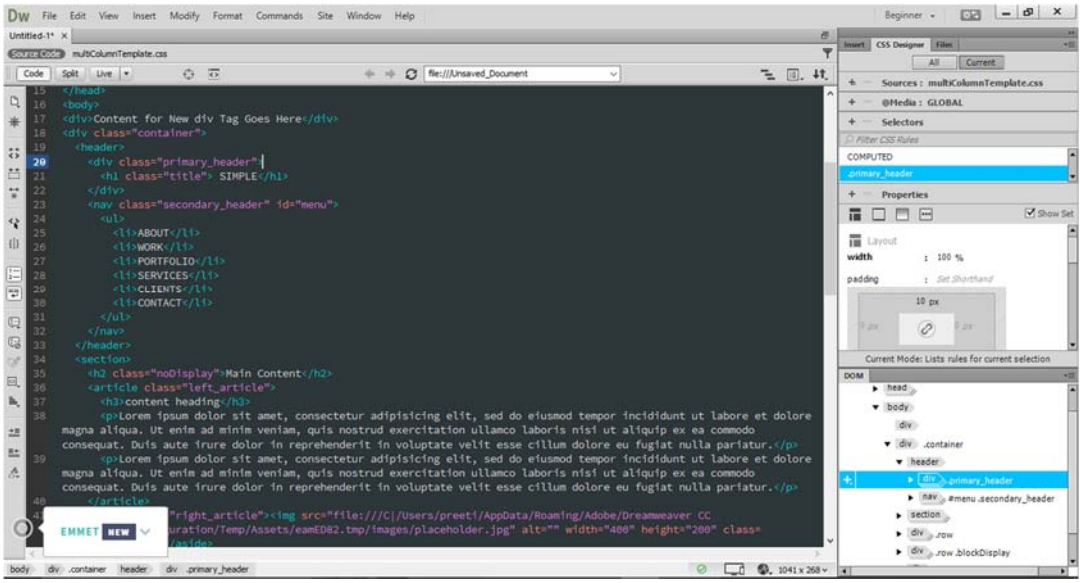


As a result you will see an image added to your page.

To avoid errors please make sure:

- Image should be on the same drive with your HTML page file
- We recommend to put an image into the **'images'** directory located in the same directory with the HTML file
- If you upload images and HTML files to your hosting server – make sure to keep same files structure

the above is the example of the image inserted in the particular div or place. "src" is the source file name that is the image name that has to be inserted. Style can be given in the same tag according to the need of the image size.




Using GUI based editor is very easy to manage and write code. In the above screen shot, we can see the different body,div clearly in the same window. And by clicking on that particular div, code can be edited easily.

**Editing HTML and CSS templates using editor**

A template is a tool for enforcing a standard layout and look and feel across multiple pages or within content regions. When you change a template, any pages or regions that are based on that template are automatically changed as well. Templates provide additional standardization controls, depending on the type you use.

With HTML page skins, you define where the body area of the page is placed. The body area is where portal content displays, that is, the regions, tabs, items, and portlets.

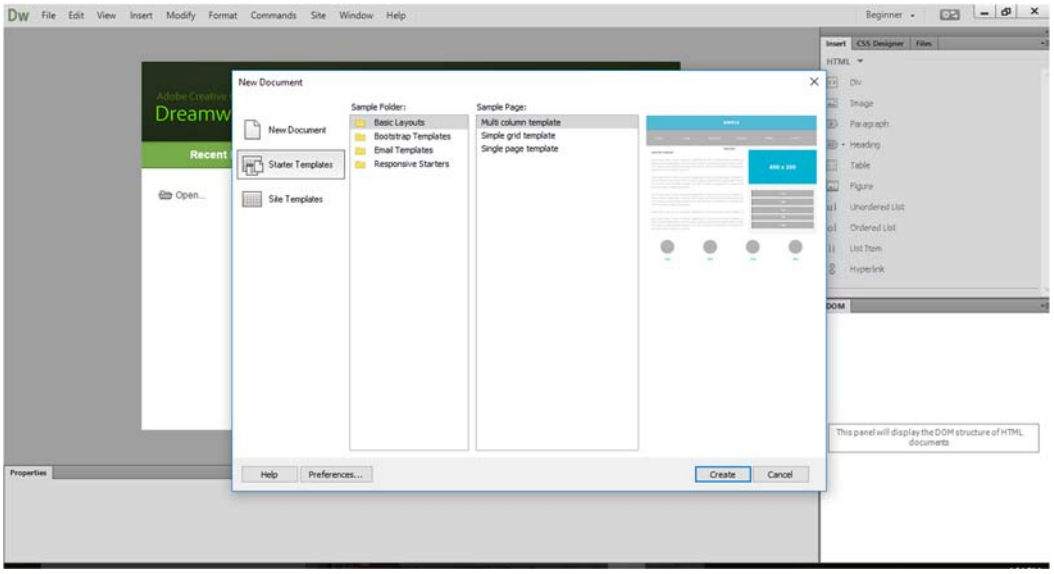
**HTML Content Layouts**

Use HTML Content Layouts to define a formatting scheme for individual regions. Use HTML to create tables, font designs, colors, and any other encoding or object type you could place in any other HTML template. You could even call a cascading style sheet (CSS) to apply a standard format that you use in other forms of company documentation.

Here is shown an example of editing html and css templates. A already developed templates is download(Presentation is the name of templates here) and edited using

html editor Dreamweaver , although any editor can be used to edit.

**Import template folder in html Editor :**




After opening editor, go to the file menu and select “New Document”. After that select the template folder then we can see all the files and sub folder in left side of the editor.

We can see different sub folder and files in side those sub folder which can be editable. But in template we have to edit or change only those area so that design, formatting will be the same.

**Change/Edit contents:**

Index.html and other .html files can be opened and edited its contents as our need. Those editable file can be found in the “pages” folder as full-width.html, style.html ect(The name can be different according to template).

Files can be edited easily in the editor. In below screen shot, we can clearly see the different tag and div so clicking on those tag and div the contents can easily be changed.

**Open a template for editing**

You can open a template file directly for editing, or you can open a template-based document, then open the attached template for editing.

When you make a change to a template, Dreamweaver prompts you to update the documents based on the template.

**Open and edit a template file**

- In the Assets panel (Window> Assets), select the Templates category on the left side of the panel.
- The Assets panel lists all of the available templates for your site and displays a preview of the selected template.
- In the list of available templates, do one of the following:
- Double-click the name of the template you want to edit.
- Select a template to edit, then click the Edit button at the bottom of the Assets panel.
- Modify the contents of the template.

To modify the template's page properties, select Modify > Page Properties. (Documents based on a template inherit the template's page properties.)

- Save the template. Dreamweaver prompts you to update pages based on the template.
- Click Update to update all documents based on the modified template; click Don't Update if you don't want to update documents based on the modified template.

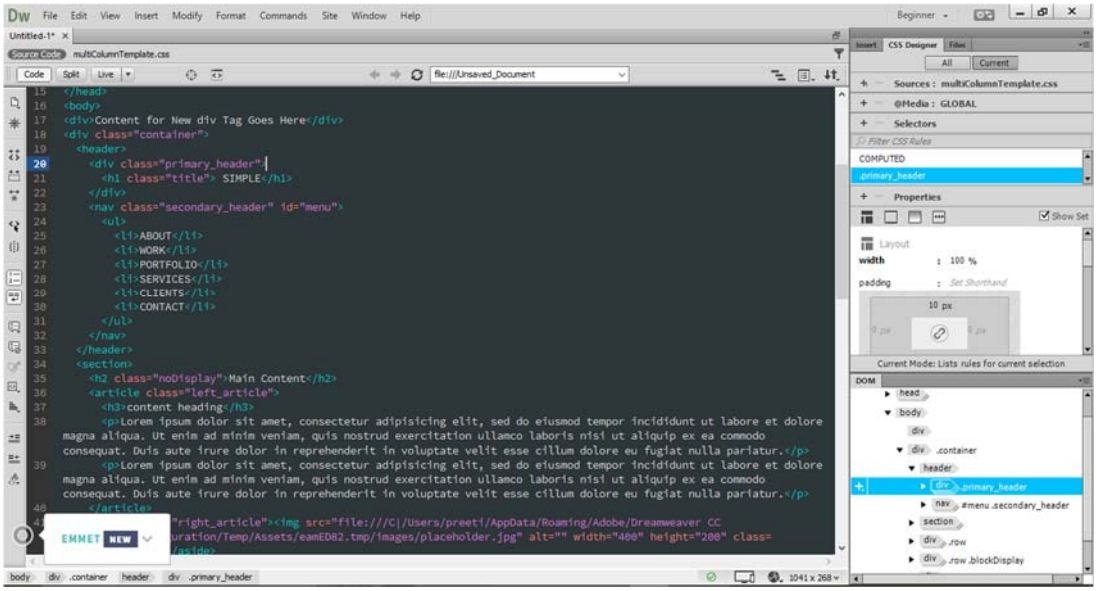
Dreamweaver displays a log indicating the files that were updated.

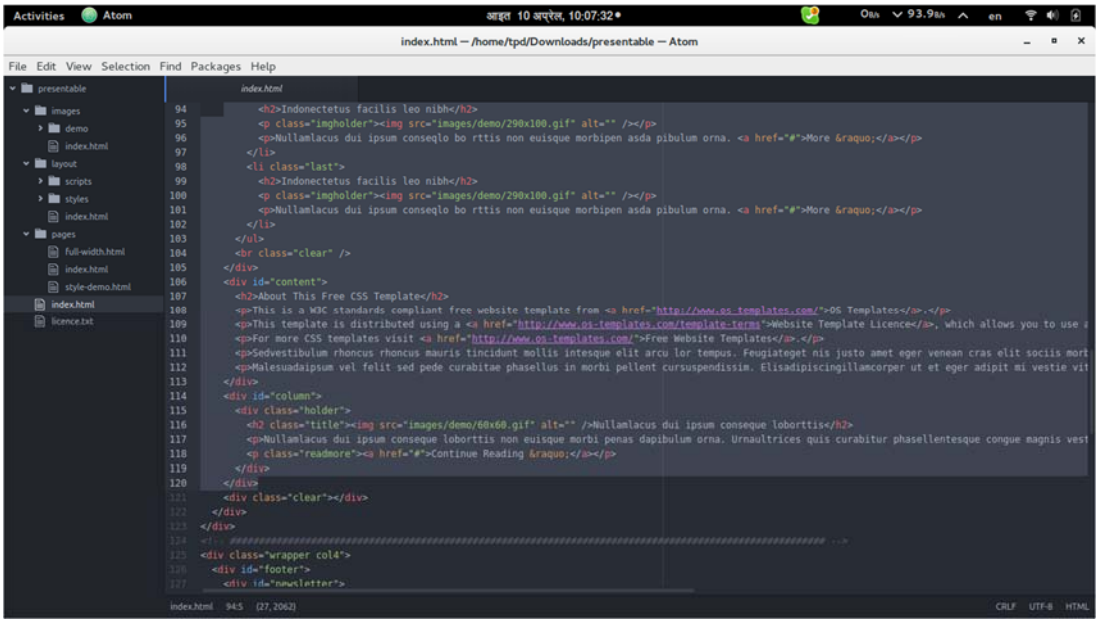
- Open and modify the template attached to the current document
- Open the template-based document in the Document window.
- Do one of the following:
- Select Modify> Templates> Open Attached Template.
- Right-click (Windows) or Control-click (Macintosh), then select Templates> Open Attached Template.
- Modify the contents of the template.

