WEB TECHNOLOGIES

CSC353

BSc. CSIT – Sixth Semester

Imagine that everything you are typing is being read by the person you are applying to for your first job. Imagine that it's all going to be seen by your parents and your grandparents and your grandchildren as well.

– Tim Berners-Lee

Note: This handout is for simple reference only. Do not completely depend on it.

CHAPTER 01 (9 Hours)

INTRODUCTION

Note: This handout is for simple reference only. Do not completely depend on it.

What is Computer Networking?

In the world of computers, **networking** is the practice of linking two or more computing devices together for the purpose of sharing data. Networks are built with a mix of computer hardware and computer software.

Internet

The Internet is a global network connecting millions of computers worldwide. The Internet is a massive public spider web of computer connections. Internet connects millions of computers together globally, forming a network in which any computer can communicate with any other computer as long as they are both connected to the Internet. Information that travels over the Internet does so via a variety of languages known as protocols.

What is The Web (World Wide Web)?

The World Wide Web, or simply Web, is a way of accessing information over the medium of the Internet. It is an information-sharing model that is built on top of the Internet. The Web uses the HTTP protocol, only one of the languages spoken over the Internet, to transmit data. The Web utilizes browsers, such as Internet Explorer or Firefox, to access Web documents called Web pages that are linked to each other via hyperlinks. Web documents also contain graphics, sounds, text, video etc.

Servers are powerful computers or processes dedicated to managing disk drives (file servers), printers (print servers), or network traffic (network servers), database (database servers) or web documents (Web Servers).

Clients are PCs on which users run applications. Clients rely on servers for resources.

A **web browser** or **Internet 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) and may be a web page, image, video, or other piece of content Hyperlinks present in resources enable users to easily navigate their browsers to related resources.

***In order of release*:**

- Netscape Navigator and Netscape Communicator, October 13, 1994
- Internet Explorer 1, August 16, 1995
- Opera, 1996
- Mozilla Navigator, June 5, 2002
- Safari, January 7, 2003
- Mozilla Firefox, November 9, 2004

- Google Chrome, September 2, 2008

Web Servers

Web servers are computers that deliver (*serves up*) Web pages. Every Web server has an IP address and possibly a domain name. Web servers are computers on the Internet that host websites, serving pages to viewers upon request. This service is referred to as web hosting. Every web server has a unique address so that other computers connected to the Internet know where to find it. The Internet Protocol (IP) address looks something like this: 69.93.141.146. This address maps to a more human friendly address, such as http://www.sarojpandey.com.np. Web hosts rent out space on their web servers to people or businesses to set up their own websites. The web server allocates a unique website address to each website it hosts.

For example, if you enter the URL *http://www.sarojpandey.com.np/index.html* in your browser, this sends a request to the Web server whose domain name is *sarojpandey.com.np*. The server then fetches the page named *index.html* and sends it to your browser.

Any computer can be turned into a Web server by installing server software and connecting the machine to the Internet. Two leading Web servers are Apache, the most widely installed Web server, and Microsoft's Internet Information Server (IIS).

Apache HTTP Server **FREE & Open Source**

Apache HTTP Server (also referred to as simply "Apache") has, at the time of writing, been the most popular web server on the web since 1996. Apache is developed and maintained by the Apache Software Foundation, which consists of a decentralized team of developers. The software is produced under the Apache license, which makes it free and open source. Apache is available for a range of operating systems, including Unix, Linux, Novell Netware, Windows, Mac OS X, Solaris, and FreeBSD.

Microsoft Personal Web Server (**PWS**) **(Proprietary)** is a scaled-down web server software for Windows operating systems. It has fewer features than Microsoft's Internet Information Services (IIS) and its functions have been superseded by IIS and Visual Studio. Microsoft officially supports PWS on Windows 95, Windows 98, Windows 98 SE, and Windows NT 4.0. Prior to the release of Windows 2000, PWS was available as a free download as well as included on the Windows distribution CDs.

Microsoft Internet Information Services (IIS) **(Proprietary)**

IIS is, at the time of writing, the second most popular web server on the web. It is however, gaining market share, and if the current trend continues, it won't be long before it overtakes Apache.

IIS comes as an optional component of most Windows operating systems. You can install IIS by using *Add/Remove Windows Components* from *Add or Remove Programs* in the Control Panel.

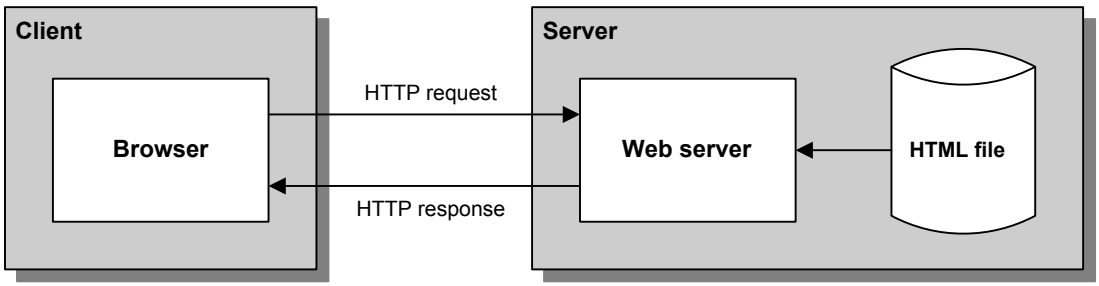
ASSIGNMENT 01

Evolution of the Internet.

Compile on a DOC file and send mail to *saroj@sarojpandey.com.np*

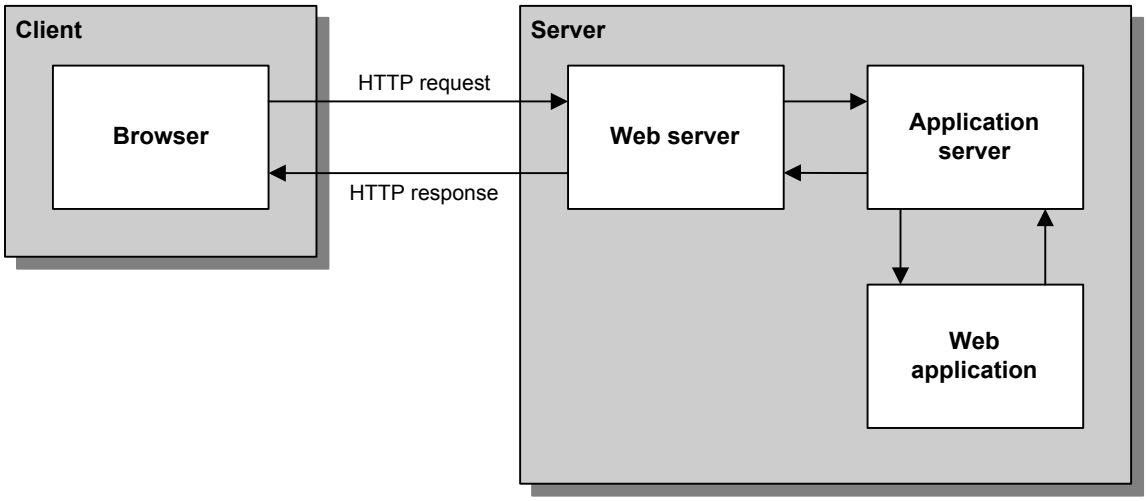
Types of Web Pages and its Processing

Static web page



- A *static web page* is an HTML document that is the same each time it's viewed. It doesn't change in response to user input.
- Static web pages are usually simple HTML files that are stored on the web server with a file extension of .htm or .html. When a browser requests a static web page, the web server retrieves the file from disk and sends it back to the browser.
- A web browser requests a page from a web server by sending the server an HTTP message known as an HTTP *request*.
- The HTTP request includes, among other things, the name of the HTML file being requested and the Internet address of both the browser and the web server.
- A user working with a browser can initiate an HTTP request in several ways. One way is to type the address of a web page, called a *URL*, or *Uniform Resource Locator*, into the browser's address area and press Enter. Another way is to click a link that refers to a web page.
- A web server replies to an HTTP request by sending a message known as an *HTTP response* back to the browser.
- The HTTP response contains the addresses of the browser and the server as well as the HTML document that's being returned.

Dynamic web page



- A *dynamic web page* is an HTML document that's generated by a web application. Often, the web page changes according to information that's sent to the web application by the browser.
- When a web server receives a request for a dynamic web page, the server passes the request to an *application server*.
- The application server executes the web application, which generates an HTML document. This document is returned to the application server, which passes it back to the web server. The web server, in turn, sends the document back to the browser.
- After the page is displayed, the user can interact with it using its controls. Some of those controls let the user *post* the page back to the server, so it's processed again using the data the user entered.
- Each application mapping specifies which application should be run to process files with that extension.
- If the file extension isn't in the list of application mappings, the requested file is returned to the browser without any processing.
- The process that begins with the user requesting a web page and ends with the server sending a response back to the client is called a *round trip*.
- After a web application generates an HTML document, it ends. Then, unless the data the application contains is specifically saved, that data is lost.

HTTP Overview

- HTTP, Hyper Text transfer Protocol is the standard Web transfer protocol.
- The primary technology protocol on the Web that allows linking and browsing.
- The HTTP is the language that Web clients and Web servers use to communicate with each other.
- It is essentially the backbone of the web.
- It is constantly evolving protocol with several versions in use and others are still under development.
- This protocol has two items: the set of requests from browsers to servers and the set of responses going back the other way.
- It is a stateless protocol and does not maintain any information from one transaction to the next, so the next transaction needs to start all over again
- The advantage is that an HTTP server can serve a lot more clients in a given period of time, since there's no additional overhead for tracking sessions from one connection to the next.
- Default Port used by HTTP is 80.

HTTPS

- HTTPS stands for Hyper Text Transfer Protocol Secure. It is a secured version of the HTTP protocol that uses SSL (protocol primarily developed with secure, safe Internet transactions in mind as the encryption layer. SSL layer also offers strong authentication methods.
- HTTPS allows secure ecommerce transactions, such as online banking.
- HTTPS connects on port 443, while HTTP is on port 80
- HTTPS encrypts the data sent and received with SSL, while HTTP sends it all as plain text.
- When a user connects to a website via HTTPS, the website encrypts the session with a digital certificate. A user can tell if they are connected to a secure website if the website URL begins with https:// instead of http://.

- Secure Sockets Layer uses a cryptographic system that encrypts data with two keys.

- SSL transactions are negotiated by means of a key based encryption algorithm between the client and the server, this key is usually either 40 or 128 bits in strength (the higher the number of bits the more secure the transaction).
- When a SSL Digital Certificate is installed on a web site, users can see a padlock icon at the bottom area of the navigator. When an Extended Validation Certificates is installed on a web site, users with the latest versions of Firefox, Internet Explorer or Opera will see the green address bar at the URL area of the navigator.

[***Note***: HTTPS should not be confused with S-HTTP, a security-enhanced version of HTTP. SSL and S-HTTP have very different designs and goals so it is possible to use the two protocols together. Whereas SSL is designed to establish a secure connection between two computers, S-HTTP is designed to send individual messages securely.]

Why Is A SSL Certificate Required?

With booming Internet trends and fraud, most will not submit their private details on the web unless they know that the information they provide is securely transmitted and not accessible for anyone to view.

HTTP Transaction

All HTTP transactions follow the same general format. Each client request and server response has three parts: the request or response line, a header section, and the entity body. ***The client initiates a transaction as follows:***

1. The client contacts the server at a designated port number (by default, 80). Then it sends a document request by specifying an HTTP command called a *method*, followed by a document address, and an HTTP version number. For example:

   GET /index.html HTTP/1.0

   Uses the GET method to request the document *index.html* using version 1.0 of HTTP.

2. Next, the client sends optional header information to inform the server of its configuration and the document formats it will accept. All header information is given line by line, each with a header name and value. For example, this header information sent by the client indicates its name and version number and specifies several document preferences:

   User-Agent: Mozilla/2.02Gold (WinNT; I)
   Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */*

   The client sends a blank line to end the header.

3. After sending the request and headers, the client may send additional data. CGI programs using the POST method mostly use this data. Clients like Netscape Navigator-Gold, to publish an edited page back onto the Web server, may also use it.

The server responds in the following way to the client's request:

1. The server replies with a status line containing three fields: HTTP version, status code, and description. The HTTP version indicates the version of HTTP that the server is using to respond. The status code is a three digit number that indicates the

server's result of the client's request. The description following the status code is just human-readable text that describes the status code. For example, this status line:

HTTP/1.0 200 OK

- Indicates that the server uses version 1.0 of HTTP in its response. A status code of 200 means that the client's request was successful and the requested data will be supplied after the headers.

2. After the status line, the server sends header information to the client about itself and the requested document. For example:

Date: Fri, 20 Sep 1996 08:17:58 GMT
Server: NCSA/1.5.2
Last-modified: Mon, 17 Jun 1996 21:53:08 GMT
Content-type: text/html
Content-length: 2482

A blank line ends the header.

3. If the client's request is successful, the requested data is sent. This data may be a copy of a file, or the response from a CGI program. If the client's request could not be fulfilled, additional data may be a human-readable explanation of why the server could not fulfill the request.

In HTTP 1.0, after the server has finished sending the requested data, it disconnects from the client and the transaction is over unless a *Connection: Key Admin* header is sent. In HTTP 1.1, however, the default is for the server to maintain the connection and allow the client to make additional requests.

Being a stateless protocol, HTTP does not maintain any information from one transaction to the next, so the next transaction needs to start all over again. The advantage is that an HTTP server can serve a lot more clients in a given period of time, since there's no additional overhead for tracking sessions from one connection to the next.

Client-side scripting is the class of computer programs on the web that are executed in *client-side*, by the user's web browser, instead of *server-side* (on the web server). This type of computer programming is an important part of the Dynamic HTML (DHTML) concept, enabling web pages to be scripted; that is, to have different and changing content depending on user input, environmental conditions (such as the time of day), or other variables. Web authors write client-side scripts in languages such as JavaScript (Client-side JavaScript) and VBScript.

Client-side scripts are embedded within an HTML document (hence known as an "embedded script"), but they may also be contained in a separate file, which is referenced by the document (or documents) that uses it (known as an

"external script"). Upon request, the necessary files are sent to the user's computer by the web server (or servers) on whom they reside. The user's web browser executes the script, and then displays the document, including any visible output from the script. By viewing the file that contains the script, users may be able to see its source code.

Server-side scripting is a web server technology in which a user's request is fulfilled by running a script directly on the web server to generate dynamic web pages. It is usually used to provide interactive web sites that interface to databases or other data stores. This is different from client-side scripting where the viewing web browser, usually in JavaScript, runs scripts. The primary advantage to server-side scripting is the ability to highly customize the response based on the user's requirements or queries into data stores. Server side scripts are written in languages such as Perl, PHP, ASP.NET etc.

SSS produce output in a format understandable by web browsers (usually HTML), which is then sent to the user's computer. The user cannot see the script's source code (unless the author publishes the code separately), and may not even be aware that a script was executed.

From security point of view, server-side scripts are never visible to the browser as these scripts are executed on the server and emit HTML corresponding to user's input to the page.

Programs that run on a user's local computer without ever sending or receiving data over a network are not considered clients, and so the operations of such programs would not be considered client-side operations.

Comparison: Client-side scripts have greater access to the information and functions available on the user's browser, whereas server-side scripts have greater access to the information and functions available on the server. Server-side scripts require that their languages interpreter be installed on the server, and produce the same output regardless of the client's browser, operating system, or other system details. Client-side scripts do not require additional software on the server; however, they do require that the user's web browser understands the scripting language in which they are written.

ASSIGNMENT 02

SMTP and POP: Why they are important?

Compile on a DOC file and send mail to *saroj@sarojpandey.com.np*

FTP & Its Types

***F**ile **T**ransfer **P**rotocol,* the protocol for exchanging files over the Internet. FTP works in the same way as HTTP. FTP uses the Internet's TCP/IP protocols to enable data transfer. FTP is most commonly used to download a file from a server using the Internet or to upload a file to a server (e.g., uploading a Web page file to a server).

- File Transfer Protocol (FTP) is a method of transferring files from a client to a server or vice versa.
- Files are transferred over the Internet using TCP/IP protocol.
- FTP is old-time protocol that maintains two simultaneous connections.
- The first connection uses the telnet remote login protocol to log the client into an account and process commands via the *protocol interpreter*.
- The second connection is used for the *data transfer process*.
- Whereas the first connection is maintained throughout the FTP session, the second connection is opened and closed for each file transfer.
- The FTP protocol also enables an FTP client to establish connections with two servers and to act as the third-party agent in transferring files between the two servers.
- FTP servers rarely change, but new FTP clients appear on a regular basis.
- Although FTP is a command-line oriented protocol, the new generation of FTP clients hides this orientation under a GUI environment.
- A client makes a TCP connection to the server's port 21. This connection, called the *control connection*, remains open for the duration of the session, with a second connection, called the *data connection*, either opened by the server from its port 20 to a negotiated client port or opened by the client from an arbitrary port to a negotiated server port as required to transfer file data.

[FTP is a TCP based service exclusively. There is no UDP component to FTP. FTP is an unusual service in that it utilizes two ports, a 'data' port and a 'command' port (also known as the control port). Traditionally these are port 21 for the command port and port 20 for the data port. The confusion begins however, when we find that depending on the mode, the data port is not always on port 20.]

