

Architecture of the Internet

Chapter 1 Internet Concepts	4
Introduction	4
Connecting to a Network	5
World Wide Web.....	6
Internet Service Providers (ISPs)	12
The Structure of the Internet.....	13
Uniform Resource Locators (URLs)	15
The Domain Name System (DNS)	16
The Client-Server Relationship	17
More on Browsers	18
Main Elements of Web Browsers	19
Other Web Browser Choices	20
Hyperlinks:	22
Browser Cache:.....	22
Search Engines	23
Email.....	24
File Transfer	25
Client-Server Architecture.....	25
Peer-to-Peer Architecture	31
Static vs. Dynamic pages.....	35
Types of Web programming.....	38
Exercise 1	39
Chapter 2: Basic Web Applications.....	41
Email.....	41
E Commerce	53
Web portal	63
Search engines:	69
E-Learning.....	78
Exercise 2	82
Chapter 3 Internet and Web Security	84
Security:.....	84
Threats, vulnerability and risk mitigation.....	89
Viruses, Worms and Spams.....	96
Antivirus and Antispam Software	105
Firewall.....	113
Exercise 3	122
Chapter 4 Societal Impact of the Web.....	125

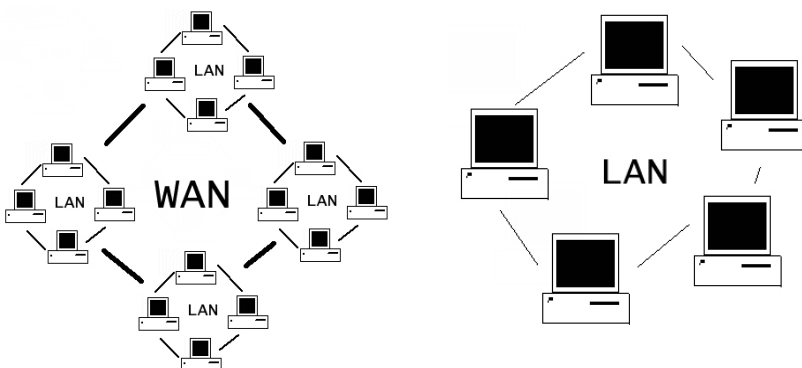
The Net Generation and the new Web culture	125
The Digital divide:	128
Blogging	130
Chapter 5: Laboratory 1 Hypertext Markup Language	138
HTML Project 1	210
Exercise 5	212
Chapter 5 Laboratory 2 CSS Cascaded Style Sheet	215
CSS Projects:	246
Exercise 6	247
Chapter 7 Laboratory 3 Java Script	248
Java Script Project	313
Exercise 7	315

Chapter 1 Internet Concepts

Introduction

Over the past century and a half, important technological developments have created a global environment that is drawing the people of the world closer and closer together. Currently, we are in the information Age, where magnifying the computation power is an essential Goal.

A computer network is two or more computers connected together. When the networked computers are physically near each other, it is called we have a local area network or LAN. While if they are not near each other, it is called we have a wide area network or WAN. In other definition, WAN is a collection of LANs. When networks LANs and WANs are connected to each other, interconnected network is formed.



When computers are networked together, they allow us to generate exchange, share and manipulate information in an uncountable number of ways. So, what is the Internet? **Internet** is a specific interconnected network that connects computers all over the world using a common set of interconnection standards or protocols. So, internet is a part of the system that is primarily including hardware infrastructure (Telecommunications, routers, servers, disk drives,).

Connecting to a Network

There is a difference between to connect to a network and to communicate over a network. Connecting **to** a network requires that special hardware be installed in your computer. This hardware is called connection devices. There are two common types of connection devices:

- 1- **Modem** – short for modulator / demodulator - which connects computers using a standard telephone line
- 2- Network interface card (**NIC**) – which connects computers using a special type of network cabling

Note that, NICs are not synonymous with Ethernet cards. An Ethernet card is a particular type of NIC, and is the most popular in personal computers (PCs). Most of the time, these devices are internal hardware, although they are available as an external peripherals.

In order to communicate with another computer **over** a network, you must do two things:

1. Use the set of **rules** governing communication over the network, called a **protocol**. Your computer will generally handle this.
2. Know the **address** of the computer you want to communicate with. There are two types of network addresses:
 - a. Medium access control (**MAC**) address which is used inside a single network
 - b. Internet protocol (**IP**) address which is used on the Internet

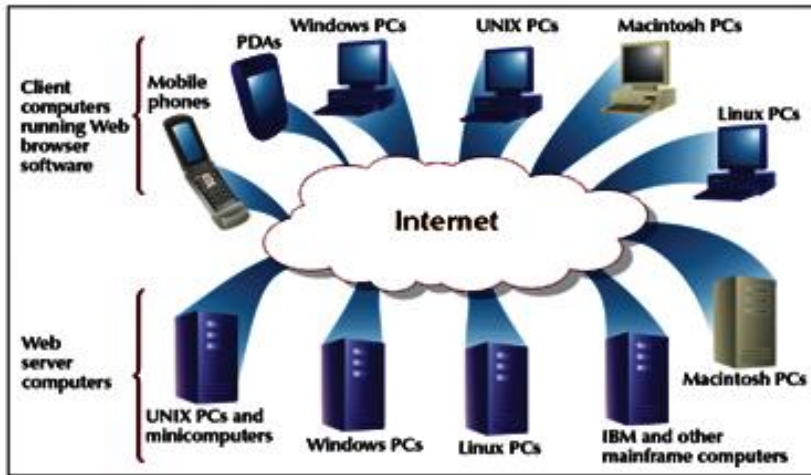
World Wide Web

The World Wide Web (web for short or **www**) *is a collection of interlinked multimedia documents that are stored on the Internet and accessed using a common protocol (HTTP)*. From this definition we can say that the web is part of the system that is primarily include software components (text files, graphics files, sound files, video files...). So, what is the relation between the internet and the World Wide Web? *World Wide Web (WWW) is an Internet based software application*. Other Internet applications are:

- Email
- ftp (file transfer protocol)
- Messenger

More than 90 percent of the internet applications are World Wide Web so it is valid to use Internet and World Wide Web terms interchangeably. World Wide Web is tied up to several terms such as web server, web browser and HTML or Hypertext Markup Language. In the following paragraph those terms are briefly explained.

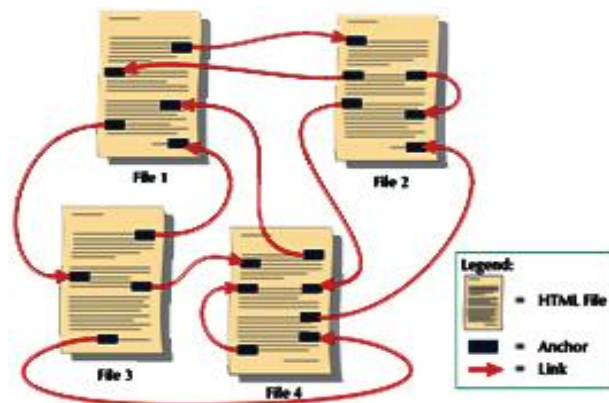
A **web server** is a *computer programs that delivers (serves) content, such as web pages, using the Hypertext Transfer Protocol.* The term **web server** can also refer to a computer connected to the Internet that contains files their owners have made available publicly through their Internet connections. When you use your Internet connection to become part of the Web, your computer becomes a **Web client** in a worldwide network called a **client/server network** that will be discussed later. *Web browsers are software that you run on your computer to make it work as a Web client.* The following figure show how servers and clients are organized in the client server structure to enable the World Wide Web application to run in an effective way.



There are several **Web servers** available today, however the most popular are:

- Apache HTTP Server which occupies (69.01%) of the general use. It is developed in 1994 by Rob McCool. The original core system has many patches applied to it, and thus its name. Apache has dominated the Web since 1996. it is Available for free charge. It Runs on operating systems including FreeBSD-UNIX, HP-UX, Linux, Microsoft Windows, SCO-UNIX and Solaris
- Microsoft Internet Information Server (**IIS**) which occupies (23.26%) of the general use. It comes bundled with Microsoft Windows Server operating system. IIS used on many corporate intranets (Microsoft standard product). It was originally written to run on Windows NT and Windows 2000. It runs on Windows 2003 Server and Windows XP. It Supports ASP, ActiveX Data objects and SQL queries
- Sun Java System Web Server (**JSWS**) (former names are Sun One, iPlanet Enterprise Server and Netscape Enterprise Server) (0.86%)

Hypertext Markup Language or (**HTML**) is a *standard language used on the Web to format documents. HTML uses codes (tags) to tell the Web browser software how to display text.* HTML document is defined as a text file that contains HTML tags. When a Web browser displays an HTML document, it is referred to as a **Web page**. One of the most important features of html is **HTML anchor tag** which *enables Web designers to **link** HTML documents to each other and create Hypertext links.* Hypertext links can connect HTML documents together or can connect one part of HTML document to another part. When hyperlinks connect to files that contain pictures, graphics, and media objects such as sound and video clips, the whole thing is called Hypermedia links.



Having a collection of linked Web pages with a common theme or focus is called a **website**. Each website should have a main or home page to start the web site from.

Each computer on the Web is given a unique identification number called Internet Protocol Address or (**IP**). Because IP addresses are

tedious to deal with and hard to remember, *an IP address may also be assigned a host or domain name*. You can think of an IP address as someone's physical street address and the host/domain as Ahmed's House. So, *Domain name are unique name associated with specific IP address by a program that runs on an Internet host computer. This program is called DNS (Domain Name System) software.*

Domain Name Software or (DNS) is an Internet service that translates domain names into IP addresses. Because domain names are alphabetic, they're easier to remember. The Internet however, is really based on IP addresses. Every time you use a domain name, therefore, a DNS service must translate the name into the corresponding IP address. For **example**, the *domain name www.example.com might translate to 198.105.232.4*. The host computer that runs the DNS service or software is called Domain name server or DNS server. If one DNS server doesn't know how to translate a particular domain name, it asks another one, and so on, until the correct IP address is returned. *The last part of domain name is called its top-level domain (TLD)*. The following table shows the most common Top Level Domains (TLDs).

Original General TLDs		Country TLDs		General TLDs Added Since 2000	
TLD	Use	TLD	Country	TLD	Use
.com	U.S. Commercial	.au	Australia	.uk	United Kingdom
.edu	U.S. Four-year educational institution	.ca	Canada	.asia	Companies, individuals, and organizations based in Asian-Pacific regions
.gov	U.S. Federal government	.de	Germany	.biz	Businesses
.mil	U.S. Military	.fi	Finland	.info	General use
.net	U.S. General use	.fr	France	.int	International organizations and programs endorsed by a treaty between or among nations
.org	U.S. Not-for-profit organization	.jp	Japan	.name	Individual persons
.us	U.S. General use	.se	Sweden	.pro	Professionals (such as accountants, lawyers, physicians)

Now let us talk about the **addressing schema** used to tell the browser the needed *information* to *allocate* the wanted *web site* to be *displayed*. The Addressing schema consists of **four parts** and it is known as *uniform resource allocator (URL)*. URLs, or **Uniform Resource Locators**, are the *schema* by which *documents* or *data* are *addressed* in the *World Wide Web*. The **URL** contains the following information:

- **Transfer protocol** to use when transporting the file
- **Domain name** of computer on which file resides
- **Pathname** of folder or directory on computer on which file resides
- **Name** of the file

The following is an outline of the most common form of a URL:



Internet Service Providers (ISPs)

Most commonly, people connect to the Internet through an Internet Service Provider (ISP), also sometimes referred to as an Internet access provider (IAP). *ISP is a **company** that offers its customer access to the Internet.* The ISP connects to its customers using a data transmission technology appropriate for delivering Internet Protocol datagrams, such as dial-up, DSL, cable modem, wireless or dedicated high-speed interconnects.

ISPs charge persons for a fee and provide the following services:

- Adding your computer to the ISP's network, allowing you to communicate with other computers on the Internet
- Giving your computer an IP address, allowing other computers on the Internet to communicate with you
- Providing you access to a DNS server.

Many ISPs provide other services like email and web hosting as part of their service, but these are not required to access the Internet. The following table shows different Types of Internet Connections:

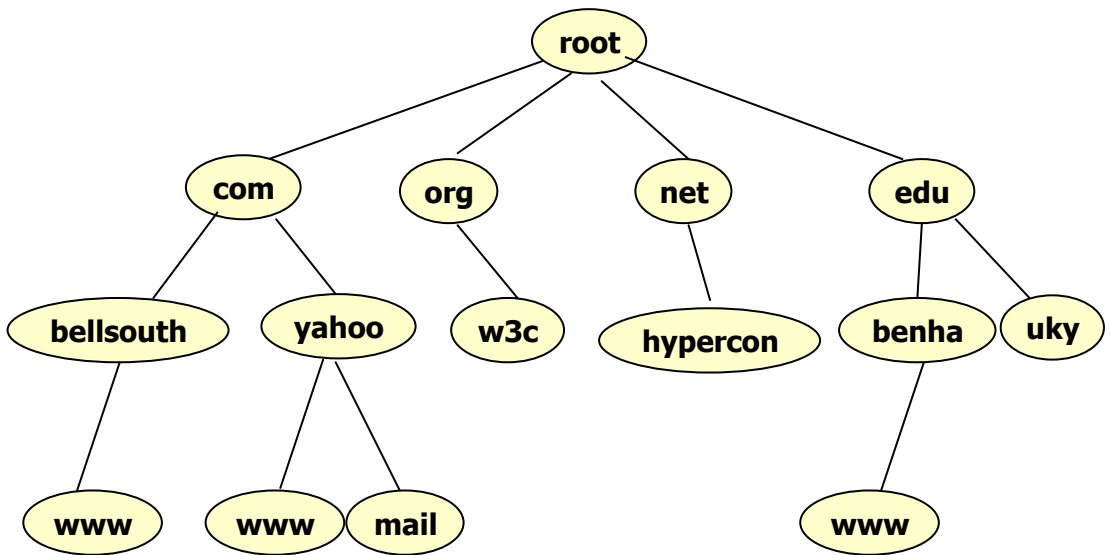
Connection Type	Cost	Speed	Hardware	Notes
Dial-up	Inexpensive (\$10-\$25 / mo)	Very slow	Modem; telephone line	
Digital Subscriber Line (DSL)	Moderate (\$30-\$50 / mo)	Fast	NIC card; dedicated phone line; DSL filter	Dedicated line; constant bandwidth

Cable	Moderate (\$30-\$60 / mo)	Fast	Cable “modem”; NIC card	Shared line; bandwidth fluctuates
T1	Expensive	Very fast	T1 line installed; NIC card	Businesses and institutions

So now that you’re connected to the Internet, how do you find the computer you want to communicate with? You may think you have the computer’s address, (e.g. www.yahoo.com) but what you actually have is its uniform resource locator (URL). To understand *how to get an IP address from a URL*, we need to understand the basic structure of the Internet.

The Structure of the Internet

Every computer connected to the Internet is in a certain zone. *Zones are divided into domains and subdomains*, making them **hierarchical**. At the top of the hierarchy is the **root**. All the computers on the Internet are in root’s zone. Root’s zone is divided into top-level domains (**TLDs**) such as TLDs: com, net, org, edu, gov, mil, and others. Each TLD is divided into **subdomains** Such as yahoo, google, benha, ebay, and many others. Subdomains can be further divided until they reach a **server**, the main computer in a given network. The following graph shows a visualized graph for the internet structure:



Before start explaining the steps followed to retrieve a certain page through the internet There are two important questions we need to answer, *Who Structuring the Internet?* And *who is controlling the internet?* For the first, Currently, **Internet structure** is provided by the following Entities:

- 1- **Networks** from corporations, commercial firms, and other companies
- 2- **Telephone** companies
- 3- **Cable** companies
- 4- **Satellite** companies
- 5- **Government**

For the second, generally speaking *there is no single entity controls or owns the Internet. The Internet is a public, cooperative, and independent network.* However, *there are several non-profit organizations that advise and define standards for internet* such as:

- 1- Internet Corporation for assigned names and numbers (**ICANN**). Its task is to manage the logistics of Internet Protocol (IP) addresses and domain names.
- 2- World Wide Web Consortium (**W3C**): Its task is to develop and tests advanced Internet technologies.

Uniform Resource Locators (URLs)

Now you are probably beginning to see how a URL is constructed, but let's take a closer look. Consider the URL for the following website:

<http://www.feng.benha.edu/staff/ahmed/index.html>

Where:

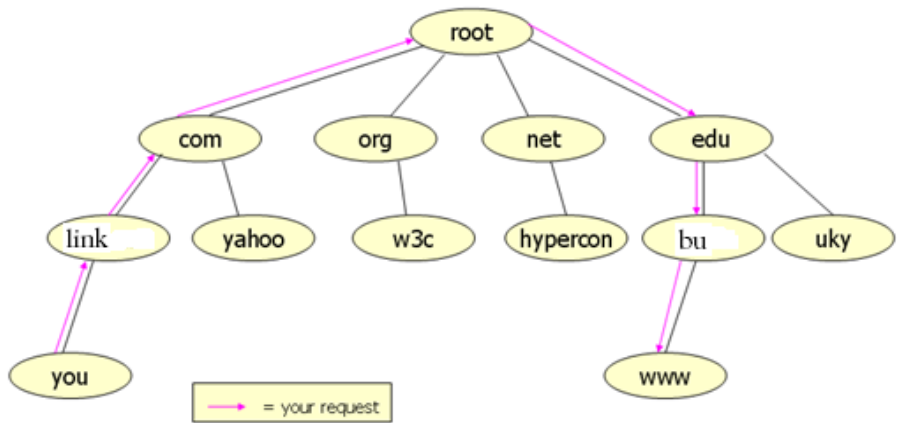
- **http** stands for *hypertext transfer protocol*, the *protocol for retrieving web pages*
- **www** is the name of the **server** (with a specific IP address)
- **feng** is a subdomain of **benha**
- **benha** is the subdomain of **edu**
- **edu** is the top level domain
- **/staff/ahmed/** is the name of the **folder** (or *directory*) on the server where the page is stored. This part of the URL is called the *path*.
- **index.html** is the name of the file we want to see

The Domain Name System (DNS)

As mentioned earlier, your ISP gives you access to one or more DNS servers. When you try to find a web page using a URL, you must first submit the URL to a DNS server so it can resolve the URL into an IP address. Let us look at the following DNS example:

- Suppose your ISP is BellSouth (www.link.com).
 - The DNS server provided by BellSouth is *authoritative* for any computer in the zone link. This means it knows the IP address of any server in the domain link or any of link's subdomains.
- If you try to reach www.bu.edu, link's DNS server does not know how to reach this server (i.e. it is *unauthoritative*) because bu is not in zone link.
- link's DNS server must ask the server above it (i.e. com's DNS server) to help. But bu is not in zone com either. (It is in zone edu.) Therefore, com must ask the server above it (root).
- Every computer is in zone root, but root cannot store all of the IP addresses on the Internet. It can, however, point com's DNS server to edu's DNS server, which it does.
- Now your request passes through edu's DNS to bu's DNS, which returns to you the IP address of the requested server (www).

The following figure shows visualization for the example:



The Client-Server Relationship

Once you have located the computer you wish to communicate with (the *remote host*), your computer (the *local host*) will attempt to establish a channel of communication between the two.

In most cases, *the local host is requesting information (such as the content of a web page) from the remote host*. We call the requesting computer the **client** and the responding computer the **server**; therefore, this type of relationship between the hosts is called a **client-server relationship**. Now you've got the page you want. So, what is it?

Web pages are constructed using a special language called hypertext markup language (HTML). HTML is text-only. So how are you able to see colors, links, and graphics?

Web pages are viewed using a program called a **browser**. The most common browser by far is **Microsoft's Internet Explorer**, which commands approximately 88% of the market. A *browser* is a program that renders the *HTML* page. This means it interprets the *HTML* text to determine how text, graphics, etc. should be displayed. The following shows an *HTML* example and its result when shown through the web browser:

What the browser sees...

What you see.....

```
<HTML>
  <BODY>
    <CENTER>
      <B> Hello, World! </B>
    </CENTER>
    <BR>
    <FONT COLOR="RED">
      This text is red.
    </FONT>
    <BR>
    <A
  HREF="www.feng.benha.edu">
      Go to feng.benha.edu.
    </A>
  </BODY>
</HTML>
```

Hello, World!
This text is red.
[Go to feng.benha.edu.](http://www.feng.benha.edu)

More on Browsers

Because it is up to the browser to decide how the *HTML* should be rendered, some pages look differently when viewed with different browsers. Some parts of *HTML* work in one browser, but not in another. *The World Wide Web Consortium (W3C)* has tried to

establish standards for HTML that all browsers should support, although browser makers (especially Microsoft) have bucked this trend. A good web page designer will test his or her page in several browsers before publishing it. As web content grows to include Flash animations, Java Applets, CGI scripts, and the like, these differences are likely to widen.

Main Elements of Web Browsers

Most web browser includes the several elements such as Title Bar, Scroll Bars, Status Bar, Menu Bar, Page Tab and Home Button. The following figure shows the following elements in Microsoft Internet Explorer and FireFox:

1- Microsoft Internet Explorer



2- FireFox



Other Web Browser Choices

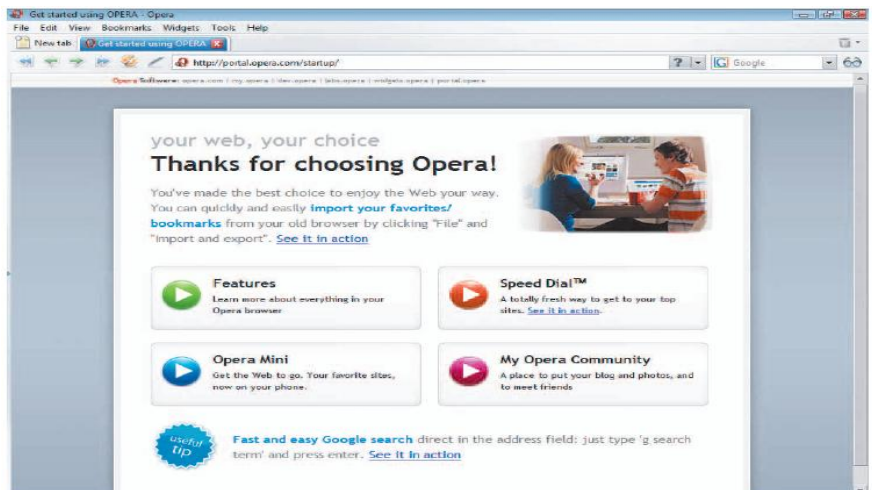
90o99090--..Dozens of innovative web 0 have been created by various people and teams over the years. The first widely used web browser was NCSA Mosaic. The Mosaic programming team then created the first commercial web browser called **Netscape** Navigator, later renamed Communicator, then renamed back to just Netscape. The Netscape browser led in user share until Microsoft Internet Explorer took the lead in 1999 due to its distribution advantage.

A free open source software version of Netscape was then developed called Mozilla, which was the internal name for the old Netscape browser, and released in 2002. Mozilla has since gained in market share, particularly on non-Windows platforms, largely

due to its open source foundation, and in 2004 was released in the quickly popular **FireFox** version.

In 1994, the **Opera** browser was developed by a team of researchers at a telecommunication company called Telenor in Oslo, Norway. The following year, two members of the team -- Jon Stephenson von Tetzchner and Geir Ivarsøy -- left Telenor to establish Opera Software to develop the browser commercially. Opera 2.1 was first made available on the Internet in the summer of 1996. Opera was First Web browser to offer:

- Tabbed browsing
- Button to toggle on and off the download of images with a Web page
- Search window that the user could configure to run searches in specific search engines automatically



Hyperlinks:

- The true power of HTML lies in the use of *hyperlinks*.
- Any text or graphic can be enhanced with a hyperlink. When a *hyperlinked object is clicked, the browser requests another resource (web page, Word document, etc.) either from the same server or from a completely different one.*
 - This allows quick and easy navigation between web pages.
- There are **three** ways to recognize a hyperlinked object.
 - Most browsers render hyperlinked text in **blue** and **underline** it. Hyperlinked graphics have a blue border.
 - The browser's status bar shows the URL of the resource linked to by the hyperlink.
 - The browser changes the cursor, usually to a **hand**.
- All three of these behaviors can be overridden by the page's author, but at least one usually isn't so the user can clearly tell which objects are hyperlinked.

Browser Cache:

In the interest of **speed**, *most browsers will cache web pages the first time they are visited. On subsequent visits, if the browser determines that the page has not been modified, it displays the cached copy instead of requesting the page from the server.* Occasionally, the browser does not detect a change that occurred. If

you do not believe you are seeing the most up-to-date information, click your browser's **Refresh** or Reload button to *force the browser to request the page from the server*. Some web sites appear to “remember” your preferences on subsequent visits. This is primarily accomplished in one of two ways.

1. A server-side database stores your preferences. When you visit the site, you enter a username and password, and your preferences are loaded from the database.
2. On your first visit, the site silently places a small text file (called a *cookie*) on your hard drive. This file contains your preferences and is automatically loaded by the site when you visit it again.
 - Use of cookies can be turned off in your browser, but many sites require that it be turned on in order to function.
 - Many privacy advocates are opposed to cookies.

Some browsers and third-party applications offer to remember your passwords. This is not tied in any way to the web site itself.

Search Engines

With such a vast quantity of information on the Internet, how can you find what you are looking for? There are many **sites** on the Internet called *search engines*.

1. At the minimum, a search engine allows the user to type in a word or phrase to search for, then returns results that it determines most closely match the user's request. The user's request is called a **query**. Each individual result is called a **hit**.
2. Some popular search engines are Yahoo, Google, AltaVista, and Ask Jeeves.

Search engines are constantly updating their "knowledge" of information on the Internet using *spiders*. *Spiders are programs that crawl the Internet (i.e. follow all possible links and reporting the information found there back to the search engine.)*. Some search engines do not use spiders. Instead, they *query many other search engines and combine the list of hits*. These engines are called **meta-search engines**. Some popular meta-search engines are Dogpile, MetaCrawler, and Excite.

Email

One of the most popular uses for the Internet is sending and receiving email. To send and receive email, you need an email account. This will include at least two things:

1. An email address of the form **username@domainname.tld**.
2. Mail storage where your incoming messages are stored. While most ISP's provide a small amount of storage for email (< 10 MB), many web-based mail services like Yahoo Mail, Hotmail, and GMail allow users 2 GB or more of storage. A person may have numerous email accounts.

There are two ways to access an email account.

1. With a mail client like Outlook, Eudora, or Mozilla Thunderbird. In this type of email, the software must be configured for each email account. Messages are downloaded to your local machine.
2. Through a web mail interface (if your email provider has one). In This kind of email, email accounts can be accessed from any computer connected to the Internet with no extra configuration. Also, Messages stay on the mail server.

File Transfer

Files can be transferred from one computer to another over the Internet using the file transfer protocol (FTP). The process of transferring a file from a remote host to your local machine is called *downloading*. The process of transferring a file from your local machine is called *uploading*. Most browsers natively support downloading, but uploading often requires a special utility called an FTP client. To download with an Internet browser, the following steps are followed:

1. Right click the link for the resource you want to download.
2. A context menu will appear. Choose Save Target As... (Internet Explorer) or Save Link As... (Firefox).
3. A save dialog will appear. Choose where you want to save file.
4. Change the filename if desired, then click Save

Client-Server Architecture

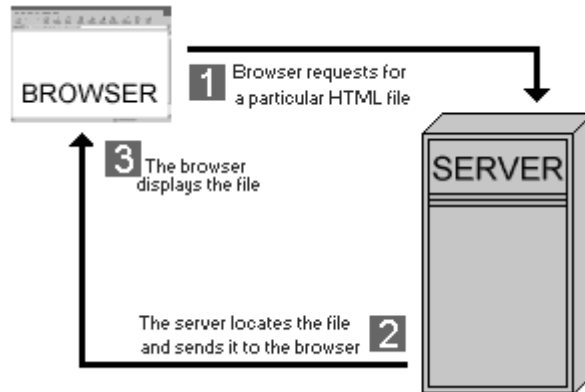
The client-server architecture is a way to structure a distributed application so that it consists of two distinct software modules:

1. A server module, only one instance of which is present in the system.
2. A client module, of which multiple instances are present in the system.

The only communication in the system is between the client modules and the server module. The client and server modules themselves may be quite complex systems with further submodules and components. However, the key characteristic of the client-server architecture is that there is a server module that is the central point for communication. Clients do not communicate with each other, only with the server module. In the client-server architecture, the server is usually the more complex piece of the software. The clients are often (although not always) simpler. With the wide availability of a web browser on most desktops, it is quite common to develop distributed applications so that they can use a standard web browser as the client. In this case, no effort is needed to develop or maintain the client (or, rather, the effort has been taken over by a third party—the developer of the web browser). This simplifies the task of maintaining and upgrading the application software. In any distributed application, the different components must discover each other in order to communicate. In the client-server architecture, only the clients need to communicate with the server. Therefore, each client needs to discover the network address of the server, and the server needs to know the network address of each of the clients. The solution used for discovery in

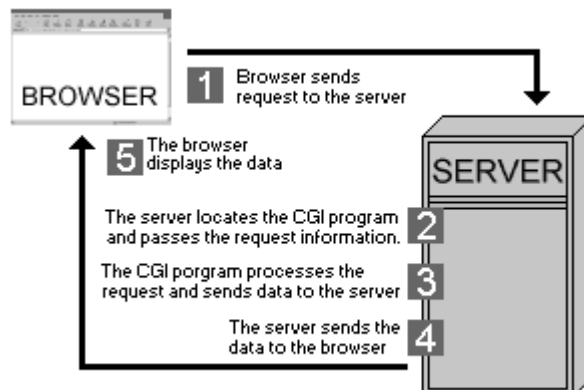
the client-server architecture is quite simple. The server runs on a port and network address that is known to the client module. The clients connect to the server on this well-known network address. Once the client connects to the server, the client and server are able to communicate with each other. The server need not be configured with any information about the clients. This implies that the same server module can communicate with any number of clients, constrained only by the physical resources needed to provide a reasonable response time to all of the connected clients. For most common applications that run on the Internet, the port numbers on which the server side can run have been standardized. Thus the clients only need to know the IP address of the computer on which the server is running. Any individual client can also easily switch to another server module by using the IP address (or, in general, the IP address and the port number) of the new server. As an example, a web server typically runs on port 80 and web browsers can connect to the web server when a user specifies the name of the computer running the web server. The browser also has the option of connecting to a server running on a port different than 80. The simplicity and ease of maintenance of client-server architecture are the key reasons for its widespread usage in the design of distributed applications at the present time. However, the client-server architecture has one drawback—It does not utilize the computing power of the computers running the client modules as effectively as it does the computing power of the server module. Let us now examine the client-server inter-communication with three models:

Model #1 of the client-server architecture - Static HTML pages



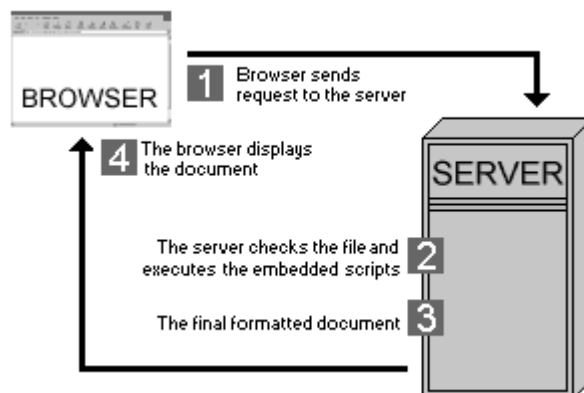
The client (browser) requests for an HTML file stored on the remote machine through the server software. The server locates this file and passes it to the client. The client then displays this file on your machine. In this case, the HTML page is static. Static pages do not change until the developer modifies them.

Model #2 of the client-server architecture – CGI Scripts



The scenario is slightly different for CGI applications. Here the server has to do more work since CGI programs consume the server machine's processing power. Let us suppose you come across a searchable form on a web page that runs a CGI program. Let us also suppose you type in the word 'computers' as the search query. Your browser sends your request to the server. The server checks the headers and locates the necessary CGI program and passes it the data from the request including your search query "computers". The CGI program processes this data and returns the results to the server. The server then sends this formatted in HTML to your browser which in turn displays the HTML page. Thus the CGI program generates a dynamic HTML page. The contents of the dynamic page depend on the query passed to the CGI program.

Model #3 of the client-server architecture - Server side scripting technologies



The third case also involves dynamic response generated by the use of server side technologies. There are many server side technologies today.

Active Server Pages (ASP): A Microsoft technology. ASP pages typically have the extension .asp.

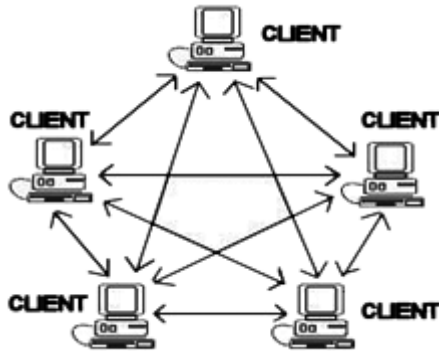
Personal Home Pages (PHP): An open source technology. PHP pages typically have .php, .phtml or .php3 file name extensions.

Java Server Pages: .jsp pages contain Java code.

With these server technologies it has become easier to maintain Web pages especially helpful for a large web site. The developer needs to embed the server-side language code inside the HTML page. This code is passed to the appropriate interpreter which processes these instructions and generates the final HTML displayed by the browser. Note, the embedded server-script code is not visible to the client (even if you check the source of the page) as the server sends ONLY the HTML code. Let's look at PHP as an example. A request sent for a PHP page from a client is passed to the PHP interpreter by the server along with various program variables. The interpreter then processes the PHP code and generates a dynamic HTML output. This is sent to the server which in turn redirects it to the client. The browser is not aware of the functioning of the server. It just receives the HTML code, which it appropriately formats and displays on your computer.

Peer-to-Peer Architecture

The peer-to-peer architecture is a way to structure a distributed application so that it consists of many identical software modules, each module running on a different computer. The different software modules communicate with each other to complete the processing required for the completion of the distributed application. One could view the peer-to-peer architecture as placing a server module as well as a client module on each computer. Thus each computer can access services from the software modules on another computer, as well as providing services to the other computer. However, it also implies that the discovery process in the peer-to-peer architecture is much more complicated than that of the client-server architecture. Each computer would need to know the network addresses of the other computers running the distributed application, or at least of that subset of computers with which it may need to communicate. Furthermore, propagating changes to the different software modules on all the different computers would also be much harder. However, the combined processing power of several large computers could easily surpass the processing power available from even the best single computer, and the peer-to-peer architecture could thus result in much more scalable applications.



Types of P2P Networks

Peer-to-peer networks come in three forms. The category classification is based on the network and application.

1. Collaborative Computing:

Also referred to as distributed computing, it combines the idle or unused CPU processing power and/or free disk space of many computers in the network. Collaborative computing is most popular with science and biotech organizations where intense computer processing is required. Examples of distributed computing can be found at GRID.ORG where United Devices is hosting virtual screening for cancer research on the Grid MP platform. This project has evolved into the largest computational chemistry project in history. United Devices has harnessed the power of more than 2,000,000 PCs around the world to generate more than 100 teraflops of power. Most distributed computing networks are created by users volunteering their unused computing resources to contribute to public interest research projects.

2. Instant Messaging

One very common form of P2P networking is Instant Messaging (IM) where software applications, such as MSN Messenger or AOL Instant Messenger, for example, allow users to chat via text messages in real-time. While most vendors offer a free version of their IM software others have begun to focus on enterprise versions of IM software as business and corporations have moved towards implementing IM as a standard communications tool for business.

3. Affinity Communities

Affinity communities is the group of P2P networks that is based around file-sharing and became widely known and talked about due to the public legal issues surrounding the direct file sharing group, Napster. Affinity Communities are based on users collaborating and searching other user's computers for information and files.

How Peer-to-peer File-sharing Clients Work

Once you have downloaded and installed a P2P client, if you are connected to the Internet you can launch the utility and you are then logged into a central indexing server. This central server indexes all users who are currently online connected to the server. This server does not host any files for downloading. The P2P client will contain an area where you can search for a specific file. The utility queries the index server to find other connected users with the file you are looking for. When a match is found the central

server will tell you where to find the requested file. You can then choose a result from the search query and your utility when then attempt to establish a connection with the computer hosting the file you have requested. If a successful connection is made, you will begin downloading the file. Once the file download is complete the connection will be broken. A second model of P2P clients works in the same way but without a central indexing server. In this scenario the P2P software simply seeks out other Internet users using the same program and informs them of your presence online, building a large network of computers as more users install and use the software.