To modify the template's page properties, select Modify> Page Properties. (Documents based on a template inherit the template's page properties.)

- Save the template. Dreamweaver prompts you to update pages based on the template.

- Click Update to update all documents based on the modified template; click Don't Update if you don't want to update documents based on the modified template. Dreamweaver displays a log indicating the files that were updated.





Screen shot of Editing contents of html templates in ATOM editor.

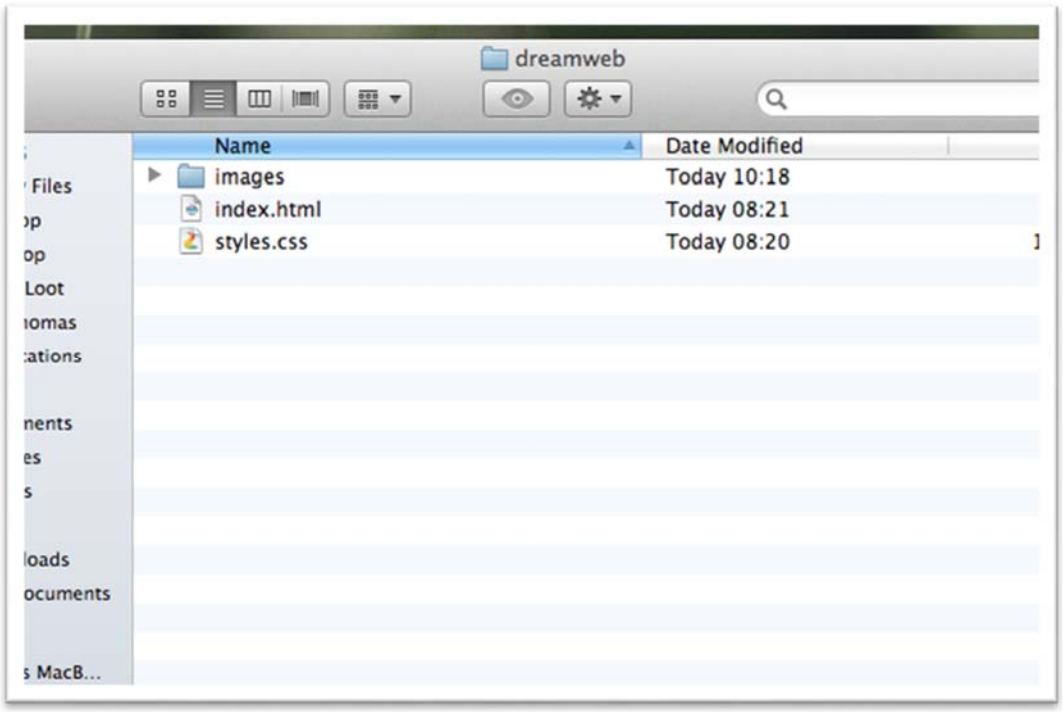
**Create a simple HTML and CSS templates**

When it comes to coding up designs, there is a lot of personal preference, everybody has their own way of doing things, some people like to jump right in and juggle writing HTML, CSS and exporting images from the layout as needed all at the same time (this is a surprisingly common method actually), but some people break it down into stages, for example: export images, write HTML and then style with CSS, but personally I don't think that we always know which images we will need to extract from the design until we have broken down the layout into a HTML structure. so my preferred workflow goes something like:

- Plan structure and write HTML code
- Export images from the layout
- Style the HTML with CSS

So, with that in mind, let's begin by writing our HTML.

Start by creating a new folder and index.html and styles.css files. Then also create a new folder called images.



Open up the index.html file in a text editor, Notepad++ or Dreamweaver.

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8" />
        <title>Dreamweb</title>
        <link rel="stylesheet" href="styles.css" type="text/css">
        <link href='...' rel='stylesheet' type='text/css'>
        <!--[if IE]>
        <script src="….."></script>
        <![endif]-->
    </head>
    <body>
    </body>
</html>
```

Begin by inserting the basic HTML skeleton, we are using HTML for this project so the DOCTYPE declaration is a very simple one, just “html”. You will also notice that we have set the title of the page, included our stylesheet (styles.css).

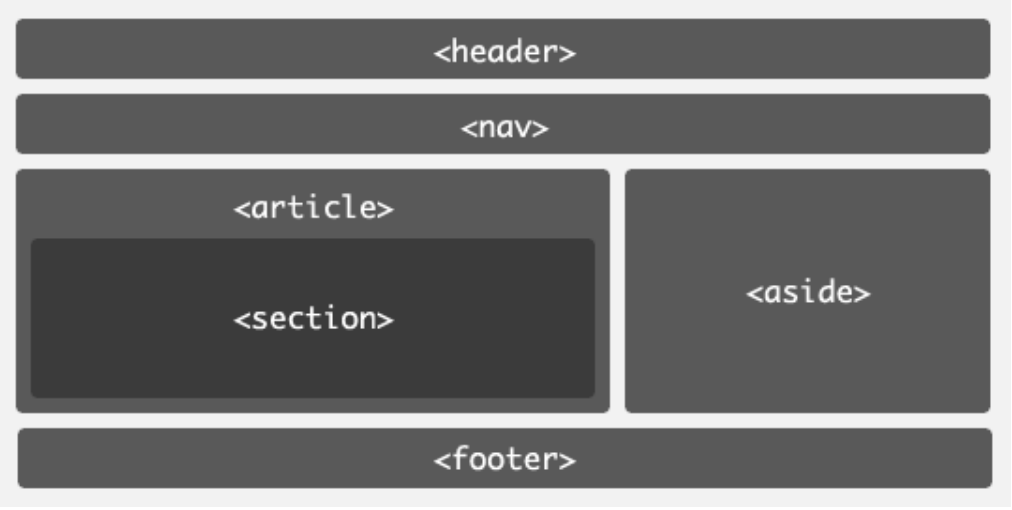



**Fig:** Above screen shot is the example that should be include while making own templete

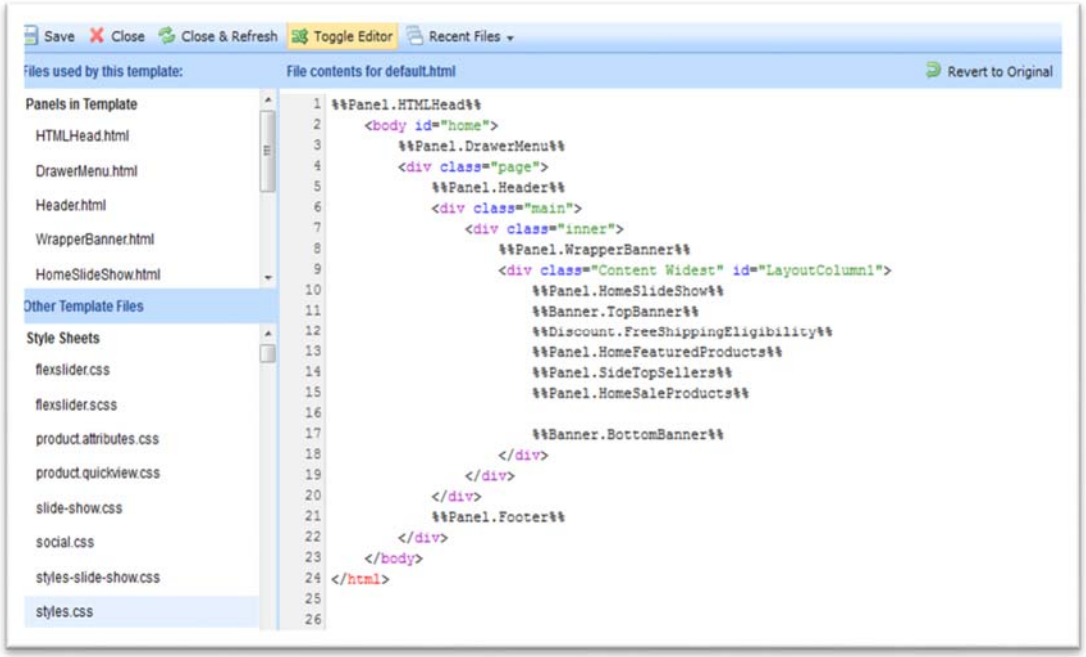



**fig:** Screen shot for the template files that should be include while making own templetes.

Below is the example of peace of that deals how to make own templates.(This is just for example.)

HEADER

```
<header>
    <div class="wrapper">
        <h1 class="logo"><a href="index.html">Dreamweb</a></h1>
        <ul>
            <li class="active"><a href="index.html">Home</a></li>
            <li><a href="#">Portfolio</a></li>
            <li><a href="#">About Us</a></li>
            <li><a href="#">Blog</a></li>
        </ul>
    </div><!--.wrapper-->
</header>
```

Define the header tag, and then place a wrapper inside inside it so that the header can be 100% wide, but the content will be wrapped to a set width (960px), add the logo as a H1 and create an unordered list (UL) for the main menu.