Types of FTP

- Active FTP

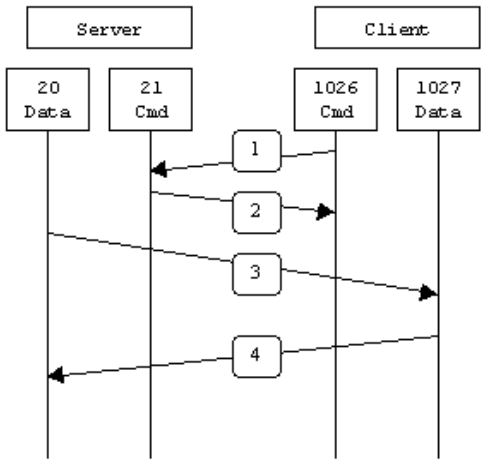
In active mode FTP the client connects from a random unprivileged port (N > 1023) to the FTP server's command port, port 21. Then, the client starts listening to port N+1 and sends the FTP command PORT N+1 to the FTP server. The server will then connect back to the client's specified data port from its local data port, which is port 20.

From the server-side firewall's standpoint, to support active mode FTP the following communication channels need to be opened:

- FTP server's port 21 from anywhere (Client initiates connection)

- FTP server's port 21 to ports > 1023 (Server responds to client's control port)
- FTP server's port 20 to ports > 1023 (Server initiates data connection to client's data port)
- FTP server's port 20 from ports > 1023 (Client sends ACKs to server's data port)

When drawn out, the connection appears as follows:



- In step 1, the client's command port contacts the server's command port and sends the command PORT 1027.
- The server then sends an ACK back to the client's command port in step 2.
- In step 3 the server initiates a connection on its local data port to the data port the client specified earlier.
- Finally, the client sends an ACK back as shown in step 4.

The main problem with active mode FTP actually falls on the client side. The FTP client doesn't make the actual connection to the data port of the server - it simply tells the server what port it is listening on and the server connects back to the specified port on the client. From the client side firewall this appears to be an outside system initiating a connection to an internal client - something that is usually blocked.

- Passive FTP

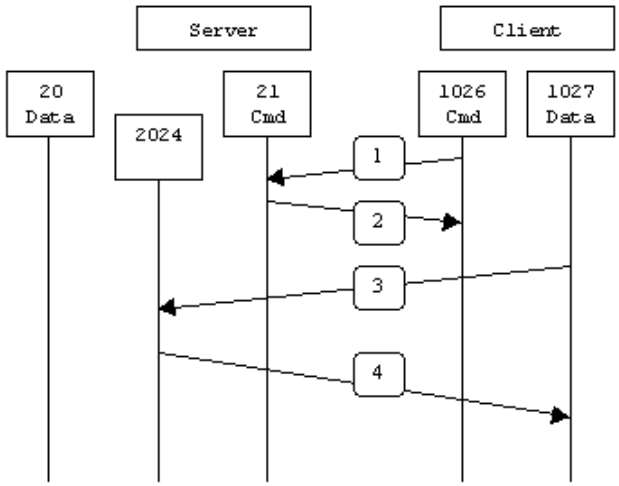
In order to resolve the issue of the server initiating the connection to the client a different method for FTP connections was developed. This was known as passive mode, or PASV, after the command used by the client to tell the server it is in passive mode.

In passive mode FTP the client initiates both connections to the server, solving the problem of firewalls filtering the incoming data port connection to the client from the server. When opening an FTP connection, the client opens two random unprivileged ports locally (N > 1023 and N+1). The first port contacts the server on port 21, but instead of then issuing a PORT command and allowing the server to connect back to its data port, the client will issue the PASV command. The result of this is that the server then opens a random unprivileged port (P > 1023) and sends the PORT P command back to the client. The client then initiates the connection from port N+1 to port P on the server to transfer data.

From the server-side firewall's standpoint, to support passive mode FTP the following communication channels need to be opened:

- FTP server's port 21 from anywhere (Client initiates connection)
- FTP server's port 21 to ports > 1023 (Server responds to client's control port)
- FTP server's ports > 1023 from anywhere (Client initiates data connection to random port specified by server)
- FTP server's ports > 1023 to remote ports > 1023 (Server sends ACKs (and data) to client's data port)

When drawn, a passive mode FTP connection looks like this:



- In step 1, the client contacts the server on the command port and issues the PASV command.
- The server then replies in step 2 with PORT 2024, telling the client which port it is listening to for the data connection.
- In step 3 the client then initiates the data connection from its data port to the specified server data port. Finally, the server sends back an ACK in step 4 to the client's data port.

- While passive mode FTP solves many of the problems from the client side, it opens up a whole range of problems on the server side. The biggest issue is the need to allow any remote connection to high numbered ports on the server. Fortunately, many FTP daemons, including the popular WU-FTPD allow the administrator to specify a range of ports, which the FTP server will use. The second issue involves supporting and troubleshooting clients, which do (or do not) support passive mode. FTP Clients like FileZilla give user option to select the FTP mode.

With the massive popularity of the World Wide Web, many people prefer to use their web browser as an FTP client. Most browsers only support passive mode when accessing ftp:// URLs. This can either be good or bad depending on what the servers and firewalls are configured to support.

What is HTML?

- HTML or **H**yper**T**ext **M**arkup **L**anguage is designed to specify the logical organization of a document, with important hypertext extensions.
- HTML instructions divide the text of a document into blocks called *elements*.
- These can be divided into two broad categories:
- Those that define how the **BODY** of the document is to be displayed by the browser, and
- Those that define information about the document, such as the **title** or relationships to other documents.
- The detailed rules for HTML (the names of the tags/elements, how they can be used) are defined using another language known as the SGML (**S**tandard **G**eneralized **M**arkup **L**anguage).
- HTML is a set of special codes that can be embedded in text to add formatting and linking information.
- HTML is the language interpreted by a Browser.
- The HTML file must have an extension "**.htm**" or "**.html**".
- Any text editor can be used to create HTML file.

HTML stands for **H**yper **T**ext **M**arkup **L**anguage. HTML is a presentation language. We use HTML language to display information according to our need. An HTML file is a text file containing small **markup tags**. The markup tags tell the Web browser **how to display** the page. An HTML file must have an .**htm** or .**html** file extension. An HTML file can be created using a **simple text editor**.

HTML is very popular language used on web because of its interoperability. HTML language is platform independent i.e. HTML files can be opened on any platform. HTML files can be written using a simple text editor like notepad, which is present in all the operating system.

HTML language is used to create web pages. Web pages can be viewed using application software called a **Web browser**. Popular browsers are **Internet Explorer, Mozilla Firefox and Netscape Navigator**. A web browser parses the HTML file containing markups (html tags) and displays the information with the proper format as specified in the HTML document. HTML tags are also called mark-up. HTML tags are surrounded by the two characters < and >. The surrounding characters are called angle brackets. HTML tags normally come in pairs like and . The first tag in a pair is the start tag, the second tag is the end tag. The text between the start and end tags is the element content. HTML tags are not case sensitive; means the same as <B>.

Structure of an HTML document is shown below:

```
<html>
<head>
    <title> Title of page </title>
</head>
<body>
    This is the place where the information to be displayed in a web page is written.
</body>
</html>
```

Every html document must start with <html> tag. It shows the starting of HTML document. <head> tag contains information like the title of the document and other information which describes about the content of the document.

BODY part of a HTML document contains the information and its format to be displayed by the browser. HEAD part of a HTML document contains the information that is not displayed on the browser window. It defines information `about' the document, such as the title or relationships to other documents.

<body> tag is the place where we write all the information that is to be displayed in the web browser. It also contains other tags which defines how information are to be displayed in the web browser. <body> tag shows the starting of the body tag and </body> tag shows the ending of the body tag. </html> shows the ending of the HTML document. Every ending tag must have a forward slash as shown in </html> tag.

HTML discards whitespaces. HTML only considers a single space as a space. The browser automatically discards rest of the whitespace. Hence, we can use as much whitespace as we want while creating our HTML document. This makes html document easy to read or edit.

Versions of HTML

HTML 2.0

- It set the standard for core HTML features based upon current practice in 1994.

HTML 3.2

- W3C's first Recommendation for HTML which represented the consensus on HTML features for 1996.
- **HTML 3.2** added widely-deployed features such as tables, applets, text-flow around images, superscripts and subscripts, while providing backwards compatibility with the existing HTML 2.0 standard.

HTML 4.0

- First released as a W3C Recommendation on 18 December 1997.
- A second release was issued on 24 April 1998 with changes limited to editorial corrections.
- This specification has now been superseded by HTML 4.01.

HTML 4.01

- HTML 4.01 is the current official standard.
- It includes support for most of the proprietary extensions, plus support for extra features (Internationalized documents, support for **C**ascading **S**tyle **S**heets, extra TABLE, FORM, and JavaSctipt enhancements), that are not universally supported.
- This is the last version of HTML.
- After this XHTML was released which stands for e**X**tensible **H**yper**T**ext **M**arkup **L**anguage.

HTML 5.0

- This is the new version of HTML with many exciting new features. This version is still under development.

ASSIGNMENT 03

HTML 5.0 features: What Web Developers Can Use NOW?

Compile on a DOC file and send mail to *saroj@sarojpandey.com.np*

HTML Elements/Tags

- The HTML instructions, along with the text to which the instructions apply, are called HTML *elements*.
- The HTML instructions are themselves called ***tags***, and look like <element_name> -- that is, they are simply the element name surrounded by left and right angle brackets.
- The content in the web-page is written after the starting tag, and closed with the end tag.
- E.g: **<element_name>** text to be written HERE **</element_name>**
- The end tag has slash character in front of it.

- HTML tags are not case sensitive; **<b>** means same as **<B>**.

Empty Elements

- Some elements are *empty* -- that is, they do not affect a block of the document in some way.
- These elements do not require an ending *tag*.
  - An example is the <HR> element, which draws a horizontal line across the page.

HTML Tag Attributes

- Many elements can have arguments that pass parameters to the interpreter handling the element.
- These arguments are called *attributes* of the element.
- An attribute is a customizable option for a tag.
- In other words, attributes are used to define the properties of a tag.
  - Example: **<p** align = "left"**>** Trial Example **</p>**.
  - In the above example the align attribute allows you to specify how text in a paragraph is arranged on the page.
- Not all tags support attributes.
- Some tags support multiple attributes, and the attributes are listed one after another in the start tag, seperated by space.
- Attributes are always set to the opening tag.

HTML Meta

```
<head>
<meta name="description" content="Free Web tutorials">
<meta name="keywords" content="HTML,CSS,XML,JavaScript">
<meta name="author" content="Hege Refsnes">
<meta charset="UTF-8">
</head>
```

Definition and Usage

Metadata is data (information) about data.

The <meta> tag provides metadata about the HTML document. Metadata will not be displayed on the page, but will be machine parsable.

Meta elements are typically used to specify page description, keywords, author of the document, last modified, and other metadata.

The metadata can be used by browsers (how to display content or reload page), search engines (keywords), or other web services.

Tips and Notes

Note: <meta> tags always goes inside the <head> element.

Note: Metadata is always passed as name/value pairs.

Note: The content attribute MUST be defined if the name or the http-equiv attribute is defined. if none of these are defined, the content attribute CANNOT be defined.

Differences Between HTML 4.01 and HTML5

The scheme attribute is not supported in HTML5.

HTML5 has a new attribute, charset, which makes it easier to define charset:

- HTML 4.01: <meta http-equiv="content-type" content="text/html; charset=UTF-8">
- HTML5: <meta charset="UTF-8">

- **Example 1 - Define keywords for search engines:**
- `<meta name="keywords" content="HTML, CSS, XML, XHTML, JavaScript">`
- **Example 2 - Define a description of your web page:**
- `<meta name="description" content="Free Web tutorials on HTML and CSS">`
- **Example 3 - Define the author of a page:**
- `<meta name="author" content="Hege Refsnes">`
- **Example 4 - Refresh document every 30 seconds:**
- `<meta http-equiv="refresh" content="30">`

HTML4.01 Tags Lists

TITLE	TAG	DESCRIPTION
Basic Elements		
Document Type	<HTML> </HTML>	document root element, beginning and end of file
Title	<TITLE> </TITLE>	document title, must be in header
Header	<HEAD> </HEAD>	descriptive info, such as title
Body	<BODY> </BODY>	bulk of the page, notes body of document
Formatting		
Bold	<B> </B> or	bold text style
Italic	<I> </I>	italic text style
Underline	<U> </U>	underlined text (not widely implemented)
Strikeout	<STRIKE> </STRIKE>	strike-through text (not widely implemented)
Strikeout	<S> </S>	strike-through text (not widely implemented)
Subscript		subscript numbers like footnotes
Superscript		superscript numbers like cross - reference numbers
Pre formatted	<PRE> </PRE>	pre formatted text (display text spacing as-is)
Center	<CENTER> </CENTER>	centers text and images
Blinking	<BLINK> </BLINK>	blinking text, Netscape only
Font Size	<FONT SIZE=?> </FONT>	local font size(ranges from 1-7)
Change Font Size	<FONT SIZE="+|-?"> </FONT>	controls font size rendered
Font Color	<FONT COLOR="#$$$$$$"> </FONT>	controls font color rendered
Select Font	<FONT FACE="***"> </FONT>	the style of the text, such as Times New Roman
Marquee	<MARQUEE> </MARQUEE>	scrolling text (IE only)
Links		
Link Something	<A HREF="URL"> </A>	links text or graphic to another URL
Link to Location	<A HREF="URL#***"> </A>	links text or graphic an anchor in an other document

Note: This handout is for simple reference only. Do not completely depend on it.

Link to Location in Current Page	<A HREF="#***"> </A>	links text or graphic an anchor in current document
Target Window	<A HREF="URL" TARGET="***"> </A>	links text or graphic to a URL in a new browser widow
Action on Click	<A HREF="URL" ONCLICK="***"> </A>	takes effect when user clicks on the item (Javascript)
Mouseover Action	<A HREF="URL" ONMOUSEOVER="***"> </A>	takes effect when user moves pointer over item
Link to Email	<A HREF="mailto:@"> </A>	creates blank e-mail to indicated address with visitor's default e-mail client
Graphics and Sound		
Display Image	<IMG SRC="URL">	displays image from the indicated URL
Alignment	<IMG SRC="URL" ALIGN= TOP|BOTTOM|MIDDLE|LEFT|RIGHT>	aligns the image
Dimensions	<IMG SRC="URL" WIDTH=? HEIGHT=?>	the dimensions, in pixels, of the image
Border	<IMG SRC="URL" BORDER=?>	border, in pixels, around the image
Dividers		
Paragraph	<P> </P>	paragraph (closing tag often unnecessary)
Align Text	<P ALIGN=LEFT|CENTER|RIGHT> </P>	aligns paragraph
Justify Text	<P ALIGN=JUSTIFY> </P>	justify's paragraph's text
Line Break	 	a single carriage return
Horizontal Rule	<HR>	horizontal line
Alignment	<HR ALIGN=LEFT|RIGHT|CENTER>	alignment of horizontal line
Thickness	<HR SIZE=?>	thickness, in pixels, of horizontal line
Width	<HR WIDTH=?>	width, in pixels, of horizontal line
Width Percent	<HR WIDTH="%">	width(as a percentage of page width), in pixels, of horizontal line

Solid Line	<HR NOSHADE>	horizontal line without the 3D cutout look
No Break	<NOBR> </NOBR>	prevents line breaks
Structural Elements		
Heading	<H?> </H?>	document header, the ? defines 6 levels (#'s 1-6)
Strong Emphasis	<STRONG> </STRONG>	strongly emphasized text, usually displayed as bold
Address	<ADDRESS> </ADDRESS>	author information
Large Font Size	<BIG> </BIG>	uses a large text size
Small Font Size	<SMALL> </SMALL>	use a small text size
Backgrounds		
Tiled Background	<BODY BACKGROUND= "URL">	causes the image to tile as the background of the page
Watermark	<BODY BGPROPERTIES= "FIXED">	Static image which remains in the same location as visitors scroll.
Background Color	<BODY BGCOLOR= "#$$$$$$">	solid background color of the page
Text Color	<BODY TEXT="#$$$$$$">	color of the text throughout the page
Link Color	<BODY LINK="#$$$$$$">	color of all links throughout the page
Visited Link	<BODY VLINK="#$$$$$$">	color of all links that have already been clicked on by visitor
Active Link	<BODY ALINK="#$$$$$$">	color of link while being selected
Lists		
Unordered List	<UL> </UL>	list with bulleted items
List Item	<LI> </LI>	indicates an item on the list
Bullet Type	<UL TYPE=DISC|CIRCLE|SQUARE>	shape of bullet for the whole list
Bullet Type	<LI TYPE=DISC|CIRCLE|SQUARE>	shape of bullet for specific list item
Ordered List	<OL> <LI> </OL>	numbered list
Numbering Type	<OL TYPE=A|a|I|i|1>	type of numbering for the whole list

