HTML, DHTML & JAVASCRIPT

Table of Content

CHAPTER 1: Introduction to Web and Internet 3
CHAPTER 2: What You Need for a Web Site 14
CHAPTER 3: Creating a Web Page and Entering Text 21
CHAPTER 4: Changing and Customizing HTML Text 27
CHAPTER 5: Displaying Text in Lists 35
CHAPTER 6: Adding Graphics to Your Web Pages 43
CHAPTER 7: Hypertext and Creating Links 51
CHAPTER 8: Clickable Image Maps and Graphical Interfaces 59
CHAPTER 9: HTML Forms 67
CHAPTER 10: Images, Multimedia Objects, and Background Graphics 83
CHAPTER 11: Netscape Frames 87
CHAPTER 12: Introduction to DHTML 97
CHAPTER 13: DHTML and StyleSheet 99
CHAPTER 14: An Overview of JavaScript 113
CHAPTER 15: Writing JavaScript code 115
CHAPTER 16: Variables, Date Types and Operators in JavaScript 127
CHAPTER 17: Conditional Statements and Looping Constructs in JavaScript 134
CHAPTER 18: Built-in Objects 145
CHAPTER 19: Functions in JavaScript 166
CHAPTER 20: Document Object Model 173
CHAPTER 21: Event handling 194
CHAPTER 22: JavaScript and cookies 214
CHAPTER 23: Regular expressions in JavaScript 221

CHAPTER 1: Introduction to Web and Internet

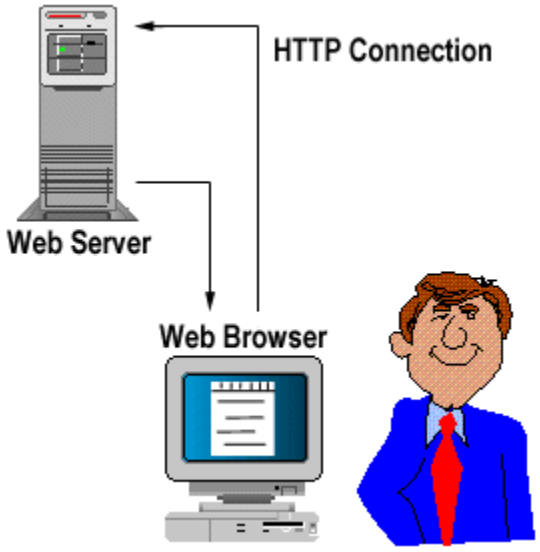
Obejctives:

- Explain Introduction of WWW
- Understand Client/Server Architecture
- Explain Internet Concepts

Introduction to Web-
The web is a complex, international, cross platform, cross language, cross cultural mesh of servers, clients, users, databases, and quite a few artificial intelligences all talking, working, searching, viewing, accessing, downloading together. Tim Berners-Lee, a computer specialist from the European Particle Physics Laboratory (CERN) in 1989, initially created the web. It is the largest client/server system implemented to date. A client/server system is a very keen way of distributing information across information systems like a local area network (LAN), a wide area network (WAN), or the Internet.

Working of client – server Architecture

A client/server system works like this: A computer (called a server) sits in some office somewhere with a bunch of files that people might want access to. This computer runs a software package that listens to requests over the networks i.e. LAN (Local Area Network) or WAN (Wide Area Network). The "server software" will then access the server hardware, finds the requested file, send it back over the wires to the "client" who requested it, and then wait for another request from the same or another client. The "client" is actually a software program, like Netscape Navigator, that is being operated by a person who is the one who really wants to see the file.
Process will look as follow:



Protocol use for communication in between client and server is HTTP(HyperText Transfer Protocol). HTTP is a "request-response" type protocol that specifies that a client will open a connection to a server then send a request using a very specific format. The server will then respond and close the connection.

These requests will be in some language and some format that the computer understands normally HTML (Hyper Text Markup Language). Upon Request of client (Web browser) sever send a response (Web page), actually the web browser will display a document exactly the way it receives it from the web server. For example, if the document requested is an image, the web

browser will display it directly. However, if the document is an HTML document, the web browser will "interpret" the HTML and display it according to the instructions contained within the HTML code.
HTML (Hyper Text Markup Language) is a very simple language used to "describe" the logical structure of a document.
Though HTML is often called a programming language it is really not. In HTML itself, there is no programming-just the "marking up" of regular text for emphasis and organization. Programming languages can be used to compute something such as the square root of pi or some other such task. Typically programming languages use conditional branches and loops and operate on data contained in abstract data structures. HTML is much easier than all of that. HTML is simply a 'markup language' used to define a logical structure rather than compute anything.

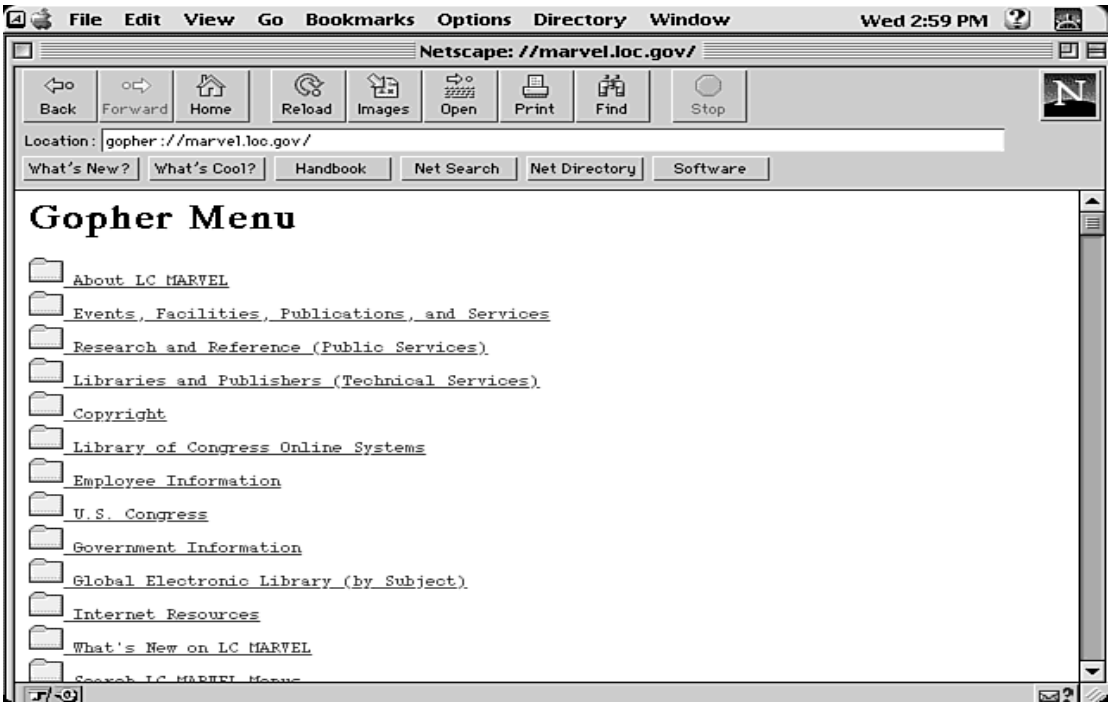
The World Wide Web and Web Servers

Probably the most important thing to remember about the World Wide Web and the Internet in general is that they are global in scale and often a very cooperative venture. Information on the Web tends to be distributed around the world, and it's just as easy for you to access a site in New Zealand or Japan as it is to access Web information in your own state.
The basic reason for learning HTML is to create pages for the World Wide Web. Before you start, though, you'll want to know a little about how this whole process works. We'll begin by taking a look at Web browsing programs, then we'll talk about how the World Wide Web works, and we'll discuss some of the terms associated with surfing the Web. Finally, we'll round out the discussion by talking about the Internet in general and the different services available on the Internet and how they interact with the Web.

World Wide Web

The World Wide Web is an Internet service, based on a common set of protocols, which allows a particularly configured server computer to distribute documents across the Internet in a standard way. This Web standard allows programs on many different computer platforms (such as UNIX, Windows 95, and the Mac OS) to properly format and display the information served. These programs are called Web browsers.

The Web is fairly unique among Internet services (which include Internet e-mail, Gopher, and FTP) in that its protocols allow for the Web server to send information of many different types



(text, sound, graphics), as well as offer access to those other Internet services. Most Web browsers are just as capable of displaying UseNet newsgroup messages and Gopher sites as they are able to display Web pages written in HTML

Here's a Gopher site as displayed through Netscape Navigator

The Web Page

The World Wide Web is composed of millions of Web *pages*, each of which is served to a browser (when requested) one page at a time. A Web page is generally a single HTML document, which might include text, graphics, sound files, and hypertext links. Each HTML document you create is a single Web page, regardless of the length of the document or the amount of information included.

The Web Site

A Web site, then, is a collection of Web pages under the control of a particular person or group. Generally, a Web site offers a certain amount of organization of its internal information. One might start with an *index* or *default* page for a Web site, and then use hypertext links to access more detailed information. Another page within the Web site may offer links to other interesting sites on the Web, information about the organization, or just about anything else.

Web site organization is an important consideration for any HTML designer, including those designing and building corporate Web sites. The typical corporate Web site needs to offer a number of different types of information, each of which might merit its own Web page or pages.

Hypermedia: Text and Graphics on the Web

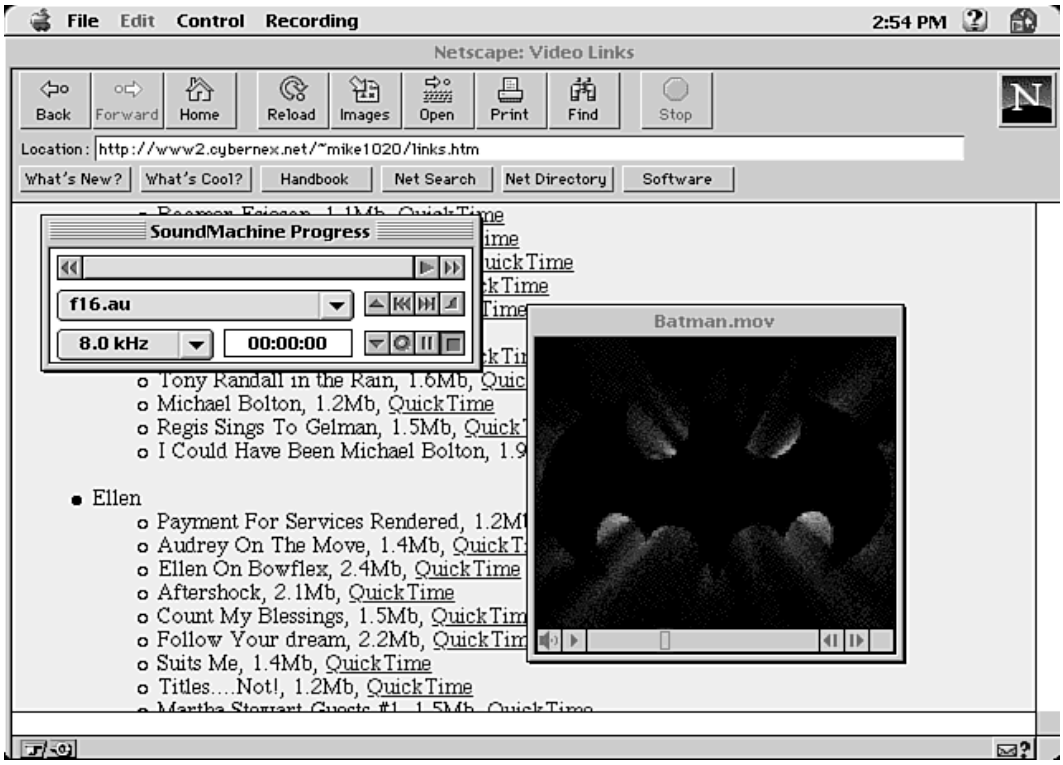
With graphical browsers such as NCSA Mosaic and Netscape Navigator, the hypertext concept of the Web was introduced to the world of multimedia, resulting in the hypermedia links that are possible in HTML.

Now, this really isn't much different from the hypertext links we talked about in the previous section-the only difference is that hypermedia links point to files other than HTML documents. For instance, a hypermedia link might point to an audio file, a QuickTime movie file, or a graphic file such as a GIF- or JPEG-format graphic.

Because of the flexibility of the Web protocol, a Web server can send these files just as easily as can an HTML document. All you need to do is create the link to a multimedia file. When users click that link, the multimedia file will be sent over the Web to their browser programs.

Helper Applications

Once the user's Web browser receives the multimedia file, it's up to the browser to decide how to display or use that multimedia file. Some browsers have certain abilities built in-especially the basics, such as displaying graphics files or plain ASCII text files. At other times, browsers will employ the services of a helper application



Examples of Web browser helper applications.

Most of these helper applications will be add-on programs that are available as commercial or shareware applications. The browser will generally need to be configured to recognize particular types of multimedia files, which, in turn, will cause the browser to load the appropriate helper application. Once loaded, the downloaded multimedia file will be fed to the helper applications, which can then play or display the multimedia file

Common Multimedia Formats

Although it seems that multimedia formats are constantly being added and improved for the Web, some of the more common types of multimedia files are listed in Table 1.1 with their associated

file extensions. This list isn't exhaustive, but it should give you an idea of the types of files that can be distributed on the Web.

File Format	*Type of File*	*Extension*
Sun Systems sound	audio	.au
Windows sound	audio	.wav
Audio Interchange	audio	.aiff, .aifc
MPEG audio	audio	.mpg, .mpeg
SoundBlaster VOiCe	audio	.voc
RealAudio	audio	.ra, .ram
CompuServe GIF	graphics	.gif
JPEG (compressed)	graphics	.jpg, .jpeg
TIFF	graphics	.tif, .tiff
Windows Bitmap	graphics	.bmp
Apple Picture	graphics	.pict
Fractal Animations	animation	.fli, .flc
VRML	3D world animation	.wrl
MPEG video	video	.mpg, .mpeg
QuickTime	video	.mov, .moov, .qt
Video For Windows	video	.avi
Macromedia Shockwave	multimedia presentation	.dcr
ASCII text	plain text	.txt, .text
Postscript	formatted text	.ps
Adobe Acrobat	formatted text	.pdf

Multimedia Formats Common to the Web

Not all of these different file formats necessarily require a special helper application. Many sound helpers will play the majority of different sound files, for instance, and some graphics programs can handle multiple file types. For the most part, you will need different helper applications for the various video, animations, and formatted text file types.

Internet Services and Addresses

Aside from being hypertext-based and capable of transferring a number of multimedia file formats, the Web is unique in its ability to access other Internet services. Being the youngest of the Internet services, the Web can access all of its older siblings, including Internet e-mail, UseNet newsgroups, Gopher servers, and FTP servers. Before we can access these services, though, we need to know what they do and how their addressing schemes work.

Internet E-mail

Internet e-mail is designed for the transmission of ASCII text messages from one Internet user to another, specified user. Like mail delivered by the U.S. Post Office, Internet e-mail allows you to address your messages to a particular person. When sent, it eventually arrives in that person's e-mail box (generally an Internet-connected computer where he or she has an account) and your recipient can read, forward, or reply to the message.

Internet e-mail addresses follow a certain convention, as follows:

username@host.sub-domain.domain.first-level domain

where *username* is the name of the account with the computer, *host* is the name of the computer that provides the Internet account, *sub-domain* is an optional internal designation, *domain* is the name assigned to the host organization's Internet presence, and *first-level domain* is the two- or three-letter code that identifies the type of organization that controls the host computer.

An example of a simple e-mail address (mine) is tstauffer@aol.com, where tstauffer is the username, aol is the domain, and com is the first-level-domain. *com* is the three-letter code representing a *commercial* entity.

First-level domain	*Organization Type*
.com	Commercial
.edu	Educational
.org	Organization/Association
.net	Computer Network
.gov	Government
.mil	Military Installation
.ca	Canadian
.fr	French
.au	Austrailian
.uk	United Kingdom
.jp	Japanese

Common First-Level Domain Names

You may have also noticed that the address doesn't include a host name or a sub-domain. For this particular address, it is unnecessary because America Online handles all incoming Internet e-mail through a gateway. Once it receives the e-mail, it may indeed send it to another computer within its online service, but this is an internal operation that doesn't require a specified host in the Internet address.

Consider todd@lechery.isc.tamu.edu. Notice how it uses all of the possible parts of an Internet address. todd is the username, lechery is a host computer (in this case, an actual, physical computer named "lechery"), isc is a sub-domain name that represents the computers in the Institute for Scientific Computation, tamu is the domain name for all Internet-connected computers at Texas A&M University, and edu is the three-letter code for *educational*, which is the type of organization that Texas A&M is considered to be on the Internet.

UseNet Newsgroups

The next Internet service we'll talk about is UseNet newsgroups. These are the discussion groups on the Internet, where people gather to post messages and replies on thousands of topics

ranging from computing to popular entertainers, sports, dating, politics, and classified advertising. UseNet is a very popular Internet service, and most Web browsers have some built-in ability to read UseNet discussion groups.
Like Internet e-mail, UseNet discussion groups have their own system of organization to help you find things. This system uses ideas and syntax that are similar to e-mail addresses, but you'll notice that UseNet doesn't require that you find specific hosts and servers on the Internet-just a particular group. UseNet newsgroup names use the following format:
first-level name.second-level.third.forth...
The *first-level name* indicates the type of UseNet group this is, the second narrows the subject a bit, and the address continues on until it more or less completely describes the group. For instance, the following are both examples of UseNet newsgroup addresses:
co.general
comp.sys.ibm.pc.misc
The first-level name co means this is a local UseNet group for the Colorado area, and general shows that it's for discussion of general topics. comp is a common first-level name that suggests this is an internationally available newsgroup about some sort of computing issue (see Table 1.3). The other levels of the name tell you more about the group.

First-Level Name	*Description*
alt	Alternative groups
biz	Business issues
clari	Clarinet news stories
comp	Computing topics
misc	Other general discussions
news	General news and help about UseNet
rec	Recreational topics
sci	Scientific discussions
soc	Social issues
talk	Debate-oriented groups

Common UseNet First-Level Newsgroup Names

Gopher and WAIS

Gopher has been described as the poor man's Web, and it's definitely true that Gopher is a precursor to some of the Web's capabilities. Gopher is a system of menu items that link sites around the world for the purpose of information retrieval. This isn't a hypertext system like the Web, but it is similar to the Web in that it's designed for document retrieval.
While Gopher can only offer access to text files and allows you to download files using the FTP protocol, it is still used occasionally by academic, government, and similar sites. Fortunately, your Web browser can easily offer Gopher access too, so there's no need to have a separate application.

WAIS, or *Wide Area Information Servers*, are basically database servers that allow you to search databases that are attached to Gopher menus. Library databases, academic phonebooks, and similar information are kept in WAIS systems.

Gopher and WAIS both generally require that you have the exact address of the Gopher server available to you. These addresses are in the following form:
host.sub-domain.domain.first-level domain

This works essentially like an e-mail address without a username. All the Gopher application needs to know is the exact Internet location of the Gopher server computer you'd like to talk to.

An example might be marvel.loc.gov. This takes you to a Gopher menu for the Library of Congress.

FTP

The File Transfer Protocol (FTP) is the Internet service that allows computers to transfer binary files (programs and documents) across the Internet. This is the *uploading/downloading* protocol that you might use to obtain copies of shareware or freeware programs, or that might be useful for downloading new software drivers from a particular computer hardware company.
Using a model identical to the Gopher system, FTP addresses use the following format:
host.sub-domain.domain.first-level domain

Like Gopher addresses, an FTP address is simply the Internet address of a particular host computer. In fact, the same host address can be used to serve you both Gopher documents and FTP file directories, based on the type of protocol your access software requests. The following example is the FTP address for downloading support and driver files for Apple Macintosh computers and Apple-created Mac and Windows software:

ftp.support.apple.com
In most cases, FTP connections also require some sort of *login* procedure, which means you'll need a username and password from the system administrator to gain access. The majority of public FTP sites, however, are anonymous sites, which allow anyone access to their files. For these sites, the username is generally anonymous, and you're asked to enter your e-mail address for the system's password.

CHAPTER 2: HTML's Role on the Web

Obejctives:

- Explain Advantages and Disadvantages of Web
- Current state of HTML

Another emerging use for HTML on the Web is as a basis for something called a *Web application*. In essence, a Web application is a Web site designed to do more than simply present pages and hypermedia links to its users-it actually acts as a front end for data processing. Once the data are entered on the page, the Web server passes them to programs that process the information looking up the product in the database or taking the order. The results of these programs can be generated complete with HTML codes, so that the answers can be viewed by the salesperson in her Web browser.

Advantages and Disadvantages of the Web

Most small or large businesses have a compelling reason to create a presence on the World Wide Web. It's an important new medium for communication that is relatively inexpensive to implement, it's a boon for dealing with customer service issues, and it's gaining popularity in leaps and bounds. But any good HTML designer should realize that there are also certain disadvantages to the Web.

Advantages

There are many good reasons to commit to creating a presence on the World Wide Web. Most of these are geared toward businesses, but you'll notice that these advantages are available to any Web site:

Multimedia presentation-A Web site allows you to do things that are simply not possible in any other medium. With some of the visual impact of television, the informational utility of print, and the personal appeal of radio, the Web is an effective tool for taking marketing information to another level. Products can be explained and offered in depth, along with pictures, video, sound, and even animation.

Interactivity-There are a number of different areas where the fact that your user can interactively determine what to view or hear can really make the difference for a business. Especially important is the added value the Web gives you for customer service, technical or product support, and immediate feedback. While most of any Web site is automated, it gives you an opportunity to answer frequently asked questions and point customers to resources that may help them solve problems on their own. While this may seem like an advantage reserved for computer companies, consider the implications for service-oriented industries like travel, consulting, catalog sales, and business-to-business sales.

Flexibility-If your business relies on printing or publishing as a medium, you may immediately see the advantage of the Web. Changes on the Web are relatively instantaneous, and the speed with which an update can be made is measured in minutes, not weeks. Consider the financial planner's or real estate agent's sales newsletter. Instant changes on the World Wide Web give their Net-savvy clients a time-based edge. Incorporating the Web into the services you offer a client gives you an added value in their eyes, especially in time-sensitive industries.

Easy High-Tech-Whether you're a small or large business, it's important to keep up with technology in order to satisfy customers and be up on the "latest." Web pages are moving toward a point where they'll be expected of large businesses and not unusual from small ones. Like e-mail a couple years ago, and fax machines before that, it's become important to keep up with the Web. Fortunately, it's also rather easy to get started with HTML and quickly develop a Web site.

Disadvantages

It's difficult to say that there are disadvantages in having a Web site, since most people and companies will use a Web site to enhance their marketing and customer service efforts, not supplant them. That said, there are a few hurdles to leap, and they should definitely be considered before your Web project takes off:

Learning Curve-It will take a while for folks to learn HTML, figure out how to upload pages, create appropriate graphics, and design effective Web sites. You'll also need to find an effective and helpful Internet service provider (or a similar in-house IS employee at a larger corporation) who can help you get online.

Appearance-To be truly effective, a Web site also needs to be attractive and easy to use. For many companies, especially larger ones, that will mean using professional artists, writers, and designers. Beginning this task can be daunting, and will require a reasonable budget-which may be intimidating when management isn't sure what the benefits will be.

Maintenance and Timeliness-One of the worst things that can happen to a Web site is for it to sit dormant for weeks or months because it's the pet project of an interested employee who has less time for it than she originally anticipated, or because every change to the Web site must first be approved by a committee. It's important that a Web developer be relatively free to spend time on the project, and that someone be available to make timely decisions. Without this, the Web site loses some of its inherent advantages.

Security-Transmitting data via Internet technology, including the Web, is inherently a rather unsecured process. For data to be transmitted over the Web, it has to pass through a number of different servers and hosts-and any of the information you offer could potentially be read or held by any of these people. This has been a strong argument against commerce on the Web, as people recognize the dangers in revealing personal information (for instance, credit card numbers). Currently, it's difficult to create completely secure Web sites that offer access only to password-bearing users, and those passwords are often not impossible to intercept.

Copyright Issues-The lack of security holds true for the Web designer-nearly anything you create on the Web can easily be read or copied by anyone with Web access. This is intimidating both to artists and publishers who want to make sure that Internet access doesn't, in some way, devalue their published (and profitable) efforts.

Cost-Depending on the size of your organization and the expertise of its people, a Web site can quickly become expensive. Learning HTML and creating a reasonable site isn't that difficult (as you'll see in this book), but maintaining the appropriate equipment, paying the dedicated staffers, and bringing in consultants, designers, programmers, and IS technicians as the site grows can quickly expand the budget. The advantages will often outweigh these costs, but any Web developer should be aware that Web sites tend to get bigger and more time-consuming as time goes on.

Secure Connections on the Internet

Some Web server software packages offer an implementation of the Secure Sockets Layer (SSL), a protocol that sits "on top" of TCP/IP (the Internet networking protocol) and "below" HTTP. Its purpose is to secure the transmission of HTTP data over the Web.

With an SSL server (usually noted by its **https:**//-protocol URL) and an SSL-capable browser program, transmissions over the Web are encrypted in such a way that users trying to read the data as they pass over the Internet are treated to nothing but garbled text.

SSL is a feature of, among others, the Netscape Enterprise Server, which is designed to allow users to access a Web site in a secure fashion so that credit cards and other personal information can be passed with relative assurance.

Although this is not directly relevant to HTML designers, if you have the opportunity to create a commercial Web site (or otherwise ask for personal information from users), you might look into the possibility of using an SSL-based secure Web server to offer your users peace of mind. And, while SSL isn't the only security scheme, it's the most widely supported.

The Current State of HTML

With these commercial demands, however, have come different solutions. For every extension Netscape adds to HTML, there is generally (eventually) a standard agreed to by the World Wide Web Consortium (W3C) that meets the same need. Unfortunately, the implementation isn't always the same. So, it's possible for an HTML 4.0 level standard, for instance, to provide for exactly the same layout functions as Netscape-but do it in a way that isn't compatible with Netscape's browser.

So HTML is currently in a bit of a flux. The best you can hope for is that the HTML standard is agreed upon and maintained more quickly in the future as more ideas pop up. At the same time, it's important that the standard remain well thought-out, and that it isn't allowed to become bloated and unworkable.

In fact, this is probably the justification for recent changes to the standard's bodies. With the W3C taking control of HTML, it suggests a shift in the ultimate power over HTML to the corporate players. From now on, you can probably assume that HTML extensions beyond what is generally considered HTML 3.0 will become standard on a case-by-case basis. Overall, this is probably a good thing, since standards can be agreed on as technology emerges-and competing browsers can all use the same methods to incorporate new technology.

CHAPTER 2: What You Need for a Web Site

Obejctives:

- Understand what is web Server.
- Speed of web server, ISP for web sites
- Discuss uploading of pages on web server.

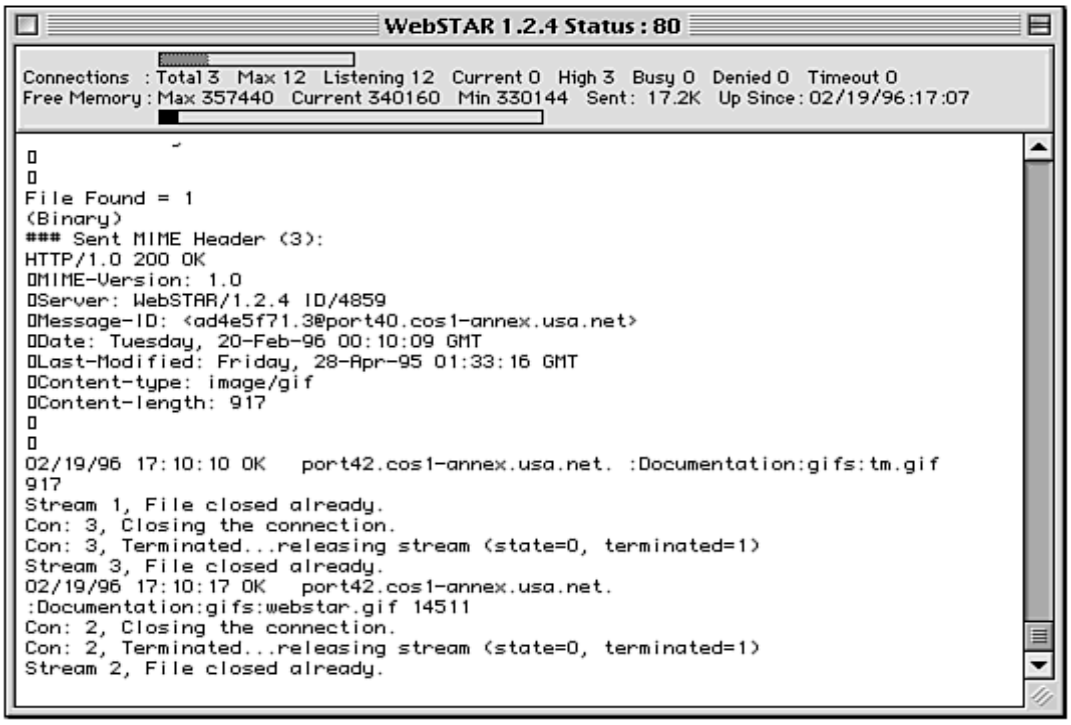
Although creating HTML pages is easily the most time-consuming part of building your Web site, another equally important part is figuring out how you're going to get those pages on the Web. You'll need Web server software, an Internet connection, a Web URL for your pages, and a system for organizing your pages and graphics. Depending on how you gain access and how complicated your site is, just getting your first page up on the Web can take a certain amount of planning.

Web Server

Before you can display your HTML pages on the Web, you'll need access to a Web server. This may already be taken care of for you, especially if you work with an Information Systems (IS) department in a larger corporation. If this is the case, you'll just need to know how and where to send your HTML files when you want to update the site. Otherwise, you'll need to make some arrangements on your own.
It isn't terribly difficult to set up your own Web server-especially if you already have a high-speed connection to the Internet. If you access the Internet through an Internet service provider (ISP), you'll want to discuss this with them, though. More than likely, they're willing to provide you with space on their Web server computers. If your Web site is a fairly small venture, or if you're not ready for a heavy investment in equipment, then using your ISP's Web server is a great (and very common) alternative.

What is a Web Server?

In its essence, it's the job of a Web server to accept connections from Web browsers all over the Internet and, when requested, send them the HTML documents that are available from your site.
A Web server is simply a computer with an Internet connection that runs software designed to send out HTML pages and other file formats (such as multimedia files) (see fig. 3.1). The server computer should have a relatively high-speed connection to the Internet (faster than any available modem connections, for instance) and be powerful enough to deal with a number of simultaneous connections from the Internet.



WebSTAR Web server software running on a Macintosh computer.

Web server software generally requires a fairly robust operating system (like UNIX, Windows NT, or OS/2), although software is available for other versions of Microsoft Windows, and the Macintosh OS is a very popular choice for Web server computers. The software you use depends on your level of experience with Internet connections and various operating systems.

Speed of the Server

The other major consideration is how popular your Web site will be. The more *hits*, or connections, your Web server receives at one time, the more powerful the computer should be-and the faster your connection to the Internet.

Most Internet connections are measured in terms of *bits per second* (*bps*), which translates loosely as "how many bits of data can be transmitted across the Internet in a second." In computer, it takes eight bits to make up one *byte* of computer information-and a byte is what is required to create a character of text.

The typical modem connection is 14,400 bps, which translates to roughly 1,800 characters (bytes) transferred every second. If a typical page of text contains 300 words then, and each word averages six characters per word, this connection would yield roughly a page-per-second transmission rate. A 25-kilobyte (KB) file (such as a very small GIF file) would take about 14 seconds to transmit over this connection.

This doesn't sound terribly slow, until you start to take into account the idea that more than one connection might occur with the Web server. If ten people connect to our server over this connection, it will take ten seconds to complete the task of sending each of them a single page of data. If that page totaled 25 KB in size (that is, if it included graphics and other elements), it could take over 140 seconds to complete that same task.

These transmission rate numbers all reflect ideal conditions. In real life, phone line noise, traffic on the Internet, and other factors will slow down transmission rates. Throughput on a 14,400 bps connection is often somewhere between 1,100 and 1,300 characters per second.

If the typical well-designed Web page is between 30 KB and 50 KB in size, you can see that we're going to start running into problems with this type of connection. There's the potential for someone to wait a number of minutes between the transfers of each page they request on your

Web site. If the average commercial break on television is three minutes, just think how annoyed your users are going to get.

Types of Internet Connections

So your server will need a faster connection. If Internet access is available to you through your company's Local Area Network (LAN), you probably already have a high-speed connection. Ask around your IS department. If you're running a small business or home office, you won't have to worry about high speed if you make your Web pages available on your ISP's Web server.
If you're going to use your own Web server computer, though, you'll need a high-speed Internet connection that you can connect to that computer. Details some of the possible connections.

Connection Speed	*Connection Technology*
14.4/28.8 Kbps	High-speed modem
56 Kbps	56K leased line
64 Kbps	Single-B-Channel ISDN
128 Kbps	Basic Rate ISDN
up to 1.5 Mbps	Primary Rate ISDN (U.S.)
1.5 Mbps	T-1 dedicated line
45 Mbps	T-3 dedicated line

Internet Connection Speeds and Technologies

The minimum for an acceptable Web server connection is probably a basic-rate ISDN (Integrated Services Digital Network) connection, which offers 128,000 bps connections to the Internet. ISDN technology uses your existing phone wiring to provide an enhanced, digital, telephone connection. Using a special network adapter card for your computer, you can use the ISDN line to dial an appropriately equipped ISP. You can also use the ISDN connection for regular telephone calls.
A T-1 line is the typical connection for an ISP or a large business, and these lines generally cost thousands of dollars per month for Internet access, as do primary-rate ISDN connections. T-3 lines currently serve as the backbone of the Internet, and are generally only found connecting university, government, and supercomputing organizations.

Dealing with an ISP

For any sort of connection to the Internet, you'll probably need to deal with an Internet service provider. These companies offer dial-up and special high-speed connections to the Internet, as well as generally offering Web and other types of Internet servers for your use.

For the typical smaller Web site, you'll want to buy space on the ISP's Web site. Generally this will give you an URL that begins with the name of the ISP's host computer, but points to a special directory for your HTML pages, such as **http://www.isp.com/username/index.html**.

With most Web server programs, the default page that is first loaded is named index.html, so that's the name you'll use for the first page you'd like presented to users when they access your Web site.

Uploading HTML pages to the server

Once you've decided on an ISP that you feel is reasonably priced, you're ready to create your HTML pages and upload them to the server. To do all this correctly, though, you'll probably need to keep these points in mind:

- **Site's default URL**-This should be something like the ISP's host address and a directory for your username. For instance, if my username is **tstauffer** and my ISP's Web server is **www.webco.net**,then then the default URL for Site will be http://www.webcom.net/tstauffer/.
  Different ISPs will organize this in different ways, so you'll need to make sure you get this right.

- **Uploading files to the site's directory**-You should get instructions for accessing your Web site's directory on the Web server computer using either FTP or a UNIX shell account.

- **Limitations to the names you can give to your files**-The operating system in use by the Web server may not be instantly obvious to you. If this is the case, you'll want to ask if there is a certain filename length or a certain format for naming files you need to follow.

- **Creating subdirectories within your main Web site directory**-Most Web servers will give you this capability, but some will not allow you to create new subdirectories.

- **Support offered for CGI programming** - Some servers wouldn't allow you to add CGI scripts to your Web site for processing forms or adding other interactive features. At the same time, some will, but require you to pay extra or pay to have the provider write those scripts (regardless of your ability). If you plan a highly interactive site, then you should ask about CGI support.

Organizing a Web Site

The most important thing to remember when organizing a Web site is how the server computer you're using will differ from the computer you use to create Web pages. This is because you'll need to know the exact path to HTML pages and multimedia files you use in creating your Web page. As we've seen before, an URL requires both a server name and a path statement to the file. This includes files that you've placed on your own Web server-so while you're creating your Web pages, you'll need to know where your files will eventually be.

Although there are a number of different ways to arrange a Web site, there are some rules of thumb to keep in mind. For the most part, any organization you create for your Web site files should be designed to make updating your pages easy in the future. If you have to move all your files around every time you change something on a Web page, you'll also be forced to change all the hypertext links on many other pages-and that can be incredibly time-consuming.

Different types of organization for Web sites:

- **Single-directory sites**-Smaller sites (with just a few HTML pages and graphics) can often get by with a single directory on the Web server. All your graphics and HTML pages are in this one directory. One of the biggest advantages of this system is that links to local files and graphics require no special path statements.
- **Directory by function**-One way to organize more complicated sites is to put each section of related Web pages in the same directory. For instance, in your main directory you might offer only your first (index) page and its associated graphics. For a business

site then, you'd have subdirectories for About the Business, Product Information, Technical Support, and so on. In each of these subdirectories, you'd include all the related HTML files and the graphics for those pages.

- **Directory by file type**-Some people prefer to create subdirectories according to the type of file as opposed to the content of the page. Your main directory may have only the index page of your site. Other subdirectories might be Graphics, Web Pages, Downloadable Files, and so on. The main advantage in organizing this way is that files generally have to be replaced only once. If you use a graphic on a number of different pages, for instance, you replace it once in the Graphics subdirectory, and all the HTML pages that access this graphic will use the new one.
- **Hybrid**-The best way to organize a large site might be a hybrid of the last two methods above. Creating separate subdirectories for nonrecurring items (such as individual Web pages in each category) while creating other subdirectories for items used multiple times (such as graphics) lets you get to all the files in an efficient way.

Naming Your Files

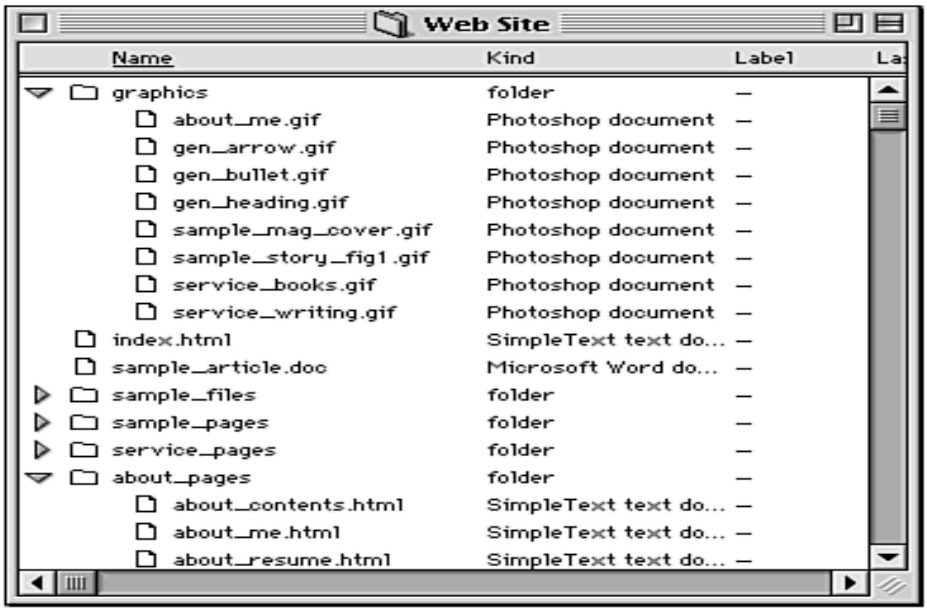
We've already mentioned that file extensions are an important part of all the filenames you use for your Web site. Because other Web browsers may rely on the file extension to know what sort of document or file it is, you'll need to include the appropriate extensions with all your Web site files. Web site will almost always begin with a file called index.html. Most Web server software programs will automatically load this page if the URL of your site is accessed without a specific path and file reference. For example, entering http://www.sun.com/ in your browser actually results in the URL http://www.sun.com/index.html being loaded in your browser. Web site's first page (whether it's a "front door" page or the first page of your site) should be designed with this in mind. If you plan to offer only Netscape-enhanced pages, for instance, you'll want to let your users know this on the index.html page. The other consideration for naming your files is the organization you plan to use for your site. If you're using a single-directory organization, your filenames should be as unique as possible, and graphics and other files should probably have names that relate to associated Web pages.

For instance:
about_company.html
about_header.jpeg
about_ceo_photo.jpeg

When possible, these names will help you determine which files are associated with which HTML pages when you go to update those files.
For graphics and other files that show up on multiple pages, you might want to come up with a memorable prefix, like gen_ or site, just so you can easily replace these universal files when necessary.

Example: Organizing a Site

To create a reasonably sized site for home-business Web site, we are going to use the hybrid style of organization. We have three different sections on my site: About our Business, Services, and Samples. Each of these sections will have its own directory structure. Graphics will be in their own subdirectory, as will downloadable files that we are including



The directory organization for my site.

The directory names, then, will be as follows:
about_pages
service_pages
sample_pages
graphics
sample_files
Files and graphics are named for where they appear, unless they show up in multiple Web pages. For this site, the prefixes I'm using are as follows:
about_
serv_samp_
gen_
index_
By naming files in this way, I'll be able to replace any graphics or update my sample files easily-without being forced to load each file or graphic to figure out what it is. Making the names as descriptive as possible (aside from the prefix) will help too, as in the following:
about_photo_me.jpeg
samp_resume1.doc
sampl_catalog_copy.txt

Updating Your Web Site

If you organize your site well, updating the site is simply a matter of replacing an outdated file with a new file using the same filename. For instance, if I wanted to replace the picture of me in the previous example, I'd simply name the new file about_photo_me.jpeg, and save it in the same directory. Now the associated Web page will load the new graphic without requiring any changes to the HTML codes.
You'll need to check with your company's IS contact or your ISP to figure out exactly how you'll update files. With an ISP, you can generally use an FTP program to put new files in your directory organization on the Web site. You might instead be required to use a UNIX-based shell account for your uploading. In either case, it's a fairly simple process.
Your Web space provider will require you to enter a username and password to gain access to the Web server, whether by FTP or shell account. Generally, you will point your FTP server to the

Web server itself (for instance, **www.isp.com**), unless the provider has created a *mirror* site to avoid direct access to the Web server.

After clearing the security procedure, you'll most likely be in your personal Web site's main directory. (If not, you'll need to use the cd command in UNIX or otherwise change directories in your FTP program.) From that point, you can update files using the Put command. Simply upload the updated files with the same names as the outdated files-in nearly every case, the old files will simply be overwritten. If you're using new files, upload them using the names and paths that your Web page links use to refer to them.

Tip: It's a good idea to maintain a folder or directory on your own hard drive that is as identical as possible to the Web site you make available on a server-so you can test your organization and filenames.

CHAPTER 3: Creating a Web Page and Entering Text

Obejctives:

- Use various tools of web designing.
- List type of tags used in HTM pages
- Create a HTML page

With the basics behind you, it's time to start creating your first HTML pages. As has already been mentioned, the basic building block of an HTML page is text. To create these pages, all you really need is a text editor and a Web browser for testing your creation (you'll eventually need a graphics program to create and edit your graphics, too). So let's look at the basic tools for Web publishing, and then create your own HTML template.

The Tools for Web Publishing

We have already mentioned it above all you need is a text editor. In Windows 95, that's Notepad or WordPad. For Mac users, SimpleText is the perfect HTML editor. UNIX users can opt for VI or Emacs. Basically, all you need to remember is that HTML pages, while they include the .htm or .html file extensions, are simply ASCII text files. Any program that generates ASCII text files will work fine as an HTML editor-even a word processor like WordPerfect or Microsoft Word.