```
<section class="home_feature">
    <div class="wrapper">
        <article class="main_display">
            <img src="images/display_image_1.jpg" alt="Client Work"
title="Client Work" />
            <div class="display_gloss"></div>
            <article class="banner_new"><p>New</p></article>
        </article>
        <article class="feature_text">
            <h2>Vivamus sagittis lacus vel augue laoreet rutrum faucibus
auctor.</h2>
            <p>Cras ju Maecenas faucibus mollis posuere velit aliquet
interdum.</p>
            <a href="#">Click me to find out more</a>
        </article>
    </div><!--.wrapper-->
</section>
```

Use the same wrapper technique from the header for the main feature, we are using a <section> to contain the two <article>s in this part of layout. The feature text is quite straight forward, we have an H2, some paragraph text and a link. The main_display article will be our computer screen, and then we are layering on top of that:

- the image that will be on the screen
- followed by the gloss from the screen
- and finally the banner in the corner than reads “NEW”.

**Featured Services**

```
<ul class="featured_services">
    <li class="service_1">
        <Web Design&rt;
        <p>Cras justo odio, dapibus ac facilisis in, egestas eget quam mollis
interdum.</p>
```

```
            <a href="#" class="button_1">Find Out More</a>
        </li>
        <li class="service_2">
            <Development&rt;
            <p>Cras justo odio, dapibus ac facilisis in, egestas eget m. Integer
posuere erat</p>
            <a href="#" class="button_1">Find Out More</a>
        </li>
        <li class="service_3">
            <Optimization&rt;
            <p>Cras a ante venenatis dapibus posuere maecenas faucibus mollis
interdum.</p>
            <a href="#" class="button_1">Find Out More</a>
        </li>
</ul>
```

The featured services section of the layout is a very basic unordered list, we have put a different class on each list item so that it can be targeted and styled appropriately. This section doesn't need the wrapper div because there is no defined background, we can just set the <ul&rt; to 960px wide in the CSS.

```
About Us and Testimonials
<div class="wrapper">
            <article class="about_us_blurb">
                <About Us&rt;
                <img src="images/about_us.png" />
                <p> arturient montes, nascetur ridiculu.</p>
                <p>s. Lorem ipsum dolor sit amet, consectetur adipiscing
elit.</p>
            </article>
            <article class="testimonials">
                <Testimonials&rt;
                <div class="testimonial_wrapper">
                    <p class="testimonial_quote">
                        Donec sed odio dui. Aenean lacinia bibendum
```

```
consectetur.
                        </p>
                            <p class="testimonial_name">John Doe,
Company</p>
                    </div>
                </article>
            </div><!--.wrapper-->
```

Here we have created 2 articles for the about us blurb and testimonials, for the testimonials we have a wrapper containing both the quote and name.

**Footer**

```
        <footer>
            <div class="wrapper">
<!--Twitter Widget-->
                <div class="column">
                    <article class="widget_twitter">
                        <h4>Twitter Updates</h4>
                        <ul>
                                <li><strong>@MediaLoot</strong>
Cur venenatis ultricies.</li>
                                <li><strong>@MediaLoot</strong>
Cur venenatis ultricies.</li>

                        </ul>
                        <a href="#" class="button_1">Read More</a>
                    </article>
                </div>
<!--Newsletter Widget-->
                <div class="column">
                    <article class="widget_newsletter">
                        <h4>Newsletter</h4>
                            <p>Cras justo odi Integer posuere
erat.</p>
                        <form>
```

```
                        <input type="text" value="Email Address">
                        <input type="submit" value="Sign Up">
                    </form>
                </article>
<!--Social Widget-->
                <article class="widget_social">
                    <h4>Social</h4>
                    <ul>
                        <li class="facebook"><a
href="#">FaceBook</a></li>
                        <li class="twitter"><a
href="#">Twitter</a></li>
                        <li class="linkedin"><a href="#">Linked
In</a></li>
                    </ul>
                </article>
            </div>
<!--Feedback Widget-->
            <div class="column">
                <article class="widget_feedback">
                    <h4>Feedback</h4>
                    <form>
                        <input type="text" value="Full Name"
name="name">
                        <input type="text" value="Email Address"
name="email">
                        <textarea name="message">Your
message</textarea>
                        <input type="submit" value="Send Mesge"
class="button_1">
                    </form>
                </article>
            </div>
<!--Copyright Notice-->
```

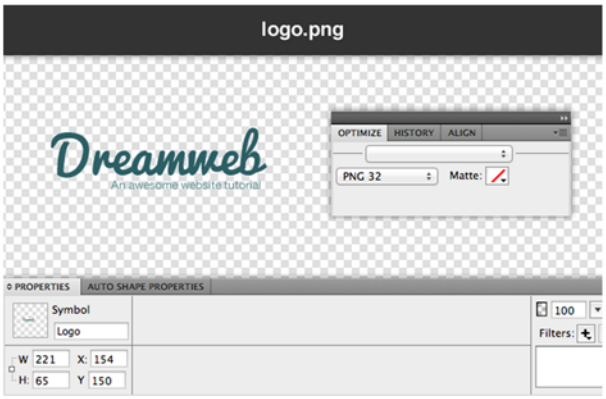
```
                <p class="copyright_notice">Copyright 2011 © <strong>Brand
Name</strong>. All Rights Reserved.</p>
                <div class="logo"><p>Dreamweb</p></div>
            </div><!--.wrapper-->
        </footer><!--End Footer-->
```

This is quite a big one, but the actual code itself is very straight forward, each widget in the footer area has a class that begins 'widget_' and we have 3 s with a 'column' class, the newsletter and social widgets are contained within the same column.

**Images**

Let's prepare the images we need, in total we need 17 images(it can be any number). Open up the original Fireworks design file if you haven't already and we can begin extracting the images we need. The basic method for extracting images is to select the object/s that you need, copy them and then create a new document using the clipboard dimensions (the same size as whatever you copied) and paste the objects into the new document.

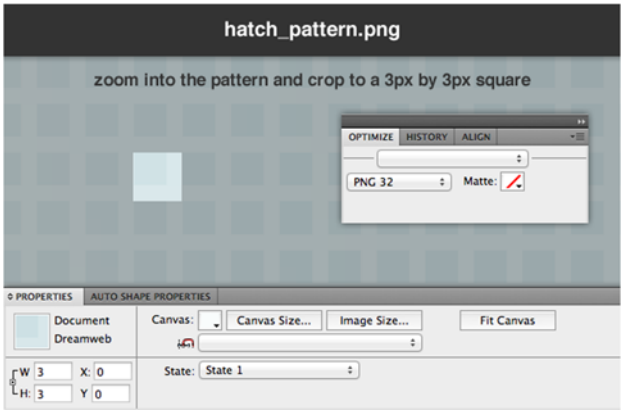
**1. logo.png**




So open up the original Fireworks document if you haven't already and select the Logo symbol. hit CTRL+C to copy it, CTRL+N to create a new document, and CTRL+V to paste the logo again. Set the background of the document to transparent and in the Optimize window select PNG32 as the exporting option. Go to File > Export

and select Images Only and Current Page. Choose the images folder we created as the export destination.

**2. hatch_pattern.png**




Zoom in as far as possible on the design and use the Crop tool in Fireworks to select a 3x3px area of the pattern (as shown in the image below). Export using the same methods as used previously.

3. feature_display.png
4. display_image_1.jpg
5. display_gloss.png
6. btn_main_feature.png
7. service_1.png
8. service_2.png
9. service_3.png
10. hr_shadow.png

**The CSS**

First things first, open up style.css and let's use a basic CSS reset so that all browsers, new and old are working on the same playing field.

```
/* CSS Reset */
body,div,dl,dt,dd,ul,ol,li,h1,h2,h3,h4,h5,h6,pre,form,fieldset,input,p,blockquote,th,td
{margin:0;padding:0;}

table{border-collapse:collapse;border-spacing:0;}
fieldset,img{border:0;}
```

```
ul{ list-style:none; list-style-position:outside;}
a {  outline: none;}
/* Tell old browsers how to handle HTML5 elements */
header, footer, aside, nav, article {display: block;}
```

Now we will do the essential styles, these are the body, headings, paragraph, link and wrapper styles. These tend to stay the same for most web projects and are usually the first things defined in a stylesheet.

```
/* Essentials */
body {
    background: #F9FCFC;
    color: #666666;
    font-family: "Helvetica Neue", Helvetica, Arial, sans-serif;
    font-size: 12px;
}
p {
    line-height: 150%;
}
h1, h2, h3, h4, h5, h6 {
    font-family: Pacifico, "Helveitca Neue", Helvetica, Arial, sans-serif;
    font-weight: lighter; /*counteract strong browser anti-aliasing*/
}
a:link, a:visited {
    color: #168FAD;
    text-decoration: none;
}
.wrapper {
    width: 960px;
    margin: 0 auto;
}
```

For the header we only really need to set the height and background, thanks to CSS3 we no longer need to use images for gradient backgrounds, we can just set the start and end points and type of gradient, even IE6 supports basic gradients with filters.

```
/* Header */
header {
    height: 110px;
}
header {
    /* Fallback Color */
    background: #F4F8F9;
    /* Firefox 3.6+ */
    background-image: -moz-linear-gradient(#FFFFFF, #E9F2F3);
    /* Safari 4+, Chrome 1+ */
    background-image: -webkit-gradient(linear, 0% 0%, 0% 100%,
from(#FFFFFF), to(#E9F2F3));
    /* Safari 5.1+, Chrome 10+ */
    background-image: -webkit-linear-gradient(#FFFFFF, #E9F2F3);
    /* IE */
    filter:
progid:DXImageTransform.Microsoft.gradient(startColorstr='#FFFFFF',
endColorstr='#E9F2F3');
}
```

Here we are targeting the <a> link inside the <h1> with a class of ‘logo’. Note that as well as bringing in the logo.png file, we are hiding the text from visibility by setting the text-indent to -5000px. The reason for adding text and then hiding it is two-fold, it helps with your Search Engine Optimization and also boosts accessibility.

```
/* Logo */
h1.logo a {
    display: block;
    width: 221px;
    height: 65px;
    background: url('images/logo.png') no-repeat;
    float: left;
    margin: 20px 0 0 10px;
    text-indent: -5000px;
}
```

Time to style the main navigation, here is used some new CSS3 properties in this section including border-radius and box-shadow, these are both pretty much safe to use now for non-essential visual elements, in modern browsers, the vendor prefixes (i.e.. -moz-) allow you target particular browsers, -moz- for Mozzila (Firefox), -webkit- for WebKit (Safari and Chrome), but when the HTML5 and CSS3 specs are complete, vendor prefixes will not be required anymore, so for good measure include the non-prefixed version too, and put it at the end of the list so that it is applied last.