Numbering Type	<LI TYPE=A|a|I|i|1>	type of numbering for specific list item
Starting Number	<OL START=?>	starting number for list
Starting Number	<LI VALUE=?>	starting number for this & subsequent items
Definition List	<DL> </DL>	a list of definitions
Definition Term	<DT> </DT>	definition term
Definition	<DD> </DD>	definition of a term
Menu List	<MENU> </MENU>	display menu type list
Directory List	<DIR> </DIR>	directory link
Tables		
Define Table	<TABLE> </TABLE>	signals the beginning of a table
Table Alignment	<TABLE ALIGN= LEFT|RIGHT|CENTER>	aligns the table within the browser window
Table Border	<TABLE BORDER=?> </TABLE>	border of table, you can set the value (aka width)
Cell Spacing	<TABLE CELLSPACING=?>	places specific amount of space between the individual cells within a table
Cell Padding	<TABLE CELLPADDING=?>	places specific amount of space between the cells border and its contents
Desired Width	<TABLE WIDTH=?>	width of table in pixels
Width Percent	<TABLE WIDTH=%>	width of table in percentage of page
Table Color	<TABLE BGCOLOR="#$$$$$$"> </TABLE>	overall background color of table
Border Color	<TABLE BORDERCOLOR="#$$$$$$"> </TABLE>	the color of the table border
Table Row	<TR> </TR>	table row
Alignment	<TR ALIGN= LEFT| RIGHT| CENTER| MIDDLE| BOTTOM>	alignment of the table row
Table Cell	<TD> </TD>	specific table cell, must appear within table rows
Alignment	<TD ALIGN= LEFT|RIGHT|CENTER	alignment of the table cell

	VALIGN= TOP|MIDDLE|BOTTOM>	
Columns to Span	<TD COLSPAN=?>	identifies the the number of columns the cell should span
Rows to Span	<TD ROWSPAN=?>	identifies the the number of rows the cell should span
Desired Width	<TD WIDTH=?>	width of cell in pixels
Width Percent	<TD WIDTH="%">	width of cell as percentage of table
Cell Color	<TD BGCOLOR="#$$$$$$">	background color of table cell
Header Cell	<TH> </TH>	table cell for header information (bold & centered)
Alignment	<TH ALIGN= LEFT| RIGHT| CENTER| MIDDLE|BOTTOM>	alignment of the header cell
Table Body	<TBODY>	identifies the specific body section of the table
Table Footer	<TFOOT> </TFOOT>	separates group of cells to serve as footer material for the table (must come before <THEAD>)
Table Header	<THEAD> </THEAD>	separates group of cells to serve as header material for the table
Table Caption	<CAPTION> </CAPTION>	caption for a table
Alignment	<CAPTION ALIGN=TOP|BOTTOM|LEFT|RIGHT>	alignment for the caption of a table

HTML Lists

- HTML provides three type of lists.
- They are listed below:

1. **Ordered List:**
   - A list of multi-line paragraphs, listed separately and ordered numerically in some way.
   - The list items are marked with numbers.
   - <OL ...> creates an ordered list.
   - "Ordered" means that the order of the items in the list is important.
   - By default, the number starts with 1,2,3…….
   - An ordered list starts with the <ol> tag.

- Each list item starts with the <li> tag.
    - Example:

```
<ol>
<li>Coffee</li>
<li>Milk</li>
</ol>
```

Here is how it looks in a browser:

1. Coffee
2. Milk

2. Unordered List:

- A list of multi-line paragraphs, listed separately and usually marked by a bullet or similar symbol (Unordered List)
- <UL ...> creates an unordered list.
- The *unordered* part means that the items in the list are not in any particular order.
- The list items are marked with bullets (typically small black circles).
- An unordered list starts with the <ul> tag.
- Each list item starts with the <li> tag.
    - Example:

```
<ul>
<li>Coffee</li>
<li>Milk</li>
</ul>
```

Here is how it looks in a browser:

- Coffee
- Milk

3. Definition List:

- A definition list is **not** a list of items.
- This is a list of terms and explanation of the terms.
- A definition list starts with the <dl> tag.
- Each definition-list term starts with the <dt> tag.

- Each definition-list definition starts with the <dd> tag.
    - Example:

```
<dl>
<dt>Coffee</dt>
<dd>Black hot drink</dd>
<dt>Milk</dt>
<dd>White cold drink</dd>
</dl>
```

Here is how it looks in a browser:

Coffee
  Black hot drink
Milk
  White cold drink

HTML Links

- A link is a connection from one Web resource to another.
- A *link* has two ends -- called *anchors* -- and a direction.
- The link starts at the "source" anchor and points to the "destination" anchor, which may be any Web resource (e.g., an image, an HTML document, an element within an HTML document, etc.).
- The text or an image that provides such linkages is called hypertext, hyperlink, or hotspot.

What is Hyperlink?

- A Hyperlink is a connection between an HTML element such as text, an image, or anything else on a page and other resource.
- That link might be to another web page, an external image, or an e-mail address.

Difference between Hyperlink and Normal HTML Text:

- Appears in blue color.
    - The default color setting in a browser for hyperlink text or image.
    - The color can be set dynamically via HTML program if required.
- The Hyperlink text/image is underlined.

- When the mouse cursor is placed over it, the standard arrow shaped mouse cursor changes to the shape of a hand.

Changing the color of Links:

- To change the link color there are three attributes that can be specified with the **<body>** tag.
- These are:
  - LINK (Normal)
  - ALINK (Active)
  - VLINK (Visited)

Types of Hyperlink

There are three types of Hyperlinks:

1. **Inter-page Hyperlink**

- In this type of link the control flows from one-page to another.

Example:

<a HREF="myExample.htm"> Click for Example ****

You can specify the relative as well as the absolute path of the file that you want to call.

2. **Intra-page Hyperlink**

- Intra-page Hyperlink is a link within a same page.
- Sometimes, a jump is required to a different location in the same document.
- Since the jump has to be targeted to a specific location the two steps need to perform.
  a) Identify the location with a name and
  b) Jump to that location using the name.

Example:

<a name ="top"> The HTML text is written here ****

<a HREF="#top"> Goto Top ****

3. **Email Hyperlink**

- This type of Hyperlink is used especially to write e-mail.
- The link does not open any web-pages but opens the outlook express for writing mail.
- You can write the mail and send.

Steps:

-> First type any text like:

Email: info@kcc.edu.np

Surround the email address with the anchor tags i.e. <a>, but instead linking to the web page, use the '*mailto'* command to link it to an e-mail program.

Email: **<a** HREF="mailto: info@kcc.edu.np"> info@kcc.edu.np ****

-> Save the page and view it in browser.

4. **External Links**

You can also have external links like links, when clicking upon them you can jump to next web page.

In such scenario you have to give the path of web page like:

<a HREF="http://www.google.com"> Goto Google ****

HTML Forms

Forms are the most popular way to make web pages interactive. A form on a web page looks similar to a form on a sheet of paper that allows the user to enter requested information and submit it for further processing. A form can have different types of form elements for different purpose like textbox, list box, checkbox, radio buttons, dropdown menus, text area etc.

1. HTML Text Fields

type - Determines what kind of input field it will be. Possible choices are text, submit, and password.

name - Assigns a name to the given field so that you may reference it later.

size - Sets the horizontal width of the field. The unit of measurement is in blank spaces.

maxlength - Dictates the maximum number of characters that can be entered.

HTML Code:

```
<form method="post">
    Name: <input type="text" size="10" maxlength="40" name="name"> <br />
    Password: <input type="password" size="10" maxlength="10" name="password">
</form>
```

'Do not use the password feature for security purposes. The data in the password field is not encrypted and is not secure in any way.'

2. **Submit Buttons**

HTML Code:

```
<form method="post">
    Name: <input type="text" size="10" maxlength="40" name="name"> <br />
    Password: <input type="password" size="10" maxlength="10" name="password"><br />
    <input type="submit" value="Send">
</form>
```

3. **HTML Radio Buttons**

Radio buttons are a popular form of interaction. You may have seen them on quizzes, questionnaires, and other web sites that give the user a multiple-choice question. Below are a couple attributes you should know that relate to the radio button.

value - specifies what will be sent if the user chooses this radio button. Only one value will be sent for a given group of radio buttons.

name - defines which set of radio buttons that it is a part of.

HTML Code:

```
<form method="post">
What kind of shirt are you wearing? <br />
Shade:
<input type="radio" name="shade" value="dark">Dark
<input type="radio" name="shade" value="light">Light <br />
Size:
<input type="radio" name="size" value="small">Small
<input type="radio" name="size" value="medium">Medium
<input type="radio" name="size" value="large">Large <br />
<input type="submit" value="Email Myself">
</form>
```

4. HTML Check Boxes

Check boxes allow for multiple items to be selected for a certain group of choices. The check box's name and value attributes behave the same as a radio button.

HTML Code:

```
<form method="post">
Select your favorite cartoon characters.
        <input type="checkbox" name="toon" value="Goofy">Goofy
        <input type="checkbox" name="toon" value="Donald">Donald
        <input type="checkbox" name="toon" value="Bugs">Bugs Bunny
        <input type="checkbox" name="toon" value="Scoob">Scooby Doo
        <input type="submit" value="Email Myself">
</form>
```

5. HTML Drop down Lists (Known as Combo Box)

Drop down menu are created with the <select> and <option> tags. <select> is the list itself and each <option> is an available choice for the user.

HTML Code:

```
<form method="post">
College Degree?
        <select name="degree">
                <option>Choose One</option>
                <option>Some High School</option>
                <option>High School Degree</option>
                <option>Some College</option>
                <option>Bachelor's Degree</option>
                <option>Doctorate</option>
        <input type="submit" value="Email Yourself">
        </select>
</form>
```

TRY IT YOURSELF

HTML Code:

```
        <form method="post" action="mailto:youremail@email.com">
        Musical Taste
        <select multiple name="music" size="4">
                <option value="emo" selected>Emo</option>
                <option value="metal/rock" >Metal/Rock</option>
                <option value="hiphop" >Hip Hop</option>
```

```
        <option value="ska" >Ska</option>
        <option value="jazz" >Jazz</option>
        <option value="country" >Country</option>
        <option value="classical" >Classical</option>
        <option value="alternative" >Alternative</option>
        <option value="oldies" >Oldies</option>
        <option value="techno" >Techno</option>
</select>
<input type="submit" value="Email Yourself">
</form>
```

6. HTML Text Areas

Text areas serve as an input field for viewers to place their own comments onto. Forums and the like use text areas to post what you type onto their site using scripts. For this form, the text area is used as a way to write comments to somebody.

Rows and columns need to be specified as attributes to the <textarea> tag. Rows are roughly 12pixels high, the same as in word programs and the value of the columns reflects how many characters wide the text area will be. i.e. The example below shows a text area 5 rows tall and 20 characters wide.

HTML Code:

```
<form method="post”>
        <textarea rows="5" cols="20" wrap="physical" name="comments">
                        Enter Comments Here
        </textarea>
        <input type="submit" value="Email Yourself">
</form>
```

Note that any text placed between the opening and closing textarea tags will show up inside the text area when the browser views it.

[Self Study: Forms on HTML 5]

Entities References used in HTML

In HTML we cannot directly use the special symbols so we use a technique called Entities References. With this we can keep any symbols in a web page. It takes a form: **&Entity_Name;**

Some of the mostly used symbols and their corresponding entities are as follows:

Result	Description	Entity Name
	non-breaking space	
<	less than	&lt;
>	greater than	&gt;
&	ampersand	&amp;
"	quotation mark	&quot;
'	apostrophe	' (does not work in IE)
¢	cent	&cent;
£	pound	&pound;
¥	yen	&yen;
€	euro	&euro;
§	section	&sect;
©	copyright	&copy;
®	registered trademark	&reg;
×	multiplication	&times;
÷	division	&divide;

And more…

Points for review:

- HTML stands for **H**yper **T**ext **M**arkup **L**anguage.
- HTML file consist of tags also called markups to display information in an arranged way.
- Tags in HTML are predefined i.e. we cannot create our own tags.
- Web browser is application software that is used to view web pages created using HTML.
- HTML document is mainly divided into two parts: head and body.
- The current version of HTML that we are using now is HTML4.
- W3C stands for World Wide Web consortium is a body which looks after the standardization of HTML language.
- HTML file ends with .htm or .html extension.
- Every starting tag has its corresponding ending tag in HTML for e.g. …. .
- HTML also has empty tags like <br /> which is used to break row.
- The browser ignores comments. <!-- … … … … -->
- Tag can also have attributes, which are used to define the properties of a tag.

  For e.g. …

Note: This handout is for simple reference only. Do not completely depend on it.

- Character entities are used to display some special characters, which cannot be typed from the keyboard. It is also used to display some of the characters, which are forbidden to be written as element content.

CASE STUDY 01

Different Internet services providers on your City and their way of connecting.

Compile on a DOC file and send mail to *saroj@sarojpandey.com.np*

HTML Assistants

HTML Assistant is a software tool, which helps you to build web pages with HTML without a lot of extraneous features. HTML assistants are favorite among Web developers which makes HTML page design easy.

E.g.

BROOKLYN NORTH HTML Assistant Pro
Adobe Dreamweaver
Microsoft FrontPage

HTML Assistant Basic Features

- Text editor
- Color coding
- FTP or Site manager
- Search and replace / Extended search and replace (across multiple documents)
- Can edit JavaScript
- Additional CSS assistance
- Spell checking
- Form wizard
- Includes a built-in RTF to HTML converter.
    - Etc.

HTML Editors

A **HTML editor** is an authoring software program that is used to create content for web sites. HTML software is easy to use since it has a feature that is known as **WYSIWYG**.

It is a software application for creating web pages. Although the HTML markup of a web page can be written with any text editor, specialized HTML editors can offer convenience and added functionality. For example, many HTML editors work not only with HTML, but also with related technologies such as CSS, XML and JavaScript.

When you design web pages you want to use editor features that are simple to understand. You can buy and download HTML management software from the Internet as well as templates that will help you create web pages for your business or personal use.

HTML editors are also great for creating tables; building borders around images and changing background color in no time at all. You can easily change the design of your website in just a few minutes to reflect your business style.

Examples:

- Vim Editor, gEdit Editor, Emacs Editor, BlueFish Editor (Linux)
- Notepad, Wordpad, Notepad+ (Windows)
- BlueFish Editor, TextEdit, TextWrangler (Mac)

WYSIWYG - ***W****hat* ***Y****ou* ***S****ee* ***I****s* ***W****hat* ***Y****ou* ***G****et*. The term is used in computing to describe a system in which content (text and graphics) displayed onscreen during editing appears in a form exactly corresponding to its appearance when printed or displayed as a finished product, which might be a printed document, web page, or slide presentation.

WYSIWYG implies a user interface that allows the user to view something very similar to the end result while the document is being created. In general WYSIWYG implies the ability to directly manipulate the layout of a document without having to type or remember names of layout commands. The actual meaning depends on the user's perspective.

Examples:

- Adobe Dreamweaver (With this software user can view the output as he writes the HTML code.)
- Microsoft Front Page

CSS

- CSS is an acronym for **C**ascading **S**tyle **S**heets.
- A CSS (Cascading Style Sheet) file allows to separate your web sites HTML content from it's style. HTML file is used to arrange the content, but all of the presentation (fonts, colors, background, borders, text formatting, link effects & so on…) is accomplished with a CSS.
- Styles define **how to display** HTML elements
- **External Style Sheets** can save a lot of work
- External Style Sheets are stored in *.**CSS files**

What can be done with CSS?

CSS is a style language that defines layout of HTML documents. For example, CSS covers fonts, colors, margins, lines, height, width, background images, advanced positions and many other things.

HTML can be (mis-)used to add layout to websites. But CSS offers more options and is more accurate and sophisticated. All browsers support CSS.

What is the difference between CSS and HTML?

HTML is used to structure content. CSS is used for formatting structured content.

Back in the good old days when a guy called Tim Berners Lee invented the World Wide Web, the language HTML was only used to add structure to text. An author could mark his text by stating, "this is a headline" or "this is a paragraph" using HTML tags such as <h1> and <p>.

As the Web gained popularity, designers started looking for possibilities to add layout to online documents. To meet this demand, the browser producers (at that time Netscape and Microsoft) invented new HTML tags such as for example <font> which differed from the original HTML tags by defining layout - and not structure. This also led to a situation where original structure tags such as <table> were increasingly being misused to layout pages instead of adding structure to text. Many new layout tags such as <blink> were only supported by one type of browser. "You need browser X to view this page." became a common disclaimer on web sites.

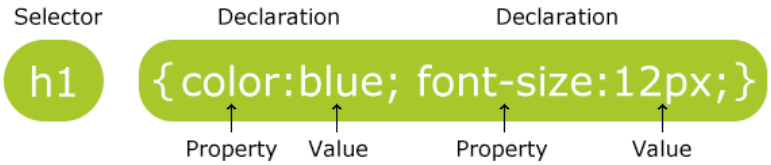
CSS was invented to remedy this situation by providing web designers with sophisticated layout opportunities supported by all browsers. At the same time, separation of the presentation style of documents from the content of documents makes site maintenance a lot easier.

Which benefits will CSS give?

CSS was a revolution in the world of web design. The concrete benefits of CSS include:

- control layout of many documents from one single style sheet;
- more precise control of layout;
- apply different layout to different media-types (screen, print, etc.);
- numerous advanced and sophisticated techniques.