Tip :If you create an HTML page in a word processor, don't forget to use the Save As command to save it as an ASCII text file.

We'll also need a Web browser to check on the appearance of your Web page as you create it. All Web browsers should have the ability to load local pages from your hard drive, just as they can load HTML pages across the Web. Check the menu of your Web browser (if it's a graphical browser) for a command like File,

In Microsoft Internet Explorer for Windows 95, the File, Open command opens the the Open Internet Address dialog box which contains an Open File command button to open a file from a drive.

You may have heard of some dedicated HTML editing programs that are designed to make your work in HTML easier. They do indeed exist, and they can be very useful. Unfortunately, many of them also hide the HTML codes from the designer, so they would be difficult for us to use as you learn how HTML works. Once you understand HTML, though, it can be a great benefit to use one of these browsers.

Document Tags

The first HTML tags we're going to look at are the document tags. These are the tags that are required for every HTML page we create. They define the different parts of the document.

Just like a magazine article, an HTML document has two distinct parts-a head and a body. The head of the HTML document is where you enter the title of the page.
To create the head portion of HTML document and to give the document a title, type the following in your text editor:

```
<HEAD>
<TITLE>My First Page</TITLE>
</HEAD>
```

This tells a Web browser what information should be considered to be in the head portion of the document, and what it should call the document in the title bar of the browser window.
The body is where we'll do most of our work-you'll enter text, headlines, graphics, and all our Web goodies. To add the body section, start after the </HEAD> tag, and enter the following:
<BODY>

</BODY>
Between these two tags, you'll eventually enter the rest of the text and graphics for your Web page.
There's one last thing you need to consider. In order that all Web browsers understand that this is an HTML document (remember that you're saving it as ASCII text, so the browser could be confused), you need to add some tags on either side of the head and body tags you've created. Above the first <HEAD> tag, enter the following:

```
<HTML>
After the last </BODY> tag, type the following:
</HTML>
```

Now, at least as far as your Web browser is concerned, you have a complete Web document!

Example: Creating an HTML Template

Let's take what you know and create a template. By saving this template as a generic text file, you'll have a quick way to create new HTML files-simply load the template and use the File, Save As command to save it as your new Web page.
Start by entering the following in a blank text file:

```
<HTML>
<HEAD>
<TITLE>Enter Title Here</TITLE>
</HEAD>
<BODY>

</BODY>
</HTML>
```

Now save this as an ASCII text file called template.html (or template.htm if you're using DOS or Windows 3.1). Now, whenever we are ready to create a new HTML document, simply load template.html into your text editor and use the Save As command to rename it.

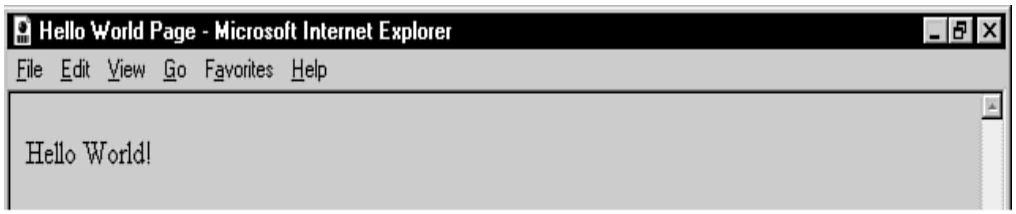
Example: Hello World

When learning a new programming language, it's traditional that the first program you create is designed to say "Hello World." Well, HTML isn't a programming language-but I can use the Hello World example to prove that your template is a complete Web document.

Load the template.html file into your text editor, and use the Save As command to rename it hello_world.html or something similar. Now, edit the document so that it looks like this:

```
<HTML>
<HEAD>
<TITLE>Hello World Page</TITLE>
</HEAD>
<BODY>
Hello World!
</BODY>
</HTML>
```

Select the File, save command from your text editor. Now load our Web browser and select the Open File (or similar) command from the File menu. In the dialog box, find the document hello_world.html and select OK to load it into your Web browser. If everything goes as planned, your browser should display something similar to the diagram below.



The Hello World page as viewed in Microsoft Internet Explorer.

Understanding Tags: Container and Empty Tags

In creating your HTML template, you've already dealt with some of the most basic tags in HTML. The first thing you should notice about these HTML tags is that all tags include < and > on either side of the tag's command. This is how HTML recognizes tags. If you don't use the brackets, then a Web browser will assume your commands are text that you want displayed-even if that text is the same as an HTML command.

While a Web browser would consider the following to be a tag:

<HTML>

that same Web browser would interpret the following as text to be displayed on-screen:

HTML

Tip: Tags are not case-sensitive, so they don't have to be all uppercase-even though that's how they appear in this book. I suggest you type them as uppercase, though, since it makes them stand out in your text editor.

Because tags aren't considered text by the document, they also don't show up in the document. If the browser interprets something as a tag, it won't appear in the browser window.

Container Tags

You may have noticed that for every tag, such as the title tag, you actually entered two different HTML commands-an "on" tag and an "off" tag. The off tag is the same as the on tag, except for the / after the <.

In HTML, tags that include both an on and an off tag are called container tags. These tags wrap around text in your document and perform some sort of formatting on the text. They hold, or contain, the text between the two tags. The title, HTML, head, and body tags are all container tags-the relevant text goes between the on and off tags.

Container tags always have the following form:

<TAG>text being formatted or defined</TAG>

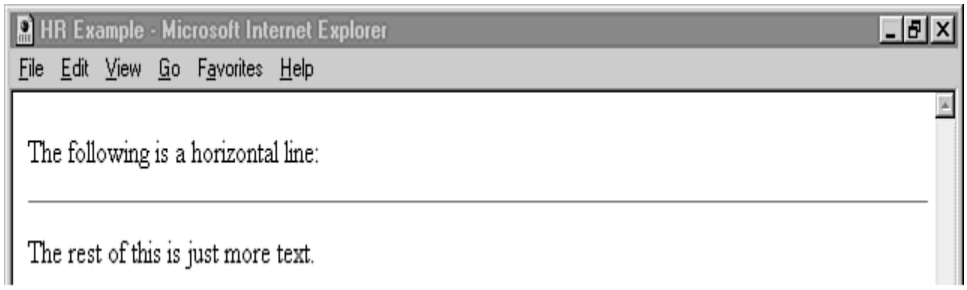
In fact, you've already been introduced to a fairly common container tag in the first chapter of this book, the <EM> (emphasis tag). An example of the emphasis tag would be:

Here's some <EM>really important</EM> text.
Because <EM> is an implicit formatting tag, it's up to the browser to decide what to do to the text between the on and off tags. But only the words really important will be affected in this example, since they're the only text that is being "contained" by the tags.

Empty Tags

All other tags in HTML fall into one other category, called empty tags. These tags have only on tag-there are no off tags. The reason for this is that empty tags don't act on blocks of text. Instead, they do something all on their own. An example of this would be the <HR> (horizontal rule) tag. This tag draws a line across the width of your document. For example:
The following is a horizontal line

<HR>
The rest of this is just more text.
When viewed in a Web browser, a horizontal line, as shown below, will separate the two sentences:



Here are your two sentences, separated by a horizontal line.

Entering Paragraph Text on Your Web Page

With our template prepared, and with an understanding of the two types of tags in HTML, we're ready to enter text on a Web page. As mentioned earlier, all the text that you enter on a page should come between the <BODY> and </BODY> tags. Like <EM>, the body tags are container tags that tell a Web browser what parts of the HTML document should be displayed in the browser window.
You've seen that you can just type text into an HTML document and it will be displayed in the browser. Technically, though, most of the text you type should be in another container tag: the <P> (paragraph) tag. This tag is used to show a Web browser what text in your document constitutes a paragraph. For the most part, Web browsers ignore more than one space between words and will ignore returns that you add to your HTML file while we're creating it.
In order to give the appearance of paragraphs, then, you have to use the paragraph container tag.
The paragraph tag uses the following format:
<P>Here is the text for my paragraph. It doesn't matter how long it is, how many spaces are between the words or when we decide to hit the return key. It will create a new paragraph only when we end the tag and begin with another one.

</P>

<P> Here's the next paragraph. </P>
Like the emphasis tag, the paragraph container tells the Web browser that all of the text between the on and off tags is in a single paragraph. When we start another paragraph, the Web browser will drop down a line between the two.
Here's that same example, except you'll throw in some spaces. Remember, spaces and returns almost never affect the way the text will be displayed on the screen. In a paragraph container, the browser will ignore more than one space and any returns.
<P>Here is the text for my paragraph.
It doesn't matter how long it is, how many spaces are between the words
or when I decide to hit the return key. It will create a new paragraph
only when I end the tag and begin with another one. </P>

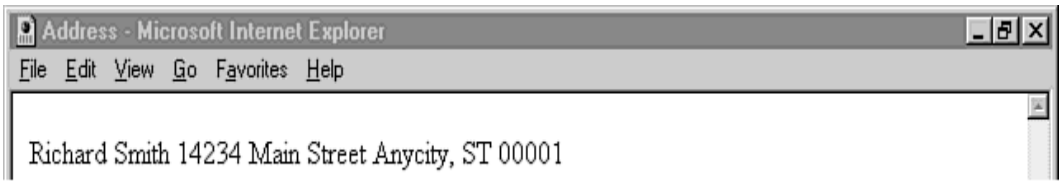
<P> Here's the next paragraph. </P>
Both this example and the previous example will be displayed in the Web browser in exactly the same way.

The
 Tag for Line Breaks

But what if you want to decide where a line is going to end Consider the example of entering an address in a Web document, as follows:

```
<P>
Richard Smith
14234 Main Street
Anycity, ST 00001
</P>
```

It looks about right when you type it into your text editor. However, when it displays in a Web browser, it looks like



The Post Office would never deliver this.
We already know what the problem is: Web browsers ignore extra spaces and returns! But if we put each of those lines in a paragraph container, we'd end up with a space between each line- and that would look wrong, too.
The answer is the empty tag
, which forces a line returns in your Web document. Properly formatted, your address would look like this:

```
<P>
Richard Smith<BR>
14234 Main Street<BR>
Anycity, ST 00001<BR>
</P>
```

And it would look just right in your Web browser, just as in figure below

This address looks much better

The Comment Tag

There's one other tag I'd like to discuss in this chapter, called the comment tag. This tag is fairly unique, in that it's actually used to make the Web browser ignore anything the tag contains. That can be text, hypertext links, image links, even small scripts and programs.
For now, you'll use the comment tag to hide text. The point in hiding the text is that it allows you to create a private message that is intended to remind you of something or to help those who view the raw HTML document to understand what you're doing. That's why it's called the comment tag. For instance:
<! --This is a comment that won't display in a browser-->
The comment tag isn't the most elegant in HTML, but it usually works. Anything you type between <! -- And --> should be ignored by the browser. Even multiple lines are ignored-as with most tags, the comment tag ignores returns.
Generally, you'll use the comment tag for your own benefit-perhaps to mark a point in a particular HTML document where you need to remember to update some text, or perhaps to explain a particularly confusing part of your page. Since it's fairly easy for anyone to view your raw HTML document, you might also use the comment tag to create a copyright message or give information about yourself.
Viewing the Source of Web Pages: Ever been out on the Web looking at a particularly well-designed HTML document-and wondering how they did it?

If you'd like to, most browsers will let you view the document source for any Web page they can load. This allows you to download the raw HTML codes and ASCII text, just as if you'd created the page yourself.
To do this, select the View Document command in the Edit menu of your Web browser (the command may differ slightly, so look for a similar name if you can't find View Document). What results is the plain ASCII text file that was used to create that Web page.
Depending on your browser, this source file will either be displayed in the browser window, or saved to your hard drive and displayed in the default text editor. If the source is displayed in the browser window, then select File, Save As to save the source to your hard drive.
Now you might be able to imagine how comments can come in handy. If you would rather not have people copy and use the source from your Web pages (or if your pages contain otherwise copyrighted material that you want to protect), you can use the comment tag to let others know that you consider the page your property. For instance:
<! --Contents of this document Copyright 1996 Todd Stauffer. Please do not copy or otherwise reproduce the source HTML code of this document without permission. -->
Of course, that's not to say that you shouldn't also offer a visible copyright notice or other legal disclaimers. But comments within the code tend to talk directly to folks a little more HTML-savvy. Using a comment tag like this is a great way to encourage other Web designers to ask you before using your HTML pages for their own private use.

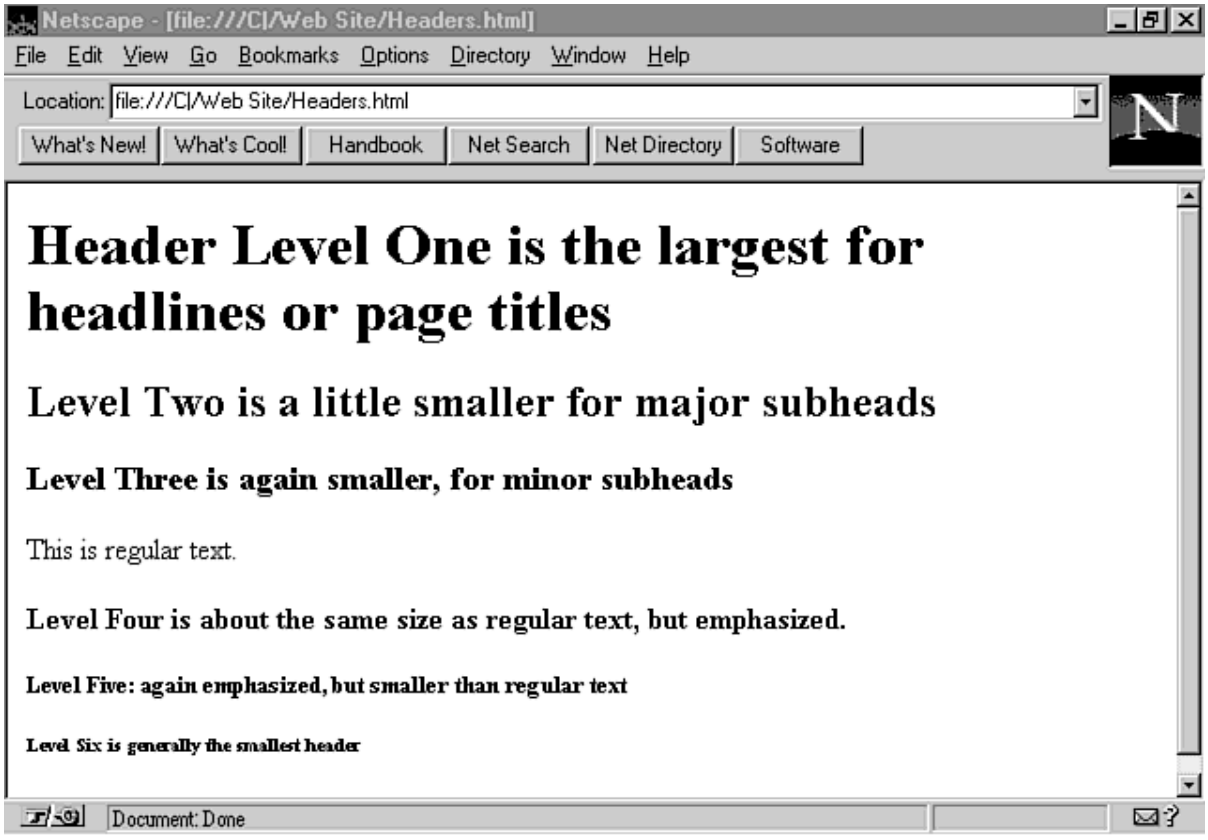
CHAPTER 4: Changing and Customizing HTML Text

Obejctives:

- Use of header and headline in document
- Quoting, Citing, Definitions, and Addresses
- Preformatting of text

Creating Headers and Headlines

Header tags are containers, and unlike many other HTML tags, they double as paragraph tags. Ranging from level 1 to level 6, headers allow you to create different levels of emphasized headlines to help you organize your documents. The following is an example; see figure 5.1 for the results:



HTML header tags at work. Notice that the fourth entry is regular text between <P> and </P>tags.

```
<H1>Header Level One is the largest for headlines or page titles</H1>
<H2>Level Two is a little smaller for major subheads</H2>
<H3>Level Three is again smaller, for minor subheads</H3>
<P>This is regular text. </P>
<H4>Level Four is about the same size as regular text, but emphasized</H4>
```

```
<H5>Level Five: again emphasized, but smaller than regular text</H5>
<H6>Level Six is generally the smallest header</H6>
```

We cannot include a header tag on the same line as regular text, even if we close the header tag and continue with unaltered text. A header tag has the same effect as a <P>, in that it creates a new line after its "off" tag. The following:

<H1>This is a header</H1> And this is plain text.
offers the same results as:
<H2>This is also a header</H2>
<P>And this is also plain text</P>

In both cases, the Web browser will place the header text and plain text on different lines, with the header text appearing larger and the plain text appearing "normal" in size.
Note: The HTML standard technically requires that using a particular header level requires that the larger header tags be used previously. So, for instance, if you use an <H2> tag, you should have an <H1> tag somewhere before it. Very few browsers (if any) actually require this and, for the most part, HTML designers use header tags as simply a way to change the size of text for emphasis. That's how I use them, even going so far as to use <H5> or <H6> for "fine print" on my pages. If you're an absolute stickler for standards, though, realize that it's more correct to only use header tags for true headers in your documents, and then only in order (i.e., <H1>, <H2>, <H3>, and so on).

Implicit and Explicit Text Emphasis

Implicit tags are those that allow the browser to choose, within limitations, how the marked-up text will be displayed. Header tags are actually an example of an implicit tag, since the HTML designer has no control over how much bigger or smaller a header tag will be. Although most browsers will render header tags in somewhat similar ways, others (for instance, nongraphical browsers) have to come up with another system for emphasis, such as underlining or highlighting the text.
Because HTML was originally created with the overriding mission of being displayed on nearly any computer system, implicit tags for emphasis were a necessity. HTML allows the designer to decide what text will be emphasized. But only explicit tags tell the Web browser how to render that text.

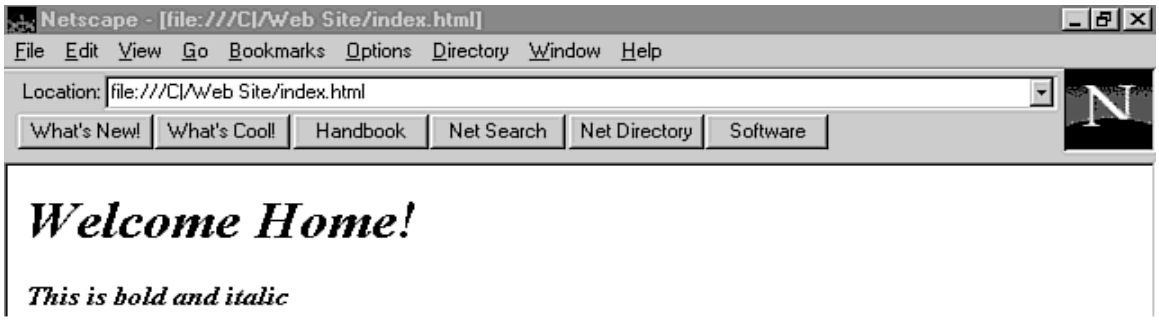
Explicit Styles

Explicit tags are also often called *physical tags*, since they very specifically tell the Web browser how you want the text to physically appear. The browser is given no choice in the matter.
The basic explicit tags are containers that let the user mark text as bold, italic, or underlined.

Tags	*Meaning*
<B>, </B>	Bold text
<I>, </I>	Italic text
<U>, </U>	Underlined text

HTML Physical Container Tags

Note: Not all browsers will render underlined text (notable among them is Netscape Navigator), because hypertext links are also often displayed as underlined, which could potentially be confusing.
With these tags, the browser really has no choice-it must either display the text as defined or, if it can't do that, then it must add no emphasis to the text. If we prefer that text not be emphasized at all if it can't be italic, for example, then we should use the <I> tag.
Another feature of explicit (physical) tags is that they can generally be used in combination with other tags. As we'll see in the next section, this isn't always a good idea with implicit tags. For instance, most graphic browsers will render the following example by applying both tags to the text



Most browsers can render two physical tags applied to the same selection of text.

<H1><I>Welcome Home!</I></H1>
<B><I>This is bold and italic</I></B>

Implicit HTML Tags

Implicit styles are often called *logical styles*, since they allow the browser some freedom in how it will display the text. These tags, like the header tags, are generally relative to one another, depending on the browser being used to view them. Some of the common implicit (logical) tags

Tags	*Meaning*	*Generally Rendered as…*
<EM>, </EM>	Emphasis	Italic text
<STRONG>, </STRONG>	Strong emphasis	Bold text
<TT>, </TT>	Teletype	Monospaced text

Some Basic Logical HTML Tags

Above table includes a section that tells you how these tags are often rendered in graphical Web browsers. There's no rule for this, though, and the tags don't necessarily have to be rendered in that way.
There are two other distinctions between these tags and the physical tags (such as bold and italic) that you've already discussed. First, any Web browser that views them will always render these logical tags. Even text browsers (which are unable to show italic text) will display the <EM> or <STRONG> tags by underlining, boldfacing, or highlighting the text.
Second, these tags are generally not effective when used together. Where <B><I>*text*</I></B> will sometimes offer useful results, <EM><STRONG>*text*</STRONG></EM> rarely will. Combining these tags with other tags (such as header tags or physical tags) is often either ineffective or redundant.

Other Implicits: Programming, Quoting, and Citing

At the beginning of this chapter, I mentioned that the creation of HTML tags took place before the standard was ever conceived of-which might explain some of the tags that we discuss in this section. For the most part, these tags are implicit (logical) and aimed directly at certain areas of expertise. At the same time, however, the bulk of these tags will look exactly the same in a Web browser.

Programmer's HTML Tags

One of the early, more common uses for HTML was for documenting computer programs and offering tips or advice to computer programmers. Part of the HTML 3.0 standard, then, offers some implicit (logical) HTML tags that allow HTML designers to mark text in a way that makes it easier to present computer-programming codes. Those tags are listed in table below.

Tags	*Meaning*	*Generally Rendered as…*
<CODE>, </CODE>	Programming lines	Monospaced (like <TT>)
<KBD>, </KBD>	Keyboard text	Monospaced
<SAMP>, </SAMP>	Sample output	Monospaced
<VAR>, </VAR>	Variable	Italic

HTML Tags for Computer Programming

Notice that the majority of these tags are often displayed in exactly the same way-in the default monospaced font for the browser. Then why use them?
First, not all browsers will necessarily follow the "general" way. Some browsers will actually render these tags in slightly different ways from one another, so that <SAMP>, for instance, might appear in a slightly larger font than <CODE>.
Second, using these tags is a great way to internally document your HTML pages, so that you can tell at a glance what certain text is supposed to be. This will help you later when you return to the document to update it or fix errors-especially as the document becomes more complex.

Quoting, Citing, Definitions, and Addresses

Along the same lines as the HTML "programmer's" tags, you have available certain implicit tags that work as typographer's or publisher's codes. As shown in below table, these codes often work in ways similar to others you've already seen-with a few twists.

Tags	*Meaning*	*Generally Rendered as…*
<CITE>, </CITE>	Bibliographical citation	Italic text
<BLOCKQUOTE>, </BLOCKQUOTE>	Block of quoted text	Indented text
<DFN>, </DFN>	Term definition	Regular text
<ADDRESS>, </ADDRESS>	Street or e-mail address	Italic text

HTML Publisher-Style Tags

Again, notice that the <CITE> tag isn't going to be rendered any differently from the italics, emphasis, or variable tags we've seen previously. The <DFN> tag is often not rendered as any

special sort of text at all, whereas the <ADDRESS> tag is identical in function to the italics tag. So the best use for these tags (with the exception of the <BLOCKQUOTE> tag) is as internal documentation of your HTML documents.

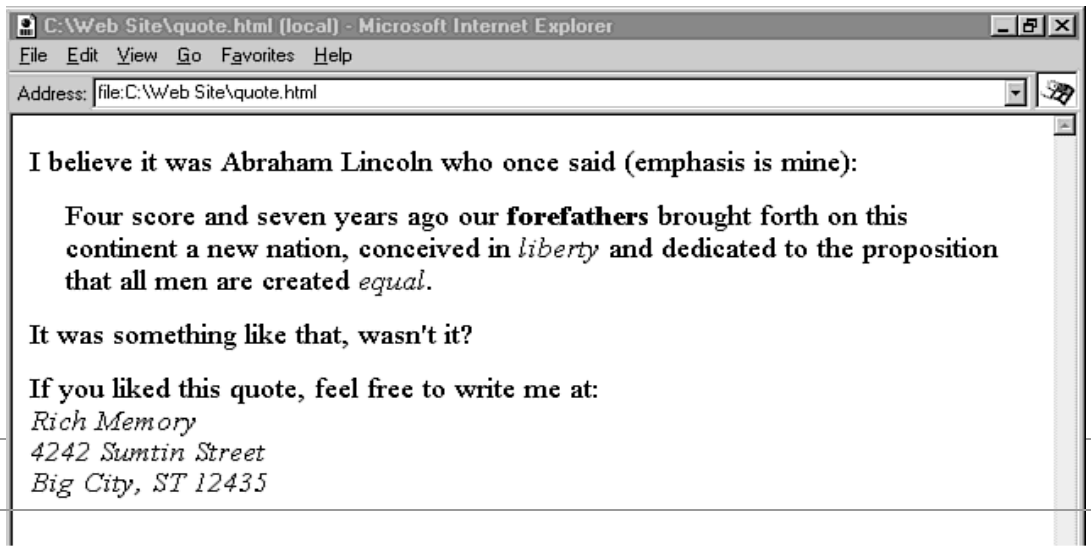
Example: Using the <BLOCKQUOTE> and <ADDRESS> Tags

The only really new tag in the above table is the <BLOCKQUOTE> tag. This tag usually indents the left margin of regular text in the browser window, just as you might find a blocked quotation formatted in a printed document.
Also as part of the tag, <BLOCKQUOTE> generally adds a return or one extra line on either side of the tag, so no paragraph tags are needed. Paragraph tags should, however, be used to contain text on either side of the blockquote.
Although the <ADDRESS> tag is similar to italics or emphasis, I've thrown in an example of using it correctly. Remember to include a line break after each line of the address.
Listing -between the body tags.

emphasis.html The <BLOCKQUOTE> and <ADDRESS> Tags

```
<BODY>
<P>I believe it was Abraham Lincoln who once said (emphasis is mine):
<BLOCKQUOTE>Four score and seven years ago our <B>forefathers</B> brought
forth on this continent a new nation, conceived in <I>liberty</I> and
dedicated to the proposition that all men are created <EM>equal</EM>.
</BLOCKQUOTE>
It was something like that, wasn't it?
</P>
<P>If you liked this quote, feel free to write me at:<BR>
<ADDRESS>
Rich Memory<BR>
4242 Sumtin Street<BR>
Big City, ST 12435<BR>
</ADDRESS>
</P>
</BODY>
```

Notice that an off paragraph tag isn't required before you get into the address tag-remember, <ADDRESS> works very much as italics does, and the
 tag is designed to work as well inside a paragraph container as it does outside one. So you can put the paragraph tag after the address, to contain both address listing and the text in the same paragraph.
Take a look at next figure <BLOCKQUOTE>, unlike some of the tags you've looked at, really



does offer unique abilities that make it worth using in your documents.

Blockquote and address HTML tags.

Preformatted Text

The HTML 3.0 standard is not designed for layout. In fact, you haven't even learned how to put two blank lines between paragraphs. We've also said that spaces and returns in between tags (like the paragraph tag) don't matter. Well, there is at least one exception to this rule: the <PRE> tag.

The <PRE> (preformatted text) tag is designed to allow us to keep the exact spacing and returns that we've put between the on and off tags. The basic reasoning behind this tag is the notion that every once in a while we'd like your text to stay exactly as we put it-for instance, in a mathematical formula, or if we create a table. While there are other ways to do both tables and math, they don't fall under the HTML 3.0 standard. On top of that, you can use <PRE> for a number of other reasons: lists, lining up decimals for dollar figures, and even poetry.

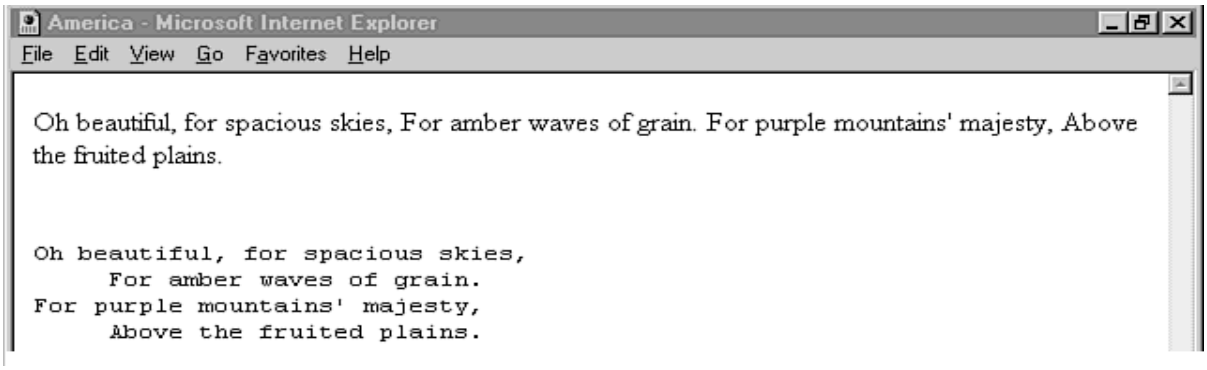
Consider the following example:

```
<P>Oh beautiful, for spacious skies,
For amber waves of grain.
For purple mountains' majesty,
Above the fruited plains.</P>
```

Sure it's a familiar refrain, but it won't look so familiar in a browser if we leave it between paragraph tags. Instead, we can use the <PRE> tag to keep things exactly the way we want them:

```
<PRE>Oh beautiful, for spacious skies,
    For amber waves of grain.
For purple mountains' majesty,
    Above the fruited plains.</PRE>
```

In a browser, it'll look exactly the way we want it to



Paragraph versus preformatted text.

We may have noticed that the preformatted text is in a monospaced font-it will always be that way. Otherwise, the <PRE> tag works pretty much like the paragraph font, except that it lets us decide where the line breaks and spaces will appear. Look at the following example:
<PRE>I simply want to make this <B>really</B> clear to you.

</PRE>
With the above code, the browser will display this line in nearly exactly the same way as it would using the <P> tag, except that it will be in a monospaced font, and the extra spaces and extra return will appear as well. In fact, there will be two blank lines below the line of text-one for the return, and one for the </PRE> tag itself.
We can even use the <PRE> tags to create extra lines in a document without typing any text between them. This example adds two blank lines to a document:
<PRE>
</PRE>
For each additional blank line you want to add, just press Enter after the first tag one time.

Note: There is one potential drawback to the <PRE> tag. It doesn't allow the browser screen to wrap text automatically-instead, users need to expand their browser window if we use particular long lines within a <PRE> container. Just keep this in mind, and make sure your lines of text are reasonably short so that all browsers can view them without scrolling.

Example: Using <PRE> for Spaces and Tables

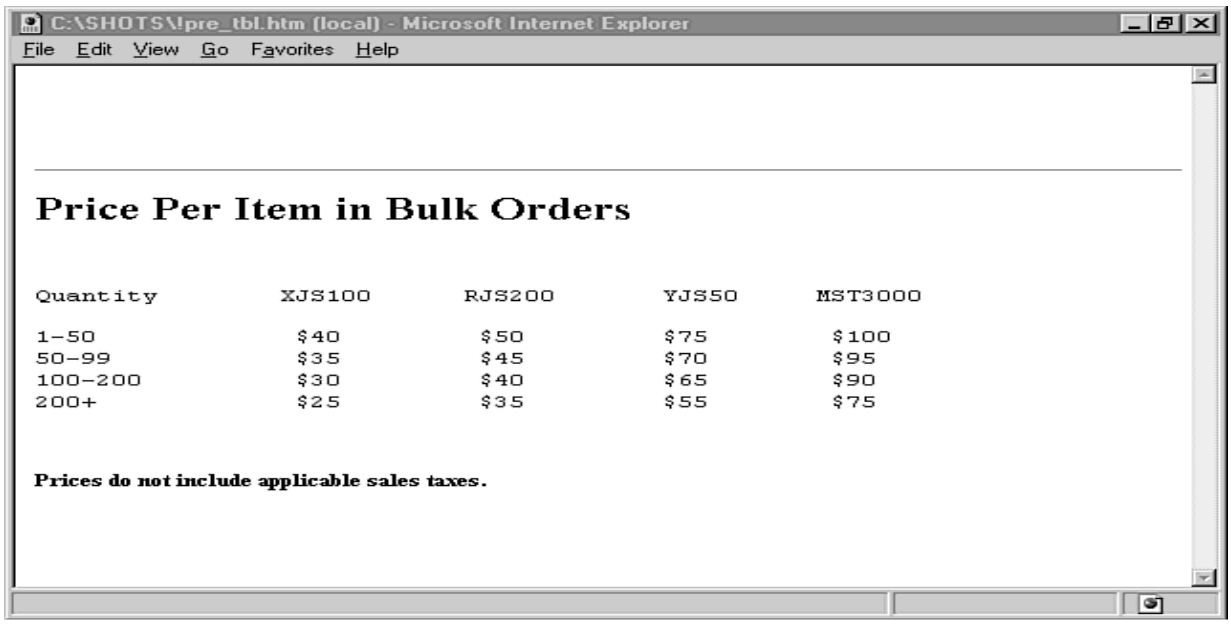
In the same way that we created the film script using the <PRE> tag, we can also format a primitive table using the <PRE> tag along with some others. The key to making this work correctly is alignment. Realize that each space taken up by a character of an invisible tag (like <B>) will not appear in the browser's display, so we'll need to compensate.

Tip : One way to keep the columns in a table straight is to type your table first, and then add emphasis tags afterward.
Load your template and save it as pre_tbl.html. Now enter listing between the body tags.

```
<BODY>
<PRE>

</PRE>
<HR>
<H2>Price Per Item in Bulk Orders</H2>
<PRE>


Quantity       XJS100      RJS200       YJS50     MST3000

1-50           $40         $50          $75       $100
50-99          $35         $45          $70       $95
100-200        $30         $40          $65       $90
200+           $25         $35          $55       $75

</PRE>
<H5>Prices do not include applicable sales taxes.</H5>
</BODY>
```

We may need to play with the spacing a bit to line everything up. Save the HTML document, and then choose the Open File command in browser to proof it. Keep playing with it until it looks right.

Once we have everything aligned correctly, it's actually a fairly attractive and orderly little table



Use the <PRE> tag to create a table.

CHAPTER 5: Displaying Text in Lists

Objectives:

- To list content of page by using List tag
- Use Directories, Definitions, and Menus
- Quoting, Citing, Definitions, and Addresses

Using Lists in HTML

List tags, like paragraphs and preformatted text, are generally HTML containers that are capable of accepting other container and empty tags within their boundaries. These list tags are responsible for affecting the spacing and layout of text, not the emphasis, so they are applied to groups of text, and allow individual formatting tags within them.
Most HTML lists are created following the form:

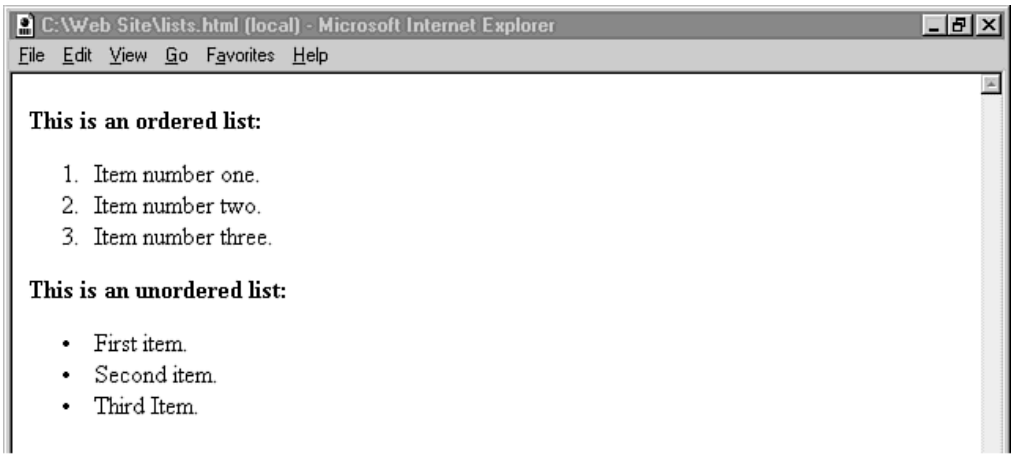
```
<LIST TYPE>
<ITEM> First item in list
<ITEM> Second item in list
<ITEM> Third item
</LIST TYPE>
```

Each of the items appears on its own line, and the <ITEM> tag itself is generally responsible for inserting either a bullet point or the appropriate number, depending on the type of list that's been defined. It's also possible that the <ITEM> tag could insert no special characters (bullets or otherwise), as is the case with definition listings.

You'll look at each type in the following sections. The basics to remember are to use the main container tags for list type and the individual empty tags to announce each new list item. The type of list you choose is basically a question of aesthetics.
Ordered and Unordered Lists

It might be better to think of these as *numbered* (ordered) and *bulleted* (unordered) lists, especially when we're talking about their use in HTML. The only drawback to that is the fact that the HTML codes for each suggest the ordered/unordered names. For numbered/ordered lists, the tag is <OL>, and for bulleted/unordered lists, the tag is <UL>.
For either of these lists, a line item is designated with the empty tag <LI>. In the case of ordered lists, the <LI> tag inserts a number; for unordered lists, it inserts a bullet point. Examples of both follow. The following is an ordered list:

```
<OL>
<LI> Item number one.
<LI> Item number two.
<LI> Item number three.
</OL>
And here's an unordered list:
<UL>
<LI> First item.
<LI> Second item.
<LI> Third Item.
</UL>
```



To see how these look in a browser, check below figure

The subtle differences between ordered and unordered lists.

As we've already mentioned, both ordered and unordered lists can take different types of internal HTML tags. It's even possible to include paragraph, line break, and header tags in lists.While you may see the potential in creating ordered lists that conform to standard outlining conventions (for instance, Roman numerals and letters), HTML 3.0 doesn't really help much. There is no way to change the <LI> number from Arabic numbers, and there's no way in HTML 3.0 to create a list that starts with something other than 1. Netscape, however, has added both of these abilities, and you can be much freer in your outline, as long as you warn your users ahead of time to view your page with Netscape Navigator (or a Netscape-compatible browser).

Directories, Definitions, and Menus

Your other lists have something in common with one another that they don't share with ordered and unordered lists: all of them use some permutation of the previous line-item system, but none of them consistently use numbers or bullets. Directories and menus are basically just plain lists. Definitions are unique among all lists because they offer two levels of line items within the list structure-one for the definition item and one for the definition itself.

Directory and Menu Lists

To create a directory or menu list, you start with its respective container tag: <DIR> or <MENU>. Of these two, the directory list is probably more useful. Most browsers don't currently render the <MENU> command consistently some use a bulleted list, others use no bullets. The following is an example of <MENU>:

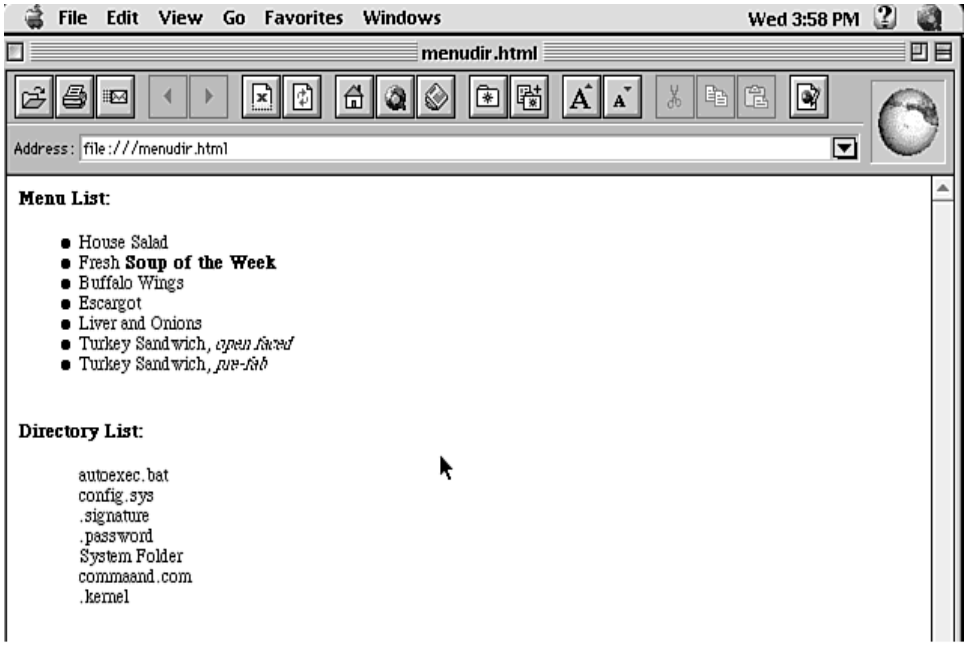
```
<MENU>
<LI>House Salad
<LI>Fresh <B>Soup of the Week</B>
<LI>Buffalo Wings
<LI>Escargot
<LI>Liver and Onions
<LI>Turkey Sandwich, <EM>open faced</EM>
<LI>Turkey Sandwich, <EM>pre-fab</EM>
</MENU>
```

Note: You might use the <MENU> tag when creating a list of hypertext links. It's thought that future interpretations of the menu list may be built into future browsers, and that designers will eventually see more benefit in using the <MENU> tag.

In theory, the <DIR> tag is a little more limiting. It's designed as a mechanism for listing computer file directories in HTML pages. Technically, it doesn't support interior HTML tags, although most browsers will display them. The <DIR> tag is also supposed to be limited to 24 characters (for some unknown reason) and show the filenames in rows and columns, like a DIR/W command in MS-DOS, but the bulk of browsers seems to ignore both of these constraints as well, as in the following example:

```
<DIR>
<LI> autoexec.bat
<LI> config.sys
<LI> .signature
<LI> .password
<LI> System Folder
<LI> commaand.com
<LI> .kernel
</DIR>
```

Most browsers (including Netscape) will use the same font and layout for menus and directories, as they will for unordered lists. In some cases, browsers will display one or the other (more often directory lists) without a bullet point, which can make them mildly useful. Some browsers can be set to a different font for directories and menus (versus ordered lists). So you may want to use these types, if only because some Web-savvy users' browsers will make an effort to display them differently (as shown in below figure).