```
/* Main Navigation */
header ul {
	float: right;
	margin: 35px 0;
}
header li {
	float: left;
	display: inline-block;
	width: 90px;
	height: 36px;
	margin: 0 10px 0 0;
}
header li a {
	color: #2C6069;
	display: block;
	width: 90px;
	height: 36px;
	text-align: center;
	font-weight: bold;
	line-height: 36px;
	border: 1px solid #CDE0E4;
	-moz-border-radius: 5px; -webkit-border-radius: 5px; border-radius: 5px;
	-moz-box-shadow: 0 1px 0 #FFFFFF; -webkit-box-shadow: 0 1px 0
#FFFFFF; box-shadow: 0 1px 0 #FFFFFF;
}
header li a {
	/* Fallback Color */
```

```
        background: #E8F4F6;
        background-image: -moz-linear-gradient(#FFFFFF, #DFF0F3);
        background-image: -webkit-gradient(linear, 0% 0%, 0% 100%,
    from(#FFFFFF), to(#DFF0F3));
        background-image: -webkit-linear-gradient(#FFFFFF, #DFF0F3);
    }
    header li a:hover {
        /* Fallback Color */
        background: #FFFFFF;
        background-image: -moz-linear-gradient(#FFFFFF, #E8F4F6);
        background-image: -webkit-gradient(linear, 0% 0%, 0% 100%,
    from(#FFFFFF), to(#E8F4F6));
        background-image: -webkit-linear-gradient(#FFFFFF, #E8F4F6);
        filter:
    progid:DXImageTransform.Microsoft.gradient(startColorstr='#FFFFFF',
    endColorstr='#E8F4F6');}
    header li.active a {
        color: #5E8D94;
        -moz-box-shadow: inset 0 2px 3px #436E7D; -webkit-box-shadow:inset
    0 2px 3px #436E7D; box-shadow: inset 0 2px 3px #436E7D;
    }
    header li.active a {
        /* Fallback Color */
        background: #DFF0F3;
        background-image: -moz-linear-gradient(#DFF0F3, #FFFFFF);
        background-image: -webkit-gradient(linear, 0% 0%, 0% 100%,
    from(#DFF0F3), to(#FFFFFF));
        background-image: -webkit-linear-gradient(#DFF0F3, #FFFFFF);
    }
```

Here is the style used for the main feature, we layer up the computer screen, image on the screen, gloss from the screen and banner separately using ‘position: relative’ and ‘position:absolute’ properties.

```
/* Main Feature */
section.home_feature {
	height: 480px;
	background: #CDE0E4 url('images/hatch_pattern.png') repeat;
	-moz-box-shadow: inset 0 5px 9px #66888E; -webkit-box-shadow: inset
0 5px 9px #66888E; box-shadow: inset 0 5px 9px #66888E;
}
article.main_display {
	display: block;
	float: left;
	position: relative;
	width: 450px;
	height: 397px;
	margin: 70px 0;
	background: url('images/feature_display.png') no-repeat;
}
article.main_display img {
	position: absolute;
	top: 23px;
	left: 19px;
}
article.banner_new {
	position: absolute;
	left: -6px;
	top: -6px;
	background: url('images/feature_banner.png') no-repeat;
	width: 103px;
	height: 103px;
}
article.banner_new p {
	text-indent: -5000px;
}
article.main_display .display_gloss {
	position: absolute;
	top: 2px;
```

```
        right: 2px;
        width: 269px;
        height: 283px;
        background: url('images/display_gloss.png') no-repeat;
    }
    article.feature_text {
        float: right;
        width: 450px;
        margin: 60px 10px 0 0;
    }
    article.feature_text h2 {
        font-size: 44px;
        color: #2C5F66;
        line-height: 120%;
        margin: 0 0 20px 0;
        text-shadow: 0 1px 0 #F1F7F8;
    }
    article.feature_text p {
        color: #5E8C92;
        text-shadow: 0 1px 0 #F1F7F8;
    }
    article.feature_text a {
        display: block;
        width: 310px;
        height: 62px;
        color: #FFFFFF;
        background: url('images/btn_main_feature.png') no-repeat;
        margin: 20px 0 0 0;
        text-align: center;
        line-height: 54px;
        font-family: Pacifico, "Helveitca Neue", Helvetica, Arial, sans-serif;
        font-weight: lighter; /*counteract strong browser anti-aliasing*/
        font-size: 22px;
        text-shadow: 0 -1px 0 #38474A;
    }
```

```
article.feature_text a:hover {
    background-position: 0 -63px;
    line-height: 57px;
}
article.feature_text a:active {
    color: #EFEFEF;
    background-position: 0 -126px;
    line-height: 58px;
}
```

The featured services features the first instance of our generic button class ‘button_1’ that we will use multiple times though out the rest of the design.

```
/* Featured Services */
ul.featured_services {
    clear: both;
    height: 240px;
    width: 960px;
    margin: 0 auto;
    background: url('images/hr_shadow.png') no-repeat bottom;
}
ul.featured_services li {
    width: 300px;
    margin: 0 10px;
    float: left;
}
ul.featured_services h3 {
    font-size: 25px;
    color: #168FAD;
    text-indent: 40px;
}
ul.featured_services li.service_1 h3 {
    background: url('images/icn_service_1.png') no-repeat; background-
position: 0 10px;
}
ul.featured_services li.service_2 h3 {
```

```
        background: url('images/icn_service_2.png') no-repeat; background-
position: 0 10px;
}
ul.featured_services li.service_3 h3{
        background: url('images/icn_service_3.png') no-repeat; background-
position: 0 10px;
}
ul.featured_services p {
        border-left: 3px solid #CDE0E4;
        margin: 10px 0 10px 15px;
        padding: 0 0 0 25px;
}
.button_1 {
        display: block;
        width: 133px;
        height: 28px;
        -moz-border-radius: 50px; -webkit-border-radius: 50px; border-radius:
50px;
        text-align: center;
        line-height: 28px;
        border: 1px solid #ADC0C4;
        color: #5E8C92;
        font-weight: bold;
        font-size: 12px;
        cursor: pointer;
}
.button_1 {
        /* Fallback Color */
        background: #DFF0F3;
        background-image: -moz-linear-gradient(#FFFFFF, #EFEFEF);
        background-image: -webkit-gradient(linear, 0% 0%, 0% 100%,
from(#FFFFFF), to(#EFEFEF));
        background-image: -webkit-linear-gradient(#FFFFFF, #EFEFEF);
}
.button_1:hover {
```

```
        background: #FFFFFF;
}
.button_1:active {
        background: #EFEFEF;
}
Styles for the About Us blurb
/* About Us Blurb */
article.about_us_blurb {
        clear: both;
        float: left;
        width: 620px;
        margin: 20px 10px;
}
article.about_us_blurb img {
        float: left;
        margin: 0 10px 0 0;
}
article.about_us_blurb h3 {
        font-size: 25px;
        color: #168FAD;
        margin: 10px 0;
}
article.about_us_blurb p {
        margin: 5px 20px 30px 10px;
}
article.testimonials {
        float: right;
        width: 300px;
        margin: 20px 10px;
}
Styles for the testimonials
/* Testimonials */
.testimonial_wrapper {
        background: #E7F4F6;
        border: 1px solid #CDE0E4;
```

```
        width: 298px;
        height: 150px;
        -moz-border-radius: 5px; -webkit-border-radius: 5px; border-radius: 5px;
}
article.testimonials h3 {
        font-size: 25px;
        color: #168FAD;
        margin: 10px 0;
}
p.testimonial_quote {
        text-align: center;
        background: #FFFFFF;
        color: #5E8C92;
        font-style: italic;
        height: 55px;
        -moz-border-radius: 5px; -webkit-border-radius: 5px; border-radius: 5px;
        padding: 30px 20px;
}
p.testimonial_name {
        line-height: 36px;
        font-weight: bold;
        text-align: center;
        color: #5E8C92;
}
The footer and it's widgets
/* Footer */
footer {
        clear: both;
        height: 400px;
        background: #CDE0E4 url('images/hatch_pattern.png') repeat;
        color: #5E8C92;
}
footer h4 {
        font-size: 25px;
        color: #2C5F66;
```

```
        margin: 0 0 10px 0;
        clear: both;
    }
    footer .column {
        float: left;
        display: inline_block;
        width: 300px;
        margin: 20px 10px;
    }
    /* Widgets */
    article.widget_twitter li {
        margin: 20px 0;
    }
    article.widget_social li {
        float: left;
        display: inline_block;
    }
    footer input[type=text], footer textarea {
        width: 200px;
        height: 20px;
        border: none;
        color: #FFFFFF;
        -moz-border-radius: 5px; -webkit-border-radius: 5px; border-radius: 5px;
        padding: 5px 10px;
        margin: 10px 0;
        text-shadow: 0 1px 0 #0F2326;
    }
    footer input[type=text]:focus, footer textarea:focus {
        outline: none;
    }
    footer textarea {
        height: 53px;
    }
    footer input[type=text], footer textarea {
        /* Fallback Color */
```

```
        background: #46737B;
        background-image: -moz-linear-gradient(#2C5F67, #5C858C);
        background-image: -webkit-gradient(linear, 0% 0%, 0% 100%,
from(#2C5F67), to(#5C858C));
}
article.widget_newsletter input[type=text] {
        width: 130px;
        -moz-border-radius: 5px 0 0 5px; -webkit-border-radius: 5px 0 0 5px;
border-radius: 5px 0 0 5px;
        float: left;
}
article.widget_newsletter input[type=submit] {
        margin: 10px 0 0 0;
        float: left;
        display: block;
        border: 1px solid #4E7A81;
        width: 70px;
        height: 30px;
        -moz-border-radius: 0 5px 5px 0; -webkit-border-radius: 0 5px 5px 0;
border-radius: 0 5px 5px 0;
        text-align: center;
        line-height: 28px;
        border: 1px solid #ADC0C4;
        color: #5E8C92;
        font-weight: bold;
        font-size: 12px;
        cursor: pointer;
}
article.widget_newsletter input[type=submit] {
        /* Fallback Color */
        background: #DFF0F3;
        background-image: -moz-linear-gradient(#FFFFFF, #EFEFEF);
        background-image: -webkit-gradient(linear, 0% 0%, 0% 100%,
from(#FFFFFF), to(#EFEFEF));
        background-image: -webkit-linear-gradient(#FFFFFF, #EFEFEF);
```

```
}
article.widget_newsletter input[type=submit]:hover {
	background: #FFFFFF;
}
article.widget_newsletter input[type=submit]:active {
	background: #EFEFEF;
}
```

For the social links we use one master style for the dimensions and position etc. plus an individual style for each icon setting a different background image.