P2P Security Concerns

One major concern of using P2P architecture in the workplace is, of course, network security. Security concerns stem from the architecture itself. Today we find most blocking and routing handles by a specific server within network, but the P2P architecture has no single fixed server responsible for routing and requests. The first step in securing your P2P network is to adopt a strict usage policy within the workplace. In securing your network against attacks and viruses there are two main strategies where focus is on controlling the network access or the focus is put on controlling the files. A protocol-based approach is where system administrators use a software or hardware solution to watch for and block intrusive network traffic being received through the P2P clients. A second method of protection is a software solution which

would provide file surveillance to actively search for files based on their type, their name, their signature or even their content.

P2P at Work

P2P is not only popular with home users but many small business have come to rely on this cost-effective solution for sharing files with co-workers and clients. P2P promotes the ease of working together when you're not physically located in the same office. In just seconds updated files and data can be shared with peers and confidential files can be blocked for security. Additionally, companies can also block access to Internet music and video files to assist in maintaining a work-oriented P2P network. Not only does this keep the company free and clear from legal issues regarding music downloading and sharing but it also keeps the corporate bandwidth usage down.

Static vs. Dynamic pages

Static Pages:

There is a difference between Static and Dynamic Web pages. In Static pages, the contents (text/ links/ images) are the same each time it is delivered by the web server. Static pages are retrieved according the following steps:

Step 1 – Author writes an HTML Page

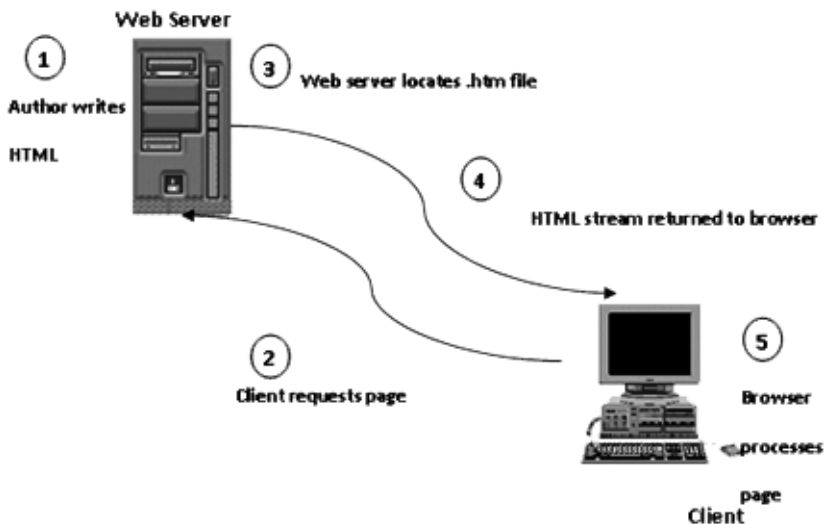
Step 2 – Client browser requests HTML page from web server

Step 3 – Web server locates the .htm file

Step 4 – Web server sends the HTML file back to the client browser

Step 5 – The browser processes the HTML web page.

Of course, Web server is the software that manages web pages and makes them available via a network.



Dynamic Pages

In Dynamic pages, the contents are:

- fluid, changeable (e.g., rotating banners)
- Able to react to the user's actions, request and process info, tailor services.

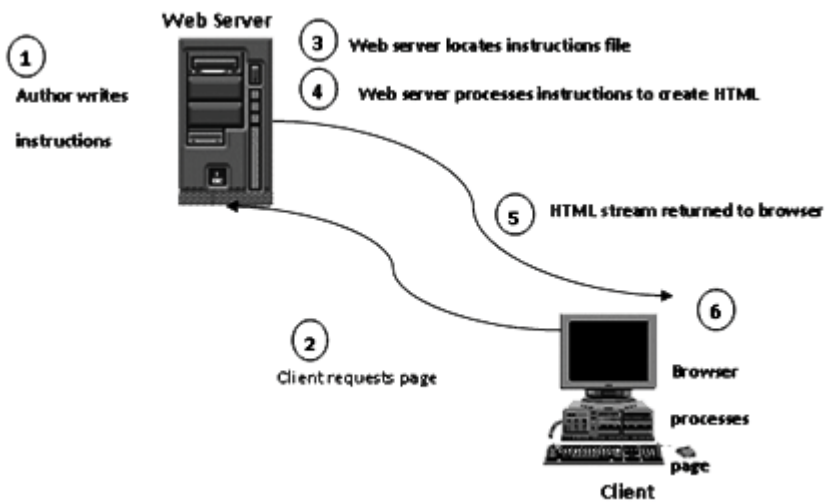
Computer processing for Dynamic pages can happen in 2 locations:

- Server side: Where server Accepts request, finds page and sends it
- Client side :Where client Gets HTML (or more?) from the internet, processes it then displays it

Server-side Dynamic Page Delivery

When server-side Dynamic page is delivered as follow:

- Step 1- Author writes instructions
- Step 2- Clients request pages
- Step 3- Web server locates instructions file
- Step 4- Web server processes instructions to create HTML
- Step 5- HTML stream returned to browser
- Step 6- Browser processes page



Client-side Dynamic Page Delivery

When client-side Dynamic page is delivered as followed:

- Step 1- Author writes instructions
- Step 2- Clients request pages
- Step 3- Web server locates instructions file
- Step 4- HTML and script are returned to browser
- Step 5- Web browser processes script to create HTML
- Step 6- Browser displays HTML

Types of Web programming

There are two main types of Web Programming, Client side and server side programming. Client-side programming is run on the user's computer such as JavaScript and Java applets. Those languages can be used to run checks on form values and send alerts to the user's browser.

Server-side Programming (or scripts) are run on the server such as **ASP (Active server page), JSP (Java server page), Java servlets and PHP**. Server-side programming can also be encrypted when users send form variables, protecting users against any hack attempts.

Exercise 1

Complete:

1. ----- is the series of local, regional, national, and international interconnected networks allowing communication of data among millions of computers worldwide.
2. ----- is the standard set of rules for electronically addressing and transmitting data over the Internet.
3. ----- is a non-profit Organization that is responsible for developing and testing advanced Internet technologies.
4. ----- is a Client-side programming language used to check on form values and send alerts to the user's browser.
5. -----, ----- and ----- are an Internet based software application.
6. Computer processing for Dynamic pages can happen in -----, -----:
7. ----- was established to Translate URLs to IP addresses

True / False

1. The Internet is owned and managed by the U.S. government.
2. Internet consists of a huge number mixture of DOS-based and Windows-based computers, Apple Macintosh computers, and UNIX workstations connected to each other.
3. The World Wide Web (WWW) is a collection of millions of Web Pages.
4. Hyperlink is a reference to another external document but not within the same document.

5. In Dynamic pages, the design is changeable while the contents are fixed.
6. Server-side programming can be encrypted to protect users against any hack attempts.

Answer the following:

1. What is the difference between LAN and WAN? How Internet is related to both?
2. Define the following two terms Web caching. And cookies.
3. Given the following hyperlink show in steps how DNS is used to retrieve the requested file. Use graphs to show your answer:

[www.eelu .edu.eg/wcl/portal/index.html](http://www.eelu.edu.eg/wcl/portal/index.html)

4. Explain the basic difference between peer to peer and client server architecture.
5. To access the Internet, you need a computer, a browser program, and an ISP. What is an ISP and what does it do?

Bonus Question:

- 1) Prepare a power point presentation that describe the following:
 - a) Apache(“A Patchy” Server)
 - b) Mail server

Presentation should be No more than 10 pages.

Chapter 2: Basic Web Applications

Applications of the Internet are numerous. They span all aspects of life from communicating and socializing with others, to searching for information, to reading newspapers, to taking courses and degrees, to shopping even for groceries, to seeking fatwa or medical or personal advice, etc. We are going to cover the most important ones that made the Internet so popular and important.

In this chapter, we will learn about electronic mail or email, search engines, electronic commerce or e-Commerce, e-Learning, peer-to-peer applications and briefly we will point to a few other applications.

Email

Now let us start by talking about one of the most popular Internet applications, the Electronic Mail or Email. Email, as the name suggests, is a form of exchanging mail messages electronically, using the Internet. Every day, the citizens of the Internet world exchange billions of emails. It is perhaps the Internet's 'killer' application as more people use email services than browse the web. Email has become the way to communicate efficiently, quickly and cheaply. It is just as easy to send an email to the person sitting next to you, as it is to send an email to a hundred different people

around the world. In the following we will look at the parts and workings of the email system.

What is email?

An email is an electronic message transmitted over a network from one user to another. When the Internet was first introduced it didn't take long before 75% of all network traffic was email related. An email can be simply a few lines of text sent from one user to another, or include attachments such as pictures or documents. It can be a newsletter sent to subscribers daily detailing what's happening and when, or an encrypted message sent between government organizations containing classified information. Email has many uses and it is now becoming a mature, feature-rich method of communication that appears complicated, but has an underlying technology that is very simple.

What makes up an email?

The actual email itself is organized into parts called the header and the body. The header contains information about

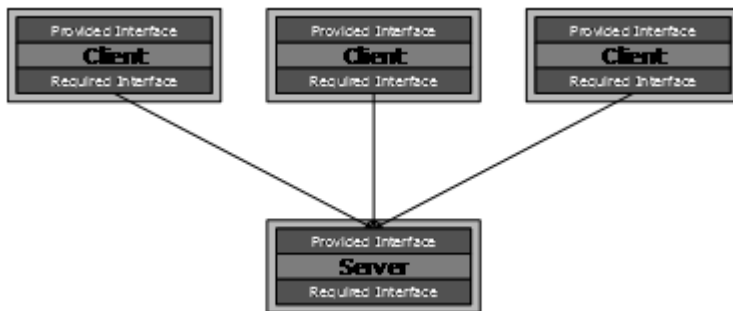
- 1- The sender and the receiver
- 2- The sent time
- 3- The subject and size of the message.

This is the part that is normally displayed to users before they open the full email. When the email is opened it displays the body as well as the header – the body contains the actual message sent.

Attachments are often sent with emails and these are normally displayed as icons within the email that relate to the attached document's type. Alternatively, they may be embedded into the message body.

How does email work?

Now, let us talk about how it works. Email is based on a client-server model where the client carries out the user's interactions with the email server and the server provides the processing capability.



Clients can appear in various forms:

1. Application based – these are installed onto users' machines and include Microsoft Outlook and the freely available Outlook Express and Eudora.
2. Web based – these appear in a web browser's window and include Hotmail, Yahoo and Outlook web client.

Clients vary greatly in functionality, but all provide a basic level of functionality that assists the user. Basic functions include:

- Ability to create new emails.
- Display and store received emails.
- Hold address lists of contacts, a calendar, journal and other extra functions that help organize the user's working day.
- The client is also configured with the account information and names or IP addresses of the email servers with which it will be communicating.

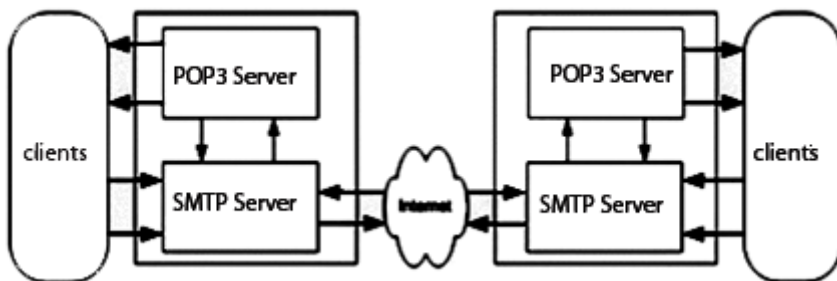
The email server is typically a combination of processes running on a server with a large storage capacity. This storage capacity is used to store several things such as the list of users and rules, and the capability to receive, send and store emails and attachments. These servers are designed to operate without constant user intervention and should process emails for months as sending, receiving and maintenance tasks are carried out at scheduled times. The client only has to connect to the email server when it sends and checks/receives new email. Sometimes it may be permanently connected to the server to allow access to shared address books or calendar information – this is typical of a LAN-based email server.

How do email servers work?

Most email servers conduct email services by running two separate processes on the same machine.

- 1- One process is the POP3 (Post Office protocol 3) server which holds emails in a queue and delivers emails to the client when they are requested.
- 2- The other is the SMTP (simple mail transfer protocol) server that receives outgoing emails from clients and sends and receives email from other SMTP servers.

These two processes are linked by an internal mail delivery mechanism that moves mail between the POP3 and SMTP servers. When the client calls the email server to send or check for mail it connects to the server on certain TCP/IP ports (see below): SMTP on port 25 to send email and when it checks for new email, POP3 on port 110. Figure 1 shows the relationship between the clients, servers, POP3, SMTP and the internet.

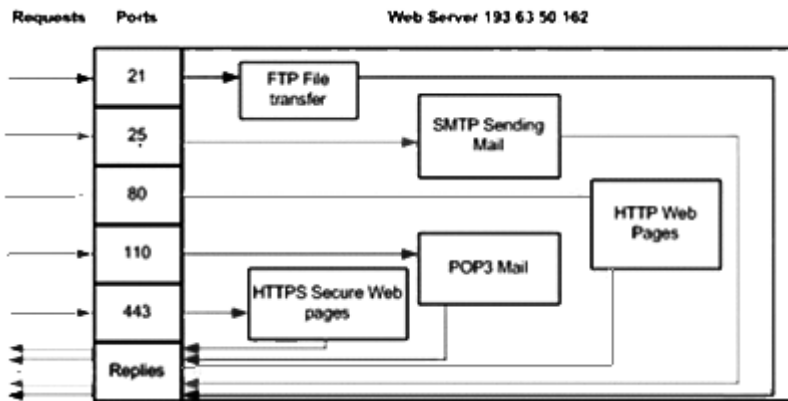


What are TCP/IP ports and how are they related to email?

Most email servers run on a webserver platform with email services installed and possibly other services as well. Each server has one or more unique TCP/IP (transmission control protocol/internet protocol) addresses. Attached to all TCP/IP

addresses are many ports that range from 0 to 65,535. TCP/IP uses ports to allocate different jobs to different services. The server will listen for a client or application to call it on a port and direct traffic from that port to the required service.

Figure 2 shows an example of a web server with the TCP/IP address of 193.63.50.162. It carries out many different tasks such as website hosting and FTP (file transfer protocol) as well as email. As each request is made from clients requiring information, they are made to a specific port that is allocated for that task. TCP/IP uses different ports for different requests so that it can easily identify what process each request is for.



These TCP/IP ports are used for all kinds of internet jobs: port 80 for serving browsers with web pages and port 21 for file transfers to be routed to the FTP application. Emails usually use port 25 for SMTP and port 110 for POP3.

What makes up an email address?

An email address has two parts: the email account (which is the actual user's account) and the domain name (which is the internet registered name for the organisation). These two parts are separated by the @ (pronounced 'at') symbol. The following example shows a user called Ahmed Mohamed with an account on the yahoo domain. The email account name will normally be set out in a certain format, the most popular being 'forename.lastname' or 'forename_lastname'. This is followed by the @ symbol to show the end of the email account and the start of the domain name.

Ahmed.Mohamed@yahoo.com

The domain name is usually the company's registered address on the Internet and placing 'www.' before the domain name should give the organizations' website. Email addresses never have spaces in them.

How does an email get transmitted from client to client?

Here is a simplified version of what takes place in the transmission of an email from one client user to another: A sender called Nader Samy works as a teacher at Manaret Alfarouk School in New Cairo. His email address is Nader.Samy@ManaretAlfarouk.edu.eg. He wishes to send an email to Salma Far who works at Etawfeqia School. Nader has Salma's email address on a business card. Nader starts his email client and selects 'Compose new message'. Ahmed types Salma's email address 'Salma_Ahmed@Eltawfiqia.edu.eg' into the 'To' box, and chooses not to 'CC' (carbon copy) or 'BCC'

(blind carbon copy) the message to any other users. He types in his subject as 'Computer lessons'. He then proceeds to type the rest of the message into the main body: 'Hi Salma, can you send me some information on Computer lessons at your school? Thanks, Nader'. As emails are fairly informal, Nader decides this is enough text and clicks the 'Send' button. Now the automated processes take over.

The client makes a connection to its local email server 'mail.ManaretAlfarouk.edu.eg, using SMTP port 25. The client passes the email's sender and recipient addresses along with the main body of the message to the email server.

When the server has received the email it looks at the domain part of the recipient's address, the part after the @ symbol – 'ManaretAlfarouk.edu.eg,'. If the domain is local to the server – 'Eltawfiqia.edu.eg' – it will simply pass the message onto the local POP3 server for delivery. As the recipient's domain is not local to the email server, it passes it onto the email server that looks after that specific domain.

It does this by asking another server or service called DNS (domain names server) for the IP (internet protocol) address or MX (mail exchange) record of the email server that deals with email for 'ManaretAlfarouk.edu.eg'. The DNS replies with an IP address for the other domain's SMTP server. Once the email server has this IP address, it opens up a connection to the recipient's SMTP server on port 25 and passes the message on.

Once the receiving SMTP server has the email, it checks the domain part of the email address recognizes it as its own local domain and passes it internally to the POP3 server. The POP3 server holds the message in Salma's mailbox. The POP3 mailbox holds the messages in a queue until the user, in this case Salma Ahmed, connects to the email server on port 110 to collect mail. She downloads his queued POP3 mailbox from the server to the client and receives Nader's email.

What should we look for in an email server?

Email servers come with many features that are designed to keep the system running and reduce maintenance time for administrators. A typical email server should include or provide:

- Full support for POP3, SMTP and web-based clients Web access to mail (Webmail) is becoming more important and allows users to access their email with only an internet-connected web browser. This is especially useful for offsite access.
- Some form of filtering Third party software companies can provide a more comprehensive solution, but the server should have some inbuilt filtering functionality.
- Support for multiple antivirus products this allows for unforeseen changes in antivirus software suppliers and the installation of multiple antivirus scanners on an email server.

- Comprehensive support, logging, monitoring and administration features Support and administration from remote locations is extremely useful when sites do not have a permanent support presence.
- Multiple domain support most establishments have, or will have, more than one domain.
- Public folders and shared address books for all users to access.
- Auto-responder support for users that are away for a period of time.
- Relay control to prevent the server from being used by unauthorized users, such as spammers.
- Easy backup and restore tasks.

How does an email system send attachments?

SMTP can only transmit text – this creates a problem when it comes to sending images, video and other attachments via email. SMTP gets around this problem by using two different methods.

One method is a program called UUencode. UUencode assumes that the file or attachment contains binary information (1s and 0s). It converts this binary information into text using a simple mathematic equation, almost a type of encryption. Once UUencode has converted an attachment into text, it can travel via SMTP.

The UUencoded file begins with a header that tells the receiving system the original filename. The receiving system's UUdecoding

software sees the header and translates the data that follows until it reaches the end marker. This translation results in the file being converted back into its original form.

UUencoding does have problems and there are various formats available, some of which are unable to handle multiple attachments.

An alternative to UUencode is MIME (multi-purpose internet mail extensions). MIME was developed because of the need for a system to translate the array of constantly changing attachment formats. MIME works in a similar way to Uuencode, but creates a header that it wraps around each encoded attachment. This allows it to encode images and sound.

The choice between using UUencode or MIME is normally dictated by the sending email server. Most modern servers use MIME to send their mail, however almost all servers still carry support for UUencode to allow for backwards compatibility.

Issues related to Email process:

Till now we talked about two types of email protocols, POP3 and SMTP. What about other protocols. IMAP (Interactive Mail Access Protocol) is another protocol that used to access e-mail. IMAP is the better option when you need to check your e-mail from multiple computers at different locations such as at work, home, or on the road. On the other hand protocol such as POP3 is useful for checking your e-mail from one computer at a single location.

Let us make a quick comparison between POP3 and IMAP. if we start by the definition of each:

POP3 works by accessing the inbox on the mail server and then downloading the new messages to your computer. With this type of account you do not have to stay logged on to the Internet. You can log on when you want to receive and send new messages. Once your new messages have been downloaded to your computer you can log off to read them. This option is good when you connect to the internet through a dial up service and are charged for your connection time. IMAP lets you view the headers of the new messages on the server and then retrieves the message you want to read when you click on them. When using IMAP, the mail is stored on the mail server. Unless you copy a message to a "Local Folder" the messages are never copied to your PC. Using this protocol, all your mail stays on the server in multiple folders, some of which you have created. This enables you to connect to any computer and see all your mail and mail folders. In general, IMAP is great if you have a dedicated connection to the internet or you like to check your mail from various locations.

It is important to point out that POP3 can be set to leave your e-mail on the server instead of downloading to your computer. This feature enables you to check your e-mail from multiple locations just like IMAP. So why choose IMAP rather than POP3 with the leave mail on server setting? With the POP3 leave mail on server

setting only your e-mail messages are on the server, but with IMAP your e-mail folders are also on the server.

When should you use POP3?

- When you only check e-mail from one computer at a single location.
- You want to remove your e-mail from the mail server.
- You connect to the internet through dial up and are charged for connection time.

When should you use IMAP?

- When you need to check e-mail from multiple computers at multiple locations.
- You use Web mail such as Gmail, Yahoo or Hotmail.
- You need to preserve the different e-mail folders you have created.

E Commerce

E Commerce is one of the most important facets of the Internet to have emerged in the recent times. In its broadest definition, E Commerce is digitally enabled commercial transactions between and among organizations and individuals. Digitally enabled means, for the most part, transactions that occur over the Internet and World Wide Web(“Web”). Commercial transactions involve the exchange of value (e.g. money) across organizational or individual boundaries in return for products and services. Ecommerce and E Business are two different terms. E Business is a digitally enabled transactions and processes within a firm. Those transactions

involve Information Systems controlled by the firm. E Business Doesn't involves commercial transactions across organizational boundaries. So, **E-Commerce (EC)** is an emerging concept that describes the process of buying and selling or exchanging of products, services; and information via computer networks including the Internet. E-Commerce is not just buying and selling but it includes the following:

- Info sharing (Web catalogues, ads, communities)
- Ordering (e-mail, e-forms)
- Payment (traditional, credit cards, EDI, digital cash)
- Fulfillment (Web-site, e-mail, fax, phone)
- Service & support (Web notes, FAQs, bulletin boards, e-mail)

However, Depending on who is asked, electronic commerce (e-Commerce) has different definitions:

- From a communications perspective, EC is the delivery of information, products/services, or payments over telephone lines, computer networks, or any other electronic means.
- From a business process perspective, EC is the application of technology to-ward the automation of business transactions and work flow.
- From a communications perspective, EC is the delivery of information, products/services, or payments over telephone lines, computer networks, or any other electronic means.

- From a business process perspective, EC is the application of technology to-ward the automation of business transactions and work flow.
- From a service perspective, EC is a tool that addresses the desire of firms, consumers, and management to cut service costs while improving the quality of goods and increasing the speed of service delivery.
- From an online perspective, EC provides the capability of buying and selling products and information on the Internet and other online services.

Benefits and Limitations of E-Commerce

Like any technology, ecommerce has both benefits and limitations.

The benefits of Ecommerce can be summarized as follows:

1. Ecommerce allows people to carry out businesses without the barriers of time or distance. One can log on to the Internet at any point of time, be it day or night and purchase or sell anything one desires at a single click of the mouse.
2. The direct cost-of-sale for an order taken from a web site is lower than through traditional means (retail, paper based), as there is no human interaction during the on-line electronic purchase order process. Also, electronic selling virtually eliminates processing errors, as well as being faster and more convenient for the visitor.
3. Ecommerce is ideal for niche products. Customers for such products are usually few. But in the vast market place i.e. the Internet, even niche products could generate viable volumes.

4. Another important benefit of Ecommerce is that it is the cheapest means of doing business.

At the same time, ecommerce has several limitations. Limitations can be classified into technical and non technical. Technological limitations may include the following:

1. Lack of universal for standard, security and reliability.
2. Lack of bandwidth to support E-Commerce especially for m-commerce.
3. Software development for E-Commerce is still maturing.
4. Existing applications and database brings difficulty to integrate Internet with EC software.
5. Implementing E-Commerce require special servers to support it which may increase cost.

While Non-technological limitations include the following:

1. Legal issues to be resolve.
2. Products are not able to be tested first.
3. People lack in trust when trading with strangers.
4. FRAUD cases are increasing.

Unique Features of E-Commerce Technology

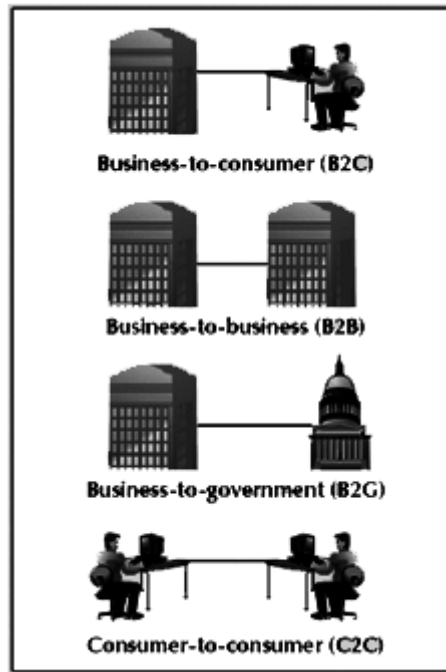
The features that set E-Commerce Technology apart from others used in traditional commerce are several such as:

1. Ubiquity – internet/web technology is available everywhere: at work, home and elsewhere via mobile devices.
 - ✓ Marketplace extended beyond traditional boundaries
 - ✓ “Marketspace” is created, available 24/7/365
 - ✓ Customer convenience increased, costs reduced.
2. Global Reach – the technology reaches across national boundaries, around the earth. Commerce enabled across cultural and national boundaries seamlessly.
 - ✓ Potential customer reach extended.
 - ✓ Reduces barriers to markets.
3. Universal standards – there is one set of technology standards, namely internet standards.
 - ✓ Promotes technology adoption
 - ✓ Reduces costs of adoption
4. Richness – Video, Audio, graphical and text messages are possible.
 - ✓ Integration to a more powerful marketing message and customer experience
5. Interactivity – the technology allows active user involvement.
 - ✓ Consumers engage in dynamic dialog
 - ✓ Experience adjusted to the individual based on responses.
 - ✓ Customer becomes co-participant in the process of delivering goods to the market.
6. Information Density - the technology reduces information costs and increase quantity and quality.

- ✓ Information processing, storage and communication costs drop dramatically.
 - ✓ Accuracy and timeliness improve greatly.
 - ✓ Information becomes plentiful, cheap and accurate.
7. Personalization/Customization – the technology reaches allows personalized messages to be delivered to individuals as well as groups.
 8. Commerce enabled across cultural and national boundaries seamlessly.
 - ✓ Potential customer reach extended.
 - ✓ Reduces barriers to markets.

There are several types of of Electronic Commerce:

- **Business-to-consumer or (B2C) electronic commerce:** process undertaken by a company to sell goods or services to individuals
- **Business-to-business or (B2B) electronic commerce:** process undertaken by a company to sell goods or services to other business firms or not-for-profit organizations
- **Business-to-government or (B2G):** using Internet technologies in business processes dealing with government agencies
- **Consumer-to-consumer or (C2C):** processes that occur when individuals who are not operating formal businesses use the Internet to conduct transactions



Steps to E-Commerce

Before you can start selling your goods and services online there are several steps that should be completed.

1. **Generating Demand:** Get people to your site, and then turn the lookers into buyers. Advertising on the internet is one way to get people to your site.
2. **Ordering & Fulfillment:** Once a consumer is at your site you need them to place an order by making sure the ordering process is simple and easy to use. Next, fill and ship your customers' orders in a timely manner and keep them informed of the progress of their order.
3. **Process Payment:** Choose one of the available methods for processing payments: Cash Model, Check Model or Credit Model.

4. **Service & Support:** Meeting the needs of your customers by providing exceptional customer support is imperative. Saving information about your customers can make it easier for them to place their next order. This information can also be used to offer them special discounts on the products they buy, enticing their return.
5. **Security:** Consumers should know that their transactions over the internet are secure. Using SSL (Secure Sockets Layer) and digital certificates provide the necessary security.
6. **ADVERTISING:** A big part of e-commerce is advertising. Attracting people to visit your site, and then convincing them to buy something while they're there.
7. **Banner Ads:** Banner ads have been used for many years and are the most popular form of advertising on the web. Most banner ads are a graphic of some type (.gif, .jpg or animated .gif) usually placed at the top, or along the side, of the web page.
8. **Banner Exchange:** Exchanging a banner ad with another web site is a way to reduce advertising costs.
9. **Referrer Sites:** The referrer programs usually send traffic in one direction. One site pays another site for the traffic sent their way. Barnes & Noble and Amazon are two businesses that offer referral services.
10. **Search Engine Placement:** Search engines collect, store and index URLs, Usenet, FTP and image files. Most search engines use an application (called robots, crawlers and spiders) to read all of the files on your site and store that information in their database.
11. **Spam E-Mail:** Spam e-mail is an unsolicited and unwanted e-mail sent to every e-mail address that a business can find or purchase. Spam e-mail is the equivalent of bulk junk mail.

12. Targeted E-Mail: Targeted e-mail is similar to spam e-mail because it's sent without permission. The difference with targeted email is the e-mails are sent only to people who match a certain demographic.
13. Opt-in E-Mail: Opt-in e-mail is the only good form of e-mail advertising. With this method, recipients have agreed to receive e-mails from the business.

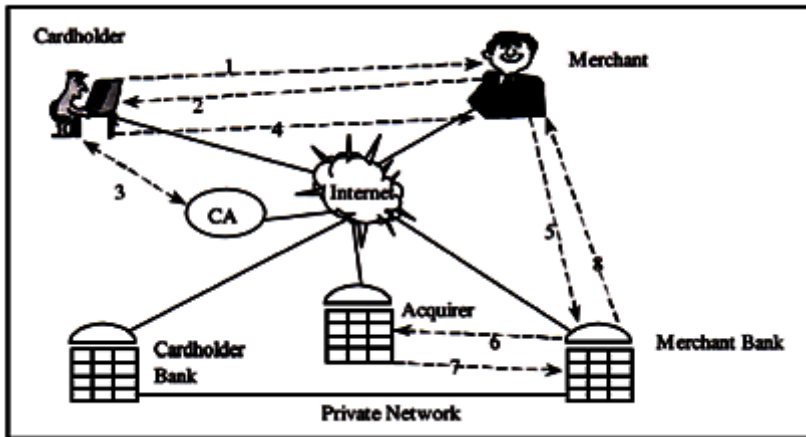
PAYMENT PROCESSING

As mentioned earlier, there are three different ways to process payments over the internet: Cash Model, Check Model and Credit Model

1. Cash Model: The Cash Model (e-cash) is the creation of electronic money or tokens, usually by a bank, which buyers and sellers trade for goods and services.
2. Check Model In using the Check Model, the consumer presents a digital version of their check to a web storefront.
3. Credit Model In the Credit Model the consumer enters their credit card information and the payment is processed by an accredited organization, usually a bank, using SET (Secure Electronic Transaction).

SET

SET, which stands for secure electronic transactions, is used with the credit model and is the most common method of payment over the internet since it uses the existing credit card processing system. The following figure shows different steps to perform SET protocol:



- Step 1.** The consumer enters an order along with their credit card information and sends it to the business.
- Step 2.** The business sends the consumer an invoice, their certificate and their bank's certificate.
- Step 3.** The consumer acknowledges and approves this information and returns it to the business.
- Step 4.** The business then generates an authorization request for your credit card and sends it to their bank.
- Step 5.** The business's bank then sends the credit authorization request to the Acquirer.
- Step 6.** The Acquirer sends an acknowledgement back to the business's bank after receiving an acknowledgement from the consumer's bank.
- Step 7.** Once the consumer's bank authorizes payment, the business's bank sends an acknowledgement back to the business with the authorization number.

Steps one through three happen while you are placing your order on the business's web site. Steps four through seven are the same steps taken when you purchase something at a store. These steps are just taking place over the internet instead of at a cash register. After selecting the type of payment processing to use, the next step is to setup an Internet Merchant Account. The Internet Merchant

Account needs to be setup with a financial institution that enables you to accept credit cards or purchase cards for payments over the internet

Web portal

What is a Portal?

In the world of science and space, a 'portal' is considered to be a two-way inter-dimensional door opening into several realities, including the astral world; the far reaches of physical, interstellar space; and alternate, parallel universes. To a certain degree, the Information technology world has high jacked the definition of a true portal and translated it into its own interpretation - but the principle of what a portal is stays the same.

What is the Information Technology (IT) definition of a Portal?

In the IT World - a portal is used to describe a browser experience that has an entry point (or gateway) that is intended to be the starting point for any user experience. As such in IT - a portal can be described as an 'anchor' or starting point that makes all the types of information (destinations) available to a designated audience by passing through the one point.

Within IT, a number of attempts have been made to classify portals - as they are significantly different in the way they split, each will be explored separately.

Information Portals vs Content Management Portals

In many organizations, it is not a case of choosing between one type or the other - and often both types are combined and deployed together in order to meet business needs.

Information Portals

These types of portal (also called Vertical Enterprise Portals - or Enterprise Information Portals) can be essentially seen as consolidating many different types of information from a multitude of sources onto a single screen or user experience. People who use an Information Portal typically are not or do not publish to it - or put another way - they are the consumers of the information prepared and published by others.

A typical Information Portal being used within a corporation or company might contain some of the following;

- Local Weather, News, Share Price information taken from content feeds such as RSS, XML.
- Access to email client, calendars, meeting room bookings, centrally stored documents and assets - or any type of central business application where viewing of items is required.
- Corporate information such as HR, events, programs, or any other cross company information.

- Reports - or forms that allow information to be requested - to assist with business choices.
- Access to smaller information portals that are not maintained centrally but have an access point via the primary information portal.

Portals can be a 'static' experience, in that the same interface can be provided as a starting point to all users - or they can be and are frequently personalized. For a more detailed explanation of how they can be personalized please view [What is Personalization?](#) - but in essence personalization allows a portal gateway or window to be modified so that the information presented to a user is tailored to their profile, chosen interests or clicking behavior.

Content Management Portals

These types of portal are designed to improve the access to and sharing of information stored within an organization. In a content management portal, self-service publishing features allow end users to post and share any kind of document, digital asset, record or Web content with other users, even those geographically dispersed (portals of this kind tend to be browser based to allow for access to be from anywhere an internet connection exists).

For example, consider a global software company consisting of developers, product managers, marketing, and sales all working at

locations across the world. Each has information they need to share with members local to themselves (both in terms of geographic location and also in terms of job function) as well as with others outside of these groups. In a Content Management Portal, most users will have the ability to add information to the portal and some users will have rights or authorizations to modify, delete, and expire information produced by others. Whereas an Information Portal is essentially a 'read only' experience - with a Content Management Portal users are able to publish, read, retrieve, modify, archive and delete content or information within the portal 'window'.

A typical Content Management Portal being used within a corporation or company might contain some of the following;

- the ability to check-in and check-out information which is 'in progress', so that users cannot overwrite each other's changes.
- Version control and Audit trail, so that successive versions of a particular item can be retained or overwritten and a track of who did what can be reported on.
- A security mechanism, so that content can be protected from unauthorized view or manipulation.
- Workflow, which establishes a process through which a document or request flows among users

Application Centric Portals vs Content Centric Portals

Another way of viewing Portals dispenses with the idea of Information and Content as a distinction and looks at it as Application Centric or Content Centric;

Application Centric Portals

This type of portal definition sees the function as one of tying together back-end systems to support users' application driven business processes. Users could be viewing the information as read only or able to create, modify, delete, expire information based on rights and permissions - but they are essentially using the portal to 'glue' together a number of applications into one view - so that rather than having to open a number of different applications to drive their business processes they are able to access them all from one point.

Content Centric Portals

This type of portal definition sees the function as one of obtaining information from a wide variety of sources and displaying that content to users in a way that is based upon users' roles and segmented information needs. In order to achieve this type of portal delivery, a personalization engine is often needed which is either intrinsic to the portal software - or an additional layer of software to compliment the portal tool. As with any type of

'personalized' experience - it can be extremely powerful if properly conducted, but have a worse impact than doing nothing if the business logic and profiles that are used to make the portal content centric are incorrect and/or delivering inappropriate information.

A content centric portal that is pulling in information from business applications such as Web Content Management System (WCMS), Document Management System (DMS), Digital asset management (DAM), Recruitment Management Software (RMS) and standard Desktop applications - and on the fly determining the most appropriate information to make available based on implicit and explicit personalization rules is arguably the utopia of what can be done with portal technology. This usually requires that all aspects that are being pulled into the gateway are based on open standards to allow the content to be delivered as a service without requiring extensive bespoke programming. A customer is more likely to find a default capability of this nature with ECMS vendors, given that they are in a position to want their products to work in this manner - and for other vendors to want to integrate with their products in the same way.