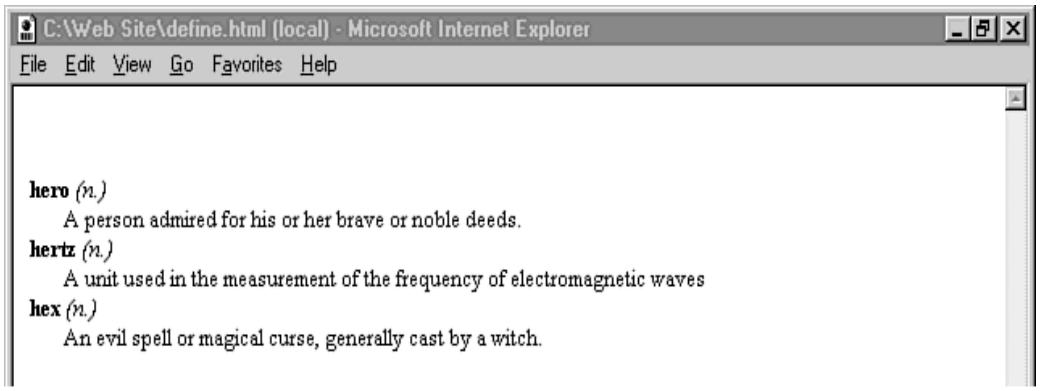
Menu and directory lists in MS Internet Explorer.

Definition Lists

The final list tag is the definition list, which is designed to allow for two levels of list items, originally conceived to be the defined term and its definition. This is useful in many different ways, though, and is also nice for its consistent lack of bullet points or numbering items (as opposed to the menu and directory listings, which are often rendered haphazardly by browsers). The tags for this list are the container tag <DL> (definition list) and two empty tags, <DT> (definition term) and <DD> (definition). The <DT> tag is designed (ideally) to fit on a single line of your Web page, although it will wrap to the beginning of the next line if necessary. The <DD> tag will accept a full paragraph of text, continuously indented beneath the <DT> term. The following is an example of all three tags:

```
<DL>
<DT><B>hero</B> <I>(n.)</I>
<DD>A person admired for his or her brave or noble deeds.
<DT><B>hertz</B> <I>(n.)</I>
<DD>A unit used in the measurement of the frequency of electromagnetic waves
<DT><B>hex</B> <I>(n.)</I>
<DD>An evil spell or magical curse, generally cast by a witch.
</DL>
```

Notice that standard HTML mark-up is permissible within the boundaries of a definition list, and that using bold and italics for the defined terms adds a certain dictionary-like quality.



A basic definition list.

Tip: Not all browsers will display definition lists in the same way, so adding spaces to <DT> items (to get them to line up with the <DD> text) is often a waste of time.

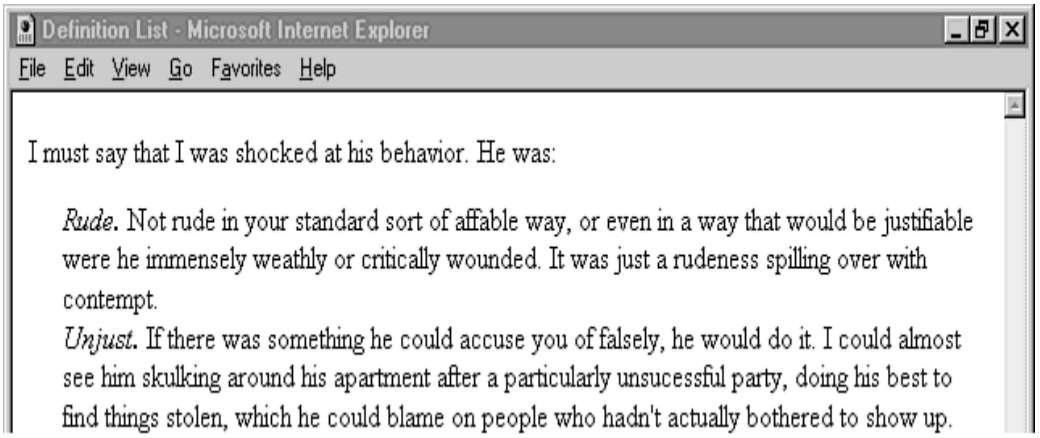
It should also be pointed out that just because definition lists allow for two different types of list items, you needn't necessarily use both. Using just the <DT> tag in your list, for instance, will result in a list not unlike an unordered list-except that nearly all browsers will display it without bullets:

```
<DL>
<DT>Milk
<DT>Honey
<DT>Eggs
<DT>Cereal
</DL>
```

And, although more difficult to find a use for, the <DD> item could be used on its own to indent paragraphs repeatedly.

```
<P>I must say that I was shocked at his behavior. He was:
<DL>
<DD><I>Rude.</I> Not rude in your standard sort of affable way, or even in a way that would be justifiable were he immensely weathly or critically wounded. It was just a rudeness spilling over with contempt.
<DD><I>Unjust.</I> If there was something he could accuse you of falsely, he would do it. I could almost see him skulking around his apartment after a particularly unsucessful party, doing his best to find things stolen, which he could blame on people who hadn't actually bothered to show up.
</DL>
</P>
```

The definition list offers some additional flexibility over the standard lists, giving you more choices in the way you layout the list items.



Definition lists using only one of the two elements.

Nesting Tags and Combining List Types

Since most of your HTML lists can accept HTML tags within their list items, it stands to reason that you could potentially create lists within lists. In fact, creating a list, then creating another list as part of an item in that first list is how you can create an outline in HTML.

Nesting Tags

The idea of nesting comes to us from computer programming. Nesting is essentially completing an entire task within the confines of another task. For HTML, that means completing an HTML tag within the confines of another container tag. This could be something like the following:
<P>She was easily the most <EM>beautiful</EM> girl in the room.</P>

This is an example of correctly nesting the <EM> tag within a paragraph container. On the other hand, many browsers would still manage to display this next code:
<P>She was easily the most <EM>beautiful</P> girl in the room.</EM>

But this second example is really poorly constructed HTML. It often works, but the <EM> tag isn't properly nested inside the <P>. In this example, that doesn't matter too much, since you can still reason out what this statement is trying to do.
With lists, however, things can get complicated. So it's best to remember the "nesting" concept when you begin to add lists within lists. As far as HTML is concerned, a nested list works as marked-up text within the previous list item. When the next list item is called for, HTML moves on.

Lists within Lists

Let's look at an example of a simple nested list:

```
<OL>
<LI>Introduction
<LI>Chapter One
    <OL>
    <LI> Section 1.1
    <LI> Section 1.2
    <LI> Section 1.3
    </OL>
<LI>Chapter Two
</OL>
```

Tip: It's a good idea to indent nested lists as shown in the example. The browser doesn't care-it just easier for you (or other designers) to read in a text editor. (Regardless of your spacing, most browsers will indent the nested lists-after all, that's the point.)

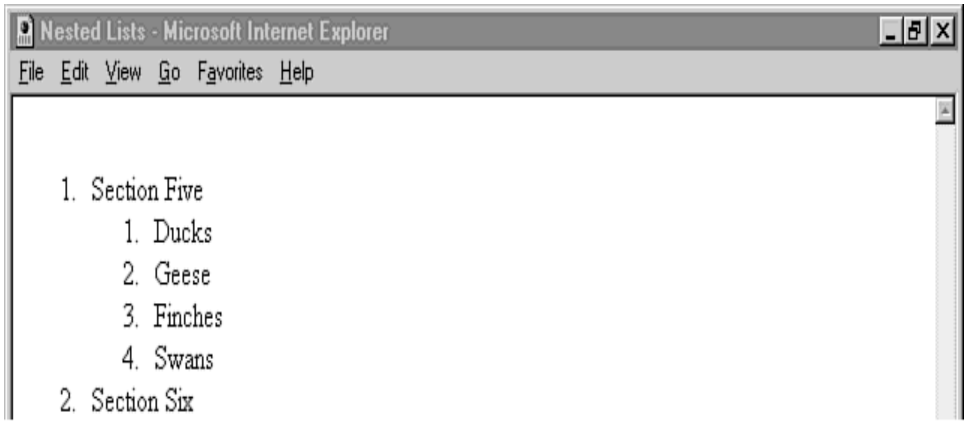
Notice that the nested list acts as a sublevel of the Chapter One list item. In this way, we can simulate an outline in HTML. Actually, the nested list is just HTML code that is part of the <LI>Chapter One list item. As we saw in below listing you can use the
 tag to create a line break in a list element without moving on to the next list item. Following the same theory, an entire nested list works as if it's a single list item in the original list.
The following:

```
<OL>
<LI>Section Five<BR>
 This section discusses ducks, geese, finches and swans.
<LI>Section Six
</OL>
```

is essentially the same as the list that follows:

```
<OL>
<LI>Section Five
    <OL>
    <LI> Ducks
    <LI> Geese
    <LI> Finches
    <LI> Swans
    </OL>
<LI> Section Six
</OL>
```

In both cases, the nest HTML container is simply a continuation of the first list item. Both the text after the
 in the first example and the ordered list in the second example are part of the list item labeled Section Five



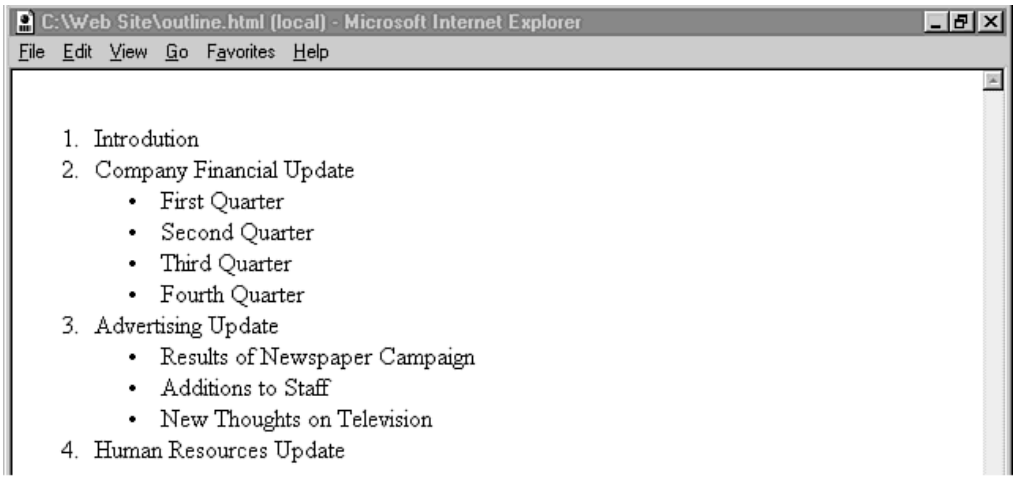
In both of the examples, the HTML container is simply part of the list.

Combining List Types

When nesting lists, it's also possible to nest different types of lists within one another. This is useful when you'd like to vary the types of bullets or numbers used in an outline form. For instance:

```
<OL>
<LI>Introduction
<LI>Company Financial Update
   <UL>
   <LI>First Quarter
   <LI>Second Quarter
   <LI>Third Quarter
   <LI>Fourth Quarter
   </UL>
<LI>Advertising Update
   <UL>
   <LI>Results of Newspaper Campaign
   <LI>Additions to Staff
   <LI>New Thoughts on Television
   </UL>
<LI>Human Resources Update
</OL>
```

There's nothing terribly difficult to learn here-just the added benefit of being able to nest different types of lists within others. You're still simply adding HTML markup code to items in the original list. This time, however, you have more choice over how your outline looks (see fig. 6.6).



Nesting different types of lists.

CHAPTER 6: Adding Graphics to Your Web Pages

Obejctives:

- Use Images in web page
- Types of graphics that can be added to web page
- Creating and Manipulating Graphics

Now that we've seen many ways we can add some character to our text-and use different tags to better communicate your ideas-it's time to jazz up your pages a little bit. Let's add some graphics! First, though, we should know a couple of important things about placing graphics. Some of these considerations may seem a bit foreign to you, especially if we're a graphic designer or commercial artist. We have to think in a slightly different way about graphics for your HTML pages.

The Special Nature of Graphics on the Web

One may be comfortable using a program such as CorelDraw! Or Adobe Photoshop to create and manipulate graphics, But if we've never done any design for the World Wide Web, there's also a good chance that we've never worried about one special graphics issue, even if we are a print design expert. How big is the graphics file that we created? Aside from using the correct graphics format, this issue is the single most important consideration in graphical Web design.

The Size of Graphics Files

Why is the size of graphics files so important? Our Web users have to download our pages to view them, including all the graphics associated with the pages. Couple that fact with the Web speed issues discussed in Chapter 5, and the need for smaller graphics files becomes apparent.
The high-color, high-resolution graphics files that color printers and professional designers work with are generally measured in the number of megabytes of information required to create the graphics file. Each image can take up more space than is available on a floppy disk. Often, special tapes and cartridges are required to transfer these files from the graphics shop to the printer.
A good average size for a Web graphic, on the other hand, is between 10K and 30K-about one to three percent of the size of those high-color, high-resolution graphics. This could be tough.

Picking Your Web Graphics File Type

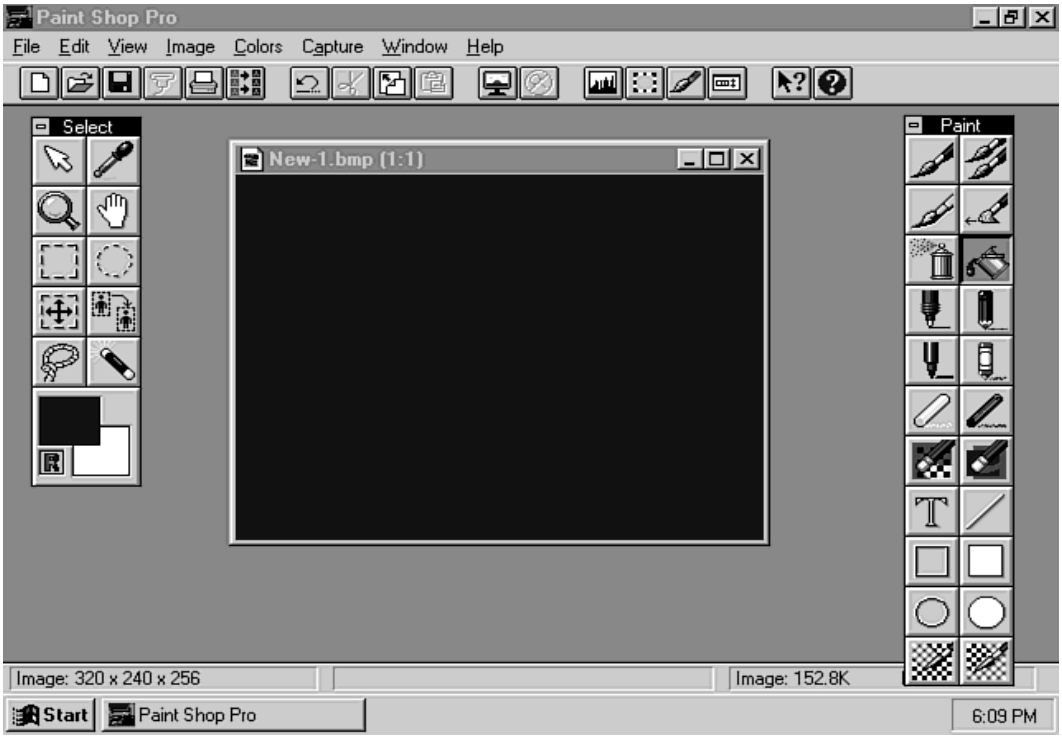
The other thing that you need to concern yourself with is the file type that we're going to use for Web graphics. In general (at least currently), we can choose either of two file types: GIF and JPEG. *GIF* (CompuServe Graphics Interchange Format) is the more popular among Web browsers, but *JPEG* (Joint Photographic Experts Group) is gaining popularity and becoming more widely used. GIF and JPEG bring different advantages to the table.

GIF Format Graphics

Any graphical browser supports the display of GIF format files *inline*, meaning that the browser doesn't require a special viewer for these files. GIFs are compressed graphics, but they tend to lose less image clarity than JPEGs. Images that have smaller color palettes (those that use 256 colors or fewer) often look better in GIF format. GIF is also the file format of choice for creating transparent graphics-graphics that make the Web page appear to be the actual background of the GIF graphic.
Although GIF files are compressed, they tend to be a bit larger than JPEGs, but they decompress more quickly and tend to be drawn more quickly than JPEGs (given the same file size). Another problem with the GIF file format is the fact that it includes certain copyrighted elements that make it less than an open standard for graphics interchange.

The JPEG Format

Gaining on GIF in popularity is the JPEG format, which is widely used by Web designers. JPEG graphics can be viewed in most new graphical browsers without a special helper application. JPEG graphics have the advantage of being better for graphics that have more colors (up to 16.7 million, in most cases) than similar GIF files; in addition, the JPEG files are smaller (look ahead to fig. below). Also, the compression scheme is in the public domain.



The paint Shop Pro interface.

On the down side, JPEGs can be a little glossier than GIFs, meaning that the higher rate of compression results in slightly lower image quality. JPEGs also take a little longer to decompress than do GIF files. So although the smaller size of JPEG files allows them to be transmitted over the Internet more quickly, the amount of time that it takes to decompress those files sometimes negates this advantage.

Creating and Manipulating Graphics

It's no secret that a lot of Web design has transitioned from manipulating text-based HTML documents to designing and integrating compelling graphics into Web pages. As the Web has become more commercial, its graphical content has become more professional. If we're not up to the task of creating professional graphics, don't worry too much; programs are available that will help you. Also, it's more important that graphics further the usefulness of the text. The graphics in and of them are not the point. The point is to make our Web pages more exciting and informative. It is a fact, however, that Web sites are leaping forward daily into a more professional, more graphical presentation of Web-based information. Commercial artists and designers are continuing to find new niches on the Web. If we're a skilled computer artist, congratulations; this is where we'll put your skills to use. If we're not, that's OK, too. Any Web designer needs to be

able to manipulate and edit graphics in a program such as Adobe Photoshop or CorelDraw! But we don't necessarily have to *create* those graphics, if that's not your forte.

Creating Graphics for the Web

As we get started with a program such as Photoshop or CorelDraw! Keep in mind that the most important consideration in creating Web graphics is the file size. File size isn't generally the first consideration for creating print graphics; almost any print shop or prepress house will accept large storage cartridges or tapes that provide access to your huge full-color graphics. Not so on the Web. Our target is as small as possible-between 15K and 35K for larger (bigger on the screen) files.
We can come up with graphics to use on your Web pages in many ways. Eventually, any graphic that you use needs to be in a standard file format (for example, GIF or JPEG) and relatively small. But how you come up with the final graphic has a lot to do with the information that you're trying to communicate and with your skills as an artist. The following are some of the different ways you might come up with Web graphics:

> **Create graphics in a graphics application**. Many programs for both professional and amateur artists can output GIF- or JPEG-format files for use on the Web. Among these programs are Adobe Photoshop, CorelDraw! Fractal Painter, and Fractal Dabbler.

Tip: Any graphics program, even Microsoft Paint, can create Web graphics, although you may need to use another program to change the graphic to an acceptable file format.

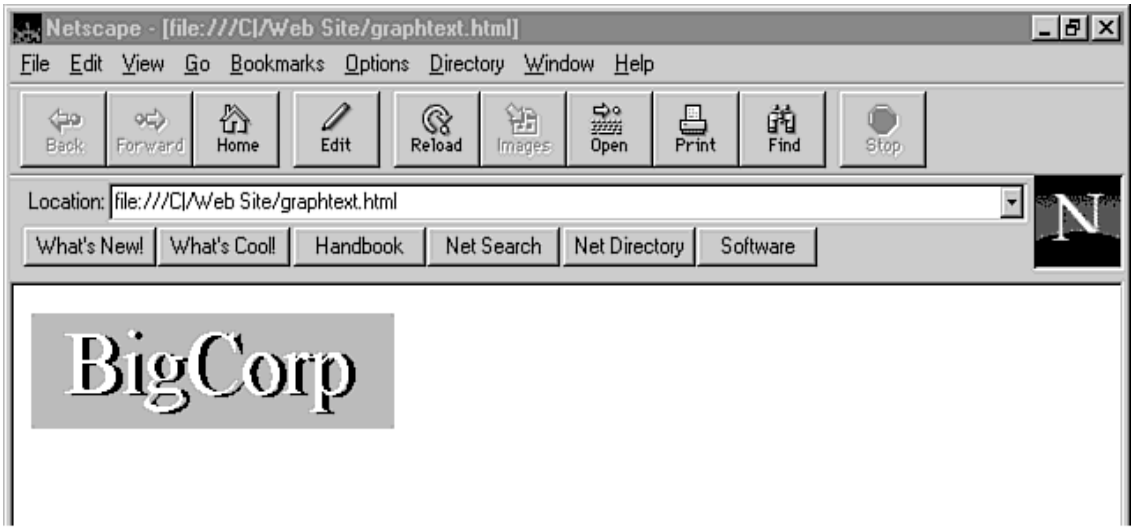
- **Download public-domain graphics**. Tons of sites on the Internet allow us to download icons, interface elements, and other graphics for our Web site. At the same time, public-domain clipart collections (such as those available on CD-ROM) can be useful for Web pages.
- **Use scanned photographs**. Using scanned photographs (especially those that we've taken yourself) is a great way to come up with graphics for your Web pages. Unless you have access to scanning hardware, though, you may need to pay someone to scan the photos.
- **Digital cameras**. Cameras are available that allow us to take photos that can be downloaded directly from the camera to your computer. While some of this equipment can be very expensive, cameras under $500 do exist, and those photos can easily be converted for use on the Web.
- **Use PhotoCDs**. Many photo development shops can create digital files of your photographs (from standard 35mm film or negatives) and save those files in PhotoCD format. Most CD-ROM drives allow you to access these photos, which you can then change to GIF or JPEG format and display on your Web pages.

Embedding Graphics in Web Pages

To add graphics, we use an empty tag called the <IMG> (image) tag, which we insert into the body section of your HTML document as follows:
<IMG SRC="*image URL*">
Or
<IMG SRC="*path/filename*">
SRC accepts the name of the file that we want to display, and *image URL (or path/filename)* is the absolute (full URL) or relative path (for a local file or a file in the current directory) to the image. As the first example shows, we can display on our page any graphic file that is generally available on the Internet, even if the file resides on a remote server. For graphics files, however, it is much more likely that the file is located on the local server, so a path and filename are sufficient.
We could enter the following text in a browser:

```
<HR>
<P>This is a test of the Image tag. Here is the image I want to
display:</P>
<IMG SRC="image1.gif">
<HR>
```

In this case, <IMG SRC="image1.gif"> is a relative path URL, suggesting that the file image1.gif is located in the same directory as the HTML document. A browser as shown in figure below would display the result:



Displaying inline graphics on a Web page.

An absolute URL is essential, however, if we were accessing an image on a remote site, as in the following example:

```
<IMG SRC="http://www.graphcom.com/pub/graphics/image1.gif">
```

(This example is fictitious.) Please realize that using a URL to a distant site on the Internet causes that site to be accessed every time this <IMG> tag is encountered on your page, so you should probably have some sort of arrangement with that Web site's system administrator before you link to a graphic on their server.

Adding Graphics to Other HTML Tags

We can add graphics links to HTML tags to do various things, including placing graphics next to text (within paragraphs) and even including graphics in lists. The following example displays the graphic flush with the left margin, with the bottom of the text that follows the image aligned with its bottom edge:

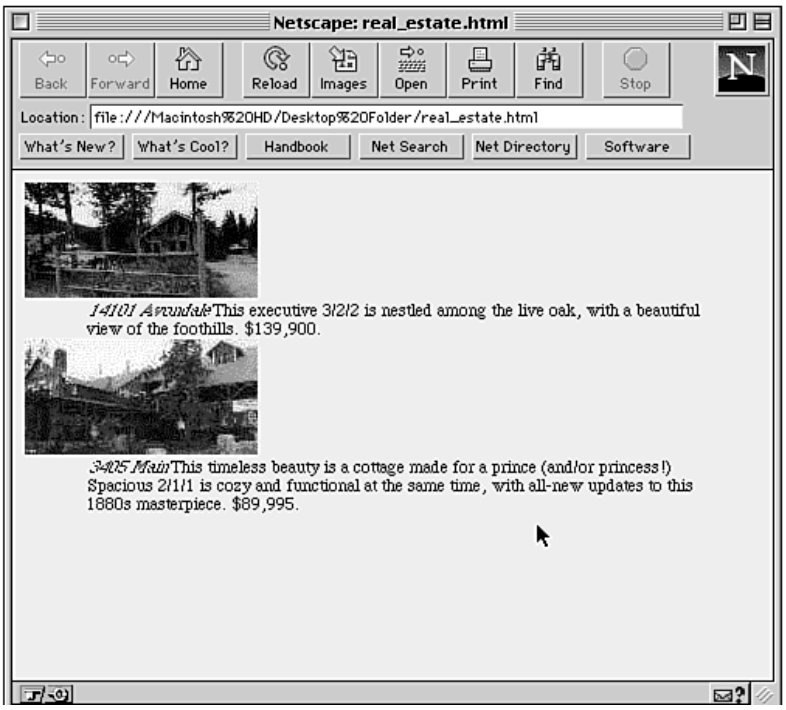
```
<P><IMG SRC="start.gif"> It's time to start our adventure in the world of
the Web. As you'll see below, there is much to learn. </P>
Words at the end of the first line wrap below the image
```

Graphics within paragraph containers.

Another popular use for graphics is including them in HTML lists. Best suited for this task is the <DL> (definition) list, which allows you to use your own graphics as bullet points. (Ordered and unordered lists display their numbers or bullets in addition to the graphic.)
A <DT> (definition term) tag can accept more than one <DD> (definition) element, so you can create a bulleted list as follows:

```
<DL>
<DT>
<DD><IMG SRC="bullet.gif"> This is the first point
<DD><IMG SRC="bullet.gif"> This is the second point
<DD><IMG SRC="bullet.gif"> Here's the third point
<DD><IMG SRC="bullet.gif"> And so on.
</DL>
```

At the same time, you could use a definition list in conjunction with thumbnail graphics in a list that uses both the <DT> and <DD> tags. An example might be the following real estate agent's pages



Use a <DL> tag to create custom bulleted lists and thumbnail lists.

```
<DL>
<DT><IMG SRC="Small_House14101.GIF">
<DD><EM>14101 Avondale</EM> This executive 3/2/2 is nestled among the live oak, with a beautiful view of the foothills. $139,900.
<DT><IMG SRC="Small_House3405.GIF">
<DD><EM>3405 Main</EM> This timeless beauty is a cottage made for a prince (and/or princess!) Spacious 2/1/1 is cozy and functional at the same time, with all-new updates to this 1880s masterpiece. $89,995.
</DL>
```

The ALT Attribute

The ALT attribute for the <IMG> tag is designed to accept text that describes the graphic, in case a particular browser can't display the graphic. Consider the plight of users who use Lynx or a similar text-based program to surf the Web (or users of graphical browsers that choose not to auto-load graphics). Because those users can't see the graphic, they'll want to know what they're missing.
The ALT attribute works this way:
<IMG SRC="image URL" ALT="Text description of graphic">
The following is an example:
<IMG SRC="image1.gif" ALT="Logo graphic">
For people whose browsers can't display the graphic, the ALT attribute tells them that the graphic exists and explains what the graphic is about.

Tip: Test your site with the Load Images option turned off so that you can see how your ALT text displays.

The ALIGN Attribute

<IMG> can accept another attribute that specifies how graphics appear relative to other elements (like text or other graphics).

Using the ALIGN attribute, you can align other elements to the top, middle, or bottom of the graphic. It follows this format:

<IMG SRC="image URL" ALIGN="direction">

The ALIGN attribute is designed to align text that comes after a graphic with a certain part of the graphic itself. An image with the ALIGN attribute set to TOP, for example, has any subsequent text aligned with the top of the image, like in the following example:

<IMG SRC="image1.gif" ALIGN=TOP> Descriptive text aligned to top.

Giving the <IMG> tag an ALIGN="MIDDLE" attribute forces subsequent text to begin in the middle of the graphic.

The ALIGN attribute for the <IMG> tag.

<IMG SRC="image1.gif" ALIGN="MIDDLE"> Descriptive text aligned to middle.

Order among the attributes that you assign to an image tag is unimportant. In fact, because SRC="URL" is technically an attribute (although a required one), we can place the ALIGN or ALT attribute before the SRC information. Anywhere you put attributes, as long as they appear between the brackets of the <IMG> tag, is acceptable.

Example: Adding Graphics to Your Web Site

Create a logo, a special bullet, and a photo for use on the page. Name GIFs LOGO.GIF >, BULLET.GIF, and PHOTO.GIF, or something similar. (.)

Then load HTML template, and save it as a new HTML document. Between the body tags, type something like

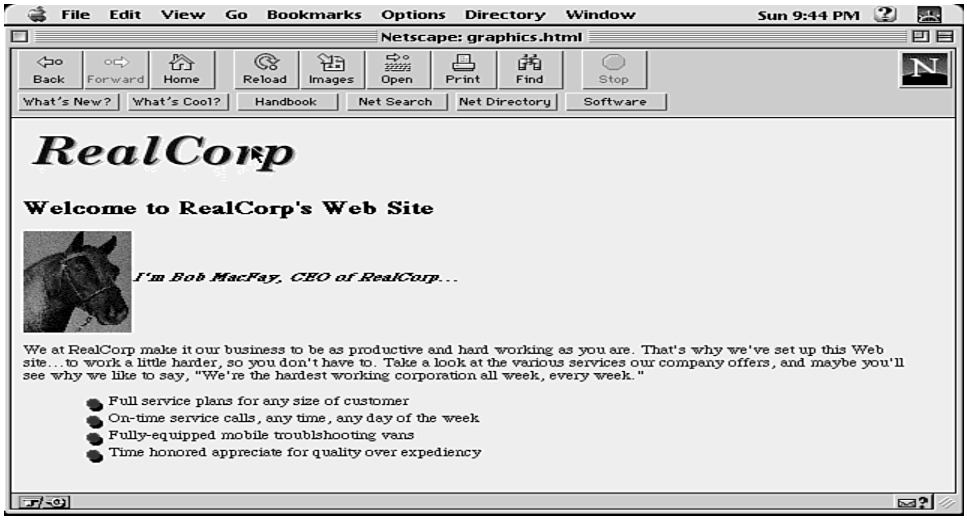
---

images.html Using <IMG> to Create Images

```
<BODY>
<IMG SRC="logo.gif" ALT="RealCorp Logo">
<H1>Welcome to RealCorp's Web Site</H1>
<H2><IMG SRC="photo.gif" ALT="Photo of CEO Bob MacFay" ALIGN=MIDDLE><EM>I'm
Bob MacFay, CEO of RealCorp...</EM></H2>
<P>We at RealCorp make it our business to be as productive and hard working as you are.
That's why we've set up this Web site...to work a little harder, so you don't have to. Take a
look at the various services our company offers, and maybe you'll see why we like to say,
"We're the hardest working corporation all week, every week."</P>
<DL>
<DT>
<DD><IMG SRC="bullet.gif" ALT="-" ALIGN=MIDDLE> Full service plans for any size of
customers
<DD><IMG SRC="bullet.gif" ALT="-" ALIGN=MIDDLE> On-time service calls, any time, any
day of the week
<DD><IMG SRC="bullet.gif" ALT="-" ALIGN=MIDDLE> Fully-equipped mobile
troublshooting vans
<DD><IMG SRC="bullet.gif" ALT="-" ALIGN=MIDDLE> Time honored appreciate for quality
over expediency
</DL>
</BODY>
```

Although the ALT attribute is optional and the bulleted list may survive without it, the example uses ASCII to substitute hyphens for the bullet graphics if the browser can't display images. In most cases, we'll want to describe an image that a user can't view. For an element such as a bullet, though, we can use the ALT attribute to substitute an ASCII character for the graphic.
For the photo of the CEO, the <IMG> tag is called within the <H2> tag, because the <H2> container (like a paragraph) otherwise would insert a carriage return and force the words I'm Bob MacFay... to appear below the photo. Including the <IMG> tag inside the <H2> tag allows the text

to appear next to the photo



CHAPTER 7: Hypertext and Creating Links

Obejctives:

- Discuss Hypertext
- List anchor a tag and it's attributes
- Demonstrate absolute and relative URL

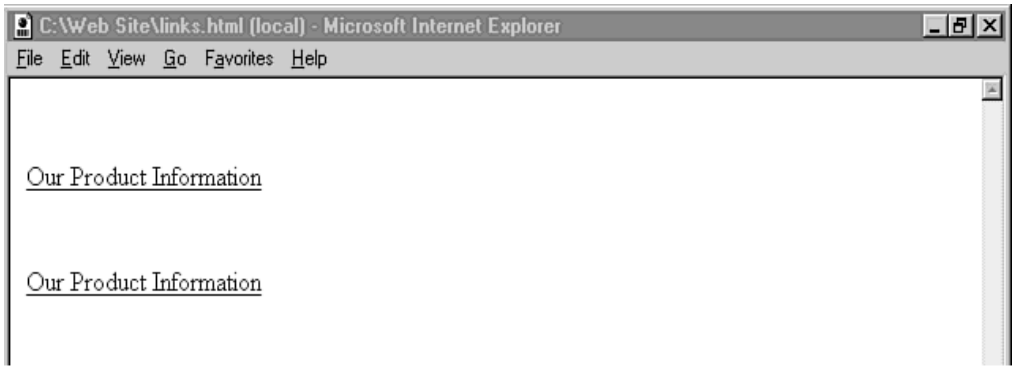
We've seen in detail the ways we can mark up text for emphasis and add images to our Web pages, it's time to take the leap into making these pages useful on the World Wide Web by adding hypertext links. The anchor tag for hypertext links is simple to add to our already-formatted pages. We'll see how URLs are useful for creating hypermedia links and links to other Internet services.

Using the <A> Tag

The basic link for creating hypertext and hypermedia links is the <A>, or anchor, tag. This tag is a container, which requires an </A> to signal the end of the text, images, and HTML tags that are to be considered to be part of the hypertext link.
Here's the basic format for a text link:
<A HREF=*"URL">Text describing link*</A>

Be aware that HREF, although it's something that you'll use with nearly every anchor tag you create, is simply an attribute for the <A> tag. Displayed in a browser, the words *Text describing link* would appear underlined and in another color (on a color monitor) to indicate that clicking that text initiates the hypertext link.

The following is an example of a relative link:
<A HREF="products.html">Our Product Information</A>
If the HTML document to which you want a link is located elsewhere on the Internet, you simply need a more complete, absolute URL, such as the following:
<A HREF="http://www.bignet.net/realcorp/products.html">Our Product Information</A>
In either case, things end up looking the same in a browser



These are the hypertext links that you've created.

Section Links

Aside from creating hypertext links to documents on our local computer or elsewhere on the Internet, we can create links to other parts of the same document in which the link appears. These "section" links are useful for moving people to a new section that appears on the same Web page without forcing them to scroll down the entire page.
Doing this, though, requires two instances of the anchor tag-one that serves as the hypertext link and another that acts as a reference point for that link, following this format:

```
<A HREF="#section_name">Link to another section of this document</A>
<A NAME="section_name">Beginning of new section</A>
```

Notice that the anchor tag that creates the hyperlink is similar to the anchor tags that you have used previously. The only difference is the pound sign (#) used at the beginning of the HREF text. This sign tells the anchor that it is looking for a section within the current document, as opposed to within an external HTML document.
The NAME attribute is used to create the actual section within the current HTML document. The text that the NAME attribute contains is relatively unimportant, and it won't be highlighted or underlined in any way when displayed by a browser. NAME is nothing more than an internal reference; without it, though, the link won't work.

Note: Remember to use the pound sign (#) only for the actual hypertext link, not the NAME anchor. Also, realize that the NAME text is case-sensitive and that the associated HREF text should use the same case for all letters, as does the NAME. If the HREF calls for Section_ONE, and the NAME is actually Section_One, the link will not work.
Example: A More Effective Definition List
Load the HTML template into your text editor, and choose the Save As command in your text editor to create a new file. In the body of your HTML document, type Listing 8.1 or something similar.

---

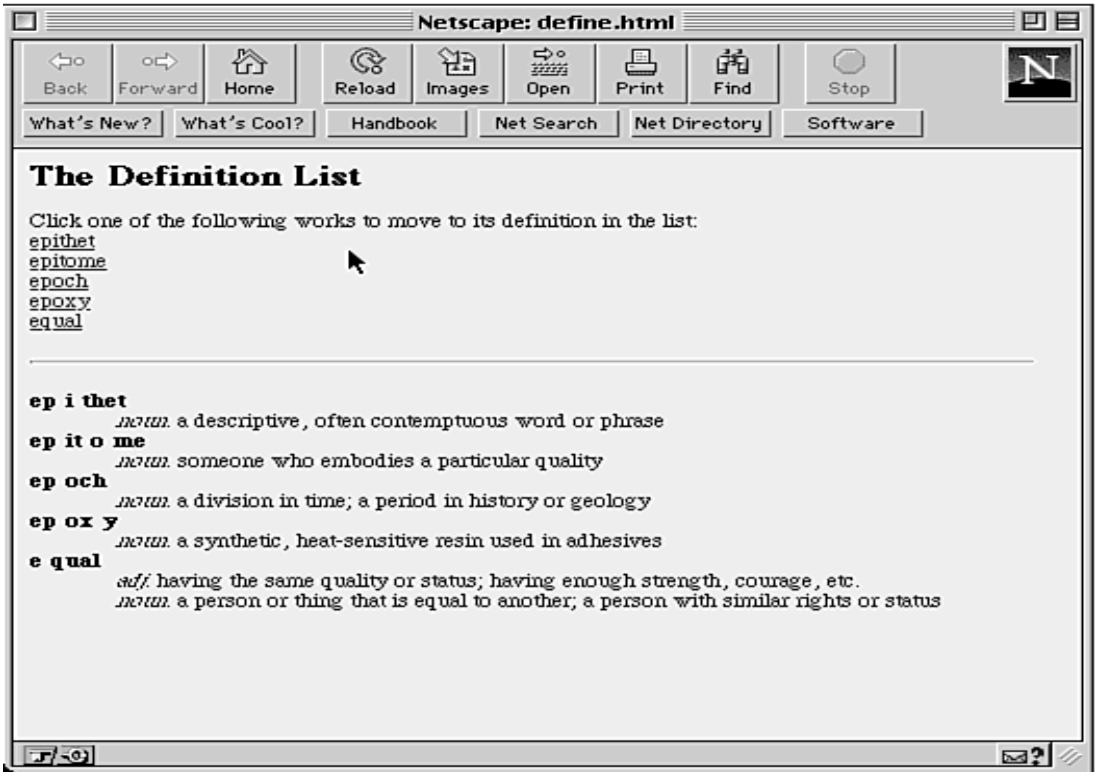
listlink.html creating a Definition List

```
<BODY>
<H2>The Definition List</H2>
<P>Click one of the following words to move to its definition in the list:
<BR>
<A HREF="#EPITHET">epithet</A><BR>
<A HREF="#EPITOME">epitome</A><BR>
<A HREF="#EPOCH">epoch</A><BR>
<A HREF="#EPOXY">epoxy</A><BR>
<A HREF="#EQUAL">equal</A><BR>
</P>
<HR>
<DL>
<DT><A NAME="EPITHET"><B>ep i thet</B></A>
<DD><EM>noun.</EM> a descriptive, often contemptuous word or phrase
<DT><A NAME="EPITOME"><B>ep it o me</B></A>
<DD><EM>noun.</EM> someone who embodies a particular quality
<DT><A NAME="EPOCH"><B>ep och</B></A>
<DD><EM>noun.</EM> a division in time; a period in history or geology
<DT><A NAME="EPOXY"><B>ep ox y</B></A>
<DD><EM>noun.</EM> a synthetic, heat-sensitive resin used in adhesives
<DT><A NAME="EQUAL"><B>e qual</B></A>
<DD><EM>adj.</EM> having the same quality or status; having enough strength, courage, and
so on.
<DD><EM>noun.</EM> a person or thing that is equal to another; a person
with similar rights or status
</DL>
</BODY>
```

In the example, clicking one of the words that appears as a hyperlink in the first section of the paragraph moves the browser window down to that link's associated NAME anchor, so that the definition becomes the focal point of the user's attention. Obviously, using section links would be of greater use in a larger list. Consider the implications for turning an entire dictionary into HTML documents.

Also notice that anchors can be placed within the confines other HTML tags, as in the first paragraph container and in the definition lists of the example. In general, other HTML tags can act anchor tags on as though they were regular text. In the case of hyper linked text, the underlining and change in color in graphical browsers take precedence, but the hyper linked text also has any other qualities of the surrounding text (for example, indenting with the rest of the definition text).

Notice which anchors cause the text to become a hyperlink and how the anchor tags respond within other container tags.



Anchor tags are used to define and move between sections of an HTML document.

Using Relative URLs

In most cases, the URL referenced by the HREF attribute within the anchor tag needs to be an absolute URL, unless it references a file located in the same directory as the current HTML document.
But consider the case of a well-organized Web site, as set out in Chapter 5, "What we need for a Web Site." That chapter discussed the fact that it's not always the best idea to drop all your Web site's files into the same directory, especially for large sites that contain many graphics or pages. How do we create links to files that may be on the same server but not in the same directory?
One obvious way is to use an absolute URL for every link in your Web site. If the current page is **http://www.fakecorp.com/index.html**, and we want to access a specific page that we organized into your products directory, we could simply create a link like the following, using an absolute URL:
<A HREF="http://www.fakecorp.com/products/new_prods.html>

Our new

products</A>

These absolute URLs can get rather tedious, not to mention the fact that if you happen to change the name of your Web server or move your site to another basic URL, we'll probably have to edit every page in your site to reflect the new URLs.

Adding the <BASE> Tag

The <BASE> tag is used to establish the absolute base for relative URLs used in your document's hypertext links. This tag is especially useful when your Web pages may appear in

different subdirectories of a single main directory, as in some of the organizational types discussed before. The format of the <BASE> tag is as follows:
<BASE HREF=*"absolute URL"*>
Note that the <BASE> tag is designed to appear only between the <HEAD> tags.
It may be helpful to think of <BASE> as doing something similar in function to a DOS path statement. The <BASE> tag tells the browser that relative URLs within this particular Web document are based on the URL defined in the <BASE> tag. The browser then assumes that relative URLs derive from the URL given in the <BASE> tag and not necessarily from the current directory of the HTML document.
Consider a document named **http://www.fakecorp.com/products/list.html** that looks something like this:

```
<HEAD>
<TITLE>Page One</TITLE>
</HEAD>
<BODY>
<A HREF="index.html">Back to Index</A>
</BODY>
```

In this example, the browser tries to find a document named index.html in the directory products, because the browser assumes that all relative addresses are derived from the current directory. Using the <BASE> tag, however, changes this example a bit, as follows:

```
<HEAD>
<BASE HREF="http://www.fakecorp.com/">
<TITLE>Page One</TITLE>
</HEAD>
<BODY>
<A HREF="index.html">Back to Index</A>
</BODY>
```

Now the browser looks for the file index.html in the main directory of this server, regardless of where the current document is stored (such as in the products directory). The browser interprets the relative URL in the anchor tag as though the complete URL were **http://www.fakecorp.com/index.html**.

Tip: If you plan to create a large Web site, you may want to add the <BASE> tag (complete with the base URL) to your HTML template file.
Using the <BASE> tag to point to your Web site's main directory allows you to create the different types of organization systems described in Chapter 5 by using relative URL statements to access HTML documents in different subdirectories.

Creating Links to Other Internet Services

Here's where the real power of URLs comes into play. Remember that an URL can be used to describe almost any document or function that's available on the Internet? If something can be described in an URL, a hyperlink can be created for it. In the following section, we start with e-mail.