```
/* Social Links */
article.widget_social ul li a {
	display: block;
	width: 32px;
	height: 32px;
	text-indent: -5000px;
	margin: 0 10px 0 0 ;
}
article.widget_social ul li.facebook a {
	background: url('images/social_facebook.png') no-repeat;
}
article.widget_social ul li.twitter a {
	background: url('images/social_twitter.png') no-repeat;
}
article.widget_social ul li.tumblr a {
	background: url('images/social_tumblr.png') no-repeat;
}
article.widget_social ul li.behance a {
	background: url('images/social_behance.png') no-repeat;
}
article.widget_social ul li.linkedin a {
	background: url('images/social_linkedin.png') no-repeat;
}
footer .button_1 {
	color: #5E8C92;
```

```
}
```

And finally, the style for the very bottom section of the footer. The logo style is almost identical to the logo used in the header.

```
/* Copyright/Logo */
footer p.copyright_notice {
	clear: both;
	padding: 20px 0 0 0;
}
footer .logo {
	float: right;
	width: 221px;
	height: 65px;
	background: url('images/logo.png') no-repeat;
	margin: -40px 10px 0 0;
	text-indent: -5000px;
}
```

And there you have it! we have successfully designed and coded a full homepage template. As always, I hope you have enjoyed reading these tutorials and perhaps gained a different perspective or even learned some new techniques.

**Integrating the web Templates in the editor**

Having a presence on the Web is no longer optional for businesses, but choosing whether to set up your Web pages from scratch or rely on a template can make all the difference. Templates suitable may already exist in Dreamweaver, and can add one to your current site instead of having to hand-code the entire website in HTML yourself. Whether you are looking to staff up, announce a sale or reach local customers, Dreamweaver templates can get you there faster.

**Existing Website**

**Step 1**

Launch Dreamweaver and click on the name of your existing HTML Web file in the left column under “Recently Used.” If you don’t see the file there, click “Open,” browse to the HTML file and double-click on it.

**Step 2**

Click the “Modify” menu. Choose “Templates,” then click “Apply Template to Page” to open the Select Template window

**Step 3**

Browse to and double-click the template you want to import. Click once on the name of the template, then click the “Select” button.

**Step 4**

Click the “Use for all” button or cherry-pick the parts of the template you wish to apply to your current page. Click the “OK” button and Dreamweaver makes the changes to import the template and applies it to your current page.

**Step 5**

Continue to work within your page and make any other changes required

**Step 6**

Click the “File” menu and choose “Save As.” Rename the page to preserve the previous version and click the “Save” button.

**From Scratch**

**Step 1**

Download or save the template file in the proprietary Dreamweaver ".dwt" file format to the Template folder in your Dreamweaver root directory.

**Step 2**

Launch Dreamweaver and click the “File” menu. Choose “New,” then click the “Page from Template” option.

**Step 3**

Scroll the "Site" column and double-click your Dreamweaver page name.

**Step 4**

Scroll the "Template for Site" column to the template you just downloaded or saved.

Click once on the name.

**Step 5**

Click the “Create” button to import the template. When Dreamweaver opens, make any changes to the template, add text, set up hyperlinks and continue setting up your site as required

**Step 6**

Click the “File” menu and choose “Save As.” Rename the template to preserve the original version, and then click the “Save” button.

**Terminologies**

**Instruction for teachers**

1. Give the practical examples found at their home and surrounding.
2. Use the Charts as far as possible to explain the content.
3. Assign the homework at the end of every class from the chapter so far covered during that class.
4. Conduct group discussions after finishing one lesson.

**References**

http://www.innovateus.net/content/what-html-editor-software
http://www.computerhope.com/htmldown.htm
https://en.wikipedia.org/wiki/Adobe_Dreamweaver
http://pc.net/helpcenter/answers/notepad_vs_wordpad
https://openclassrooms.com/courses/build-your-website-with-html5-and-css3/your-first-web-page-in-html
http://www.freeformatter.com/html-formatter.html#ad-output
https://www.w3.org/Style/Examples/011/firstcss.en.html
http://www.peachpit.com/articles/article.aspx?p=600619
http://www.getawebsite.friezedesign.co.uk/plan.htm
https://www.w3.org/standards/webdesign/graphics.html
https://www.washington.edu/accessit/webdesign/student/unit4/module1/Guidelines_for_web_graphics.htm
http://webstyleguide.com/wsg3/11-graphics/2-graphics-as-content.html

**PRACTICAL**

TITLE

GUI Based HTML Editor

**OBJECTIVE**

To learn about the GUI Based HTML Editor and its real application.

**LAB ACTIVITIES**

- Online Searching the editor and install them on local machine
- Creating the First page in the editor
- Create your own website in the editor
- Editing the source code in the editor
- Connecting external CSS file to HTML document

# UNIT-7

# Project Work

## Learning Outcomes:

After Completion of this lesion, Student will be able to:

- Start new small web based project.
- Design static WebPages using different layout.
- Manage color combination and CSS design of static pages like school, small business.

## 7.1 Introduction

This chapter will help to set the stage, plan, and preparation to develop simple web page built in HTML and CSS.

Through a series of surveys, discussion, and research, Discovery leads to understanding three critical things for information gathering related to the web. Discovery is all about gathering information and asking a lot of questions. The answers will serve as a reference for nearly every step that follows.

Clarification and Planning each consist of taking the information gathered and putting it together into documentation — the former into a Communication Brief and the latter into a Project Plan.

- What are the client's wishes and goals? What is the proposed plan to carry these out?
- Who are the team members, and what are their responsibilities?
- What are the client's responsibilities?
- What are the specific project deliverables (both client and team), when are they due, and what are the budgetary and scheduling impacts of missing deadlines?
- How will the site be tested against audience needs?
- What are the immediate measurable goals of the site redesign? What are the long-range goals for the site?
- What, if any, are the technical requirements for complex functionality?

At the end of this defining phase, the preparatory materials are distributed at the meeting, attended by all team and key client members. The goal is to communicate clearly, to keep the members of the team aligned with the same goals and terminology during the life of the project, and to make sure no one is ever left guessing as to what comes next or when what comes next is due.

Discovery is a thinking process. Its purpose is to allow team members to put themselves in the minds of the site's users and to understand as much as possible about the target audience(s). To start, you need information. There are a lot of questions to ask; the surveys will get you going. Regardless, Discovery starts with the Client Survey.

- Gathering Information
- Understanding Your Audience
- Analyzing Your Industry

Client input is the foundation on which successful websites are built. This survey will help you articulate and identify the overall goals of site redesign, including specific questions regarding message, audience, content, look and feel, and functionality. Each key decision-maker should fill out his or her own survey, answer each of the questions in a thorough but brief and clear manner, and add any additional notes or comments at the end of the survey.

## General Information

1) What is the name of your company and your current (or intended) URL?
2) Who are the primary contacts from your organization, and who has final approval on the project? list names, titles, email addresses, and phone numbers.
3) What is your intended launch date for the new site? Are there any outside considerations that might affect the schedule (for example, notice, events, annual report)?
4) Have you conducted usability tests or gathered visitor feedback for your current site? If so, how long ago? Please include any reports or findings.
5) How important is it to maintain and design look and feel, logo, and branding?

## Content

1) Will this site use existing content from the current site? If so, what is the source, who is responsible for approval, and has the content been audited? If not, will you be creating content in-house or using an outside provider?
2) What is the basic structure of the content, and how is it organized? Is it a complete overhaul of the current site or an expansion?
3) Describe visual elements or content that should be utilized from your current site or marketing materials (logo, color scheme, navigation, naming conventions, etc.)
4) How will the content of this site (along with functionality and navigation) expand or differ from your current site? Do you have an existing sitemap for the outgoing site structure? Do you already have a sitemap or outline for the proposed redesign?

## Technology

1) What is your target platform and browser? Whom can we talk to in your organization to help respond to technical issues?
2) Are there specific technologies (CSS, JavaScript, HTML, etc.) that you would like to use in the site? If so, how will they enhance the user experience? Please describe in detail.
3) Will you have database functionality (static content generation)?
4) Do you intend to keep the site updated? If so, how often? Who is responsible for updating and providing content?

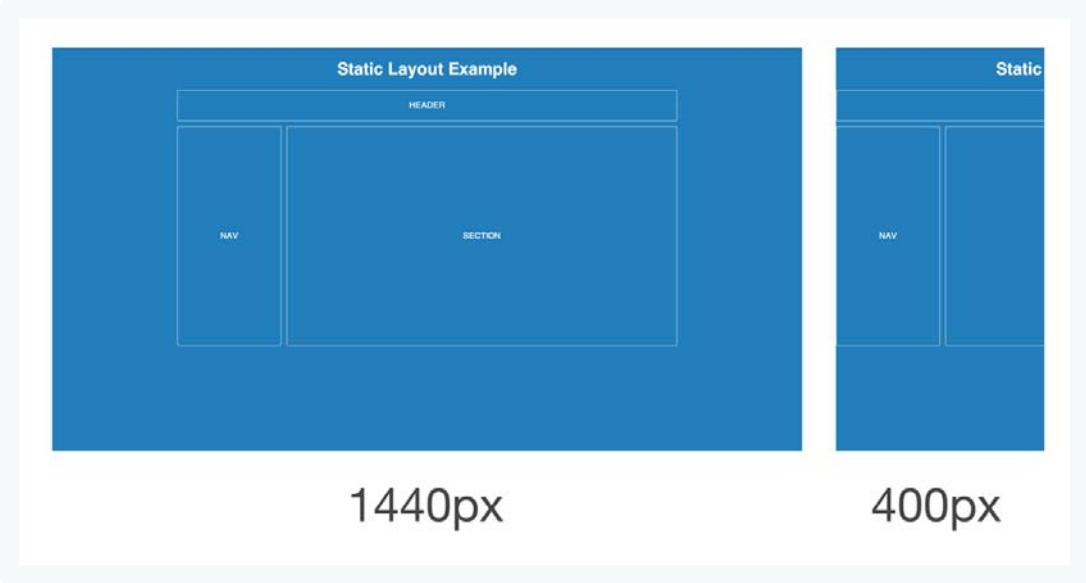
## 7.2 LayOut

When discussing a page layout, web designers will often use terms like fixed, static, liquid, adaptive, responsive, and a few others. If you're a web designer, a web developer, or even a project stakeholder or client, it's important to understand what these terms mean and when each type of layout should be utilized.In short, each page layout name describes how the layout behaves when the page is viewed at different browser widths.

The horizontal width of the browser could change because the website is being viewed on different devices (mobile phones, tablets, desktops, and so on), the website visitor

could simply resize the browser window on a desktop device, or the visitor might change their phone from portrait to landscape mode, and so on. These concepts can be difficult to grasp at first, but only because they're so closely related to one another. Understanding the differences between each one is the key to understanding them individually.

**Static Page Layout**




A static page layout (sometimes called a "fixed" layout or "fixed width" layout) uses a preset page size and does not change based on the browser width. In other words, the page layout might have a permanent width of 960 pixels no matter what. This is how web pages were traditionally built for many years until modern influences like media queries and responsive design were introduced around the start of the 2010s.

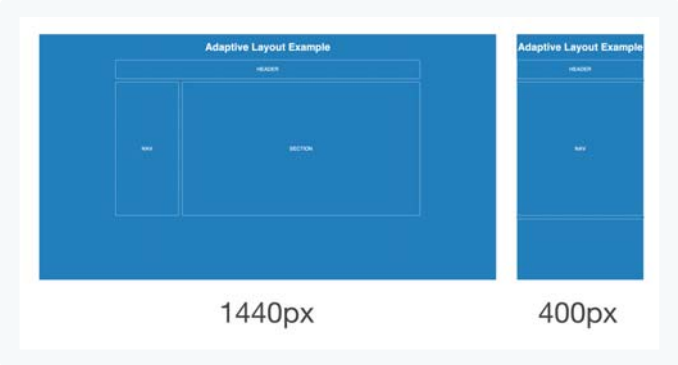
Different devices will treat a static page layout in various ways, so the rendered page could be slightly unpredictable. For example, on a desktop browser, if the window is too small horizontally, then the page will be cut off and horizontal scroll bars will be displayed. On a mobile device like an iPhone, the page will be scaled automatically, allowing the user to zoom in on points of interest ( provided that no metatags override this default behavior).When new websites are built, most of them don't opt for a static layout, because it means that the mobile experience will require a separate website.