The basic CSS syntax



Let's say we want a nice red color as the background of a webpage:

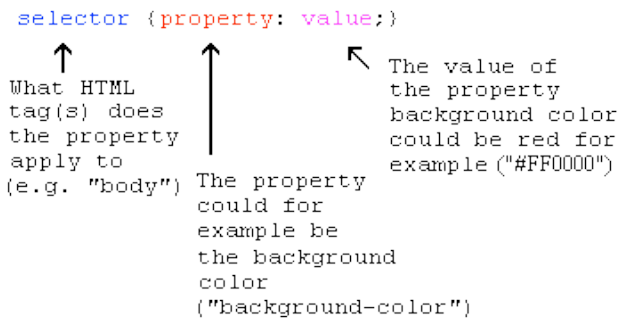
Using **HTML** we could have done it like this:

```
<body bgcolor="#FF0000">
```

With **CSS** the same result can be achieved like this:

```
body {background-color: #FF0000;}
```

As you will note, the codes are more or less identical for HTML and CSS. The above example also shows you the fundamental CSS model:



CSS Comments

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

```
/* This is a comment */
p
{
text-align:center;

/* This is another comment */
}
```

Applying CSS to an HTML document

There are three ways you can apply CSS to an HTML document. These methods are all outlined below. We recommend that you focus on the third method i.e. external.

Method 1: In-line (the attribute style)

One way to apply CSS to HTML is by using the HTML attribute style. Building on the above example with the red background color, it can be applied like this:

```
<html>
<head> <title>Example</title></head>
<body style="background-color: #FF0000;">
        <p>This is a red page</p>
</body>
</html>
```

Method 2: Internal (the tag style)

Another way is to include the CSS codes using the HTML tag <style>. For example like this:

```
<html>
<head>
        <title>Example</title>
        <style type="text/css">
                body {background-color: #FF0000;}
```

```
        </style>
</head>
<body>
        <p>This is a red page</p>
</body>
</html>
```

Method 3: External (link to a style sheet)

The recommended method is to link to a so-called external style sheet. An external style sheet is simply a text file with the extension **.css**. Like any other file, you can place the style sheet on your web server or hard disk.

For example, let's say that your style sheet is named **style.css** and is located in a folder named **style**. The situation can be illustrated like this:

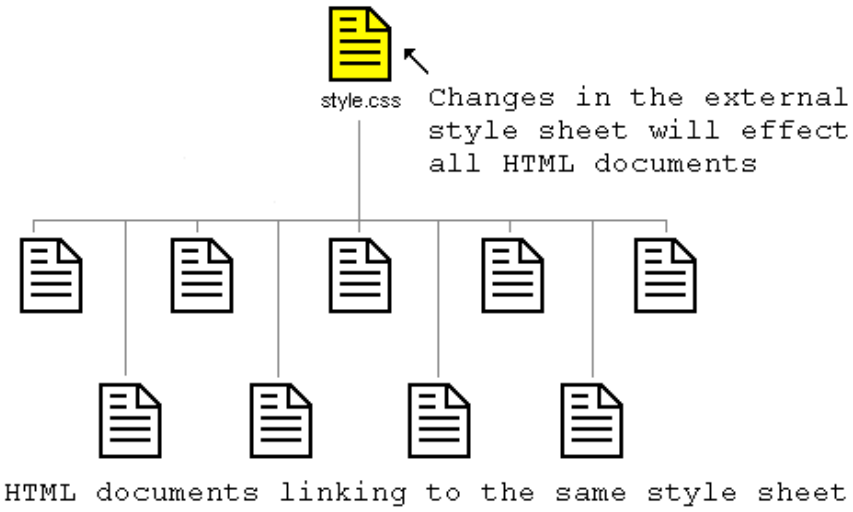
The trick is to create a link from the HTML document (default.htm) to the style sheet (style.css). Such link can be created with one line of HTML code:

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

Notice how the path to our style sheet is indicated using the attribute 'href'. The line of code must be inserted in the header section of the HTML code i.e. between the <head> and </head> tags. Like this:

```
<html>
<head>
<title>My document</title>
        <link rel="stylesheet" type="text/css" href="style/style.css" />
</head>
<body>
        … … … …
```

This link tells the browser that it should use the layout from the CSS file when displaying the HTML file. The really smart thing is that several HTML documents can be linked to the same style sheet. In other words, one CSS file can be used to control the layout of many HTML documents.



This technique can save you a lot of work. If you, for example, would like to change the background color of a website with 100 pages, a style sheet can save you from having to manually change all 100 HTML documents. Using CSS, the change can be made in a few seconds just by changing one code in the central style sheet.

CSS Classes

The class selector allows to style items within the same HTML element differently. With classes the style can be overwritten by changing out stylesheets. Same class selector can be used again and again within an HTML file.

This style will be applied to all <p> tags.

```
p {
 font-size: small;
 color: #333333
}
```

But lets say that we wanted to change some words to green bold text within the paragraph, while leaving the rest of the sentence untouched. We would do the following:

```
.greenBoldText
{
 font-size: small;
 color: #008080;
 font-weight: bold;
}
```

 This is main paragraph, it contains Green Bold Text and it still continues with main formatting.

CSS IDs

IDs are similar to classes, except once a specific id has been declared it cannot be used again within the same HTML file.

Use IDs to style the layout elements of a page that will only be needed once and use classes to style text and such that may be declared multiple times.

The main container for this page is defined by the following.

```
<div id="container">
    Everything within my document is inside this division.
</div>
```

Here, id selector is chosen for the "container" division over a class, because it will be used one time only within the file.

CSS file looks following:

```
#container
{
  width: 80%;
  margin: auto;
  padding: 20px;
  border: 1px solid #666;
  background: #ffffff;
}
```

Note: The id selector begins with a (#) sign instead of a (.)

Multiple Style Sheets

If some properties have been set for the same selector in different style sheets, the values will be inherited from the more specific style sheet.

For example, an external style sheet has these properties for the h3 selector:

```
h3
{
color:red;
text-align:left;
font-size:8pt;
}
```

And an internal style sheet has these properties for the h3 selector:

```
h3
{
text-align:right;
font-size:20pt;
}
```

If the page with the internal style sheet also links to the external style sheet the properties for h3 will be:

```
color:red;
text-align:right;
font-size:20pt;
```

The color is inherited from the external style sheet and the text-alignment and the font-size is replaced by the internal style sheet.

Mostly Used CSS$^{2.0}$ Properties

Background Properties

Property	Description	Possible Values	Examples
background-attachment	Declares the attachment of a background image (to scroll with the page content or be in a fixed position).	*fixed* *scroll*	*div { background-attachment:fixed; }* *div { background-attachment:scroll; }*

background-color	Declares the background color.	Valid color names, RGB values, hexadecimal notation.	*div { background-color:green; }* *div { color:#00FF00; }*
background-image	Declares the background image of an element.	URL values.	*div { background-image:url(images/img.jpg); }* *body { background-image:url(img.jpg); }*
background-position	Declares the position of a background image.	*top left* *top center* *top right* *center left* *center center* *center right* *bottom left* *bottom center* *bottom right*	*div { background-position:10px 50px; }* *div { background-position:bottom right; }*
background-repeat	Declares how and/or if a background image repeats.	*repeat* *repeat-x* *repeat-y* *no-repeat*	*div { background-repeat:repeat-x; }* *div { background-repeat:no-repeat; }*
background	Used as a shorthand property to set all the background properties at once.	Separate values by a space in the following order (those that are not defined will use inherited or default initial values): *background-color* *background-image* *background-repeat* *background-attachment* *background-position*	*div { background:green url(image.jpg) no-repeat fixed center center; }* *div { background:url(image.jpg) fixed; }*

Border Properties

Property	Description	Possible Values	Examples
border-top-color	Declares the color of the top border.	Valid color names, RGB values, hexidecimal notation, or the predefined value **transparent**.	*div { border-top-color:green; }* *div { border-top-color:#00FF00; }*
border-top-style	Declares the style of the top border.	*none* *hidden* *dotted* *dashed* *solid* *double* *groove* *ridge* *inset*	*div { border-top-style:solid; }* *div { border-top-style:inset; }*

		outset	
border-top-width	Declares the width of the top border.	Lengths or the following predefined values: *thin* *medium* *thick*	*div { border-top-width:2px; }* *div { border-top-width:thin; }*
border-top	Used as a shorthand property to set all the border-top properties at once.	Separate values by a space in the following order (those that are not defined will use inherited or default initial values): *border-top-width* *border-top-style* *border-top-color*	*div { border-top:2px solid green; }* *div { border-top:thick double #00FF00; }*
border-right-color	Declares the color of the right border.	Valid color names, RGB values, hexadecimal notation, or the predefined value **transparent**.	*div { border-right-color:green; }* *div { border-right-color:#00FF00; }*
border-right-style	Declares the style of the right border.	*none* *hidden* *dotted* *dashed* *solid* *double* *groove* *ridge* *inset* *outset*	*div { border-right-style:solid; }* *div { border-right-style:inset; }*
border-right-width	Declares the width of the right border.	**Lengths or the following predefined values:** *thin* *medium* *thick*	*div { border-right-width:2px; }* *div { border-right-width:thin; }*
border-right	Used as a shorthand property to set all the border-right properties at once.	Separate values by a space in the following order (those that are not defined will use inherited or default initial values): *border-right-width* *border-right-style* *border-right-color*	*div { border-right:2px solid green; }* *div { border-right:thick double #00FF00; }*
border-bottom-color	Declares the color of the bottom border.	Valid color names, RGB values, hexidecimal notation, or the predefined value **transparent**.	*div { border-bottom-color:green; }* *div { border-bottom-color:#00FF00; }*
border-bottom-style	Declares the style of	*none*	*div { border-*

	the bottom border.	*hidden* *dotted* *dashed* *solid* *double* *groove* *ridge* *inset* *outset*	*bottom-style:solid; }* *div { border-bottom-style:inset; }*
border-bottom-width	Declares the width of the bottom border.	Lengths or the following predefined values: *thin* *medium* *thick*	*div { border-bottom-width:2px; }* *div { border-bottom-width:thin; }*
border-bottom	Used as a shorthand property to set all the border-bottom properties at once.	Separate values by a space in the following order (those that are not defined will use inherited or default initial values): *border-bottom-width* *border-bottom-style* *border-bottom-color*	*div { border-bottom:2px solid green; }* *div { border-bottom:thick double #00FF00; }*
border-left-color	Declares the color of the left border.	Valid color names, RGB values, hexidecimal notation, or the predefined value **transparent**.	*div { border-left-color:green; }* *div { border-left-color:#00FF00; }*
border-left-style	Declares the style of the left border.	*none* *hidden* *dotted* *dashed* *solid* *double* *groove* *ridge* *inset* *outset*	*div { border-left-style:solid; }* *div { border-left-style:inset; }*
border-left-width	Declares the width of the left border.	Lengths or the following predefined values: *thin* *medium* *thick*	*div { border-left-width:2px; }* *div { border-left-width:thin; }*
border-left	Used as a shorthand property to set all the border-left properties at once.	Separate values by a space in the following order (those that are not defined will use inherited or default initial values): *border-left-width* *border-left-style* *border-left-color*	*div { border-left:2px solid green; }* *div { border-left:thick double #00FF00; }*

border-color	Declares the border color of all four borders at once.	Valid color names, RGB values, hexidecimal notation, or the predefined value **transparent**. Separate the color for each border by a space, declaring the colors for the borders in the following order: border-top-color border-right-color border-bottom-color border-left-color Undeclared values work as further shorthand notation. If only one color value is declared, all four borders will use that color. If two colors are declared, the top and bottom borders will use the first color while the right and left borders will use the second color. If three colors are declared, the top border will use the first color, the right and left borders will use the second color, and the bottom border will use the third color.	*div { border-color:green red blue olive; }* *div { border-color:green; }* *div { border-color:green red; }* *div { border-color:green red blue; }*
border-style	Declares the border style of all four borders at once.	*none* *hidden* *dotted* *dashed* *solid* *double* *groove* *ridge* *inset* *outset*	*div { border-style:solid dotted dashed double; }* *div { border-style:solid; }* *div { border-style:solid dotted; }* *div { border-style:solid dotted dashed; }*
border-width	Declares the width of all four borders at once.	Lengths or the following predefined values: *thin* *medium* *thick*	*div { border-width:1px 3px 5px 2px; }* *div { border-width:thin; }*
border	Used as a shorthand to declare the border properties when all four borders will have the same appearance.	Separate values by a space in the following order (those that are not defined will use inherited or default initial values): *border-width* *border-style* *border-color*	*div { border:1px double green; }* *div { border:thin solid #00FF00; }*

Classification and Positioning Properties

Property	Description	Possible Values	Examples
clear	Declares the side(s) of an element where no previous	*left*	*div { clear:right; }*

	floating elements are allowed to be adjacent.	*right* *both* *none*	*div { clear:both; }*
cursor	Declares the type of cursor to be displayed.	URL values, and the following prefefined values: *auto* *crosshair* *default* *pointer* *move* *e-resize* *ne-resize* *nw-resize* *n-resize* *se-resize* *sw-resize* *s-resize* *w-resize* *text* *wait* *help*	*div { cursor:crosshair; }* *div { cusrsor:url(image.csr); }* *div { cusrsor:url(image.csr), pointer; }*
display	Declares if/how the element displays.	*none* *inline* *block* *list-item* *run-in* *compact* *marker* *table* *inline-table* *table-row-group* *table-header-group* *table-footer-group* *table-row* *table-column-group* *table-column* *table-cell* *table-caption*	*div { display:none; }* *div { display:inline; }* *div { display:marker; }*
float	Declares whether a box should float to the left or right of other content, or whether it should not be floated at all.	*left* *right* *none*	*div { float:left; }* *div { float:right; }*
visibility	Declares the visibility of boxes generated by an element.	visible hidden collapse	*div { visibility:visible; }* *div { visibility:hidden; }*
top	Declares the distance that the top content edge of the element is offset below the top edge of its containing block. The **position** property of the element must also be set to a value other than **static**.	Lengths, percentages, and the predefined value **auto**.	*div { top:15px; }* *div { top:2%; }*

right	Declares the distance that the right content edge of the element is offset to the left of the right edge of its containing block. The **position** property of the element must also be set to a value other than **static**.	Lengths, percentages, and the predefined value **auto**.	*div { right:15px; }* *div { right:2%; }*
bottom	Declares the distance that the bottom content edge of the element is offset above the bottom edge of its containing block. The **position** property of the element must also be set to a value other than **static**.	Lengths, percentages, and the predefined value **auto**.	*div { bottom:15px; }* *div { bottom:2%; }*
left	Declares the distance that the left content edge of the element is offset to the right of the left edge of its containing block. The **position** property of the element must also be set to a value other than **static**.	Lengths, percentages, and the predefined value **auto**.	*div { left:15px; }* *div { left:2%; }*
position	Declares the type of positioning of an element.	*static* *relative* *absolute* *fixed*	*div { position:absolute; }* *div { position:relative; }*
clip	Declares the shape of a clipped region when the value of the **overflow** property is set to a value other than **visible**.	Shapes, or the predefined value **auto**. *rect(top, right, bottom, left)*	*div { clip:auto; }* *div { clip:rect(2px, 4px, 7px, 5px); }*
overflow	Declares how content that overflows the element's box is handled.	*visible* *hidden* *scroll* *auto*	*div { overflow:hidden; }* *div { overflow:scroll; }*
vertical-align	Declares the vertical alignment of an inline-level element or a table cell.	Lengths, percentages, and the following predefined values: *baseline* *sub* *super* *top* *text-top* *middle* *bottom* *text-bottom*	*span { vertical-align:middle; }* *td { vertical-align:top; }*
z-index	Declares the stack order of the element.	Integer values and the predefined value **auto**.	*div { z-index:2; }* *div { z-index:auto; }*

Dimension Properties

Property	Description	Possible Values	Examples
height	Declares the height of the element.	Lengths, percentages, and the predefined value **auto**.	*div { height:200px; }* *div { height:50%; }*

max-height	Declares the maximum height of the element.	Lengths, percentages, and the predefined value **auto**.	*div { max-height:200px; }* *div { max-height:50%; }*
min-height	Declares the minimum height of the element.	Lengths, percentages, and the predefined value **auto**.	*div { min-height:200px; }* *div { min-height:50%; }*
width	Declares the width of the element.	Lengths, percentages, and the predefined value **auto**.	*div { width:500px; }* *div { width:75%; }*
max-width	Declares the maximum width of the element.	Lengths, percentages, and the predefined value **auto**.	*div { max-width:500px; }* *div { max-width:75%; }*
min-width	Declares the minimum width of the element.	Lengths, percentages, and the predefined value **auto**.	*div { min-width:500px; }* *div { min-width:75%; }*