Hyperlinks for E-Mail Messages

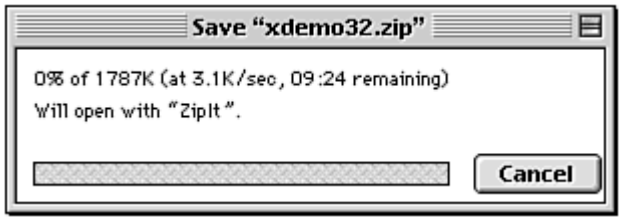
Creating a hyper linked e-mail address is simple. Using the mailto: type of URL, we can create the following link:
<A HREF="mailto:tstauffer@aol.com">Send me e-mail</A>
In many graphical browsers, this URL often loads an e-mail window, which allows you to enter the subject and body of an e-mail message and then send it via your Internet account. Even many of the major online services support this hyperlink with their built-in e-mail systems.

Clicking a mailto: link brings up an e-mail message window in Netscape**.**

Not all Web browsers accept the mailto: style of URL, however, and most of those don't return an error message. If we use this type of link, you may want to warn users. Something like the following text should work well for users of no graphical browsers:
<P>If your browser supports the mailto: command, click <A HREF="mailto:tstauffer@aol.com">here</A> to send me an e-mail message.
</P>

Other Internet Services

Links can be created for all types of Internet services. E.g. Gopher sites, for example, a hypertext link might look like the following example:
<A HREF="gopher://marvel.loc.gov/">the Library of Congress Gopher</A>
Most Web browsers can display Gopher menus. In most cases, clicking a gopher link points the browser at the Gopher site, and the Gopher menu appears in the browser window.
You can create links that cause the Web browser to download a file from an FTP server, as follows:
<P>You can also <A HREF="ftp://ftp.fakecorp.com/pub/newsoft.zip">download</A>the latest version of our software.
When the connection to the FTP server has been negotiated, the file begins to download to the user's computer. Depending on the Web browser, this file may not be formatted correctly. Each browser needs to be set up to accept files of a certain type (such as the PKZip format file in the preceding example).



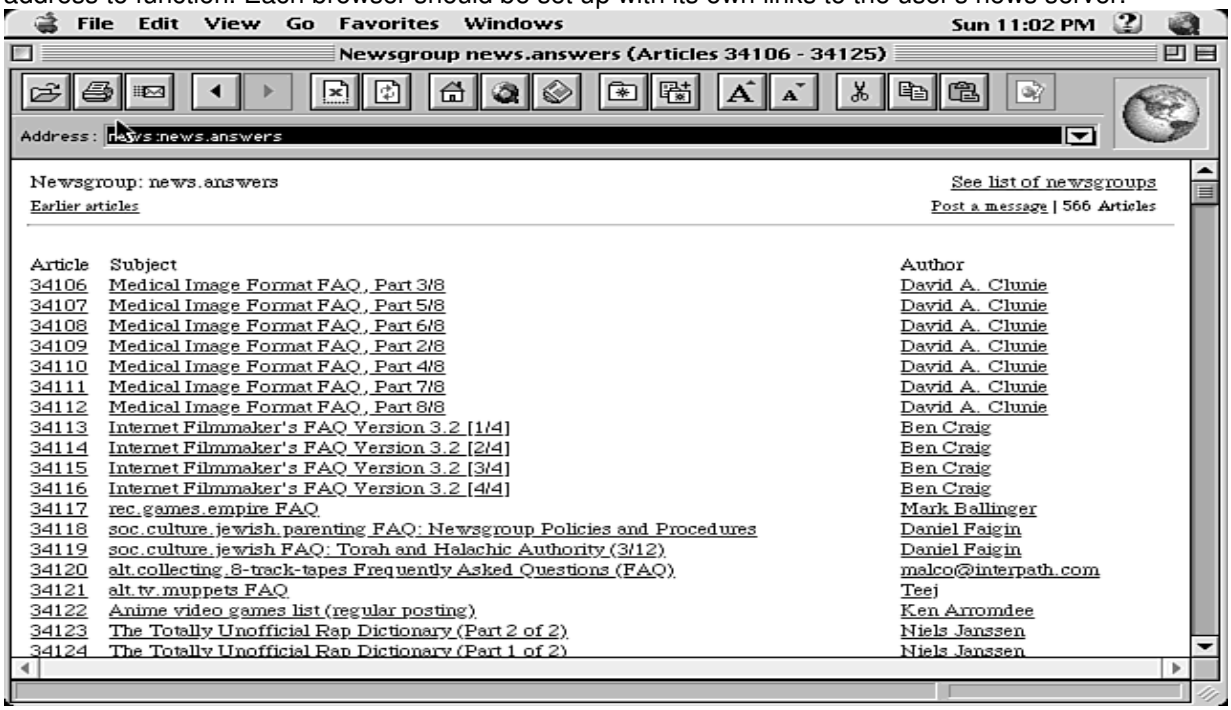
Netscape is downloading a file from an FTP server.

Note: Most browsers can accept hyperlinks only to anonymous FTP servers. We generally should not include in our HTML documents links to FTP servers that require usernames and passwords.

Again, most browsers have some mechanism (sometimes built into the browser window) for reading UseNet newsgroups. Some browsers launch a separate program to read UseNet groups. In either case, you can create a link like the following:

<A HREF="news:news.answers">UseNet Help Newsgroup</A>

This link loads whatever UseNet reading features the browser employs and displays the specified newsgroup (see fig. 8.7). The news: URL type does not require a particular Internet server address to function. Each browser should be set up with its own links to the user's news server.



MS Internet Explorer after clicking a link to the newsgroup news. answers.

Other Links for the <HEAD> Tag

We can create a couple more tags in the <HEAD> section of your HTML documents. The two tags discussed in the following sections are <LINK> and <ISINDEX>.

The <LINK> Tag

The <LINK> tag is designed to establish a hypertext relationship between the current document and another URL. Most of the time, the <LINK> tag does not create a click able hypertext link in

the user's Web viewer window. It's a little beyond the scope of this book, but programs can be written to take advantage of the <LINK> tag, such as a program that creates a toolbar that makes use of the relationship defined.

The <LINK> tag generally has either of the following formats:
<LINK HREF="*URL*" REL="*relationship*">
Or
<LINK HREF="*URL*" REV="*relationship*">

For the most part, <LINK> is used to create an author-defined structure to other HTML documents on a Web site. The attribute REL, for example, defines the relationship of the HREF URL to the current document. Conversely, REV defines the relationship between the current document and the HREF'ed URL.

Following are two examples of <LINK> statements:

```
<LINK HREF="http://www.fakecorp.com/index.html" REL="PARENT">
<LINK HREF="http://www.fakecorp.com/product2.html" REV="CHILD">
```

In the HTML 3.0 standard, these definitions are relatively irrelevant-at least publicly on the Web. We more commonly find these statements used within certain organizations (perhaps companies employing an intranet), especially for advanced Web-based documentation efforts and for efforts that use HTML and SGML together.

HTML 4.0 more than likely will introduce more widespread use of the <LINK> statement and other <HEAD> tags for more tangible benefits.
We may want to use one <LINK> frequently: the REV="MADE" link, which tells users who created the HTML page. Although this use of <LINK> doesn't actually call up a mailto: link in most browsers, some may recognize it eventually. In the meantime, it gives people who view our source code the e-mail address of the author, as in the following example:
<LINK HREF="mailto:tstauffer@aol.com" REV="MADE" REL="AUTHOR">

CHAPTER 8: Clickable Image Maps and Graphical Interfaces

Obejctives:

- Adding Image in web
- Use Imagemap for better interactivity

This chapter takes creating a graphical interface to our Web site one step further. With image maps, we can create an entire interface for our Web pages and sites that rival the interfaces of popular multimedia games, graphical operating environments, and interactive kiosks.

Image Maps Defined

The *map* part of *image map* conjures up two separate images. First, image maps on Web sites often act like road maps for the Web site, adding interface elements that make it easier to get around on the Web site. Second, the word *map* also suggests the way that image maps are created. Image maps begin life as normal graphics (usually in GIF or JPEG format), designed with the Web in mind. Then another program is used to map *hot zones* (clickable areas) on top of the graphics.
When put in place on a Web page, an image map allows users to access different HTML pages by clicking different parts of the graphic. Because each hot zone has an associated URL, and because each hot zone corresponds to part of the graphic, maneuvering about a Web site becomes more interesting, graphical, and interactive.

Example: The Apple Web Site

Apple Computer offers a very interesting example of an image map on the main page of its Web site. To check out the page, load your graphical Web browser, connect to the Internet (if you're not already connected), and enter **http://www.apple.com/**.
When the page loads in your browser, you'll see the interface, which looks a little like a futuristic hand-held computer, on-screen.
This example isn't terribly structured, but it allows you to play with the image map interface. You may already have a good deal of experience with such interfaces, especially if you've spent a lot of time on the Web.
By simply pointing at part of the graphic, you may be able to bring up a URL in the status bar at

the bottom of your browser bar as shown below. This bar shows you where the various hot zones for the image map are and at what coordinates your mouse pointer appears.
The image map interface at Apple Computer's Web site.

Check out one more things. If the image map fills your screen, scroll down in your browser window so that you can see what's below the interface on Apple's Web page. The text directly below the interface almost exactly mirrors the hyperlink options you have with the image map, because image maps, unlike click able graphics, don't offer an ALT statement for the various hot zones. So you have to include additional links to cater to your users of no graphical browsers.

Understanding How Image Maps Work

Creating an image map involves three steps: creating the graphic, mapping the graphic for hot zones, and placing the correct information (along with the correct programs) on the Web server itself. This section discusses the Web server; the next section talks about defining hot zones.
To offer your users the option of using image maps, you must have a special map server program running on your Web server. For UNIX-based servers, this program will most often be NCSA Image map; other platforms have their own map server programs.

The Map Server Program

When a user clicks an image map on a Web page, the browser determines the coordinates of the graphic (in pixels) that describe where the user clicked. The browser then passes these numbers to the map server program, along with the name of the file that contains the URLs that correspond to these coordinates.
NCSA Image map, then, simply accepts the coordinates and looks them up in the database file that defines the hot zones for that image map. When NCSA Image map finds those coordinates and their associated URL, it sends a "connect to URL" command (just as a hypertext link does) that causes your browser to load the appropriate HTML document.

Defining Your Image Map Hot Zones

As a designer, you are responsible for doing two things in the hot zone definition process. First, you need to define the hot zones to create the image map-that is, you need to decide what URL the coordinates will correspond to when the image map is clicked. Second, you need to create the map definition file that makes the hot zone information available to the Web server. For Windows and Macintosh users, luckily, programs that do both are available.

MapEdit for Microsoft Windows and X-Windows

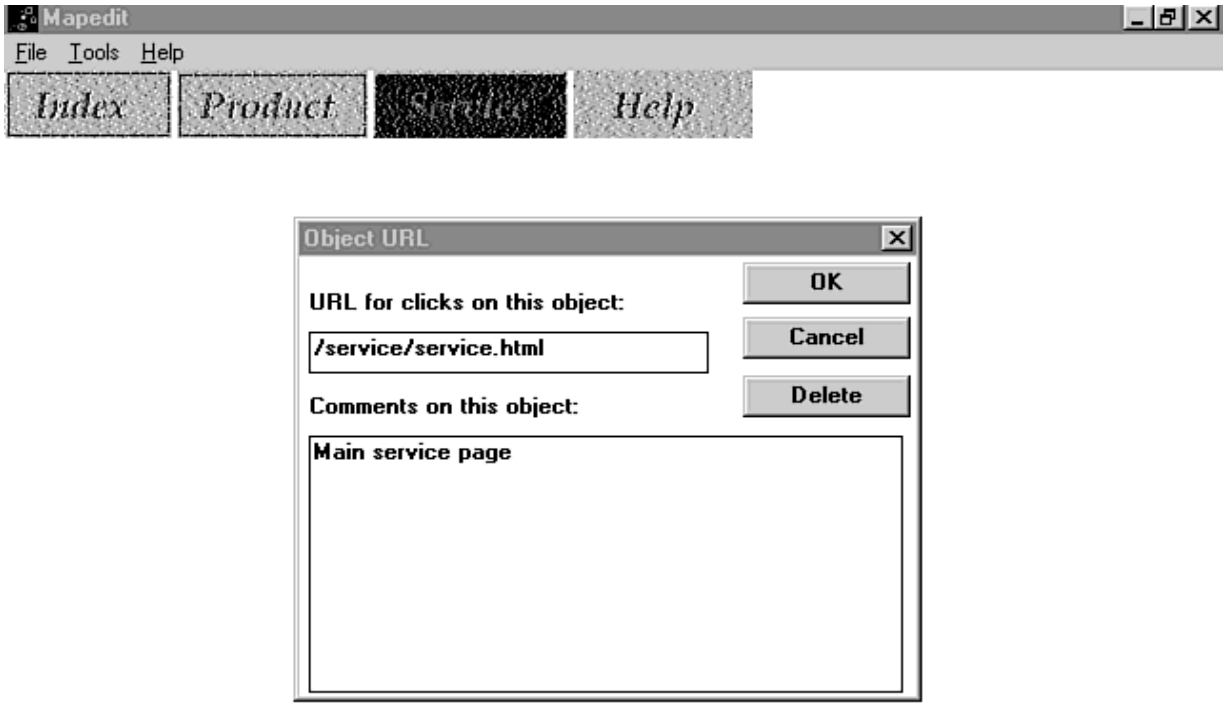
Available for all flavors of Windows (Windows 95, Windows 3.1, and Windows NT) and for most types of UNIX, MapEdit is a powerful program that allows you to graphically define the hot zones for your image maps. You can access and download the latest version of this program via the MapEdit Web site (**http://www.boutell.com/mapedit/**).
When you have the program installed and you double-click its icon to start it, follow these steps to define your map:
Choose File, Open/Create from the MapEdit menu. The Open/Create Map dialog box appears.
In the Open/Create Map dialog box, enter the name of the map definition file you want to create and the name of the graphic file you want to use for your map. You should also use the radio buttons to determine whether you'll use CERN or NCSA map definitions. (Consult your map server software or ISP if you're not sure whether to use CERN or NCSA.)

Click the OK button. The Creating New Map File dialog box appears. In this dialog box, click Yes. After a moment, MapEdit displays your image file.

To create a new hot zone, choose the shape from the Tools menu; then click one time for each point required for the shape. For a rectangle, click once to start the rectangle and then

click where you'd like the opposite corner of the triangle to appear. For a circle, click for the middle, and then drag out the circle and click when you've got the right radius. The triangle tool is actually a "polygon" tool, so click for each point in the polygon. Then, right-click at the last point (to connect your last point to the first point and complete the shape).

When the shape is created, the Object URL dialog box appears (see fig. 9.3). Enter the URL that you want to associate with your new hot zone. (You also can enter comments, if you want.) Then click OK to continue.
Add more shapes by following steps 4 and 5 until you finish mapping your graphic.
Choose File, Save. Now you have a .MAP file for your image map.



Associating an URL with the hot zone.

Tip: By choosing File, Edit Default URL, you can determine whether your image map includes a default URL for clicks outside your hot zones.

Example: MapEdit and a Simple Button Bar

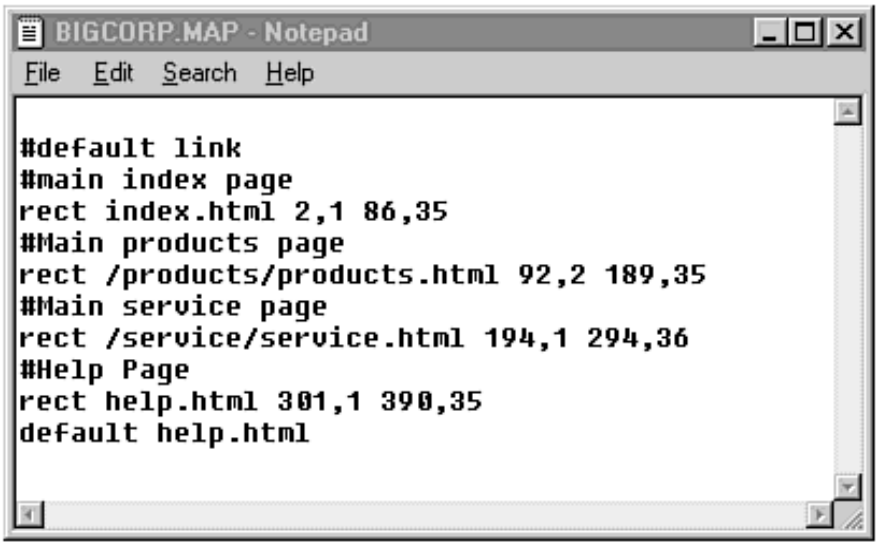
In this example, you use MapEdit to create a simple button bar-a little like the menu bar that you created with click able graphics in Chapter 11, except for the fact that this one is an image map. Start by drawing an appropriate graphic in a graphics application and saving it as a GIF file. For this example, name the file testbar.gif. Then follow these steps:

1. Open MapEdit, and choose File, Open/Create. The Open/Create Map dialog box appears.
2. In this dialog box, enter **testbar.map** for the map file and **testbar.gif** for the graphics file. (If you saved the GIF file in a different directory, use the Browse button to find and load it.)
3. When the graphic loads, pull down the Tools menu and make sure that Rect is selected.
4. Draw rectangles for the buttons, providing an appropriate URL for each button. For this example (four buttons in all), use the following URLs:

http://www.fakecorp.com/index.html
http://www.fakecorp.com/product.html
http://www.fakecorp.com/service.html
http://www.fakecorp.com/help.html

5. Choose File, Edit Default URL. The Default URL dialog box appears.
6. Enter the following URL:
   http://www.fakecorp.com/error.html
7. Choose File, Save.
8. Choose File, _Quit.

We've created your map definition file. To look at the file, open Notepad (or a similar text editor), and load the file testbar.map into it. The file should look something like figure below (although the coordinates are bound to be slightly different).



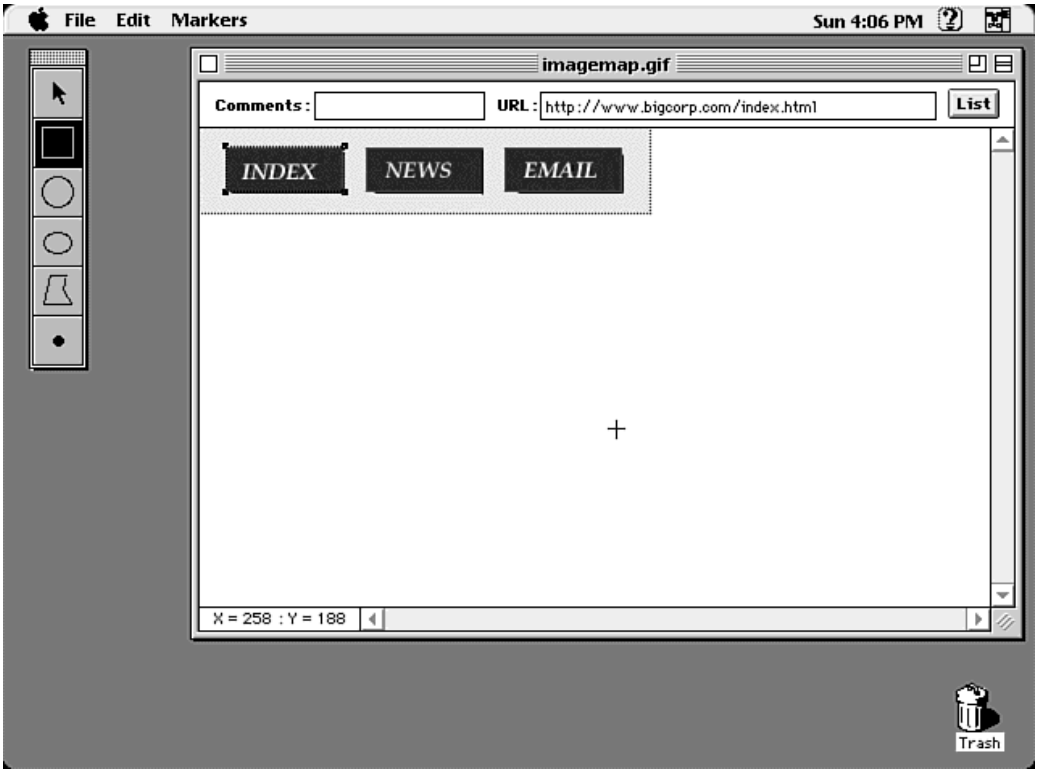
A successful map definition file created in MapEdit.

WebMap for Macintosh

If you're a Macintosh user, you can use a program called WebMap, which is similar to MapEdit. You can download WebMap from **http://www.city.net/cnx/software/webmap.html**. Install the program; then double-click its icon to start it.
To create an image map in WebMap, follow these steps:

- Choose File, Open.
- In the Open dialog box, select the graphic that you want to use for your map and the name of the map definition file that you want to create.
- Click the OK button. After a moment, MapEdit displays your image file.
- To create a new hot zone, choose the shape from the floating tool palette, and drag to create a hot zone. For a rectangle, circle, or oval, click and hold the mouse in the top left corner of your shape, drag the mouse to make the shape the desired size, and then release the mouse button. To create a polygon, choose the polygon shape from the tool palette and then click once on the graphic for each point in your polygon. To complete the shape, click once on the first point you created.
- When the shape is created, enter the URL in the space provided above the graphic file You can use the pointer tool (the one that looks like a mouse pointer) to select different shapes that you've created and then edit their URLs.

- To create a default URL, use the pointer tool to click the graphic background (not a shape). Default URL should appear in the comment window. Then enter the default URL in the URL text box.



Using WebMap to create hot zones.

To create your map definition file, pull down the File menu and choose Export As Text. In the resulting dialog box, you can name your map file and save it in CERN or NCSA format. Now you're free to save the graphic and quit the program.

Adding Image Maps to Your Web Page

After you create your image map and your map definition file, you're ready to add a link for your image map to your HTML page. You can accomplish this task in a couple of ways, depending on your Web server. In essence, though, the only major difference between an image map and a clickable image is a new attribute for the <IMG> tag: ISMAP.
Image maps follow this format:
<IMG SRC="*graphic.ext*" ISMAP>
Note: It's perfectly acceptable to add other <IMG> tag attributes (such as ALT) to your image map definition.
Using the ISMAP attribute doesn't do much for you unless the image map is also a hyperlink, so the following code is everything that you need to add an image map to your Web page:
<A HREF="*URL*"><IMG SRC="*graphic.ext*" ISMAP></A>
Our next step is to figure out what to use as the URL in this hyperlink.

The Image Map URL
The URL that you're interested in accessing isn't a particular Web page, because using an URL to a particular Web page would defeat the image map concept; the link would act like a regular clickable graphic. Instead, you want the URL to access the map definition file. You'll have to ask your ISP (or figure out for yourself) where on the server the map file is stored.

Some Web servers allow you to store the map definition file anywhere on the server; the servers are smart enough to figure out that you're accessing a map definition file and take care of the rest. In that case, you could simply store the map definition file in the current directory and access it as follows:

<A HREF="mymap.map"><IMG SRC="mymap.gif" ISMAP></A>
If you have an understanding server, this method may work for you.
Other servers may require you to access a particular directory on the server, such as the /cgi-bin/ or /bin/ directory, where server scripts (mini computer programs) are stored. In such a case, something like the following examples may be the way to access the image map:
<A HREF="http://www.myserver.com/cgi-bin/mymap.map><IMG SRC="mymap.gif" ISMAP></A>
or
<A HREF="http://www.myserver.com/bin/mymap.map><IMG SRC="mymap.gif" ISMAP></A>

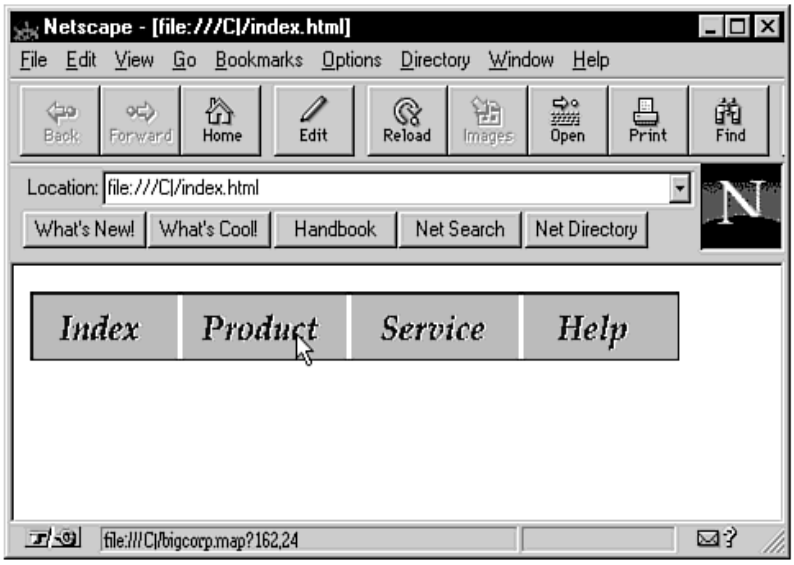
If the server requires you to access one of these scripting directories, though, it may not want you to access the map definition file directly. Instead, the server will want you to use an alias.
Some servers store all map information in a single database file (often called imagemap.conf) and require you to access information within the database by using an alias. You and your Web server administrator have to determine what this alias is. In that case, your link would look more like the following:
<A HREF="http://www.myserver.com/bin/mymap><IMG SRC="mymap.gif" ISMAP></A>
Example: Testing Your Link
The best way by far to participate in this example is to confer with your ISP, place your map definition file on the Web server, and test it from a remote location using the correct URL. If that procedure doesn't work, you can manage some testing on your own.
Save your template as a new HTML file, and have an image-mapped graphic handy in the same directory. Then enter Listing 9.1 between the <BODY> tags.

---

img_map.html Adding Image Maps in HTML

```
<BODY>
<A HREF="http://www.server.com/mymap.map><IMG SRC="mymap.gif"
ISMAP ALT=
"My Image Map"></A>
<H2>Welcome to my page!</H2>
</BODY>
```

Note: If you're going to test this example on an actual Web server, you need to replace the URL with the appropriate one for your Web site (and add the type of link to your map info file that's required for your server). Also, use the real name of the mapped GIF file in the <IMG> tag.
Save the HTML file and then load it in a graphical browser. If your graphic came up, chances are that you set the <IMG> tag correctly. Notice that many browsers do not display a colored link border around the graphic, because the graphic is now considered to be an image map.
Before clicking any of the hot zones, move your mouse pointer around on the image map graphic. If you have a status bar at the bottom of your browser window, you may notice that the link keeps changing (see fig. below). Along with the URL of your map definition file, you should be seeing the current coordinates of your pointer. All this information is sent to the map server to help it figure out what region you clicked. (If you're testing this image map from your local drive, the status bar test is the only part of the example that will work.)



An example image map, showing the URL and the coordinates that it will access if clicked

Now, if you are testing your image map on the Web server, go ahead and click the map to make sure that all the links work. If you're viewing the image map locally, turn off the graphics-loading option in your browser, and reload the page. You should notice that there's now no way to access the hyperlinks in the image map-that's why you also need text links for your image map pages.

Image Map Design Tips

This chapter has covered creating and linking an image map to your Web page fairly thoroughly. Image maps are a bit of a departure from standard text-markup HTML, however, so you should learn a little bit of design theory and Web-related netiquette before you leave this chapter. Please try to keep some of the following suggestions in mind when you add image maps to your Web pages:

- **Use image maps sparingly**. The best way to use an image map is as a clickable menu bar or some other easy-to-recognize interface element. The point isn't really to see how graphical you can make your Web pages-just how intuitive.
- Remember that image maps are usually little more than big graphics files. Ultimately, the key to graphics on the Web is keeping them small. Even if your image map is incredibly attractive, users will be annoyed if they have to wait many minutes for their four possible choices to download to their browsers. Use all the tips in Chapter 9 to keep your graphic as small as possible, and use image maps only to enhance usability.
- Image maps require redundant text links. Unless you plan to leave out everyone who can't view your graphics, you need to create text links that do everything that your image map does. Remember that with clickable graphics, the ALT attribute takes care of the problem. The ALT attribute doesn't work for image maps, because a single image map graphic can have many links, so you need to create an identical text link on your page for every hot zone link in your image map.
- **Stick to normal shapes whenever possible**. You should try to be conservative with your image maps (see fig. below). A graphic that looks as though it has rectangular buttons should function as though it has rectangular buttons. In other words, make your hot zones correspond logically to the image map graphics. Random hot zones randomly annoy users.

Some sites make it their business to use image maps that break the rules. This one doesn't.

CHAPTER 9: HTML Forms

Obejctives:

- Explain Web form and it's use
- Form Elements, it's attributes
- Formatting of web form

The next set of HTML tags is designed to allow you to enhance the interactivity of Web pages by increasing ability to request information from users. Using the forms tags, we can ask users to enter text information, choose from menus, mark checkboxes, make choices from radio buttons, and then send that information to the Web server for processing.

Using Forms and Form-Capable Browsers

Although the forms tags are a part of the HTML 3.0 standard, it's important to recognize that not all browsers are capable of viewing them-especially older browsers and text-based browsers. Users need to have forms-aware browsers, like the current versions of NCSA Mosaic, Netscape Navigator, and Microsoft Internet Explorer, among others. Generally, other browsers will simply ignore the forms commands if they can't deal with them.

Tip: It's a good idea to let your users know that they're about to jump to a form-based page whenever possible. Forms pages are a waste of time for users of older browsers that don't support them.

The idea behind a Web form is simple-it allows us to accept information or answers from our users with varying levels of guidance. Users can be asked to type answers, choose their answers from a list of possibilities you create, or even be limited to choosing one answer from a number of options that you specify.

That data is then passed on to the Web server, which hands it to a script, or small program, designed to act on the data and (in most cases) create an HTML page in response. In order to deal with forms data then, you need to understand a little something about scripting, or programming, for a Web server-or know someone who does.

Note: Most Web server scripts are written in Perl, C, or UNIX shell scripts. If your Web server is DOS, Windows, or Mac based, however, you may have other options. Some DOS Web servers allow you to script in the DOS batch language, while some Windows servers can accept Visual Basic scripts (not to be confused with Microsoft's new Visual Basic Script language). Mac Web servers generally allow for AppleScript or Frontier scripting.

Creating the Form

In an HTML document, forms are set between the <FORM> container tags. The form container works as follows:

<FORM METHOD="*how_to_send*" ACTION="*URL of script*">
...*form data*...
</FORM>

Notice that the <FORM> tag takes two attributes: METHOD and ACTION. The METHOD attribute accepts either POST or GET as its value. POST is by far the more popular, as it allows for a greater amount of data to be sent. GET is a little easier for Web programmers to deal with, and is best used with single responses, like a single textbox.

The second attribute is ACTION, which simply accepts the URL for the script that will process the data from your form. Most often the script is stored in a directory called bin/ or cgi-bin/ located on your Web server.

An example of the <FORM> tag then, would be the following:

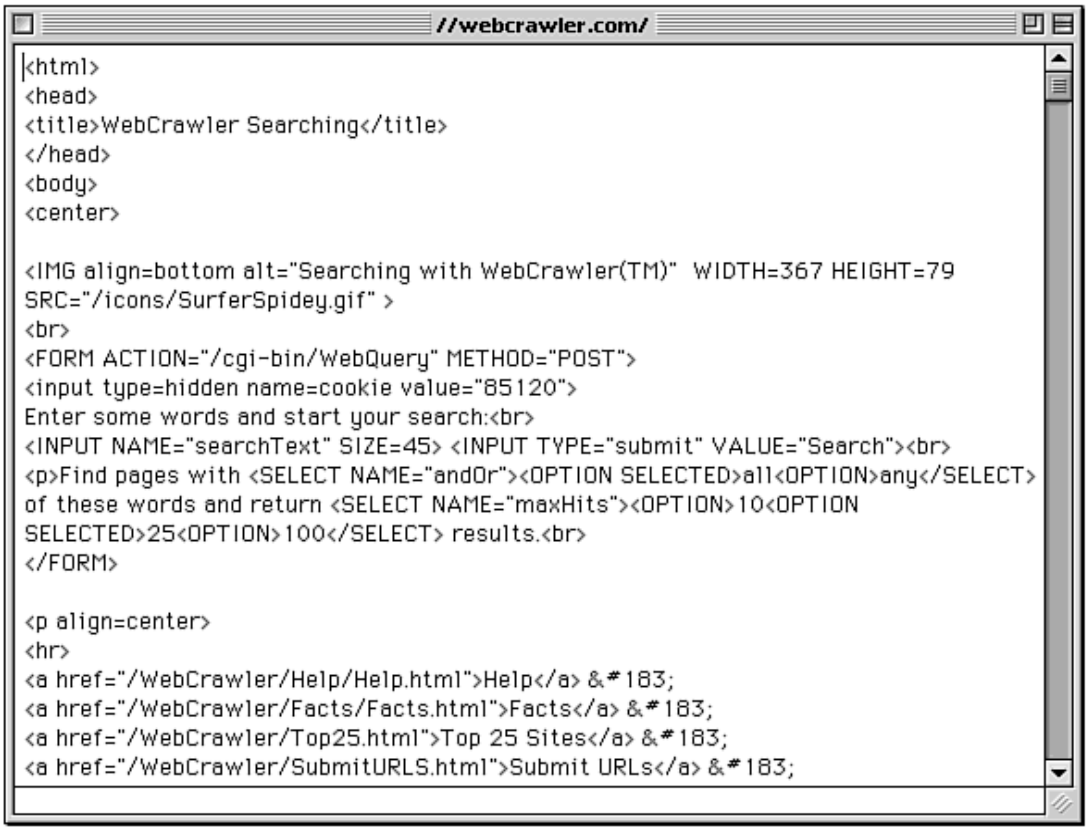
```
<FORM METHOD="SEND" ACTION="http://www.fakecorp.com/cgi-bin/register_script">
</FORM>
```

As with any HTML container tag, this implementation of the <FORM> tag has actually created a complete form (just like <P> and </P> is a complete paragraph). Unfortunately, our complete form doesn't *do* anything yet, so that's somewhat academic.

Note: You can't nest forms within one another. You need to add the end tag </FORM> for the first form before creating another one in the same document. Generally, browsers will ignore any new occurrences of the <FORM> tag, since the purpose of the tag is to tell the browser how to submit data to the server, and different parts of the form can't be submitted in different ways.

Example: Someone Else's Form

Let's take a quick look at a form that's been created by someone else-one that most seasoned Web browsers have encountered at one time or another. Load up your Web browser and point it to **http://webcrawler.com/**.
This is the WebCrawler page, a Web search engine offered by America Online. Your next step is to view the source of this document. Select the View Document Source command in your Web browser's Edit menu. What you see will look something like figure 10.1.



Example of an HTML form available on theWeb.

Note: Nearly all-graphical browsers have a View Source command. Look in the Edit menu for this command or a command with a similar name. The HTML source of the current Web document will then be displayed or saved as a text file.
Notice a couple of things here. The <FORM> tag at WebCrawler is using the ACTION and METHOD attributes that were discussed. ACTION is accessing a script called WebQuery found in the cgi-bin/ directory of the Web server. The METHOD used is SEND.

Text Fields and Attributes

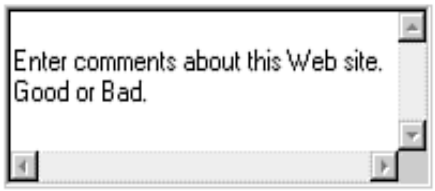
One of the more common uses for forms is to accept multiple lines of text from a user, perhaps for feedback, bug reports, or other uses. To do this, use the <TEXTAREA> tag within your form. You can set this tag to control the number of rows and columns it displays, although it will generally accept as many characters as the user desires to enter. It takes the following form:

```
<TEXTAREA NAME="variable_name" ROWS="number" COLS="number">
default text
</TEXTAREA>
```

It may surprise you to find that <TEXTAREA> is a container tag, since it just puts a box for typing on the page. What's contained in the tag is the default text-so you can guide your users by letting them know what you'd like entered there. For instance:

```
<FORM>
<TEXTAREA NAME="comments" ROWS="4" COLS="40">
Enter comments about this Web site.
Good or Bad.
</TEXTAREA>
</FORM>
```

The default text appears in the textbox just as typed. Notice in figure below that text inside the <TEXTAREA> tag works like <PRE> formatted text. Any returns or spaces you add to the text are displayed in the browser window. In fact, notice that by hitting Return after the opening <TEXTAREA> tag, I'm inserting a blank line at the top of the textarea (in many browsers).



The <TEXTAREA> tag in action.

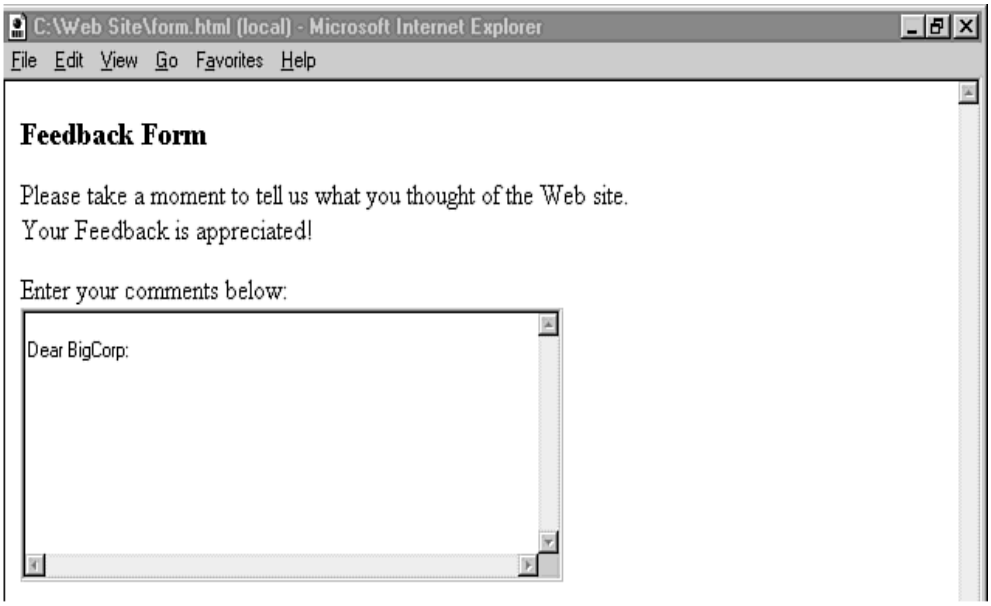
The NAME attribute is a variable name for this string of text. It gets passed on to your processing script on the Web server. ROWS and COLS can accept different numbers to change the size of the textarea box, but you should take care that the majority of browsers can see the entire box on-screen. It's best to limit COLS to 80, and ROWS to something like 24 (the typical size for a text-based computer screen). But it's up to you.
<TEXTAREA> will also accept one other attribute: WRAP. WRAP can be set to OFF (which is the default if WRAP is not included), VIRTUAL, or PHYSICAL. Setting wrap to PHYSICAL forces the browser to include actual line breaks in the text when sending it to the Web server. VIRTUAL makes the textbox seem to offer line wrap, but sends a continuous stream of words to the Web server (unless the user has entered returns on his or her own).

Example: Web-based Feedback Form

We have mentioned before that <TEXTAREA> is commonly used to gather feedback from your Web users. To create a small form to do just that, save your default template as a new HTML document and enter the following:

```
<BODY>
<H3>Feedback Form</H3>
<P>Please take a moment to tell us what you thought of the Web site.<BR>
Your Feedback is appreciated!</P>
<FORM METHOD="POST" ACTION="cgi-bin/feedback">
Enter your comments below:<BR>
<TEXTAREA NAME="comments" ROWS="10" COLS="70" WRAP="VIRTUAL">
Dear BigCorp:
</TEXTAREA>
</FORM>
</BODY>
```

You can see how this looks in figure below. Notice in the example that some descriptive text is enclosed inside the <FORM> tag, but outside of the <TEXTAREA> tag. This is completely legal-it just lets you explain what the purpose of the textarea is.



Sample textarea HTML form.

You may have realized that there's something lacking in this sample form. There's no way to submit the user's entry! You'll get to that in the next section, when I discuss this next tag for form entry.

The <INPUT> Tag

Our next tag for HTML forms give you the opportunity to be a bit more picky about the type of input you're going to accept from the user. The <INPUT> tag follows the following format:
<INPUT TYPE="*type_of_box*" NAME="*variable*" SIZE="*number*" MAXLENGTH="*number*">

Now, technically, the only required attributes are TYPE and NAME. Some other "types" of the input tag will also accept the attribute VALUE. But first, let's look at the different types of <INPUT>.
Note: By the way, notice that <INPUT> is an empty tag. There's no </INPUT> element.

Text

The first possible value for the TYPE attribute is TEXT, which creates a single-line textbox of a length you choose. Notice that the length of the box and the maximum length entered by the user can be set separately. It's possible to have a box longer (or, more often, shorter) than the maximum number of characters you allow to be entered. Here's an example of a textbox:
Last name:

```
<INPUT TYPE="TEXT" NAME="last_name" SIZE="40" MAXLENGTH="40">
```

When appropriately entered between <FORM> tags, this <INPUT> yields a box similar to figure below. If desired, the attribute VALUE can be used to give the textbox a default entry, as in the following example:

Using the TEXT option with the TYPE attribute.

Type of Computer:

```
<INPUT TYPE="TEXT" NAME="computer" SIZE="50" MAXLENGTH="50" VALUE="Pentium">
```

PASSWORD

The PASSWORD option is nearly identical to the TEXT option except that it responds to typed letters with bullet points or a similar scheme (chosen by the browser) to keep the words from being read. A sample password box could be the following:
Enter Password:
<INPUT TYPE="PASSWORD" NAME="password" SIZE="25" MAXLENGTH="25">
When characters are typed into this textbox, they are shown on the screen as in figure below

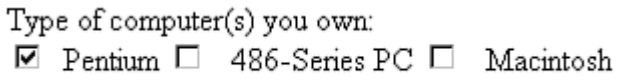
PASSWORD hides text from people looking over your user's shoulder.
Recognize that the text is still stored as the text typed by the user-not as bullet points or similar characters.

CHECKBOX

This value for TYPE allows you to create a checkbox-style interface for your form. This is best used when there are two possible values for a given choice-and no others. You can also determine whether or not a checkbox will already be checked (so that it must be unchecked by the user, if desired), by using the attribute CHECKED. Here's an example of adding checkboxes to a form:
Type of computer(s) you own:

```
<INPUT TYPE="CHECKBOX" NAME="Pentium" CHECKED> Pentium
<INPUT TYPE="CHECKBOX" NAME="486"> 486-Series PC
<INPUT TYPE="CHECKBOX" NAME="Macintosh"> Macintosh
```

In this example, it's possible to check as many of the options as are presented. CHECKBOX evaluates each item separately from any others. Figure below illustrates how CHECKBOX is displayed in a browser.



Notice that Pentium is prechecked.

RADIO

Like CHECKBOX, RADIO is designed to offer your user a choice from pre-determined options. Unlike CHECKBOX, however, RADIO is also designed to accept only one response from among its options. RADIO uses the same attributes and basic format as CHECKBOX.