## Liquid Page Layout


A liquid page layout (sometimes called “fluid” or “fluid width”) uses relative units instead of fixed units. Typically a liquid layout will use percentages instead of pixels, but any relative unit of measurement will work, such as ems.

A liquid layout often will fill the width of the page, no matter what the width of the browser might be. Liquid layouts don’t require as much thought as a responsive design, but there are some major drawbacks at very large or very small browser widths. If the browser is very wide, some content might be stretched too far. On large screens, a single paragraph might run across the page on a single line. Conversely, a multi-column layout on a small screen could be too crowded for the content.

## Adaptive Page Layout

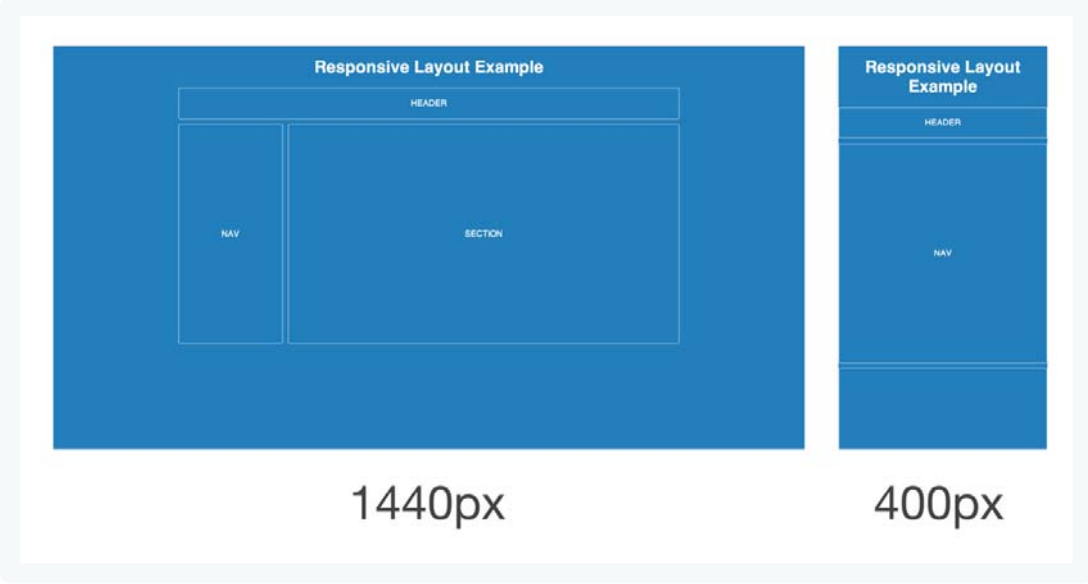


An adaptive page layout uses CSS media queries to detect the width of the browser and alter the layout accordingly. Adaptive layouts will use fixed unit like pixels, just like a static layout, but the difference is that there are essentially multiple fixed widths defined by specific media queries.

A media query is an expression of logic, and when used in combination with other media queries, they form a basic algorithm. So for example, an adaptive page layout might say something like, “If the browser 500 pixels wide, set the main content container to be 400 pixels wide. If the browser is 1000 pixels wide, then set the main content container to be 960 pixels wide.” Beyond the main content container, other parts of the page might change in width, swap places, or be removed. For example, a two column layout might become a single column layout if the browser is too narrow.

Adaptive layouts are a good stop-gap solution if a legacy static layout needs to be converted to support mobile devices. They also typically require less development time than a responsive layout. The major drawback is that device widths in-between the explicit breakpoints are often less than ideal, with either too much space or not enough space.

**Responsive Page Layout**



A responsive page layout uses both relative units and media queries, ostensibly combining the ideas of a liquid layout and an adaptive layout. As the browser increases or decreases in width, a responsive layout will flex just like a liquid layout. However, if the browser goes beyond certain widths defined by media query breakpoints, then the layout will change more dramatically to accommodate a wide or narrow width.

Typically responsive designs are built using a mobile-first approach. That means the mobile layout is designed first, and then as the browser becomes wider on tablets and desktops, the designer will find ways to expand the mobile layout. This tends to create better experiences overall, because it's easier to expand a design than to try and simplify a large layout for mobile screens.

## 7.3 Design

All web pages that you see on the Internet use HTML to format its pages for display in a web browser. This is so whether the website is a shopping site, a search engine, a blog or a tutorial site like thesitewizard.com. This tutorial is intended to be a systematic guide to teach you HTML from the ground up. It assumes no knowledge of HTML. By the end of this tutorial series, you'll be confidently creating web pages by directly coding in HTML.

**Preliminary Matters**

Hand coding a web page in HTML is not the easiest way to create a website. In fact, it's probably the slowest and least efficient way, and, depending on your inclinations, possibly the most tedious. If all you want is to create a website, you may want to consider using a visual web editor to do the job instead. Visual web editors, also called WYSIWYG web editors ("What You See Is What You Get" editors), are computer programs that let you design websites without needing any technical know-how.

Since this is an HTML tutorial, it omits some of the non-HTML-related steps that are required for creating a website, like getting your own domain name, getting a web host, etc.

HTML in and of itself will not let you do things that web designers typically want to do on their website, like create 2 or more columns on your websites, use different fonts, change colours (or "colors" if you use, or do certain other visual design work to

make your web page look nice. Things that govern the appearance of your website are handled by a separate technology called CSS.

That is to say, after completing the HTML and CSS portion, you can complete website. As in the given previous chapters, HTML tutorial will seamlessly lead into the CSS tutorial, since both sections of the tutorials were written to work with each other.

**Get a Proper Text Editor**

Before starting hard coding, The most important thing you need is to get a proper text editor.

**Do NOT use Notepad or simple word processor**

Notepad is NOT a proper text editor. If you are going to be working on your website by coding directly in HTML, you really need a fully functional editor. Notepad cannot handle certain characters and character combinations that you may encounter along the coding. As such, in certain circumstances, it may corrupt your file without your realizing it. Although, It'll work fine most of the time, but at some point along the road, if your file happens to have one or more of those characters it doesn't handle. And you won't even know it happened, since there is no warning or error message accompanying the file corruption. In addition, Notepad cannot handle large files, in which case it may transparently invoke a different program, Wordpad, to do the job. And Wordpad will definitely corrupt your web page.

Bear in mind that even the Microsoft developers do not regard (nor do they recommend) Notepad as a viable editor for normal text editing work. The editor was created for the casual computer user to view things like Windows log files and things like that. That's why it does things like transparently invoking a different program for huge files. Since it's primarily for viewing files, it doesn't matter whether Notepad or Wordpad does the work. But for the webmaster, it matters a great deal.

Even if you are striving for an ascetic life of hardship and severity, Notepad isn't the way. It may be acceptable for someone dabbling with the occasional text file (and even then it's debatable), but it's not the correct tool for someone who wants to work directly with HTML on a regular basis.

Do not use word processors to hand code HTML files. Word processors insert hidden (and invisible) code into your files which are not part of HTML.

By word processors, I mean any program that allows you to underline text, make text bold, put text in italics, create tables, etc. Wordpad is a rudimentary word processor. Do not use it for this purpose.

Some of you are probably thinking at this point that I've ruled out just about everything on your computer system. Don't worry. Text editors are a dime a dozen on the Internet. In fact, you don't even need a dime. There are zillions of free text editors around. (And zillions of commercial ones too, for that matter. Programmers seem to love creating text editors.)

If you already have Dreamweaver, you don't have to get a separate text editor. Just switch to Dreamweaver's code view. That is, click "View" from the menu bar, and "Code" from the drop-down menu that appears. This switches Dreamweaver from its visual design interface to its text editor interface.

And you're not getting some half-baked text editor either. Dreamweaver's built-in text editor is actually very good, and one that is highly optimised for people writing HTML, so you might as well not waste time looking for another one. Using Dreamweaver's code editor also has numerous other advantages. For example, it has tools to check your code for correctness, and you can also easily switch to the editor's Design view to get an idea of what your web page looks like in a browser. Other advantages include being able to use its site synchronisation facilities to publish your web pages. And so on.

**Your First Web Page**

Depending on which editor you've installed, the text above may appear in a variety of colours. The colours are what programmers call "syntax highlighting".

To save the file, click the "File" menu, followed by the "Save" item in the sub menu that appears. When prompted for a filename, select the appropriate folder for the file to be saved (for example, your desktop, so that you won't forget where you put it) and type "sample.html" (without the quotes).

Note: If you use Notepad, this won't work, since Notepad will invisibly change your filename to "sample.html.txt" behind your back.

Now open the file in your web browser. One way to do it is to doubleclick the file on your desktop. The file should appear in your default web browser. Keep both the browser and the text editor open.

**The Logic of HTML**

Angle-bracketed words (or letters) are commonly called "HTML tags" in HTML language and used for coding. In HTML, the majority of tags come in pairs, with the unadorned version, like "<body>", being used to indicate the start of something, and the version with a preceding slash ("/"), like "</body>", being used to flag the end of something.

Each tag has a specific function. For example, the " " and " " tag pair is used to indicate the start and end of a paragraph. (Think of the "p" in " " as referring to "paragraph".)

**The Structure of an HTML Document**

Before discussing the structure of a web page, consider a normal business letter in the brick and mortar world. Such letters actually have a particular form: if the paper you're using does not have a letterhead, you will normally have to type (or write) your own address at the top of the page, followed by the address of the person you're writing to below. Only then do you begin your letter proper with something like "Dear...". There is typically a subject line following that, and below that comes the body of your letter where you finally get to say the things you wanted to say.

Web pages also have a particular format.

1) The first line of the page identifies the type and version of HTML that your web page is using.
2) The rest of the page is enclosed between the "<html>" / "</html>" tag pair. Code enclosed between these two tags form the web page proper.
3) The first part of the web page is what is known as the HEAD section. The section begins with "<head>" and ends with "</head>". It contains information meant for web browsers and search engines. Code placed in this section is not displayed in a web browser. It's like the addresses you place at the beginning of a business letter — they serve a purpose, but they're not actually part of the main content.
4) The part of the web page that contains the information displayed in a web browser is called the BODY section, and it is enclosed between the "<body>" and "</body>" tag pair.

Each of these things is discussed in detail below.

**DOCTYPE or DTD**

In the example code you used above, the first line reads as follows:

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

The version line is commonly called a DOCTYPE by webmasters (since the line begins with the word DOCTYPE). Officially, it is known as a Document Type Definition (or "DTD" for those who love abbreviations). As its name indicates, it does more than tell web browsers the version of HTML you're using. It also indicates whether your web page is one of 3 types: whether it uses frames (a type of HTML technology), whether it is a normal web page ("transitional"), or whether it is a normal web page that uses a limited subset of HTML ("strict").

**The HEAD Section: Introducing The TITLE Tag**

One of the most important things that belong in the HEAD section is the TITLE tag for web page. This tag provides web browsers and search engines the title of your page. The title you give in this tag doesn't actually appear on your web page itself, but it is placed in the title bar of the web browser window when it displays page.

Switch to your text editor and add the following line into your HEAD section.