Vertical Enterprise Portals vs Horizontal Enterprise Portals

A lot of what is contained within a VEP or HEP has already been covered in the previous two definitions - but depending on your gateway you are intending to portalise it could be that this is a

more appropriate way of seeing the user experience. Unlike the other definitions, the Vertical or Horizontal Enterprise Portal does not seek to determine what or how the content is being managed that is making up the user experience - but sees the portals as meeting a niche experience - or meeting a very wide audience.

Vertical Enterprise Portals

Examples of this type of user experience would be MP3.com, pets.com or any other similar site where the portal or gateway is specific to an industry vertical or sector.

Horizontal Enterprise Portals

Examples of this type of user experience would be the Yahoo, AOL.com. They are also sometimes referred to as Mega Portals.

Search engines:

The good thing about the World Wide Web is that there are hundreds of millions of pages available, waiting to present information on an amazing variety of topics. The bad thing is that there are hundreds of millions of pages available, most of them titled according to the notion of their author, almost all of them sitting on servers with hidden names. To get information on those pages you need to use Internet search engine.

So, what is Internet search Engine?

Internet search engines are special sites on the Web that are designed to help people find information stored on other sites. There are differences in the ways various search engines work, but they all perform three basic tasks:



1. They search the Internet -- or select pieces of the Internet -- based on important words.
2. They keep an index of the words they find, and where they find them.
3. They allow users to look for words or combinations of words found in that index.

Early search engines held an index of a few hundred thousand pages and documents, and received maybe one or two thousand inquiries each day. Today, a top search engine will index hundreds of millions of pages, and respond to tens of millions of queries per day. In this section, we'll discuss how these major tasks are performed, and how Internet search engines put the pieces together in order to let people find the information they need on the Web.

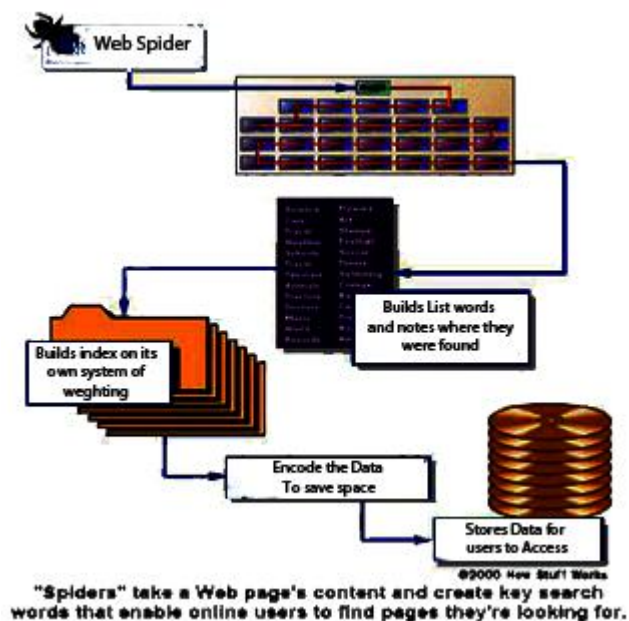
Looking at the Web

Before the Web became the most visible part of the Internet, there were already search engines in place to help people find information on the Net. Programs with names like "gopher" and "Archie" kept indexes of files stored on servers connected to the Internet, and dramatically reduced the amount of time required to

find programs and documents. In the late 1980s, getting serious value from the Internet meant knowing how to use gopher, Archie, Veronica and the rest. Today, most Internet users limit their searches to the Web and the contents of Web pages. Before a search engine can tell you where a file or document is, it must be found. To find information on the hundreds of millions of Web pages that exist, a search engine employs special software robots, called spiders, to build lists of the words found on Web sites. When a spider is building its lists, the process is called Web crawling. In order to build and maintain a useful list of words, a search engine's spiders have to look at a lot of pages. Spider software usually starts its travels over the Web by visiting lists of heavily used servers and very popular pages. The spider will begin with a popular site, indexing the words on its pages and following every link found within the site. In this way, the spidering system quickly begins to travel, spreading out across the most widely used portions of the Web.

Google.com began as an academic search engine. In the paper that describes how the system was built; the authors give an example of how quickly their spiders can work. They built their initial system to use multiple spiders, usually three at one time. Each spider could keep about 300 connections to Web pages open at a time. At its peak performance, using four spiders, their system could crawl over 100 pages per second, generating around 600 kilobytes of data each second. Keeping everything running quickly meant building a

system to feed necessary information to the spiders. The early Google system had a server dedicated to providing URLs to the spiders. Rather than depending on an Internet service provider for the domain name server (DNS) that translates a server's name into an address, Google had its own DNS, in order to minimize delays.



When the Google spider looked at an HTML page, it took note of two things:

1. The words within the page
2. Where the words were found

Words occurring in the title, subtitles, meta tags and other positions of relative importance were noted for special consideration during a subsequent user search. The Google spider was built to index

every significant word on a page, leaving out the articles "a," "an" and "the." Other spiders take different approaches.

These different approaches usually attempt to make the spider operate faster, allow users to search more efficiently, or both. For example, some spiders will keep track of the words in the title, subheadings and links, along with the 100 most frequently used words on the page and each word in the first 20 lines of text. Lycos is said to use this approach to spidering the Web.

Meta Tags

Meta tags allow the owner of a page to specify key words and concepts under which the page will be indexed. This can be helpful, especially in cases in which the words on the page might have double or triple meanings -- the Meta tags can guide the search engine in choosing which of the several possible meanings for these words is correct. There is, however, a danger in over-reliance on Meta tags, because a careless or unscrupulous page owner might add meta tags that fit very popular topics but have nothing to do with the actual contents of the page. To protect against this, spiders will correlate Meta tags with page content, rejecting the meta tags that don't match the words on the page. All of this assumes that the owner of a page actually wants it to be included in the results of a search engine's activities. Many times, the page's owner doesn't want it showing up on a major search engine, or doesn't want the activity of a spider accessing the page.

Consider, for example, a game that builds new, active pages each time sections of the page are displayed or new links are followed. If a Web spider accesses one of these pages, and begins following all of the links for new pages, the game could mistake the activity for a high-speed human player and spin out of control. To avoid situations like this, the robot exclusion protocol was developed. This protocol, implemented in the meta-tag section at the beginning of a Web page, tells a spider to leave the page alone -- to neither index the words on the page nor try to follow its links.

Building the Index

Once the spiders have completed the task of finding information on Web pages (and we should note that this is a task that is never actually completed -- the constantly changing nature of the Web means that the spiders are always crawling), the search engine must store the information in a way that makes it useful. There are two key components involved in making the gathered data accessible to users:

1. The information stored with the data
2. The method by which the information is indexed

In the simplest case, a search engine could just store the word and the URL where it was found. In reality, this would make for an engine of limited use, since there would be no way of telling whether the word was used in an important or a trivial way on the page, whether the word was used once or many times or whether

the page contained links to other pages containing the word. In other words, there would be no way of building the ranking list that tries to present the most useful pages at the top of the list of search results. To make for more useful results, most search engines store more than just the word and URL. An engine might store the number of times that the word appears on a page. The engine might assign a weight to each entry, with increasing values assigned to words as they appear near the top of the document, in sub-headings, in links, in the meta tags or in the title of the page. Each commercial search engine has a different formula for assigning weight to the words in its index. This is one of the reasons that a search for the same word on different search engines will produce different lists, with the pages presented in different orders.

Regardless of the precise combination of additional pieces of information stored by a search engine, the data will be encoded to save storage space. For example, the original Google paper describes using 2 bytes, of 8 bits each, to store information on weighting -- whether the word was capitalized, its font size, position, and other information to help in ranking the hit. Each factor might take up 2 or 3 bits within the 2-byte grouping (8 bits = 1 byte). As a result, a great deal of information can be stored in a very compact form. After the information is compacted, it's ready for indexing.

An index has a single purpose: It allows information to be found as quickly as possible. There are quite a few ways for an index to be

built, but one of the most effective ways is to build a hash table. In hashing, a formula is applied to attach a numerical value to each word. The formula is designed to evenly distribute the entries across a predetermined number of divisions. This numerical distribution is different from the distribution of words across the alphabet, and that is the key to a hash table's effectiveness.

In English, there are some letters that begin many words, while others begin fewer. You'll find, for example, that the "M" section of the dictionary is much thicker than the "X" section. This inequity means that finding a word beginning with a very "popular" letter could take much longer than finding a word that begins with a less popular one. Hashing evens out the difference, and reduces the average time it takes to find an entry. It also separates the index from the actual entry. The hash table contains the hashed number along with a pointer to the actual data, which can be sorted in whichever way allows it to be stored most efficiently. The combination of efficient indexing and effective storage makes it possible to get results quickly, even when the user creates a complicated search

Building a Search

Searching through an index involves a user building a query and submitting it through the search engine. The query can be quite simple, a single word at minimum. Building a more complex query requires the use of Boolean operators that allow you to refine and

extend the terms of the search. The Boolean operators most often seen are:

- **AND** - All the terms joined by "AND" must appear in the pages or documents. Some search engines substitute the operator "+" for the word AND.
- **OR** - At least one of the terms joined by "OR" must appear in the pages or documents.
- **NOT** - The term or terms following "NOT" must not appear in the pages or documents. Some search engines substitute the operator "-" for the word NOT.
- **FOLLOWED BY** - One of the terms must be directly followed by the other.
- **NEAR** - One of the terms must be within a specified number of words of the other.
- **Quotation Marks** - The words between the quotation marks are treated as a phrase, and that phrase must be found within the document or file

Example: When you search for Sport

Search engines have become such an integral part of our lives that at least one organized game has evolved around this tool. In Googlewhacking, you type two words into the Google search engine in the hopes of receiving exactly one result -- a single Web page on which both of those words appear. This is a pure whack. It's quite a difficult task -- you need to choose two completely unrelated words or else you'll get a whole lot more than one result, but with many completely unrelated words you get zero results. If you achieve a pure whack, you can submit it to www.googlewhack.com, where it is posted in The Whack Stack (along with your name, or whatever you want to call yourself) for

all to see. One pure whack currently in The Whack Stack is "ambidextrous scallywags."

E-Learning

E-Learning is the use of technology to enable people to learn anytime and anywhere. e-Learning includes the delivery of just-in-time information and guidance from experts. In the real world, people have jobs to do and budgets are limited. The traditional learning will need the power of technology to overcome the limitations of time, distance and resources.

As a general rule, different people need to learn in many different ways and at different times. To support these different learning needs, you will need different e-learning delivery methods. Additionally, you will need a way to develop and manage e-learning. The following show a list of e-learning delivery methodology:

1- Self-paced Courses

Self-paced courses are created with e-learning authoring tools.

Self-paced courses can be delivered in many ways including:

- Internet
- Intranet or Local Area Networks
- CD-ROM or DVD

Self-paced courses usually have these features:

- **Multimedia:** A mix of text, graphics, animation, audio and video to enhance the learning process

- Interactivity: An instructional strategy that helps a learner practice what they have learned
- Bookmarking: Lets the learner stop the course at any time and restart it from the same point
- Tracking: Report the learner's performance within a course to a Learning Management System (LMS)

Some self-paced courses have these advanced features:

- Simulation: Providing practice with a mock-up of a real system
- Online Experts: Provide access to experts through chat or online discussion
- Multiple Bookmarks: Designate one or more pages of the course to access while on the job
- Search: Search through a course to find information required to complete a task
- Notes and Highlights: Mark one or more parts of a course that contain the most important information

2- Discussion Groups

A discussion group is a collection of conversations that occur over time. Other names for discussion groups are message boards, bulletin boards and discussion forums. A discussion group might start out as a question from an individual. Some time later, another individual responds to that question. Others can respond to the question (creating a thread) or they can start their own conversation (forming another thread).

3- Virtual Classroom

A virtual classroom duplicates the capabilities found in a real classroom. A virtual classroom provides:<http://www.e-learningconsulting.com/consulting/images/webex-sample.gif>

- A place to meet: Students and teachers use their computers to go to a virtual meeting place instead of a classroom.
- Take attendance: A list of students is recorded.
- Lecture: Teachers can choose from a variety of synchronous technologies including:
 - Slide presentation
 - Audio and video conferencing
 - Application sharing
 - Shared whiteboard
- Interaction with students: Students can indicate when they want to speak by virtually raising their hand. Teachers can let students speak through audio and video conferencing. Teachers and students can use instant messaging and chat.
- Quizzes: Teachers can present questions to students.
- Breakout Sessions: Students can work together in groups.

Most companies that sell virtual classroom software provide all of these capabilities in a single package.

4- Audio and Video Conferencing

Audio conferencing can be implemented in two ways:

- Computers connected to the Internet. Common names for this kind of implementation are IP Audio Conferencing or Voice-over-IP.
- Phone conferences. People dial the same number to participate in an audio conference.

Video conferencing can also be implemented in two ways:

- Computers connected to the Internet. The computers need digital cameras.
- Special video conferencing devices that connect over the Internet or over phone lines.

5- Shared Whiteboard

A shared whiteboard lets a group of people communicate by typing comments, drawing, highlighting and pointing. A shared whiteboard is a common feature within virtual classroom software packages.

6- Application Sharing

You can demonstrate how to use software applications to remote learners with application sharing. A teacher can also let the learner take control of the application to practice performing tasks.

Exercise 2

- 1) For each of the following pairs of terms, define each term, making sure to clarify the key difference(s) between the two terms.
 - a. POP3 and IMAP
 - b. Information Portals vs Content Management Portals
 - c. E-learning and Traditional Learning
 - d. Email-server and Web server
- 2) *In search engines* over-reliance on Meta tags is sort of dangerous, explain?
- 3) Briefly, Explain different E-payment methods

True False

- 1) E-commerce and E-business are two alternative terms
- 2) Email domain name is the same as the internet registered name for the organization.
- 3) Email application use either peer to peer or client server architecture.
- 4) SET is used with the check model and is the most common method of payment over the internet
- 5) TCP/IP uses the same port to allocate different jobs to different services.
- 6) In search engines process, spiders start from the least to the most popular site,

Complete

- 1) E-Commerce is not just buying and selling but it includes another activities such as -----, -----, -----
- 2) MIME works in a similar way to Uuencode, but it has a feature of encoding -----image ----- and -----sound -----.
- 3) ----- is an ecommerce model used to use the Internet to conduct transactions between non operating formal businesses individuals.
- 4) ----- is an e-learning application that allows teacher or learner to take control of the application to practice performing tasks.
- 5) ----- is allows information to be found as quickly as possible.
- 6) ----- Portals are specific to an industry while ----- portals are for MEGA sites
- 7) To find information on the hundreds of millions of Web pages that exist, a search engine employs special software robots, called -----Spider -----.
- 8) A ___keyword ___ is a specific word that describes the information to be found in the search process.

Chapter 3 Internet and Web Security

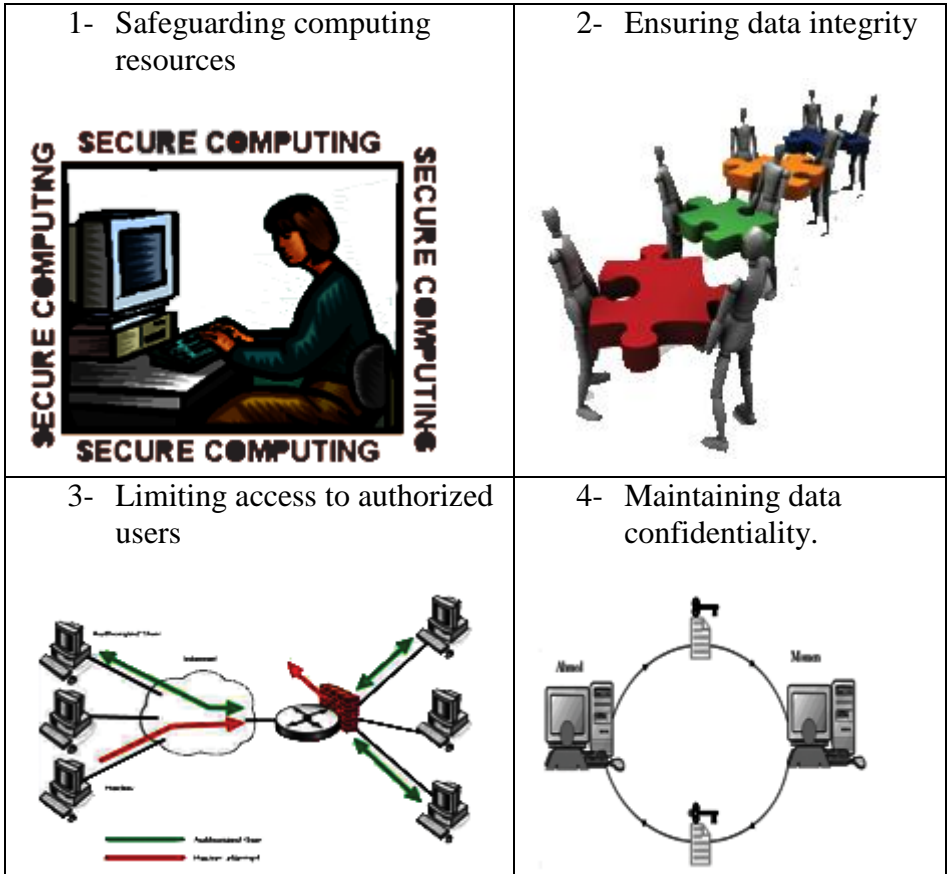
Security:

We live in a world where "information wants to be free" and in which people are getting used to having access to whatever information they want anytime, anywhere and from a wider and wider range of computing or non computing devices. Unfortunately, in terms of the security and control of the resources to which Humans or computers permit access; this can prove quite a problem. Indeed, many users unfortunately often view security and control measures as inhibitors to effective computer use. Security is broadly defined as the protection of assets from unauthorized access, use, alteration, or destruction



What about Computer Security?

Computer Security deals with the prevention and detection of unauthorised actions by users of a computer system. In other words, Computer security is broadly concerned with:



Now let us answer another question, **How Computer Security could be effective?**

For that to happen, Computer Security should involves the following items:

- 1- Taking Physical security measures (to ensure hardware and media are not stolen or damaged),
- 2- minimizing the risk and implications of error, failure or loss (for example by developing a resilient back-up strategy)

- 3- appropriate user authentication (for example by employing strong pass wording),
- 4- encryption of sensitive files

Information and computer security

The terms information security and computer security are frequently incorrectly used interchangeably. These fields are interrelated often and share the common goals of protecting the confidentiality, integrity and availability of information. Those three goals are combined on what is called CIA Triad

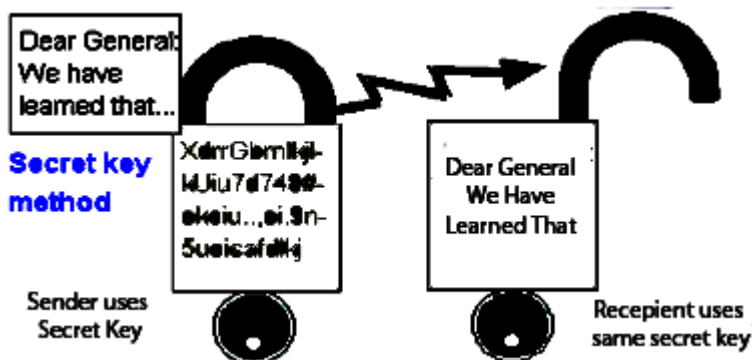


However, there are some subtle differences between them. These differences lie primarily in the approach to the subject, the methodologies used, and the areas of concentration. So, Information security is concerned with the confidentiality, integrity and availability of data regardless of the form the data may take: electronic, print, or other forms. While, Computer security can focus on ensuring the availability and correct operation of

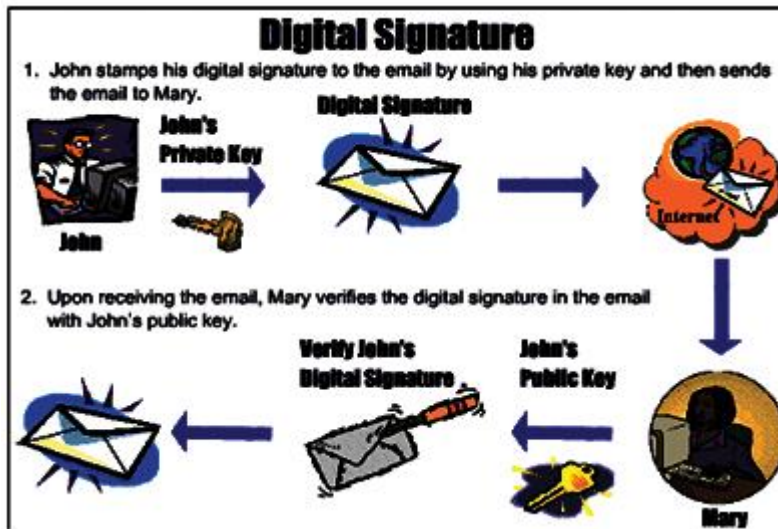
a computer system without concern for the information stored or processed by the computer.

Security Objectives

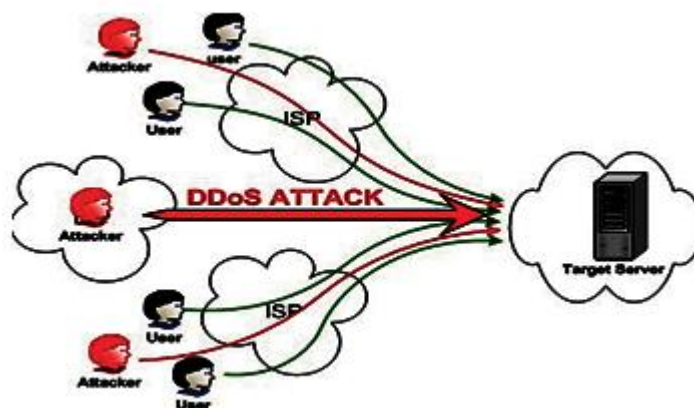
As we mentioned, CIA triad stands for Confidentiality, integrity and authentication. Those are the fundamental objectives of information security systems. Confidentiality is assurance of data privacy. Only the intended and authorized individuals, processes or devices, may read the data. Cryptography is the art and science of storing and transmitting confidential data where sender use secret key to transmit data and recipient uses same secret key to read the data.



Integrity is assurance of data non-alteration. Data integrity is having assurance that the information has not been altered in transmission, from origin to reception. Digital Signatures and hash algorithms are mechanisms used to provide data integrity.



Availability is assurance in the timely and reliable access to data services for authorized users. It ensures that information or resources are available when required. To achieve availability, make sure that you have denial of service controls (DoS) as well as intrusion detection systems. DoS attackers always try to Consume target server computational resources, such as bandwidth, disk space, or processor time.



Threats, vulnerability and risk mitigation

1) Threats

From the beginning security starts with that which we are trying to secure, namely the **asset**. The assets we secure on a computer might be:

- 1- important company secrets that could be valuable to a competing company
- 2- personal information such as the course grades of a university student
- 3- Money, since banks now process most money electronically.

The term Threat refers to a potential occurrence that can have an undesired effect on the system. Security systems are created for the purpose of protecting assets against threats. If the computer is threatened by viruses, then the security system may include such things as antivirus software, network firewalls and file protection mechanisms to guard against this threat. We talk later about all of those.



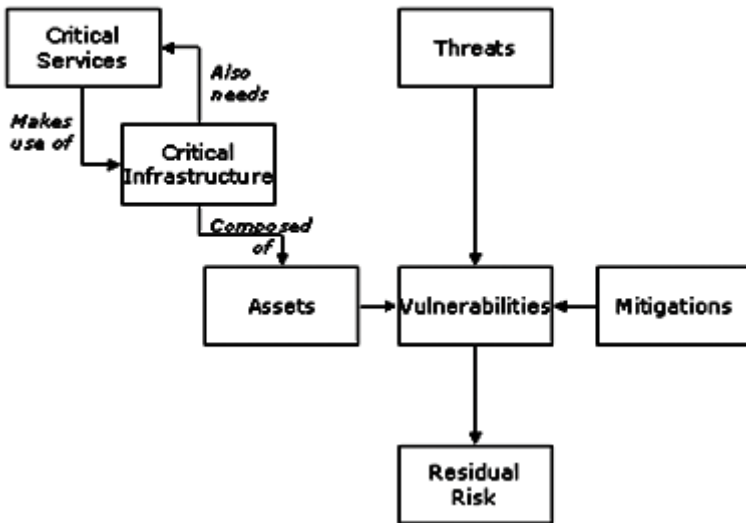
2) Vulnerability

Sometimes computer scientists confuse the term “threat” with “vulnerability”. The term: Vulnerability refers to the characteristics of the system that makes it possible for a threat to potentially occur. So, we can say that, a threat comes from outside the asset while vulnerability is a weakness within the security system intended to protect the asset.

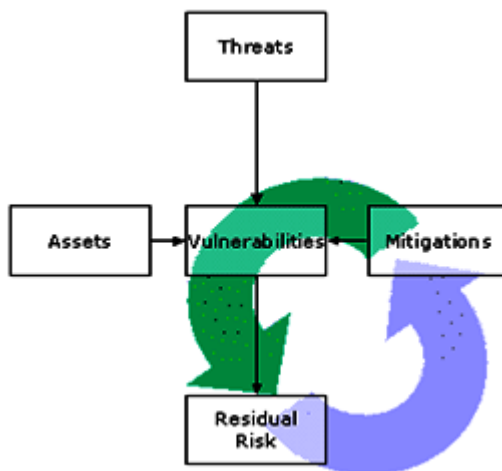
3) Risk mitigation

Any mechanism that is designed to guard against vulnerability is called mitigation. No system is ever completely secure, because every system has vulnerabilities. However, damage does not occur unless there is always an attack and risk. The term risk refers to the measure of the possibility of security breaches and severity of the damage. The following diagram shows the relation between threats, vulnerability, risks and mitigation. Risks come from critical services that depend on infrastructure, some areas of which themselves depend on other services. The components of the infrastructure, referred to as assets, are subject to vulnerabilities. Vulnerabilities may be exploited by threats. The action of a threat on vulnerability may be mitigated through various strategies.

Infrastructure Threats and Vulnerabilities



After risks have been mitigated there is always some residual risk, which needs to be assessed. If it is found unacceptable further mitigation measures will need to be applied. This is called risk mitigation cycle.

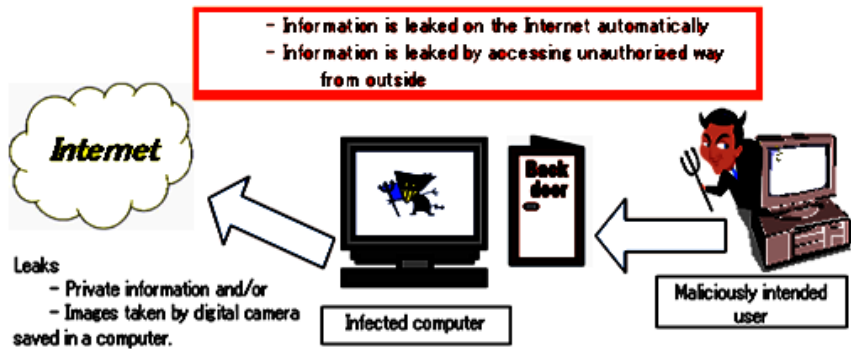


The term attack refers to the action of malicious intruder that exploits vulnerabilities of the system to cause a threat to occur in other words it is a deliberate attempt to exploit vulnerability. Threats, Vulnerabilities, attacks and risks multiply when a single computer is connected to the internet. For example, many people install wireless hubs in their homes without establishing security protocols and passwords. In such a situation anyone sitting within the transmission range of the wireless hub has unauthorized access.

Threats Types

The range of means by which the security and integrity of computing resources can be threatened and attacked is very broad, but here we can categorize threats and attacks to the following types:

- 1) **Information Disclosure threat** that refer to the dissemination of unauthorized information.
 - Information leakage is an examples of attacks considered with this kind of threats
 - Information Leakage happens when a web site reveals sensitive data, such as developer comments or error messages, which may aid an attacker in exploiting the system. Sensitive information may be present within HTML comments, error messages, source code, or simply left in plain sight.



- Leakage does not necessarily represent a breach in security but it does give an attacker useful guidance for future exploitation and may carry various levels of risk and should be limited whenever possible.

Information Disclosure threat Example:

Here we see a comment left by the development/QA personnel indicating what one should do if the image files do not show up. The security breach is the Host name of the server that is mentioned explicitly in the code, "VADER"..

```
<TABLE border="" cellPadding="" cellSpacing=""
height="59" width="591">
  <BODY>
    <TR>
      <!--If the image files are missing, restart VADET-- >
      <TD bgColor="#ffff" colSpan="5" height="17"
width="587">&nbsp;</TD>
    </TR>
```

2) **Integrity threat** – incorrect modification of information. As an Example of this threat is SQL Injection

- SQL Injection is an attack technique used to exploit web sites that Construct SQL statements from user-supplied input. Structured Query Language (SQL) is a specialized programming language for sending queries to databases. Most small and industrial strength database applications can be accessed using SQL statements. The impact of this threat can allow attackers to gain total control of the database or even execute commands on the system.

```
SQLQUERY= "SELECT Username FROM Users WHERE  
Username= '' & strUsername & '' AND Password = '' & strPassword  
& ''" strAuthCheck = GetQueryResult(SQLQUERY)
```

- In this code, the developer is taking the user-input from the form and embedding it directly into an SQL query. Suppose an attacker submits a login and password that looks like the
 - following:
 - *Login: ' OR ''='*
 - *Password: ' OR ''='*

```
SELECT Username FROM Users WHERE Username = '' OR ''='  
AND Password = '' OR ''='
```

This will cause the resulting SQL query to become Instead of comparing the user-supplied data with entries in the Users table, the query compares " (empty string) to " (empty string). This will

return a True result and the attacker will then be logged in as the first user in the Users table.

3) Denial of service (DoS) threat which happens when access to a system resource is blocked. For Example:

- Assume a Health-Care web site that generates a report with medical history. For each report request, the web site queries the database to fetch all records matching a single social security number. Given that hundreds of thousands of records are stored in the database (for all users), the user will need to wait three minutes to get their medical history report. During the three minutes of time, the database server's CPU reaches 60% utilization while searching for matching records.
- A common application layer DoS attack will send 10 simultaneous requests asking to generate a medical history report. These requests will most likely put the web site under a DoS-condition as the database server's CPU will reach 100% utilization. At this point the system will likely be inaccessible to normal user activity.
- **DoS targeting a specific user:** An intruder will repeatedly attempt to login to a web site as some user, purposely doing so within invalid password. This process will eventually lock out the user.
- **DoS targeting the Database server:** An intruder will use SQL injection techniques to modify the database so that the system

becomes unusable (e.g., deleting all data, deleting all usernames etc.)

Viruses, Worms and Spams

Today the Internet is a valuable source of information as well as a powerful communication medium, with undoubted social and economic benefits; however it also poses some security risks. Internet users are constantly threatened by the spread of new viruses hidden in appealing objects such as jokes, games, chats, and e-mails ostensibly sent by friends. Although Internet services such as e-mail and www represent the main "open doors", floppy and CD disks are still minor "contributors". The damage provoked by infections can be very costly for an organization's time and resources and becomes critical when it affects sensitive systems and data. A basic rule in computer security is to make frequent backups to avoid any kind of data destruction or corruption.

What is a Computer Virus?

A computer virus is a small software program that spreads from one computer to another computer and that interferes with computer operation. Computer viruses spread themselves by infecting executable files, system areas, hard or floppy disks, etc. without the user's knowledge. In addition to this self-replication activity viruses can perform 'bad' actions (payloads) ranging from mild disturbance (annoying the user with silly messages) to

provoking damage or outright disaster (e. g. deleting/corrupting files or performing a Denial of Service (DoS) attack).

How Computers infected with a virus

A computer is infected when a copy of the virus resides in the machine. Once the virus is loaded into the memory it can run in background and start to replicate itself. If the infected computer is on the network the infection will propagate very quickly to other machines. This process can be interrupted only by detection and elimination of the virus.

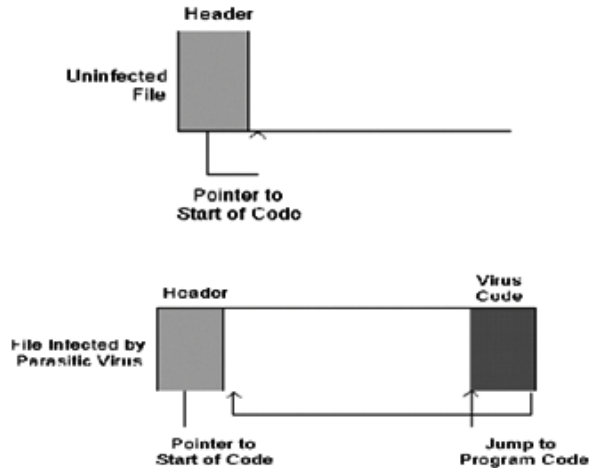
Viruses Types:

The best-known virus types include:

Parasitic:

- Parasitic viruses (or file viruses) are code fragments that reproduce by attaching themselves to executable files.
- When the user starts the infected program, the virus is launched first and then, in order to hide its presence, it triggers the original program to be opened.
- On a computer network this can be especially damaging because computer files are often given specific access permissions, sometimes referred to as “rights.”
- Networks assign parasitic viruses the same access rights as the infected file. Therefore, a high-level infected file may release a


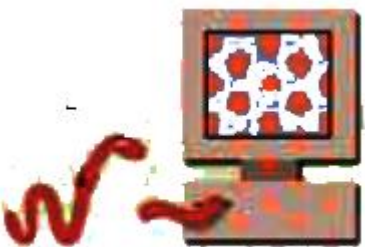
parasitic virus with enough rights to damage hundreds of thousands of other files before being detected and destroyed.

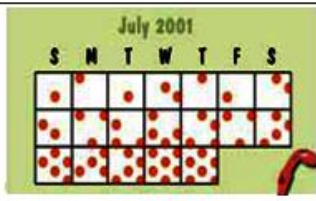
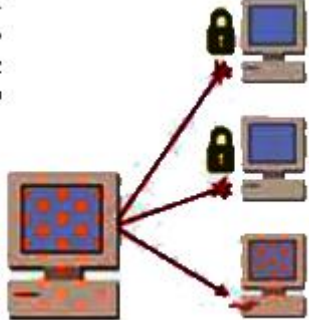

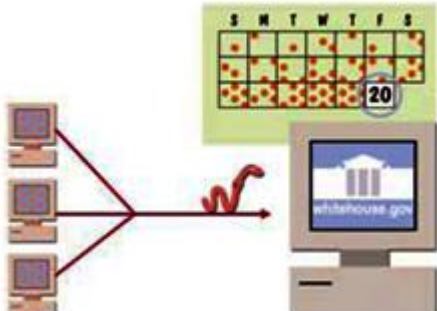


Worm:

- An Internet worm, unlike a virus, does not infect other program files but uses computer networks and takes advantages of SW bugs to replicate itself.
- A worm scans the network for another system having one specific security hole (such as buffer overflow), and copies itself into the new machine (via smtp, ftp, http, Internet Relay Chat, etc.)
- Then continues the self-replication process. Thus it has the ability to self-replicate incredibly quickly.
- Furthermore, the worm can release a payload such as scheduling a Distributed Denial of Service (DDoS) attack toward a target system or network.

- Due to the copying nature of a worm and its capability to travel across networks the end result in most cases is that the worm consumes too much system memory (or network bandwidth), causing Web servers, network servers and individual computers to stop responding.
- Code Red is a worm that was launched in 2001 as a DoS (denial of service) attack on the server administering a White House Web site (<http://www.whitehouse.gov>). the following animation show does it work.


<p>1- Code Red located systems running Windows NT 4.0, Windows 2000, or the beta version of Windows XP. Of these systems, Code Red selected those also running Microsoft IIS (Internet Information Server) 4.0 or 5.0. Code Red then replicated and attempted to exploit a vulnerability in the TCP/IP (Transmission Control Protocol/Internet Protocol) port 80 so it could enter and invade the system.</p>	
<p>2- If the Code Red worm attacked a system running WinNT, the system crashed. If Code Red attacked a system running either Win2000 or WinXP beta, the worm took advantage of the security flaw and automatically launched on the system.</p>	


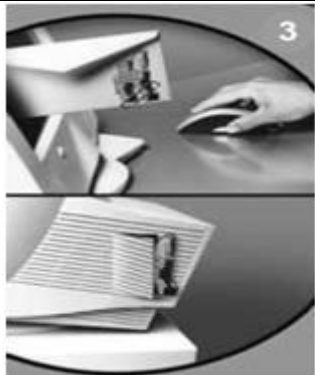

<p>3- For the first 19 days of July 2001, Code Red propagated. The most common way Code Red did this was by checking randomly generated IP addresses to find other vulnerable systems.</p>	
<p>4- After Code Red found other vulnerable systems, it replicated and launched itself on the new systems, where it would then search for other vulnerable computers to attack. Because of the massive amount of propagation, several infected systems faltered under the weight of the additional traffic. Many servers slowed to a crawl or shut down altogether.</p>	
<p>5- If Code Red infected a server that housed a Web site, it could also alter the content of HTML (Hypertext Markup Language) files. Some infected sites displayed the message “Welcome to http://www.worm.com! Hacked By Chinese!”</p>	
	

Beginning on the 20th day of the month, Code Red ceased propagation and shifted to attack mode. Infected computers flooded the White House server's IP address with numerous packets of data in an attempt to overload and bring down the site's server.