RADIO requires that you use the VALUE attribute, and that the NAME attribute be the same for all of <INPUT> tags that are intended for the same group. VALUE, on the other hand, should be different for each choice. For instance, look at the following example:

Choose the computer type you use most often:

```
<INPUT TYPE="RADIO" NAME="Computer" VALUE="P" CHECKED> Pentium
<INPUT TYPE="RADIO" NAME="Computer" VALUE="4"> 486-Series PC
<INPUT TYPE="RADIO" NAME="Computer" VALUE="M"> Macintosh
<INPUT TYPE="RADIO" NAME="Computer" VALUE="O"> Other
```

With RADIO, it's important to assign a default value, since it's possible that the user will simply skip the entry altogether. While the user can't check more than one, he or she can check none. So choose the most common value and set it as CHECKED, just so that the form-processing script doesn't have trouble.

Note: Of course, if you want to give your user the option of choosing none, then you can leave off the CHECKED attribute. It's more complete and obvious for the user, however, to include another radio button with a VALUE of none, and make it the CHECKED choice.
HIDDEN

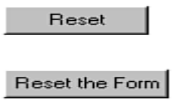
This <INPUT> type technically isn't "input" at all. Rather, it's designed to pass some sort of value along to the Web server and script. It's generally used to send a keyword, validation number, or some other kind of string to the script so that the script knows it's being accessed by a valid (or just a particular) Web page. The <INPUT TYPE="Hidden"> tag takes the attributes NAME and VALUE.

Note: This isn't really terribly covert, since an intrepid user could simply choose View Source to see the value of the hidden field. It's more useful from a programmer's standpoint. For instance, on a large Web site, the hidden value might tell a multi-purpose script which particular form (among many) is sending the data, so the script knows how to process the data.

RESET

The <INPUT> tag has built into it the ability to clear an HTML form. RESET simply creates a push button (named with the VALUE string) that resets all of the elements in that particular FORM to their default values (erasing anything that the user has entered). An example would be the following:
<INPUT TYPE="RESET">
With a VALUE statement, you could enter the following:
<INPUT TYPE="RESET" VALUE="Reset the Form">

The results are shown in figure



Default and VALUE-attributed Reset buttons.

SUBMIT

The <INPUT> tag also has a type that automatically submits the data that's been entered into the HTML form. The SUBMIT type accepts only the attribute VALUE, which can be used to rename the button. Otherwise, the only purpose of the Submit button is to send off all the other form information that's been entered by your user. See the following two examples

Creating a Submit button**.**

```
<INPUT TYPE="SUBMIT">
<INPUT TYPE="SUBMIT" VALUE="SEND IT IN!">
```

You can use just about anything you want for the VALUE, although it's best to remember that really small words, like *OK*, don't look great as buttons. To make a button larger, enter the VALUE with spaces on either end, like in the following:
<INPUT TYPE="SUBMIT" VALUE=" GO ">

Example: A More Complete Form

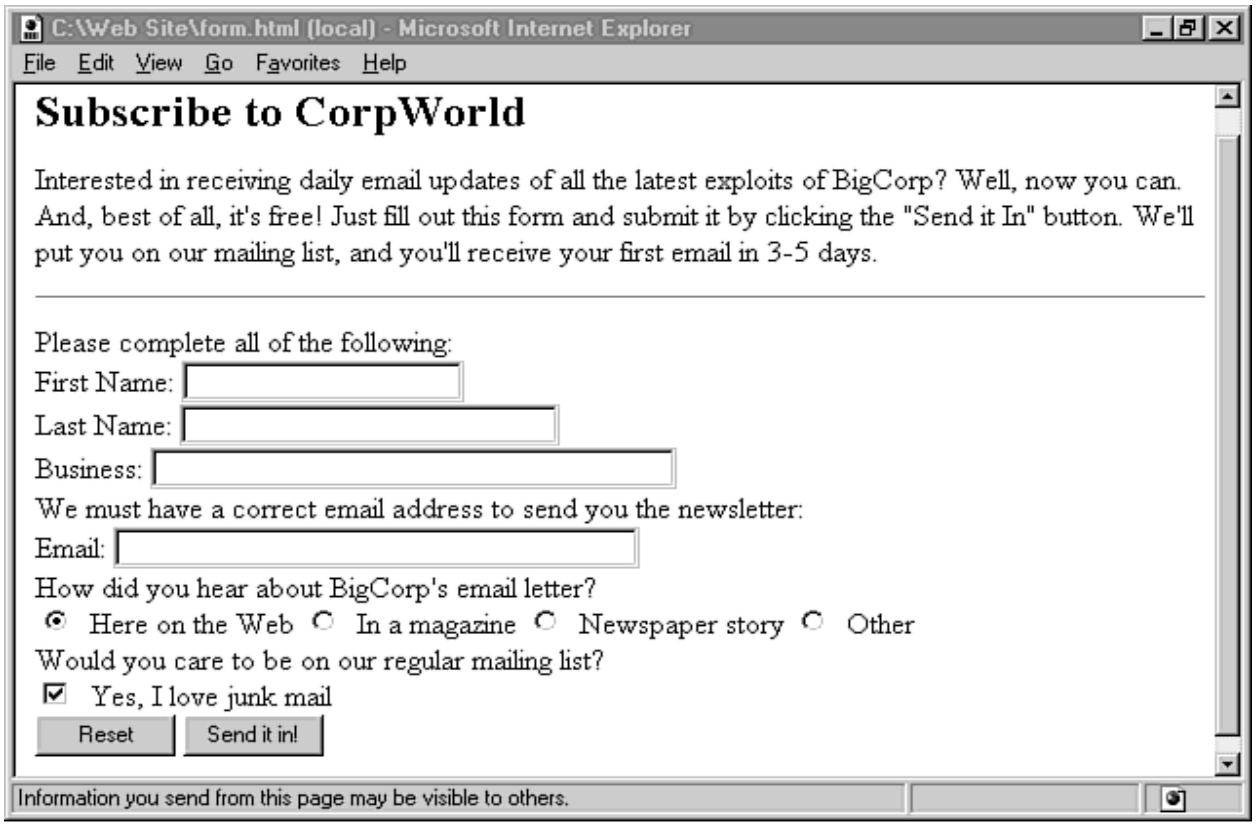
Along with all the other <INPUT> types, now you've finally got a way to submit data. So, let's create a more involved form that includes some of these examples-a subscription form.
Save your HTML template to create a new document. Then, enter something similar to Listing below.

---

scrp_frm.html Creating a Complete Form

```
<BODY>
<H2>Subscribe to CorpWorld</H2>
<P>Interested in receiving daily email updates of all the latest exploits of
BigCorp? Well, now you can. And, best of all, it's free! Just fill out this form and
submit it by clicking the "Send it In" button. We'll put you on our mailing list, and
you'll receive your first email in 3-5 days.</P>
<FORM METHOD="Send" ACTION="http://www.fakecorp.com/cgi-
bin/subscribe">
Please complete all of the following:<BR>
First Name: <INPUT TYPE="Text" Name="first" SIZE="25"
MAXLENGTH="24"><BR>
Last Name:  <INPUT TYPE="Text" Name="last" SIZE="35"
MAXLENGTH="34"><BR>
Business:   <INPUT TYPE="Text" Name="business" SIZE="50"
MAXLENGTH="49"><BR>
We must have a correct email address to send you the newsletter:<BR>
Email:      <INPUT TYPE="Text" Name="email" SIZE="50"
MAXLENGTH="49"><BR>
How did you hear about BigCorp's email letter?<BR>
<INPUT TYPE="RADIO" NAME="hear" VALUE="web" CHECKED>Here on the
Web
<INPUT TYPE="RADIO" NAME="hear" VALUE="mag">In a magazine
<INPUT TYPE="RADIO" NAME="hear" VALUE="paper">Newspaper story
<INPUT TYPE="RADIO" NAME="hear" VALUE="other">Other
<BR> Would you care to be on our regular mailing list?<BR>
<INPUT TYPE="CHECKBOX" NAME="snailmail" CHECKED> Yes, I love junk
mail<BR>
<INPUT TYPE="RESET">
<INPUT TYPE="SUBMIT" VALUE="Send it in!">
</FORM>
</BODY>
```

Notice that, for text type <INPUT> boxes, the MAXLENGTH is one less than the size of the box. This tends to look a little better, but choosing the size is up to you. Figure below shows how it looks on a Web page.



The complete form in MS Internet Explorer.

Creating Pop-Up and Scrolling Menus

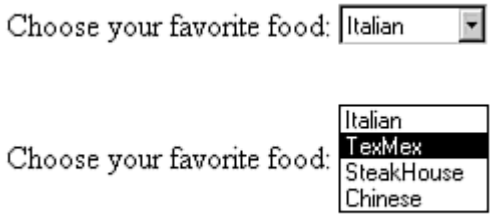
The last types of input that you can offer to users of your Web page revolve around the <SELECT> tag, which can be used to create different types of pop-up and scrolling menus. This is another element designed specifically for allowing users to make a choice-they can't enter their own text. The <SELECT> tag requires a NAME attribute and allows you to decide how many options to display at once with the SIZE attribute.

Using <SELECT>

Also notice that, like <TEXTAREA>, <SELECT> is a container tag. Options are placed between the two <SELECT> tags, each with a particular VALUE that gets associated with <SELECT>'s NAME attribute when chosen. The following is the basic format:
<SELECT NAME="*variable*">
<OPTION SELECTED VALUE="*value*"> *Menu text*
<OPTION VALUE="*value*"> *Menu text*
...
</SELECT>
The attribute SELECTED is simply designed to show which value will be the default in the menu listing. *Value* can be anything you want to pass on to the Web server and associated script for processing. An example might be:
Choose your favorite food:

```
<SELECT NAME="food">
<OPTION SELECTED VALUE="ital"> Italian
<OPTION VALUE="texm"> TexMex
<OPTION VALUE="stek"> SteakHouse
<OPTION VALUE="chin"> Chinese
</SELECT>
```

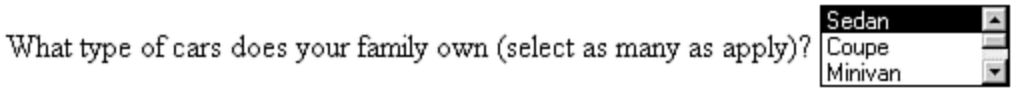
You can also use the SIZE attribute to decide to display the menu in its entirety, by simply changing the first line of the example to the following:
<SELECT NAME="food" SIZE="4">
Both examples are shown in figure below.



Two <SELECT> menus-a pop-up and a fixed.

In the first example, selecting the menu item with the mouse causes the menu to pop-up on the page. The user can then select from the choices. In the second example, it's necessary to click the desired item.

Allowing More than One Selection
One more attribute for the <SELECT> tag allows the user to select more than one option from the menu. Using the MULTIPLE attribute forces the menu to display in its entirety, regardless of the SIZE attribute. An example might be the following: (the result appears in figure below)



A <SELECT> menu can allow multiple choices.
What type of cars does your family own (select as many as apply)?

```
<SELECT NAME="cars" MULTIPLE>
<OPTION VALUE="sedan"> Sedan
<OPTION VALUE="coupe"> Coupe
<OPTION VALUE="mivan"> Minivan
<OPTION VALUE="covan"> Conversion Van
<OPTION VALUE="statn"> Stationwagon
<OPTION VALUE="sport"> SUV (4x4)
<OPTION VALUE="truck"> Other Truck
</SELECT>
```

CHAPTER 11: Adding Tables to Your Documents

Creating a Table

Tables work a lot like HTML list tags, in that you must use the table container tag to hold together a group of tags that define each individual row. The main container is the <TABLE> tag, which uses enclosing tags for table rows (<TR>) and table data (<TD>). Most tables will also use an element for table headers (<TH>), which is generally used as the title text for rows and columns. Tables take the following format:

<TABLE>
<CAPTION>*Caption text for table*</CAPTION>
<TR><TH>*column1*</TH><TH>*column2*</TH><TH>*column3*</TH>
<TR><TD>*row1data1*</TD><TD>*row1data2*</TD><TD>*row1data3*</TD>
<TR><TD>*row2data1*</TD><TD>*row2data2*</TD><TD>*row2data3*</TD>
...
</TABLE>

An example of a table using this format might be the following:

```
<TABLE>
<CAPTION>Team Members for 3-Person Basketball</CAPTION>
<TR><TH>Blue Team</TH><TH>Red Team</TH><TH>Green Team</TH>
<TR><TD>Mike R.</TD><TD>Leslie M.</TD><TD>Rick G.</TD>
<TR><TD>Julie M.</TD><TD>Walter R.</TD><TD>Dale W.</TD>
<TR><TD>Bill H.</TD><TD>Jenny Q.</TD><TD>Fred B.</TD>
</TABLE>
```

After you work with HTML list containers, it's fairly easy to make the jump to creating tables in HTML. You can see how this table looks in figure below.

Team Members for 3-Person Basketball

Blue Team	Red Team	Green Team
Mike R.	Leslie M.	Rick G.
Julie M.	Walter R.	Dale W.
Bill H.	Jenny Q.	Fred B.

A simple table in HTML.

The <TABLE> Tag

The <TABLE> tag is actually a rather complex creature, at least insofar as it can accept many different attributes. Some of the attributes are more useful than others, so let's look at the most useful of them as they currently stand:

- **ALIGN**. The ALIGN attribute is used to determine where the chart will appear relative to the browser window. Valid values are ALIGN=LEFT and ALIGN=RIGHT. As an added bonus, text will wrap around the table (if it's narrow enough) when the ALIGN=LEFT or ALIGN=RIGHT attributes are used.
- **WIDTH**. The WIDTH attribute sets the relative or absolute width of your table in the browser window. Values can be either percentages, as in WIDTH="50%", or absolute values. With absolute values, you must also include a suffix that defines the units used, as in px for pixels or in for inches (e.g., WIDTH="3.5in"). Absolute values for table widths are discouraged, though.

- **COLS**. The COLS attribute specifies the number of columns in your table, allowing the browser to draw the table as it downloads.
- **BORDER**. The BORDER attribute defines the width of the border surrounding the table. Default value is 1 (pixel).
- **CELLSPACING**. The CELLSPACING attribute tells the browser how much space to include between the walls of the table and between individual cells. (Value is a number in pixels.)
- **CELLPADDING**. The CELLPADDING attribute tells the browser how much space to give data elements away from the walls of the cell. (Value is a number in pixels.)

It is definitely not necessary to use all of these attributes for your table-in fact, the simple table example earlier didn't use any of them. Often, however, they will come in handy.

Example: Playing with Table Attributes

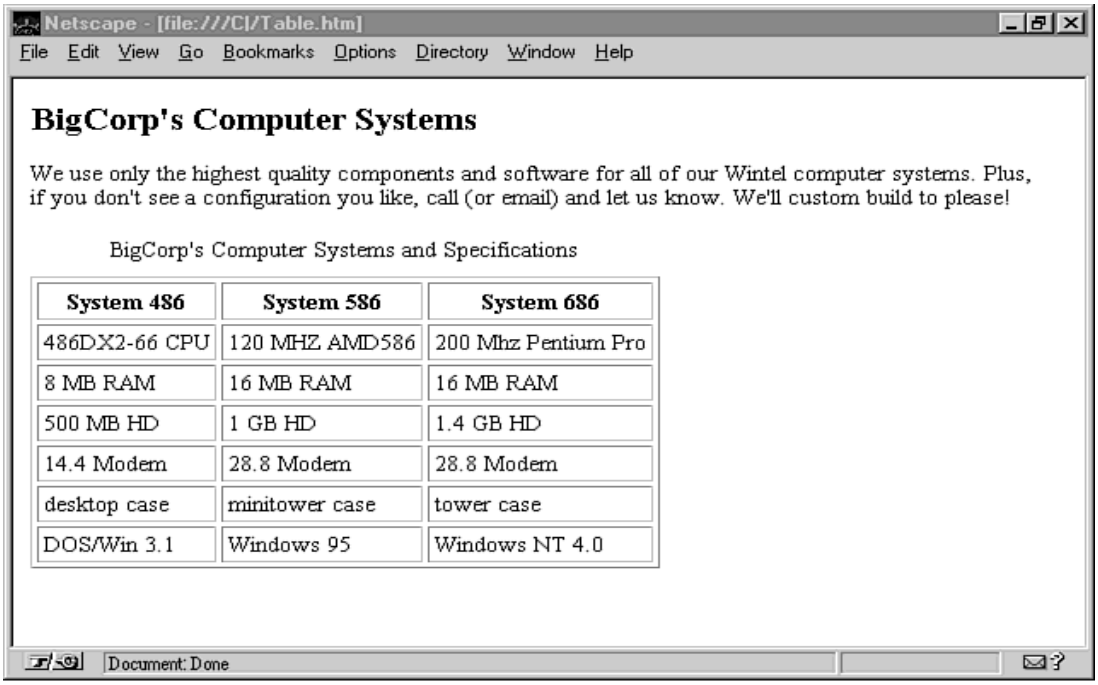
This is another fairly freeform example. Let's look at the difference between a plain table and a table embellished with a few attributes. Insert Listing below in a new HTML document.

---

badtable.html Creating a Plain Table

```
<BODY>
<H2> BigCorp's Computer Systems </H2>
<P>We use only the highest quality components and software for all of our
Wintel computer systems. Plus, if you don't see a configuration you like,
call (or email) and let us know. We'll custom build to please! </P>
<TABLE>
<CAPTION>BigCorp's Computer Systems and Specifications</CAPTION>
<TR><TH>System 486</TH><TH>System 586</TH><TH>System 686</TH>
<TR><TD>486DX2-66 CPU</TD><TD>120 MHZ AMD586</TD><TD>200 Mhz
Pentium Pro</TD>
<TR><TD>8 MB RAM</TD><TD>16 MB RAM</TD><TD>16 MB RAM</TD>
<TR><TD>500 MB HD</TD><TD>1 GB HD</TD><TD>1.4 GB HD</TD>
<TR><TD>14.4 Modem</TD><TD>28.8 Modem</TD><TD>28.8 Modem</TD>
<TR><TD>desktop case</TD><TD>minitower case</TD><TD>tower case</TD>
<TR><TD>DOS/Win 3.1</TD><TD>Windows 95</TD><TD>Windows NT
4.0</TD>
</TABLE>
</BODY>
```

Last time we tried a simple table, it communicated its data well. But this one is fairly ineffective, with everything lined up so poorly. Using just the attributes only mentioned, though, you can change this table so that it looks better to the user and is easier to read.
All that needs to change is the first <TABLE> tag:
<TABLE BORDER ALIGN="LEFT" CELLSPACING="3" CELLPADDING="3">
That makes for a much nicer looking table, complete with borders and lines for cells, and a comfortable amount of spacing to separate cell data elements from one another (see figure below).



Look how nice the table looks with spacing and borders.

The rest of this example is up to you. Play with CELLSPACING and CELLPADDING without a border, for instance, or increase all values out of proportion. See the range of what's available, to help you choose how to format your tables in the future.

Captions, Table Headers, and Table Data

To round out your tables, you have the other basic tags to examine. You've already successfully used <CAPTION>, <TH>, and <TD>, but each has its own attributes and abilities that you need to know about.

<CAPTION>

The <CAPTION> tag is a container for reasons that may be obvious-it allows you to nest other HTML tags within the description. For instance:

<CAPTION><B>Table 3.1 from the book <I>Life in El Salvador</I></B></CAPTION>

Just about any sort of markup tags are possible inside the <CAPTION> tags, although some-like list tags-wouldn't make much sense.

The <CAPTION> tag has one attribute, ALIGN. ALIGN="TOP" and ALIGN="BOTTOM" are encouraged. By default, text is also aligned to center (horizontally). By TOP and BOTTOM, I'm referring to the entire table; the caption will default to the top of the table if not otherwise specified. To align the caption to BOTTOM, for instance, enter the following:

<CAPTION ALIGN="BOTTOM">Table of Common Foods</CAPTION>

The <CAPTION> tag is commonly the first tag just inside the <TABLE> tag (although this placement is not required). Regardless of where you place the <CAPTION> tag, however, you

must use ALIGN to force it to the bottom of the table. Otherwise, it will appear at the top, according to its default.

Let's create an entire table and use the ALIGN attribute to the <CAPTION> tag to force the caption to the bottom, like this:

```
<BODY>
<H3>Favorite Ice Cream Flavors</H2>
<TABLE BORDER>
<CAPTION ALIGN="BOTTOM">Data from the <I>New Jersey
Times</I></CAPTION>
<TR><TH>Date</TH><TH>Chocolate</TH><TH>Vanilla</TH>
<TR><TH>1970</TH><TD>50%</TD><TD>50%</TD>
<TR><TH>1980</TH><TD>76%</TD><TD>24%</TD>
<TR><TH>1990</TH><TD>40%</TD><TD>60%</TD>
</TABLE>
</BODY>
```

When the browser interprets this table, it should place the caption at the bottom of the table, centered horizontally (see fig. below).

Favorite Ice Cream Flavors

Date	Chocolate	Vanilla
1970	50%	50%
1980	76%	24%
1990	40%	60%

Data from the *New Jersey Times*

You can align the caption to BOTTOM.

Table Rows

Table rows (<TR>) can accept one attribute you should concern yourself with-ALIGN. The ALIGN attribute is used to determine how text will appear (horizontally) in each of the rows data cells. For instance:

```
<TR ALIGN="CENTER"><TH>Date</TH><TH>Chocolate</TH><TH>Vanilla</TH>
<TR ALIGN="CENTER"><TH>1970</TH><TD>50%</TD><TD>50%</TD>
<TR ALIGN="CENTER"><TH>1980</TH><TD>76%</TD><TD>24%</TD>
<TR ALIGN="CENTER"><TH>1990</TH><TD>40%</TD><TD>60%</TD>
```

Here, I've added the ALIGN attribute (with a value of CENTER) to the rows in the previous example. Notice now that all cells center data horizontally (see fig. below). This ALIGN attribute can also accept LEFT and RIGHT.

Favorite Ice Cream Flavors

Date	Chocolate	Vanilla
1970	50%	50%
1980	76%	24%
1990	40%	60%

This uses the ALIGN attribute with <TR>.

Note: HTML 4.0 also supports another useful attribute, VALIGN, which accepts the values TOP, BOTTOM, and CENTER. Using this attribute, you can choose to align cells vertically as well as horizontally. Until they support VALIGN, non-HTML 4.0 browsers should ignore VALIGN. Unfortunately, those are currently the most popular browsers!

Table Data and Rows

You've already used the <TH> and <TD> tags to include headers and data in your tables. You may have noticed that, essentially, the only difference between the two is that <TH> emphasizes (boldfaces) the text and <TD> does not. Now, technically, the <TH> is a tag that the browser interprets as a header and thus displays text in a way that's distinct from the <TD> tag. In practice, that generally means it's turned bold.
Aside from accepting nearly any type of HTML markup tags within them, both tags can accept four attributes (in most HTML versions). These are ALIGN, VALIGN, COLSPAN, and ROWSPAN. If you were to add all of these attributes, a typical <TH> (or <TD>) tag would be formatted like the following:
<TH ALIGN="*direction*" VALIGN="*direction*" COLSPAN="*number*" ROWSPAN="*italics*">
ALIGN is used to align the data within the cell horizontally, accepting values of LEFT, RIGHT, and CENTER. Note that ALIGN is redundant when used with the ALIGN attribute of <TR>, unless it is used to override the <TR ALIGN=> setting.
VALIGN is used to align the data vertically within cells. Possible values are TOP, BOTTOM, and CENTER. COLSPAN and ROWSPAN are used to force a cell to span more than one column or row, respectively. An example of this might be:

```
<TABLE BORDER>
<TR><TH>Student</TH><TH>Test 1</TH><TH>Test 2</TH><TH>Average</TH>
<TR><TH>Mike M. </TH><TD>100</TD><TD>75</TD><TD ROWSPAN="3">N/A</TD>
<TR><TH>Susan T.</TH><TD>80</TD><TD>95</TD>
<TR><TH>Bill Y.</TH><TD COLSPAN="2">Dropped Course</TD>
</TABLE>
```

Viewed in a browser, the table looks like figure below

Student	Test 1	Test 2	Average
Mike M.	100	75	
Susan T.	80	95	N/A
Bill Y.	Dropped Course		

Using COLSPAN and ROWSPAN in a table.

CHAPTER 10: Images, Multimedia Objects, and Background Graphics

Obejctives:

- Explain relationship of HTML and multimedia
- To include multimedia object in web page

Inserting Multimedia Objects

One of the latest HTML 4.0 (or, at least, beyond HTML 3.0) initiatives has been the addition of a tag called the <INSERT> tag, which expands on the role of the <IMG> tag by allowing various different multimedia types to be displayed inline. As the bandwidth of connections to the Internet grows, and the technology for inline multimedia grows with it, more and more Web viewers will be capable of viewing inline animations, presentation graphics, movies, and more.
As of this writing, very few browsers support the <INSERT> tag. Unlike some other HTML initiatives, however, this specification has been written with much more involvement from industry leaders like Microsoft, Netscape, Spyglass, and Sun.

The <INSERT> Tag

This is not exactly the easiest tag to get your arms around. Like tables, the <INSERT> tag is a container for other tags that help define the element. But, somewhat unlike tables, most of those contained tags don't actually display anything.

Let's take a look at a typical <INSERT> container:

<INSERT DATA="*URL to multimedia file*" TYPE="*type of file*">
Other Insert tags...
</INSERT>

Already, there are a couple of things you're required to know. You need to know the filename of the multimedia file or the appropriate URL if it's not in the current directory. You also need to know the MIME-style "type" of the data file.

MIME-Style Data Types

MIME (Multipurpose Internet Mail Extensions) data types are simply the standardized way that certain Internet programs can identify the type of ASCII and non-ASCII files that are being transferred to a client application. A very common example of this is the text/html MIME type.
The <INSERT> tag (and HTML in general) is not limited to the official MIME categories and types, hence we'll call them MIME-*style* data types. For the purposes of the <INSERT> tag, this is just a more reliable way to tell a Web browser what type of multimedia file to expect more reliable, that is, than just the file's extension.

Some common MIME-style data types appear in Table below. These and others are all useful for the <INSERT> tag.

Type of File	*MIME Equivalent*
GIF	image/gif
JPEG	image/jpeg
AIFF sound	audio/aiff
WAV sound	audio/x-wav

QuickTime video	video/quicktime
AVI video	application/avi
Real Audio	application/x-pnrealaudio
Macromedia Director	application/x-director
OLE object	application/x-oleobject

Some MIME-Style Data Types for the <INSERT> Tag

MIME-style data types for newer multimedia formats (especially vendor-specific ones like Macromedia Director) will generally be in the form of application/x-*datatype*. More often than not, these are the types you'll use for the <INSERT> tag, since these are the data types used for browser plug-ins.

<INSERT>'s Attributes

Aside from DATA and TYPE, <INSERT> can also accept the attributes ALIGN, WIDTH, HEIGHT, and BORDER. Its format is as follows:

<INSERT ALIGN="*direction*">

ALIGN works much as it does with <IMG>. The values possible for ALIGN are shown in Table below. Notice that some of these values (LEFT, CENTER, MIDDLE) cause <INSERT> to act as a separate object, while the others assume the inserted multimedia object is supposed to be inline with the text of the document. You may recall that this is almost identical to what you learned about <IMG> at the beginning of this chapter.

Value	*Acts as*	*How Object is Aligned*
LEFT	Object	With left border and allows text wrap
RIGHT	Object	With right border and allows text wrap
CENTER	Object	Between browser borders and allows text wrap
TEXTTOP	Inline	Top vertically aligned with top of text's font
MIDDLE	Inline	Middle vertically aligned with middle of text's font
BASELINE	Inline	Bottom vertically aligned with baseline of text
TEXTBOTTOM	Inline	Bottom vertically aligned with lowest point in text

Values for the <INSERT ALIGN> Attribute

WIDTH and HEIGHT accept numbers and unit suffixes (like px for pixels and in for inches). These two attributes are used to define the size of the object for faster downloading. Some browsers will also resize objects according to these attributes, so that you might expand a smaller inline movie's object with WIDTH and HEIGHT, for instance, to save on downloading time. WIDTH and HEIGHT take the following format:

```
<INSERT WIDTH="#units" WIDTH="#units">
```

The last parameter is BORDER, which has a default value of 1. The border will generally only appear when the entire <INSERT> object in enclosed in an anchor tag, as in the following example:

```
<A HREF="intro.html"><INSERT DATA="intro.moov" TYPE="video/quicktime"
ALIGN="LEFT" WIDTH="3in"
HEIGHT="2in" BORDER="2">
</INSERT></A>
```

Using <PARAM> and <IMG> with <INSERT>

Two of the most common tags you'll want to use with the <INSERT> tag are the <PARAM> and <IMG> tags. The <IMG> tag is used just as it has been elsewhere except it's only displayed when the browser isn't able to deal with the type of multimedia file that the <INSERT> tag is trying to send. For instance, if you were sending a Macromedia Director multimedia file from your Web pages, but the receiving browser wasn't able to deal with it, the <INSERT> tag would substitute the <IMG> you'd specified instead.
The <PARAM> tag is used to offer additional parameters to the <INSERT> tag information like how many times to play a movie clip. The <PARAM> tag takes elements NAME and VALUE, which work a little like they do for certain table tags. Unfortunately, each different type of multimedia file will require different NAME and VALUE values, so you'll have to seek those out from the creator of the particular object type you want to send.

Here's an example of the <PARAM> tag:

```
<INSERT DATA="ship.avi" TYPE="application/avi">
<PARAM NAME="loop" VALUE="infinite">
</INSERT>
```

The <IMG> tag is used within an <INSERT> definition in the same way that it is used elsewhere, except that the ALIGN attribute isn't really necessary since the <IMG> will only be used to directly replace the inserted multimedia object. You can add the <IMG> like this:

```
<INSERT DATA="ship.avi" TYPE="application/avi">
<PARAM NAME="loop" VALUE="infinite">
<IMG SRC="ship.gif" ALT="The Ship">
</INSERT>
```

Clearly, you'll often want the graphic to at least represent the multimedia file that can't be displayed (see fig. below). Or, perhaps, you could cause a graphic to load that tells the user that he or she is missing out on something better.

The <INSERT> tag in action.

CHAPTER 11: Netscape Frames

Obejctives:

- Apply frames for formatting
- Explain attributes of frames
- Discuss use of Noframe in HTML
- Targeting frame window

One of the most exciting Netscape-only additions to HTML recently has been the frames specification. Although submitted to the W3C, frames, for now, remain almost exclusively Netscape-only.
Frames aren't overwhelmingly difficult to add to your pages, although they do require a slight shift in thought. Although they seem similar to tables at first, frames are considerably more powerful. So much so, in fact, that frames can even divide your Web page so that it is accessing more than one URL at a time.

The Idea Behind Netscape Frames

Netscape frames are basically another way you can create a unique interface for your Web site. By dividing the page into different parts-each of which can be updated separately there becomes a number of different interface elements you can offer. Even a simple use of the frame specification lets you add interface graphics or a corporate logo to a site, while the rest of your page scrolls beneath it (see fig. below).

A simple frames interface

Using frames in this way takes you one step closer to the ideal Web interface, because it makes it as intuitive and universal as possible.

Frames are ideal for the following:

- **Table of Contents (TOC)**. By placing the TOC in a "column" on your Web page, people can click around your site or your documentation pages without being forced to constantly move "back" to the contents page. Instead, users simply click a new content level in the static frame.
- **Fixed interface elements**. As mentioned previously, you can force click able graphics, logos, and other information to stay in one fixed portion of the screen, while the rest of your document scrolls in another frame.
- **Better forms and results**. Frames also enable you to create a form in one frame and offer results in another frame. This is something we're beginning to see extensively with Web search pages (look to fig. below). With the search text box always available, you're free to change search phrases or pinpoint your search more quickly, without moving back in the hierarchy of the Web pages.

Creating Frames

Probably most unique among the HTML-style tags so far is the <FRAMESET> tag. This container is required for frames-style pages-but it also replaces the <BODY> tag completely on these pages. When you use frames then, you're committed to using them completely-you can't just add frames to part of your page. On a typical page, <FRAMESET> is added like this:

```
<HTML>
<HEAD>
...HEAD markup...
</HEAD>
<FRAMESET>
...Frames and other HTML markup...
</FRAMESET>
</HTML>
```

The <FRAMESET> tag can accept two attributes: ROWS and COLS. Both attributes accept numerical values (size in pixels), percentages, or a combination of both. The value * can also be used to suggest that a particular row or column should take up the rest of the page. The number of values suggests the number of rows or columns you give the attribute. These attributes take the following format:

<FRAMESET ROWS="*numbers,percentages*,*" COLS="*numbers,percentages*, *">

An example like the following would create two rows: one 50 pixels long and another row that took up the rest of the page:

```
<FRAMESET ROWS="50,*">
```

(This would be useful for a page that displays a fixed map or graphic at the top.) The following example would create a Web interface with two columns: one on the leftmost 25 percent of the screen and one on the other 75 percent:

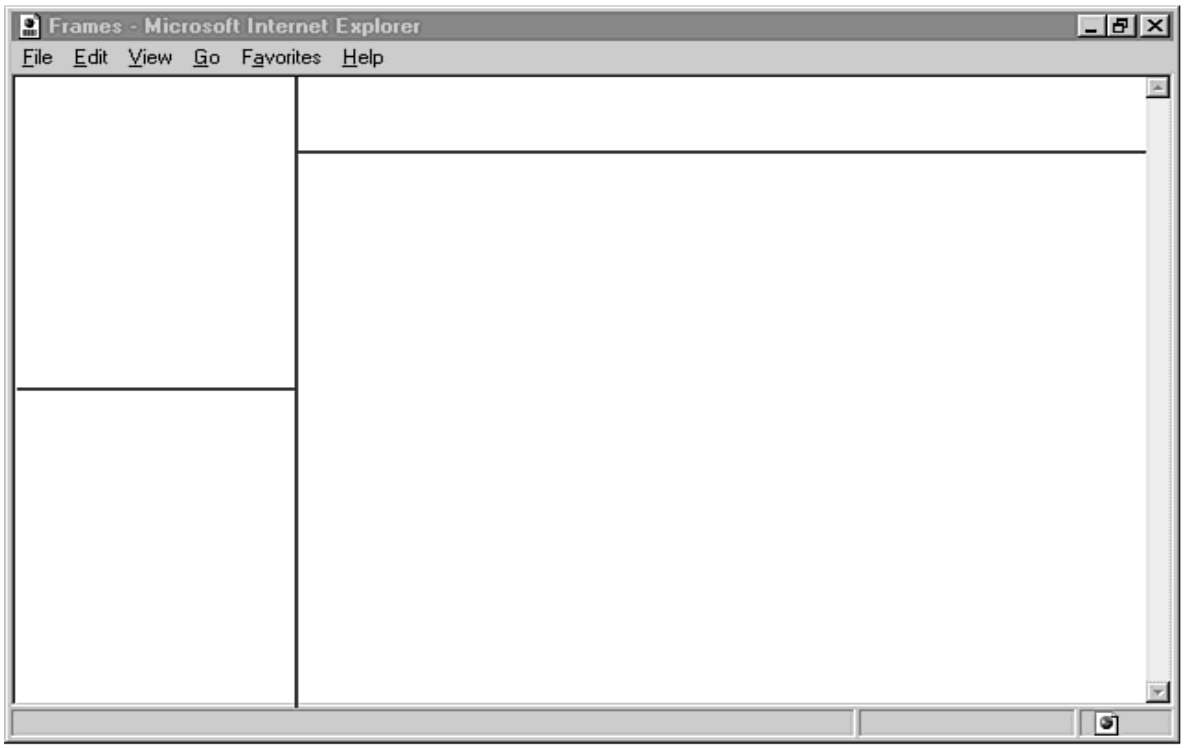
```
<FRAMESET COLS="25%,75%">
```

This would be a good way to set up a documentation (or FAQ) site, that offered contents in the first frame and actual text and examples in the second, larger frame.

Each <FRAMESET> statement will work with one attribute or the other. That means you can only create a frameset with either rows or columns. In order to create rows within columns (or vice-versa), you can nest <FRAMESET> statements. For instance, the following will create a page with two columns:

```
<FRAMESET COLS="25%,75%">
  <FRAMESET ROWS="50%,50%">
  </FRAMESET>
  <FRAMESET ROWS="10%,90%">
  </FRAMESET>
</FRAMESET>
```

The first column will be divided into two rows that take up 50 percent of that column a piece. The second column will be divided into two rows, the first taking ten percent and the second taking the rest of that column. Although this doesn't display anything in and of itself, it creates logical breaks in the page that look like figure below. You'll come back to nesting <FRAMESET> tags as you develop more advanced frame interfaces in this chapter.

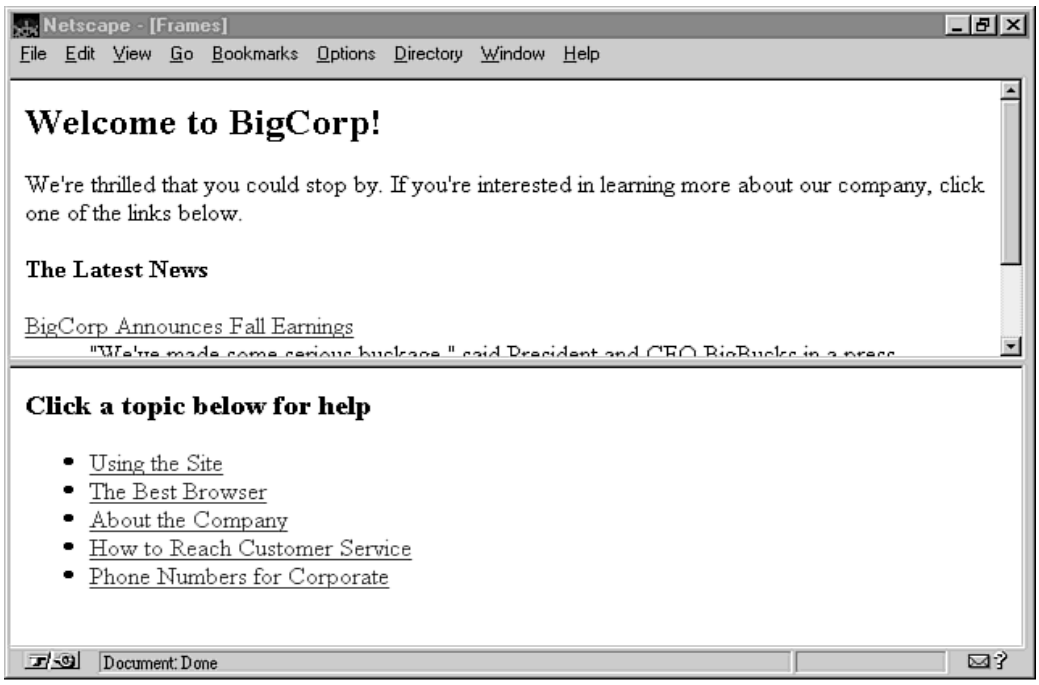


The logical breaks created by nested <FRAMESET> tags.

The <FRAME> Tag

The <FRAME> tag is used within the <FRAMESET> container to determine what will actually appear in a particular frame. Each <FRAME> tag is an empty tag-and it's not unlike the <LI> tags you add to HTML lists. It's simply there, within the <FRAMESET> container, to determine what URL or name is associated with the particular frame it defines. It takes the following format:
<FRAMESET COLS/ROWS="*numbers*">
<FRAME SRC="*URL*">
...
</FRAMESET>

The SRC attribute is used to tell the frame what URL should be loaded in that frame. For instance, the following would create two frame rows-one that loaded the URL index.html at the top of the Web page and one that loaded the URL help.html at the bottom of the page (see fig. below):



The <FRAME> tag assigns URLs to each frame window.

<FRAMESET ROWS="50%,50%">
<FRAME SRC="index.html">
<FRAME SRC="help.html">
</FRAMESET>
By using the <FRAME> tag, you create what's known as a *frame window*. Each window corresponds to a "row" or "column" definition in the <FRAMESET> tag, but nothing is drawn or displayed until an appropriate <FRAME> tag is used to define each individual window.

Example: A Simple Frame Document

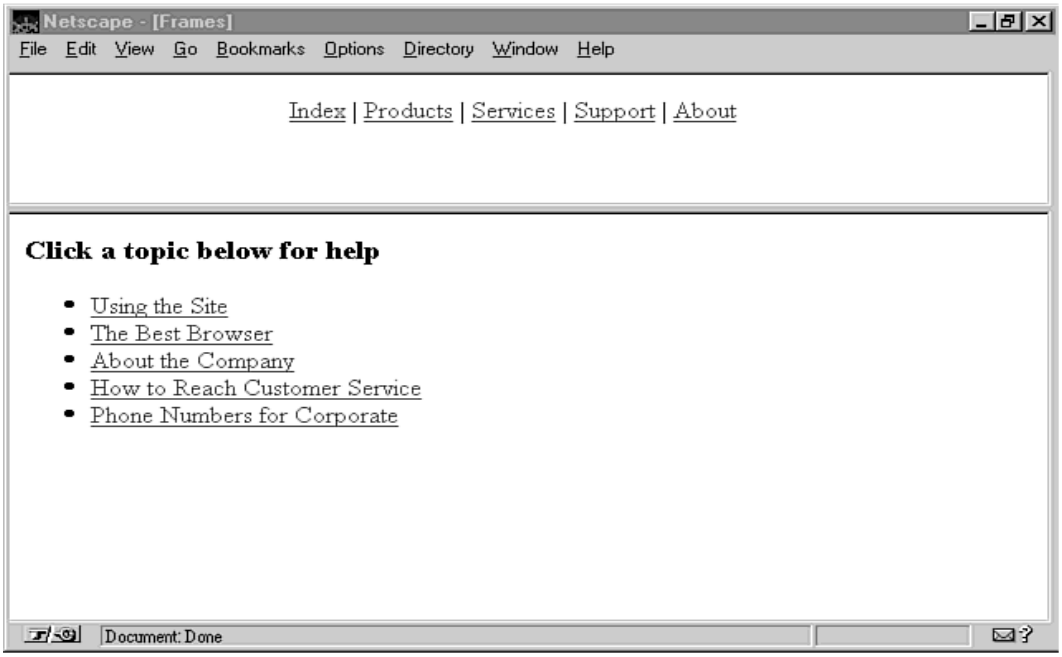
You'll essentially create the same document that was shown in the previous figure, but you should feel free to play with the numbers a bit to see how different percentages and even different attributes to <FRAMESET> changes how the page displays. Enter Listing as shown below in your text editor.

---

smpframe.html Simple Frame Document

```
<HTML>
<HEAD>
<TITLE>Frame Example</TITLE>
</HEAD>
<FRAMESET ROWS="25%, 75%">
<FRAME SRC="menu.html">
<FRAME SRC="help.html">
</FRAMESET>
</HTML>
```

While you're at it, you also need to create some files to put in those frames. If you have some HTML documents hanging around, you can rename menu.html and help.html to any HTML file you'd like to load. For this example, any HTML document names will work (see fig. below).



Loading separate HTML documents into a frame-based page.

If you'd like to experiment further, try changing the <FRAMESET> tag in Listing 20.1 to the following:
<FRAMESET COLS="25%,75%">
Or, change the percentages to see how that affects your layout.