Font Properties

Property	Description	Possible Values	Examples
font-family	Declares the name of the font to be used. Previously set in HTML via the *face* attribute in a <font> tag.	Valid font family names or generic family names, i.e. *Arial*, *Verdana*, *sans-serif*,*"Times New Roman"*, *Times*, *serif*, etc. Font family names can be separated by a comma in the same declaration to allow additional and/or generic family names to be used if the preferred font is unable to be displayed.	*div { font-family:Arial; }* *div { font-family:Arial, Helvetica, sans-serif; }*
font-size	Declares the size of the font. Previously set in HTML via the *size* attribute in a <font> tag.	Lengths (number and unit type— i.e. *1em*,*12pt*, *10px*, *80%*) or one of the following predefined values: *xx-small* *x-small* *small* *medium* *large* *x-large* *xx-large* *smaller* *larger*	*div { font-size:70%; }* *div { font-size:0.85em; }* *div { font-size:medium; }*
font-style	Declares the font style.	*normal* *italic* *oblique*	*div { font-style:italic; }*

			div { font-style:oblique; }
font-variant	Declares the font variant.	*normal* *small-caps*	*div { font-variant:normal; }* *div { font-variant:small-caps; }*
font-weight	Declares the font weight (lightness or boldness)	*normal* *bold* *bolder* *lighter* *100* *200* *300* *400* *500* *600* *700* *800* *900*	*div { font-weight:bolder; }* *div { font-weight:200; }*
font	Used as a shorthand property to declare all of the font properties at once (except font-size-adjust and font-stretch).	Separate values by a space in the following order (those that are not defined will use inherited or default initial values): *font-style* *font-variant* *font-weight* *font-size* *line-height* *font-family*	*div { font:italic small-caps bold 1em 1.2em Arial }* *div { font:bold 0.8em Verdana }*

List Properties

Property	Description	Possible Values	Examples
list-style-type	Declares the type of list marker used.	*disc* *circle* *square* *decimal* *decimal-leading-zero* *lower-roman* *upper-roman* *lower-alpha* *upper-alpha* *lower-greek* *lower-latin* *upper-latin*	*ol { list-style-type:upper-roman; }* *ul { list-style-type:square; }*
list-style-position	Declares the position of the list marker.	*inside* *outside*	*ol { list-style-position:inside; }* *ul { list-style-*

			position:outside; }
list-style-image	Declares an image to be used as the list marker.	URL values.	*ul { list-style-image:url(image.jpg); }*
list-style	Shorthand property to declare three list properties at once.	Separate values by a space in the following order (those that are not defined will use inherited or default initial values): *list-style-type* *list-style-position* *list-style-image*	*ul { list-style:disc inside url(image.gif); }* *ol { list-style:upper-roman outside; }*
marker-offset	Declares the marker offset for elements with a value of**marker** set for the **display**property.	Lengths and the predefined value**auto**.	*li:before { display:marker; marker-offset:5px; }*

Margin Properties

Property	Description	Possible Values	Examples
margin-top	Declares the top margin for the element.	Lengths, percentages, and the predefined value **auto**.	*div { margin-top:5px; }* *div { margin-top:15%; }*
margin-right	Declares the right margin for the element.	Lengths, percentages, and the predefined value **auto**.	*div { margin-right:5px; }* *div { margin-right:15%; }*
margin-bottom	Declares the bottom margin for the element.	Lengths, percentages, and the predefined value **auto**.	*div { margin-bottom:5px; }* *div { margin-bottom:15%; }*
margin-left	Declares the left margin for the element.	Lengths, percentages, and the predefined value **auto**.	*div { margin-left:5px; }* *div { margin-left:15%; }*
margin	Shorthand property used to declare all the margin properties at once.	Separate values by a space in the following order (those that are not defined will use inherited or default initial values): *margin-top* *margin-right* *margin-bottom* *margin-left*	*div { margin:5px 12px 4px 7px; }* *div { margin:5px; }* *div { margin:5px 10px; }* *div { margin:5px 7px 4px; }*

Padding Properties

Property	Description	Possible Values	Examples
padding-top	Declares the top padding for the element.	Lengths, percentages, and the predefined value **auto**.	*div { padding-top:5px; }* *div { padding-top:15%; }*

padding-right	Declares the right padding for the element.	Lengths, percentages, and the predefined value **auto**.	*div { padding-right:5px; }* *div { padding-right:15%; }*
padding-bottom	Declares the bottom padding for the element.	Lengths, percentages, and the predefined value **auto**.	*div { padding-bottom:5px; }* *div { padding-bottom:15%; }*
padding-left	Declares the left padding for the element.	Lengths, percentages, and the predefined value **auto**.	*div { padding-left:5px; }* *div { padding-left:15%; }*
padding	Shorthand property used to declare all the margin properties at once.	Separate values by a space in the following order (those that are not defined will use inherited or default initial values): *padding-top* *padding-right* *padding-bottom* *padding-left*	*div { padding:5px 12px 4px 7px; }* *div { padding:5px; }* *div { padding:5px 10px; }* *div { padding:5px 7px 4px; }*

Table Properties

Property	Description	Possible Values	Examples
border-collapse	Declares the way borders are displayed.	*collapse* *separate*	*table { border-collapse:collapse; }* *table { border-collapse:separate; }*
border-spacing	Declares the distance separating borders (if **border-collapse** is**separate**).	Lengths for the horizontal and vertical spacing, separated by a space. If one length is value is declared, that length is used for both the horizontal and vertical spacing. If two lengths are declared, the first one is used for horizontal spacing and the second one is used for vertical spacing.	*table { border-spacing:5px; }* *table { border-spacing:5px 10px; }*
caption-side	Declares where the table caption is displayed in relation to the table.	*top* *bottom* *left* *right*	*caption { caption-side:top; }* *caption { caption-side:right; }*
empty-cells	Declares the way empty cells are displayed (if**border-collapse**is **separate**).	*show* *hide*	*table { empty-cells:show; }* *table { empty-cells:hide; }*
table-layout	Declares the type of table layout.	*auto* *fixed*	*table { table-layout:auto; }* *table { table-layout:fixed; }*

Text Properties

Property	Description	Possible Values	Examples
color	Declares the color of the text.	Valid color names, RGB values, hexidecimal notation. *aqua* *black* *blue* *fuchsia* *gray* *green* *lime* *maroon* *navy* *olive* *purple* *red*	*div { color:green; }* *div {color:rgb(0,255,0); }* *div { color:#00FF00; }*

		silver *teal* *white* *yellow*	
direction	Declares the reading direction of the text.	ltr = left-to-right rtl = right-to-left	*div { direction:ltr; }* *div { direction:rtl; }*
line-height	Declares the distance between lines.	Numbers, percentages, lengths, and the predefined value of *normal*.	*div { line-height:normal; }* *div { line-height:2em; }* *div { line-height:125%; }*
letter-spacing	Declares the amount of space between text characters.	A length (in addition to the default space) or the predefined value of*normal*.	*div { letter-spacing:normal; }* *div { letter-spacing:5px; }* *div { letter-spacing:-1px; }*
text-align	Declares the horizontal alignment of inline content.	*left* *right* *center* *justify*	*div { text-align:center; }* *div { text-align:right; }*
text-decoration	Declares the text decoration.	*none* *underline* *overline* *line-through* *blink*	*div { text-decoration:none; }* *div { text-decoration:underline; }*
text-indent	Declares the indentation of the first line of text.	Lengths and percentages.	*div { text-indent:12px; }* *div { text-indent:2%; }*
text-shadow	Declares shadow effects on the text.	A list containg a color followed by numeric values (separated by spaces) that specify: 1. The color for the shadow effect 2. Horizontal distance to the right of the text 3. Vertical distance below the text 4. Blur radius	*div { text-shadow:green 2px 2px 7px; }* *div { text-shadow:olive -3px -4px 5px; }*
text-transform	Declares the capitalization effects on the letters in the text.	*none* *capitalize* *uppercase* *lowercase*	*div { text-transform:uppercase; }* *div { text-transform:lowercase; }*
word-spacing	Declares the space between words in the text.	A length (in addition to the default space) or the predefined value of*normal*.	*div { word-spacing:normal; }* *div { word-spacing:1.5em; }*

KNOW the TERMS

DHTML

Dynamic HTML is not really a new specification of HTML, but rather a new way of looking at and controlling the standard HTML codes and commands. When thinking of dynamic HTML, you need to remember the qualities of standard HTML, especially that once a page is loaded from the server, it will not change until another request comes to the server. Dynamic HTML gives you more control over the HTML elements and allows them to change at any time, without returning to the Web server.

There are four parts to DHTML:

- Document Object Model (DOM)
- Scripts
- Cascading Style Sheets (CSS)
- HTML

DOM

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

Scripts

Scripts written in either JavaScript or VBScript.

Cascading Style Sheets (CSS)

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

HTML

HTML 4.x is used to create the page itself and build the elements for the CSS and the DOM to work on. There is nothing special about HTML for DHTML - but having valid HTML is even more important.

These are most common features of DHTML:

- **Changing the tags and properties**
    - This is one of the most common uses of DHTML. It allows changing the qualities of an HTML tag depending on an event outside of the browser (such as a mouse click, time, or date, and so on). With this information can be preloaded but not displayed until the reader clicks on a specific link.
- **Real-time positioning**
    - When most people think of DHTML this is what they expect. Objects, images, and text moving around the Web page. This can allow you to play interactive games with your readers or animate portions of your screen.

Practical Exercises:

Type the following codes in a notepad and save it as a html page to see the output. Here in this example we have used tables to divide the content of web page into different sections.

Question 1:

```
<html>
<head>
<title>WEB PAGE TITLE GOES HERE</title>
</head>

<body style="margin: 0px; padding: 0px; font-family: 'Trebuchet MS',verdana;">
<table width="100%" style="height: 100%;" cellpadding="10" cellspacing="0"
border="0"><tr>
<!-- ============ LEFT COLUMN (MENU) ============== -->
<td width="20%" valign="top" bgcolor="#999f8e">
<a href="#">Menu link</a><br>
<a href="#">Menu link</a><br>
<a href="#">Menu link</a><br>
<a href="#">Menu link</a><br>
<a href="#">Menu link</a>
</td>
<!-- ============ RIGHT COLUMN (CONTENT) ============== -->
<td width="80%" valign="top" bgcolor="#d2d8c7">

<h1>Website Logo</h1>
<h2>Page heading</h2>
This is a basic two-column web page layout. The left column or the <i>menu
column</i> is a narrow band of space (usually between 15-25% of the page width)
and is reserved for a menu of hyperlinks leading to other pages on your
website. The table used to create this layout employs a single table row
containing two table cells.<br>
<br>
The right column or the <i>content column</i> takes up the lion's share of the
web page width and contains the actual content of each particular page. In a
basic two column layout like this, it is common to place the website logo at
the top of the content column on each page.</td></tr></table>
</body>
</html>
```

Question 2:

```
<html>
<head>
<title>WEB PAGE TITLE GOES HERE</title>
</head>
<body style="margin: 0px; padding: 0px; font-family: 'Trebuchet MS',verdana;">
```

```
<table width="100%" style="height: 100%;" cellpadding="10" cellspacing="0"
border="0">
<tr>
<!-- ============ HEADER SECTION ============== -->
<td colspan="2" style="height: 100px;" bgcolor="#777d6a"><h1>Website
Logo</h1></td></tr>

<tr>
<!-- ============ LEFT COLUMN (MENU) ============== -->
<td width="20%" valign="top" bgcolor="#999f8e">
<a href="#">Menu link</a><br>
<a href="#">Menu link</a><br>
<a href="#">Menu link</a><br>
<a href="#">Menu link</a><br>
<a href="#">Menu link</a>
</td>

<!-- ============ RIGHT COLUMN (CONTENT) ============== -->
<td width="80%" valign="top" bgcolor="#d2d8c7">
<h2>Page heading</h2>

Here's a two column layout with a header section that spans the width of both
columns. The first table row creates the header and contains a single table
cell which uses the colspan="2" attribute-value pair. The website logo
typically goes in the header section.<br>
<br>
The second table row contains two table cells which create the menu column
(left) and the content column (right). The colspan attribute is not set in
either so they default to colspan="1".</td></tr></table>
</body>
</html>
```

Question 3:

```
<html>
<head>
<title>WEB PAGE TITLE GOES HERE</title>
</head>
<body style="margin: 0px; padding: 0px; font-family: 'Trebuchet MS',verdana;">
<table width="100%" style="height: 100%;" cellpadding="10" cellspacing="0"
border="0">
<tr>
<!-- ============ HEADER SECTION ============== -->
<td colspan="2" style="height: 100px;" bgcolor="#777d6a"><h1>Website
Logo</h1></td></tr>

<tr>
<!-- ============ LEFT COLUMN (MENU) ============== -->
```

```
<td width="20%" valign="top" bgcolor="#999f8e">
<a href="#">Menu link</a><br>
<a href="#">Menu link</a><br>
<a href="#">Menu link</a><br>
<a href="#">Menu link</a><br>
<a href="#">Menu link</a>
</td>

<!-- ============ RIGHT COLUMN (CONTENT) ============== -->
<td width="80%" valign="top" bgcolor="#d2d8c7">
<h2>Page heading</h2>

Here's a two column layout with header and footer sections that span the width
of both columns. The first table row creates the header and contains a single
table cell which uses the colspan="2" attribute-value pair.<br>
<br>
The second table row contains two table cells which create the menu column
(left) and the content column (right). The colspan attribute is not set in
either so they default to colspan="1".<br>
<br>
The third table row creates the footer. Like the header, it contains a single
table cell which uses the colspan="2" attribute-value pair.</td></tr>

<!-- ============ FOOTER SECTION ============== -->
<tr><td colspan="2" align="center" height="20" bgcolor="#777d6a">Copyright
©</td></tr>
</table>
</body>
</html>
```

** Follow the book 'HTML/DHTML/Javascript/PHP' by Ivan Bayross along with this handout. **

JAVASCRIPT

Introduction

- JavaScript is a compact, object-based scripting language for developing client Internet applications.
- JavaScript was designed to add interactivity to HTML pages.
- JavaScript is a scripting language - a scripting language is a lightweight programming language.
- A JavaScript is lines of executable computer code.
- A JavaScript is usually embedded directly in HTML pages
- JavaScript is an interpreted language (means that scripts execute without preliminary compilation).
- Everyone can use JavaScript without purchasing a license.
- All major browsers, like Netscape and Internet Explorer, support JavaScript.
- Javascript was developed by Netscape as *Live Script* - changed to *JavaScript* when endorsed by Sun 1993, version 1.0 released with Netscape 2.0.
- JavaScript is a powerful scripting language that is also capable of performing extensive form data collecting and form processing functions.

Comparing Javascript with Java

- *While comparing javascript with java first lets see the basic differences between them*

Javascript	Java
1. Javascript is a small, lightweight programming language.	1. Java is a full-blown powerful, sophisticated programming language.
2. Developed by Netscape communications.	2. Developed by Sun Microsystems.
3. Scripting language interpreted at runtime.	3. True programming compiled to byte-code.
4. Object-based, has limited number of built-in objects.	4. Object-oriented, can create their own classes.
5. Not fully extensible.	5. Fully extensible.
6. Code integrated with, and embedded in HTML.	6. Applies distinct from HTML (accessed from HTML

	pages).
7. Variables type must not be declared.	7. Variables type must be declared.
8. Javascript source code can be viewed by everybody using "view source command".	8. Your Java source is hidden because it's only the compiled byte-code, which the browser uses, but this is not a *guarantee* of security.

- *Now lets see the similarities between java and javascript:*
    1. Both can be used for enhancing the capabilities of the web pages.
    2. Both can run on the client machine - i.e. the machine where you have your browser, not the server where the page came from.
    3. Both have some level of security built in to guard against malicious use. No computer system is ever 100% secure unless it's isolated in a locked room surrounded by armed guards, but the developers of Java and JavaScript have taken some care in their security.

Netscape & Java Script:

Java script is scripting language created by Netscape hence JavaScript works best with the Netscape suite of client and server products. The Netscape client 'browser' is called Netscape communicator. The default scripting language that Netscape communicator understands is java Script. Netscape server product is called Netscape commerce server. The default scripting language that Netscape commerce server understands is JavaScript.

Database Connectivity:

Netscape has a product called 'Live wire', which permits server side, JavaScript code, to connect to Relational Database management systems (RDBMS) like oracle, My SQL server, My SQL etc. 'Live wire' database drivers also support a no of non-relational database.

JavaScript Uses

- The most frequent uses of javascript are:
    - For displaying the clock
    - Used for creating Drop-down menus.
    - Showing Alert messages.

- Displaying Popup windows.
- Validating HTML Form Data.

Advantages of JavaScript

- Cross Browser supporting:
  - This means that the javascript codes are supported by various browsers like Internet Explorer, Netscape Navigator, and Mozilla etc.
- Platform Independent:
  - Javascript codes can be executed in any platform like Linux, Microsoft Windows and many other operating systems.
- Lightweight for fast downloading:
  - The javascript codes runs directly in the client machine. The code should be downloaded all the way from server to the client and this time duration is very minimum and the executing the codes of javascript is also very fast.
- Validating Data:
  - The data can be validated in the two different way:
    - Validating in server side or in server machine.
    - Validating in client side or in client machine.
  - In this two different types of validation of data the second one is much more faster, and this is done through javascript.
- Sophisticated User Interfaces:
  - By using javascript you can create a user interactive interfaces that can communicate with the user and the Browser.
- In-Built software:
  - To you don't need any extra tools to write JavaScript, any plain text or HTML editor will do, so there's no expensive development software to buy.
- Easy to Learn:
  - The javascript programmer should know the minimal syntax of javascript since it supports many syntax and data types as C and C++.
  - It's also an easy language to learn, and there's a thriving and supportive online community of JavaScript developers and information resources.