Trojan Horses:

A Trojan horse is defined as a malware. Malware is a program which hides malicious code disguised in appealing shapes i.e. it claims to do something "cool" or useful while actually provoking damages. Trojan horses are not designed to replicate automatically.

<p>1) Someone writes the code for the Trojan horse and disguises it as something other than a damaging program. The cracker planning the attack either posts the Trojan horse on a Web site for users to download or sends the Trojan horse to the intended victim as an email attachment. Many modern Trojan horses include a worm that sends an email message with the attached Trojan horse to different entries from a victim's address book.</p>	
<p>2) SubSeven is one of the more notorious Trojan horses. It might arrive as an email attachment and masquerade as a screen saver, a program update, or a graphics file. SubSeven also might lurk in programs users can download</p>	

<p>from the Web. In one incarnation of SubSeven, users downloaded a file called Quickflick.mpg.exe, which they</p> <p>believed was a pornographic video clip.</p>	
<p>3) When a user receives the SubSeven email message, he believes the attachment is a nice “gift” and double-clicks it, “inviting” the Trojan horse onto his computer system. The code in the SubSeven Trojan horse begins the process necessary for an attack, installing SubSeven server software on the system. This software then opens the system to a backdoor attack.</p>	
<p>4) A cracker with the necessary SubSeven server software locates the infected system. Thanks to the software the SubSeven Trojan horse installs, the cracker can access the user’s files and programs. For instance, the cracker’s motive might be to steal the user’s financial data. The cracker also might use the system as a zombie, launching DoS (denial of service) attacks on a large server or Web site.</p>	

Macro:

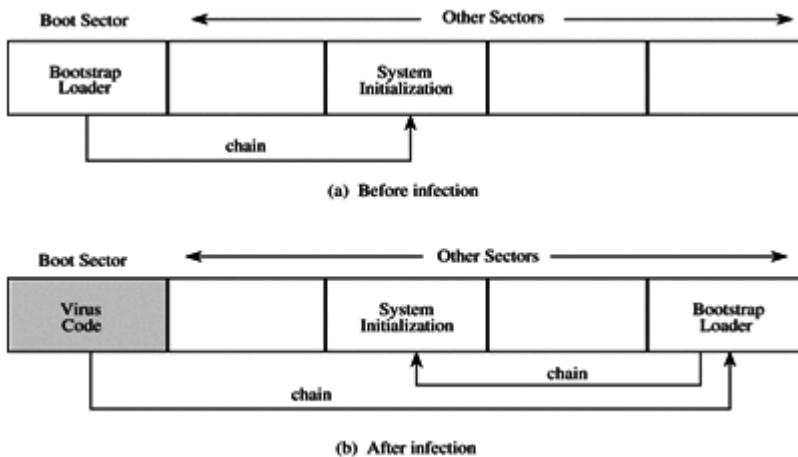
Macro virus is one of the most common types of viruses for various reasons:

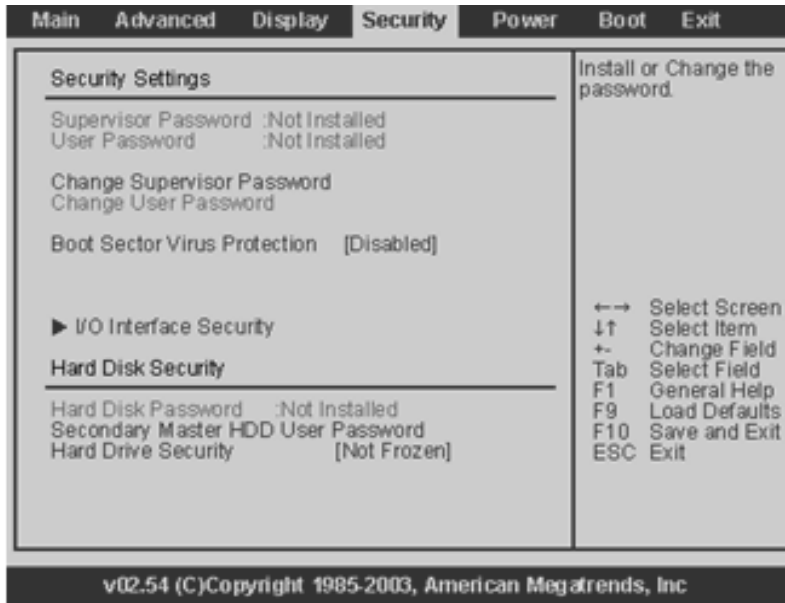
- It requires little skill to write; many common applications (word processing, spreadsheet, presentation) make use of macros; and lastly, documents (such as re-ports, slides, etc.) are frequently exchanged between users.
- Macro viruses are self-replicating macros that can become active if the user opens, closes or saves an infected document.
- However recent versions of these macro-enabling applications display an alert concerning macro-virus risks and permit disabling of macro execution.



Boot sector:

The first logical sector (sector 0) of a floppy or hard disk is designated the boot sector. A boot sector virus infects computers by modifying the contents of the boot sector program, replacing the legitimate contents with the infected version. This type of virus can only infect a machine if it is used to boot the system up. These viruses no longer represent a threat since operating systems now protect the boot sector, and floppy disks are actually unusable for storing modern (and large-size) applications.

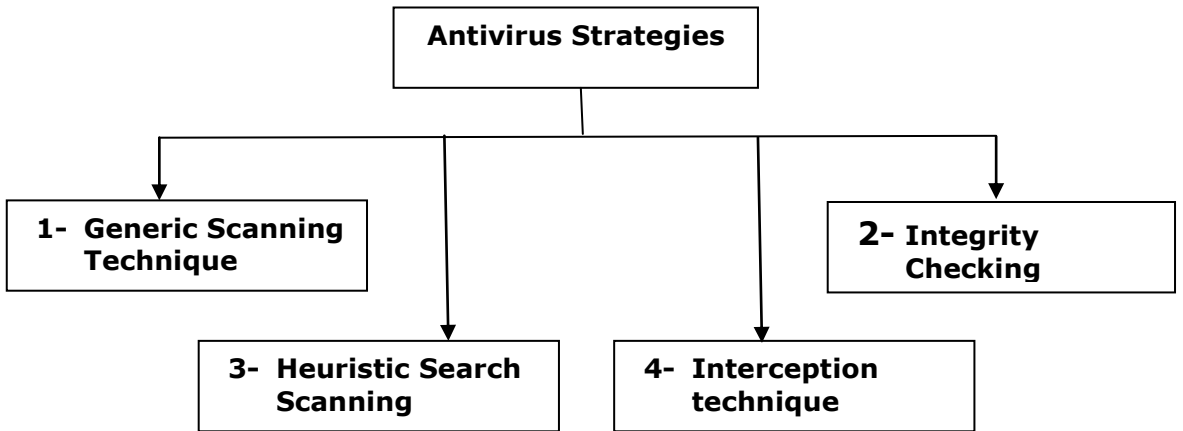




Antivirus and Antispam Software

Antivirus

Antivirus (or anti-virus) software is used to prevent, detect, and remove malware, including computer viruses, worms, and Trojan horses. Such programs may also prevent and remove adware, spyware, and other forms of malware. A variety of strategies are typically employed to detect viruses. Those strategies can be classified into the following four different algorithms:



1) Generic Scanning Technique

Most of the old viruses and some new viruses and Trojans have a recognizable pattern or signature (sequence of bytes) which anti-virus software looks for. Anti-virus software has a library of signature against which it matches the applications, boot sectors and other possible locations of infection. If it can detect a match it will then signal the end-user with the virus details and location where it was found. Anti-virus software's update these signatures at regular intervals. Unfortunately mutating and polymorphic viruses evade simple signature detection by continuously changing their code. They are detected by advanced techniques discussed below.

Generic Scanning Technique

A methodology for defending computers against Viruses

Matching viruses

For each type of virus exist in the predefined virus library

Match location of infection against virus's signature

If virus is detected

Signal end user with Virus Details

Update Viruses Signature

Advantages and Disadvantages:

The advantage of Generic Scanning Technique:

- 1- Signature detection is simple and fast.
- 2- Anti-virus software can look for virus signature in large number of files in a very short period of time.

The Disadvantage is:

- 1- Virus creators today mostly code polymorphic viruses which change the code, while retaining the functionality, thereby evading signature detection algorithm.

2) Integrity Checking Technique

Some anti-virus software can maintain a log file about important files under Windows. The integrity information of those files is stored in their database and is recorded by check-summing to be used later for detecting changes. The integrity information is a data that acts as a signature for the files. If a virus tries to modify a

system file, the anti-virus software at once notifies the user by this technique. Just detecting changes is not enough, however; the detection must have some "intelligence" behind it to avoid confusion.

Advantages and disadvantage

Advantage

- 1- The integrity checking technique perhaps is the most foolproof of them all, as it can determine if a file has been damaged by a virus or not.

Disadvantages

- 1- The problem with this is, not many softwares can implement such precise and perfect technique.
- 2- A data loss or a damage due to corruption can not be distinguished with a case where the file is damaged by a virus.

3) Heuristic Scanning Technique

Heuristic Scanning follows the behavioral pattern of a virus and has different set of rule for different viruses. If any file is observed to be following that set of particular activities then it infers that the particular file is infected. The most advanced part of Heuristic Scanning is that it can work against highly randomized polymorphic viruses too. Heuristic scanning technique has the potential to detect any future virus with ease. F-Secure Anti-virus quite successfully implements this technique

Heuristic Scanning Technique

**A methodology for defending computers against
Viruses**

Define rules for different viruses

Virus Detection

**If file is following any of the defined rule
File is infected
Signal end user with Virus Details**

Advantages and Disadvantages

Advantages

1. It has the prospect of being the only algorithm of all the anti-virus software in the future because it can lead us to very accurate virus detection if properly coded.
2. It doesn't need anti-viruses to download weekly virus database because it can detect viruses from behavioral pattern from the set of rules.

Disadvantages:

1. They are very complex to implement.
2. a virus coder can make a virus that will not obey the set of rules a heuristic scanner hopes it will. Then the virus will be infecting without being noticed.
3. Chances of false alarms are more with heuristic search techniques.

4) Interception Technique

This is the newest technique which continuously monitors your files for suspicious activities. Imagine if a virus is hidden in a CD-ROM. interceptors watches all external drives, data devices as well as internet download or even file download from email. it provides real-time protection to your computer. The key feature of an interceptor is that it has to be very fast to avoid degrading user experience. But most of the modern day anti-virus software implementing the technique do it fairly well like for example Nod 32.

Advantages and Disadvantages

Advantages

1. Gives your computer a Real Time Protection.
2. Any chance of a virus coming from an external drive (CD ROM, pen drive etc) is done away with.

Disadvantage

1. Interceptors can be very easily disabled if it is not very fast to react against threats and most of the viruses do so with perfection.
 1. It is a nuisance for a fast and busy user as it keeps coming with logs and warning messages on trivial issues and that too very frequently.

AntiSpam

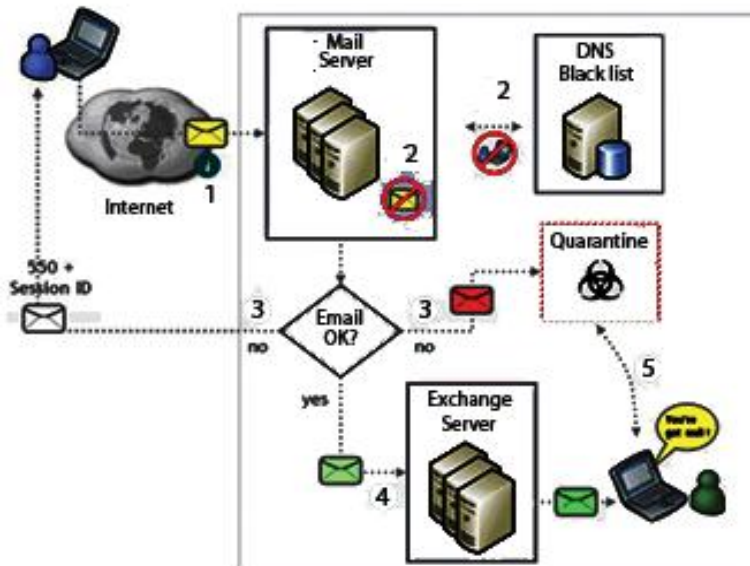
To prevent e-mail spam, both end users and administrators of e-mail systems use various anti-spam techniques. Some of these techniques have been embedded in products, services and software to ease the burden on users and administrators. No one technique is a complete solution to the spam problem, and each has trade-offs between incorrectly rejecting legitimate e-mail vs. not rejecting all spam, and the associated costs in time and effort.

AntiSpam techniques

Anti-spam techniques can be broken into four broad categories that either:

- 1- require actions by individuals,
- 2- can be automated by e-mail administrators,
- 3- can be automated by e-mail senders
- 4- Employed by researchers and law enforcement officials.

Any of the four previous techniques may be applied to detect an email spams.

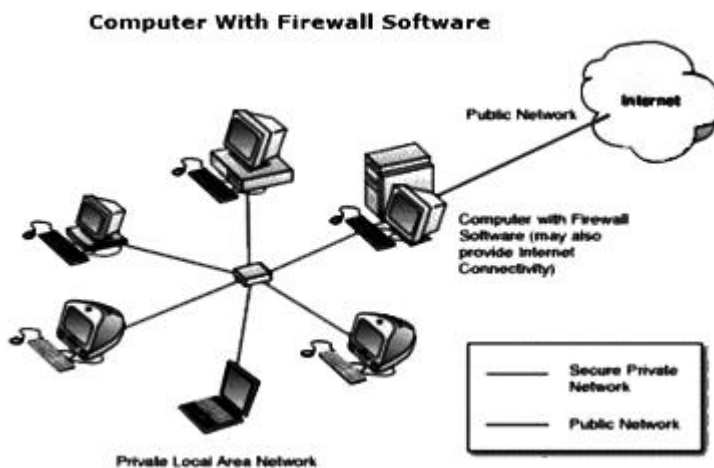
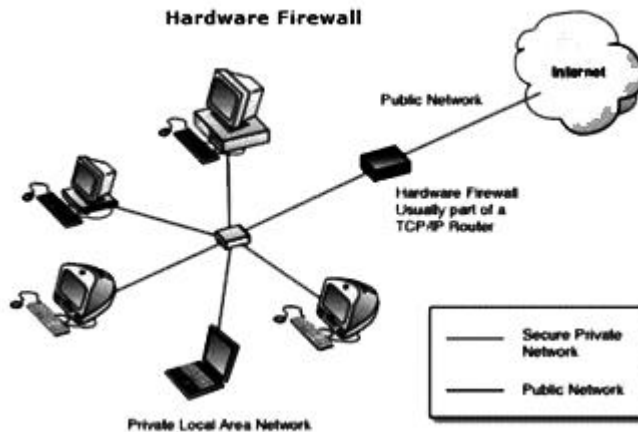


- 1) When Email arriving from outside is first routed to one of several email (NTM) servers
- 2) The first email server function is to pre-filter messages before they are routed to the Microsoft Exchange infrastructure, and ultimately reaches the recipient.
- 3) The filtering done at the NTM server (2) is based on techniques mentioned in the previous slides. If email is ok it passes to step 4 else it is quantized and proceeded to step 5 directly.
- 4) In this step Email is routed to the Exchange servers where it is further filtered by performing a more in-depth look at any attachments. For example, we look inside the actual file contents of an attachment and determine whether the file extension may have been altered (e.g., a .jpg file that is really a .exe file) or whether the attachment contains suspicious or malicious content (e.g., .zip file with a virus inside). If everything is fine, email is forwarded to the email web-based interface.
- 5) This step is related to the email web based interface where a "You've got mail" message is appearing to indicate the receiving of an email. The email either forwarded to either inbox or spam folders

Firewall

What is a Firewall?

A firewall is a system of hardware and software components designed to restrict access between or among networks, most often between the Internet and a private Internet. The firewall is part of an overall security policy that creates a perimeter defense designed to protect the information resources of the organization.

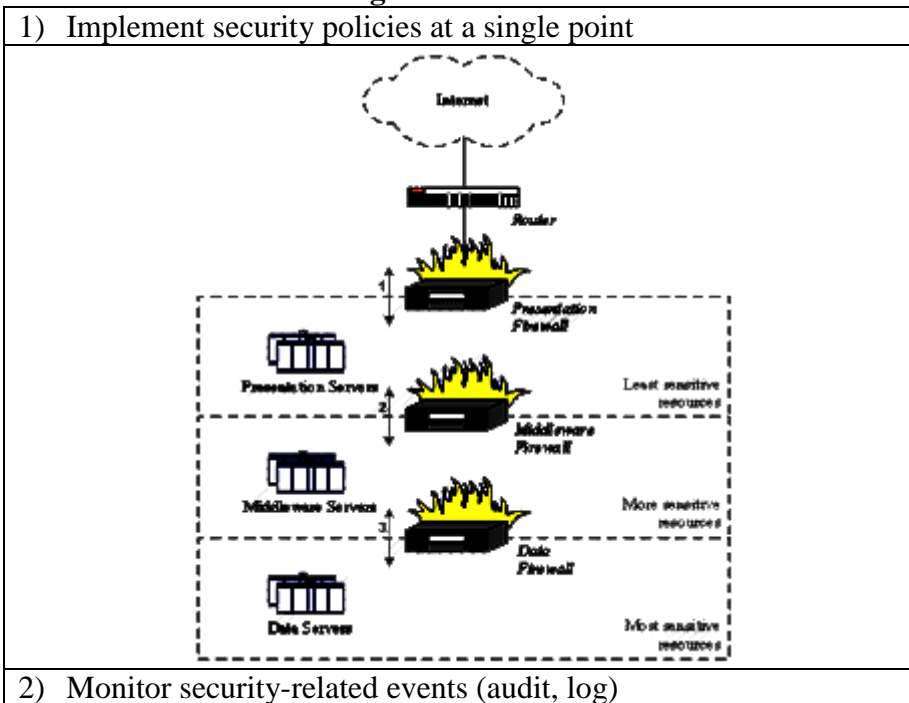


What a Firewall does?

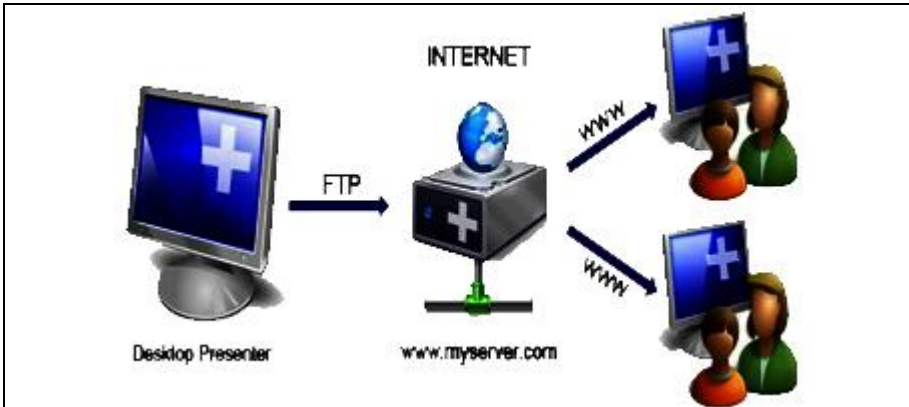
The firewall is an integral part of any security program, but it is not a security program in and of itself. .Firewalls only address the issues of data integrity, confidentiality and authentication of data that is behind the firewall. Any data that transits outside the firewall is subject to factors out of the control of the firewall. It is therefore necessary for an organization to have a well-planned and strictly implemented security program that includes but is not limited to firewall protection.

Firewalls do the following tasks:

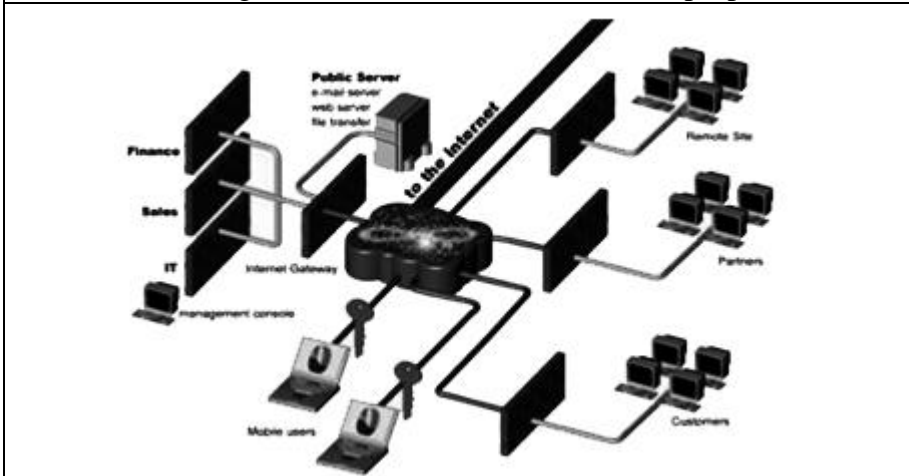
- 1) Implement security policies at a single point



- 2) Monitor security-related events (audit, log)



3) Provide strong authentication for access control purpose



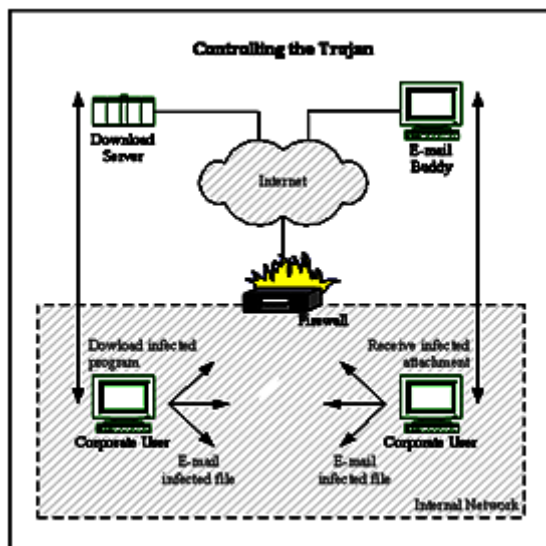
What a Firewall does not do?

- Protect against attacks that bypass the firewall
- Protect against internal threats
 - disgruntled employee
 - Insider cooperates with an external attacker





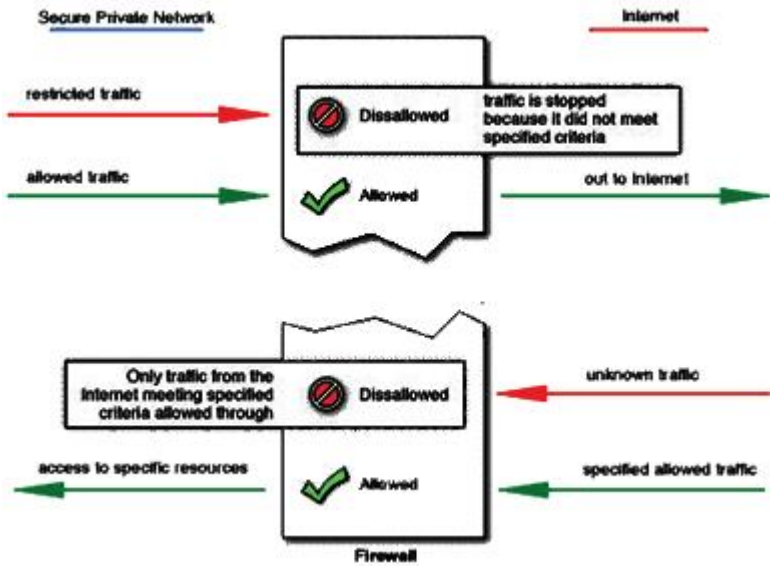
- Protect against the transfer of virus infected programs or files



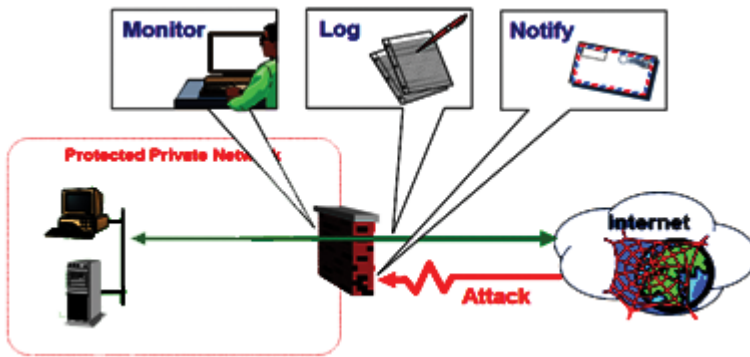
Firewall typical layout and technologies

Broadly speaking, Firewall has a typical layout that:

1. Denies or permits access based on policies and rules.



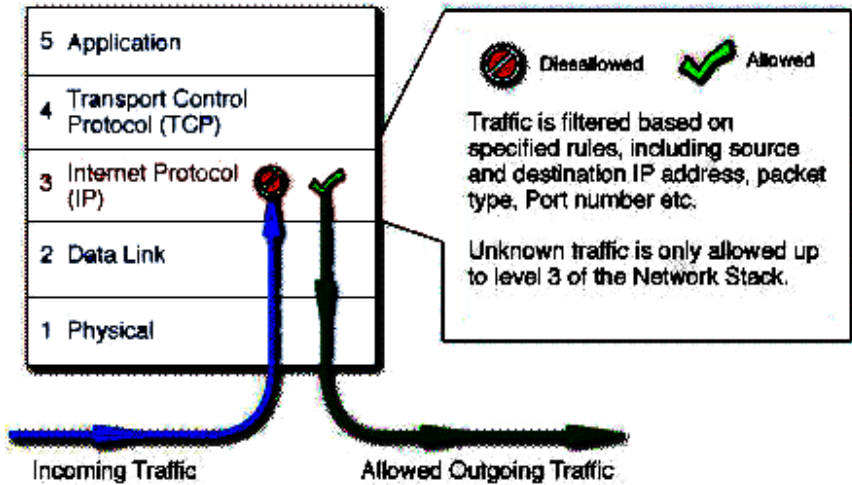
2. A Watches for Attacks



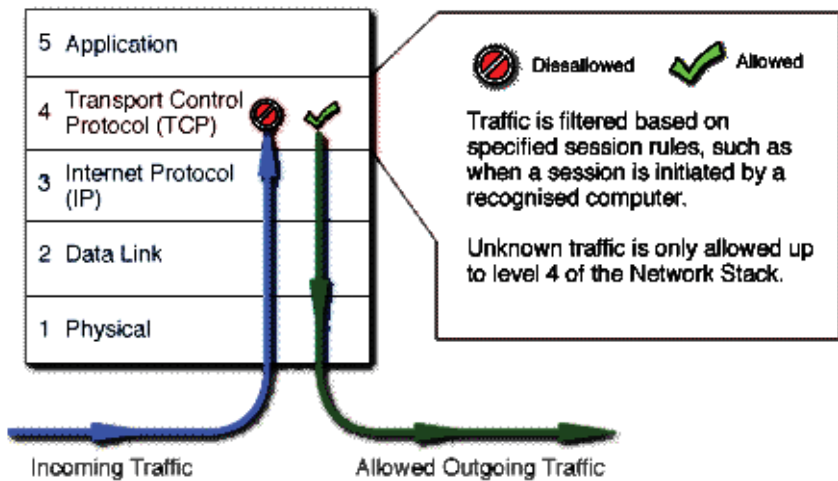
Firewall Technologies

Firewall Technologies may be classified into four categories:

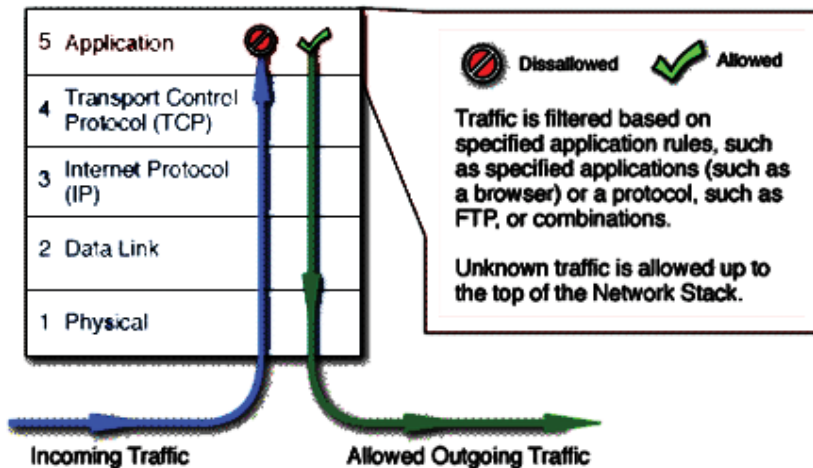
1- Packet filtering firewalls



2- Circuit level gateways



3- Application gateways (or proxy servers)



These technologies operate at different levels of detail, providing varying degrees of network access protection.

Virtual Private Network (VPN)

A virtual private network or VPN is a private, secure path across a public communication network as the Internet. A VPN is set up to allow authorized users private, secure Access to a company network without the need to pay too much money for extending the network. For instance, a VPN could allow a traveling employee, business partner, or remote company office to connect securely to the company network via the Internet. The connection between the sender and receiver acts as if it were completely private, even though it takes place over a public network. A Special encryption

technology is used to protect the data, so it cannot be understood if it is intercepted during transit.

IP spoofing

Many firewalls examine the source IP addresses of packets to determine if they are legitimate. A firewall may be instructed to allow traffic through if it comes from a specific trusted host. A malicious cracker would then try to gain entry by "spoofing" the source IP address of packets sent to the firewall. If the firewall thought that the packets originated from a trusted host, it may let them through unless other criteria failed to be met. Of course the cracker would need to know a good deal about the firewall's rule base to exploit this kind of weakness. This reinforces the principle that technology alone will not solve all security problems. Responsible management of information is essential. One of Courtney's laws sums it up: "There are management solutions to technical problems, but no technical solutions to management problems". An effective measure against IP spoofing is the use of a Virtual Private Network (VPN) protocol such as IPSec. This methodology involves encryption of the data in the packet as well as the source address. The VPN software or firmware decrypts the packet and the source address and performs a checksum. If either the data or the source address has been tampered with, the packet will be dropped. Without access to the encryption keys, a potential intruder would be unable to penetrate the firewall.

Firewall related problems and firewall benefits

1- Firewall related problems

- a) Firewalls introduce problems of their own because Information security involves constraints, and users don't like this because:
 - It reminds them that Bad Things can and do happen.
 - It restricts access to certain services.
- b) Firewalls can also constitute a traffic bottleneck.
- c) They concentrate security in one spot, aggravating the single point of failure phenomenon.

2- Benefits of a firewall.

- a) Firewalls protect private local area networks from hostile intrusion from the Internet.
- b) Firewalls allow network administrators to offer access to specific types of Internet services to selected LAN users.
 - This selectivity is an essential part of any information management program, and involves not only protecting private information assets, but also knowing who has access to what. Privileges can be granted according to job description and need rather than on an all-or-nothing basis.

Exercise 3

Complete

1. Security is broadly defined as the protection of assets from unauthorized -----, ----- , , or -----
2. CIA triad is an acronym that abbreviate the three information security goals which are -----, ----- and -----
3. If the computer is threatened by viruses, the security system may include such things as -----, ----- and file protection mechanisms to guard against this threat.
4. Dissemination of unauthorized information is considered as a(an) ----- threat.
5. Parasitic viruses (or file viruses) are code fragments that reproduce by attaching themselves to -----.
6. Macro Viruses are imbedded in a ----- or ----- document.
7. ----- infects computers by modifying the contents of the boot sector program.
8. ----- is software that prevents and removes adware, spyware, and other forms of malware.
9. As System is called -----, if it is exposed to a potential threat.
10. Heuristic Scanning follows the ----- of a virus and has different set of ----- for different viruses.
11. Real-time protection is provided by the ----- technique.

12. An effective measure against IP spoofing is the use of a -----
----- protocol such as IPSec.
13. Firewall Technologies Used for ----- and -----
14. In phishing attacks, the fraudulent website asks to enter personal information, such as account-----.

True / False

1. Developing a resilient back-up strategy is important in improving Computer security
2. Information security and computer security are identical to each other.
3. Connecting computers to the internet doesn't affect the system threats levels.
4. SQL Injection is a defense technique that used against the process of exploiting web sites information.
5. A computer virus is a small software program that spreads from one computer to another computer through networks.
6. If the virus infected computer is on a network, Virus infection propagation should be limited.
7. An Internet worm uses computer networks and takes advantages of SW bugs to replicate itself.
8. In Generic scanning techniques, Anti-virus software has a library of signature against which it matches the applications, boot sectors and other possible locations of infection.
9. One advantage of the Heuristic scanning technique is that Signature deletion is simple and fast.

10. Polymorphic viruses lead to evading signature detection algorithm.
11. In Antivirus Integrity checking techniques, detecting changes in the integrity information of the log file is enough to detect the virus.
12. In Antivirus Integrity checking techniques, it is not possible to determine the actual reason for losing the data.
13. All Antivirus techniques need to at least download a weekly virus database update.
14. A firewall is a system of only software components designed to restrict access between or among networks.
15. A firewall help in Protecting against the transfer of virus infected programs or files.
16. Firewalls can constitute a traffic bottleneck.
17. Spoofing scams are when criminals create a shadow copy of a real website.
18. Avoid filling out forms in e-mail messages haws no effect to avoid cyber Fraud.

Chapter 4 Societal Impact of the Web

The Net Generation and the new Web culture

Who is the Net Generation?

- The Net Generation, is the largest and most diverse generation in history
- The Net Generation are Those who grew up with computer access at home and/or school
- Loosely defined, The Net Generation are anyone born after 1982 (age 23 and younger)

In other words, the oldest members of this generation will be turning 30 in the year 2010, while the youngest are still in elementary school. Teachers from elementary through graduate school have been working hard to meet the learning needs of this generation, which differ significantly from previous ones. One of those needs is to achieve Computer Literacy. Computer literacy varies widely:

- Socioeconomic factors
- School use of computers/Internet
- Access to computer/Internet at home

What other technologies that helps to meet the Net Generation's learning needs? Those technologies can vary widely as :

- 1- Cable or satellite TV
- 2- Videos, DVDs, CDs
- 3- Graphing calculators
- 4- Cell phones
 - Text messaging, Internet access
 - Digital and/or video camera
 - Calculator, alarm clock, stop watch
 - Distinct ring tones, 'skins'

Now let us talk about, how technology has shaped the net generation view to the world? This view led the Net generation people to have the following needs:

- 1) They want it NOW
- 2) They want it IN COLOR
- 3) They want INTERACTIONS
- 4) They are used to MULTITASKING
- 5) They expect YOU (the instructor) to understand the technology
 - They may not know how it works

To explain those view items in a proper way, let us apply them to one of the major Net generation needs such as education.

In Education,

- 1) They want it Now using any of the following techniques:
 - a. Instant access to course materials
 - b. Fast-loading graphics
 - c. Instant feedback
 - d. Email response
 - e. Assignment/quiz grading
 - f. Instant answers
 - g. Google, Wikipedia, other online resources
 - h. Instant access to student services
 - i. Online registration

j. Online grades

2) They want it in color such as using:

- a. PowerPoint presentations
- b. Graphics (i.e. Flash animations)
- c. Videos, DVDs, Video games
- d. Interactive learning materials

3) They want INTERACTIONS.

- To achieve this need, They communicate with peers frequently using email, text messaging, instant messaging, cell phones, chat rooms
 - Friends and associates
 - Online friends
 - Complete strangers
 - Classmates

4) They are used to MULTITASKING

- They don't sit quietly to do homework
- Can do homework, surf the net, IM friends, chat on the phone, and watch TV all at the same time
 - They respond well to the use of more than one mode of delivery at a time
 - Neuroplasticity- their brains are flexible because they've been flexed
 - Use more than one method of delivery to enhance retention

5) They expect their instructor to understand the technology

- They never experienced life before computers
- They used it in high school, if not sooner

- They are fast learners when it comes to figuring out how to use technology but...
- They may not understand how it works
 - Rely on others for troubleshooting
 - If something doesn't work, they will hit the PANIC button (SEND) and email you for help

The Digital divide:

What does digital divide mean?

The digital divide means that there is significant number across the countries that not only don't have access to technology, to internet and computer but they don't have the skills to be able to use them effectively. Digital divide refers to the gap between those who benefit from digital technology and those who do not. Now let us imagine if all the following technologies are stopped:

- 1- no email
- 2- no cell phones
- 3- no e-space
- 4- no Ipad
- 5- No Internet

What we would do? Another thing imagines that if it had never started? Of course that would be a very tribally bad thing; however this is the case for around 300 million people around the world. So, there are a great effort are spend by different organization to break what is called digital divide.