Attributes for <FRAME>

Aside from SRC, the <FRAME> tag can accept the attributes NAME, MARGINWIDTH, MARGINHEIGHT, SCROLLING, and NORESIZE. All of these but NAME are appearance-oriented. Let's deal with them first and come back to NAME in a moment.
MARGINWIDTH and MARGINHEIGHT are used to control the right/left margins and the top/bottom margins of the text and graphics within a frame, respectively. Each takes a numerical value in pixels. For example:

<FRAME SRC="text.html" MARGINWIDTH="5" MARGINHEIGHT="5">

This creates a five-pixel border between the contents of text.html and the frame edges.
SCROLLING can accept the values yes, no, and auto and is used to determine whether or not scroll bars will appear in the frame window. The default value is auto, and this is probably the best to use in most cases. Since users have all different screen resolutions and available browser window space, even short documents will sometimes need to be scrolled.
The NORESIZE attribute doesn't require a value assignment, and is used to keep the user from resizing a frame window. (Frame windows can be resized by dragging the frame with the mouse in the viewer window.)

An example of SCROLLING and NORESIZE would be:

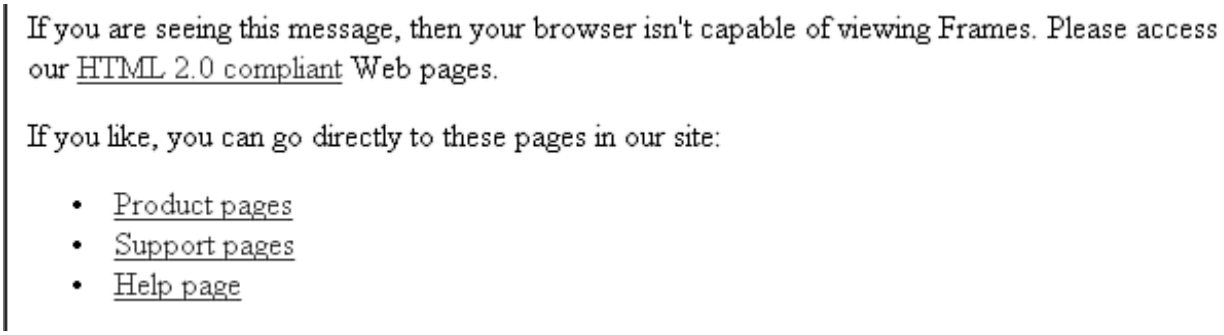
```
<FRAME SRC="text.html" SCROLLING="yes" NORESIZE>
```

The <NOFRAMES> Tag

This container tag is used to contain HTML markup intended for browsers that do not support the frames specification. Frames-capable browsers ignore text and HTML tags inside the <NOFRAMES> container. All others should generally ignore the other frames tags (which they won't recognize), but display the text in between the <NOFRAMES> tags. The following is an example:

```
<FRAMESET ROWS="25%,75%">
<FRAME SRC="menu.html">
<FRAME SRC="index.html">
<NOFRAMES>
<P>This page requires a Frames capable browser to view. If you'd prefer,
you can access our <a href="2_index.html">HTML 3.0 compliant pages</a>
to view this information without the frames interface.</P>
</NOFRAMES>
</FRAMESET>
```

Experiment with different values for the <FRAME> attributes and see what makes a big difference in terms of margins, scrolling, and resizing. Also, if you have access to a browser that isn't frames-capable, load the page and see how the <NOFRAMES> markup looks (see fig. below).



The <NOFRAME> HTML message

Targeting Frame Windows

So far, you've seen that frame windows offer you the ability to load URLs independent of one another, so that you can present two (or more) different HTML pages in the same browser window. But what good is this to you? In many cases, it may be useful to offer multiple documents at once.

For instance, with what you know now, you could use frames to add a button bar image map to the top of every HTML page you create. But that would get tedious-each link would have to point to a new frame document that would then load the button bar and the next document.

But what if you could just leave the button bar in the top frame window and load a new document in the other window? You can do just that, if you know a little about *targeting*.

The NAME Attribute

First, you need to name your frame windows-at least, you have to name the windows you might want to change. This is accomplished with the NAME attribute to the <FRAME> tag, which takes the following format:

```
<FRAME SRC="original URL" NAME="window_name">
```

This shouldn't look too foreign to you, as it's a little like the way that the NAME attribute works for <A NAME> links. Once the frame window has a distinct name, you can access it directly from other frame windows. An example of this is the following:

```
<FRAME SRC="index.html" NAME="main_viewer">
```

Although you can pretty much name your frame window anything you want, there is one restriction: you can't start the name with an underscore character (_). If you do, the name will be ignored. But, there's a good reason for that.
The underscore is used to signal a number of "magic" target names. You'll get to those after you learn how to target regular browser windows.

Targeting Frame Windows

With your frame successfully named, you're ready to target the name with a hypertext link. This is accomplished with the TARGET attribute to a typical <A> anchor tag. It follows this format:
<A HREF="*new_URL*" TARGET="*window_name*">*link text*</A>
The new_URL is the new document that you want to have appear in the frame window. The window_name is the same name that you used to name the frame windows with the NAME attribute to the <FRAME> tag. An example would be the following:
<A HREF="products.html" TARGET="main_viewer">View Products</A>

Advanced Targeting

Other link-related tags have been updated to accept a TARGET attribute along with the anchor tag. For instance, client-side image maps have their AREA tag updated to accept a target window, so that an area defined to access a certain link loads that page into the target. This is the format:
<AREA SHAPE="shape" COORDS="numbers" HREF="URL" TARGET="window_name">
Likewise, you can add a TARGET attribute to the <FORM> tag. Remember that it's the <FORM> container tag that tells the browser the URL for the script that is required to process the form data. In that same tag, you can also specify a target window for the results that are received from the server. If you want users to be able to use your form repeatedly (to allow them to generate different results), you can leave both the form and the results in separate frames. This attribute takes the following format:
<FORM ACTION="Script_URL" TARGET="window_name">

<BASE> Targets

The last example you went through (Listing 20.4) would have been a great candidate for this one type of target. What if you want the majority of your links to point to a particular frame window? In the early example, you created a file called control2.html that had nothing but hypertext links. Each one of those links required a TARGET attribute that pointed to the big_window. You could have made that easier with a <BASE> tag in the head of your document. Use this format:
<HEAD>
<BASE TARGET="window_name">
</HEAD>
A good example of this for the previous example would be:
<BASE TARGET="big_window">
You don't have to specify the target window in each individual anchor in an HTML document that has this tag in its head. Now all links will default to the target defined in the <BASE> tag.

Note: If you do use TARGET attributes, they will override the <BASE> tag for that particular link.

"Magic" Targets

Here's why you can't name frame windows with something that starts with an underscore. The "magic" target names all start with an underscore, which signals to the browser that they should treat this link extra special. The following are some examples:

- **TARGET="_blank"**-The URL specified in this link will always be loaded in a new blank browser window.
- **TARGET="_self"**-This is used for overriding a <BASE> tag, and forcing a link to load in the same window that it's clicked in.
- **TARGET="_parent"**-This causes the document to load in the current window's parent-generally, the frame window immediately preceding in the <FRAMESET> definition. If no parent exists, it acts like "_self".
- **TARGET="_top"**-The document is loaded in the topmost frame of the current browser window.

Basically, these magic targets are designed to let you break out of the current <FRAMESET> structure in some way. Experiment with them to see how you can move around to different windows.

The Elegant Frame Interface

Let's use three different frame windows. One will hold a client-side graphic for your site, one will offer a "table of contents" page, and one will display the actual information for your site.
You'll need a fixed window for the client-side map and two dynamic windows. One will load the parts of the table of contents, and one will load the information pages. Use Listing below for the client-side map.

---

control3.html HTML for a Client-Side Map

```
<HTML>
<HEAD>
<TITLE>Control Map</TITLE>
<BASE TARGET="toc_window">
</HEAD>
<BODY>
<DIV ALIGN="CENTER">
<IMG SRC="control.gif" USEMAP="#control">
</DIV>
<MAP NAME="control">
<AREA SHAPE="rect" COORDS="0,0,61,22" HREF="index_toc.html">
<AREA SHAPE="rect" COORDS="62,0,146,22" HREF="prod_toc.html">
<AREA SHAPE="rect" COORDS="146,0,222,22" HREF="serv_toc.html">
<AREA SHAPE="rect" COORDS="222,0,296,22" HREF="supp_toc.html">
<AREA SHAPE="rect" COORDS="296,0,359,23" HREF="about_toc.html">
</MAP>
</BODY>
</HTML>
```

Notice that this graphic isn't designed to change the main window. It's going to change the documents that show up in your table of contents window. Each of the TOC documents should be a simple HTML list of links that shows the branch that the users have traveled down. For instance, let's create the document serv_toc.html (see Listing below).

---

serv_toc.html A Sample TOC Document

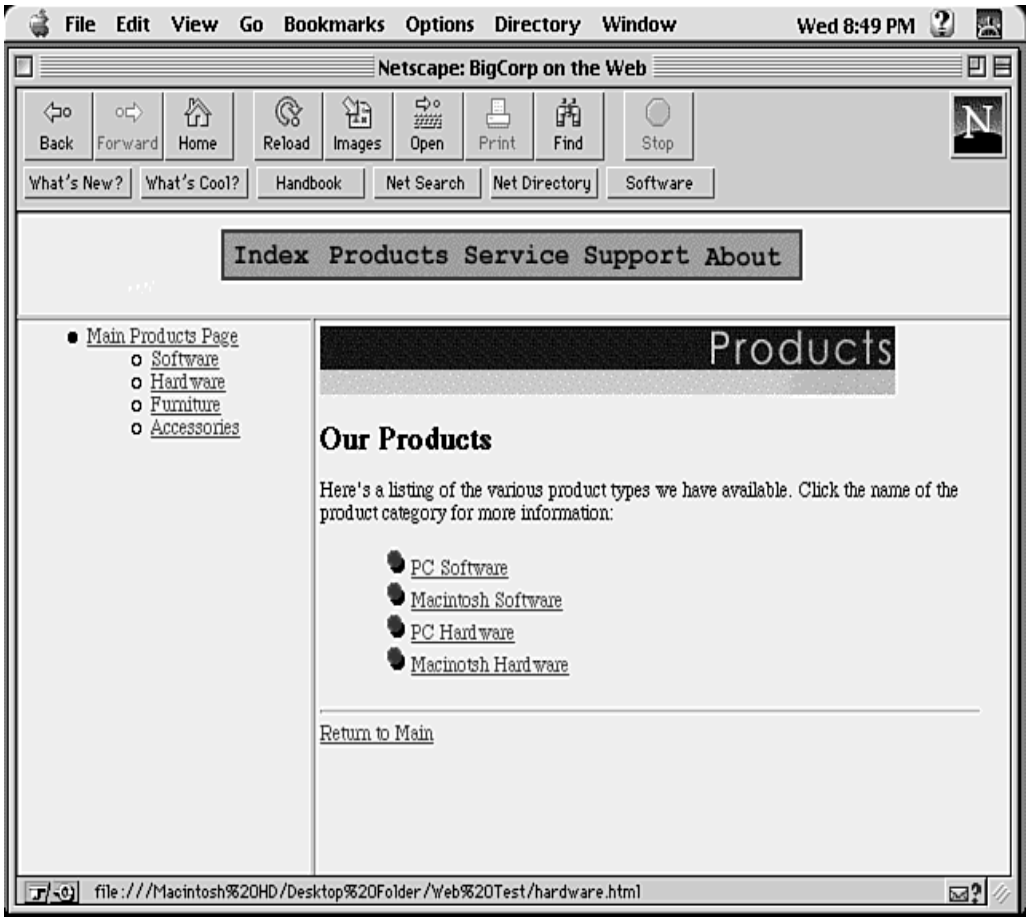
```
<HTML>
<HEAD>
<TITLE>Service Contents</TITLE>
<BASE TARGET="main_window">
</HEAD>
<BODY>
<UL>
  <LI><A HREF="products.html">Main Products Page</A>
  <UL>
     <LI><A HREF="software.html">Software</A>
     <LI><A HREF="hardware.html">Hardware</A>
     <LI><A HREF="furniture.html">Furniture</A>
     <LI><A HREF="access.html">Accessories</A>
  </UL>
</UL>
</BODY>
</HTML>
```

See what we are getting at here? The image map will change these controls, and then these links will change the main frame window. It gives a nice, easy interface to the whole site. Now let's create the frames page which we'll call main.html (see Listing below).

main.html The Main Frames Interface

```
<HTML>
<HEAD>
<TITLE>BigCorp on the Web</TITLE>
</HEAD>
<FRAMESET ROWS="15%,85%">
<FRAME SRC="controls.html" SCROLLING="no" NORESIZE>
  <FRAMESET COLS="30%,70%">
  <FRAME SRC="index_toc.html" NAME="toc_window" MARGINWIDTH="3"
   MARGINHEIGHT="3">
  <FRAME SRC="index.html" NAME="main_window" MARGINWIDTH="3"
   MARGINHEIGHT="3">
  </FRAMESET>
</FRAMESET>
</HTML>
```

This one doesn't take up much space. Here's what you've done. The first <FRAMESET> created two rows-one for image map and one for the rest of the page. In that bottom row, you used another <FRAMESET> to create two columns within that row. The smaller is for the table of contents pages. In the other frame window, you put all of the main HTML documents. Look at figure below to see how the whole thing looks in Netscape.



A three way interface is attractive, but it can be tough to manage.

Note : For this example to work well, you may need to create some other files like index_toc.html, or rename files to work in your main window.

CHAPTER 12: Introduction to DHTML

Obejctives:

- Difference between HTML and DHTML
- Explain features of DHTML
- List Dynamic content in DHTML

Web authors today face significant challenges when making their Web pages interactive. The static nature of HTML pages limits their creative choices, and interactive components can be difficult to build and reuse. In addition, using proprietary extensions means authoring browser-specific Web pages.
Microsoft® Dynamic HTML technology helps to remove these barriers for content providers and offers users more engaging and interactive Web pages. Dynamic HTML provides authors with enhanced creative control so they can manipulate any page element at any time. Dynamic HTML is also the easiest way to make Web pages interactive, using open, standards-based technologies. Microsoft is working with the World Wide Web Consortium (W3C) to help ensure interoperability and support for users on multiple systems with different browsers.
Dynamic HTML builds upon existing HTML standards to expand the possibilities of Web page design, presentation, and interaction. Ultimately, mastering DHTML will allow you to build Web-based applications, rather than mere portraits of data. Because DHTML is essentially an "added value" technology, you should be rather familiar with basic Web page design using traditional HTML specifications. Experience with JavaScript programming is also necessary to employ the potential of DHTML.

Total Creative Control for an Immediate User Experience

Making simple updates, such as changing the color of text after a Web page loads, traditionally has meant reloading the page. These limitations have slowed the user experience and have impeded interactivity on the Web.
Microsoft's Dynamic HTML takes interactivity to the next level. Pages authored with Dynamic HTML come alive, with every element in the page being truly dynamic. Whether the page has loaded already, content providers can change any element of the page -- text or graphics -- without a round-trip to the server. This increased control and flexibility result in more compelling sites. For users, the Web experience becomes more responsive.

Key features of DHTML

Key features of Dynamic HTML include these:

- **Document Object Model (DOM).** Dynamic HTML provides a comprehensive object model for HTML. This model exposes all page elements as objects. Changing their attributes or applying methods to them at any time can easily manipulate these objects. Dynamic HTML also provides full support for keyboard and mouse events on all page elements. Support for the Document Object Model enables the following:

- **Dynamic content.** Text or graphics can be added, deleted, or modified on the fly. For example, a Web page can display an updated headline, without refreshing the page. The text surrounding the headline will reflow automatically.

- **Dynamic styles.** Internet Explorer 4.0 fully supports Cascading Style Sheets (CSS). As such, any CSS attribute, including color and font, can be updated without a server round-trip. For instance, text can change color or size when a mouse pointer passes over it.

Multimedia filters and transition effects can be applied to HTML elements simply by adding the filter CSS attribute.

- **Absolute positioning.** CSS positioning coordinates for existing page content can be updated at any time to create animated effects, without reloading the page.

- **Data Binding.** Data-driven application front ends can be built that present, manipulate (e.g., sort, filter), and update data on the client without numerous round-trips to the server.

- **Scriptlets.** A scriptlet is a Web page, authored with Dynamic HTML, which content providers can use as a component in their Web applications. With scriptlets, content providers can author content once, then easily reuse the content in other Web pages or applications.

The Easiest Way to Make Pages Interactive

Adding dynamic behavior to Web pages formerly required writing complex applets or controls, and incorporating them in Web pages using scripts. Although these components perform useful tasks, many authors found them hard to develop compared with scripts or HTML.
Microsoft Dynamic HTML was designed so that Web builders can use the scripting languages they know today -- such as JavaScript and the Microsoft Visual Basic® programming system, Visual Basic Scripting Edition (VBScript) -- to make their Web pages interactive. Developers can also write full-featured Web applications with controls and applets that use Dynamic HTML.
Web builders can easily reuse Dynamic HTML-based content through support for Scriptlets in Microsoft Internet Explorer 4.0, using just HTML and script.
Dynamic HTML can be authored today using tools from third parties, including Bluestone Inc., ExperTelligence Inc., Pictorius Inc., and SoftQuad Inc., as well as Microsoft FrontPage 98.

Dynamic Content with a Broad Reach

A standards-based approach enables content providers to create interactive pages that reach a broad audience. At a base level, content providers can take advantage of standards-based features, such as CSS and CSS Positioning that are implemented in today's popular browsers. This ensures that a majority of users, regardless of the browser they are using, have the same Web experience.
Web authors can also create a single set of pages authored with Dynamic HTML for all users. Users with any browser supporting the Document Object Model, such as Microsoft Internet Explorer 4.0, can fully interact with these pages. Other browser users, such as those running Netscape Navigator, could view much of this content statically, because Dynamic HTML uses standard HTML tags to render content. For instance, an interactive table of contents could expand and contract using Microsoft Internet Explorer 4.0. In a non-dynamic browser, the table would display in a fully expanded state.
In some cases, content providers may decide to author separate sets of pages and take advantage of specific browser capabilities to ensure a truly engaging, rich experience. Dynamic HTML will be featured in all versions of Microsoft Internet Explorer 4.0, including versions for the Windows® operating system and for the Macintosh and UNIX platforms. Microsoft's implementation of Dynamic HTML will also be available free of charge as a component for third-party use.

CHAPTER 13: DHTML and StyleSheet

Obejctives:

- Use Style Sheet
- Type of StyleSheet
- Explain formatting with styleSheet

Clearly, positioning our block of content depends upon the specified STYLE properties. Several documents on Web contain references to the positioning properties available for style sheets -- two worth reviewing are Microsoft's CSS Attributes Reference (under "Positioning Properties") and Netscape's Defining Positioned Blocks of HTML Content.
We've included a handy chart summarizing the common STYLE properties, which we may want to use when using the CSS syntax to position blocks of content.
Employing the various STsLE properties gives us powerful control over the position and look of the blocks of content on your page. Conceptually, then, when we think in DHTML we think of a page as made up of one or more blocks. Of course, this is not immediately evident to the viewer, who essentially sees a flat Web page, without realizing that several smaller blocks of content are positioned here and there to create the overall effect.
Still, the question begs: how is this *dynamic*?
Consider the fact that, now that the blocks have been put into place, you can -- at any time -- change their properties. Position, background, clipping region, z-index -- it's all plastic, with the help of JavaScript, and that is the reason to be excited!

Common STYLE Properties

position	Specifies how the block should be positioned on the page with respect to other page elements. Possible values: *"position:absolute;"* Block is positioned absolutely within the browser window, relative to <BODY> block. *"position:relative*;" Block is positioned relative to its parent block, if any, or else normal flow of page. *"position:static*;" Block is positioned according to standard HTML layout rules -- follows flow of document. (**MSIE 4+ only; Netscape 5?)**
width	Specifies the width at which the block's contents should wrap. You may specify width in measured units (*50px*), as a percentage of the parent block's width (*50%*), or *auto* which wraps the block according to its parent's width. Examples: *"width:50px;"* or *"width:50%;"*
height	Specifies the height of the block, measured in units (*50px*), percentage of the parent block's height (*50%*), or *auto*. Note that the height of the block will be forced to the minimum necessary to display its contents; however, you can make the block taller than necessary. Examples: *"height:50px;"* or *"height:50%;"*
left	Specifies the offset of the left edge of the block in accordance with the *position* attribute. Positive measures (*5px*) are offset towards the right while negative measures (*-5px*) are offset towards the left. Examples: *"left:5px;"* or *"left:-5px;"*
top	Specified the offset from the top edge of the block in accordance with the

	position attribute. Positive measures (*5px*) are offset towards the bottom of the page while negative measures (*-5px*) are offset towards the top of the page. Examples: *"top:10px;"* or *"top:-10px;"*
clip	Specifies a rectangular portion of the block which is visible. Typically, you use the *clip* property if you want to show only a portion of the block, therefore hiding a portion. Syntax: **MSIE:** clip:rect(*top right bottom left*) Example: *"clip:rect(0px 30px 50px 0px);"* **Netscape**: clip:rect(*left,top,right,bottom)* Example: *"clip:rect(0,0,30,50);"* Notice that the syntaxes vary between browsers, both in the need to specify measurement units (MSIE) and the order of the parameters.
visibility	Specifies whether a block should be visible. If not visible, the block will not appear on the page, although you can make it visible later using JavaScript. The possible values for this property again vary between browsers. **MSIE:** *"visibility:inherit;"* Block inherits the visibility property of its parent. *"visibility:visible;"* Block is visible. *"visibility:hidden;"* Block is invisible. **Netscape:** *"visibility:inherit;"* Block inherits the visibility property of its parent. *"visibility:show;"* Block is visible. *"visibility:hide;"* Block is invisible.
z-index	Specifies the "stacking order" of blocks, should they happen to overlap other positioned blocks. A block is assigned a z-index, which is any integer. When blocks overlap, that which has the greater positive z-index appears above a block with a lower z-index. Blocks with an equal z-index value are stacked according to the order in which they appear in the source code (bottom-to-top: first block defined appears on bottom, last block defined appears on top). *Example: "z-index:3;"*
background-color	Specifies the background color for the block. Examples: *"background-color:green;"* or *"background-color:FF8F00;"*
background-image	Specifies a background image for the block. Example: *"background-image:url('images/tilewood.jpg');"*

Putting the D in DHTML

Once you've positioned a content block on the page using <DIV> tags in your HTML, we can use JavaScript to modify its properties. This has many possible consequences: we can move the entire block up, down, left or right. We can change its background color, or change the clipping region, causing more or less of the block to be visible. Speaking of visibility, We can even hide or show the entire block in an instant via the visibility property.

How, then, does JavaScript access the style properties of the content block? The answer is twofold:

1. Assuming that you are familiar with JavaScript, you know that data structures are represented as *objects*, and each object has a set of *properties*. JavaScript statements can read or write to the properties of an object.
2. Content blocks contained in <DIV> tags are exposed as objects to JavaScript by the Document Object Model. This means that, following the object and property specifications defined by the DOM, JavaScript can access the style sheet properties of the content block.

```
<DIV ID="mailblock"
        STYLE="position:absolute;
        width:auto;
        height:auto;
        left:400px; top:50px;
        background-color:white">
<A HREF="mailto:me@myhouse.com">Mail me!</A><BR>
<IMG    SRC     = "mailbox.gif"
        width   = 30
        height  = 15
        alt     = "Mailbox"
        >
</DIV>
```

Because Netscape and Microsoft do not currently share compatible DOMs, the above block is exposed to different objects in each browser. For now, we'll consider each case separately -- a future article will look at the issue of patching over this problem in "cross browser DHTML."
In Netscape's DOM, each <DIV> block takes the form of a Layer Object. In Microsoft, each block is exposed as a DIV object, which in turn possesses a STYLE object, whose properties reflect the familiar style attribute properties. This might sound confusing, but it's all a matter of syntax. Let's first consider how you would construct a JavaScript reference to *mailblock*. You use the identifier specified in the ID attribute:

Microsoft:
document.all.mailblock

Netscape:
document.layers["mailblock"]
Now, let's consider the background color property for *mailblock*. The CSS property for the background color was "background-color" as specified in our <DIV> tag. When reflected via the DOM, however, this property takes on a different name between browsers:

Microsoft:
document.all.mailblock.style.backgroundColor

Netscape:
document.layers["mailblock"].bgColor
Knowing that, we can easily create a JavaScript statement, which would change the background color of *mailblock* to green.

Microsoft:
document.all.mailblock.style.backgroundColor="green";

Netscape:
document.layers["mailblock"].bgColor="green";

Similarly, we can modify the *top* and *left* style properties which will move the block on-screen. Thus, once we know the correct DOM property for the style sheet *left* property, we can move *mailblock* directly to the left edge of the page:

Microsoft:

```
document.all.mailblock.style.left="0px";
OR
document.all.mailblock.style.pixelLeft=0;
```

Netscape:

```
document.layers["mailblock"].left=0;
```

Armed with the understanding of positioned content blocks, and the ability to manipulate their characteristics via JavaScript, you basically need only to rely on your imagination (although JavaScript skill can certainly help). Consider some nifty uses of dynamic positioning:

- **Animation:** images contained in a positioned block can be moved around the page following a certain path.
- **Drop-down menus:** modifying clipping regions allows you to show or hide portions of a content block. To create a drop down menu, you would initially show only the top strip of the block containing the menu choices. When the user triggers the menu to drop, you chance the clipping region to display the vertical length of the menu chosen. We'll look at triggers from user events later in this article.
- **Content swapping:** by positioning multiple blocks of content at the same spot on the page, yet keeping all but one block invisible, you can quickly swap a new block of content into place by changing their visibility properties. Alternatively, by keeping two overlapped blocks visible, you can modify their clipping regions in such a pattern as to produce a "transition" effect.

If all this sounds daunting, the real work is in the JavaScript code. The properties above make it clear how you can modify content blocks, but it is the JavaScript coding which performs these modifications, sometimes requiring complex loops to yield complex effects

Dynamic Styles

As we've seen, positioning a content block is a form of style sheet. However, there are other uses for style sheets aside from placing blocks on the page. Style sheets (Cascading Style Sheets or CSS) can be used to define any set of styles, which may apply to certain HTML elements or block of HTML. Like JavaScript, Cascading Style Sheets are their own in-depth web-authoring topic. Because our focus is on DHTML, insofar as it uses and combines web-authoring technologies, this isn't the place for a detailed tutorial on creating styles or style sheets.

In brief, there are several ways to apply styles using style sheets. You can define styles, which apply globally to all HTML elements of a certain type. For instance, you might create a style sheet, which specifies that all <H3> elements be bold (*font-weight*) and red (*color*).

You may also define style "classes," which are predefined styles applied to an element on-demand. For example, you might specify that the style class "redbold" contains the styles bold (*font-weight*) and red (*color*). This *redbold* class would only apply to elements in which you demand it;

e.g. <BLOCKQUOTE CLASS="redbold">.

Lastly, as you've already seen, you can define and apply styles on the fly, using the STYLE attribute for an element tag;

e.g. <H4 STYLE="color:red; font-weight:bold;">. These on-the-fly styles apply only to the tag in which they are specified.

Both Microsoft and Netscape support CSS as described above, although Microsoft supports additional style properties, which do not exist under Netscape. The critical difference between the browsers, however, is the degree to which their CSS support is *dynamic*.

Quasi-Dynamic HTML

The style sheet determines how, aesthetically, the document is rendered on the screen. Once rendered, though, can these styles be modified on-screen? This is where the two browsers diverge markedly.

Within Microsoft's Internet Explorer, you can, via JavaScript, modify many of the properties of a style sheet or an element's individual style. These changes will be immediately reflected on-screen -- so, if you changed the *redbold* class's *color* property to blue, any on-screen text using the *redbold* class will suddenly change to blue. Similarly, you can change any style property for those elements, which possess inline ("on-the-fly") styles.

Netscape, on the other hand, is not so accommodating. While there is an interface to style sheets via JavaScript (albeit very different from the Microsoft implementation), you can only use JavaScript to define styles *before* they are rendered. By and large, this would be used as a slightly more flexible replacement for defining your styles rather than using standard CSS syntax. On the other hand, using standard CSS syntax would maintain compatibility between browsers. For this reason, it could be argued that Netscape's support for CSS via JavaScript is not terribly useful, and not at all dynamic, since changes applied to styles once the page has been rendered do not appear on-screen.

Stuck in the middle are we -- but because Microsoft's support for style modifications is far more robust, we will focus here on dynamic styles in MSIE. Hopefully the next version of Netscape will catch up in this regard.

Inline Styles

Speaking non-strictly, we can say that an "inline style" is a style, which applies only to an individual element, rather than all elements of a certain type or class. Typically, an inline style is defined using the STYLE attribute for the element's tag; e.g. <H2 STYLE="color:blue;">.

Within MSIE, then, you can use JavaScript to modify an inline style at any time. The modification will take effect on-screen immediately. You do this using the Style Object, which we looked at earlier with Dynamic Positioning. The Style Object in MSIE's DOM supports every property, which CSS supports for styles. However, the property names differ slightly between CSS syntax and DOM syntax. Fortunately, the naming differences follow a rule. Consider the CSS property *font-weight*. The corresponding Style Object property is named *fontWeight*. Similarly, the CSS property *text-align* maps to the Style Object property *textAlign*. Of course, *color* retains the same name in both syntaxes. As you can see, then, CSS property names simply lose their hyphen and capitalize the first letter following the hyphen when used in JavaScript syntax.

So, then, confusing though that might sound, let's look at an example which illustrates just how easy changing styles can be. Imagine that you have the following HTML element:

```
<P ID="selectrule" ALIGN="left" STYLE="color:blue; font-weight:normal;">
        Please select one item below.
</P>
```

Now, some series of events occur and the user fails to follow instructions. So, you'd like to emphasize the exhortation:

```
document.all.selectrule.style.color="red";
document.all.selectrule.style.fontWeight="bold";
```

When the above statements are executed, the text associated with the element named *selectrule* will become red and bold. Of course, we can set the element back to its original state by setting *color* back to "blue" and *fontWeight* back to "normal".
Notice that our JavaScript statements are modifying the styles directly associated with the particular element. You can always do this, even if the element's style wasn't defined inline, but was inherited from a class or global style sheet. No matter how the element gained its original style, whether inline or by style sheet, directly modifying the style in the above manner only applies to the individual element. For this reason, this form of JavaScript dynamic styles is always an inline style.

Which Style Sheet Properties Map Onto Which DOM Properties For Each Browser

CSS Property	Netscape Layer Object Property	Microsoft Style Object Property
position	none	*position* Note: read-only, cannot modify via script
width	none	*pixelWidth*
height	none	*pixelHeight*
left	*left* Note: Accepts either an integer (assumed pixel units) or a percentage string. Examples: *left=10* or *left="20%"*	*left* or *pixelLeft* Note: *pixelLeft* accepts a string which specifies measurement units; e.g. *pixelLeft="10px"* whereas *left* accepts an integer and assumes pixel units; e.g. *left=10*.
top	*top* Note: Accepts either an integer (assumed pixel units) or a percentage string. Examples: *top=10* or *top="20%"*	*top* or *pixelTop* Note: *pixelTop* accepts a string which specifies measurement units; e.g. *pixelTop="10px"* whereas *top* accepts an integer and assumes pixel units; e.g. *top=10*.
clip	*clip.top* *clip.left* *clip.right* *clip.bottom* *clip.width* *clip.height* Each dimension of the clipping coordinates is its own property, as seen here. Changing any property immediately causes the on-screen clip to change.	*"rect(top right bottom left)"* To change the clip in MSIE you must redefine all clipping coordinates. The syntax can be a bit confusing. Example: document.all.blockname.style.clip ="rect(0 25 25 0)"
visibility	*visibility* May contain any of "inherit", "show", or "hide"	*visibility* May contain any of "inherit", "visible", or "hidden"
z-index	*zIndex*	*zIndex*

	Any non-negative integer.	MSIE allows negative integers, but you may as well stick with positive for the sake of Netscape.
background-color	*bgColor* Accepts string containing pre-defined color name or hex RGB triplet. Examples: document.layers["blockname"].bgColor ="black" document.layers["blockname"].bgColor ="#000000"	*backgroundColor* Accepts string containing pre-defined color name or hex RGB triplet. Examples: document.all.blockname.style.backgroundColor ="black" document.all.blockname.style.backgroundColor ="#000000"
background-image	*background.src* Example: document.layers["blockname"].background.src ="images/tile.jpg"	*backgroundImage* Example: document.all.blockname.backgroundImage ="images/tile.jpg"

The Event Connection

Quite frequently, you want some type of trigger to cause your DHTML to kick in. Whether repositioning blocks or changing style properties, some *event* usually causes these changes.
Many different types of event can occur on a Web page, most of them caused by the user. He or she might click on a button (*click* event), might move the mouse over an element (*mouseover* event), or move the mouse off of an element (*mouseout* event). The user may submit a form (*submit* event) or resize the browser window (*resize* event). Additionally, some events occur without direct user intervention -- the page may finish loading (*load* event), for example.
Talking about events in this article is tricky because, like our previous discussions, working with events crosses a couple of authoring technologies. The Document Object Model also defines types of events and their characteristics, such as to which elements they apply. More exactly, though, you manipulate and control events using JavaScript syntax. Once again, then, the handling of events is not exactly the same between Microsoft and Netscape browsers, at least until a standardized DOM comes along.
Managing events can be relatively simple or quite complex, depending on the ambitions of your project. Since this is an introduction, we'll focus mainly on event basics. Fortunately, basic events are handled most similarly between browsers.

Events occur on a Web page whether you choose to act on them or not. When the user moves the mouse over an element, a *mouseover* event occurs. If you would like to leverage this event as a trigger for some dynamic action, you must construct an *event handler*.
An event handler is created as an attribute for the tag which defines the element at which you wish to catch the event. Event handler attribute named follow the syntax *onEventname*, and they accept JavaScript statements as their action. For example, the following tag creates a hyperlink with a *mouseover* event handler specified.

```
<A HREF="page.html"
  onMouseOver="changeStatus('Read this page');
  return true">
Click here
</A>
```

The *onMouseOver* event handler catches a *mouseover* event. When this event occurs at this element, a JavaScript function is called named *changeStatus()*. This is a fictional function, you can imagine that it might exist to change the message on the browser's status bar. Any JavaScript statement is allowed in an event handler, so we could also execute direct statement rather than call a function. For example, suppose that a *mouseover* for this element in MSIE should modify a style sheet's color property:

```
<A      HREF="page.html"
        onMouseOver="document.styleSheets[0].rules[0].style.color='blue'"
        >
        Click here</A>
```

The above example assumes MSIE, since style sheet modifications aren't supported in Netscape. We also assume that a style sheet exists in this document! But, hey, it's just an example. Once again, a convenient table will help summarize the common event and their event handler names.

Event	**Event Handler Syntax**	**Description**
click	OnClick	User clicks (left) mouse button on an element.
submit	OnSubmit	User submits a form, this event fires before the form submission is processed.
reset	OnReset	User resets a form.
focus	OnFocus	User brings focus to an element either via mouse click or tabbing.
blur	OnBlur	User loses focuses from an element by clicking away or tabbing away.
mouseover	OnMouseOver	User moves mouse over an element.
mouseout	OnMouseOut	User moves mouse off of an element.
mousemove	OnMouseMove	User moves mouse.
change	OnChange	User changes value in a text, textarea, or select field.
select	OnSelect	User selects (highlights) a portion of text in a text or textarea field.
resize	OnResize	User resizes browser window or frame.
move	OnMove	User moves browser window or frame.
load	OnLoad	Page completes loading.
unload	OnUnload	User exits page (by navigating to a new page or quitting browser).
error	OnError	An error occurs loading an image or document.
abort	OnAbort	User stops an image loading using the stop button.

The above table is a quick reference guide. There are some important caveats to keep in mind. For one, this is not a comprehensive list of all events, although these are by far the most commonly used. Both browser support additional events for detecting key presses and other mouse actions, while MSIE supports additional events above and beyond Netscape's. Secondly, you must keep in mind that not every event is applicable to every element. This also varies between browsers. For example, within Netscape a *mouseover* event only applies to a hyperlink

<A>, area <AREA> or layer <LAYER>. Yet, within MSIE, you can apply a *mouseover* event to almost any element, including images <IMG>, and paragraphs <P>.

In general, the above rule holds between browsers: Netscape restricts each event to certain limited elements, while MSIE allows most events to be handled at most elements. Mastering event handling is most certainly a topic unto itself. At the surface, handling basic events is simple, as you've seen. The classic "rollover effect" is a perfect example of a simple event used in DHTML. A rollover occurs when the user moves the mouse over an element; while the mouse hovers over the element, its appearance changes to reflect that it is "active". When the mouse moves off the element is reverts to a more subdued state. Rollovers commonly use changes in either image or style. Below is a basic MSIE style-based rollover, which makes the "active" text red and bold, and returns to normal blue when inactive. Remember that such dynamic styles don't work so easily under Netscape, although you could certainly re-imagine this example for Netscape, perhaps by swapping an image for the rollover effect; or swapping a layer.

MSIE Dynamic Style Rollover Example

```
<A HREF="somepage.html" TARGET="mainframe"
STYLE="color:blue; font-weight:normal; font-family:Arial;"
onMouseOver="this.style.color='red';this.style.fontWeight='bold'"
onMouseOut="this.style.color='blue';this.style.fontWeight='normal'">
Today's special</A>
```

Cascading Style Sheets

Cascading Style Sheets are design templates that provide augmented control over presentation and layout of HTML elements. They allow you to separate the way you design information from the HTML content.

Benefits of Cascading Style Sheets

Using style sheets, you can create Web pages with minimal graphics, and therefore much smaller downloads. Style sheets also provide you with a higher level of typographic control, and they enable you to make changes to an entire site through the use of linked style sheets.

10 CSS features

Here are 10 of the many features CSS technology brings to Web design.

- Fonts
- Colors and Backgrounds
- Text Decoration
- Margins
- Text Indent
- Line Height
- Letter Spacing
- Initial Caps achieved using float
- Clip
- Position and *Z*-index
- Filters

Dynamic Layering Techniques

The ability to create layers in HTML allows developers to place elements on top of each other to create intriguing effects with text and graphics. However, some confusion may arise from the term "layer." While it is true that there is an actual HTML element from Netscape, the **<LAYER>** tag, there are several ways to create layers for both Internet Explorer and Netscape Navigator. In this article we will learn the difference between IE and NN layers as well as how to develop a layers-enhanced site for both browsers.

Since both IE and Netscape view layers differently, it's important that you recognize these differences so that you can work around them. We cannot design an effective and useful site without an understanding of the technical limitations of the user's applications or the limitations of the technology. First, let's look at what layers are, and see how they can help improve your Web site.

Layering Documents

Most graphic designers are familiar with layering; they are an integral part of the apps used by most designers, including Photoshop, Illustrator, Freehand, Fireworks and Image Ready. Layers allow the designer to create complex graphics that integrate images or illustrations, and text.
For instance, a designer may begin with a photo of a house, then add some trees or shrubs in front on a top layer, and maybe some text that goes over the house and trees on another. It is through the use of layering -- determining an item's order in a stack of other items -- that makes this technique work. Now Web developers can enjoy a similar presentation control in their HTML documents.
The basic technology for layering comes to us from the W3C via Cascading Style Sheets. A single property that, when assigned to an element, determines an element's stacking order; its layered position. The property is called z-index -- it gives us the ability to define the third-dimension on the Web page. Listing below shows the syntax for the z-index property as it would be assigned in a style sheet and inline.

The z-index property

```
<STYLE TYPE="text/css">
.layer1 { z-index:1; visibility: visible; }
.layer2 { z-index:2; visibility: hidden; }
</STYLE>
...
<BODY>
<IMG CLASS="layer1" SRC="myimage.gif">
<P STYLE="z-index:3; ">Here is some text...</P>
<IMG CLASS="layer2" SRC="myimage.gif">
</BODY>
```

Rather than go into CSS-positioning, we will assume the images are the same height and width, and that they occupy the same space on the document. The layering of these elements is determined according to their z-index properties. In the example above the first image could be some sort of background; the second image could be a graphic with a large transparent area; and the final element, the paragraph, contains text that is layered on top of the images.

Creating a Layer

This section contains a discussion about some of the properties and elements used for creating layers in HTML. The z-index property is universal -- both browsers treat it the same. But other aspects of layering can vary greatly from browser to browser.

Consider the <DIV> tag as an example. It provides an excellent way to define static layers in both browsers. However, the <DIV> tag can only be used in IE for dynamic layering. Then there's the <LAYER> tag that was first introduced by Netscape. It does the same thing as the <DIV> tag, but it only works in Netscape browsers.

z-index

By now, we're probably familiar with the use of x and y coordinates as they relate to the horizontal and vertical placement of elements on a page. And as the Web has evolved, it is only natural that we would see the emergence of a z element. z is the order in which an element exists within a stack of other elements. Actually, without even realizing it, we use the z-index whenever we design a page. The body is an element unto itself, the z-index position of the body is 0 and that cannot be changed; so all other elements should have a z-index value of 1 by default.
If you want to layer several elements, you will need to specify which element goes where, exactly the same as a graphic designer might specify that image5 is in front of image3, but behind image1.

<BODY>

The <BODY> tag is used to trigger an onLoad JavaScript event for setting up any arrays used by other elements. Aside from running simple setup procedures, the onLoad event handler may also be used to run a dynamic positioning script. Below listing shows the HTML code used to trigger an onLoad event.
onLoad Layers

```
<BODY onLoad="loadup()">
```

The code is very simple to understand and reuse. It catches the load event when the document loads, which then causes the loadup() function to be executed. This code is used in the demo, so if you are concerned that the listing seems a bit skimpy, don't be; the code for the complete demo is available later and you can study it further at that time.

<DIV>

Container elements are necessary for layering. Both IE and Netscape allow you to create layered presentations using the <DIV> tag. In IE you can make the layers dynamic with JavaScript; to do the same in NN requires that you use the <LAYER> tag (this will be discussed later).
<DIV> works by creating a block of content separate from the rest of the document. You can assign a <DIV> tag as often as necessary and specify how and where it will be displayed. Obviously, this requires using positioning properties, but we will overlook that for now. CSS positioning will be discussed in a future article. Take a look at the code in Listing below. Here, we demonstrate the use of the <DIV> tag to create a simple layered document.

Using DIV

```
<STYLE TYPE="text/css">
.div1 {z-index:1; display:none; }
.div2 {z-index:2; display:none; }
</STYLE>
...
<DIV ID=mydiv CLASS="div1">
<H1>Here is a Header</H1>
```

```
<P>Here is a Paragraph</P>
<IMG SRC="img1.gif">
</DIV>
<DIV ID=mydiv2 CLASS="div2">
<H1>Here is a Header</H1>
<P>Here is a Paragraph</P>
<IMG SRC="img2.gif">
</DIV>
```

Using the <DIV> tag is pretty simple. In fact, it is no more complicated than manipulating your document's layout in a table. Things start to get more complicated when you want to use scripting to dynamically control the layering of the <DIV>s. It's not too complicated though, especially if you are comfortable with JavaScript. Take a look at Listing 16.4 for an example of how JavaScript can be added to control your layers.

```
Scripting your DIVs
<SCRIPT LANGUAGE="JavaScript">
function loadup(){
div = document.all.tags("DIV");
lay = new Array();
lay[0] = div[0].style;
lay[1] = div[1].style;
SetTimeout("runshow()", 2000);
}
function runshow() {
for (i=0; i< lay.length; i++) {
    lay[i].display="block";
    }
}
</SCRIPT>
```

The loadup() function first creates a collection of all <DIV> elements in the document using the tags ("tagname") method. Once created, a collection behaves as if it were any other JavaScript object. For example, the Images object has an associated array, so you need not create a separate array to work with the collections members either. It is still necessary, however, to create a new array for the <DIV>s because display is a property of the style object. (According to the requirements of the IE Object Model, the keyword style needs to be used whenever working with a property of the style object.)