```
<title>My First Web Page</title>
```

Move your cursor to the end of the line containing the <head> tag. Hit the ENTER key to create a new line underneath. Then type in the words I gave in the block above. Actually, you don't really have to use the words "My First Web Page". Use whatever words you want as the title. But make sure those words are enclosed between the opening <title> tag and the closing </title> tag. And ensure that this entire sequence occurs before the closing </head> tag.

Save your page and reload (refresh) it in your web browser.

Now look at the title bar of the browser window. You should be able to see the title you set. Notice that the words you typed do not appear in the main body of your web page itself. The <title> only sets the title of your web page for the benefit of web

browsers and search engines. It does not change your web page's appearance. Anything that you want displayed on your web page itself has to be inserted into the BODY section.

Important: The TITLE tag is required on every web page.

**The BODY Section and The Paragraph Tag**

How does one add a new paragraph to a web page? I'm sure at this point, most of you have already (long) guessed the answer. But let's try it out so that you can see it in action. (I also want to use it to demonstrate something else about HTML. So please do it even though it seems really obvious.)

Move your cursor to the end of the  line. Hit the ENTER key. Then insert the following into the new line that you created.

This is a new paragraph because I used a new paragraph tag to start it off.

Save the page and reload it in your browser. The newest addition to your web page should now appear as a new paragraph in your browser.

Switch back to your text editor again. Notice that unlike the previous paragraph, I didn't put my  tag on its own line. Instead, I just typed the sentence belonging to the paragraph immediately after the tag, and yet everything worked out the same as when I put the tag on a separate line. Whether or not there is any whitespace between the paragraph tag and my actual text doesn't matter at all.

I recommend that you play around with this a while to get familiar with how the web browsers interpret white space and the paragraph tags. That is, add new paragraphs, or new sentences to your existing paragraph tags, put all sentences and tags on a single line, and reload the page in your browser to see its effects. Visitors come to web sites for their own interest, not because they like you and want to please you by visiting your web site. It’s about them, their needs and interests, not yours. They don’t really care about your website, they care about them. Give them what they want, provide VALUE for them by offering relevant contents, this is why they are coming. Did I mention that they could easily decide to ignore you? I am always amazed on visting web sites that lack the most elementary information I came to the site for in the first

place. So you own a restaurant. I love all the photos you want to show me, it's great to know that the chef is your wife and that you have three happy kids. But for heaven's sake, WHAT IS YOUR PHONE NUMBER SO THAT I CALL CALL AND MAKE A RESERVATION. Not in the "about us". Not in the "contacts", where you just have your e-mail. So I will just ignore you. This is a very simple example but make the point. Offer the expected contents, generously and well accessible and organized.

Designing a web site needs careful thinking and planning. The most important thing is to KNOW YOUR AUDIENCE.

**Users are Scanners**

A typical visitor will NOT read the entire content of your Web page!

No matter how much useful information you put into a Web page, a visitor will only spend a few seconds scanning it before he/she decide whether to leave or to stay.

Be sure to make your point in the very first sentence of the page! After that, try to keep the user occupied with short paragraphs, and new headers down the page.

**Less is More**

Keep the paragraphs as short as possible.
Keep the pages as short as possible.
Keep the chapters as short as possible.
Use a lot of space! Pages overloaded with text will kill your audience.
If you have a lot to say, break your information into smaller chunks and place it on different pages.

**Navigation**

Create a consistent navigation structure that is used by all the pages in your Web site.

Don't use hyperlinks inside each paragraph, to send visitors to every page of your Web. This will destroy the feeling of a consistent navigation structure. If you must use hyperlinks, add them to the bottom of a paragraph, or to the menu.

**Download Speed**

Sometimes developers are not aware of the fact that some pages take a long time to download.

Most visitors will leave a Web page that takes more than 7 seconds to download.

Test your web pages over a low-speed modem connection. If your pages take a long time to download, consider removing graphic or multimedia content.

**Let your Audience Speak**

Feedback is a very good thing.

Your visitors are your "customers". Often they will give you some valuable hints about what you could have done better.

Provide a simple way to reach you, and you will get a lot of input from people with different skills and knowledge.

**Visitor's Monitor**

Not everyone on the internet has the same monitor as you. If you design a Web site to be displayed on a monitor with a high resolution, visitors with lower resolution monitors (like 800x600) might have problems reading your pages.

Make sure you test your Web site on different monitors.

**What Browsers Do They Use?**

Don't forget to test your Web site on different browsers.

The most popular browsers today are Internet Explorer, Firefox and Google Chrome.

One wise thing to do when designing Web pages is to use correct HTML. Correct coding will help the browsers to display your pages correctly.

**What Plug-Ins Do They Have?**

Sound, video clips, or other multimedia content might require the use of separate programs (plug-ins).

Be sure that your visitors have access to the software needed to view them.

**What About Disabilities?**

Some people have viewing or hearing disabilities. They might try to read your pages with Braille or speech-based browsers. Always add text alternatives for images and graphic elements.

## 7.4 Color Combination

Maybe you have some basic skills in html/css and know how to open an hosting account, register a domain name and put up a simple web page. Does this qualifies you as a webmaster? Well, yes indeed. If you want to setup a personal website, or a website for friends and family, fine. Are you a professional webmaster then? Setting up a beautiful web site with a consistent navigation framework requires experience. Designing website graphics is a specialized job that comes with his own set of skills, distinct from the web coding skills. As all artistic skills, it conjugates personal inclinations and sensibility with technical knowledge of some specialized software tools required to generate a professional graphics.

If your goal is to get to a professionally designed, gorgeous web site (and indeed this is where you should aim for a great and competitive online presence, even for a simple showcase web site), you should definitely use a design produced by a professional web designer. won't this be horribly expensive? The answer to this is articulated. First of all, how do you define horribly expensive. You have to relate the costs to the expected benefits of your online presence. If you are a freelance, mostly working "offline", and all you need is a "business card" website, where you contact information is posted, and you are generally on a tight budget, then the threshold for "horribly" will be low. If on the other hand you run a purely online business, say an online shop, a great graphics could have a dramatic impact on sales (for sure, an ugly graphics and poor navigation WILL have a strong impact on sales).

The other important thing to consider is that you have a choice between hiring a professional to make a new custom design for you significantly expensive or using a non-exclusive professionally designed commercial or free template, that may be customized for your needs. This second option can be one order of magnitude cheaper than the custom design option, and the result will possibly be equally great. The difference is that in the second case, your layout/design will not be unique on the web, there might be similar sites around. This similarity will be greatly limited if your webmaster is able to customize the color scheme and other relevant aspects of the web site for you. Which way to go will depend also on what we are going to discuss next: set an appropriate budget.

## 7.5 Usages of Graphical Images

The Web is about more than text and information, it is also a medium for expressing artistic creativity, data visualization, and optimizing the presentation of information for different audiences with different needs and expectations. The use of graphics on Web sites enhances the experience for users, and W3C has several different and complementary technologies that work together with HTML and scripting to provide the creators of Web pages and Web Applications with the tools they need to deliver the best possible representation of their content.

**What are Graphics?**

Web graphics are visual representations used on a Web site to enhance or enable the representation of an idea or feeling, in order to reach the Web site user. Graphics may entertain, educate, or emotionally impact the user, and are crucial to strength of branding, clarity of illustration, and ease of use for interfaces.

Examples of graphics include maps, photographs, designs and patterns, family trees, diagrams, architectural or engineering blueprints, bar charts and pie charts, typography, schematics, line art, flowcharts, and many other image forms.

Graphic designers have many tools and technologies at their disposal for everything from print to Web development, and W3C provides many of the underlying formats that can be used for the creation of content on the open Web platform.

**What are Graphics Used For?**

Graphics are used for everything from enhancing the appearance of Web pages to serving as the presentation and user interaction layer for full-fledged Web Applications.

Different use cases for graphics demand different solutions, thus there are several different technologies available. Photographs are best represented with PNG, while interactive line art, data visualization, and even user interfaces need the power of SVG and the Canvas API. CSS exists to enhance other formats like HTML or SVG. WebCGM meets the needs for technical illustration and documentation in many industries.

**What is PNG?**

Portable Network Graphics (PNG) is a static file format for the lossless, portable, well-compressed storage and exchange of raster images (bitmaps). It features rich color control, with indexed-color, grayscale, and truecolor support and alpha-channel transparency. PNG is designed for the Web, with streaming and progressive rendering capabilities. It is supported ubiquitously in Web browsers, graphical authoring tools, image toolkits, and other parts of the creative tool chain. PNG files have the file extension ".PNG" or ".png" and should be deployed using the Media Type "image/png". PNG images may be used with HTML, CSS, SVG, the Canvas API, and WebCGM.

**What is SVG?**

Scalable Vector Graphics (SVG) is like HTML for graphics. It is a markup language for describing all aspects of an image or Web application, from the geometry of shapes, to the styling of text and shapes, to animation, to multimedia presentations including video and audio. It is fully interactive, and includes a scrip table as well as declarative animation (via the SMIL specification). It supports a wide range of visual features such as gradients, opacity, filters, clipping, and masking.