If digital divide population is a fact, so where is all the technology?

- Industrialized countries are home to 88% of all internet users
 - Yet
- They make only 15% of the world's population

So, what is the meaning of closing (beak) the digital divide?

1- Is closing the digital divide to give poor people access to technology? → (False) → Why →

This because, it is not the matter about accessing to digital technology but about the benefits derived from it. With a closer look, it turns out that upper-to-middle classes have high-quality access to digital technology because the profit motive pushes technologists to work hard at creating "solutions" designed specifically for them.

However, the poor are ignored because the assumption is that designing solutions for them will not be profitable. The result is that even where the poor are provided access to digital technology, it is low-quality. Furthermore, the digital technology they do have access to be often of a design that ends up being harmful rather than beneficial. This, in turn, widens the digital divide not closing.

2- Is the key to closing the digital divide to invest in literacy and education? → (false) → Why

New technologies such as Web 2.0, voice recognition and icon touch screens provide a more sensory experience, so people can benefit without becoming literate.

The clear thing is that, the divide is widening, not narrowing, at an ever-increasing rate. The new view is that closing the digital divide will be most effectively achieved through a two-pronged approach, one direct and the other indirect:

- 1- The direct approach will be for governments and businesses to work together to change the incentives that shape digital markets.
- 2- The indirect approach will be for them to team up on e-government digital technology initiatives that extend rural health care and quality education to the poor.

Through these two approaches, the poor will be able to reap many of the same benefits from digital technology now derived by the wealthy.


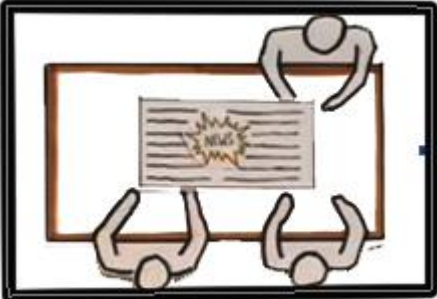
Blogging

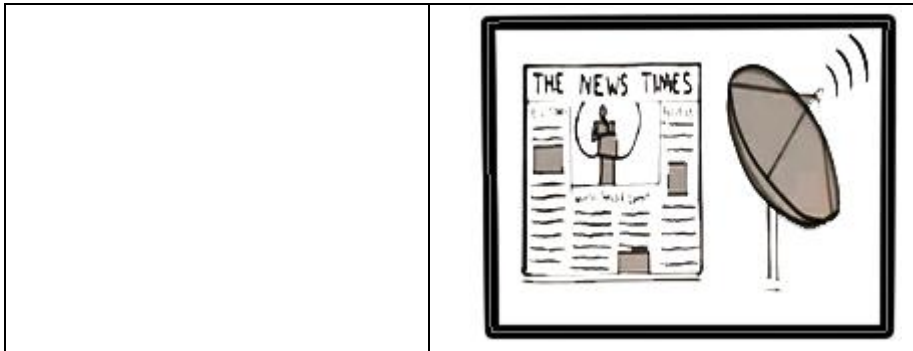
What is a blog? The word blog comes from the combination of the two words: Web + Log

Some people call it weblog but most of the people call it Blog. So, what is a blog in plain English? To make sense of Blog, you have to think about the news and who makes it.



Let us compare making the news in 20's century and in 21 century.
 In 20's century the news produced professionally as follows:

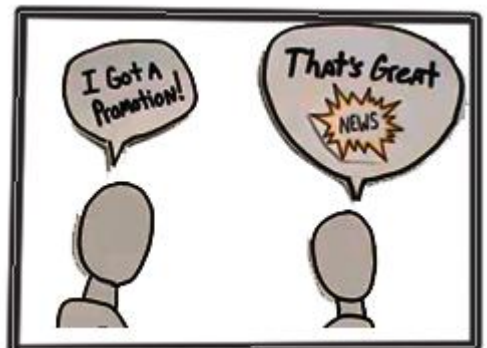
<p>1. When news happens, reporters wrote the story.</p>	
<p>2. A tiny group of people decide what appears in the newspaper</p>	
<p>3. And to broadcast</p>	



While, In 21's century news became professional and personal as follows:

- | | |
|---|--|
| <p>1. A new website called web log or Blog came out on the scene that let anyone be a reporter and publisher often for free.</p> | |
| <p>2. As blogs be popular, it created millions of new sources that give anyone of audience for his own version of news .</p> | |

Of course, this news blog could be applied to a lot of things because it is a fact that everything is news to someone.



With a blog the following tasks could be done easily:

1. a business owner can share news about his business.
2. A mother can share news about her family
3. Or sport star can share news with fans.

These people are all bloggers

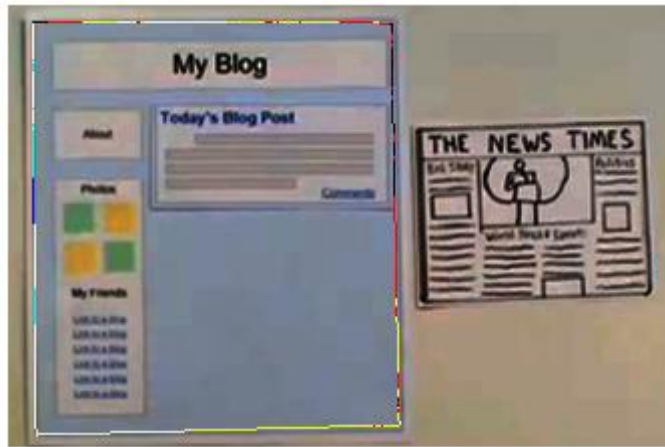


How that does happen?

1. Bloggers may share news on the web easy:
2. Anyone with an idea can start a new blog by clicking a button
3. And share news minutes later.

Here is how blogs work?

1. Blogs websites is organized by blog posts which may include articles in a paper.



2. Bloggers simply fill a form such as this one to post a new story



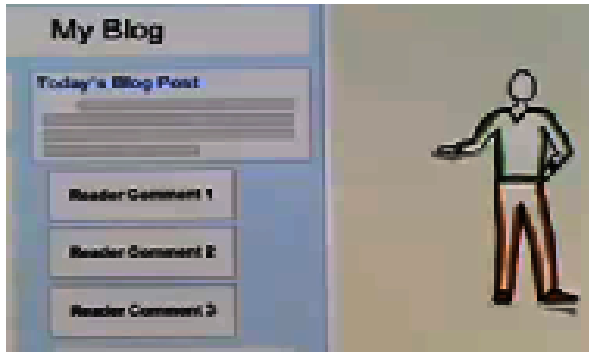
3. By clicking the “post now” button, the blogger's post appears on the top of the web page.



4. Overtime the site becomes a collection o posts, all achieved for easy reference



5. Also each a blog post can become a discussion by comments left by readers.



6. Blogs make the news a 2 way street.



Bloggers usually work together when they have similar interests. Being in an interest relationship, bloggers often work together so that in addition to putting comments they read each other blogs, quote each other and link their blogs together. This creates community of bloggers that inspire and motivate each other. (Circulate). Over lat years blogs have been adopted in a very big way. Since 2003 there have been over 7 million blogs created, each with his own version of new. So, the big deal of the blogs is that it gives normal people the power of media and created a personal kind of news to high number of small audience. Finally you can start your free blog @ blogger.com or wordpress.com

Chapter 5: Laboratory 1

Hypertext Markup Language

Hypertext Markup Language (HTML) is the language for specifying the static content of Web pages. Hypertext refers to the fact that Web pages are more than just text. They can contain multimedia files and provide links for jumping within & without the page. Markup refers to the fact that it works by augmenting text with special symbols (tags) that identify structure and content type. HTML pages are presented using different web browsers such as IE, FireFox...etc. There are many versions of HTML and different browsers have their own add-ons. Webpage is defined as a document or resource of information that is suitable for the World Wide Web and can be accessed through a web browser and displayed on a computer screen. This Information is usually in HTML format and may provide navigation to other WebPages via hypertext links. There are many high-level tools exist for creating Web pages such as Microsoft FrontPage, Adobe PageMill, Macromedia DreamWeaver, HotDog, ...etc. Also, there are many commercial applications that have "save to HTML" options such as Microsoft Word. Those applications are called WYSIWYG (What You See Is What You Get) editors, which can easily be used to edit HTML files. For most users who want to develop basic, static Web pages, these tools are fine, so why we are we learning low-level

HTML using a basic text editor such as notepad or edit plus? There are several reasons:

- 1) Be a skillful Web Developer
- 2) may care about size/readability of pages
- 3) may want to "steal" page components and integrate into existing pages
- 4) may want dynamic features such as scripts or applets
- 5) remote editing of web pages may only be possible using a basic text editor
- 6) In order to write web-based application one needs to know HTML since server-side scripts and programs that programmers write them generate HTML.

Tags

You control how your page looks by using HTML *elements* mixed in with your text. An element is usually made up of two tags, a *start tag* and an *end tag*. Tags are formed by typing words inside of angle brackets like this: `<title>`

This is a start tag. An end tag has a slash after the left angle bracket. Here's how you would write the title of an HTML document:

```
<title>
Romeo and Juliet
</title>
```

The title is everything between the start and end title tags. You could also type it all on one line, like this

```
<title>Romeo and Juliet</title>
```

In the following lab we are going to cover most common used group of HTML tags

Simplest HTML Document

Note: [Type all tags in lowercase letters!]

Here is The tags you should use in every Web page.

```
<html>
<head>
<title>Your Title Here</title>
</head>
<body>
    Your content here ...
</body>
</html>
```

Notes

- Type in a copy of the sample and call it template.html
- Use it as the starting point for all your HTML documents. As you learn new features that you want to use all the time, add them to your template file.
- Use only lowercase letters, numbers, underscores _, and dashes -, in your file names. Don't use spaces.
- The order of your tags is **important**. Don't overlap them. Use them in the order they appear in the sample HTML code.
- The arrangement of your content is not important. The browser formats it based on the HTML tags. Use spaces and blank lines to make it easier to read your HTML code.
- The text in the title-tags does **not** appear on your Web page. It is use in your browser's title bar. Later you'll learn how to use headings.

Try This

Replace **your content here ...** with some of your own content.

1- HEADINGS

<hx> Use headings to mark off different sections of your Web pages.

```
<html>  
<head>  
<title>Headings</title>  
</head>  
<body>  
    <h1>A Level One Heading</h1>  
    Your content here ...  
  
    <h2>A Level Two Heading</h2>  
    Your content here ...  
  
    <h3>A Level Three Heading</h3>  
    Your content here ...  
  
    <h4>A Level Four Heading</h4>  
    Your content here ...  
  
    <h5>A Level Five Heading</h5>  
    Your content here ...  
  
    <h6>A Level Six Heading</h6>  
    Your content here ...  
</body>  
</html>
```

The text of your content stays the regular size, except for the parts between the heading-tags.

Notes

- Heading-tags add white space before and after the heading text.
- Use heading-tags only for headings. Don't use them just to make something big or stand out. There are other tags to do those things.

2- ENTERING TEXT

In this section we try to show a formatted text without using **any HTML Tags to control** the context styles.

```

<html>
<head>
<title>Entering Text</title>
</head>

<body>

    Mary's Lamb

    Mary had a Little Lamb,
    Its fleece was white as snow,
And everywhere that Mary went,
The Lamb was sure to go.

    It followed her to school one day,
That was against the rule,
It made the children laugh and play,
To see a Lamb at school.

</body>
</html>

```

There are no new tags in this topic. Notice how the browser ignores your formatting. The lines of the poem are run together.

Notes

The text is rearranged to fit between the sides of the window. No matter how much white space you use, the browser puts one space between each word and only starts a new line when needed. Browsers work like this so that text can be automatically rearranged whenever your reader changes the size of the window.

Try This

Change the width of your browser's window and watch what happens to the sample HTML document.

3- PARAGRAPHS

The paragraph-tag creates a paragraph.

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

<h1>      Mary's Lamb</h1>
<P>
  Mary had a Little Lamb,
    Its fleece was white as snow,
  And everywhere that Mary went,
    The Lamb was sure to go.
</p>
<p>
  It followed her to school one day,
    That was against the rule,
  It made the children laugh and play,
    To see a Lamb at school.
</p>
</body>
</html>
```

This is a little better. We added a heading, `<h1>`, and paragraphs, `<p>`. The poem's title looks like a title. The stanzas of the poem are run together. All the pretty formatting is lost.

Notes

- The browser removes all of the extra white space.
- The paragraph-tag, `<p>`, inserts a blank line and then starts a new line.
- Although you can leave out the closing paragraph-tag, `</p>`, don't. Future versions of HTML require it.
- Don't use empty paragraphs to insert blank lines. It doesn't work correctly. Use `
`.

Try This

Change the width of your browser's text window and watch what happens to the text at the right edge.

4- LINE BREAKS

The `
` break-tag starts a new line without starting a new paragraph.

```
<html>
<head>
<title>Line Breaks</title>
</head>
<body>

<h1>      Mary's Lamb</h1>

<p>
  Mary had a Little Lamb,<br>
  Its fleece was white as snow,<br>
  And everywhere that Mary went,<br>
  The Lamb was sure to go.
</p>
<p>
```



```
It followed her to school one day,<br>
  That was against the rule,<br>
  It made the children laugh and play,<br>
  To see a Lamb at school.
```

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

Notes

- The browser removes all of the extra white space (if you promise to remember, I won't say this anymore).
- The break tag, `
`, causes a new line to start.
- Use break-tags to add extra blank lines.
- Don't use the break-tag for lists. Instead use one of the list tags, which are coming soon.

Try This

Change the width of your browser's text window and watch what happens to the text at the right edge. Compare with the previous topic, *Paragraphs*. Can you see the differences between `
` and `<p>`?

Emerging versions of HTML require closing tags. For empty tags, like `
`, the proposed standard requires the break-tag to be written like this: `
`. Unfortunately, this causes problems. A work around is to use `
`; note the space after the 'r'. This appears to work fine, even in older browsers.

5- ATTRIBUTES 1

bgcolor, align Attributes change how a tag works. Most tags take attributes. This topic shows how to use attributes with values.

```
<html>
<head>
<title>Attributes</title>
</head>
```

```
<body bgcolor="#00FFFF" text="#FF00FF">
<h1 align="center">Center</h1>
<p align="right">Right Alignment</p>
</body>
</html>
```

Note the color of the background and text on the right. The heading is centered. The paragraph's text is aligned to the right.

Notes

- Attributes go inside the start tag's angle brackets. Values should be inside quotation marks. The general form for an attribute that takes a value is **attribute="value"**
- An attribute should be typed in lowercase letters. A value should be typed in lowercase letters, unless it is a file name or a JavaScript keyword. Then it should be typed exactly as it appears.
- **<body>**
 bgcolor sets the background color for the whole page.
 text sets the letter, or foreground color for the whole page. It out black (#000000).
- Color codes (see below) must start with a #. You can also use color names. For example, "#FF00FF" could be replace by "fuchsia". In either case, don't forget the quotation marks.
- **<p>** and **<hx>**

align changes the position of the paragraph or heading.

- **center** centers the paragraph or heading between the right and left margins. **right** lines up the paragraph or heading against the right margin. **left** is the default value for **align**.

Aqua	00FFFF	Black	000000
Blue	0000FF	Fuchsia	FF00FF
Gray	808080	Green	008000
Lime	00FF00	Maroon	800000
Navy	000080	Olive	808000
Purple	800080	Red	FF0000
Silver	C0C0C0	Teal	008080
White	FFFFFF	Yellow	FFFF00

Here are some colors to experiment with.

Try This

- Change the values of the attributes presented here. Note how the new values alter the appearance of your page.
- Color names are browser-dependent. Although they are easier to remember, your best bet is to stick with color codes.

ATTRIBUTES 2

Background: Using an image as a background.

```
<html>
<head>
<title>Attributes</title>
</head>

<body background="bk/pat.gif">

</body>
</html>
```

Setting the **background** attribute to a valid image file causes your browser to paint the image over and over in the background of your page. Everything else on the page sits on top of it.

Notes

- This sample won't work if you type it in. You have to use an image file that is on your computer.
- You can use .gif and .jpg image files as backgrounds. With newer browsers you may be able to use .png files, too.
- Some backgrounds might make it hard to read your page.
- Notice that the image file is in the subdirectory **bk**.

Try This

Bring your own image and try it out

Tips

The easiest way to use images in your Web pages is to store them in the same folder. Then to use an image called back.gif as your background all you need to do is add

background="back.gif"

Another way is to keep your images in a separate folder. This folder should be in the same folder as your Web pages.

COMMENTS

<!-- helpful info --> The text with a comment is not shown on your Web page.

```
<html>  
<head>  
<title>Comments</title>  
</head>  
<body>  
<!-- This is a comment. -->  
<!-- This comment is spread  
    out over two lines. -->  
<!--  
    This is a good  
    comment too.  
-->  
</body>  
</html>
```

The output window should be empty.

Notes

- Newer browsers allow HTML tags in comments. Older browsers may give unusual results. It is probably better not to put tags in comments.
- Use comments to include information that you don't want to appear on your Web page, but might be part of it: dates, authors, to-do lists, etc.

- Type the comment tag as it appears in the sample, even though some authors leave out dashes.

6- PREFORMATTED TEXT

`<pre>` Keeps spaces, tabs and returns exactly like you type them.

```
<html>
<head>
<title>Entering Text</title>
</head>

<body>

<pre>
    Mary's Lamb

    Mary had a Little Lamb,
      Its fleece was white as snow,
    And everywhere that Mary went,
      The Lamb was sure to go.

    It followed her to school one day,
      That was against the rule,
    It made the children laugh and play,
      To see a Lamb at school.
</pre>
</body>
</html>
```

Everything between `<pre>` and `</pre>` appears exactly as typed. If you resize the window, the text is not reformatted to fit. Scroll bars may appear.

Notes

- The output from `<pre>` is displayed in a typewriter-style (monospaced) font, not the usual one.
- Use `<pre>` for information that has to be lined up to be understandable, like tables of information, columns of numbers or ASCII art (simple pictures made from keyboard letters, number and symbols).
- It is recommended to use spaces instead of tabs. Different browsers use different size tabs.
- Don't make your preformatted lines too long. Keep them to about 60 characters.
- Earlier advice was that inside of `<pre>` you can use `<a>`, ``, ``, `<i>` and ``, but **not** `<hx>`, `<p>` or `
`.

HTML 4 and up says don't use ``, `<object>`, `<applet>`, `<big>` and , `<small>`, `<sub>`, `<sup>`, `` or `<basefont>` in `<pre>`.

7- ENTITIES

Entities are used to enter special characters and characters you can't type from the keyboard, like ©.

```

<html>
<head>
<title>Entities</title>
</head>
<body>

<p>
5 &gt; 2
</p>
<p>
2 &lt; 5 or 2 &#60; 5
</p>
<p>
&amp;

```

```

</p>
<p>
Copyright &copy; 2000
</p>
<p>
&#65; &#66; &#67;
</p>

</body>
</html>

```

Entities start with an ampersand, & and end with a semicolon, ;.
Notes

- Escape codes are used to 'escape from' special meanings. <, >, & and " have special meanings in HTML. To put one of these characters in a Web page use its entity.
- An entity is formed by typing an ampersand, &, a sharp sign, #, a decimal ASCII code and semicolon, ;. Don't leave out the semicolon even though some browsers allow it. Some entities have names. Name entities never contain a sharp sign.
- To enter <, > or & always use its entity. You're supposed to use " for " but browsers don't seem to care. Use the entity anyway.
- You can enter any character using its entity: the ASCII codes for A, B and C and 65, 66 and 67, respectively.
- Table in right includes a list of some entities and their name entities.

Character	Entity	Name
<	<	<
>	>	>
&	&	&
"	"	"
©	©	©
÷	÷	÷
×	×	×
±	±	±
¢	¢	¢
£	£	£
¥	¥	¥
	 	

8- UNORDERED LISTS

, Make bulleted lists.

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

<body>
<ul>
<li>The first item in the list.</li>
<li>The second item in the list.
  This line is still part
  of the second item. </li>
<li>The third item in the list. </li>
</ul>
</body>
</html>
```

Use `` ... `` tag for each item in the list.

Notes

- Each item in the list is set off with a **bullet**, which is usually, but not always, small solid circle. For lists with numbers see the next topic, **Ordered Lists**. Ordered lists and unordered lists work the same.
- Use unordered lists for lists of items where the order does not matter.
- A list item can be as long as you want. It can contain paragraphs and other lists.
- Don't use `` outside of lists.

Try This

Make a list of lists. That is, make a list where each list item is a list.

9- ORDERED LISTS

``, `` Make numbered lists.

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



```
</head>
<body>
<ol>
<li>The first item in the list.</li>
<li>The second item in the list.
  This line is still part
  of the second item.</li>
<li>The third item in the list.</li>
</ol>
</body>
</html>
```

Use `` ... `` tag for each item in the list.

Notes

- Each item in the list is set off with a number. For lists without numbers see the previous topic, **Unordered Lists**. Ordered lists and unordered lists work the same.
- Use ordered lists for lists that should be numbered, like lists of instructions. Don't try to number lists yourself. When you need to add an item or change the order, you get stuck making the changes. If you use an ordered list, your list gets renumbered automatically (and correctly) every time.
- A list item can as long as you want. It can contain paragraphs and other lists.

Try This

- Make an ordered list that contains unordered lists as list items.
- Try different combinations of list of lists to see how they nest.

10- DEFINITION LISTS

`<dl>`, `<dt>`, `<dd>` Make a list of terms and definitions.

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

```

</head>
<body>
<dl>

<dt>The first term.</dt>
<dd>The definition of the first term
in the list.</dd>

<dt>The second term.</dt>
<dd>The definition of the second
term in the list.
This definition is a big one
that might span a
couple of lines.</dd>
<dd>The second term has a two part
definition.</dd>

<dt>The third term.</dt>
<dd>
The definition of the third term
in the list.
</dd>

</dl>
</body>
</html>

```

Each item in a **definiton list**, `<dl>`, is made up of a **definition term**, `<dt>`, followed by zero or more **definition descriptions**, `<dd>`.

Notes

- Other lists, including definition lists, are allowed within the definition part, but **not** within the term part.
- Note the different arrangements of text within the tags.

11- LINKS TO DIFFERENT PAGES

<a>, **href** Set up an **anchor** with a **hyperlink**.

```
<html>
<head>
<title>A Sample Link
</title>
</head>

<body>

Here is
<a href="http://www.yahoo.com">a link</a>
to a test page.

</body>
</html>
```

Click the link in the right frame. The link is probably blue and underlined, but it does not have to be. The mouse cursor should change to a hand with a pointing finger.

Notes

- Use this kind of link to jump to any page on the WWW.
- Clicking the link brings the page from the URL listed after the **href** attribute to your browser.
- The URL to another Web page usually starts with `http://`, but it can be any protocol understood by your browser, such as `ftp://`, `file://`, `mailto:` and others. It can also be a bare file name of a Web page, an image file or a plain text file.

Try This

Make a link to a Web page that is in the same folder as the one in the sample. It should have the form ``

LINKS TO ANCHORS ON THE SAME PAGE

<a>, **name** Making a named anchor.

```
<html>
<head>
<title>Named Anchors</title>
</head>

<body>

<a href="#bottom">Jump down.</a>

<p>
Testing<br>
One<br>
Two<br>
Three
</p>

<p>
Testing<br>
One<br>
Two<br>
Three
</p>

<p>
Testing<br>
One<br>
Two<br>
Three
</p>

<p>
Testing<br>
One<br>
Two<br>
Three
</p>

<a name="bottom">You jumped here!*****</a>
```

```
<p>  
Testing<br>  
One<br>  
Two<br>  
Three  
</p>
```

```
<p>  
Testing<br>  
One<br>  
Two<br>  
Three  
</p>
```

```
<p>  
Testing<br>  
One<br>  
Two<br>  
Three  
</p>
```

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

Click the link in the right frame. Click the output button, above right, to restore the right side.

Notes

- Most of text in this sample is just to fill up the screen so there is someplace to jump to!
- There are two anchors in this example. The anchor near the top is a hyperlink, or link, to the named anchor near the bottom.
- Note that the sharp sign, #, should only appear with the **href** attribute. It means the link is to *anamed* spot in

the current document. A spot is named using the **name** attribute in an anchor.

- You can use as many **name** attributes as you like in a Web page, but each name has to be different. You can reuse names in other Web pages. You can use the name in many **href**'s.
- Use named anchors to make a 'live' table of contents for a Web page.

Try This

Rewrite this sample so that you can jump back and forth between *Jump down* and *You jumped here*.

12- LINKS TO ANCHORS ON DIFFERENT PAGES

<a>, **href** Linking to named places on other pages.

```
<html>
<head>
<title>A Sample Link</title>
</head>

<body>

<a href="http://test.com/testpage.html#namedplace">Jump to
the anchor on a different page.</a>

</body>
</html>
```

Sets up a link to a named place in a different page. Click the link in the right frame. Note that when the page loads it is not at the top.

Notes

- The sharp sign, #, following the URL means the link is to a *named* spot in some other document. The **name** attribute had

to be used in the other document. Usually you can only use this feature with your own pages.

13- LINKS: MAILTO

`<a>`, **href**: making an email link.

```
<html>
<head>
<title>A mailto Link
</title>
</head>

<body>

Comments? Email P.J. LaBrocca at
<a href="mailto:ahmed@yahoo.com">p@LaBrocca.com</a>

</body>
</html>
```

Click the link in the right frame. Your email program should run.

Notes

- There are no slashes, //, in a **mailto** URL.
- A **mailto** link runs an email program.
- Since not all browsers support this feature, it's probably a good idea to use the email address as the link text, as in the sample. You can use any other text you want.

14- INLINE IMAGES

``, **src** Displays images mixed in with text.

```
<html>
<head>
<title>Inline Images</title>
```

```
</head>

<body>
<p>
This is an inline image.
Notice it appears

right in with your text.
</p>
</body>
</html>
```

Inline images display right inline with your text. They act like big words. Note the big space between the lines of text. Inline images do not start new line or paragraphs

Notes

- The **src** attribute is required. It should be the file name of a gif, jpg, or, for newer browsers, png image file.
- Use **width** and **height** even though they're not required. This information allows your browser to lay out the page before the images start downloading and then fill in the images as they arrive. Your pages will load faster and more smoothly.
- You can get the width and height of your images from a graphics program. Some HTML editors insert them automatically.
- Don't use **width** and **height** to change the size of an image unless you're doing it for effect. Doing so distorts the image. Use a graphics program.
- The **alt** attribute should contain some descriptive text to be displayed if the image can't.
- Setting **border** to zero removes any border that might appear around an image.

Try This

Use an inline image as a link.

15- INLINE IMAGES AS LINKS

``, **border** Outlining images.

```
<html>
<head>
<title>Inline Images as Links</title>
</head>
<body>
<p>
<a href="http://www.ms88.com/testpage.html">

a link.
</a>
<p>
<a href="http://www.ms88.com/testpage.html">

a link.
</a>
<p>

not a link.
</body>
</html>
```

Click on each of images in the right frame. The first two are links to a test page. Click the **Output** in the upper right frame to restore the original output. Use an image that you have on your computer to get the sample to work correctly.

Notes

- The browser highlights the image by drawing a blue border around it. Once you click the link, the border changes to purple. (These are usually the colors; browsers and Web page authors can change them.) You can include some text with the image.
- To turn off the border set the **border** attribute to zero.

- The third image in the sample is not a link and has its **border** attribute set to three. If you're using Netscape Navigator or newer versions of Internet Explorer, you see a three pixel wide border around the image. Other browsers may ignore **border** outside of a link.

Try This

Change **border** to different values to see what happens.

16- INLINE IMAGES 2

****, **align**, **top**, **middle**, **bottom** Changing how the text and images line up.

```
<html>
<head>
<title>Inline Images 2</title>
</head>
<body>

<p>

ONCE upon a time there
lived a rich man, who had a wife,
and one daughter, a very sweet and pretty
girl. The wife fell sick and died
</p>
<p>

ONCE upon a time there
lived a rich man, who had a wife,
and one daughter, a very sweet and pretty
girl. The wife fell sick and died
</p>
```

```
<p>

ONCE upon a time there
lived a rich man, who had a wife,
and one daughter, a very sweet and pretty
girl. The wife fell sick and died
</p>
</body>

</html>
```

The blue rectangle takes up the entire image.

Notes

- If text wraps around it moves to the line below the image.
- Compare the position of the text with the image for each of **top**, **middle** and **bottom**.
- The default alignment is **bottom**.
- **top**, **middle** and **bottom** are useful for small images that you want mixed in with your text. See the next topic for flowing text around larger images.

Try This

Change the size of the output window and observe how the text gets rearranged.

17- INLINE IMAGES 3

****, **align**, **left**, **right** Wrapping text around floating images.

```
<html>
<head>
<title>Inline Images 3</title>
</head>
```

```

<body>
<p>

ONCE upon a time there lived a rich man, who had a wife,
and one daughter, a very sweet and pretty
girl. The wife fell sick and died, and,
after a while, the father
married again.
But he did not choose wisely this time,
for the lady he married was proud
and cross, and she had two grown-up
daughters, just like herself in all things.
</p>

<p>

ONCE upon a time there lived a rich man, who had a wife,
and one daughter, a very sweet and pretty
girl. The wife fell sick and died, and,
after a while, the father
married again.
But he did not choose wisely this time,
for the lady he married was proud
and cross, and she had two grown-up
daughters, just like herself in all things.
</p>

</body>
</html>

```

left and **right** create *floating* images. The text wraps around them like in a book or magazine.

Notes

- Floating images seem to float in the text and can appear on the left or right side of the screen.

- Once you use **left** or **right**, the text continues to wrap to the bottom of the image, even if you insert a new paragraph or a line break. This presents a problem if the image takes up more vertical space than the text you want next to it. The next topic shows how to break out of a **left** or **right**.

Try This

- Change the width of the browser output window to get a feel for how **left** and **right** work with the image tag.
- Try changing the size of the image.
- Use different combinations of paragraphs and line breaks to convince yourself that they have no effect with **left** or **right**.

18- INLINE IMAGES 4

**
, **clear, **all**, **left**, **right**, Breaking out of wrapping around floating images.

```

<html>
<head>
<title>Inline Images 4</title>
</head>
<body>

<p>

ONCE upon a time there lived a rich man, who had a wife,
and one daughter, a very sweet and
pretty girl.
<br clear="left">
The wife fell sick and died, and,
after a while, the father
married again.
But he did not choose wisely this time,
for the lady he married was proud
and cross, and she had two grown-up
daughters, just like herself in all things.

```

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

The **clear** attribute means to stop wrapping text around the image. The value of "**left**" makes the **The wife...** continue on the first available line that starts at the left margin..

Notes

- **clear** adds enough vertical space to reach the next full margin. It is used only to stop wrapping text around floating images.
- **right** works exactly like **left** except on the right side. **all** breaks to the first full line available on both the left and right sides.

Try This

Put floating images with different heights on opposite sides of the screen and try breaking using different values for **clear**.

19- INLINE IMAGES 5

Faking a centered floating image.

```
<html>
<head>
<title>Inline Images 5</title>
</head>
<body>

<p align="center">



Title<br>A truly thrilling story!
</p>
<p align="right">
```

```
The Hero  
<br>The Heroine  
</p>  
  
<br clear="all">  
<p>  
The amazing story of ...  
</p>  
  
</body>  
</html>
```

The three images keep their positions (left - center - right) when the window is resized.

Notes

- The first two images use **left** and **right**. The third is a regular inline image. **The order of the image tags matters.**
- `<p align="center">` centers the third image and the text, but has no effect on the floating images.
- `<p align="right">` right justifies **The Hero** and **The Heroine**. Note the position of the `
`. It matters for some browsers.
- `<br clear="all">` breaks out of the text wrap that's occurring between the two floating images. In this sample, **left** or **right** would work just as well because the images are the same height.

Try This

Remove `<br clear="all">` and Rearrange the image tags.

20- INLINE IMAGES 6a

Making horizontal images touch.

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

```
<title>Inline Images 6a</title>
</head>
<body>

<p>

</p>

</body>
</html>
```

Type the `` tags on one line with *no spaces* between them.

Notes

- Inline images work like letters. If you don't put spaces between them, neither will the browsers.
- I left out **alt** and **border** to simplify the sample. When you are building up a picture from several images it might be a good idea to leave out **alt**.

21- INLINE IMAGES 6b

Making vertical images touch.

```
<html>
<head>
<title>Inline Images 6b</title>
</head>
<body>

<p>
<br>
<br>

</p>
```



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

Note the **
** at the end of the first two images.

Notes

- The images can also be typed all on one line if you like.
- I left out **alt** and **border** to simplify the sample. When you are building up a picture from several images it might be a good idea to leave out **alt**.

22- EXTERNAL IMAGES

External images are loaded through an HTML link.

```
<html>
<head>
<title>External Images</title>
</head>

<body>

An external image appears as a link
on your page.
When you click it you get to see the
picture.
<a href="88.gif">Some eight balls.</a>

</body>
</html>
```

External images load only when clicked. The image replaces the page it was requested from.

Notes

- You call an external image exactly as you would an HTML document. The browser knows what to do with it by its extension, .gif or .jpg (and maybe png).
- You can't do anything to an external image except load it. It's always positioned in the browser window's upper left hand corner.
- You might want to put an external image into a separate Web page as an inline image. Then call the page when you want the image.

23- LOGICAL STYLES

****, ****, **<code>** Logical styles tell how you want words to be used. The browser decides how the words look.

```

<html>
<head>
<title>Logical Styles</title>
</head>

<body>

<p>
Here is an
<em>emphasized</em>
word.
</p>
<p>
These words are
<strong>strongly emphasized.</strong>
</p>
<p>
This is some text.<br>
<code>This is some text as code.</code>
</p>

</body>
</html>

```

These tags give a hint about how the browser should display their contents.

Notes

- `` is usually displayed in *italics*.
- `` is usually displayed in **bold**.
- `<code>` is usually displayed in a monospaced font. It is supposed to mean that the enclosed text is in a computer programming language.
- Compare the logical styles with the physical styles in the next section.
- Logical styles allow the browser to decide how to display your content. For example, `` is a hint by you to the browser that you want the enclosed text emphasized. If your reader's browser cannot render italics, it might emphasize the text by using underlining or a different color. If you use `<i>` (see next topic) to force italics, and your reader has a browser that can't display them, then your text gets **no** emphasis.
- Remember, browsers ignore tags they don't understand.

24- PHYSICAL STYLES

`<i>`, ``, `<tt>` Physical styles tell the browser how to display text.

```
<html>
<head>
<title>Physical Styles</title>
</head>

<body>

<p>
Here is a word in
<i>italics.</i>
</p>
<p>
<b>These words are in bold.</b>
```

```
</p>
<p>
This is some text.<br>
<tt>This is some text in teletype.</tt>
</p>
</body>
</html>
```

These tags tell the browser the *style* you want used to display their contents.

Notes

If your reader's browser is unable to display italics, bold or teletype it does nothing.

25- RULES

<hr> Rules are horizontal lines. They can be draw in various widths, thicknesses and positions.

```
<html>
<head>
<title>Rules</title>
</head>
<body>
1
<hr>
2
<hr width="50">
3
<hr width="50%">
4
<hr align="left" width="25%">
5
<hr size="12">
6
<hr noshade>
</body>
```

```
</html>
```

Use rules to separate parts of your page. Don't use too many.

Notes

- Rules are centered between the left and right margins unless you use **left** or **right**.
- The numbers are in pixels (screen dots) or percents.
- Note the difference between rule #2

```
<hr align="50">
```

and rule #3

```
<hr align="50%">
```

The rule #2 will always be 50 pixels long. The rule #3 will always be half the distance between the left and right margin.

- The **size** attribute sets the thickness, or height, of the rule.

26- SUBSCRIPTS AND SUPERSCRIPTS

`<sub>`, `<sup>` Adding text below and above the line.

```
<html>
<head>
<title>Subscripts and Superscripts</title>
</head>
<body>
<p>
H<sub>2</sub>O
</p>
<p>
x<sup>3</sup>
</p>
</body>
</html>
```

`<sub>` and `<sup>` are useful for pages containing equations and chemical formulas. They can also be used for footnotes.

Notes

- Subscripts are written about halfway below the base line of normal characters. Superscripts are written about halfway above the top of normal characters. Both should be slightly smaller than normal.
- Although the results may not be satisfactory, you can put subscripts and superscripts inside each other. Be sure not to overlap them. For example, you can put a subscript on a superscript.