Now that the setup work is finished we can get down to the task of displaying the layers dynamically. In Listing 16.4, I used the setTimeout() method to give the script a bit of a pause before the layers are shown. This is included only to emphasize that this is dynamic layering; I wouldn't want people to think that it was merely a slow download. When the two-second pause elapses the runshow() function is executed.

The counter code in the runshow() function works something like this -- first it sets the value of i to 0, then it checks to see if the current execution is less than the total expected count. If i is less than lay.length, the counter adds one to the value of i for the next run, with i++.

Now the current value of i is placed into the code for lay[i].display="block". If this were the third rotation through the script, i would be 2, causing the JavaScript to interpret the code to read lay[2].display="block", and the DIV, whose position in the array corresponds to lay[2], would appear.

Arrays can also be useful for separating the <DIV>s used in code when there is more than one layered presentation on the same document.

```
New Div Array()
<SCRIPT LANGUAGE="JavaScript">
function loadup(){
div = document.all.tags("DIV");
d1 = new Array()
d1[0] = div[0].style
d1[1] = div[1].style
d2 = new Array()
d2[0] = div[2].style
d2[1] = div[3].style
setTimeout("runshow()", 2000);
}
```

In Listing above we created the <DIV> collection and two new arrays. Both arrays get their members from the same collection, but in splitting them like this we are able to view the two layered presentations separately.

LAYER

The <LAYER> tag is browser dependent; if you are not using Netscape it won't work. Basically it does the same thing in Netscape that <DIV> does in IE -- it allows you to make dynamic layers. The Netscape Object Model lets you create scripts that work with layers dynamically. Rather than go into a long discussion about the <LAYER> tag, we will get right to the listing.

```
Dynamic Layers in Netscape Navigator
<HTML>
<HEAD>
<STYLE TYPE="text/css">
.layer1 { z-index: 5; font-size: 12px; }
.layer2 { z-index: 4; font-size: 16px; color:red;}
.layer3 { z-index: 3; font-size: 20px; color: brown; }
.layer4 { z-index: 2; font-size: 24px; color:green; }
.layer5 { z-index: 1; font-size: 28px; color:blue; }
</STYLE>

<SCRIPT LANGUAGE="JavaScript">
function loadup(){
  lay = document.layers;
for (i=0; i< lay.length; i++) {
      lay[i].visibility="show"
if (i==4){
alert("That's all Folks!")
}
else
alert("show next layer")
  }
}
</SCRIPT>
</HEAD>
<BODY onload="loadup()">
<LAYER CLASS="layer1" visibility="hide">I'm a layer!</LAYER>
```

```
<LAYER CLASS="layer2" visibility="hide">I'm a layer</LAYER>
<LAYER CLASS="layer3" visibility="hide">I'm a layer</LAYER>
<LAYER CLASS="layer4" visibility="hide">I'm a layer</LAYER>
<LAYER CLASS="layer5" visibility="hide">I'm a layer</LAYER>
</BODY>
</HTML>
```

The script is a little different. For one thing, I decided to do the whole thing using a single function. The extra step is nice to add a little delay. Since layers are part of the Netscape Object Model, there is no need to create a collection (collections only work in IE). And because of the differences in the way NN and IE implement dynamic styling, there is no style keyword to worry about with Navigator.

The code is much shorter and easier to read and understand. We create a variable called lay based on the layers object. Next, we initiate a counter and rotate it through the script until the counter tells Navigator that i equals lay.length. Each layer in the array between lay[0] and lay.length will have the value of its visibility property changed from hide to show, making all of the layers appear. The alert messages are included only to slow things down a bit, but remember, there are other ways JavaScript can work with layers.

CHAPTER 14: An Overview of JavaScript

Scripting language

A high-level programming language that is interpreted by another program at runtime rather than compiled by the computer's processor as other programming languages (such as C and C++) are. Scripting languages, which can be embedded within HTML, commonly are used to add functionality to a Web page, such as different menu styles or graphic displays or to serve dynamic advertisements. These types of languages are client-side scripting languages, affecting the data that the end user sees in a browser window. Other scripting languages are server-side scripting languages that manipulate the data, usually in a database, on the server.
Scripting languages came about largely because of the development of the Internet as a communications tool. JavaScript, ASP, JSP, PHP, Perl, Tcl and Python are examples of scripting languages.

Objectives

- To examine the evolution of JavaScript and describe some of its key features
- To explain JavaScript fundamentals
- To start with scripting - sample program
- To understand Browser Compatibilities

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

The Nature of JavaScript

JavaScript is *the* scripting language of the Web!

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**.**

Introduction to JavaScript

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, and Opera.

What You Should Already Know?

Before you continue you should have a basic understanding of the following:

- HTML

- Ability to work with either Netscape Navigator or Internet Explorer for browsing.
- No prior programming experience required.

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

Are Java and JavaScript the Same Languages?

NO!

Java and JavaScript are two completely different languages in both concept and design!

Java (developed by Sun Microsystems) is a powerful and much more complex programming language - in the same category as C and C++.

What can a JavaScript Do?

- **JavaScript gives HTML designers a programming tool -** HTML authors are normally not programmers, but JavaScript is a scripting language with a very simple syntax! Almost anyone can put small "snippets" of code into their HTML pages
- **JavaScript can put dynamic text into an HTML page -** A JavaScript statement like this: document.write("<h1>" + name + "</h1>") can write a variable text into an HTML page
- **JavaScript can react to events -** A JavaScript can be set to execute when something happens, like when a page has finished loading or when a user clicks on an HTML element
- **JavaScript can read and write HTML elements -** A JavaScript can read and change the content of an HTML element
- **JavaScript can be used to validate data -** A JavaScript can be used to validate form data before it is submitted to a server. This saves the server from extra processing
- **JavaScript can be used to detect the visitor's browser** - A JavaScript can be used to detect the visitor's browser, and - depending on the browser - load another page specifically designed for that browser
- **JavaScript can be used to create cookies** - A JavaScript can be used to store and retrieve information on the visitor's computer

CHAPTER 15: Writing JavaScript code

Obejctives:

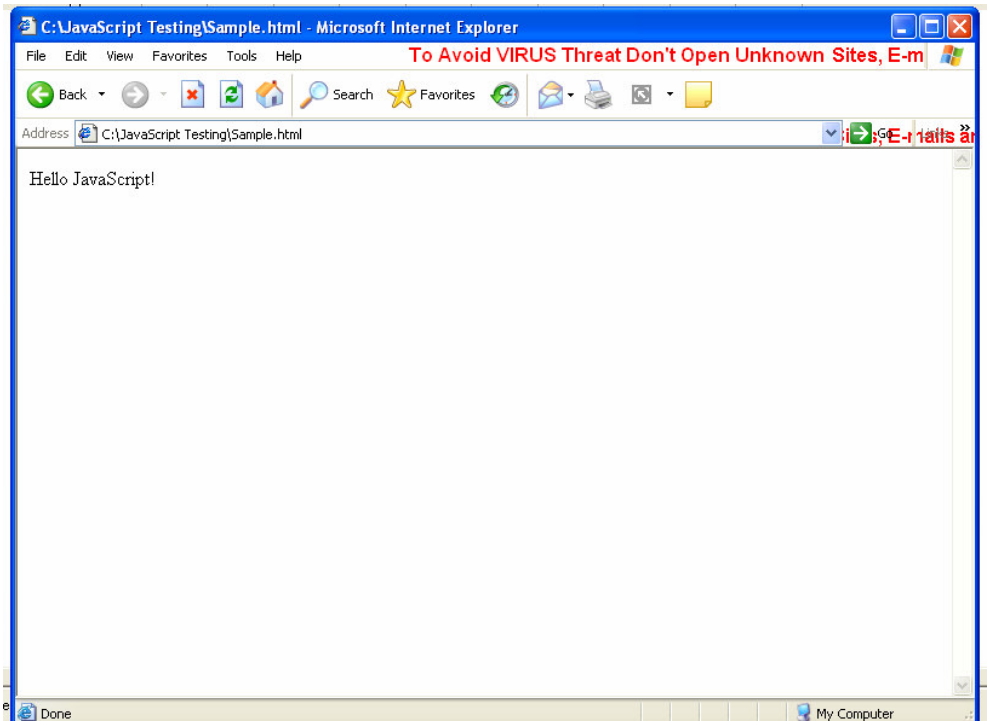
- Explain how to use script
- Use script in HTML Page

The HTML <script> tag.

How to Put a JavaScript Into an HTML Page?

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

The code above will produce **this output on an HTML page as follows:**



Explanation for the Example

To insert a JavaScript into an HTML page, we use the <script> tag (also use the language attribute to define the scripting language).
So, the <script language="JavaScript"> and </script> tells where the JavaScript starts and ends:

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

```
<script language="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 language="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 Java! to the page:

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

Note: If we had not entered the <script> tag, the browser would have treated the document.write("Hello JavaScript!") command as pure text, and just write the entire line on the page.

Note: Though we have used JavaScript, we have embedded the code of JavaScript in HTML file, therefore we have to give '.html' or '.htm' extension to the file !

Ending Statements with a Semicolon is necessary?

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 as follows:

```
<script language="JavaScript">
<!--     document.write("Hello JavaScript!")//-->
</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

JavaScript in the body section will be executed WHILE the page loads.

JavaScript in the head section will be executed when CALLED.

Where to Put the JavaScript?

JavaScript 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 language=”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 language=”javascript">
      ....</script></body>
```

Scripts in both the body and the head section:
We 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 language=”JavaScript">
     ....</script>
</head>
<body>
<script language=”JavaScript">
   ....</script>
</body>
```

Using an External JavaScript in HTML file

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="fileName.js">
</script>
</head>
<body>
</body>
</html>
```

Note: Remember to place the script exactly where you normally would write the script!

Check your progress:

Q. 1 JavaScript must reside in which of the following tag?
a) <object> tag
b) <applet> tag
c) <script> tag
d) <cgi> tag

Q.2 write() is a type of _____
a) Object
b) Method
c) Property
d) Variable

Q. 3 How do you prevent JavaScript code from being displayed by older browser?
a) Place JavaScript within <script> tag
b) Place JavaScript within header
c) Place JavaScript within comment
d) Place JavaScript within body.

Q. 4 Who created Javascript?
a) Microsoft
b) Netscape
c) Sun Micro Systems

Q. 5 A dot is used to____--
a) Identify a JavaScript comment
b) Separate lines of JavaScript
c) End JavaScript statement
d) Separate an object name from either a property or a method.

JavaScript Popup Boxes

In JavaScript we can create three kinds of popup boxes: Alert box, Confirm box, and Prompt box.

Alert Box

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

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

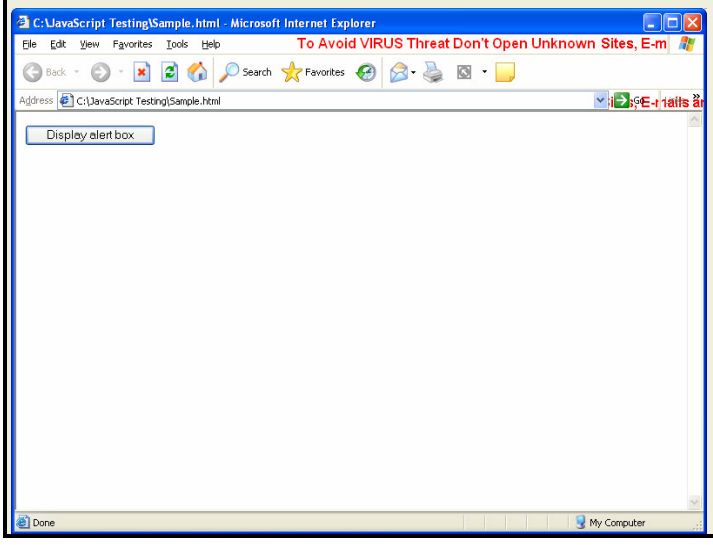
Syntax:

```
alert("Alert Message")
```

Examples
a) Alert Box

```
<html>
<head>
<script language="javascript">
function show_alert()
{
        alert("Hi! This is alert box!!")
}
</script>
</head>
<body>

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

</body>
</html>
```

output:

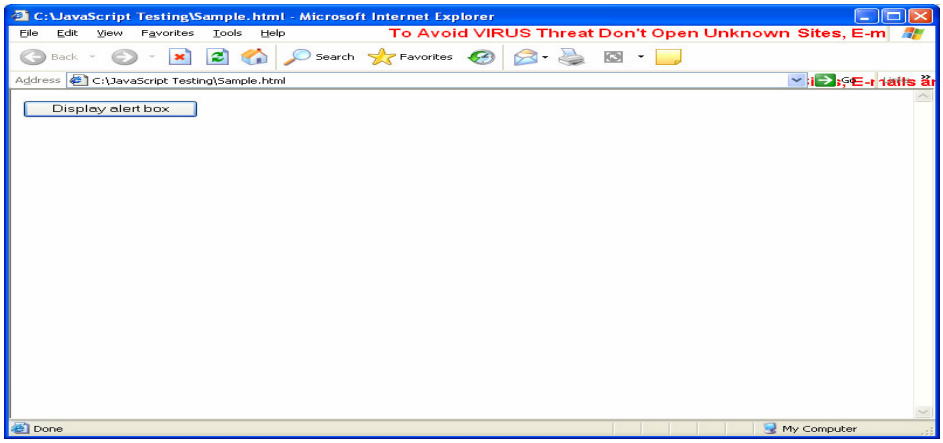


As soon as user clicks on button, user will get one alert box like this.

b) Alert box with line breaks

```
<html>
<head>
<script language="javascript">
function disp_alert()
{
        alert("Hello again! This is how we" + '\n' + "add line breaks to an alert box!")
}
</script>
</head>
<body>
<input type="button" onclick="disp_alert()" value="Display alert box" />
</body>
</html>
```

Output:



When user click on button, will get above alert message

Confirm Box

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

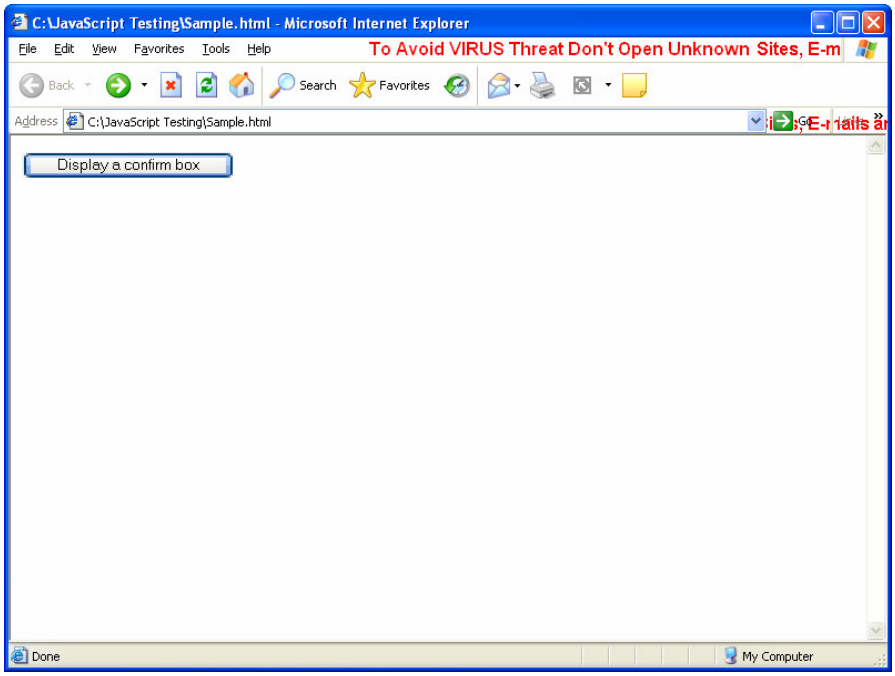
Syntax:

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

Example:

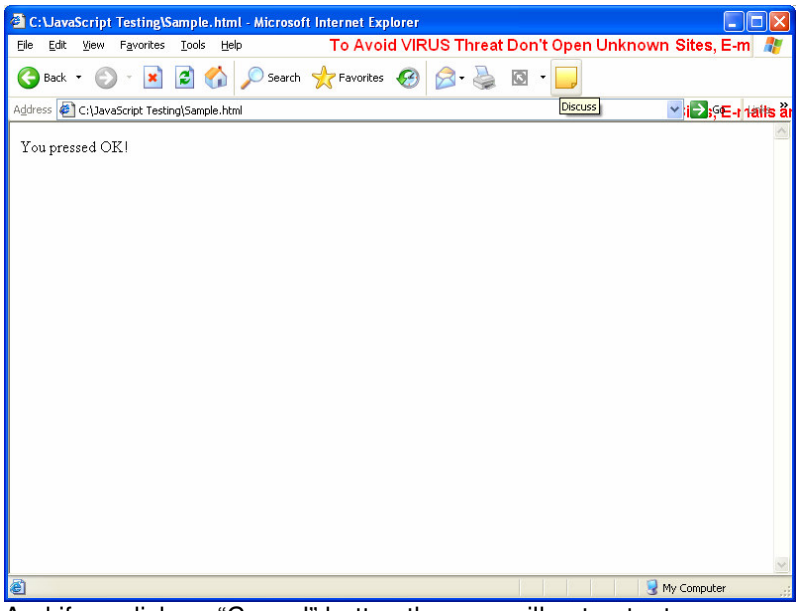
```
<html>
<head>
<script language="javascript">
function disp_confirm()
  {
  var r=confirm("Press a button")
  if (r==true)
    {
    document.write("You pressed OK!")
    }
  else
    {
    document.write("You pressed Cancel!")
    }
  }
</script>
</head>
<body>

<input type="button" onclick="disp_confirm()" value="Display a confirm box" />

</body>
</html>
```

Output:

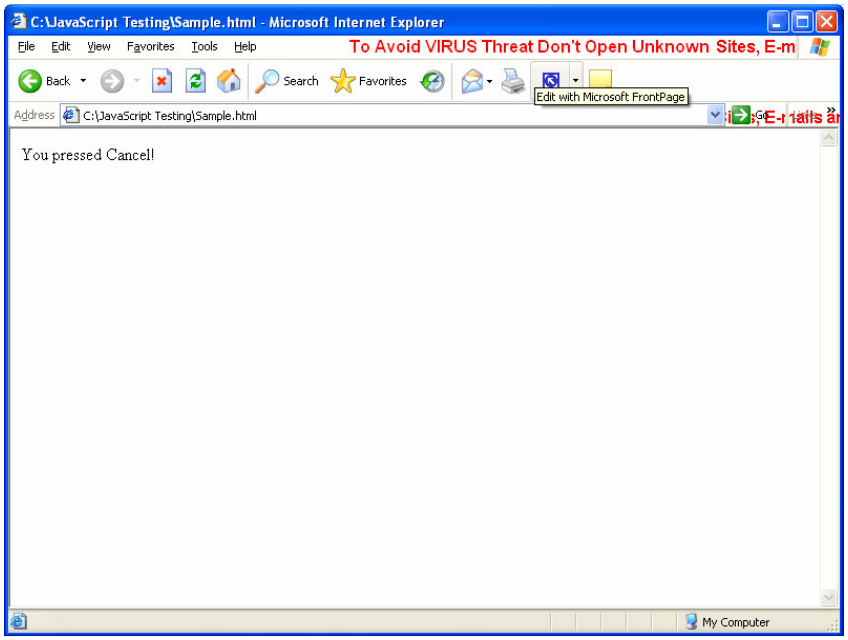


When user clicks on button he will get confirm box as follows

After clicking on ‘OK’ button we will get output as



And if we click on “Cancel” button then we will get output as



Prompt Box

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

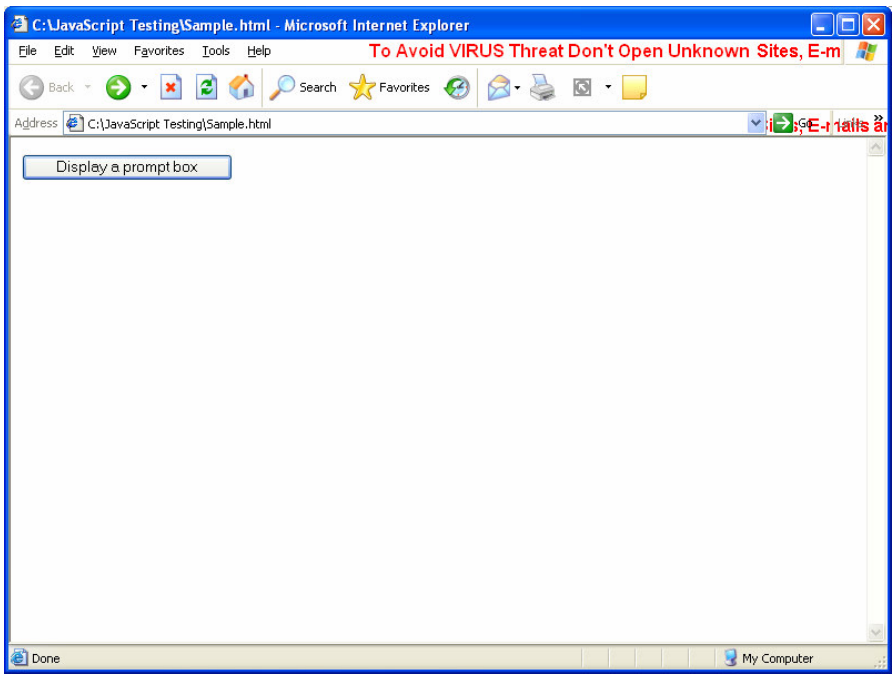
Syntax:

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

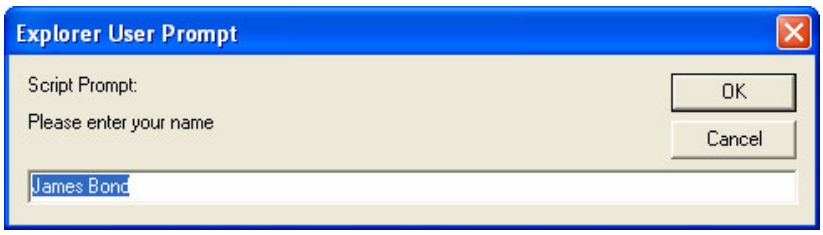
Example:

```
<html>
<head>
<script language="JavaScript">
function disp_prompt()
{
  var name=prompt("Please enter your name","James Bond")
  if (name!=null && name!="")
    {
      document.write("Hello " + name + "! How are you
      today?")
    }
}
</script>
</head>
<body>

<input type="button" onclick="disp_prompt()" value="Display a prompt box">

</body>
</html>
```

Output:



As soon as user clicks on button, user will get prompt asking for name as follows:


Explorer User Prompt
Script Prompt:
Please enter your name
OK
Cancel
James Bond

CHAPTER 16: Variables, Date Types and Operators in JavaScript

Objectives:

- To understand Variables and data types
- To Use Arithmetic, Assignment, Logical and String Operators

Variables

A variable is a "container" for information you want to store. A variable's value can change during he 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 variablename = some value

You can also create a variable without the var statement:

variablename = some valueAssign

a Value to a Variable

You can assign a value to a variable like this:

```
var fName = "Jack"
```

Or like this:

fName = "Jack"

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 "fName" has the value "Jack".

Lifetime of Variables

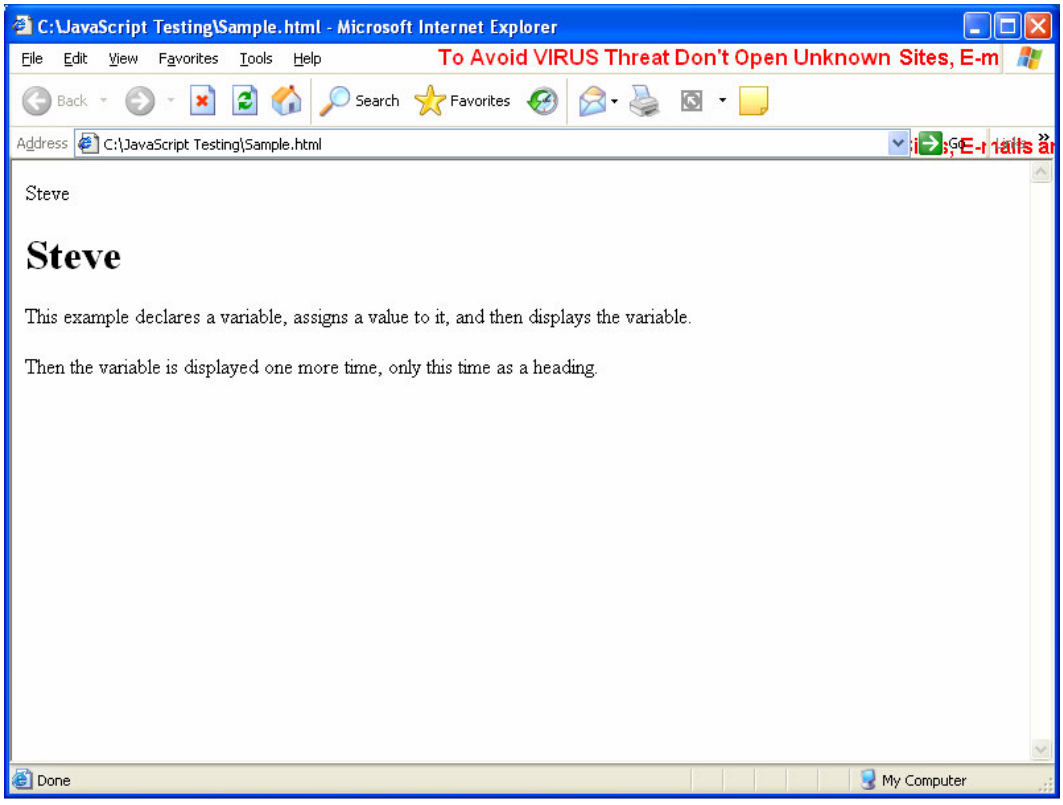
When you declare a variable within a function, the variable can only be accessed within that function. When you exit the function, the variable is destroyed. These variables are called local variables. You can have local variables with the same name in different functions, because each is recognized 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.

Examples

Following example will show you how Variables are used to store data.

```
<html>
<body>

<script language=”javascript”>
var name = "Steve"
document.write(name)
document.write("<h1>"+name+"</h1>")
</script>

<p>This example declares a variable, assigns a value to it, and then displays the
variable.</p>

<p>Then the variable is displayed one more time, only this time as a heading.</p>

</body>
</html>
```

Output:



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

Example:

```
 <html>
<head>
<title>Operator Example</title>
</head>
<body>
<script language="JavaScript">
<!--
          var a = 5
          ++a
          alert("The value of a = " + a )
 -->
</script>
</body>
</html>
```

Output

Assignment Operators

Operator	Example	It is 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

Example:

```
<html>
<head>
<title>Operator Example</title>
</head>
<body>
<script language="JavaScript">
<!--
            var a = 10
            var b = 2
            a += b
            alert("a += b is " + a )
 -->
</script>
</body>
</html>
```

Output :

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

Example

```
<html>
<head>
<title>Operator Example</title>
</head>
<body>
<script language="JavaScript">
<!--
        var userID
        var password
        userID = prompt("Enter User ID"," ")
        password = prompt("Enter Password"," ")
        if(userID == "user" && password == "secure")
        {
                alert("Valid login")
        }
        else
        {
                alert("Invalid login")
        }
 -->
</script>
</body>
</html>
```

Output

Figure: The browser tells the user if the user ID and password are valid

Figure: The browser tells the user if the user ID and password are invalid

String Operator

A string is most often text, for example "Hello L&T Infotech!". To stick two or more string variables together, use the + operator.

```
txt1="Welcome "
txt2="to L&T Infotech Ltd.!"
txt3 = txt1 + txt2
```

The variable txt3 now contains "Welcome to L&T Infotech Ltd.!".

To add a space between two string variables, insert a space into the expression, OR in one of the strings.

```
txt1="Welcome"
txt2="to L&T Infotech Ltd.!"
txt3=txt1 + " " + txt2
or
txt1="Welcome "
txt2="to L&T Infotech Ltd.!"
txt3=txt1 + txt2
```

The variable txt3 now contains "Welcome to L&T Infotech Ltd.!".

Conditional Operator

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

Syntax

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

Example

```
Greeting = (visitor=="PRES")?"Dear President ":"Dear "
```

If the variable visitor 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.

CHAPTER 17: Conditional Statements and Looping Constructs in JavaScript

Conditional Statements

A conditional statement is a set of commands that executes if a specified condition is true. JavaScript supports two conditional statements: if...else and switch.

If condition

Use the if statement to perform certain statements if a logical condition is true; use the optional else clause to perform other statements if the condition is false.

Syntax of if … else condition

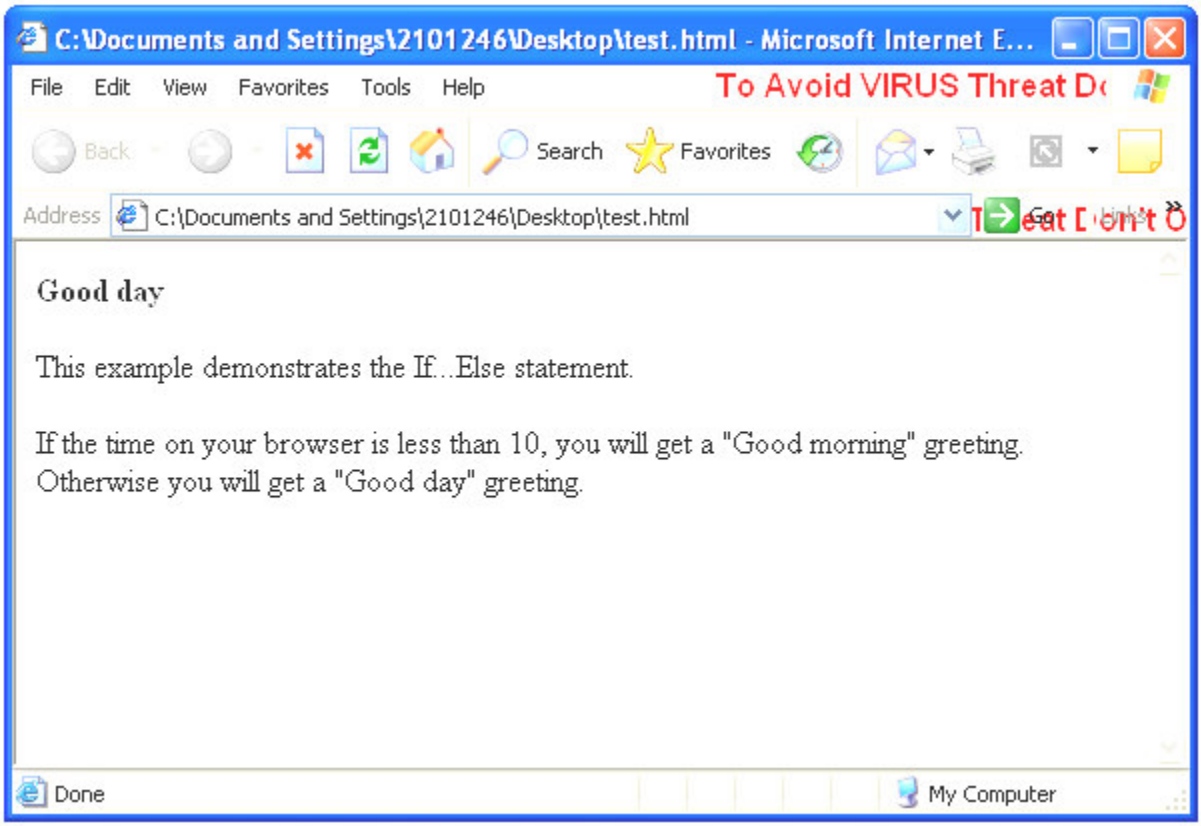
```
if ( condition) {
statements1
}
 else {
statements2
}
```

Example

```
<html>
<body>
<script language="javascript">
var d = new Date()
var time = d.getHours()
if (time < 10)
{
document.write("<b>Good morning</b>")
}
else
{
document.write("<b>Good day</b>")
}
</script>
<p>
This example demonstrates the If...Else statement.
</p>
<p>
If the time on your browser is less than 10,
you will get a "Good morning" greeting.
Otherwise you will get a "Good day" greeting.
</p>
</body>
</html>
</html>
```

Explanation – variable time will store current time in hours, if a condition check for time is less than 10. If time is less than 10 then "Good morning" message is printed otherwise control goes to else block and "Good day" message is displayed on web page

Output of the code-



Switch Statement

A switch statement allows a program to evaluate an expression and attempt to match the expression's value to a case label. If a match is found, the program executes the associated statement.

Syntax of Switch Statement

```
switch ( expression)
{
        case label :
        statement;
                break;
        case label :
        statement;
                break;
        ...
        default : statement;
}
```

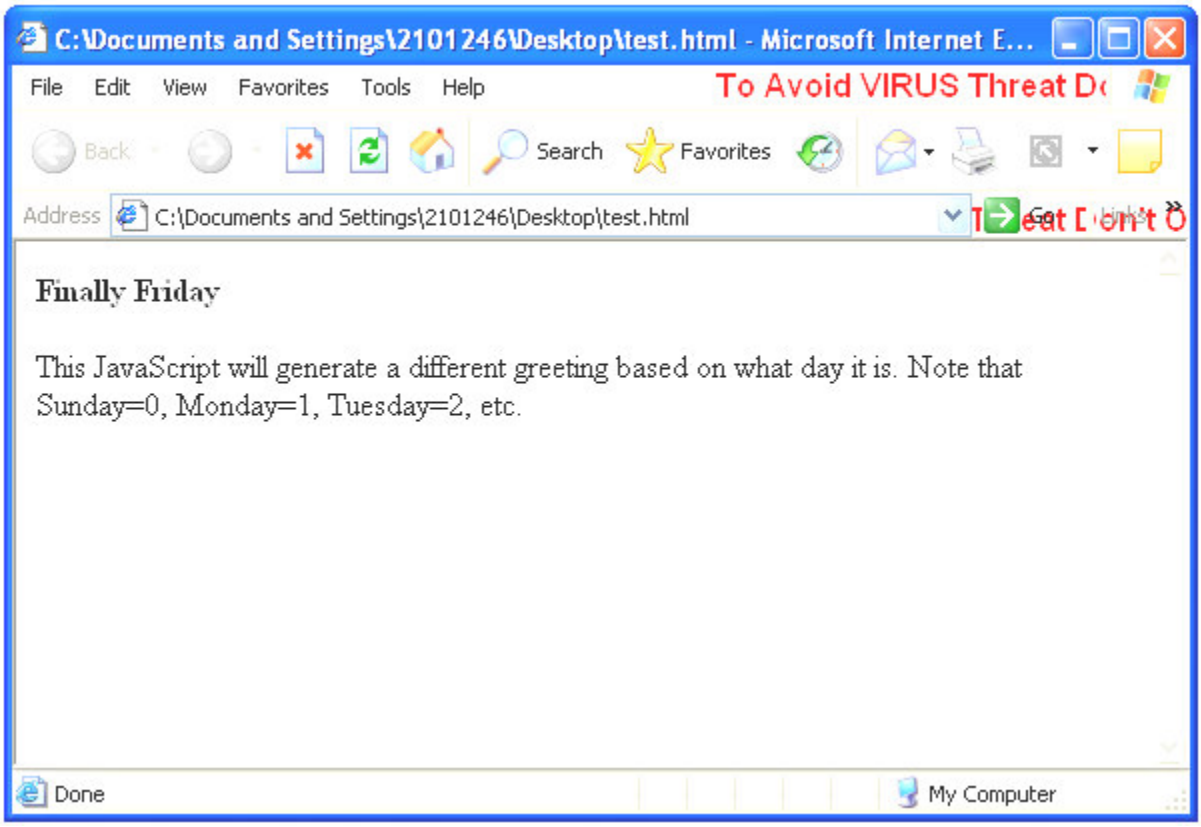
The program first looks for a label matching the value of expression and then executes the associated statement. If no matching label is found, the program looks for the optional default statement, and if found, executes the associated statement. If no default statement is found, the program continues execution at the statement following the end of switch. The optional break statement associated with each case label ensures that the program breaks out of switch once

the matched statement is executed and continues execution at the statement following switch. If break is omitted, the program continues execution at the next statement in the switch statement. Example –

```
<html>
<body>
<script language="javascript">
var d = new Date()
theDay=d.getDay()
switch (theDay)
{
case 5:
  document.write("<b>Finally Friday</b>")
  break
case 6:
  document.write("<b>Super Saturday</b>")
  break
case 0:
  document.write("<b>Sleepy Sunday</b>")
  break
default:
  document.write("<b>I'm really looking forward to this weekend!</b>")
}
</script>
<p>This JavaScript will generate a different greeting based on what day it is. Note that
Sunday=0, Monday=1, Tuesday=2, etc.</p>
</body>
</html>
```

Explanation – the code will store day value in variable theDay and depending on this value message inside case is displayed on web page. D.getDay will always give some numerical value. Numerical value for Sunday is 0, and for Saturday it is 6. If theDay value doesn't satisfy given case statement then default case is executed.

Output of code-



Looping Construct:

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 kinds of loops:

- **for**
- **while**

for - loops through a block of code a specified number of times
Syntax of for loop

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

When a for loop executes, the following occurs:
1. The initializing expression initial-expression, if any, is executed. This expression usually initializes one or more loop counters, but the syntax allows an expression of any degree of complexity.
2. The condition expression is evaluated. If the value of condition is true, the loop statements execute. If the value of condition is false, the for loop terminates.
3. The statements execute.

4. The update expression incrementExpression executes, and control returns to Step 2.

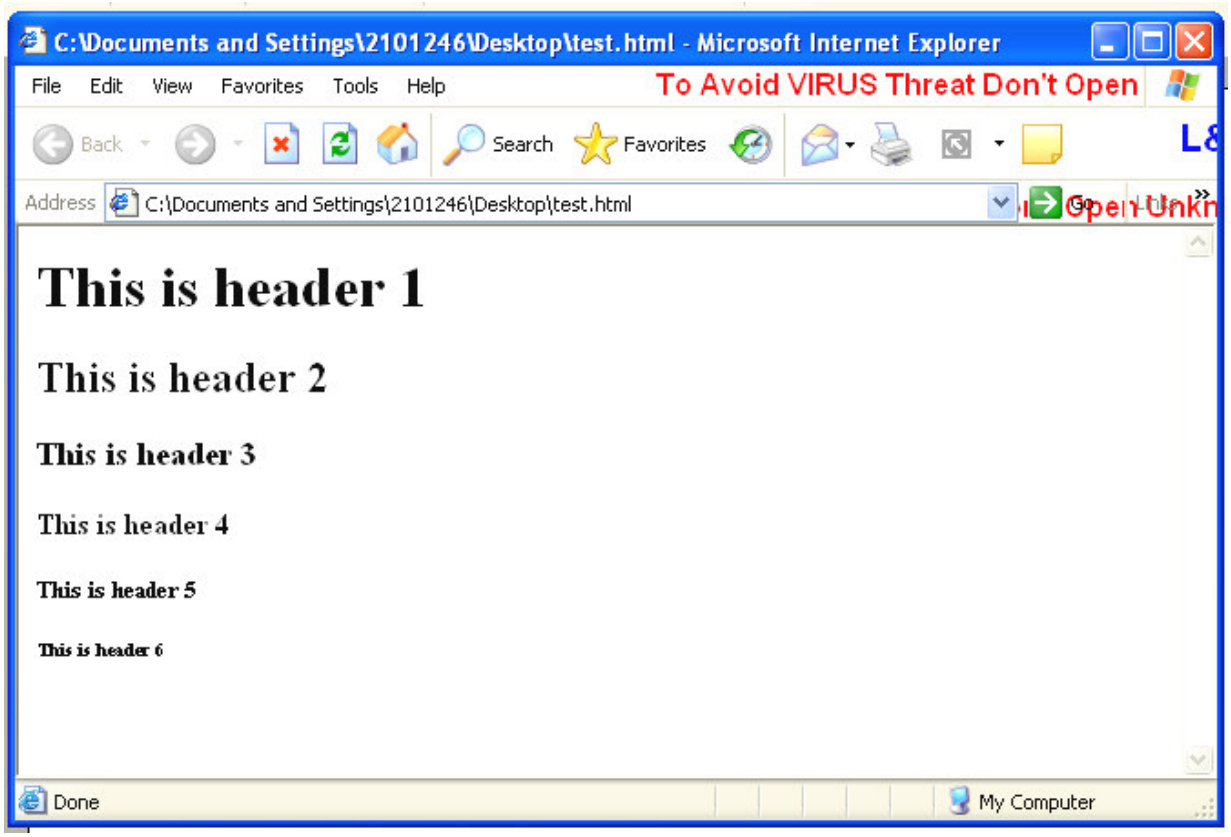
following example of for loop will print all header on web page

Example 1.