The use of SVG allows fully scalable, smooth, reusable graphics, from simple graphics to enhance HTML pages, to fully interactive chart and data visualization, to games, to standalone high-quality static images. SVG is natively supported by most modern browsers (with plugins to allow its use on all browsers), and is widely available on mobile devices and set-top boxes. All major vector graphics drawing tools import and export SVG, and they can also be generated through client-side or server-side scripting languages.

**What is CSS?**

Cascading Style Sheets (CSS) is the language for describing the presentation of Web pages, including colors, layout, and font information. It may be used to enhance the graphical aspects of HTML and SVG. You can read more on the page for HTML & CSS.

**Purpose of graphics in web design**

1) Graphics add visual/aesthetic appeal to the information on the web page, helping to maintain viewer interest and attention
2) Help to create visual structure for information and links on the page
3) Communicate or explain ideas visually

**Guidelines for effective use of web graphics**

1. **Graphics should fit in with the purpose, organization, and style of the page.** They should enhance the design, structure, or informative content of the web page without distracting attention. As much as you may want to add an image to a page because it shows off some fancy new effect you learned in the graphics program, think first about whether it fits in or not. You may want to keep these types of graphics in your personal portfolio under the heading, “Cool Effects I Know in My Graphics Program.”
2. **Avoid using graphics with large file sizes that add to the load time of the page.**
   Also, consider the cumulative file size of all images on the page. Excessive “page weight” caused by poor image use can result in slow load times for pages. According to the Yale University Web Style Guide, 2nd Edition, "At today's average modem speeds most pages designed for users dialing in from home should contain no more than 50 to 75 kilobytes of graphics."
3. **Graphics should help to guide the viewers’ focus to the important content on the page.**

   Using visually strong graphic elements on a page can be useful in directing viewers' attention and providing structure for the page. Be careful, though. Strong graphic elements can also pull attention away from central content or compete with one another on the page. This results in the page appearing overly busy or cluttered and makes it difficult to read. Remember, if you attempt to emphasize everything then nothing ends up standing out.
4. **Avoid repetitive use of overly bright or potentially “obnoxious” images.**

   These types of "eye catching" images may be attractive at first, but after the novelty wears off, they may cause viewers to lose patience with the site. While

an animation of a dancing monkey may seem interesting and funny at first, over time it may become annoying and drive viewers away.

5. **Avoid the use of graphics to convey textual content information.**

   While it may be tempting from a design standpoint to use images for textual information because of the greater number of design options such as font choices and text effects, graphic based text presents a variety of problems: Images of text can't be resized like true text, so users with poor vision are unable to resize it to meet their visual needs; images require much longer to download than text; users can't search for images of text using their browser's find feature; and search engines are better able to index websites that contain actual text, well-structured with HTML.

6. **Provide textual equivalent alternatives for graphic content.**

   Remember that not all users are able to see the images on your web page. Whenever images are used, it is important to provide equivalent content or descriptions of the image in a textual format. The most common way this is done is to provide descriptive text using the ALT attribute. This is especially important where images are used as navigation buttons or links. For a more in depth look at using alternate text with graphics to improve accessibility, read the section on Graphics Markup within the Web Style Guide.

7. **When using text in graphics, make sure there is sufficient contrast between the text and the background color.**

   Design graphic elements so that users can easily distinguish the text from its background. Also, be sure to avoid color combinations (reds and greens for example) that color blind users will not be able to distinguish apart from one another.

**The Role of Web Graphics**

First and foremost, consistent interface and identity graphics across a collection of web pages define the boundaries of a web "site." Although web designers could build a site without graphics, most users would not readily recognize a collection of bare pages as a cohesive "site," and such a site would seem unpleasantly odd, well outside of design norms and user expectations. Site-defining identity graphics do not need to

be elaborate, but they do need to be consistent across the range of pages in a site for the user to establish a sense that your pages are a discrete region—related perhaps to a larger whole if you work in a major enterprise—but in some ways distinct as a "place".

**Graphics as content**

Graphics serve a number of purposes as elements of content—along with and complementary to text content:

- Illustrations: Graphics can show you things, bringing pieces of the world into your document
- Diagrams: Quantitative graphics and process diagrams can explain concepts visually
- Quantitative data: Numeric charts can help explain financial, scientific, or other data
- Analysis and causality: Graphics can help take apart a topic or show what caused it
- Integration: Graphics can combine words, numbers, and images in a comprehensive explanation

**Illustrations**

Photographs and illustrations bring the world into your document in a concise and unforgettable way that saves that proverbial "thousand words" of explanation. Why describe something when you can simply—and more memorably—show it? The web is a color medium with an enormous range of display colors, and color graphics on web pages are almost the equal of fine printing on paper, especially when graphics are reproduced at roughly the same size as they might appear in print.

**Diagrams and quantitative data**

The resolution limitations of the computer screen (72–90 pixels per inch) versus print (300+ dots per inch) begin to show when you create complex charts and diagrams for the computer page. Most chart graphics work as well on the screen as on paper, but the label typography in charts and diagrams must be optimized for display on web screens. The visual detail of complex diagrams converted from print uses is often

surprisingly good, but always check the typography within the figure, as labels may become illegible in the conversion process.

**Integrated visual presentations**

Multifaceted information graphics often integrate quantitative charts, three-dimensional illustrations, and extensive captioning in free-form layouts that become visual narratives capable of explaining complex concepts and natural phenomena.

**Graphic communication on the web**

A chart or diagram is an implicit promise to the user that you'll make a complex world easier to understand. Our advice on graphic communications is the same as on written communications:

- Trust the reader's intelligence. Don't dumb down your material on the supposition that web users are somehow fundamentally different kinds of people from print readers and have no interest in complexity. Regular readers of web sites may once have been distinguishable from other publishing audiences, but now everyone reads the web.
- Respect the medium. The readers are the same as in print media, but the web has a different profile of strengths and weaknesses. Take advantage of the web's enormous capabilities to communicate complex color visuals without the expense of printing and physical distribution.
- Tell the truth as you understand it. Distorting quantitative data isn't just a failure to communicate; it's a betrayal of the reader's trust.
- Don't cherry-pick your data. If you are making a case with visual evidence, don't process and edit your visuals so heavily that the audience has no choice but to accept your point of view. Trust your audience enough to give them the data: let them look at the same higher-resolution images or ambiguous results that you saw and decide the issue for themselves.
- Be bold and substantial. A serious interest in visual communication doesn't require that you use only small, mousy graphics in pale colors. Visual evidence can't become persuasive if no one ever notices it. Just don't ever try to wow an audience with bright graphics to make up for thin content.

## 7.5 Home Page

Many (far from all though!) of your visitors might enter your website from the home page. Together with the sections and subsections design, planning the various elements that will populate the home page of your site is one of the first things to consider, before starting development. Here is a tentative list of things you might consider including. None is really essential, and you could decide to think entirely out of the box here. Some elements might be missing. Here you go, anyway:

- Logo
- Site name (can be in the logo)
- Site slogan (can be in the logo), a short sentence, a few words about your site. Mind that text in the images cannot be read by search engines, so consider carefully what you have to do to optimise your search engine popularity here. If you use images, be sure to use (or to have your webmaster use) ALT tags generously.
- A paragraph of text describing what your website/business/activity/project is about. Again, great for search engines and for your visitors aswell, don't keep them guessing too much. Did I already mention that they could decide to ignore you?
- Navigation menu(s), see point 5 above
- A news box and/or important event box? Maybe to be activated or deactivated depending on the availability of events or more in general, things to point out.
- A website footer. The contents are up to you. They could typically be links to "secondary/service" pages such as "contact us" or "site map", copyright statements, credits etc..

You should consider that the contents of the header (logo, slogan, top bar navigation menu) and footer, maybe with some variations, might well be displayed over your entire web site, in each single page. They are important elements to be designed with care.

**Instruction for teachers**

1. Give the practical examples found at their home and surrounding.
2. Use the Charts as far as possible to explain the content.
3. Assign the homework at the end of every class from the chapter so far covered during that class.
4. Conduct group discussions after finishing one lesson.
5. Give shall titile to make a project work

---

**References**

http://www.peachpit.com/articles/article.aspx?p=600619
http://www.getawebsite.friezedesign.co.uk/plan.htm
https://www.w3.org/standards/webdesign/graphics.html
https://www.washington.edu/accessit/webdesign/student/unit4/module1/Guidelines_for_web_graphics.htm
http://webstyleguide.com/wsg3/11-graphics/2-graphics-as-content.html

**PRACTICAL**

TITLE

Design and develop simple static web-pages.

**OBJECTIVE**

To learn about the design and develop simple static web-pages.

**LAB TASKS**

- Gather requirement and analyze the problem by categorizing information.
- Develop the simple HTML and CSS template and web-page of school and own businesses.