Try This

Write HTML to display:



$$a^2 + b^2 = c^2$$



27- QUOTING BLOCKS OF TEXT

<blockquote> Used for long quotations.

```
<html>
<head>
<title>Quoting Blocks of Text</title>
</head>
<body>
<p>
The blockquote tag should be used
for quoting long blocks of text.
Usually the quoted text is indented
from the left and right margins, and
is sometimes in italics.
</p>
```

```
<blockquote>
```

The **blockquote** tag should be used for quoting long blocks of text.

Usually the quoted text is indented from the left and right margins, and is sometimes in italics.

```
</blockquote>
```

```
<p>
```

The **blockquote** tag should be used for quoting long blocks of text.

Usually the quoted text is indented from the left and right margins, and is sometimes in italics.

```
</p>
```

```
</body>
```

```
</html>
```

The middle paragraph should look different from the other two.

Notes

- **<blockquote>** is used to mark some text as a quotation from another source. Usually, the left and right margins are indented. Don't use it just to indent, because a browser can use another effect, such as italics or underlining.
- **<blockquote>** starts a new paragraph, so don't put it inside **<p>** tags.

28- CENTERING

<center> Centers content between the left and right margins.

```
<html>
```

```
<head>
```

```
<title>Centering</title>
```

```
</head>
```

```
<body>
```

```
<center>
```

```

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

In this sample an inline image is centered using the center tag.

Notes

- `<center>` inserts a line break before and after the centered content.
- **You can use the the `<center>` tag with inline images, headings, tables and other elements.**
- Many elements have an **align** attribute. It is probably better to use this
`<h1 align="center">Hi</h1>` than
`<center><h1>Hi</h1></center>`.
- `<center>...<center>` is an abbreviation for
`<div align="center">...</div>`

29- FONT SIZE 1

``, **size** Changing the size of letters with absolute values.

```
<html>
<head>
<title>Fonts 1 - SIZE</title>
</head>
<body>

<font size="7">HTMLEmentary</font> <br>
<font size="6">HTMLEmentary</font> <br>
<font size="5">HTMLEmentary</font> <br>
<font size="4">HTMLEmentary</font> <br>
<font size="3">HTMLEmentary</font> <br>
<font size="2">HTMLEmentary</font> <br>
<font size="1">HTMLEmentary</font>

</body>
```



```
</html>
```

The `` tag has to have an attribute.

Notes

- Fonts usually start out with **size** set to 3,
- The allowed sizes 1-7.
- Another way to change the size of fonts is presented in the next topic. Compare it with this topic.
- The default font size can be changed using a `<basefont>` element in a document's `<head>` section.

30- FONT SIZE 2

``, **size** Changing the size of letters with relative values.

```
<html>
<head>
<title>Fonts 2 - SIZE</title>
</head>
<body>

<font size="+5">HTMLEmentary</font> <br>
<font size="+4">HTMLEmentary</font> <br>
<font size="+3">HTMLEmentary</font> <br>
<font size="+2">HTMLEmentary</font> <br>
<font size="+1">HTMLEmentary</font> <br>
HTMLEmentary <br>
<font size="-1">HTMLEmentary</font> <br>
<font size="-2">HTMLEmentary</font> <br>
<font size="-3">HTMLEmentary</font>

</body>
</html>
```

Using a + or - with the **size** attribute lets you change the font size relative to the base font size. Note that the first two lines and the last two lines are the same size.

Notes

- The allowed relative values are +1 to +6 and -1 to -6.
- Values of +4 and +5 have the same effect, as do values of -2 and -3. If the resulting value is larger than 7, the value of **size** is set to 7. If the resulting value is less than 1, the value of **size** is set to 1.
- It's usually better to use relative sizes. Here's why. Let's say you want to make a word one size bigger, so you set **size** to 4. On a browser where the default size is set to 4, the word will appear the same size as all the others. However, if you had set **size** to +1, the word would appear in size 5.
- Compare this with the previous section.

31- FONT 3 SMALL CAPS

****, **size** This topic shows two ways to make small caps.

```
<html>
<head>
<title>Fonts 3 - Small Caps</title>
</head>
<body>
<p>
<font size="+2">
  <font size="+3">S</font>MALL
  <Font size="+3">C</font>APS
</font>
</p>
<p>
<font size="+5">
```

```
<font size="6">S</font>MALL
<font size="6">C</font>APS
</font>
</p>
</body>
</html>
```

The two versions above produce the same result.

Notes

- The first, or outer, font tag sets the size for all the letters. The inner font tags change the size for each first letter.
- The inner tags are completely enclosed by the one outer font tag.
- This sample assumes a default font size of 3. The +2 in the first part is (3 + 2) and means the same as the 5 in the second part. The +3 is (3 + 3) or 6.

32- FONT 4 COLOR

****, **color** How to change the color of a block of text.

```
<html>
<head>
<title>Fonts 4 - Color</title>
</head>
<body>
<p>
<font color="#ff0000">Red</font>
<p>
</p>
<font color="#00ff00">Green</font>
</p>
<p>
<font color="#0000ff">Blue</font>
</p>
</body>
```

```
</html>
```

Use care when picking colors. Fancy colors aren't much good if your readers can't make out what you wrote.

Notes

- The **color** attribute is used to change the color of a letter, a word or a block of text. (Use the body tag's **text** attribute to set the text color for the whole file.)
- A color code, **#rrggbb**, is made up of sharp symbol, #, followed by three hex numbers, **rr**, **gg**, **bb**, that give values for red, green and blue, respectively. Color code always work, even if they are a little harder to read. The color names may not work with some browsers.

Aqua	00FFFF	Black	000000
Blue	0000FF	Fuchsia	FF00FF
Gray	808080	Green	008000
Lime	00FF00	Maroon	800000
Navy	000080	Olive	808000
Purple	800080	Red	FF0000
Silver	C0C0C0	Teal	008080
White	FFFFFF	Yellow	FFFF00

33- FONT5 SIZE SHORTCUTS

<big>, **<small>** Draws the enclosed text bigger or smaller than normal.

```
<html>
<head>
<title>Big and Small</title>
</head>

<body>
<p>
<big>HTMLEmentary - Big</big>
</p>
<p>HTMLEmentary - Normal size</p>
```

```
<p>
<small>HTMLEmentary - Small</small>
</p>

</body>
</html>
```

<big> and **<small>** are shortcuts for convenience.

Notes

- **<big>** is usually like ****.
- **<small>** is usually like ****.

34- TABLES

<table>, **<tr>**, **<td>** A simple table with three rows and two columns.

```
<html>
<head>
<title>Tables 1</title>
</head>
<body>

<table border>
<tr>
<td>Row 1, cell 1</td>
<td>Row 1, cell 2</td>
</tr>
<tr>
<td>Row 2, cell 1</td>
<td>Row 2, cell 2</td>
</tr>
<tr>
<td>Row 3, cell 1</td>
<td>Row 3, cell 2</td>
</tr>
</table>
```

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

The **border** attribute makes the outline of the table visible. It's not required. Each row has the same number of columns.

Notes

- Plan your table. At first you might want to draw it on paper.
- An HTML table is made of rows that contain cells. The important idea is that the cells are set up in the order they appear in your HTML.
- To Set Up a Table
 - Use a table tag to create the table.

```
<TABLE>
```

```
</TABLE>
```

- For each row include a table row tag. Here's a table with two rows.

```
<TABLE>
```

```
<TR>
```

```
</TR>
```

```
<TR>
```

```
</TR>
```

```
</TABLE>
```

- For each cell include a table data tag.

```
<TABLE>
```

```
<TR>
```

```
<TD>Row 1, cell 1</TD>
```

```
<TD>Row 1, cell 2</TD>
```

```
</TR>
<TR>
  <TD>Row 2, cell 1</TD>
  <TD>Row 2, cell 2</TD>
</TR>
</TABLE>
```

This table has two rows. Each row has two cells. I arranged the tags to make it easier to keep track of the data. You can arrange them anyway that's helpful. (Of course, the *order* of the tags is important.)

Think about a table as growing from left to right and top to bottom. Cells are added to a row from left to right until the row ends, and so on until there are no more rows.

Try This

Set up tables with different numbers of rows and columns until you get the hang of it. Pay attention to how the size of the cells changes with the amount of data you put in them. Save these tables for experimenting with in the next couple of topics.

35- TABLES SPANNING COLUMNS

`<td>`, `colspan` Spanning columns.

```
<html>
<head>
<title>Tables - colspan</title>
</head>
<body>

<table border>
  <tr>
    <td colspan="2">Row 1, cell 1 + 2 spanned</td>
    <td>Row 1, cell 3</td>
  </tr>
  <tr>
```

```
<td>Row 2, cell 1</td>
<td>Row 2, cell 2</td>
<td>Row 2, cell 3</td>
</tr>
<tr>
<td>Row 3, cell 1</td>
<td>Row 3, cell 2</td>
<td>Row 3, cell 3</td>
</tr>
</table>

</body>
</html>
```

In the table on the right, the first row has one fewer column than the others.

Notes

- To span columns add a **colspan** attribute to a table data tag.
- You can span columns anywhere in a table.
- Spanned columns don't have color added automatically; I added it to make the spanned cells stand out.
- You have to pay attention the number of cells in the row when spanning columns. To span two columns you use one fewer **<td>** in that row, to span three columns you use two fewer **<td>**'s in that row, and so on.

Try This

Set up tables with different numbers of rows and columns (or use the ones from the previous topic) and practice spanning columns until you get the hang of it.

36- TABLES SPANNING ROWS

<td>, **rowspan** Spanning rows.


```

<html>
<head>
<title>Tables - rowspan</title>
</head>
<body>

<table border>
  <tr>
    <td rowspan="2">Row 1+2, cell 1<br> spanned</td>
    <td>Row 1, cell 2</td>
    <td>Row 1, cell 3</td>
  </tr>
  <tr>
    <td>Row 2, cell 2</td>
    <td>Row 2, cell 3</td>
  </tr>
  <tr>
    <td>Row 3, cell 1</td>
    <td>Row 3, cell 2</td>
    <td>Row 3, cell 3</td>
  </tr>
</table>

</body>
</html>

```

In the table on the right, the first column has one fewer row than the others.

Notes

- To span rows add a **rowspan** attribute to a table data tag.
- You can span rows any place in a table.
- Spanned rows don't have color added automatically; I added it to make the spanned cells stand out.
- You have to pay attention the number of **<td>**'s in the **<tr>**'s following the one with the spanned rows. To span two rows you use one fewer **<td>** in the next row; to span three rows you use one fewer **<td>** in *each* of the next two rows, and so on.

Try This

Set up tables with different numbers of rows and columns (or use the ones from the previous topics) and practice spanning rows until you get the hang of it.

37- FRAMES 1 ROWS

<frameset>, **rows** **<frame>**, **src** Display multiple files in separate subwindows.

```
<html>
<head>
<title>Frames 1 - Rows</title>
</head>

<frameset rows="30%,*,30%">
  <frame src="fr1.html">
  <frame src="fr2.html">
  <frame src="fr3.html">
</frameset>

</html>
```

To get this sample to work you have to set up four files! One frameset file (the one above) and three regular Web pages. The frameset file controls the layout of the Web page.

Notes

- The **<frameset>** tag in the sample uses the **rows** attribute to create three frames. Three frames are created because three values are assigned to **rows**, "30%,*,30%". The first and third frames each take up 30% of the height of the whole window. The * means that the second frame gets whatever is left over, in this case 40%.
- The **<frame>** tags use the **src** attribute show which HTML files to load into each frame.

Details

- Don't use a **<body>** tag. See the sample above.
- The **rows** attribute creates frames that run from left to right, that is, as rows.
- You count the frames from top to bottom.
- The order of the frame tags matters. The first frame tag goes with the top frame, the second frame tag goes with the second tag, and so on.
- You can use a frameset tag in place of a frame tag (see the topics Frames 3 and Frames 4).
- There is no closing tag for the frame tag.
- The values in the **rows** attribute should add up to 100%.
- It's a good idea to use at least one *. When you use two or more, the remaining space is divided equally between them.

Try This

Change the height of the window and watch how the frames change.

Frames can become complicated for you to manage and cause navigational problems for your readers. Some search engines either can't or won't index them. Some older browsers don't support them. Unless frames offer some clear advantage over regular pages, you're probably better off not using them. On the other hand, frames can be used to present content in new, interesting or more informative ways.

38- FRAMES 2 COLS

<frameset>, **cols** Display multiple files in separate subwindows.

```

<html>
<head>
<title>Frames 2 - cols</title>
</head>

<frameset cols="30%,*,30%">

<frame src="fr1.html">

```

```
<frame src="fr2.html">
<frame src="fr3.html">

</frameset>
</html>
```

The only difference between this and the previous sample is the **cols** attribute.

Notes

This sample works the same as the last, except that the frames form columns instead of rows.

Details

- Don't use a **<body>** tag. See the sample above.
- The **cols** attribute creates frames that run from top to bottom, that is, as columns.
- You count the frames from left to right.
- The order of the frame tags matters. The first frame tag goes with the leftmost frame, the second frame tag goes with the second tag, and so on.
- You can use a frameset tag in place of a frame tag (see the topics Frames 3 and Frames 4).
- There is no closing tag for the frame tag.
- The values in the **rows** attribute should add up to 100%.
- It's a good idea to use at least one *. When you use two or more, the remaining space is divided equally between them.

Try This

Change the width of the window and watch how the frames change.

Tips

In this and the previous sample, we used percents for the **rows** and **cols** values, but you don't have to. You can use an

exact number of pixels, as in "200,25%,*". Here the first frame will be exactly 200 pixels, the second 25% of the window, and the last whatever is left over.

Percents are safer to use because you don't know the size of your reader's window. With percents, the browser can keep your frames in the same proportion. Fixed values are useful for areas that should not change size.

39- FRAMES 3

`<frameset>`, `rows`, `cols` Adding columns to a row.

```
<html>
<head>
<title>Frames - 3</title>
</head>
<frameset rows="25%,*">

  <frame src="fr1.html">
  <frameset cols="50%,50%">
    <frame src="fr2.html">
    <frame src="fr3.html">
  </frameset>

</frameset>
</html>
```

This sample displays a window with two rows. The second row is divided into two columns.

Notes

The first `<frameset>` uses a `rows` attribute with two values but has only one `<frame>` tag. **The second `<frame>` tag is replaced with a `<frameset>` tag.** This `<frameset>` tag divides the second row into two columns because its `cols` attribute has two values.

The **<frame>** tags within this nested **<frameset>** tag display a file in each of the two frames.

Details

- When a **<frameset>** tag replaces a **<frame>** tag, the new frames are created inside that frame.
- When you're finished, there should still be one **<frame>** tag for each frame.
- The **<frameset>** tag creates frames; the **<frame>** tag fills them.

Try This

1. Change the width and height of the window and watch what happens to the frames.
2. Copy the sample and get it to work. Then rewrite it so that the frame containing `fr3.html` is divided into two columns. (Hint: replace the last **<frame>** tag with a **<frameset>** tag.)

40- FRAMES 4

<frameset>, **rows**, **cols** Adding rows to a column.

```
<html>
<head>
<title>Frames - 4</title>
</head>
<frameset cols="25%,*">

  <frame src="fr1.html">
  <frameset rows="50%,50%">
    <frame src="fr2.html">
    <frame src="fr3.html">
  </frameset>

</frameset>
</html>
```

This sample presents a window with two columns. The second column is divided into two rows. The only difference between this and the previous sample is that the **rows** and **cols** attributes are reversed.

Notes

- The first **<frameset>** tag uses a **cols** attribute with two values but has only one **<frame>** tag. **The second <frame> tag is replaced with a <frameset> tag.** This **<frameset>** tag divides the second column into two rows because its **rows** attribute has two values. The **<frame>** tags within this nested **<frameset>** tag display a file in each of the two frames.

Details

- When a **<frameset>** tag replaces a **<frame>** tag, the new frames are created inside that frame.
- When you're finished, there should still be one **<frame>** tag for each frame.
- The **<frameset>** tag creates frames; the **<frame>** tag fills them.

Try This

1. Change the width and height of the window and watch what happens to the frames.
2. Copy the sample and get it to work. Then rewrite it so that the frame containing `fr3.html` is divided into two columns. (Hint: replace the last **<frame>** tag with a **<frameset>** tag.)

41- FRAMES 5

<noframes>, **scrolling**, **noresize** A more complicated frame example.

```

<html>
<head>
<title>Frames - 5</title>
</head>

<frameset rows="60,2*,*">

  <frameset cols="200,*">
    <frame scrolling="no" noresize src="htmltitl.html">
    <frame noresize src="htmltitl.html">
  </frameset>

  <frame src="i_ht0101.html">

  <frameset cols="50%,50%">
    <frame src="i_ht0101.html">
    <frame scrolling="no" src="i_ht0101.html">
  </frameset>

  <noframes>
<!-- For browsers without frames. -->
  </noframes>

</frameset>
</html>

```

This sample presents a window with three rows. The first and third rows are divided into two columns.

Notes

- The first `<frameset>` tag creates three rows. The first row is exactly 60 pixels tall. The second is twice as tall as the third (because of the 2*) and combined they fill up the remainder of the window.
- The second `<frameset>` tag breaks the first row into two columns, the first of which is exactly 200 pixels wide. The second column fills the remaining width.
- The second row is a plain `<frame>`.
- In the third row the final `<frameset>` tag creates two equal size columns.

- When a plain number is used with rows or cols, it means an exact number of pixels.
- There are five <frame> tags on the left and five <frames> on the right.
- The noresize attribute prevents the user from changing the size of a frame.
- Setting scrolling to "no" removes any scroll bars. Now the user cannot scroll the contents of the frame. The default value for scrolling is "yes"
- Compare the use of noresize and scrolling in the frame tags with the appearance of the frames.

noframes

Some browser doesn't support frames. At least one browser allows you to turn frames off. You should always include a <noframes> section in your frameset pages. In it either display a message stating that the content contains frames and can't be displayed in the current browser, or create a separate version of your page that does not use frames. Note that the <noframes> tag is inside of the outermost <frameset> tag.

Try This

Try changing the size of all the frames.

42- FRAMES 6

<frame>, **name** <a>, **target** Changing the contents of a frame from another frame.

```
<html>
<head>
<title>Frames - 6</title>
</head>
<frameset rows="75,*">
  <frame src="title.html">
</frameset cols="25%,*">
```

```
<frame src="menu.html">
<frame name="rframe" src="right1.html">
</frameset>
</frameset>

</html>
```

The sample has three frames. The title spans the top. The frame labeled **Menu** holds links that change the contents of the right frame. The frame on the lower right changes as you click the links from the left frame.

Click each of the menu choices. The only frame that changes is the lower right one.

Notes

- The lower right frame is named **rframe** using **name="rframe"** in the third **<frame>** tag. The right frame can now be *targeted* from other frames.
- In menu.html the important lines are

```
<a href="right1.html" target="rframe">One</a>
<a href="right2.html" target="rframe">Two</a>
<a href="right3.html" target="rframe">Three</a>
```

Usually, when you click a link in a frame the new file loads into that frame. If you use a target attribute in the link, the file loads into the frame pointed to by the target attribute.

You can use any html files fr menu, title, right1,....

43- FRAMES 7 NESTED FRAMES

<frame>, **name** **<a>**, **target** Changing the contents of two frames using nested frames.

```
<html>
<head>
<title>Frames - 7</title>
</head>
<frameset rows="75,*">

<frame src="title.html">
<frame name="fr" src="firstframe.html">

</frameset>
</html>
```

In the sample above there are only two **<frame>** tags, but in the output on the right there are **three** frames! This Web page looks like the one in **Frames 6**, but it's put together a different way. Click each of the *next* and *previous* choices. Notice that **both** the left and right frames change.

Notes

The frameset document above sets up a window with **two** frames. The second **<frame>** tag names the lower frame **fr** and loads into it **another frameset document, firstframe.html** (instead of a regular HTML document). This splits the lower frame into two frames. This technique is called *nesting* frames.

In **firstframe.html** HTML files are loaded into the left and right frames:

```
<frameset cols="35%,*">
  <frame src="left1.html">
  <frame src="right1.html">
</frameset>
```

To change **both** frames at the same time, we use a link to another frameset document, **secondframe.html**, targeted to **fr** as we did in **firstframe.html**:

`` For each set of of frames you want to update at the same time you need another frameset document.

Try This

- Rewrite the sample using only a left and right frame. (Easy)
- Rewrite the sample so that the links are on the right side.
- Rewrite the sample so three frames change at the same time.

44- FRAMES onClick

`<a>`, **onClick** Changing the contents two frames.

```
<html>
<head>
<title>Frames - 8</title>
</head>

<frameset rows="75,*">

  <frame name="title" src="title.html">
  <frameset cols="35%,*">
    <frame name="left" src="onleft1.html">
    <frame name="right" src="right1.html">
  </frameset>

</frameset>

</html>
```

This frameset document sets up a window with three frames. Clicking **Back** and **Next** in the output window causes both the left and right frames to change.

Notes

All of the frames are named using the name attribute. In onleft1.html the important line is

```
<a href="onleft2.html" onClick= "parent.right.location.href =  
'right2.html'"> Next </a>
```

The link contains a JavaScript *event handler*, **onClick**. An event handler is some code that responds to an event, such as clicking or moving the mouse around. When a link containing **onClick** is clicked, the link is activated the usual way and the code listed after **onClick** is executed. In this case, a new file is displayed in the frame that contains the link and in the frame named **right**. Note the use of single quotation marks around the file name within the event handler.

In the script following **onClick**, **parent** is the frameset that contains the frame the **onClick** is in. **right** is the name of the frame within this frameset that we want to put the new file into. **parent.right.location.href** is where the name of the file displayed in the right frame is stored. Assigning a URL to **parent.right.location.href** causes the file at that URL to be displayed. (**parent.right.location** works the same way here.)

45- FRAMES NESTED FRAMES 2

<frame>, **name <a>**, **target** Hiding and restoring frames.

```
<html>  
<head>  
<title>Frames - 9</title>  
</head>  
<frameset rows="75,*">  
  
<frame src="titlebar.html">  
<frame name="fr" src="changeframe.html">  
  
</frameset>  
</html>
```

Click each of the links in the top frame at the right. The lower frames seem to disappear and the reappear.

Notes

The frameset document above sets up a window with **two** frames. The second **<frame>** tag names the lower frame **fr** and loads into it another frameset document, `changeframe.html` (instead of a regular HTML document). This splits the lower frame into two frames. This technique is called *nesting* frames.

In `changeframe.html` HTML files are loaded into the left and right frames:

```
<frameset cols="35%,*">
  <frame src="left.html">
  <frame src="right.html">
</frameset>
```

Clicking the links labeled **Left** and **Right** in the top frame loads a regular HTML file into the lower frame, so the nested frames disappear. Clicking the link labeled **Both** reloads the original file that contains a **<frameset>**, so the frames reappear. The important lines from `titlebar.html` are

```
<a href="left.html" target="fr">left</a>
<a href="changeframe.html" target="fr">both</a>
<a href="right.html" target="fr">right</a>
```

Each of the links targets the frame named **fr**. You can put regular files or frameset files into a frame.

46- FORMS

<form>, **action**, **method**, **post**
<input>, **type**, **name**, **size**
<input>, **text**, **submit**, **rest**

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

<h1>Sign up!</h1>

<form action="/cgi-bin/formtest.pl" method="post">

<p>
<input type="text" name="last_text"><br>
Last Name<br>
<input name="first_text" size="30"><br>
First Name<br>
</p>
<p>
<input type="submit" name="button_submit">
<input type="reset">
</p>

</form>

</body>
</html>
```

47- FORMS

<input>, type, checkbox, value

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

```

<h1>Options</h1>

<form action="/cgi-bin/formtest.pl" method="post">
<p>Check all that apply.</p>
<p>
<input type="checkbox" name="check1">
the first checkbox.<br>
<input type="checkbox" name="check1"
value="secondcheckbox">
the second checkbox.<br>
<input type="checkbox" name="check2"
value="thirdcheckbox">
the third checkbox.
</p>
<p>
<input type="submit" name="button_submit">
<input type="reset" value="clear">
</p>

</form>

</body>
</html>

```

48- FORMS

<input>, type, radio, checked

```

<html>
<head>
<title>Forms</title>
</head>
<body>

<h1>Choose</h1>

<form action="/cgi-bin/formtest.pl" method="post">

```



```

<p>
<input type="radio" name="yesnomaybe" value="yes"
checked>
Yes. <br>
<input type="radio" name="yesnomaybe" value="no">
No. <br>
<input type="radio" name="yesnomaybe" value="maybe">
Maybe.
</p>
<p>
<input type="submit" name="button_submit">
</p>
</form>

</body>
</html>

```

49- FORMS

<input>, type, hidden

```

<html>
<head>
<title>Forms</title>
</head>
<body>

<h1>Eyes Only</h1>

<form action="/cgi-bin/formtest.pl" method="post">

<p>
<input type="hidden" name="secret" value="hidden_value">
<input type="hidden" name="secret2" value="treasure">
</p>
<p>

```

```
<input type="submit" name="button_submit">
</p>

</form>

</body>
</html>
```

50- FORMS

<input>, type, password

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

<h1>Shh</h1>

<form action="/cgi-bin/formtest.pl" method="post">
<p>
Enter the password:<br>
<input type="password" name="pw">
<p>
</p>
<input type="submit" name="button_submit">
</p>
</form>

</body>
</html>
```

51- FORMS

<input>, type, image

```
<html>
```

```
<head>
<title>Forms</title>
</head>
<body>

<h1>Click Me!</h1>

<form action="/cgi-bin/formtest.pl" method="post">

<p>
<input type="image" name="monster" src="monster.gif"
width="47" height="47" border="0">
</p>
</form>
</body>
</html>
```

52- FORMS

<select>, name

<option>, value, selected

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

<h1>Dropdown List</h1>

<form action="/cgi-bin/formtest.pl" method="post">

<select name="dropdown">

<option>First</option>
<option value="2">Second</option>
<option>Third</option>
<option selected>Fourth</option>
```

```
<option>Fifth</option>
<option>Sixth</option>
<option>Seventh</option>
<option>Eighth</option>

</select>
<p>
<input type="submit">
<input type="reset">
</p>

</form>

</body>
</html>
```

53- FORMS

<select>, name, size
<option>, value, selected

```
<html>
<head>
<title>Forms - Select List</title>
</head>
<body>

<h1>Select List</h1>

<form action="/cgi-bin/formtest.pl" method="post">

<select name="list" size="5">

<option>First</option>
<option value="2">Second</option>
<option>Third</option>
<option selected>Fourth</option>
```

```
<option>Fifth</option>
<option>Sixth</option>
<option>Seventh</option>
<option>Eighth</option>

</select>
<p>
<input type="submit">
<input type="reset">
</p>

</form>

</body>
</html>
```

54- FORMS

<select>, name, size, multiple
<option>, value, selected

```
<html>
<head>
<title>Forms - Multiple Select List</title>
</head>
<body>

<h1>Multiple Select List</h1>

<form action="/cgi-bin/formtest.pl" method="post">

<select name="list" multiple size="5">

<option>First</option>
<option value="2">Second</option>
<option>Third</option>
<option selected>Fourth</option>
<option>Fifth</option>
```

```
<option>Sixth</option>
<option>Seventh</option>
<option>Eighth</option>

</select>
<p>
<input type="submit">
<input type="reset">
</p>

</form>

</body>
</html>
```

55- FORMS

<textarea>, rows, cols, name

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

<h1>Type!</h1>

<form action="/cgi-bin/formtest.pl" method="post">

<p>
Enter some text<br>
<textarea rows="6" cols="25" name="text1">
Type something here...
</textarea>
</p>

<p>
```

```
<input type="submit" name="button_submit">
<input type="reset">
</p>

</form>

</body>
</html>
```

56- FORMS

<input>, **type**, **radio**

Updating frames with radio buttons. This is an example of using forms without CGI.

```
<html>
<head>
<title>Updating Frames with Radio Buttons</title>
</head>

<frameset rows="75,*">

  <frame src="title.html">
  <frameset cols="35%,*">
    <frame src="radiomenu.html">
    <frame name="rframe" src="right1.html">
  </frameset>

</frameset>

</html>
```

57- RUNNING JAVA APPLETS 1

<applet>, **code**, **width**, **height** Running Java applets.

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

```
<title>Java Applets 1</title>
</head>
<body>

<applet code="NervousText.class" width="200" height="50">
Your browser cannot run
Java applets!
</applet>

<p>
This Java applet was
written by Daniel Wyszynski.
</p>

</body>
</html>
```

The **<applet>** tag runs a Java applet on a Web browser that supports Java. Any HTML within the **<app>**let tags is ignored.

On a browser that does not support Java the **<applet>** tags are ignored and any HTML within them is displayed.

Notes

- The **<applet>** tag must include the **code**, **width** and **height** attributes.
- **code** gives the name of the class file that contains the Java applet. **width** and **height** specify the width and height of the applet in pixels.

Details

- A Java applet can have many class files.
- The class files must be in the same directory as the HTML file.

58- BACKGROUND MUSIC

Displaying background music with <EMBED>.

```
<html>
<head>
<title>Background Music</title>
</head>
<body>

<p>
This might take a while to load.
</P>

<embed src="bjogpron.mid" hidden="true">

<p>
Make sure the volume is turned up.
</p>

</body>
</html>
```

The <**embed**> tag embeds a document in a Web page. The browser then uses the appropriate application or plug-in to display (or play, etc.) the document.

Notes

- Although <**embed**> is supported by many browsers, it is not part of any HTML standard.
- Even though the tag has the **hidden** attribute set to true, it still takes up space on the page.
- Background music can be very annoying.

HTML Project 1

In this Project, you will create three linked web pages on you and your interests. Please do the HTML tagging by hand, using NotePad or EditPlus. Do not use an HTML editor? You will need to write the code yourself in Notepad, or EditPlus. (If you use EditPlus, do not use any of the creation options.) **If you use an HTML editor, you will receive 0 (zero) points for the project.**

You will be creating a total of three linked pages. You must be able to move from one to any of the others without returning to the home page or using the back button on the browser. (You will have some sort of consistent, logical links area - sidebar or header.)

Page 1- Home page: Create a webpage which explains who you are. It should contain the following topics, reflecting your personality:

- interests
- hobbies
- favorites (or current) reading list
- favorite movie list
- current academic
- social interests and activities

Save this page as index.html

Page 2 - Schedule page: Using tables, create your current work and school schedule. This page should include the same (or similar) elements of good navigation and clear links to other portions of your webpage.

Save this page as schedule.html

Page 3 - Interests page: The third page containing your ten favorite websites with a short paragraph about them (two sentences). This page should include the same (or similar) elements of good navigation and clear links to other portions of your webpage.

Save this page as interests.html

All data should be marked up in a clean presentation using elements that show off your skill in creating tables, lists, formatting of text and designing style wisdom. All work should be hand coded. You may use ideas from other web pages and learn from their code but you must hand code every tag in this exercise.

Note: all pages must be hand coded, not using html WYSIWIG program. Use of MS Word, FrontPage, DreamWeaver, InDesign, or similar program will result in a grade of ZERO

Exercise 5

1. What does **HTML** stand for? Answer
2. Write an HTML comment which contains the text "Web design is fun!".
3. Write the code to display the characters "©", "®", "&", "<". Answer
4. What tag encloses the text that will be displayed in the browser's title bar?
5. What tag contains what will be seen on the browser page?
6. How many heading-style tags are there? Which one displays the largest font size?
7. What tag lets you superscript text?
8. What attribute of the *body* tag lets you set the background color?
9. Write the proper HTML code needed to insert an image into a web page with the following characteristics: The image is named ImageNo1.jpg and is located in the *images* subfolder. The alternate text should be: "Picture of ImageNo1, 2010". There should be a margin of 10 pixels on both the left and right sides. There should be a margin of 5 pixels on both the top and bottom. The dimensions of the image should be specified as 305 pixels tall and 417 pixels wide. The image should also be centered on the page.
10. Write the *proper HTML* code to create a link to a page named **info.html**. The link should contain the text **Product Information**. Assume that both pages are in the same directory.

11. Write the proper HTML code to create a link to the external website www.yahoo.com. The link should contain the text Yahoo! and it should open up a new window.
12. Write the proper HTML code to create a named anchor which contains the text Address List. The anchor should have the name/id addr.
13. Write the proper XHTML code to create a link to a named anchor with the name addr. Assume the named anchor and the link are on the same page. The link should contain the text Members.
14. Create a table using proper HTML that does the following: The table should be as wide as the browser window and have a border 3 pixels wide. The table should have three rows and three columns. The first row should be 35 pixels high and contain one cell that spans three columns, is a heading cell, and contains the text "First row" aligned to the left. The second row should contain three cells. The first and third cells should both span two rows and each should take up 40% of the width of the table. They should have the background colors blue and brown, respectively, and both should contain the text "Kish" centered horizontally and aligned vertically to the top. The second cell in the second row should be 30 pixels tall and contain the text "40". The second cell in the third row should be 40 pixels tall and contain the text "years".
15. What attribute of the `td` and `th` tags lets you set how tall they are?
16. What attribute of the `td` and `th` tags lets you set their background color?
17. What attribute of the `table` tag lets you set how much space there is between cells?
18. What is the difference between the `td` and `th` tags?

19. Figure 1 shows an HTML page `faq.html`. Question 1, Question 2, and Question 3 are HTML links. When one of the links is clicked, the first line of the correspondent answer will be displayed in the SAME browser window within the SAME HTML file. For example, figure 2 demonstrates the result after the link Question 2 is clicked. Write the correspondent HTML file (tip: use jumping targets).

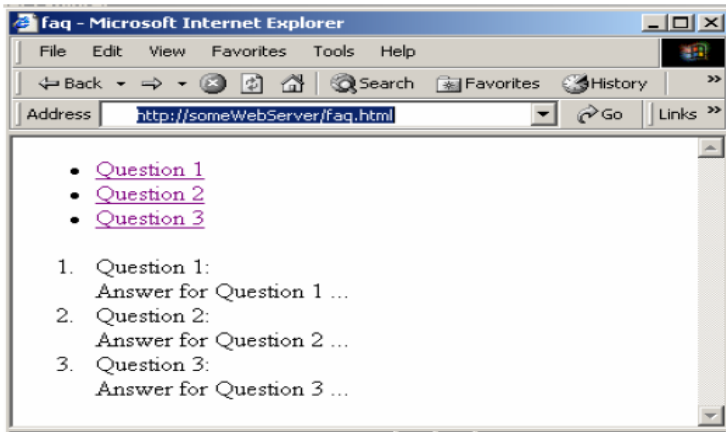


Figure 1

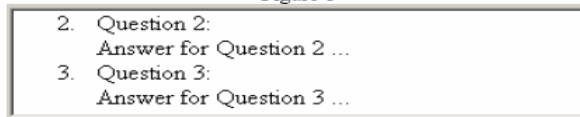


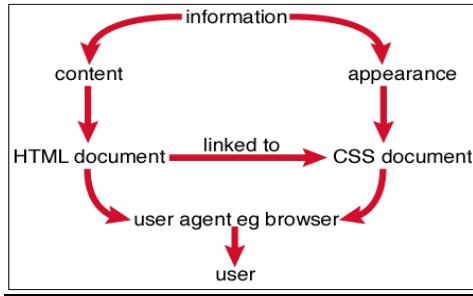
Figure 2

Chapter 5 Laboratory 2 CSS Cascaded Style Sheet

HTML contained tags to indicate how to render pages: tags for structure and tags for style. CSS is a style sheet language used to describe the presentation semantics (that is, the look and formatting) of a document written in HTML. CSS is an excellent addition to plain HTML. The idea of style sheets is to separate page content and page style such as (text vs. color, font, format, etc.). So How do style sheets work?

You can think of a style sheet as a set of instructions, suggesting to a web browser how to draw a page. The following figure shows the process that relate HTML to CSS:

- 1- Assume that HTML and CSS are two separate documents that contain certain information.
- 2- CSS document contains information related to the text appearance while HTML contains information related to content”.
- 3- CSS and HTML documents are linked together in order to be viewed in the browser by the user”



Cascaded Style Sheets Rules

CSS has a simple syntax and uses a number of English keywords to specify the names of various style properties. A style sheet consists of a list of rules to customize HTML elements. Style rules can be: external (imported), inline (within an HTML element), or embedded (declared in a document). They could be different styles applied for the same text section. There are general rules for determining the precedence (cascading) of the styles.

Specifying Style Rules

The syntax for specifying style properties is:

selector {property : value}

or

*selector {property1 : value1;
property2: value2;*

...

propertyN : valueN}

Each rule consists of one or more *selectors*

- Each rule consists of one or more *selector* and a *declaration block*
- A declaration-block consists of a list of *declarations* in braces.
- Each declaration itself consists of a *property*, a colon (:), a *value*, then a semi-colon (;)

There are many properties pertaining to: font, size, color, background, margins, borders, width, height, alignment, text appearance, etc. (and even position as we shall see later).

Style Sheets Advantages

- 1) Separation of text content and displaying style
- 2) Possibility to create external style templates
- 3) Consistent rendering of style throughout site
 - Can be written so the user will only need to download it once in the external style sheet document. When surfing the rest of your site the CSS will be cached on the users computer, and therefore speed up the loading time.
- 4) No need for new HTML tags for new styles
- 5) Offers much more detailed attributes than plain HTML for defining the look and feel of your site.

To define a style Rule, there is the tag `<STYLE>` that allows the definition of formatting rules. Style tag has a type attribute to indicate the style type used. When using style sheet type = "text/css"

```
<STYLE type = "text/css">
<!--
    Style rules
-->
</STYLE>
```

For example

Property Value

```
<STYLE>
<!--
    BODY {font: 12pt Helvetica; color:blue; margin-left: 0.5in}
```



```
H1 {font: 18pt Palatino; color: red}
H2 {font-family: MeppDisplayShadow}
KBD {text-decoration: underline}
-->
</STYLE>
```

No let us talk about the Cascaded Style sheet Syntax and its elements.

Selectors

- Selectors are the names that you give to your different styles.
- In the style definition you do the following two steps:

- 1- Define how each selector should work (font, color etc.).
- 2- Then, in the body of your pages, you refer to these selectors to activate the styles. For example:

```
<html>
<head>
<style type="text/css">
  b.headline {color:red; font-size:22px; font-family:arial;
  text-decoration:underline}
</style>
</head>
<body>
<b>this is normal bold</b><br>
<b class="headline">this is headline style bold</b>
</body>
```

```
</html>
```

Selectors are divided into three types:

- HTML selectors which are used to define styles associated to HTML tags.
- Class selectors which are used to define styles that can be used without redefining plain HTML tags.
- ID selectors which are used to define styles relating to objects with a unique ID (most often layers).


Let us now talk about each of those selector types in more details:

1- HTML (tag) selectors: HTML tags selectors are used to redefine the look of tags. HTML tags have the following syntax:

- HTMLSelector {Property:Value;}

The following example associate new styles to tag where it:

- changes font-family into “arial”
- Make the font size = 14
- Change the default color to be red



```
<html>
<head>
<style type="text/css">
  B {font-family:arial; font-size:14px; color:red}
</style>
</head>
<body>
  <b>This is a customized headline style bold</b>
</body>
</html>
```

2- Class selectors

Class Selectors have the following syntax:

- .ClassSelector {Property:Value;}

Class selector can be used either to :

- 1) To specify a style for a group of elements. for example the following statement make all HTML elements with class="center" center-aligned:

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

- 2) To specify that only specific HTML elements should be affected by a class. For example, the following statement make all p elements with class="center" center-aligned :

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

Before checking out the syntax of ID selector, let us introduce tags `` and `<div>` which are used by the ID selectors

- The `` tag provides no visual change by itself.
- The `` tag provides a way to add a hook to a part of a text or a part of a document.
- When the text is hooked in a span element you can add styles to the content, or manipulate the content with for example JavaScript.
- The `<div>` tag defines a division or a section in an HTML document.
- The `<div>` tag is often used to group block-elements to format them with styles.

3) ID Selectors

- ID Selectors have the following syntax:
 - `#IDSelector {Property:Value;}`
- The ID selector is used to specify a style for a single, unique element.
- The style rule below will be applied to the element with `id="para1"`:

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

Let us define two layers, Layer1 and Layer2, which has the following properties:

- 1- Its position type is absolute
- 2- Its left and top coordinates are 100, 100 and 140, 140 consequently
- 3- Z-index value is 1. This property is used to specify the stack order of an element. An element with greater stack order is always in front of an element with a lower stack order.

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

    #layer1 {position:absolute; left:100;top:100; z-Index:0}
    #layer2 {position:absolute; left:140;top:140; z-Index:1}

</style>
</head>

<body>

<div ID="layer1">
<table border="1" bgcolor="#FFCC00"><tr><td>THIS IS LAYER
1 <br>POSITIONED AT 100,100</td></tr></table>
</div>

<div ID="layer2">
<table border="1" bgcolor="#00CCFF"><tr><td>THIS IS
LAYER2 <br>POSITIONED AT 140,140</td></tr></table>
</div>

</body>
</html>
```

Layer1 and layer2 are applied in two divisions positioned at locations defined with the coordinates in the Style rule. Note that div (also spam) tags are used in association with the ID

CSS IDs are similar to classes in that they define a special case for an element. Below is an example of a CSS ID.