- Designed for programming User-Events:
  - Javascript supports Object/Event based programming. So the code written in javascript can easily be break down into sub-modules.

Disadvantages of JavaScript:

- Launching an application on the client computer.
  - Javascript is not used to create stand-alone application; it is only used to add some functionality in any web page.
- Reading or writing files:
  - Javascript cannot read and write files into the client machines. It can only be used as a utility language to develop any web site.
- Retrieving the text contents of HTML pages or files from the server.
- Reading and Writing files to the server:
  - Javascript can read and write to any file in the server as well.
- Sending secret e-mails from Web site visitors to you:
  - Javascript cannot be used to send email to the visitors or user of the web site. This can be done only with the server side scripting.
- Cannot create Database Application:
  - By using javascript you cannot connect the web site to the database. For this you need to use server-side scripting.
- Browser Compatibility Issues:
  - Not all browsers support the javascript codes. The browser may support javascript as a whole, but may not support the codes or lines of codes written in javascript and may interpret differently.
- Javascript does not implement multiprocessing or multithreading.
- Use printers or other devices on the user's system or the client-side LAN.
- Javascript has limitations of writing in a client machine. It can only write the cookie in client machine that is also of a certain size i.e. 4K.

ASSIGNMENT 04

JavaScript Myths

Compile on a DOC file and send mail to *saroj@sarojpandey.com.np*

Adding Javascript codes

- You have to focus on three major steps while adding javascript codes in HTML document.
  1. Use a **<script>** tag to tell the browser that you are using javascript.

     Example:

     <script language = ”Javascript”**>**

  2. Write the javascript codes.

     Example:

     <script language = ”Javascript”**>**

     // javascript codes HERE

     </script>
  3. Test the scripts.
- While writing the javascript codes there can be many types of errors like:
  - Human Errors.
  - Browser compatibility issues.
  - Incorrect behavior based on the different operating systems.
- So when using javascript, be sure that you test your scripts out on a wide variety of systems and setups.
- You can add javascript codes between the **<head>** tag as well as in the **<body>** tag. And you can have multiple **<script>** tag in a single web page, but you must have the closing **</script>** tag of each opening tag.

First Example in Javascript

<Script language = “javascript”>

<!- -

```
            document.write ("Hello World");
        // - - >
    </script>
```

- The first line <script language = "javascript"> tells the browser that the code written between the <script> and </script> tags are the javascript codes.
- Some browsers doesn't support scripts, and may display the scripts as the HTML content in the web-page.
- To prevent this you must write the javascript codes in between <! - - and // - ->.
- The two forward slashes at the end of the line are the javascript comments.
- You cannot put this slashes to the starting of the comment line like (// <!- -), because the older browser will display it.
- Now the above code will display **Hello World** in the browser.

Using External JavaScript Files

- While using external javascript files in a web-page, the javascript files must have the three main features:
  - First, the file that you are importing must be a valid javascript file.
  - Second, the file must have the extension "**.js**".
  - Lastly, you must know the location of the file.

- Here is the example of using external javascript files in a web-page.

```
<HTML>
        <HEAD>
            <Script src = "myExample.js"></script>
            <TILTE></TITLE>
        </HEAD>
            <BODY>
            </BODY>
<HTML>
```

- Here myExample.js contains the javascript codes and whatever is written in the file will be displayed in the browser.

- In this condition the myExample.js file and the above HTML file must be in the same location.
- For implementing the external javascript file you must have at least two files: one for HTML codes and other for javascript file.

Points to Ponder

- Optional semicolon (;) to end the statement.
- Some browser doesn't support javascript codes, and display the script as the content of the web-page.
- To prevent this we have to use the HTML comments.
- To use JavaScript functions the code needs to follow the line:

  <SCRIPT language = "JavaScript"> and closed with **</SCRIPT>.**
- JavaScript uses both // and /* */ to indicate comments.
- JavaScript supports data types: numeric, string, Boolean, and null.
- JavaScript contains three user interfaces: **alert** (message), **prompt** (message, [Default]), **confirm**(message).
- JavaScript does not have built-in support for arrays, but does allow for building of arrays.
- JavaScript supports operators: +, -, *, /, %, ++, --, =, +=, -=, *=, /=, %=, &&, ||, !, &, |, ^.
- JavaScript is case sensitive.
- It's a Object-based language.

Case Sensitivity

- Javascript is a case-sensitive language.
- This means the Upper and Lower case of any variables or methods may effect.
- The variable in Uppercase is different then the variable declared in Lowercase.
- For Example:

  myVar, myvar, MyVar and MYVAR are four different types of variables.
- The same rules occur in the name of function as well as in objects of javascript.

Optional Semicolons

- The semicolon (;) identifies the end of the statement in javascript.
- But it is not always necessary to put semicolon at the end of statement.

- If you are writing two different statements in two different lines without semicolon than it is valid in javascript.
- But if you are writing the two different statements in same single line then you have to keep semicolon after the end of the first statement.
- Examples:

  var a=5

  var b=2

  The above statement are valid in javascript since javascript has optional semicolon to end the statement.

  var a=5; var b=5

  This is also valid in javascript

  var a=5;

  var b=2;

  These two statements are also valid in javascript.

Reserved Words

- JavaScript have some reserved words that cannot be used as variable, objects or a function.
- These words are used for some reserved purpose in JavaScript.
- ***'Some' reserved words:***

abstract	boolean	break
case	catch	char
const	continue	debugger
delete	do	double
false	final	finally

…and Many More…

- **Objects:**
  - An *object* is a collection of named values, called the *properties* of that object.
  - Functions associated with an object are referred to as the *methods* of that object.

- Properties and methods of objects are referred to with a dot.notation that starts with the name of the object and ends with the name of the property.
- Objects in JavaScript can be treated as associative arrays.
- JavaScript has many predefined objects, such as a **String** Object, **Date** object and a **Math** object. ***[Refer TextBook.]***

- **Functions***:*
  - A *function* is a piece of code, predefined or written by the person creating the JavaScript, that is executed based on a call to it by name.
  - Any JavaScript that is not inside a function (or is not associated with some even attribute or hyperlink attribute) is executed the moment the Web browser reaches it when first parsing the document.
  - They also allow for the same piece of code to be re-used many times in the same document, since functions allow the section of code they contain to be referred to by name.
  - A function is a data type in JavaScript.
  - This means that they can be treated as containing values that can be changed.

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

- Therefore, the third element (position 2) of or inside the ninth element (position 8) would be arrayName[8][2].

Variable - Introduction

- Variable is a name that can be used to store values.
- It is a name assigned to a location in computer's memory to store data.
- These variables can take different values but one at a time and the value can be changed during the execution of program.
- Variable names may consist of uppercase character, lowercase character and underscore.

- *Rules to giving the name of variable are as:*
  1. First character should be a letter or underscore.
  2. The variable name cannot be a keyword.
  3. Uppercase and lowercase letters are significant for example code, Code, CODE are three different variables. But variables are in lowercase.
- **Note:** The Netscape supports the ($) sign as the first character in variable but the Internet Explorer does not supports.

Typing

- Javascript is a untyped language.
- This means you can use variables directly where you want to use it.
- But in javascript the variables are declared with the **var** keyword.
- This keyword declares all types of variables and you can use a same variable as string, as Integer, as Number and any type of Objects.
- For Example:

```
var a="Mr. ABC";
a=12345;
```

- The both statements are valid in javascript, but in C and C++ the statements are not valid.

- The both statements are valid due to javascript is untyped language.

Local and Global Variables

- Javascript supports both local as well as global variables.

  Local Variables:

  - The variables which are defined within a body of the function or block is local to that function or block only is called local variable.
  - They have no presence outside the function.
  - The values of such Local variables cannot be changed by the main code or other functions.
  - Example:

```
<script language="javascript">
    function testLocal()
    {
        var a =5;
        alert("The local value of a is: " + a);
    }
</script>
```

  Global Variables:

  - The variables that are declared outside the function and is used inside the function is called global variables.
  - The global variable has the same data type and same name throughout the program.
  - It is useful to declare the variable global when the variable has constant value throughout the program.
  - These are the variables that can be used throughout the scripts and the value of which can be changed.
  - ***Example:***

```
<script language="javascript">
    Var a=10;
    function testGlobal_1()
    {
```

```
            alert("The local value of a is: " + a);
            //displays the value of a as 10
        }
        function testGlobal_2()
        {
            alert("The Global value of a is:" + a);
            //displays the value of a as 10
        }
    </script>
```

- Here the value of a is global and is accessed within both functions testGlobal_1 and testGlobal_2.

What is variable scoping?

- *Local variables* exist only inside a particular function hence they have ***Local Scope***.
- *Global variables* on the other hand are present throughout the script and their values can be accessed by any function. Thus, they have ***Global Scope***.

Constants

- You can create a read-only, named constant with the const keyword.
- The syntax of a constant identifier is the same as for a variable identifier: it must start with a letter or underscore and can contain alphabetic, numeric, or underscore characters.

- *Example:*

```
const prefix = '212';
```

- A constant cannot change value through assignment or be re-declared while the script is running.
- The scope rules for constants are the same as those for variables, except that the **const** keyword is always required, even for global constants.

- If the keyword is omitted, the identifier is assumed to represent a **var**.

- You cannot declare a constant at the same scope as a function or variable with the same name as the function or variable.

- *For example:*

```
//THIS WILL CAUSE AN ERROR
    function f{};
    const f = 5;

//THIS WILL ALSO CAUSE AN ERROR
    function f
    {
    const g=5;
    var g;
    }
```

Garbage Collection

- is a way of reclaiming the memory occupied by objects that are no longer in use.
- In C and C++, garbage collection is manual--the programmer explicitly decides when to free memory for reuse.
- In Java, on the other hand, garbage collection is handled automatically--the system can detect when objects are no longer in use and free them appropriately.
- JavaScript also supports automatic garbage collection.
- In Internet Explorer 3.0, garbage collection is implemented in a technically sound way and you don't have to understand any of its details.
- It is enough to know that when your objects are no longer in use, the memory they occupy will automatically be reclaimed by the system.
- Navigator 4.0 will also have a perfectly transparent garbage collection scheme like this.
- Unfortunately, garbage collection in earlier versions of Navigator is less than perfect.
- In Navigator 3.0, it is pretty good, but requires you to be aware of a couple of issues.
- In Navigator 2.0, garbage collection is seriously flawed, and you must take a number of steps to avoid crashing the browser.

[***Note:*** Self Study: If you want to know more about Garbage Collection.]

Operators

- Javascript has different types of operator, which can be classified by several criteria, such as:

1. **Number of Operands**

   - In this type of category, javascript supports 3 basic types of Operators:

     - ***Unary***
       - Converts a single expression into a single more complex expression.
       - A unary operator requires a single operand, either before or after the operator.
       - For Example:

         ```
         i--;
         ++i;
         ```

     - ***Binary***
       - Combine two expressions into a single, more complex expression.
       - A binary operator requires two operands, one before the operator and one after the operator.
       - For Example:

         ```
         var x = 13 + 14
         ```

     - ***Ternary***
       - Also known as conditional operator.
       - The conditional operator is the only JavaScript operator that takes three operands.
       - The operator can have one of two values based on a condition.
       - *The syntax is:*

         ```
         condition ? val_1 : val_2
         ```

       - If condition is true, the operator has the value of val_1. Otherwise it has the value of val_2.
       - You can use the conditional operator anywhere you would use a standard operator.
       - *For example:*

         ```
         status = (age >= 18) ? "adult" : "minor"
         ```

       - This statement assigns the value "adult" to the variable status if age is 18 or more.
       - Otherwise, it assigns the value "minor" to status.

2. **Number of Operation.**

   - Javascript supports many types of operators according to this classification.

- They can be categorized as:

➢ ***Comparison Operators***

- A comparison operator compares its operands and returns a logical value based on whether the comparison is true.
- The operands can be numerical, string, logical, or object values.
- Strings are compared based on standard lexicographical ordering, using Unicode values.
- The following table describes the comparison operators.

Operator	Description	Example
Equal (==)	Returns true if the operands are equal. If the two operands are not of the same type, JavaScript attempts to convert the operands to an appropriate type for the comparison.	3 == var1 "3" == var1 3 == '3'
Not equal (!=)	Returns true if the operands are not equal. If the two operands are not of the same type, JavaScript attempts to convert the operands to an appropriate type for the comparison.	var1 != 4 var2 != "3"
Strict equal (===)	Returns true if the operands are equal and of the same type.	3 === var1
Strict not equal (!==)	Returns true if the operands are not equal and/or not of the same type.	var1 !== "3" 3 !== '3'
Greater than (>)	Returns true if the left operand is greater than the right operand.	var2 > var1
Greater than or equal (>=)	Returns true if the left operand is greater than or equal to the right operand.	var2 >= var1 var1 >= 3
Less than (<)	Returns true if the left operand is less than the right operand.	var1 < var2
Less than or equal (<=)	Returns true if the left operand is less than or equal to the right operand.	var1 <= var2 var2 <= 5

- ***Arithmetic Operators***
  - Arithmetic operators take numerical values as their operands and return a single numerical value.
  - The standard arithmetic operators are addition(+), subtraction(-), multiplication(*), and division(/).
  - These operators work as they do in most other programming languages, except the / operator returns a floating-point division in JavaScript.
  - In addition, JavaScript provides the arithmetic operators listed in the following table.

Operator	Description	Example
%(Modulus)	Binary operator. Returns the integer remainder of dividing the two operands.	12 % 5 returns 2.
++(Increment)	Unary operator. Adds one to its operand. If used as a prefix operator (++x), returns the value of its operand after adding one; if used as a postfix operator (x++), returns the value of its operand before adding one.	If x is 3, then ++x sets x to 4 and returns 4, whereas x++ sets x to 4 and returns 3.
--(Decrement)	Unary operator. Subtracts one to its operand. The return value is analogous to that for the increment operator.	If x is 3, then --x sets x to 2 and returns 2, whereas x++ sets x to 2 and returns 3.
-(Unary negation)	Unary operator. Returns the negation of its operand.	If x is 3, then -x returns -3.

- ***Assignment Operators***
  - An assignment operator assigns a value to its left operand based on the value of its right operand.
  - The basic assignment operator is equal (**=**), which assigns the value of its right operand to its left operand.
  - That is, x = y assigns the value of y to x.
  - The other assignment operators are shorthand for standard operations, as shown in the following table.

Shorthand operator	Meaning
x += y	x = x + y
x -= y	x = x – y
x *= y	x = x * y
x /= y	x = x / y
x %= y	x = x % y
x <<= y	x = x << y
x >>= y	x = x >> y
x >>>= y	x = x >>> y
x &= y	x = x & y
x ^= y	x = x ^ y
x |= y	x = x | y

- ***Logical Operators***
  - Logical operators are typically used with Boolean (logical) values; when they are, they return a Boolean value.
  - However, the && and || operators actually return the value of one of the specified operands, so if these operators are used with non-Boolean values, they may return a non-Boolean value.
  - The logical operators are described in the following table.

Operator	Usage	Description
&&	expr1 && expr2	(Logical AND) Returns expr1 if it can be converted to false; otherwise, returns expr2. Thus, when used with Boolean values, && returns true if both operands are true; otherwise, returns false.
||	expr1 || expr2	(Logical OR) Returns expr1 if it can be converted to true; otherwise, returns expr2. Thus, when used with Boolean values, || returns true if either operand is true; if both are false, returns

		false.
!	!expr	(Logical NOT) Returns false if its single operand can be converted to true; otherwise, returns true.

Conditional Operator

JavaScript also contains a conditional operator that assigns a value to a variable based on some condition.

Syntax

```
variablename=(condition)?value1:value2
```

Example

```
greeting=(visitor=="PRES")?"Dear President ":"Dear ";
```

If the variable **visitor** has the value of "PRES", then the variable **greeting** will be assigned the value "Dear President " else it will be assigned "Dear".

Conditional statements are used to perform different actions based on different conditions.

Conditional Statements

Very often when you write code, you want to perform different actions for different decisions. You can use conditional statements in your code to do this. In JavaScript we have the following conditional statements:

- **if statement** - use this statement to execute some code only if a specified condition is true
- **if...else statement** - use this statement to execute some code if the condition is true and another code if the condition is false
- **if...else if....else statement** - use this statement to select one of many blocks of code to be executed
- **switch statement** - use this statement to select one of many blocks of code to be executed

If Statement - Use the if statement to execute some code only if a specified condition is true.

Syntax

```
if (condition)
 {
 code to be executed if condition is true
 }
```

Note that if is written in lowercase letters. Using uppercase letters (IF) will generate a JavaScript error!

Example

```
<script type="text/javascript">
//Write a "Good morning" greeting if
//the time is less than 10

var d=new Date();
var time=d.getHours();

if (time<10)
 {
 document.write("<b>Good morning</b>");
 }
</script>
```

Notice that there is no ..else.. in this syntax. You tell the browser to execute some code **only if the specified condition is true**.

If...else Statement

Use the if....else statement to execute some code if a condition is true and another code if the condition is not true.