For example, in the following code we change either the background color or the text transform for the text written using <p> tag by associating an ID selector (either exampleID1 or examplID2)

```
<HTML>
<HEAD>
<TITLE> New Document </TITLE>
<Style>
  p#exampleID1 { background-color: red; }
  p#exampleID2 { text-transform: uppercase; }
</style>
</HEAD>
<BODY>
  <p id="ExampleID1">This paragraph has an ID name of
  "exampleID1" and has a white CSS defined background</p>
  <p id="ExampleID2">This paragraph has an ID name of
  "exampleID2" and has had its text transformed to uppercase
  letters. </p>
</BODY>
</HTML>
```

ExampleID1 style is applied to text associated the first <p> tag
ExampleID2 style is applied to text associated to the second <p>

Grouped selectors

Most often selectors will share some of the same style such as being based on the same font. In these cases, we can assign the font to all the selectors at once using a group selector technique. For example, assume we have the following class style rules, headlines, sublines and infotext.

```
.headlines{  
font-family:arial; color:black; background:yellow; font-size:14pt;  
}  
  
.sublines {  
font-family:arial; color:black; background:yellow; font-size:12pt;  
}  
  
.infotext {  
font-family:arial; color:black; background:yellow; font-size:10pt;  
}
```

It is clear that all three classes share the same properties except the font size. So, By using group selectors technique, we can group common properties rewrite the previous style rules as follow:

Group common properties together

```
.headlines, .sublines, .infotext {  
font-family:arial; color:black; background:yellow;  
}  
  
.headlines {font-size:14pt;}  
.sublines {font-size:12pt;}  
.infotext {font-size: 10pt;}
```


Context dependant selectors

Sometimes it is desired to use a group of selectors with certain context (Context dependant selectors). For example, the following style rule is applied only when TAGS <I> and comes together with a certain order (context dependent).

So when the following elements comes in the html body,

- `<i>example</i>`

The following rule is applied:

```
I B {font-size:16px;color:red}
```

However, the rule is not applicable to the following element because and <i> are reversed in order:

- `<i>example</i>`

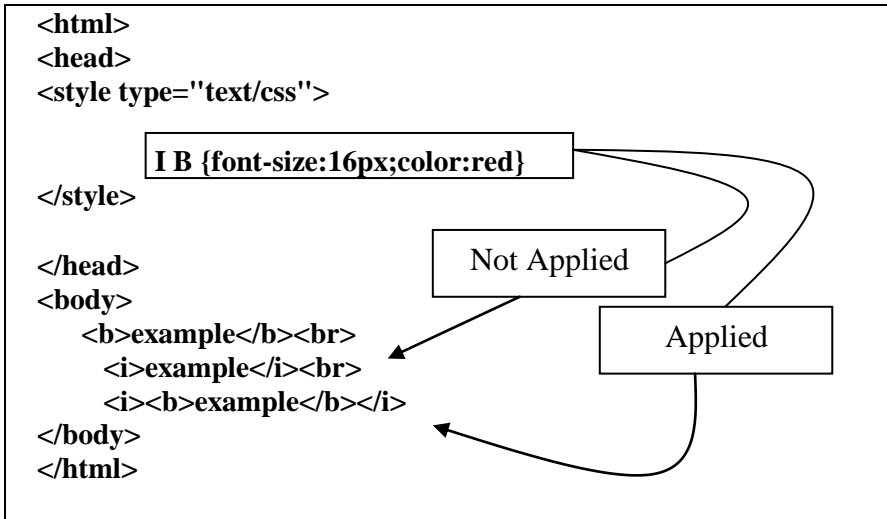
Context selectors Example

The following example shows some examples with their description:

Example	Description
<code>DIV P { font: small sans-serif }</code>	have some form of special formatting for <p> tag inside <div> tag
<code>.reddish H1 { color: red }</code>	have some form of special formatting for H1 inside .reddish class
<code>DIV.sidenote H1 { font-size: large }</code>	have some form of special formatting for H1 inside

	.sidenote class inside DIV tag
--	--------------------------------

Tryout this example to check the output when using context dependent selectors



Sometimes it is desirable to use both grouped and context dependent selectors at the same time. For example, In the following rule:

```

I B, .headlines, B .sublines {font-size:16px;}

```

Which indicate that the font-size of 16 pixels is in effect on:

- All tags enclosed by <I> tags
- All headlines classes.
- sublines classes enclosed by tags.

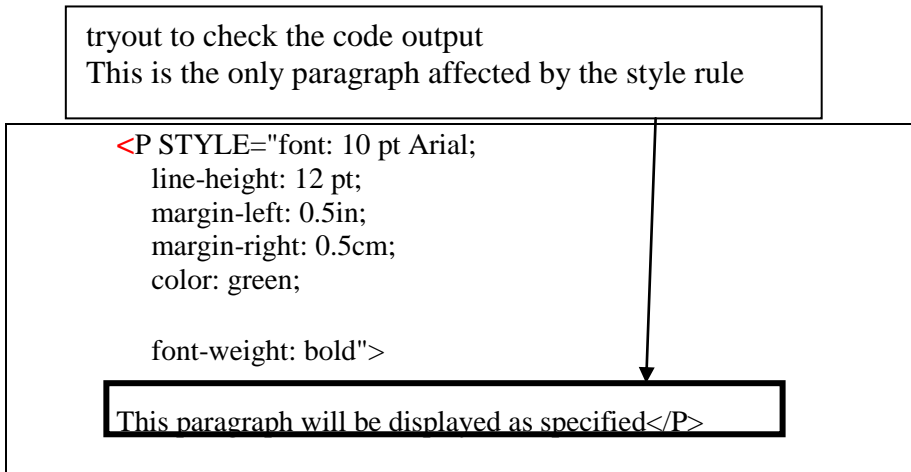
Where to place CSS

There are three ways of inserting a style sheet:

- 1) Inline Style Sheets
- 2) Embedded Style Sheets
- 3) External Style Sheets

Inline Style Sheets

In inline Style Sheets, we can create style rules within a document directly inside an HTML element tag (certain section). The formatting rules apply to that section.



For inline Style Sheets,, it is recommended to limit your use of single tag CSS. If you define your styles for each and every tag they're used on, you will lose much of the power associated with CSS. Furthermore, if you wanted to change a certain style, you'd have to change it all over in your document, rather than in one place.

Embedded Style Sheets

In this type, we can add style information in the document HEAD. The formatting rules apply for the whole document. In the following Example, we embedded two styles for H1, H2 and H3 in the Head of the document. Those styles could be applied to any section of the document Body.

```
<HTML>
<HEAD>
<TITLE>Style Sheets Demo 1</TITLE>
<STYLE>
  H1 {text-align:center; color:blue; font-family:Arial}
  H2, H3 {text-decoration:underline; font-style:italic}
```

```
</STYLE>
</HEAD>
```

Embedded Style Sheets (single page)

In this type, we can add style information in the document HEAD. The formatting rules apply for the whole document. In the following Example, we embedded two styles for H1, H2 and H3 in the Head of the document. Those styles could be applied to any section of the document Body.

```
<HTML>
<HEAD>
<TITLE>Style Sheets Demo 1</TITLE>
<STYLE>
  H1 {text-align:center; color:blue; font-family:Arial}
  H2, H3 {text-decoration:underline; font-style:italic}
</STYLE>
</HEAD>
```

Here is a complete example:

```
<html>
<head>
<title>MY CSS PAGE</title>
<style type="text/css">
  .headlines, .sublines, infotext {font-face:arial; color:black;
background:yellow; font-
weight:bold;}
  .headlines {font-size:14pt;}
  .sublines {font-size:12pt;}
  .infotext {font-size: 10pt;}
</style>
</head>
<body>
<span class="headlines">Welcome</span><br>
<div class="sublines">
  This is an example page using CSS.<br>
  The example is really simple,<br>
  and doesn't even look good,<br>
  but it shows the technique.
</div>
```

```

<table border="2"><tr><td class="sublines">
  As you can see:<br>
  The styles even work on tables.
</td></tr></table>
<hr>
<div class="infotext">
  Example from yahoo.Com.
</div>
<hr>
</body>
</html>

```

External Style Sheets (Entire site)

An external style sheet is ideal when the style is applied to many pages (entire web site). With an external style sheet,

- 1- You can change the look of an entire Web site by changing one file.
- 2- Each page must link to the style sheet using the <link> tag.
- 3- The <link> tag goes inside the head section:

```

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

```

An external style sheet can be written in any text editor. The file should not contain any html tags. Your style sheet should be saved with a .css extension. An example of a style sheet file is shown below:

```

hr {color:sienna}
p {margin-left:20px}
body {background-image:url ("images/back40.gif")}

```

External Style Sheet Example:

The following html file is designed to use the external style sheet whatevee.css

```

<html>
<head>
  <title>MY CSS PAGE</title>
  <link rel=stylesheet href="whatever.css" type="text/css">
</head>
<body>
<span class="headlines">Welcome</span><br>

<div class="sublines">
  This is an example of a page using CSS.<br>
  The example is really simple,<br>
  and doesn't even look good,<br>
  but it shows the technique.
</div>

<table border="2"><tr><td class="sublines">
  As you can see:<br>
  The styles even work on tables.
</td></tr></table>

<hr>

<div class="infotext">Example from Yahoo.Com.</div>

<hr>
</body>
</html>

```

The following is the style sheet file whatever.css.

```

.headlines, .sublines, infotext {font-face:arial; color:black;
background:yellow; font-weight:bold;}
.headlines {font-size:14pt;}
.sublines {font-size:12pt;}
.infotext {font-size: 10pt;}

```

Applying styles for non tag contained text?

When we want to apply a style to part of a document that is not contained between an opening and closing tag, we can use the

- ` ... ` tag.

In the following example, the style is applied only on the first two items of the ordered list, in other words on section enclosed in the span tag.

```
<OL TYPE=A>
```

```
<SPAN STYLE="font-style:italic; color:red">  
<LI> my first element  
<LI> my second element  
</SPAN>
```

```
<LI> this element is normal  
</OL>
```

CSS Text

The following elements are used for formatting text:

Font properties

```
B {font-family:arial, helvetica; font-size:12px; font-weight:bold;}
```

Which is equivalent to:

```
B {font:arial, helvetica 12px bold}
```

The following attributes are used for changing Text properties

1) line-height :

- 1.5 lines spacing (using the current font size).

2) Text-transform:

- Capitalize sets the first letter of each word in uppercase.
 - Uppercase forces all letters to uppercase.
 - Lowercase forces all letters to lowercase.
- 3) text-indent :
- Use this to indent the first word of a paragraph.
- 4) Color:
- B {font:arial, helvetica 12px bold; color:red}
- 5) white-space :
- pre the browser will show all spaces in the text,
 - This is similar to the <pre> tag in plain HTML.

Example

CSS rule and its use

```
<html>
<head>
  <style type="text/css">
    .headline {font-family:arial; font-size:14px; color:red; line-height: 4 px; Text-transform:capitalize}
  </style>
</head>
<body>
  <b class="headline">This is a bold tag carrying the headline class</b>
  <br>
  <i class="headline">This is an italics tag carrying the headline class</i>
</body>
```

```
</html>
```

CSS Colors

With CSS, you can define an area to have a specific color without that area being part of a table. With CSS you can simply refer to a certain class in your `<TD>` tags. For example, the following code change the background color for the table cell to orange

```
<TABLE style="background-color:orange" border="1">  
<TD> Pumpkin Background! </TD>  
</TABLE>
```

AAAGH! Pumpkin Background!

CSS Colors properties

There are a lot of CSS properties that affect coloring schemas.

Basically there are three color options with CSS:

- Setting the foreground color for contents
- Setting the background color for an area
- Setting a background image to fill out an area

For example:

- 1) The background-color property sets the background color of an element. For example:

Set Document background to green color

```
BODY { background-color: green }  
H1 { background-color: orange }
```

Set Background color for `<H>` tag text to orange

- 2) The backgroundimage property sets the background image of an element and the background position sets the x and y coordinate for the image position: For example:

Set the background image to myimage.gif on position (75,75)

```
BODY {backgroundimage:url(myimage.gif); background-position: 75px 75px;}
```

```
BODY {background-image:url(myimage.gif); background-attachment: fixed;}
```

Set the background image to mayimage.gif and The **background-attachment** property determines if a specified background image (myimage.gif) will scroll with the content or be fixed with regard to the canvas. This example specifies a fixed background image. The default value of this property is scroll, which allow cbackground to scroll with the contents..

CSS Colors properties (Example)

The following example defines a class selector “myclass” which have the several CSS properties such as:

In this statement the properties are:

- 1- Foreground color = red
- 2- Background color = blue

In the second statement

```
.myclass { color:red; background-color:blue;}
```

```
.myclass { color:#000000; background-color:#FFCC00;}
```

```
.myclass { color:rgb(255,255,204); background-color:rgb(51,51,102);}
```

In this statement the properties are:

- 3- Foreground color code = “#000000”
- 4- Background color = “:#FFCC00”

In this statement the properties are:

- 1- R G B values for the Foreground color = rgb(255,255,204)
- 2- R G B values for Background color = rgb(51,51,102)

CSS Links

CSS has several options for redefining the style of links. The following table shows a list, value and description for properties that affect links styles.

Property	Value	Description
A:link	<style>	Defines the style for normal unvisited links.
A:visited	<style>	Defines the style for visited links.
A:active	<style>	Defines the style for active links. A link becomes active once you click on it.
A:hover	<style>	Defines the style for hovered links. A link is hovered when the mouse moves

over it.

CSS Links Example:

The following example shows different properties that affect link style.

For Class1 selector,

Set the text-decoration style for:

- 1) A:link property to none,
- 2) A:visited property to none
- 3) A:active property to none and
- 4) A: hover to underline and red color (when moving over text affected by class 1, its appearance change to red underline.

A

<style type="text/css">

```
.class1 A:link {text-decoration: none}
.class1 A:visited {text-decoration: none}
.class1 A:active {text-decoration: none;}
.class1 A:hover {text-decoration: underline; color: red;}
```

```
.class2 A:link {text-decoration: underline overline}
.class2 A:visited {text-decoration: underline overline}
.class2 A:active {text-decoration: underline overline}
.class2 A:hover {text-decoration: underline; color: green;}
```

```
.class3 A:link {background: #FFCC00; text-decoration: none}
.class3 A:visited {background: #FFCC00; text-decoration: none}
.class3 A:active {background: #FFCC00; text-decoration: none}
.class3 A:hover {background: #FFCC00; font-size:30; font-weight:bold; color: red;}
```

</style>

For Class2 selector,

Set the text-decoration style for:

- 1) A:link property to overline,
- 2) A:visited property to overline
- 3) A:active property to overline
- 4) A: hover to underline and green color

For Class3 selector,

Set the text-decoration style for:

- 1) A:link property to none with content background color #FFCC00,
- 2) A:visited property to none with content background color #FFCC00
- 3) A:active property to none with content background color #FFCC00
- 4) A: hover to underline and red color, background color #FFCC00, font weight 30.

A

CSS List

With CSS, HTML lists can be styled further, and images can be used as the list item marker.

CSS List Properties

The following table shows a listing of value and description for properties that affect CSS list styles.

Property	Values	Description
list-style type	disc circle square decimal lower-roman upper-roman lower-alpha upper-alpha none	Defines the look of the bullets used in your list.
list-style image	none url(<url>)	Let's you use a custom graphic for bullets.
list-style position	outside inside	Often the text in a list is longer than one line. <ul style="list-style-type: none">– outer lets the second line align with the first line. That is: the bullet is to the left of both lines.– inner lets the second line align with the bullet.
list-style	<list-style	Used to group all the previous

	type> <list style position> <list-style image>	properties in one statement
--	--	-----------------------------

CSS List Example:

Define two style LI.list1 and LI.list2 where the first apply a green circled outside list and the second one apply a blue squared inside

```

<html>
<head>
<style type="text/css">
    LI.list1 {list-style: circle outside; color:green;}
    LI.list2 {list-style: square inside; color:blue}

    .blacktext {color:black}

</style>
</head>
<body>

<ul>
<li class="list1"><span class="blacktext">This is one black
line</span>
<li class="list1">This is another line that is much longer than the
first. But it isn't a black line since we did not specify a style for the
text that goes here other than the style we defined for the list.
</ul>
<br>
<br>
<ul>
<li class="list2"><span class="blacktext">This is one black
line</span>
<li class="list2">This is another line that is much longer than the

```

first. But it isn't a black line since we did not specify a style for the text that goes here other than the style we defined for the list.

``

CSS Layers

With CSS, it is possible to work with layers. Layers are pieces of HTML that are placed on top of the regular page with pixel precision. For example, to define a layer, `<div>` tag is used with a certain inline style that defines layer properties. In this example, the `<div>` in line style includes:

- 1) font size 50
- 2) Z-index = 2 (which indicate the layer stack order in the case of overlapping with other layers)
- 3) Its position which relative to its normal position. Position may take one of the following values:

Value	Description
absolute	Generates an absolutely positioned element, positioned relative to the first parent element that has a position other than static. The element's position is specified with the "left", "top", "right", and "bottom" properties
fixed	Generates an absolutely positioned element, positioned relative to the browser window. The element's position is specified with the "left", "top", "right", and "bottom" properties
relative	Generates a relatively positioned element, positioned relative to its normal position, so "left:20" adds 20 pixels to the element's LEFT position

Try out the following example:

```
<HTML>
<HEAD>
  <TITLE>New Document</TITLE>
```



```

</HEAD>
<BODY>LAYER 1 ON TOP:
  <DIV style="FONT-SIZE: 50px; Z-INDEX: 2; POSITION:
  relative">LAYER 1</DIV>
  <DIV
  style="FONT-SIZE: 80px; Z-INDEX: 1; LEFT: 5px; COLOR:
  red; POSITION: relative; TOP: -50px; BACKGROUND-
  COLOR: green">LAYER
  2</DIV>LAYER 2 ON TOP:
  <DIV style="FONT-SIZE: 50px; Z-INDEX: 3; POSITION:
  relative">LAYER 1</DIV>
  <DIV
  style="FONT-SIZE: 80px; Z-INDEX: 4; LEFT: 5px; COLOR:
  red; POSITION: relative; TOP: -50px">LAYER
  2</DIV>

</BODY>
</HTML>

```

Layer Visibility

CSS Layers may be visible or hidden using the Visibility property. For example the following statement makes the defined layer hidden.

- `<div style="position:relative; visibility:hidden;">HELLO!!!</div>`
- Visibility property takes either : visible or hidden

CSS display Property:

The display property sets how/if an element is displayed. The most fundamental types of display are inline, block-line and none:

- inline does just what it says - elements that are displayed inline follow the flow of a line.
- block puts a line break before and after the element Header.
- none, doesn't display the element, which may sound pretty useless but can be used to good effect with accessibility considerations.

- The following table show different examples for display property values:

Value	Description
none	The element will not be displayed
block	The element will be displayed as a block-level element, with a line break before and after the element
inline	Default. The element will be displayed as an inline element, with no line break before or after the element
list-item	The element will be displayed as a list
run-in	The element will be displayed as block-level or inline element depending on context
compact	The element will be displayed as block-level or inline element depending on context
marker	
table	The element will be displayed as a block table (like <table>), with a line break before and after the table
inline-table	The element will be displayed as an inline table (like <table>), with no line break before or after the table
table-row-group	The element will be displayed as a group of one or more rows (like <tbody>)
table-header-group	The element will be displayed as a group of one or more rows (like <thead>)
table-footer-group	The element will be displayed as a group of one or more rows (like <tfoot>)
table-row	The element will be displayed as a table row (like <tr>)
table-column-group	The element will be displayed as a group of one or more columns (like <colgroup>)

table-column	The element will be displayed as a column of cells (like <col>)
table-cell	The element will be displayed as a table cell (like <td> and <th>)
table-caption	The element will be displayed as a table caption (like <caption>)

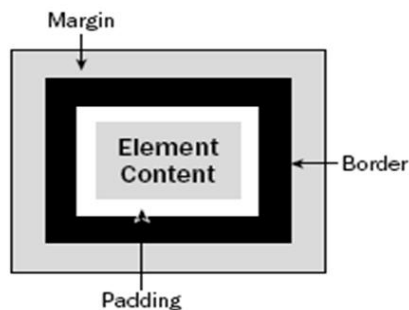
Try out this Example:

```
P {display: block }
LI{display: inline }
table{display: table}
td, th{display: table-cell }
```

CSS Margin, padding and border

An element's padding defines the space between the element and the space its border would occupy.

The CSS margin clears an area around an element (outside the border). The margin does not have a background color, and is completely transparent. The top, right, bottom, and left margin can be changed independently using separate properties. A shorthand margin property can also be used, to change all margins at once.



The following example show how to use top, bottom, right and left margin property:

```
margin-top:100px;
margin-bottom:100px;
margin-right:50px;
margin-left:50px;
```

The shorthand property for all the margin properties is "margin"

```
margin:100px 50px;
```

The CSS padding property defines the space between the element border and the element content. it is possible to specify different padding for different sides:

```
padding-top:25px;  
padding-bottom:25px;  
padding-right:50px;  
padding-left:50px;
```

The shorthand property for all the padding properties is "padding"

```
padding:25px 50px;
```

CSS Overflow property

The overflow property, Controls what happens when an element's content is larger than its containing box. The following table shows the different values for overflow property

Value	Description
visible	The overflow is not clipped. It renders outside the element's box. This is default (which causes the element to be displayed in its entirety, despite its containing box size)
hidden	The overflow is clipped, and the rest of the content will be invisible
scroll	The overflow is clipped, but a scroll-bar is added to see the rest of the content
auto	If overflow is clipped, a scroll-bar should be added to see the rest of the content

CSS Pseudoclasses

CSS pseudo-classes are used to add special effects to some selectors.

- The syntax of pseudo-classes:
Selector : pseudo-class {property: value}
- For example, the Anchor Pseudo-classes is used to display links in different ways in a CSS-supporting browser:

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

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

CSS classes can also be used with pseudo-classes:

- selector.class:pseudo-class {property: value}
- when the link in the following example has been visited, it will be displayed in red.

```
a.red:visited {color:#FF0000}  
  
<a class="red" href="css_syntax.asp">CSS Syntax</a>
```

CSS Projects:

Do the following Projects in Lab:

1. Write the CSS code to implement the following vertical Menu:



2. Write the CSS code to implement the following Horizontal Menu:



3. Write the code for a CSS based image gallery that displays larger thumbnail images when the mouse hovers move over them.



[Zoka Coffee](#)

Simply beautiful



Run wild with horses.

Exercise 6

1. What are the three ways of including CSS in your web page? Answer
2. What CSS rule can control the size of letters?
3. What CSS rule can control the color of letters?
4. What CSS rule can control the font type?
5. What CSS rule can set a background image?
6. What CSS rule can set the color of the background?
7. Write the CSS rule that will make the contents of all "li" tags bold.
8. Use the CSS property text-align with the value center to center the top level headline
9. Use CSS to change the default font for the entire page to a sans-serif font. The property you need is font-family and the value is sans-serif.
10. Use CSS to make all the second level headlines red.
11. Use an id attribute on the appropriate HTML element, along with CSS to make the words "Green Tea" the color green.
12. Write the CSS rule that will make a class named "chill" that will cause the displayed font to be "Chiller".
13. Write the CSS rule that will make the contents of both "p" and "li" tags have a 16pt font size.
14. Write the CSS rule that will make any element with the id equal to "highlight" have a yellow background color.

15.

Chapter 7 Laboratory 3 **Java Script**

Introduction

JavaScript is used in millions of Web pages to improve the design, validate forms, detect browsers, create cookies, and much more. JavaScript is the most popular scripting language on the internet, and works in all major browsers, such as Internet Explorer, Mozilla, Firefox, Netscape, Opera.

What is JavaScript?

- JavaScript was designed to add interactivity to HTML pages
- JavaScript is a scripting language (a scripting language is a lightweight programming language)
- A JavaScript consists of lines of executable computer code
- A JavaScript is usually embedded directly into HTML pages
- JavaScript is an interpreted language (means that scripts execute without preliminary compilation)
- Everyone can use JavaScript without purchasing a license

What can a JavaScript Do?

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

- **JavaScript can put dynamic text into an HTML page** - A JavaScript statement like this: `document.write("<h1>" + name + "</h1>")` can write a variable text into an HTML page
- **JavaScript can react to events** - A JavaScript can be set to execute when something happens, like when a page has finished loading or when a user clicks on an HTML element
- **JavaScript can read and write HTML elements** - A JavaScript can read and change the content of an HTML element
- **JavaScript can be used to validate data** - A JavaScript can be used to validate form data before it is submitted to a server, this will save the server from extra processing
- **JavaScript can be used to detect the visitor's browser** - A JavaScript can be used to detect the visitor's browser, and - depending on the browser - load another page specifically designed for that browser
- **JavaScript can be used to create cookies** - A JavaScript can be used to store and retrieve information on the visitor's computer

JavaScript How To ...

The HTML `<script>` tag is used to insert a JavaScript into an HTML page. How to Put a JavaScript Into an HTML Page

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

```
</script>
</body>

</html>
```

The code above will produce this output on an HTML page:

Hello World!

Example Explained

To insert a JavaScript into an HTML page, we use the `<script>` tag (also use the `type` attribute to define the scripting language). So, the `<script type="text/javascript">` and `</script>` tells where the JavaScript starts and ends:

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

The word `document.write` is a standard JavaScript command for writing output to a page. By entering the `document.write` command between the `<script type="text/javascript">` and `</script>` tags, the browser will recognize it as a JavaScript command and execute the code line. In this case the browser will write **Hello World!** to the page:

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

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

Note: If we had not entered the `<script>` tag, the browser would have treated the `document.write("Hello World!")` command as pure text, and just write the entire line on the page.

Ending Statements with a Semicolon?

With traditional programming languages, like C++ and Java, each code statement has to end with a semicolon. Many programmers continue this habit when writing JavaScript, but in general, semicolons are optional! However, semicolons are required if you want to put more than one statement on a single line.

How to Handle Older Browsers

Browsers that do not support JavaScript will display the script as page content. To prevent them from doing this, we may use the HTML comment tag:

```
<script type="text/javascript">
<!--
document.write("Hello World!")
//-->
</script>
```

The two forward slashes at the end of comment line (`//`) are a JavaScript comment symbol. This prevents the JavaScript compiler from compiling the line.

JavaScript Where To ...

JavaScripts in the body section will be executed WHILE the page loads. JavaScripts in the head section will be executed when called.

Where to Put the JavaScript

JavaScripts in a page will be executed immediately while the page loads into the browser. This is not always what we want. Sometimes we want to execute a script when a page loads, other times when a user triggers an event. Scripts in the head section: Scripts to be executed when they are called, or when an event is triggered, go in the head section. When you place a script in the head section, you will ensure that the script is loaded before anyone uses it.

```
<html>
<head>
<script type="text/javascript">
...
</script>

</head>
```

Scripts in the body section: Scripts to be executed when the page loads go in the body section. When you place a script in the body section it generates the content of the page.

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

```
....  
</script>  
  
</body>
```

Scripts in both the body and the head section: You can place an unlimited number of scripts in your document, so you can have scripts in both the body and the head section.

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

Using an External JavaScript

Sometimes you might want to run the same JavaScript on several pages, without having to write the same script on every page. To simplify this, you can write a JavaScript in an external file. Save the external JavaScript file with a .js file extension.

Note: The external script cannot contain the <script> tag!

To use the external script, point to the .js file in the "src" attribute of the <script> tag:

```
<html>
<head>
<script src="xxx.js"></script>
</head>
<body>
</body>
</html>
```

Note: Remember to place the script exactly where you normally would write the script!

JavaScript Variables

A variable is a "container" for information you want to store. A variable's value can change during the script. You can refer to a variable by name to see its value or to change its value.

Rules for variable names:

- Variable names are case sensitive
- They must begin with a letter or the underscore character

IMPORTANT! JavaScript is case-sensitive! A variable named `strname` is not the same as a variable named `STRNAME`!

Declare a Variable

You can create a variable with the `var` statement:

```
var strname = some value
```

You can also create a variable without the `var` statement:

```
Strname = some value
```

Assign a Value to a Variable

You can assign a value to a variable like this:

```
var strname = "Hege"
```

Or like this:

```
Strname = "Hege"
```

The variable name is on the left side of the expression and the value you want to assign to the variable is on the right. Now the variable "strname" has the value "Hege".

Lifetime of Variables

When you declare a variable within a function, the variable can only be accessed within that function. When you exit the function, the variable is destroyed. These variables are called local variables. You can have local variables with the same name in different functions, because each is recognized only by the function in which it is declared. If you declare a variable outside a function, all the functions on your page can access it. The lifetime of these variables starts when they are declared, and ends when the page is closed.

Conditional Statements

Conditional statements in JavaScript 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 if you want to execute some code only if a specified condition is true
- **if...else statement** - use this statement if you want 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 if you want to select one of many blocks of code to be executed
 - **switch statement** - use this statement if you want to select one of many blocks of code to be executed

If Statement

You should use the if statement if you want to execute some code only if a specified condition is true.

```
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 1

```
<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>
```

Example 2

```
<script type="text/javascript">
//Write "Lunch-time!" if the time is 11
var d=new Date()
var time=d.getHours()

if (time==11)
{
document.write("<b>Lunch-time!</b>")
}

</script>
```

Note: When comparing variables you must always use two equals signs next to each other (`==`). Notice that there is no `..else..` in this syntax. You just tell the code to execute some code only if the specified condition is true.

If...else Statement

If you want to execute some code if a condition is true and another code if the condition is not true, use the if...else statement.

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

Example

```
<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

You should use the if...else if...else statement if you want to select one of many sets of lines to execute.

```
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 condition1 and
condition2 are not true
}
```

Example

```
<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

You should use the switch statement if you want to select one of many blocks of code to be executed.

```
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.

Example

```
<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()
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 Operators

Arithmetic Operators

Operator	Description	Example	Result
+	Addition	x=2 y=2 x+y	4
-	Subtraction	x=5 y=2 x-y	3
*	Multiplication	x=5 y=4 x*y	20
/	Division	15/5 5/2	3 2.5
%	Modulus (division remainder)	5%2 10%8 10%2	1 2 0
++	Increment	x=5 x++	x=6
--	Decrement	x=5 x--	x=4

Assignment Operators

Operator	Example	Is The Same As
=	x=y	x=y

+=	x+=y	x=x+y
-=	x-=y	x=x-y
=	x=y	x=x*y
/=	x/=y	x=x/y
%=	x%=y	x=x%y

Comparison Operators

Operator	Description	Example
==	is equal to	5==8 returns false
===	is equal to (checks for both value and type)	x=5 y="5" x==y returns true x===y returns false
!=	is not equal	5!=8 returns true
>	is greater than	5>8 returns false
<	is less than	5<8 returns true
>=	is greater than or equal to	5>=8 returns false
<=	is less than or equal to	5<=8 returns true

Logical Operators

Operator	Description	Example
&&	and	x=6 y=3 (x < 10 && y > 1) returns true

	or	x=6 y=3 (x==5 y==5) returns false
!	not	x=6 y=3 !(x==y) returns true

String Operator

A string is most often text, for example "Hello World!". To stick two or more string variables together, use the + operator.

```
txt1="What a very"  
txt2="nice day!"  
txt3=txt1+txt2
```

The variable txt3 now contains "What a verynice day!". To add a space between two string variables, insert a space into the expression, OR in one of the strings.

```
txt1="What a very"  
txt2="nice day!"  
txt3=txt1+" "+txt2  
or  
txt1="What a very "  
txt2="nice day!"  
txt3=txt1+txt2
```

The variable txt3 now contains "What a very nice day!".

Conditional Operator

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

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

Example

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

if the variable visitor is equal to PRES, then put the string "Dear President " in the variable named greeting. If the variable visitor is not equal to PRES, then put the string "Dear " into the variable named greeting.

JavaScript Popup Boxes

In JavaScript we can create three kind 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.

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

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.


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

Prompt Box

A prompt box is often used if you want the user to input a value before entering a page.

JavaScript Functions

A function is a reusable code-block that will be executed by an event, or when the function is called. To keep the browser from executing a script as soon as the page is loaded, you can write your script as a function. A function contains some code that will be executed only by an event or by a call to that function. You may call a function from anywhere within the page (or even from other pages if the function is embedded in an external .js file). Functions are defined at the beginning of a page, in the <head> section.