Syntax

```
if (condition)
 {
 code to be executed if condition is true
 }
else
 {
 code to be executed if condition is not true
 }
```

```
<script type="text/javascript">
//If the time is less than 10, you will get a "Good morning" greeting.
//Otherwise you will get a "Good day" greeting.
```

```
var d = new Date();
var time = d.getHours();

if (time < 10)
  {
  document.write("Good morning!");
  }
else
  {
  document.write("Good day!");
  }
</script>
```

If...else if...else Statement

Use the if....else if...else statement to select one of several blocks of code to be executed.

Syntax

```
if (condition1)
  {
  code to be executed if condition1 is true
  }
else if (condition2)
  {
  code to be executed if condition2 is true
  }
else
  {
  code to be executed if neither condition1 nor condition2 is true
  }
```

```
<script type="text/javascript">
var d = new Date()
var time = d.getHours()
if (time<10)
  {
  document.write("<b>Good morning</b>");
  }
```

```
else if (time>10 && time<16)
  {
  document.write("<b>Good day</b>");
  }
else
  {
  document.write("<b>Hello World!</b>");
  }
</script>
```

The JavaScript Switch Statement

Use the switch statement to select one of many blocks of code to be executed.

Syntax

```
switch(n)
{
case 1:
  execute code block 1
  break;
case 2:
  execute code block 2
  break;
default:
  code to be executed if n is different from case 1 and 2
}
```

This is how it works: First we have a single expression *n* (most often a variable), that is evaluated once. The value of the expression is then compared with the values for each case in the structure. If there is a match, the block of code associated with that case is executed. Use **break** to prevent the code from running into the next case automatically.

```
<script type="text/javascript">
//You will receive a different greeting based
//on what day it is. Note that Sunday=0,
//Monday=1, Tuesday=2, etc.

var d=new Date();
var theDay=d.getDay();
switch (theDay)
```

```
{
case 5:
 document.write("Finally Friday");
 break;
case 6:
 document.write("Super Saturday");
 break;
case 0:
 document.write("Sleepy Sunday");
 break;
default:
 document.write("I'm looking forward to this weekend!");
}
</script>
```

JavaScript has three kinds of popup boxes: Alert box, Confirm box, and Prompt box.

Alert Box

An alert box is often used if you want to make sure information comes through to the user.

When an alert box pops up, the user will have to click "OK" to proceed.

Syntax

```
alert("sometext");
```

```
<html>
<head>
<script type="text/javascript">
function show_alert()
{
alert("I am an alert box!");
}
</script>
</head>
<body>

<input type="button" onclick="show_alert()" value="Show alert box" />
```

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

Confirm Box

A confirm box is often used if you want the user to verify or accept something. When a confirm box pops up, the user will have to click either "OK" or "Cancel" to proceed. If the user clicks "OK", the box returns true. If the user clicks "Cancel", the box returns false.

Syntax

```
confirm("sometext");
```

```
<html>
<head>
<script type="text/javascript">
function show_confirm()
{
var r=confirm("Press a button");
if (r==true)
  {
  alert("You pressed OK!");
  }
else
  {
  alert("You pressed Cancel!");
  }
}
</script>
</head>
<body>

<input type="button" onclick="show_confirm()" value="Show confirm box" />

</body>
</html>
```

Prompt Box

A prompt box is often used if you want the user to input a value before entering a page. When a prompt box pops up, the user will have to click either "OK" or "Cancel" to proceed after entering an input value. If the user clicks "OK" the box returns the input value. If the user clicks "Cancel" the box returns null.

Syntax

```
prompt("sometext","defaultvalue");
```

```
<html>
<head>
<script type="text/javascript">
function show_prompt()
{
var name=prompt("Please enter your name","Harry Potter");
if (name!=null && name!="")
  {
  document.write("Hello " + name + "! How are you today?");
  }
}
</script>
</head>
<body>

<input type="button" onclick="show_prompt()" value="Show prompt box" />

</body>
</html>
```

JavaScript Loops

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

The for Loop

The for loop is used when you know in advance how many times the script should run.

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

Example

The example below defines a loop that starts with i=0. The loop will continue to run as long as **i** is less than, or equal to 5. **i** will increase by 1 each time the loop runs.

```
<html>
<body>
<script type="text/javascript">
var i=0;
for (i=0;i<=5;i++)
{
document.write("The number is " + i);
document.write("<br />");
}
</script>
</body>
</html>
```

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

The while Loop : The while loop loops through a block of code while a specified condition is true.

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

Note: The '<=' could be any comparing operator.

Example

The example below defines a loop that starts with i=0. The loop will continue to run as long as **i** is less than, or equal to 5. **i** will increase by 1 each time the loop runs:

```
<html>
<body>
<script type="text/javascript">
var i=0;
while (i<=5)
 {
 document.write("The number is " + i);
 document.write("<br />");
 i++;
 }
</script>
</body>
</html>
```

The do...while Loop

The do...while loop is a variant of the while loop. This loop will execute the block of code ONCE, and then it will repeat the loop as long as the specified condition is true.

Syntax

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

Example

The example below uses a do...while loop. The do...while loop will always be executed at least once, even if the condition is false, because the statements are executed before the condition is tested:

```
<html>
<body>
<script type="text/javascript">
var i=0;
do
 {
 document.write("The number is " + i);
 document.write("<br />");
 i++;
 }
while (i<=5);
</script>
</body>
</html>
```

The break Statement

The break statement will break the loop and continue executing the code that follows after the loop (if any).

Example

```
<html>
<body>
<script type="text/javascript">
var i=0;
for (i=0;i<=10;i++)
 {
 if (i==3)
  {
  break;
  }
 document.write("The number is " + i);
 document.write("<br />");
 }
</script>
</body>
</html>
```

The continue Statement

The continue statement will break the current loop and continue with the next value.

```
<html>
<body>
<script type="text/javascript">
var i=0
for (i=0;i<=10;i++)
 {
 if (i==3)
  {
  continue;
  }
 document.write("The number is " + i);
 document.write("<br />");
 }
</script>
</body>
</html>
```

What is an Array?

An array is a special variable, which can hold more than one value at a time. If you have a list of items (a list of car names, for example), storing the cars in single variables could look like this:

```
var car1="Saab";
var car2="Volvo";
var car3="BMW";
```

What if you want to loop through the cars and find a specific one? And what if you had not 3 cars, but 300?

The solution is an array!

An array can hold many values under a single name, and you can access the values by referring to an index number.

Create an Array

An array can be created in three ways.

The following code creates an Array object called myCars:

1: Regular:

```
var myCars=new Array();
myCars[0]="Saab";
myCars[1]="Volvo";
myCars[2]="BMW";
```

2: Condensed:

```
var myCars=new Array("Saab","Volvo","BMW");
```

3: Literal:

```
var myCars=["Saab","Volvo","BMW"];
```

Access an Array

You refer to an element in an array by referring to the **index** number.

This statement access the value of the first element in myCars:

```
var name=myCars[0];
```

This statement modifies the first element in myCars:

```
myCars[0]="Opel";
```

JavaScript Foreach

In JavaScript, we make use of the for() loop construct to fake a foreach scenario. It looks something like this:

```
myArray = new Array("item 1", "item 2", "item 3");

for(i=0; i<myArray.length; i++) {

        myArray[i] = "Do Something Here".

}
```

Introduction to Built in Classes

[**Refer:** *Javascript_Classes.PDF*]

Built-In JavaScript Objects

JavaScript has many predefined, built-in objects that enable to work with Strings and Dates, and perform mathematical operations.

String

In JavaScript, there are two types of string data types: primitive strings and *String* objects. String objects have many methods for manipulating and parsing strings of text. Because these methods are available to primitive strings as well, in practice, there is no need to differentiate between the two types of strings.

var webucator ="Webucator".

Property	Description
length	Read-only value containing the number of characters in the string.

```
1  webucator.length
2  //Returns 9
```

Method	Description

Method	Description
charAt(position)	Returns the character at the specified position. Note that it is zero-based, so the first letter in the string is at position 0.

```
1  webucator.charAt(4)
2  //Returns c (the fifth character)
```

charCodeAt(position)	Returns the Unicode character code of the character at the specified position.

```
1  webucator.charCodeAt(4)
2  //Returns 99
```

fromCharCode(characterCodes)	Returns the text representation of the specified comma-delimited character codes. Used with String rather than a specific String object.

```
1  String.fromCharCode(169)
2  //Returns ©
```

indexOf(substring,startPosition)	Searches from startPosition for substring. Returns the position at which the substring is found. If substring is not found, returns -1.

```
1  webucator.indexOf("cat");
2  //Returns 4
3
4  webucator.indexOf("cat", 5);
```

Method	Description
5 //Returns -1	
lastIndexOf(substring,endPosition)	Searches from the end of the string for substring until endPosition is reached. Returns the position at which the substring is found. If substringis not found, returns -1.
1 webucator.lastIndexOf("cat"); 2 //Returns 4 3 4 webucator.lastIndexOf("cat", 5); 5 //Returns 4	
substring(startPosition,endPosition)	Returns the substring beginning at startPosition and ending with the character before endPosition. endPosition is optional. If it is excluded, the substring continues to the end of the string.
1 webucator.substring(4, 7); 2 //Returns cat 3 4 webucator.substring(4); 5 //Returns cator	
substr(startPosition,length)	Returns the substring of Length characters beginning at startPosition.length is optional. If it is excluded, the substring continues to the end of the string.
1 webucator.substr(4, 3);	

Method	Description
2 //Returns cat 3 4 webucator.substr(4); 5 //Returns cator	
slice(startPosition,endPosition)	Same as substring(startPosition, endPosition).
1 webucator.slice(4, 7); 2 //Returns cat	
slice(startPosition,positionFromEnd)	positionFromEnd is a negative integer. Returns the substring beginning atstartPosition and ending positionFromEnd characters from the end of the string.
1 webucator.slice(4, -2); 2 //Returns cat	
split(delimiter)	Returns an array by splitting a string on the specified delimiter.
1 var s = "A,B,C,D"; 2 var a = s.split(","); 3 document.write(a[2]); 4 //Returns C	
toLowerCase()	Returns the string in all lowercase letters.

Method	Description
webucator.toLowerCase() //Returns webucator	
toUpperCase()	Returns the string in all uppercase letters.
webucator.toUpperCase(); //Returns WEBUCATOR	

Math

The Math object is a built-in static object. The Math object's properties and methods can be accessed directly (e.g, Math.PI) and are used for performing complex math operations. Here are two commonly used properties of the Math object.

Property	Description
Math.PI	Pi (Π)
Math.PI; //3.141592653589793	
Math.SQRT2	Square root of 2.
Math.SQRT2; //1.4142135623730951	

Method	Description
Math.abs(number)	Absolute value of number.
Math.abs(-12); //Returns 12	

Method	Description
Math.ceil(number)	number rounded up.
1 Math.ceil(5.4); 2 //Returns 6	
Math.floor(number)	number rounded down.
1 Math.floor(5.6); 2 //Returns 5	
Math.max(numbers)	Highest Number in numbers.
1 Math.max(2, 5, 9, 3); 2 //Returns 9	
Math.min(numbers)	Lowest Number in numbers.
1 Math.min(2, 5, 9, 3); 2 //Returns 2	
Math.pow(number, power)	number to the power of power.
1 Math.pow(2, 5); 2 //Returns 32	
Math.round(number)	Rounded number.
1 Math.round(2.5); 2 //Returns 3	
Math.random()	Random number between 0 and 1.
1 Math.random(); 2 //Returns random 3 //number from 0 to 1	

Method for Generating Random Integers

You can easily generate random numbers in JavaScript using the Math.random() method.

```
var rndDec = Math.random();
```

Date

The Date object has methods for manipulating dates and times. JavaScript stores dates as the number of milliseconds since January 1, 1970.

Method	Description
getDate()	Returns the day of the month (1-31).
1 rightNow.getDate(); 2 //Returns 14	
getDay()	Returns the day of the week as a number (0-6, 0=Sunday, 6=Saturday).
1 rightNow.getDay(); 2 //Returns 4	
getMonth()	Returns the month as a number (0-11, 0=January, 11=December).
1 rightNow.getMonth(); 2 //Returns 3	
getFullYear()	Returns the four-digit year.
1 rightNow.getFullYear(); 2 //Returns 2011	
getHours()	Returns the hour (0-23).
1 rightNow.getHours(); 2 //Returns 0	

Method	Description
getMinutes()	Returns the minute (0-59).

```
1  rightNow.getMinutes();
2  //Returns 23
```

Method	Description
getSeconds()	Returns the second (0-59).

```
1  rightNow.getSeconds();
2  //Returns 54
```

Method	Description
getMilliseconds()	Returns the millisecond (0-999).

```
1  rightNow.getMilliseconds();
2  //Returns 650
```

Method	Description
getTime()	Returns the number of milliseconds since midnight January 1, 1970.

```
1  rightNow.getTime();
2  //Returns 1113452634650
```

Method	Description
getTimezoneOffset()	Returns the time difference in minutes between the user's computer and GMT.

```
1  rightNow.getTimezoneOffset();
2  //Returns 240
```

Method	Description
toLocaleString()	Returns the Date object as a string.

```
1  rightNow.toLocaleString();
2  //Returns Thursday, April 14,
3  //2011 12:23:54 AM
```

Method	Description
toGMTString()	Returns the Date object as a string in GMT timezone.

```
1  rightNow.toGMTString();
2  //Returns Thu, 14 Apr 2011
```

Method	Description
3	//04:23:54 UTC

A few things to note:

- To create a Date object containing the current date and time, the Date() constructor takes no arguments.
- When passing the date as a string to the Date() constructor, the time portion is optional. If it is not included, it defaults to 00:00:00. Also, other date formats are acceptable (e.g, "4/14/2011" and "14-04-2011").
- When passing date parts to the Date() constructor, dd, hh, mm, ss, and ms are all optional. The default of each is 0.
- Months are numbered from 0 (January) to 11 (December).

Javascript Events & Event Handlers

Events are keyboard, mouse or other actions that can be detected by JavaScript. Every element on a web page has certain events, which can trigger a JavaScript. For example, we can use the onClick event of a button element to indicate that a function will run when a user clicks on the button. We define the events in the HTML tags.

Examples of events:

- A mouse click
- A web page or an image loading
- Mousing over a hot spot on the web page
- Selecting an input field in an HTML form
- Submitting an HTML form
- A keystroke

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

Events Handlers

Event Handlers are very powerful and useful mechanism. They are *JavaScript code that are not added inside the <script> tags, but rather, inside the html tags,* that execute JavaScript when something happens, such as pressing a button, moving your mouse over a link, submitting a form etc.

To allow you to run your bits of code when these events occur, JavaScript provides ***event handlers***. All the event handlers in JavaScript start with the word on, and each event handler deals with a certain type of event. Here's a list of all the event handlers in JavaScript, along with the objects they apply to and the events that trigger them:

The basic syntax of these event handlers is:

name_of_handler="JavaScript code here"

For example:

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

When the above link is clicked, the user will first see an alert message before being taken to Google.

Different event handlers with with different HTML tags. For example, while "onclick" can be inserted into most HTML tags to respond to that tag's onclick action, something like "onload" only works inside the <body> and <img> tags. Below are some of the most commonly used event handlers supported by JavaScript:

Event Handlers:

Event handler	Applies to:	Triggered when:
onAbort	Image	The loading of the image is cancelled.
onBlur	Button, Checkbox, FileUpload, Layer, Password, Radio, Reset, Select, Submit, Text, TextArea, Window	The object in question loses focus (e.g. by clicking outside it or pressing theTAB key).
onChange	FileUpload, Select, Text, TextArea	The data in the form element is changed by the user.
onClick	Button, Document, Checkbox, Link, Radio, Reset, Submit	The object is clicked on.
onDblClick	Document, Link	The object is double-clicked on.
onDragDrop	Window	An icon is dragged and dropped into the browser.
onError	Image, Window	A JavaScript error occurs.
onFocus	Button, Checkbox, FileUpload, Layer, Password, Radio, Reset, Select, Submit, Text, TextArea, Window	The object in question gains focus (e.g. by clicking on it or pressing the TAB key).
onKeyDown	Document, Image, Link, TextArea	The user presses a key.
onKeyPress	Document, Image, Link, TextArea	The user presses or holds down a key.
onKeyUp	Document, Image, Link, TextArea	The user releases a key.
onLoad	Image, Window	The whole page has finished loading.

onMouseDown	Button, Document, Link	The user presses a mouse button.
onMouseMove	None	The user moves the mouse.
onMouseOut	Image, Link	The user moves the mouse away from the object.
onMouseOver	Image, Link	The user moves the mouse over the object.
onMouseUp	Button, Document, Link	The user releases a mouse button.
onMove	Window	The user moves the browser window or frame.
onReset	Form	The user clicks the form's Reset button.
onResize	Window	The user resizes the browser window or frame.
onSelect	Text, Textarea	The user selects text within the field.
onSubmit	Form	The user clicks the form's Submit button.
onUnload	Window	The user leaves the page.

Using an event handler

To use an event handler, you usually place the event handler name within the HTML tag of the object you want to work with, followed by ="*SomeJavaScriptCode*", where*SomeJavaScriptCode* is the JavaScript you would like to execute when the event occurs.

For example:

```
<input type="submit" name="clickme" value="Click Me!" onclick="alert('Thank You!')"/>
```

JavaScript Functions

- A function is really nothing more than a named block of code.
- It is a statement block that has been assigned a name.
- Functions are small groups of instructions that are carried out when they are called upon.
- It is a reusable code-block that will be executed by an event, or when the function is called.
- A function typically contains a set of commands for a specific purpose, which you want to run at a certain time.
- Each function in a script is given a unique name.
- The layout of a function always follows the same format, including these three things:
  - The word "**function**"
    - **Function** tells the browser "Do these instructions when this name is called upon.".
  - The function's name, followed by parentheses (which may or may not be empty)
    - Function's name must be unique and every function name is contains the parentheses, which may be empty if the function does not take any parameter.
  - Curly braces containing the function's code.
    - The curly brackets (**{ }**)are used to contain the set of instructions.
- A script can contain any number of functions, according to the terms and conditions.

Defining Functions:

- The functions in javascript is defined by using the **function** keyword followed by function-name and the parentheses.
- *The Syntax of defining the functions:*

```
function function-Name(Argument lists...)
{
        /*
                some lines of codes
                return;
        */
}
```

- The **function** here is a keyword and should be explictly written in-front of the function name.

- The function is followed by the function name, which should be unique and should follow all the naming conventions that any function name should follow.
- The parentheses should be attached with every function name and in-between the parantheses you should lists the parameters if the function contains any parameter.
- If not then you must include an empty parentheses, but you should include parantheses.
- Then every function contains some lines of codes and these codes are included within the curly braces.
- If the function returns any value then it is returned with the keyword **return** from the function.

Invoking Functions

- Functions do not run automatically. When the page loads, each function waits quietly until it is told to run.
- To run a function you must *call* it.
- This means sending an instruction to the function telling it to run.
- ***There are two common ways to call a function:***
  - From an *event handler* and
  - From another function.
  - *Calling Functions from Event-handlers*
    - An event handler is a command, which calls a function when an event happens, such as the user clicking a button.
    - The command is written in the format **on*Event***, where ***Event*** is the name for a specific action.
    - *Here are some common examples:*
    - User clicks a button (**onClick**):

      <input type="button" value="Click Here" **onClick**="doSomething();">
    - User places their cursor in a text field (**onFocus**):

      <input type="text" **onFocus**="doSomething();">

  - ***Calling Functions from another Functions***
    - Functions can also call other functions. Simply enter the name of the function to be called, with its parentheses.

    - ***Example:***

      function doSomething()

```
{
        doSomethingElse(); //this line calls the another function
}
function doSomethingElse()
{
        //the function codes go here
}
```

The Function constructor

- JavaScript has a Function constructor which allows you create a variable and assign it a function as a value.
- *Example:*

```
var q = new Function('a', 'b', 'return a / b;');
```

- The Function constructor takes any number of arguments, each of which must be defined as a string, and the last of which must be the body of the code for the function.
- A function defined this way can still be invoked by the variable named, res = q(20, 5).
- But technically it has no name. Instead the function name is a variable that contains a reference to that function as a value.
- Such function are referred to as *anonymous* functions, since there is no way to refer to them by name.
- The benefit of declaring a function this way is that it is reparsed each time it is run, meaning, you could replace the strings that define the arguments and function body with variables, dynamically assign values to these variables, and thus rewrite the function each time it ran.
- Since the function is reparsed each time, it can really slow down processing if it is run repeatedly.

Function Properties

- As an object, a function has other properties as well.
- One is the length property, which specifies how many arguments the function is expecting.
- As with objects, you can also define properties for functions.
- This allows you to create values that persist across repeated calls to the function.
- To create a new property, you can just assign it a value.

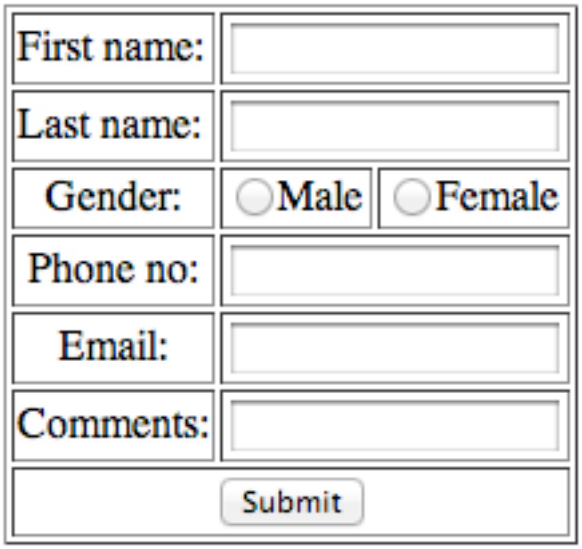
- Since the function declaration is parsed before the code in the script is executed, we can put the initialization statement outside of the function.

- **Example:**

```
q.counter = 0;
function q (x)
{
        q.counter ++;
        //statements of the function
}
```

- Each time function q runs, it will increment the property q.counter by one.

Form Validation : EXAMPLE




```
<html>
<head>
<title>Form Validation</title>
<script language="javascript">
function validate()
{
var b=document.frm1.txtEmail.value;
var len=b.length;
var a='@';
var dot='.';
var valid1,valid2;
```

```
//form validation for First Name
if(document.frm1.txtFirstName.value=="")
{
alert("The First Name is Empty");
document.frm1.txtFirstName.focus();
return false;
}
// form validation for Last Name
else if(document.frm1.txtLastName.value=="")
{
alert("The Last Name is Empty");
document.frm1.txtLastName.focus();
return false;
}
//form validation for Gender
else if(document.frm1.optGender[0].checked==false &&
document.frm1.optGender[1].checked==false)
{
alert("Check for the Gender option");
return false;
}
//form validation for Phone no
else if(document.frm1.txtPhone.value=="")
{
alert("The Phone no field is Empty");
document.frm1.txtPhone.focus();
return false;
}
//for checking numbers in Phone no field
else if(isNaN(document.frm1.txtPhone.value))
{
alert("Please enter the valid numbers");
document.frm1.txtPhone.focus();
return false;
}
//validation for Comment
else if(document.frm1.txtcomment.value=="")
{
alert("The comment field is empty");
document.frm1.txtcomment.focus();
return false;
}
//validation for Email
```

```
else if(document.frm1.txtEmail.value=="")
{
alert("The Email field is empty");
document.frm1.txtEmail.focus();
return false;
}
//validation for correct email
else if(b!="")
{
for(var i=0;i<=len;i++)
{
if(b.charAt(0)!=a)
{
if(b.charAt(i)==a)
valid1=true;
if(b.charAt(i)==dot)
{
valid2=true;
}
}
else
{
alert("EnterValidEmail address");
document.frm1.txtEmail.focus();
}
}
if(valid1!=true || valid2!=true)
{
alert("Please enter the valid email");
document.frm1.txtEmail.focus();
return false;
}
}
}
</script>
</head>
<body>
<form name="frm1">
<table border=1 align="center">
<tr align>
<td>First name:</td>
<td colspan=2><input type="textbox" name="txtFirstName"></td>
</tr>
<tr align>
```

```
<td>Last name:</td>
<td colspan=2><input type="text" name="txtLastName"></td>
</tr>
<tr align="center">
<td>Gender:</td>
<td><input type="radio" name="optGender">Male</td>
<td><input type="radio" name="optGender">Female</td>
</tr>
<tr align="center">
<td>Phone no:</td>
<td colspan=2><input type="text" name="txtPhone"></td>
</tr>
<tr align="center">
<td>Email:</td>
<td colspan=2><input type="text" name="txtEmail"></td>
</tr>
<tr align="center">
<td>Comments:</td>
<td colspan=2><input type="textarea" name="txtcomment" row=5 col=3></td>
</tr>
<tr align="center">
<td colspan=3><input type="submit" value="Submit" onclick="validate()"></td>
</tr>
</table>
</form>
</body>
</html>
```

Cookies

Web Browser and Server use HTTP protocol to communicate and HTTP is a stateless protocol. But for a commercial website it is required to maintain session information among different pages. For example one user registration ends after completing many pages.

In many situations, using cookies is the most efficient method of remembering and tracking preferences, purchases, commissions, and other information required for better visitor experience or site statistics.

How It Works ?

Your server sends some data to the visitor's browser in the form of a cookie. The browser may accept the cookie. If it does, it is stored as a plain text record on the visitor's hard drive. Now, when the visitor arrives at another page on your site, the browser sends the same cookie to the server for retrieval. Once retrieved, your server knows/remembers what was stored earlier.

Cookies are a plain text data record of 5 variable-length fields:

- **Expires :** The date the cookie will expire. If this is blank, the cookie will expire when the visitor quits the browser.
- **Domain :** The domain name of your site.
- **Path :** The path to the directory or web page that set the cookie. This may be blank if you want to retrieve the cookie from any directory or page.
- **Secure :** If this field contains the word "secure" then the cookie may only be retrieved with a secure server. If this field is blank, no such restriction exists.
- **Name=Value :** Cookies are set and retrieved in the form of key and value pairs.

Cookies were originally designed for CGI programming and cookies' data is automatically transmitted between the web browser and web server, so CGI scripts on the server can read and write cookie values that are stored on the client.

JavaScript can also manipulate cookies using the *cookie* property of the *Document* object. JavaScript can read, create, modify, and delete the cookie or cookies that apply to the current web page.

Storing Cookies:

The simplest way to create a cookie is to assign a string value to the *document.cookie* object, which looks like this:

```
document.cookie = "key1=value1;key2=value2;expires=date";
```

Here *expires* attribute is option. If you provide this attribute with a valid date or time then cookie will expire at the given date or time and after that cookies' value will not be accessible.

Note: Cookie values may not include semicolons, commas, or whitespace. For this reason, you may want to use the JavaScript *escape()* function to encode the *value* before storing it in the cookie. If you do this, you will also have to use the corresponding *unescape()* function when you read the cookie value.

Example:

Following is the example to set a customer name in *input* cookie.

```
<html>
<head>
<script type="text/javascript">
<!--
function WriteCookie()
{
  if( document.myform.customer.value == "" ){
    alert("Enter some value!");
    return;
  }
  cookievalue= escape(document.myform.customer.value) + ";";
  document.cookie="name=" + cookievalue;
  alert("Setting Cookies : " + "name=" + cookievalue );
}
//-->
</script>
</head>
<body>
<form name="myform" action="">
Enter name: <input type="text" name="customer"/>
```

```
<input type="button" value="Set Cookie" onclick="WriteCookie();"/>

</form>

</body>

</html>
```

View the output. Enter something in the text box and press the button "Set Cookie" to set the cookies.

Now your machine has a cookie called *name*. You can set multiple cookies using multiple *key=value* pairs separated by comma.

Reading Cookies:

Reading a cookie is just as simple as writing one, because the value of the *document.cookie* object is the cookie. So you can use this string whenever you want to access the cookie.

The *document.cookie* string will keep a list of *name=value* pairs separated by semicolons, where *name* is the *name* of a cookie and value is its string value.

You can use strings' *split()* function to break the string into key and values as follows:

Example:

Following is the example to get the cookies set before.

```
<html>

<head>

<script type="text/javascript">

<!--

function ReadCookie()

{

  var allcookies = document.cookie;

  alert("All Cookies : " + allcookies );

  // Get all the cookies pairs in an array

  cookiearray  = allcookies.split(';');



  // Now take key value pair out of this array
```

```
    for(var i=0; i<cookiearray.length; i++){
      name = cookiearray[i].split('=')[0];
      value = cookiearray[i].split('=')[1];
      alert("Key is : " + name + " and Value is : " + value);
    }
  }
  //-->
  </script>
  </head>
  <body>
  <form name="myform" action="">
  <input type="button" value="Get Cookie" onclick="ReadCookie()"/>
  </form>
  </body>
  </html>
```

View the output, press the button "Get Cookie" to see the cookies.

Note: *There may be some other cookies already set on your machine. So above code will show you all the cookies set at your machine.*

Setting the Cookies Expiration Date:

You can extend the life of a cookie beyond the current browser session by setting an expiration date and saving the expiration date within the cookie. This can be done by setting the *expires* attribute to a date and time.

Example:

```
<html>
<head>
<script type="text/javascript">
<!--
function WriteCookie()
{
  var now = new Date();
  now.setMonth( now.getMonth() + 1 );
  cookievalue = escape(document.myform.customer.value) + ";"
  document.cookie="name=" + cookievalue;
  document.cookie = "expires=" + now.toUTCString() + ";"
  alert("Setting Cookies : " + "name=" + cookievalue );
}
//-->
</script>
</head>
<body>
<form name="formname" action="">
Enter name: <input type="text" name="customer"/>
<input type="button" value="Set Cookie" onclick="WriteCookie()"/>
</form>
</body>
</html>
```

Deleting a Cookie:

To delete cookies, you just need to set the expiration date to a time in the past.

Example:

```
<html>
<head>
<script type="text/javascript">
<!--
function WriteCookie()
```

```
{
  var now = new Date();
  now.setMonth( now.getMonth() - 1 );
  cookievalue = escape(document.myform.customer.value) + ";"
  document.cookie="name=" + cookievalue;
  document.cookie = "expires=" + now.toUTCString() + ";"
  alert("Setting Cookies : " + "name=" + cookievalue );
}
//-->
</script>
</head>
<body>
<form name="formname" action="">
Enter name: <input type="text" name="customer"/>
<input type="button" value="Set Cookie" onclick="WriteCookie()"/>
</form>
</body>
</html>
```

ASSIGNMENT 05

ERROR HANDLING in JAVASCRIPT PROGRAMS

Compile on a DOC file and send mail to *saroj@sarojpandey.com.np*

Lab Questions

1. Write a program that finds the sum of odd numbers from 0 to 100. i.e. sum = 1 + 3 + ...99.

2. Write a program to generate the Fibonacci series less than 200.

3. Write a program to calculate the factorial of the given number.

4. Write a program to display a number randomly. The Random number should be generated when you click upon a link 'Click Me'. If the number is less than 20 then show a alert box with a message "Hey (random Number) is less than 20" , and if the random number is greater than 20 then show a alert box with a message "Your (random Number) is greater than or equals to 20".

5. Write a program in JavaScript to accept any three numbers from a user and display the largest of those entered numbers.

6. Use JavaScript codes to create a table. Input the number of rows and columns from user and display the table with the value 1, 2, 3.....in the <td> field.

7. Write a program to display output like the following:

N	10 * N	100 * N	1000 * N
1	10	100	1000

You should ask the starting and ending value for N and the table should be displayed dynamically according to the value inputted by the user.

8. Write a program that displays the continuous time in the web page. The Time should be in the format of HH: MM: SS.

9. If thisstring = "Internet World", what value will be returned by the substring method thisstring.substring (9)?

10. Write a program that will change the background color in every 2 seconds. Use at least 10 colors.

11. Given a single Text-box, write a program to store the value of this textbox into a cookie and also display the value of cookie when the page is loaded.

12. Write a program which includes a function sum(). This function sum() should be designed to add an arbitrary list of parameters. (For e.g. if you call the function sum () as sum (2, 3) it should return the result 5 and if again you call the function sum() as sum(2,3,4) it should return the result 9).

13. Write down a program to covert a decimal number 24 to hexadecimal, octal and binary number.

14. Write a program to ask the background color with user and change the background of the page.

15. Write a program to ask the color value and change the color of the scrollbar.

16. Write a program to ask a URL with user and open that URL in new browser window.

17. Write a program to add 2 matrixes, ask the size & values of the matrix from user.

18. WAP to calculate compound interest for the given principle, no. of years and rate of interest.
    [Hint: $C=P[(1 + r/100)^{n-1}]$]

19. Write a program to create two text-boxes and two buttons. If the user inputs the value in first text-box and click upon the button then the value entered into the first text-box should be displayed into the second text-box (in uppercase if the user have inputted in lowercase and vice-versa), similarly for the second text-box.

20. Write a program that reads 5 words from user (Use Prompt for receiving input) and displays the word that has 'ha' continuous character in between the words.

21. Write a program that will ask height & width and change the size of the browser window.

22. Write a program that will show the how long you have been visiting the site. Show the time in H:M:S format.
23. Write a program that will show a text box and 2 radio buttons named start & stop. When Start is clicked then display the current time in textbox. The time will be stopped while the user selects the end radio button.
24. Write a program to store 10 names in array and print the array elements by joining them.

25. WAP to read a no. & find out if it is Armstrong no. or not.

26. Write a program which contains 2 textbox, first to ask for the mathematical expression and second for result which need to be read-only. Clicking upon Calculate button evaluate the expression and show the result.
27. Write a program that will display current system time in status bar.

28. Find the cube, square & square root of a number given by user. Use the methods of math object.

29. WAP to input two points A(x1,y1) & B(x2,y2) and find the distance between them.

30. Write a program that will display current date and time in following format.

Saturday, August 25, 2010 10:50 AM

31. Write a program to display the text “Hello, This is my First Sliding Text Example”, in a text-box in a sliding manner. First the letter ‘H’ displays in the text-box then ‘e’ then simultaneous till last ‘e’ of Example. The letter should be displayed after every 2 seconds.

32. Write a program which display a button, clicking on that the browser window needs to be closed.

33. Write HTML code to embed the Video & Audio in web page.

~