```
<html>
<head>
<script type="text/javascript">
function displaymessage()
{ alert("Hello World!") }
</script>
</head>
<body>
<form>
<input type="button" value="Click me!"
onclick="displaymessage()" >
</form>
</body>
</html>
```

If the line: `alert("Hello world!!")`, in the example above had not been written within a function, it would have been executed as soon as the line was loaded. Now, the script is not executed before the user hits the button. We have added an `onClick` event to the button that will execute the function `displaymessage()` when the button is clicked.

How to Define a Function

The syntax for creating a function is:

```
function functionname(var1,var2,...,varX)
{
some code
}
```

`var1`, `var2`, etc are variables or values passed into the function. The `{` and the `}` defines the start and end of the function. **Note:** A function with no parameters must include the parentheses `()` after the function name:

```
function functionname()
{
some code
}
```

Note: Do not forget about the importance of capitals in JavaScript! The word `function` must be written in lowercase letters, otherwise a JavaScript error occurs! Also note that you must call a function with the exact same capitals as in the function name.

The return Statement

The return statement is used to specify the value that is returned from the function. So, functions that are going to return a value must use the return statement.

Example: The function below should return the product of two numbers (a and b):

```
function prod(a,b)  
{  
x=a*b  
return x}
```

When you call the function above, you must pass along two parameters:

```
product=prod(2,3)
```

The returned value from the prod() function is 6, and it will be stored in the variable called product.

JavaScript Loop

Loops in JavaScript are used to execute the same block of code a specified number of times or while a specified condition is true. Very often when you write code, you want the same block of code to run over and over again in a row. Instead of adding several almost equal lines in a script we can use loops to perform a task like this. In JavaScript there are two different kind of loops:

- for - loops through a block of code a specified number of times

- while - loops through a block of code while a specified condition is true

The for Loop

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

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

Example

Explanation: 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 10. i will increase by 1 each time the loop runs.

Note: The increment parameter could also be negative, and the <= could be any comparing statement.

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

Result

```
The number is 0  
The number is 1  
The number is 2  
The number is 3  
The number is 4  
The number is 5  
The number is 6  
The number is 7  
The number is 8  
The number is 9  
The number is 10
```

The while loop

The while loop is used when you want the loop to execute and continue executing while the specified condition is true.

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

Note: The <= could be any comparing statement.

Example

Explanation: 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 10. i will increase by 1 each time the loop runs.

```
<html>  
<body>  
<script type="text/javascript">  
var i=0
```

```
while (i<=10)  
{  
document.write("The number is " + i)  
document.write("<br />")  
i=i+1  
}  
</script>  
</body>  
</html>
```

Result

```
The number is 0  
The number is 1  
The number is 2  
The number is 3  
The number is 4  
The number is 5  
The number is 6  
The number is 7  
The number is 8  
The number is 9  
The number is 10
```

The do...while Loop

The do...while loop is a variant of the while loop. This loop will always execute a block of code ONCE, and then it will repeat the loop as long as the specified condition is true. This loop will always be executed once, even if the condition is false, because the code is executed before the condition is tested.

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

Example

```
<html>
<body>
<script type="text/javascript">
var i=0
do
{
document.write("The number is " + i)
document.write("<br />")
i=i+1
}
while (i<0)
</script>
</body>
</html>
```

Result

The number is 0

JavaScript Break and Continue

JavaScript break and continue Statements. There are two special statements that can be used inside loops: break and continue.

Break

The break command 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>
```

Result

```
The number is 0
The number is 1
The number is 2
```

Continue

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

Example

```
<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>
```

Result

```
The number is 0
The number is 1
The number is 2
The number is 4
The number is 5
The number is 6
The number is 7
The number is 8
The number is 9
The number is 10
```

JavaScript Events

Events are actions that can be detected by JavaScript.

Events

By using JavaScript, we have the ability to create dynamic web pages. Events are actions that can be detected by JavaScript. Every element on a web page has certain events which can trigger JavaScript functions. 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 box in an HTML form
- Submitting an HTML form
- A keystroke

The following lists the events recognized by JavaScript:

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

Onload and onUnload

The `onload` and `onUnload` events are triggered when the user enters or leaves the page. The `onload` event is often used to check the visitor's browser type and browser version, and load the proper version of the web page based on the information. Both the `onload` and `onUnload` events are also often used to deal with cookies that should be set when a user enters or leaves a page. For example, you

could have a popup asking for the user's name upon his first arrival to your page. The name is then stored in a cookie. Next time the visitor arrives at your page, you could have another popup saying something like: "Welcome John Doe!".

OnFocus, onBlur and onChange

The onFocus, onBlur and onChange events are often used in combination with validation of form fields. Below is an example of how to use the onChange event. The checkEmail() function will be called whenever the user changes the content of the field:

```
<input type="text" size="30"
id="email" onchange="checkEmail()">
```

onSubmit

The onSubmit event is used to validate ALL form fields before submitting it. Below is an example of how to use the onSubmit event. The checkForm() function will be called when the user clicks the submit button in the form. If the field values are not accepted, the submit should be cancelled. The function checkForm() returns either true or false. If it returns true the form will be submitted, otherwise the submit will be cancelled:

```
<form method="post" action="xxx.htm"
onsubmit="return checkForm()">
```

OnMouseOver and onMouseOut

onMouseOver and onMouseOut are often used to create "animated" buttons. Below is an example of an onMouseOver

event. An alert box appears when an onmouseover event is detected:

```
<a href="http://www.wGoogle.com"
onmouseover="alert('An onmouseover event');return false">

</a>
```

JavaScript - Catching Errors

When browsing Web pages on the internet, I guess we have all seen a JavaScript alert box, telling you there is a runtime error, and asking: "Do you wish to debug?" on some pages. Error message like that may be useful for developers, but not for the users. When users see errors, they often leave the Web page.

There are two ways of catching errors in a Web page:

- By using the try...catch statement (available in IE5+, Mozilla 1.0, and Netscape 6)
- By using the onerror event. This is the old standard solution to catch errors (available since Netscape 3)

JavaScript Try...Catch Statement

The try...catch statement allows you to test a block of code for errors. The try...catch statement allows you to test a block of code for errors. The try block contains the code to be run, and the catch block contains the code to be executed if an error occurs.

```
try
```

```
{  
//Run some code here  
}  
catch(identifier)  
{  
//Handle exceptions here  
}
```

Note that try...catch is written in lowercase letters. Using uppercase letters will generate a JavaScript error!

Example 1

The example below contains a script that is supposed to display the message "Welcome guest!" when you click on a button. However, there's a typo in the message() function. alert() is misspelled as adddalert(). A JavaScript error occurs:

```
<html>  
<head>  
<script type="text/javascript">  
function message()  
{  
adddalert("Welcome guest!")  
}  
</script>  
</head>  
<body>  
<input type="button" value="View message" onclick="message()" />  
</body>  
</html>
```

To take more appropriate action when an error occurs, you can add a try...catch statement. The example below contains the "Welcome guest!" example rewritten to use the try...catch statement. Since alert() is misspelled, a JavaScript error occurs. However, this time,

the catch block catches the error and executes a custom code to handle it. The code displays a custom error message informing the user what happened:

```
<html>
<head>
<script type="text/javascript">
var txt=""
function message()
{
try
{
  adddler("Welcome guest!")
}
catch(err)
{
  txt="There was an error on this page.\n\n"
  txt+="Error description: " + err.description + "\n\n"
  txt+="Click OK to continue.\n\n"
  alert(txt)
}
}
</script>
</head>

<body>
<input type="button" value="View message" onclick="message()"
/>
</body>

</html>
```

Example 2

The next example uses a confirm box to display a custom message telling users they can click OK to continue viewing the page or click Cancel to go to the homepage. If the confirm method returns

false, the user clicked Cancel, and the code redirects the user. If the confirm method returns true, the code does nothing:

```
<html>
<head>
<script type="text/javascript">
var txt=""
function message()
{
try
{
addlert("Welcome guest!")
}
catch(err)
{
txt="There was an error on this page.\n\n"
txt+="Click OK to continue viewing this page,\n"
txt+="or Cancel to return to the home page.\n\n"
if(!confirm(txt))
{
document.location.href="http://www.wGoogle.com/"
}
}
}
</script>
</head>
<body>
<input type="button" value="View message"
onclick="message()" />
</body>
</html>
```

The onerror Event

The onerror event will be explained soon, but first you will learn how to use the throw statement to create an exception. The throw statement can be used together with the try...catch statement.

The Throw Statement

The throw statement allows you to create an exception. If you use this statement together with the try...catch statement, you can control program flow and generate accurate error messages.

```
throw(exception)
```

The exception can be a string, integer, Boolean or an object. Note that throw is written in lowercase letters. Using uppercase letters will generate a JavaScript error!

Example 1

The example below determines the value of a variable called x. If the value of x is higher than 10 or lower than 0 we are going to throw an error. The error is then caught by the catch argument and the proper error message is displayed:

```
<html>  
<body>  
<script type="text/javascript">  
var x=prompt("Enter a number between 0 and 10:", "");  
try  
{  
if(x>10)  
throw "Err1"  
else if(x<0)  
throw "Err2"  
}  
catch(er)  
{  
if(er=="Err1")  
alert("Error! The value is to high")  
if(er == "Err2")
```

```
alert("Error! The value is to low")
}
</script>
</body>
</html>
```

Using the `onerror` event is the old standard solution to catch errors in a web page.

The `onerror` Event

We have just explained how to use the `try...catch` statement to catch errors in a web page. Now we are going to explain how to use the `onerror` event for the same purpose. The `onerror` event is fired whenever there is a script error in the page. To use the `onerror` event, you must create a function to handle the errors. Then you call the function with the `onerror` event handler. The event handler is called with three arguments: `msg` (error message), `url` (the url of the page that caused the error) and `line` (the line where the error occurred).

```
onerror=handleErr
function handleErr(msg,url,l)
{
//Handle the error here
return true or false
}
```

The value returned by `onerror` determines whether the browser displays a standard error message. If you return `false`, the browser displays the standard error message in the JavaScript console. If you return `true`, the browser does not display the standard error message.

Example

The following example shows how to catch the error with the onerror event:

```
<html>
<head>
<script type="text/javascript">
onerror=handleErr
var txt=""
function handleErr(msg,url,l)
{
txt="There was an error on this page.\n\n"
txt+="Error: " + msg + "\n"
txt+="URL: " + url + "\n"
txt+="Line: " + l + "\n\n"
txt+="Click OK to continue.\n\n"
alert(txt)
return true
}
function message()
{
addlert("Welcome guest!")
}
</script>
</head>
<body>
<input type="button" value="View message" onclick="message()"
/>
</body>
</html>
```

JavaScript Special Characters

In JavaScript you can add special characters to a text string by using the backslash sign.

Insert Special Characters

The backslash (\) is used to insert apostrophes, new lines, quotes, and other special characters into a text string. Look at the following JavaScript code:

```
var txt="We are the so-called "Vikings" from the north."  
document.write(txt)
```

In JavaScript, a string is started and stopped with either single or double quotes. This means that the string above will be chopped to: We are the so-called To solve this problem, you must place a backslash (\) before each double quote in "Viking". This turns each double quote into a string literal:

```
var txt="We are the so-called \"Vikings\" from the north."  
document.write(txt)
```

JavaScript will now output the proper text string: We are the so-called "Vikings" from the north. Here is another example:

```
document.write ("You \& me are singing!")
```

The example above will produce the following output:

```
You & me are singing!
```

The table below lists other special characters that can be added to a text string with the backslash sign:

Code	Outputs
\'	single quote
\"	double quote

\&	ampersand
\\	backslash
\n	new line
\r	carriage return
\t	tab
\b	backspace
\f	form feed

Comments

You can add comments to your script by using two slashes //:

```
//this is a comment  
document.write("Hello World!")
```

or by using /* and */ (this creates a multi-line comment block):

```
/* This is a comment  
block. It contains  
several lines */  
document.write("Hello World!")
```

JavaScript Objects Introduction

JavaScript is an Object Oriented Programming (OOP) language. An OOP language allows you to define your own objects and make your own variable types.

Object Oriented Programming

JavaScript is an Object Oriented Programming (OOP) language. An OOP language allows you to define your own objects and make your own variable types. However, creating your own objects will be explained later, in the Advanced JavaScript section. We will

start by looking at the built-in JavaScript objects, and how they are used. The next pages will explain each built-in JavaScript object in detail. Note that an object is just a special kind of data. An object has properties and methods.

Properties

Properties are the values associated with an object. In the following example we are using the length property of the String object to return the number of characters in a string:

```
<script type="text/javascript">  
var txt="Hello World!"  
document.write(txt.length)  
</script>
```

The output of the code above will be:

```
12
```

Methods

Methods are the actions that can be performed on objects. In the following example we are using the toUpperCase() method of the String object to display a text in uppercase letters:

```
<script type="text/javascript">  
var str="Hello world!"  
document.write(str.toUpperCase())  
</script>
```

The output of the code above will be:

```
HELLO WORLD!
```

String object

The String object is used to manipulate a stored piece of text.

Examples of use: The following example uses the length property of the String object to find the length of a string:

```
var txt="Hello world!"  
document.write(txt.length)
```

The code above will result in the following output:

```
12
```

The following example uses the toUpperCase() method of the String object to convert a string to uppercase letters:

```
var txt="Hello world!"  
document.write(txt.toUpperCase())
```

The code above will result in the following output:

```
HELLO WORLD!
```

JavaScript Date Object

The Date object is used to work with dates and times. We define a Date object with the new keyword. The following code line defines a Date object called myDate:

```
var myDate=new Date()
```

Note: The Date object will automatically hold the current date and time as its initial value!

Manipulate Dates

We can easily manipulate the date by using the methods available for the Date object. In the example below we set a Date object to a specific date (14th January 2010):

```
var myDate=new Date()  
myDate.setFullYear(2010,0,14)
```

And in the following example we set a Date object to be 5 days into the future:

```
var myDate=new Date()  
myDate.setDate(myDate.getDate()+5)
```

Note: If adding five days to a date shifts the month or year, the changes are handled automatically by the Date object itself!

Comparing Dates

The Date object is also used to compare two dates.

The following example compares today's date with the 14th January 2010:

```
var myDate=new Date()  
myDate.setFullYear(2010,0,14)  
var today = new Date()  
if (myDate>today)  
  alert("Today is before 14th January 2010")  
else  
  alert("Today is after 14th January 2010")
```

JavaScript Array Object

The Array object is used to store a set of values in a single variable name. We define an Array object with the new keyword. The following code line defines an Array object called myArray:

```
var myArray=new Array()
```

There are two ways of adding values to an array (you can add as many values as you need to define as many variables you require).

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

You could also pass an integer argument to control the array's size:

```
var mycars=new Array(3)  
mycars[0]="Saab"  
mycars[1]="Volvo"  
mycars[2]="BMW"
```

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

Note: If you specify numbers or true/false values inside the array then the type of variables will be numeric or Boolean **instead of** string.

Accessing Arrays

You can refer to a particular element in an array by referring to the name of the array and the index number. The index number starts at 0.

The following code line:

```
document.write(mycars[0])
```

will result in the following output:

```
Saab
```

Modify Values in Existing Arrays

To modify a value in an existing array, just add a new value to the array with a specified index number:

```
mycars[0]="Opel"
```

Now, the following code line:

```
document.write(mycars[0])
```

will result in the following output:

```
Opel
```

JavaScript Boolean Object

The Boolean object is used to convert a non-Boolean value to a Boolean value (true or false).

Boolean Object

The Boolean object is an object wrapper for a Boolean value. The Boolean object is used to convert a non-Boolean value to a Boolean value (true or false). We define a Boolean object with the

new keyword. The following code line defines a Boolean object called myBoolean:

```
var myBoolean=new Boolean()
```

Note: If the Boolean object has no initial value or if it is 0, -0, null, "", false, undefined, or NaN, the object is set to false. Otherwise it is true (even with the string "false"). All the following lines of code create Boolean objects with an initial value of false:

```
var myBoolean=new Boolean()  
var myBoolean=new Boolean(0)  
var myBoolean=new Boolean(null)  
var myBoolean=new Boolean("")  
var myBoolean=new Boolean(false)  
var myBoolean=new Boolean(NaN)
```

And all the following lines of code create Boolean objects with an initial value of true:

```
var myBoolean=new Boolean(true)  
var myBoolean=new Boolean("true")  
var myBoolean=new Boolean("false")  
var myBoolean=new Boolean("Richard")
```

JavaScript Math Object

The Math object allows you to perform common mathematical tasks. The Math object includes several mathematical values and functions. You do not need to define the Math object before using.

Mathematical Values

JavaScript provides eight mathematical values (constants) that can be accessed from the Math object. These are: E, PI, square root of 2, square root of 1/2, natural log of 2, natural log of 10, base-2 log of E, and base-10 log of E. You may reference these values from your JavaScript like this:

```
Math.E  
Math.PI  
Math.SQRT2  
Math.SQRT1_2  
Math.LN2  
Math.LN10  
Math.LOG2E  
Math.LOG10E
```

Mathematical Methods

In addition to the mathematical values that can be accessed from the Math object there are also several functions (methods) available.

Examples of functions (methods):

The following example uses the round() method of the Math object to round a number to the nearest integer:

```
document.write(Math.round(4.7))
```

The code above will result in the following output:

```
5
```

The following example uses the `random()` method of the `Math` object to return a random number between 0 and 1:

```
document.write(Math.random());
```

The code above can result in the following output:

```
0.24763142494185597
```

The following example uses the `floor()` and `random()` methods of the `Math` object to return a random number between 0 and 10:

```
document.write(Math.floor(Math.random()*11));
```

The code above can result in the following output:

```
4
```

JavaScript HTML DOM Objects

The HTML DOM is a W3C standard and it is an abbreviation for the Document Object Model for HTML. The HTML DOM defines a standard set of objects for HTML, and a standard way to access and manipulate HTML documents. All HTML elements, along with their containing text and attributes, can be accessed through the DOM. The contents can be modified or deleted, and new elements can be created. The HTML DOM is platform and language independent. It can be used by any programming language like Java, JavaScript, and VBScript.

JavaScript Browser Detection

Almost everything in this chapter works on all JavaScript-enabled browsers. However, there are some things that just don't work on certain browsers - especially on older browsers. So, sometimes it can be very useful to detect the visitor's browser type and version, and then serve up the appropriate information. The best way to do this is to make your web pages smart enough to look one way to some browsers and another way to other browsers. JavaScript includes an object called the Navigator object that can be used for this purpose. The Navigator object contains information about the visitor's browser name, browser version, and more.

The Navigator Object

The JavaScript Navigator object contains all information about the visitor's browser. We are going to look at two properties of the Navigator object:

- `appName` - holds the name of the browser
- `appVersion` - holds, among other things, the version of the browser

Example

```
<html>
<body>
<script type="text/javascript">
var browser=navigator.appName
var b_version=navigator.appVersion
var version=parseFloat(b_version)
document.write("Browser name: "+ browser)
document.write("<br />")
document.write("Browser version: "+ version)
```

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

The variable `browser` in the example above holds the name of the browser, i.e. "Netscape" or "Microsoft Internet Explorer". The `appVersion` property in the example above returns a string that contains much more information than just the version number, but for now we are only interested in the version number. To pull the version number out of the string we are using a function called `parseFloat()`, which pulls the first thing that looks like a decimal number out of a string and returns it.

Example: The script below displays a different alert, depending on the visitor's browser:

```
<html>
<head>
<script type="text/javascript">
function detectBrowser()
{
var browser=navigator.appName
var b_version=navigator.appVersion
var version=parseFloat(b_version)
if ((browser=="Netscape"||browser=="Microsoft Internet
Explorer")
&& (version>=4))
{alert("Your browser is good enough!")}
else
{alert("It's time to upgrade your browser!")}
}
</script>
</head>
<body onload="detectBrowser()">
</body>
```

</html>

JavaScript Cookies

A cookie is a variable that is stored on the visitor's computer. Each time the same computer requests a page with a browser, it will send the cookie too. With JavaScript, you can both create and retrieve cookie values.

Examples of cookies:

- Name cookie - The first time a visitor arrives to your web page, he or she must fill in her/his name. The name is then stored in a cookie. Next time the visitor arrives at your page, he or she could get a welcome message like "Welcome John Doe!" The name is retrieved from the stored cookie
- Password cookie - The first time a visitor arrives to your web page, he or she must fill in a password. The password is then stored in a cookie. Next time the visitor arrives at your page, the password is retrieved from the cookie
- Date cookie - The first time a visitor arrives to your web page, the current date is stored in a cookie. Next time the visitor arrives at your page, he or she could get a message like "Your last visit was on Tuesday August 11, 2005!" The date is retrieved from the stored cookie

Create and Store a Cookie

In this example we will create a cookie that stores the name of a visitor. The first time a visitor arrives to the web page, he or she will be asked to fill in her/his name. The name is then stored in a cookie. The next time the visitor arrives at the same page, he or she will get welcome message. First, we create a function that stores the name of the visitor in a cookie variable:

```
function setCookie(c_name,value,expiredays)
{
var exdate=new Date()
exdate.setDate(exdate.getTime()+expiredays)
document.cookie=c_name+ "=" +escape(value)+
((expiredays==null) ? "" : ";expires="+exdate)
}
```

The parameters of the function above holds the name of the cookie, the value of the cookie, and the number of days until the cookie expires. In the function above we first convert the number of days to a valid date, then we add the number of days until the cookie should expire. After that we store the cookie name, cookie value and the expiration date in the document.cookie object. Then, we create another function that checks if the cookie has been set:

```
function getCookie(c_name)
{
if (document.cookie.length>0)
{
c_start=document.cookie.indexOf(c_name + "=")
if (c_start!=-1)
{
c_start=c_start + c_name.length+1
c_end=document.cookie.indexOf(";",c_start)
if (c_end==-1) c_end=document.cookie.length
}
```

```
    return unescape(document.cookie.substring(c_start,c_end))
  }
}
return null
}
```

The function above first checks if a cookie is stored at all in the document.cookie object. If the document.cookie object holds some cookies, then check to see if our specific cookie is stored. If our cookie is found, then return the value, if not - return null. Last, we create the function that displays a welcome message if the cookie is set, and if the cookie is not set it will display a prompt box, asking for the name of the user:

```
function checkCookie()
{
username=getCookie('username')
if (username!=null)
{alert('Welcome again '+username+'!')}
else
{
username=prompt('Please enter your name:','')
if (username!=null||username!='')
{
setCookie('username',username,365)
}
}
}
```

All together now:

```
<html>
<head>
<script type="text/javascript">
function getCookie(c_name)
{
if (document.cookie.length>0)
{
```



```

c_start=document.cookie.indexOf(c_name + "=")
if (c_start!=-1)
{
c_start=c_start + c_name.length+1
c_end=document.cookie.indexOf(";",c_start)
if (c_end== -1) c_end=document.cookie.length
return unescape(document.cookie.substring(c_start,c_end))
}
}
return null
}
function setCookie(c_name,value,expiredays)
{
var exdate=new Date()
exdate.setTime(exdate.getTime()+(expiredays*24*3600*1000))
document.cookie=c_name+ "=" +escape(value)+
((expiredays==null) ? "" : "; expires="+exdate)
}
function checkCookie()
{
username=getCookie('username')
if (username!=null)
{alert('Welcome again '+username+'!')}
else
{
username=prompt('Please enter your name:','')
if (username!=null||username!="")
{
setCookie('username',username,365)
}
}
}
</script>
</head>
<body onLoad="checkCookie()">
</body>
</body>
</html>

```

The example above runs the checkCookie() function when the page loads.

JavaScript Form Validation

JavaScript can be used to validate input data in HTML forms before sending off the content to a server. Form data that typically are checked by a JavaScript could be:

- has the user left required fields empty?
- has the user entered a valid e-mail address?
- has the user entered a valid date?
- has the user entered text in a numeric field?

Required Fields

The function below checks if a required field has been left empty. If the required field is blank, an alert box alerts a message and the function returns false. If a value is entered, the function returns true (means that data is OK):

```
function validate_required(field,alerttxt)
{
with (field)
{
if (value==null||value=="")
{alert(alerttxt);return false}
else {return true}
}
}
```

The entire script, with the HTML form could look something like this:

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

```

<script type="text/javascript">
function validate_required(field,alerttxt)
{
with (field)
{
if (value==null||value=="")
{alert(alerttxt);return false}
else {return true}
}
}
function validate_form(thisform)
{
with (thisform)
{
if (validate_required(email,"Email must be filled out!")==false)
{email.focus();return false}
}
}
</script>
</head>
<body>
<form action="submitpage.htm"
onsubmit="return validate_form(this)"
method="post">
Email: <input type="text" name="email" size="30">
<input type="submit" value="Submit">
</form>
</body>
</html>

```

E-mail Validation

The function below checks if the content has the general syntax of an email. This means that the input data must contain at least an @ sign and a dot (.). Also, the @ must not be the first character of the email address, and the last dot must at least be one character after the @ sign:

```
function validate_email(field,alerttxt)
{
with (field)
{
apos=value.indexOf("@")
dotpos=value.lastIndexOf(".")
if (apos<1||dotpos-apos<2)
{alert(alerttxt);return false}
else {return true}
}
}
```

The entire script, with the HTML form could look something like this:

```
<html>
<head>
<script type="text/javascript">
function validate_email(field,alerttxt)
{
with (field)
{
apos=value.indexOf("@")
dotpos=value.lastIndexOf(".")
if (apos<1||dotpos-apos<2)
{alert(alerttxt);return false}
else {return true}
}
}
```

```

function validate_form(thisform)
{
with (thisform)
{
if (validate_email(email,"Not a valid e-mail address!")==false)
{email.focus();
return false}
}
}
</script>
</head>
<body>
<form action="submitpage.htm"
onsubmit="return validate_form(this)"
method="post">
Email: <input type="text" name="email" size="30">
<input type="submit" value="Submit">
</form>
</body>
</html>

```

JavaScript Animation

It is possible to use JavaScript to create animated images. The trick is to let a JavaScript change between different images on different events. In the following example we will add an image that should act as a link button on a web page. We will then add an onMouseOver event and an onMouseOut event that will run two JavaScript functions that will change between the images.

The HTML code looks like this:

```

<a href="http://www.Google.com" target="_blank"
onmouseover="mouseOver()"
onmouseout="mouseOut()">

</a>

```

Note that we have given the image a name to make it possible for JavaScript to address it later. The `onMouseOver` event tells the browser that once a mouse is rolled over the image, the browser should execute a function that will replace the image with another image. The `onMouseOut` event tells the browser that once a mouse is rolled away from the image, another JavaScript function should be executed. This function will insert the original image again.

IMPORTANT! The mouse events are added to the `<a>` tag, and not to the `` tag. Unfortunately, browsers do not support mouse events on images!

The JavaScript Code

The changing between the images is done with the following JavaScript:

```
<script type="text/javascript">
function mouseOver()
{
document.b1.src ="b_blue.gif"
}
function mouseOut()
{
document.b1.src ="b_pink.gif"
}
</script>
```

The function `mouseOver()` causes the image to shift to `"b_blue.gif"`.

The Entire Code

```
<html>
head>
<script type="text/javascript">
function mouseOver()
{
document.b1.src ="b_blue.gif"
}
function mouseOut()
{
document.b1.src ="b_pink.gif"
}
</script>
</head>
<body>
<a href="http://www.wGoogle.com" target="_blank"
onmouseover="mouseOver()"
onmouseout="mouseOut()">

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

JavaScript Image Maps

From our HTML chapter we have learned that an image-map is an image with clickable regions. Normally, each region has an associated hyperlink. Clicking on one of the regions takes you to the associated link.

Example: The example below demonstrates how to create an HTML image map, with clickable regions. Each of the regions is a hyperlink:

```
<img src ="planets.gif"
width ="145" height ="126"
alt="Planets"
```

```

usemap = "#planetmap" />
<map id = "planetmap"
name = "planetmap">
<area shape = "rect" coords = "0,0,82,126"
href = "sun.htm" target = "_blank"
alt = "Sun" />
<area shape = "circle" coords = "90,58,3"
href = "mercur.htm" target = "_blank"
alt = "Mercury" />
<area shape = "circle" coords = "124,58,8"
href = "venus.htm" target = "_blank"
alt = "Venus" />
</map>

```

Adding some JavaScript

We can add events (that can call a JavaScript) to the <area> tags inside the image map. The <area> tag supports the onClick, onDbClick, onMouseDown, onMouseUp, onMouseOver, onMouseMove, onMouseOut, onKeyPress, onKeyDown, onKeyUp, onFocus, and onBlur events. Here's the above example, with some JavaScript added:

```

<html>
<head>
<script type = "text/javascript">
function writeText(txt)
{
document.getElementById("desc").innerHTML=txt
}
</script>
</head>
<body>
<img src = "planets.gif" width = "145" height = "126"
alt = "Planets" usemap = "#planetmap" />

<map id = "planetmap" name = "planetmap">
<area shape = "rect" coords = "0,0,82,126"

```



```

onMouseOver="writeText('The Sun and the gas giant
planets like Jupiter are by far the largest objects
in our Solar System.')"
href ="sun.htm" target ="_blank" alt="Sun" />

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

<area shape ="circle" coords ="124,58,8"
onMouseOver="writeText('Until the 1960s, Venus was
often considered a twin sister to the Earth because
Venus is the nearest planet to us, and because the
two planets seem to share many characteristics.')"
href ="venus.htm" target ="_blank" alt="Venus" />
</map>

<p id="desc"></p>

</body>
</html>

```

JavaScript Timing Events

With JavaScript, it is possible to execute some code NOT immediately after a function is called, but after a specified time interval. This is called timing events. It's very easy to time events in JavaScript. Two key methods are used:

- `setTimeout()` - executes a code some time in the future
- `clearTimeout()` - cancels the `setTimeout()`

`setTimeout()`

```
var t=setTimeout("javascript statement",milliseconds)
```

The `setTimeout()` method returns a value - In the statement above, the value is stored in a variable called `t`. If you want to cancel this `setTimeout()`, you can refer to it using the variable name. The first parameter of `setTimeout()` is a string that contains a JavaScript statement. This statement could be a statement like `"alert('5 seconds!')"` or a call to a function, like `"alertMsg()"`. The second parameter indicates how many milliseconds from now you want to execute the first parameter.

Note: There are 1000 milliseconds in one second.

Example: When the button is clicked in the example below, an alert box will be displayed after 5 seconds.

```
<html>
<head>
<script type="text/javascript">
function timedMsg()
{
var t=setTimeout("alert('5 seconds!')",5000)
}
</script>
</head>
<body>
<form>
<input type="button" value="Display timed alertbox!"
onClick="timedMsg()">
</form>
</body>
</html>
```

Example - Infinite Loop

To get a timer to work in an infinite loop, we must write a function that calls itself. In the example below, when the button is clicked, the input field will start to count (for ever), starting at 0:

```
<html>
<head>
<script type="text/javascript">
var c=0
var t
function timedCount()
{
document.getElementById('txt').value=c
c=c+1
t=setTimeout("timedCount()",1000)
}
</script>
</head>
<body>
<form>
<input type="button" value="Start count!"
onClick="timedCount()">
<input type="text" id="txt">
</form>
</body>
</html>
```

clearTimeout()

```
clearTimeout(setTimeout_variable)
```

Example: The example below is the same as the "Infinite Loop" example above. The only difference is that we have now added a "Stop Count!" button that stops the timer:

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

```

var c=0
var t
function timedCount()
{
document.getElementById('txt').value=c
c=c+1
t=setTimeout("timedCount()",1000)
}
function stopCount()
{
clearTimeout(t)
}
</script>
</head>
<body>
<form>
<input type="button" value="Start count!"
onClick="timedCount()">
<input type="text" id="txt">
<input type="button" value="Stop count!"
onClick="stopCount()">
</form>
</body>
</html>

```

JavaScript Create Your Own Objects

Earlier in this chapter we have seen that JavaScript has several built-in objects, like String, Date, Array, and more. In addition to these built-in objects, you can also create your own. An object is just a special kind of data, with a collection of properties and methods. Let's illustrate with an example: A person is an object. Properties are the values associated with the object. The persons' properties include name, height, weight, age, skin tone, eye color, etc. All persons have these properties, but the values of those properties will differ from person to person. Objects also have

methods. Methods are the actions that can be performed on objects. The persons' methods could be eat(), sleep(), work(), play(), etc.

Properties

The syntax for accessing a property of an object is:

```
objName.propName
```

You can add properties to an object by simply giving it a value. Assume that the personObj already exists - you can give it properties named firstname, lastname, age, and eyecolor as follows:

```
personObj.firstname="John"  
personObj.lastname="Doe"  
personObj.age=30  
personObj.eyecolor="blue"  
document.write(personObj.firstname)
```

The code above will generate the following output:

```
John
```

Methods

An object can also contain methods. You can call a method with the following syntax:

```
objName.methodName()
```

Note: Parameters required for the method can be passed between the parentheses. To call a method called sleep() for the personObj:

```
personObj.sleep()
```

Creating Your Own Objects

There are different ways to create a new object:

1. Create a direct instance of an object

The following code creates an instance of an object and adds four properties to it:

```
personObj=new Object()  
personObj.firstname="John"  
personObj.lastname="Doe"  
personObj.age=50  
personObj.eyecolor="blue"
```

Adding a method to the personObj is also simple. The following code adds a method called eat() to the personObj:

```
personObj.eat=eat
```

2. Create a template of an object

The template defines the structure of an object:

```
function person(firstname,lastname,age,eyecolor)  
{  
  this.firstname=firstname  
  this.lastname=lastname  
  this.age=age  
  this.eyecolor=eyecolor  
}
```

Notice that the template is just a function. Inside the function you need to assign things to `this.propertyName`. The reason for all the "this" stuff in is that you're going to have more than one person at a time (which person you're dealing with must be clear). That's what "this" is: the instance of the object at hand. Once you have the template, you can create new instances of the object, like this:

```
myFather=new person("John","Doe",50,"blue")  
myMother=new person("Sally","Rally",48,"green")
```

You can also add some methods to the person object. This is also done inside the template:

```
function person(firstname,lastname,age,eyecolor)  
{  
  this.firstname=firstname  
  this.lastname=lastname  
  this.age=age  
  this.eyecolor=eyecolor  
  this.newlastname=newlastname  
}
```

Note that methods are just functions attached to objects. Then we will have to write the `newlastname()` function:

```
function newlastname(new_lastname)  
{  
  this.lastname=new_lastname  
}
```

The `newlastname()` function defines the person's new last name and assigns that to the person. JavaScript knows which person you're










talking about by using "this.". So, now you can write:
`myMother.newlastname("Doe").`

Java Script Project

- 1- Add a random "fortune generator" to your home page. That is, your page should contain a list of fortunes (stored as an array of strings), and should randomly select one of those fortunes to display each time the page is loaded. The fortune should be displayed just above the page footer, centered and enclosed in a box. Here's an example given below.

لا تؤجل عمل اليوم الى الغد

- 2- You can find the (relatively) current exchange rates amongst several currencies in the table given below. Note that conversions are given by reading down the table. For example, 1 USD = 0.49246 GBP, and 1 CAD = 1.01941 USD.

	 USD	 GBP	 CAD	 EUR	• Egypt
	1	2.03032	1.01941	1.41544	5.96505
	0.49246	1	0.50221	0.69714	8.82994
	0.98054	1.99169	1	1.38814	4.91902
	0.70641	1.43448	0.72037	1	7.78439
	0.16764	0.11325	0.20329	0.12846	1

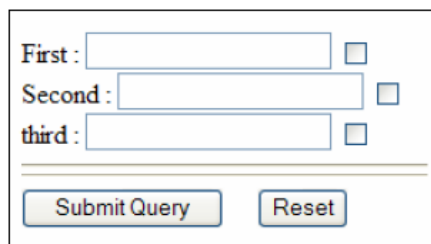
Create a Web page named `convert.html` that can be used to convert some value in one currency to the respective values of the other currencies. That is, your page should include a form which consists of five currency fields, and on inserting a number in one of the fields, this number, taken as the respective value of the currency, be converted to the values of the other currencies.

- 3- **Bonus Question:** create a Web page named `quiz.html` that can be used to conduct multiple choice quizzes over the Web. The page should contain at least 10 potential quiz questions,

each with three possible answers (A, B, and C). When loaded, the page should first prompt the person for the number of desired questions in the quiz, with a default of 5 questions. The page should then randomly select questions and prompt the user with each question and possible answers. Each answer entered by the user should be compared with the correct answer, and the result displayed within the page (either CORRECT or INCORRECT). At the end, the number and percentage of correct guesses should be displayed in the page.

Exercise 7

- 1- Write a java script function that implement Formatting a text with HTML tags?
- 2- Write an HTML document that uses an external JavaScript file?
- 3- Write a JavaScript function that use a for...in statement to loop through the elements of an array?
- 4- Write a JavaScript program that Detect the visitor's browser and browser version?
- 5- Write a JavaScript function that .Create a welcome cookie?
- 6- Write a JavaScript function that makes a simple timing alert?
- 7- Write a JavaScript function that makes a Timing event in an infinite loop - with a Stop button?
- 8- Write a JavaScript code for the following form:



First :

Second :

third :

- 9- Write a java script function that makes a simple timing alert?
- 10- How Does E-Commerce Work?