```
<html>
<body>
<script language="javascript">
for (i = 1; i <= 6; i++)
{
document.write("<h" + i + ">This is header " + i)
document.write("</h" + i + ">")
}
</script>
</body>
</html>
```

Explanation: The example above defines a loop that starts with i=1. The loop will continue to run as long as i is less than, or equal to 6. i will increase by 1 each time the loop runs. Results in printing six predefined headers in HTML.

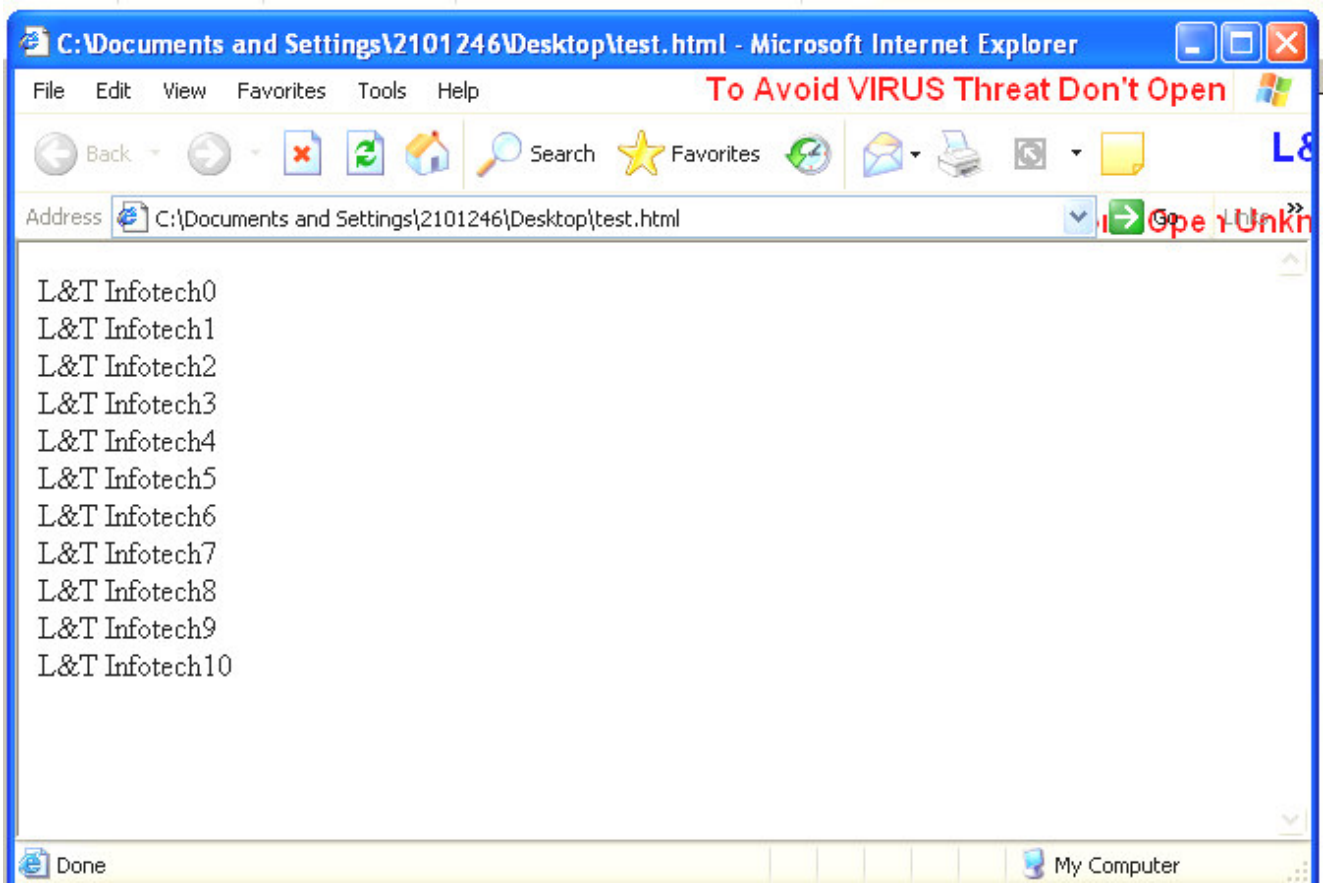
Output of the code

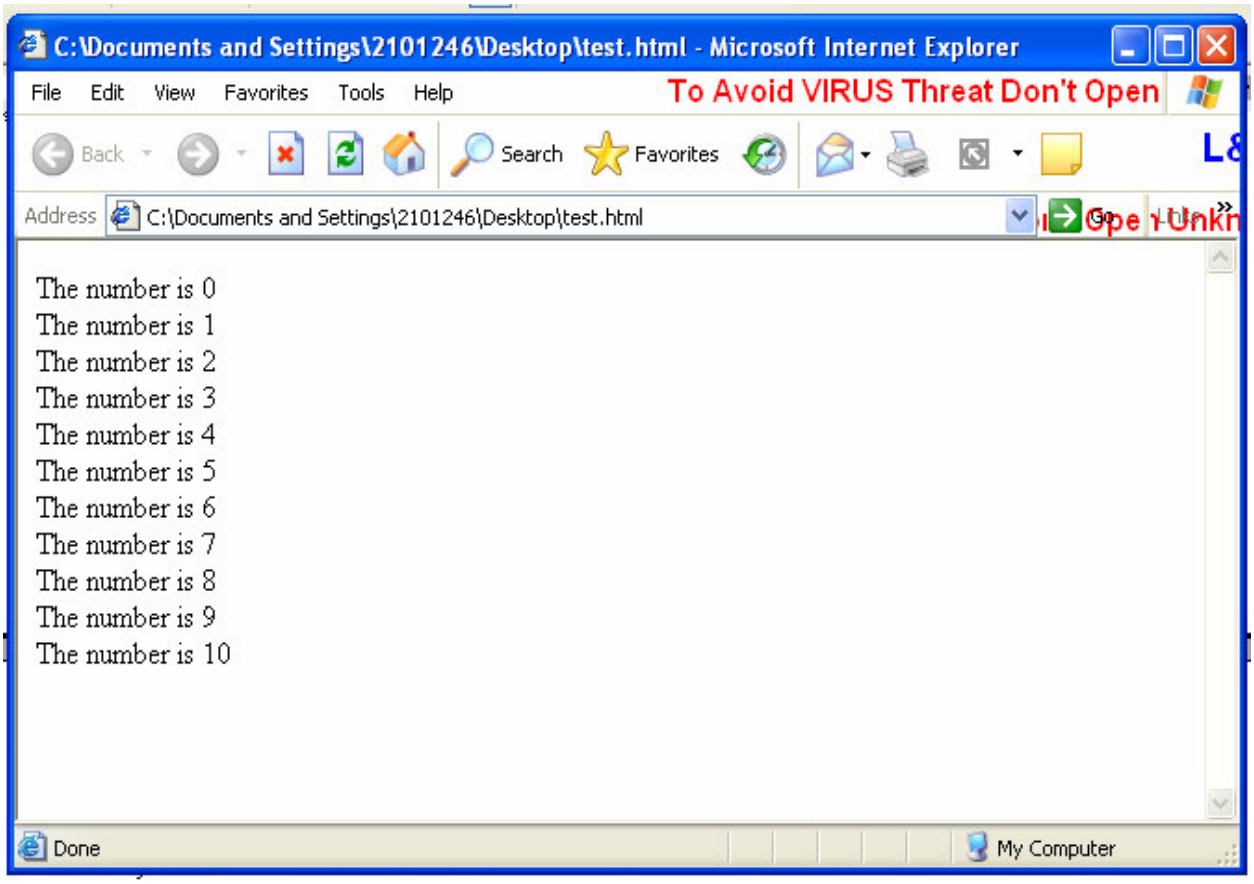


Example 2

```
<html>
<body>
<script language="javascript">
var i=0
for (i=0;i<=10;i++)
{
document.write("L&T Infotech" + i)
document.write("<br />")
}
</script>
```

Explanation:
The example above 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.





while Loop-

Loops through a block of code while a specified condition is true while loop is used to run the same block of code while a specified condition is true.

Syntax for while loop

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

Example 1

```
<html>
<body>
<script language="javascript">
var i=0
while (i<=10)
{
document.write("The number is " + i)
document.write("<br />")
i=i+1
}
</script>
</body>
```

```
</html>
```

Explanation: The example above 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.

Output of the code

do….while loop –

The do...while statement repeats until a specified condition evaluates to false.
Syntax of do…..while loop

```
do {
statement
} while
```

statement executes once before the condition is checked. If condition returns true, the statement executes again. At the end of every execution, the condition is checked. When the condition returns false, execution stops and control passes to the statement following do...while.

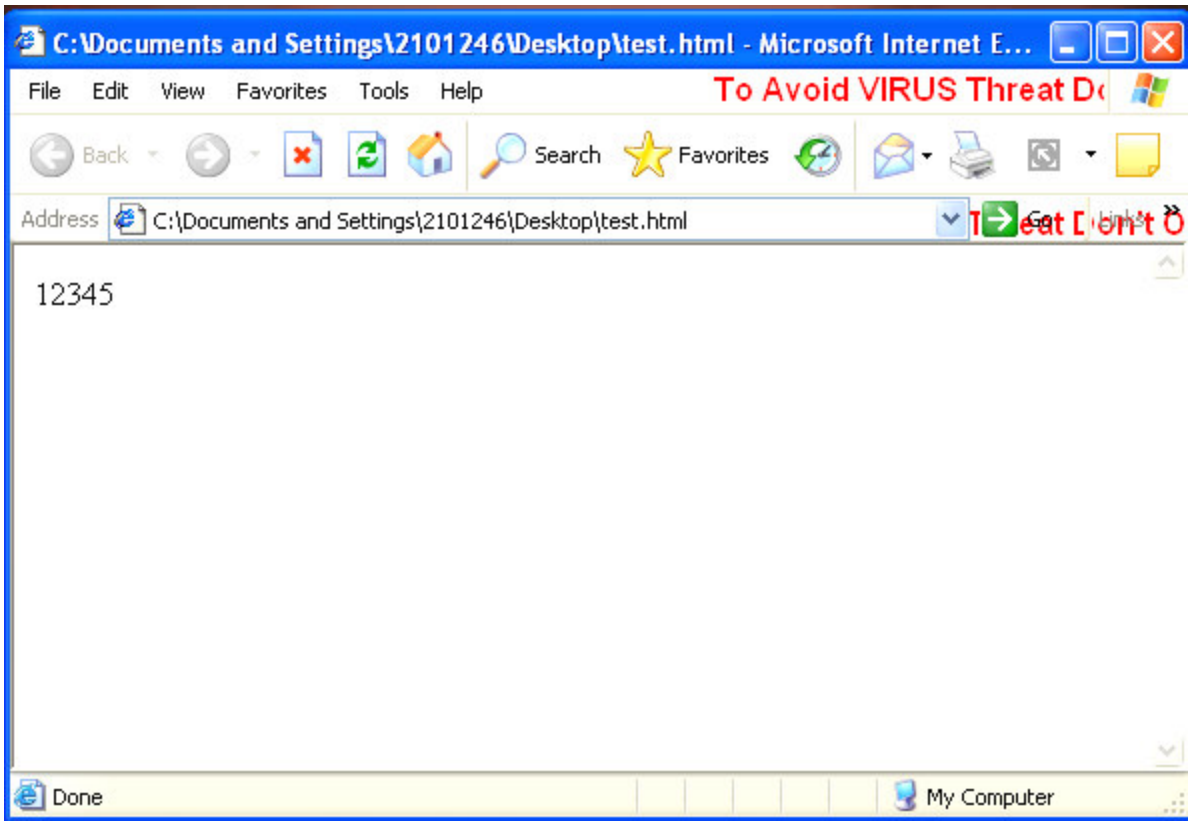
Example-

```
<html>
<body>
<script language="javascript">
i=0
do {
i++;
document.write(i);
} while (i<5);
</script>
</body>
```

Explanation-

In the following example, the do loop iterates at least once and reiterates until i is no longer less than 5.

Output of the code



Break statement

Use the break statement to break the loop.

Continue statement

Use the continue statement to break the current loop and continue with the next value.

For...In Statement –

The for...in statement is used to loop (iterate) through the elements of an array or through the properties of an object. The code in the body of the for ... in loop is executed once for each element/property.

Syntax of for……in Statement

```
for (variable in object)
{
   code to be executed
}
```

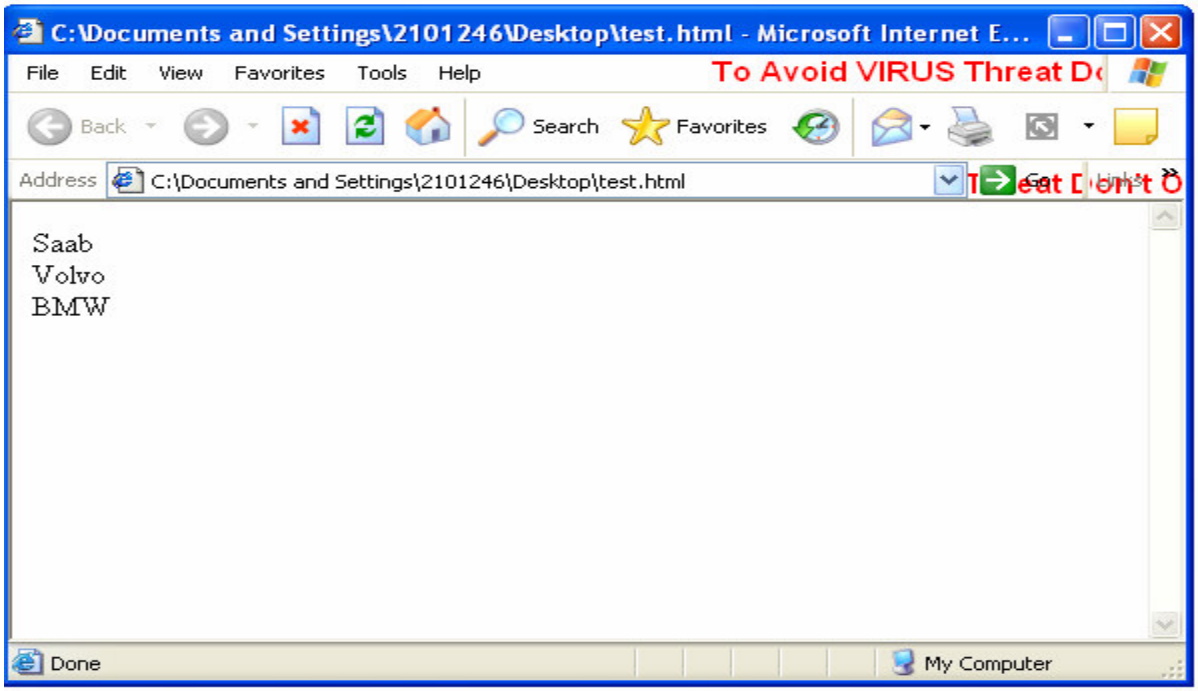
Example –

```
<html>
<body>
<script language="javascript">
var x
var mycars = new Array()
mycars[0] = "Saab"
mycars[1] = "Volvo"
mycars[2] = "BMW"

for (x in mycars)
```

```
{
document.write(mycars[x] + "<br />")
}
</script></body>
</html>
```

Explanation-
Above code will display all the elements of array mycars on web page. With each element on new line

output of the code –



Test your progress:

Q.1 What is happening in the expression ++a?

a) The value of a is increased by 2
b) The value of a is increased by 1
c) The value of a is multiplied by itself
d) Nothing; this is not a valid JavaScript expression.

Q. 2 Evaluate following expression:

7<10 ? 'You win' :' You lose'

a) 10
b) You lose
c) You win
d) 7

Q.3 State whether following statement is true or false

The ++ can be either the right(c=a++) or left (c=++a) side of an expression without any effect on the expression

a) True
b) False

Q. 4 State whether following statement is true or false

The x +=y expression adds values of x and y and stores the sum in x.

a) True
b) False

Q.5 State whether following statement is true or false

The !=operator makes a false true

a) True
b) False

CHAPTER 18: Built-in Objects

Objectives:

- To describe the core objects, and core functions
- To understand Date() object, all it's methods and Properties
- To know how to declared, initialized and populate array, its methods, properties and practical application
- To Work with string object, all it's methods and Properties
- To Understand Math object

Java Script is an object-oriented language. Therefore it is important to understand what objects are and how they work. Objects allow you to organize information. Let's take a house for example. It has certain properties, such as style color, age, location, ... These properties describe the house. In Java Script, properties describe the object. In the case of the house, the house is the object and its color is one of its properties.
For our example lets say we have a red, two story house that is 16 years old.

The house is our object.
house.color= "red"
house.style= "two story"
house.age= "16 years old"

In Java Script the name of the object comes first, followed by the property you are referring to. For the color property house comes first because we are talking about the house object. The object and property are separated by a period.

An object also has functions associated with it that are known as the object's methods. This section describes the predefined objects in core JavaScript like Array, Date, Math and String.

Date object

Date - 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()
Methods with date object

getDate() - Returns the day of the month. It is returned as a value between 1 and 31.

```
var curdate = new Date()
var mday = curdate.getDate()
document.write(mday + "<BR>")
```

The above code prints the day of the month.

getDay() - Returns the day of the week as a value from 0 to 6

```
var curdate = new Date()
var wday = curdate.getDate()
document.write(wday + "<BR>")
```

The above code prints the day of the week.

getHours() - The value returned is 0 through 23.

```
var curdate = new Date()
```

```
var hours = curdate.getHours()
document.write(hours + "<BR>")
```

The above code prints the hours since midnight.

getMinutes() - The value returned is 0 through 59.

```
var curdate = new Date()
var minutes = curdate.getMinutes()
document.write(minutes + "<BR>")
```

The above code prints the minutes past the hour.

getMonth() - Returns the month from the date object as a value from 0 through 11.

```
var curdate = new Date()
var month = curdate.getMonth()
document.write(month + "<BR>")
```

The above code prints the numeric value of the month.

getSeconds() - The value returned is 0 through 59.

```
var curdate = new Date()
var seconds = curdate.getSeconds()
document.write(seconds + "<BR>")
```

The above code prints the seconds since the last minute.

getTime() - The number of milliseconds since January 1, 1970. this function allows us to manipulate the date object based on a millisecond value then convert it back to the form we want. In the example below, it is used to set a future expiration time of a cookie.

```
var futdate = new Date()
var expdate = futdate.getTime()
expdate += 3600*1000 //expires in 1 hour(milliseconds)
futdate.setTime(expdate)
```

getTimeZoneOffset() - Time zone offset in hours which is the difference between GMT and local time.

```
var curdate = new Date()
var offset = curdate.getTimeZoneOffset()
document.write(offset + "<BR>")
```

The above code prints the number of hours different between current time zone and GMT.

getYear() - Returns the numeric four-digit value of the year.

```
var curdate = new Date()
var year = curdate.getYear()
document.write(year + "<BR>")
```

The above code prints the numeric value of the year, which is currently 2000.

parse() - The number of milliseconds after midnight January 1, 1970 till the given date expressed as a string in the example which is IETF format.

```
var curdate = "Wed, 18 Oct 2000 13:00:00 EST"
var dt = Date.parse(curdate)
document.write(dt + "<BR>")
```

setDate(value) - Set the day of the month in the date object as a value from 1 to 31.

setHours(value) - Set the hours in the date object with a value of 0 through 59.

setMinutes(value) - Set the minutes in the date object with a value of 0 through 59.

setMonth(value) - Set the month in the date object as a value of 0 through 11.

setSeconds(value) - Set the seconds in the date object with a value of 0 through 59.

setTime(value) - Sets time on the basis of number of milliseconds since January 1, 1970.

The below example sets the date object to one hour in the future.

```
var futdate = new Date()
var expdate = futdate.getTime()
expdate += 3600*1000 //expires in 1 hour(milliseconds)
futdate.setTime(expdate)
```

setYear(value) - Set the year in the date instance as a 4 digit numeric value.

toGMTString() - Convert date to GMT format in a form similar to "Fri, 29 Sep 2000 06:23:54 GMT".

```
var curdate = new Date()
dstring = curdate.toGMTString()
document.write(dstring + "<BR>" + curdate.toLocaleString() + "<BR>")
```

The above example produces:

Wed, 18 Oct 2000 18:08:11 UTC
10/18/2000 14:08:11

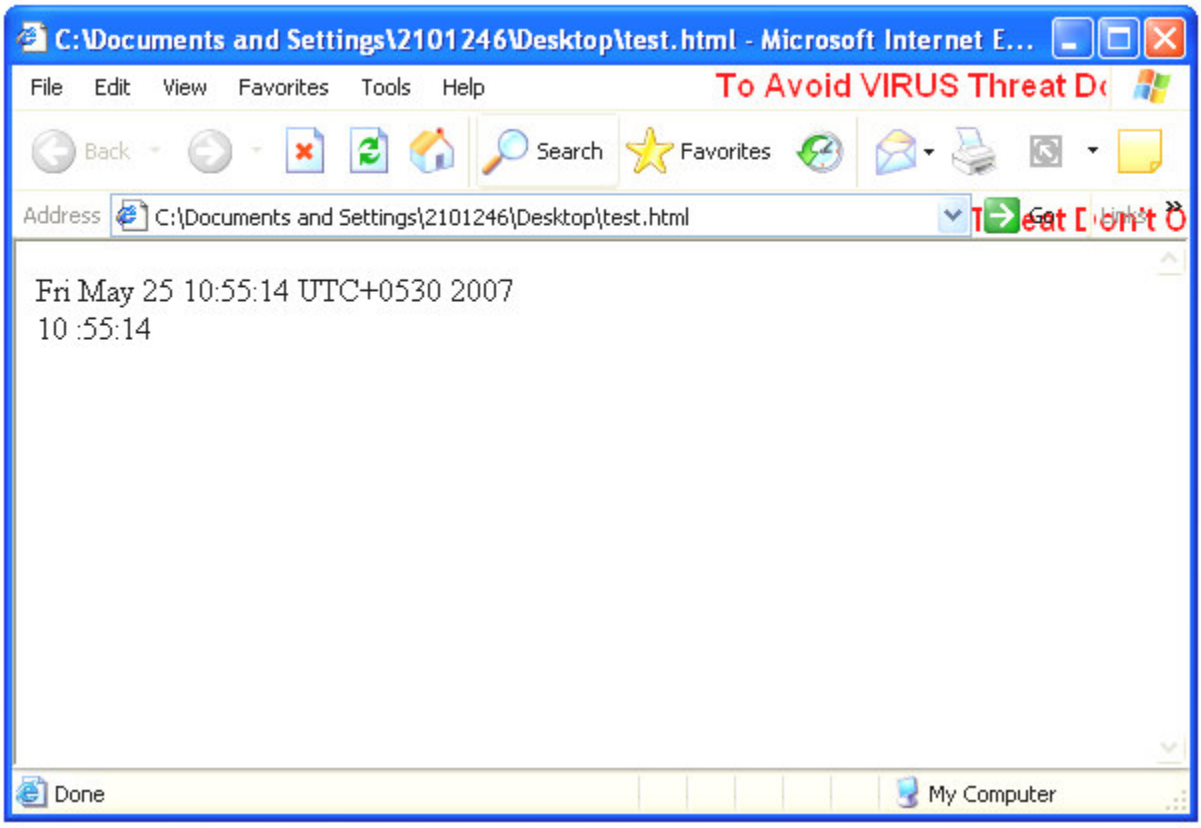
toLocaleString() - Convert date to local time zone format.

Example –

```
<html>
<head>
<script language="javascript">
        var today= new Date()
        document.write(today)
        var h= today.getHours()
        var m=today.getMinutes()
        var s=today.getSeconds()
        document.write("<br>")
        document.write(h +" :"+m+":"+s)
</script>
</head>
</html>
```

Explanation – First statement of above code will display current date and time in UTC (Universal Time Coordinates) format. Variables h, m and s, will display hours, minutes and seconds from given date object respectively

Output of the code –



Arrays

Array Object

JavaScript does not have an explicit array data type. However, we can use the predefined Array object and its methods to work with arrays in applications. The Array object has methods for manipulating arrays in various ways, such as joining, reversing, and sorting them. It has a property for determining the array length and other properties for use with regular expressions. An array is an ordered set of values that you refer to with a name and an index. For example, you could have an array called emp that contains employees' names indexed by their employee number. So emp[1] would be employee number one, emp[2] employee number two, and so on.

Creating an Array

To create an Array object:
1. arrayObjectName = new Array(element0, element1, ..., element N)
2. arrayObjectName = new Array(arrayLength)

arrayObjectName is either the name of a new object or a property of an existing object. When using Array properties and methods, arrayObjectName is either the name of an existing Array object or a property of an existing object. element0, element1, ..., element N is a list of values for the array's elements. arrayLength is the initial length of the array. The following code creates an array:

```
billingMethod = new Array(5)
coffees = ["French Roast", "Columbian", "Kona"]
```

Populating an Array – Arrays can be populated by following ways

```
emp[1] = "Casey Jones"
emp[2] = "Phil Lesh"
emp[3] = "August West"
myArray = new Array("Hello", myVar, 3.14159)
```

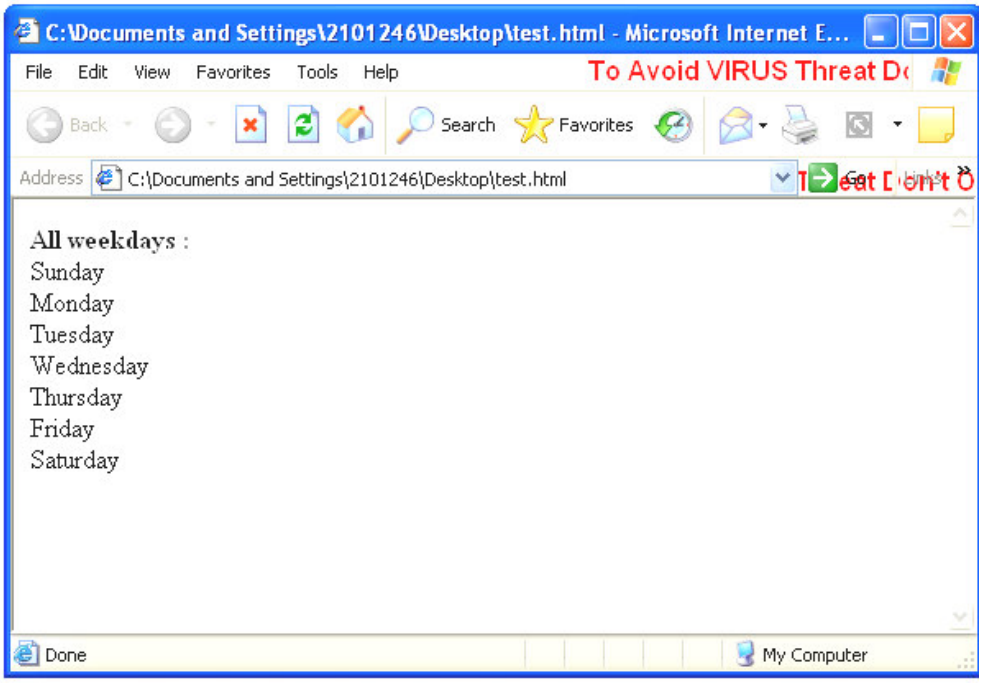
Unlike other programming language array can hold elements of different data types.

Example –

```
<html>
<body>
<script language="javascript">
var weekday=new Array(7)
weekday[0]="Sunday";
weekday[1]="Monday";
weekday[2]="Tuesday";
weekday[3]="Wednesday";
weekday[4]="Thursday";
weekday[5]="Friday";
weekday[6]="Saturday";
document.write("<b>All weekdays :</b> "+"<br>")
for (i=0;i<=6;i++)
{
document.write(weekday[i]+"<br>")
}
</script>
</body>
</html>
```

Explanation – Above code initializes array weekday and stores values as dayname. All the values of array elements are displayed on web page by using for loop

Output of the code



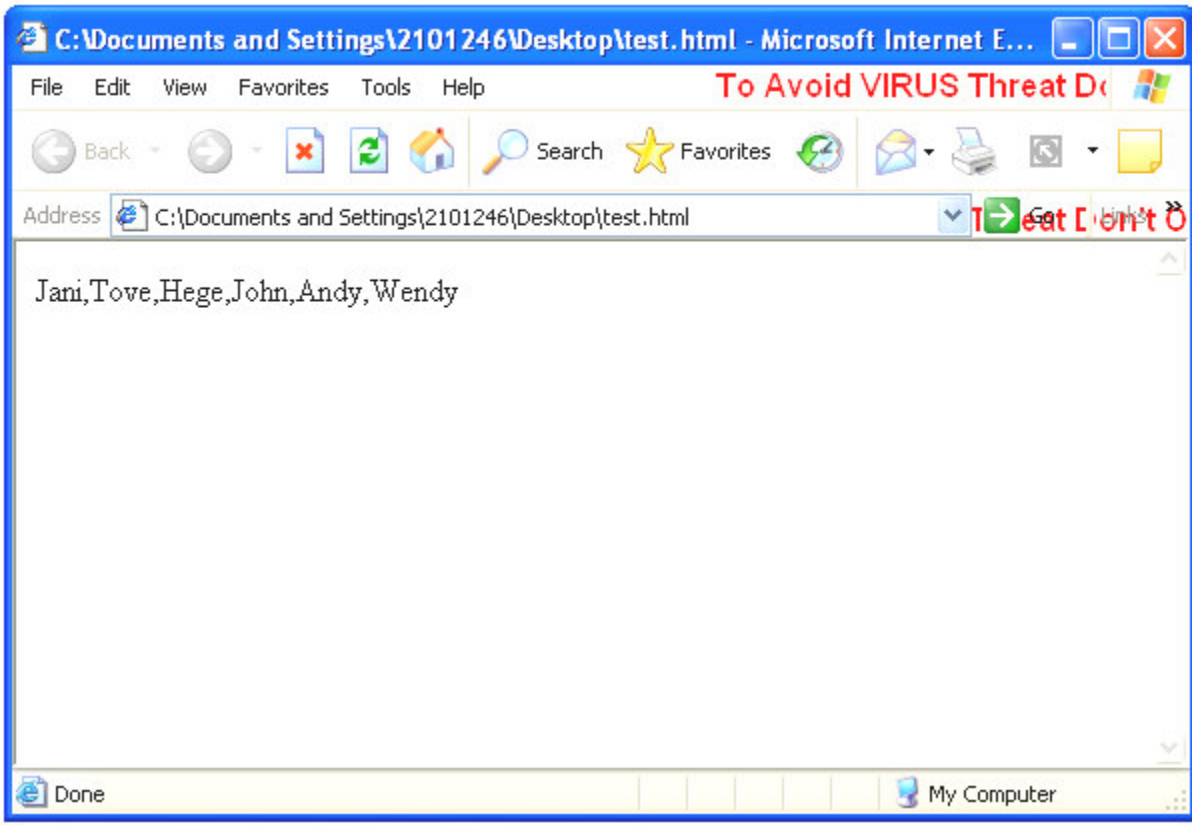
Array Method –

Arrayname.concat() -concat joins two arrays and returns a new array.

Example –

```
<html>
<body>
<script language="javascript">
var arr = new Array(3)
arr[0] = "Jani"
arr[1] = "Tove"
arr[2] = "Hege"

var arr2 = new Array(3)
arr2[0] = "John"
arr2[1] = "Andy"
arr2[2] = "Wendy"

document.write(arr.concat(arr2))
</script>
</body>
</html>
```

Output of the code



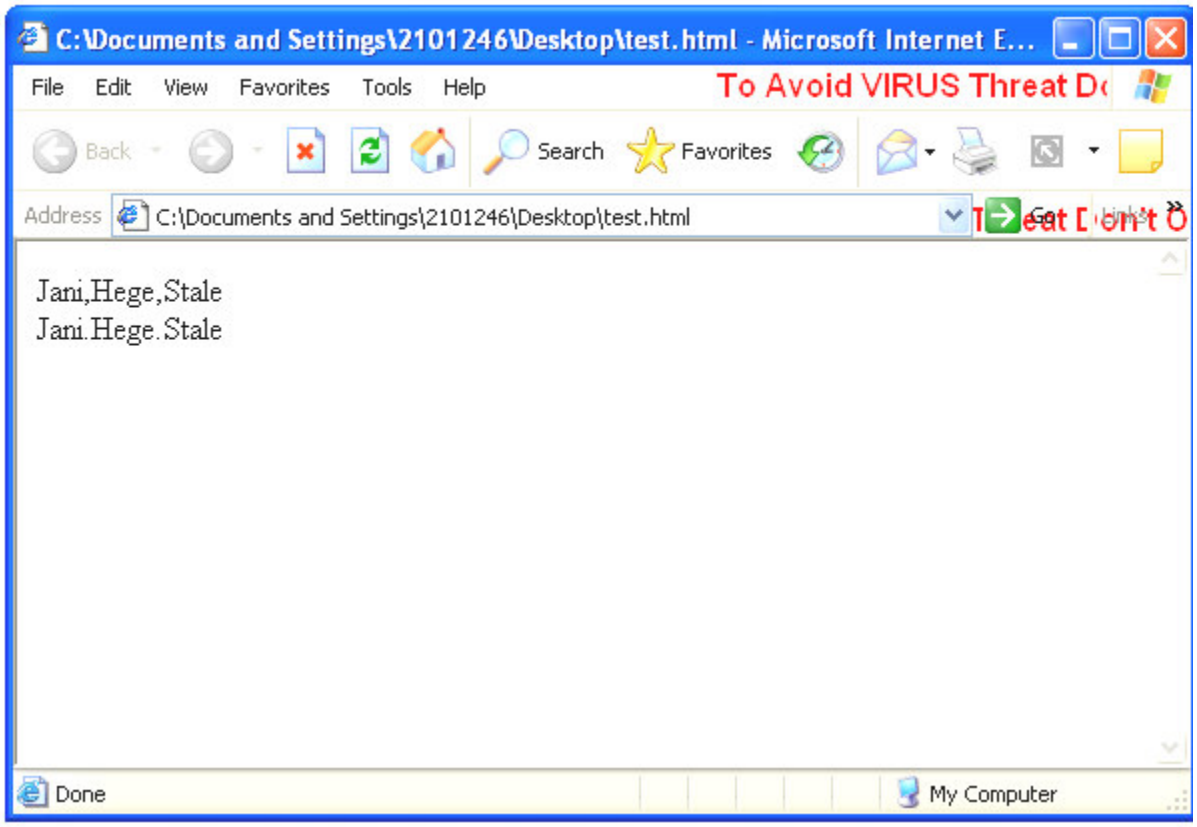
arrayname.join() - join joins all elements of an array into a string.

Example –

```
<html>
<body>
<script language="javascript">
var arr = new Array(3)
arr[0] = "Jani"
arr[1] = "Hege"
arr[2] = "Stale"

document.write(arr.join() + "<br />")
document.write(arr.join("."))
</script>
</body>
</html>
```

Explnation - join in first document.write will join arrayelements by **",",** join in second document.write will join arrayelemts by **"."**

Output of the code –



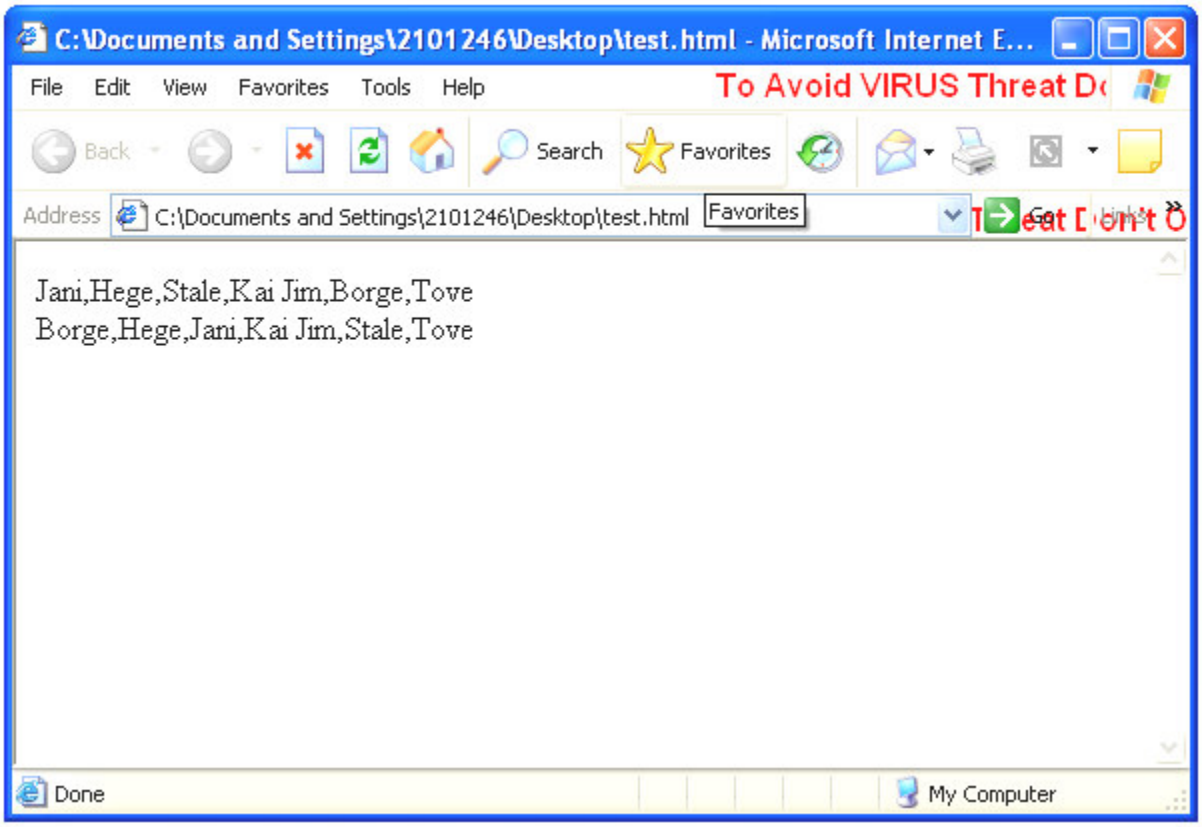
Arrayname.sort() - sort sorts the elements of an array.

Example –

```
<html>
<body>
<script language="javascript">
var arr = new Array(6)
arr[0] = "Jani"
arr[1] = "Hege"
arr[2] = "Stale"
arr[3] = "Kai Jim"
arr[4] = "Borge"
arr[5] = "Tove"
document.write(arr + "<br />")
document.write(arr.sort())
</script>
</body>
</html>
```

Explanation – above example display original on first line and sorted array in second line of a web page.

Output of the code-



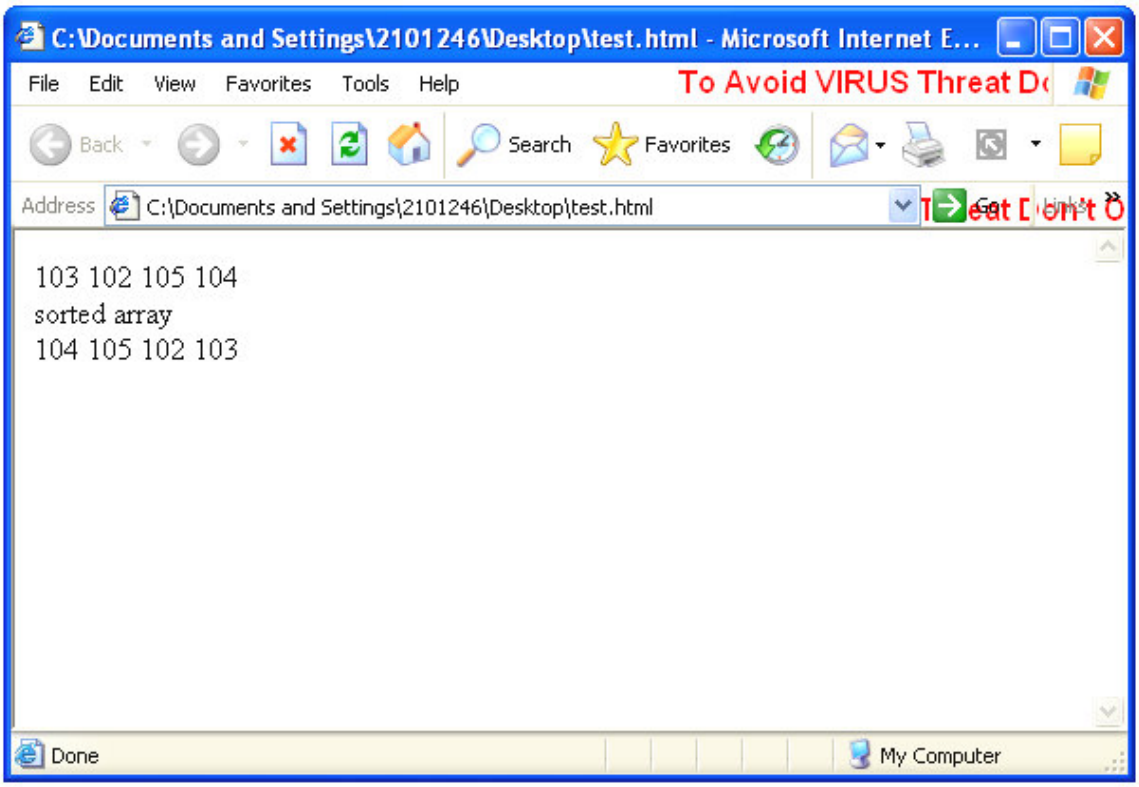
Arrayname.reverse() – reverse transposes the elements of an array: the first array element becomes the last and the last becomes the first.

Example –

```
<html>
<body>
<script language=“javascript”>
var batch=new Array(3)
batch[0]=103;
batch[1]=102;
batch[2]=105;
batch[3]=104;
for(i=0;i<=3;i++)
{
        document.write(batch[i]+" ")
}
document.write("<br>")
document.write("sorted array")
document.write("<br>")

batch.reverse()
for(i=0;i<=3;i++)
```

```
{
        document.write(batch[i]+" ");
}
</script>
</body>
</html>
```

Output of the code-



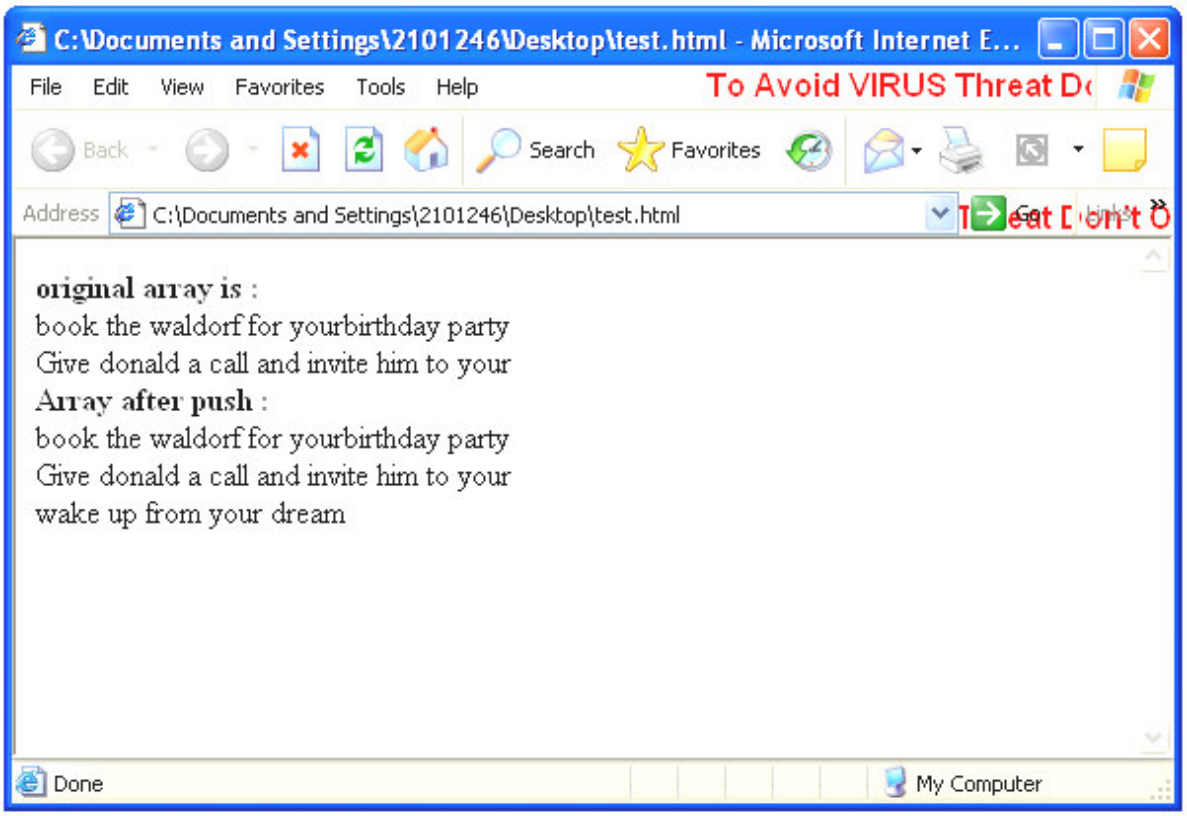
arrayname.push() - push adds one or more elements to the end of an array and returns that last element added.

Example –

```
<html>
<body>
<script language="javascript">
var todolist=new Array()
todolist[0]="book the waldorf for yourbirthday party"
todolist[1]="Give donald a call and invite him to your"
document.write("<b>original array is  : </b>"+"<br>")
for(i=0;i<=1;i++)
{
document.write(todolist[i]+"<br>")
}
todolist.push("wake up from your dream")
document.write("<b>Array after push  :</b> "+"<br>")
for(i=0;i<=2;i++)
{
document.write(todolist[i]+"<br>")
}
```

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

Output of the code –

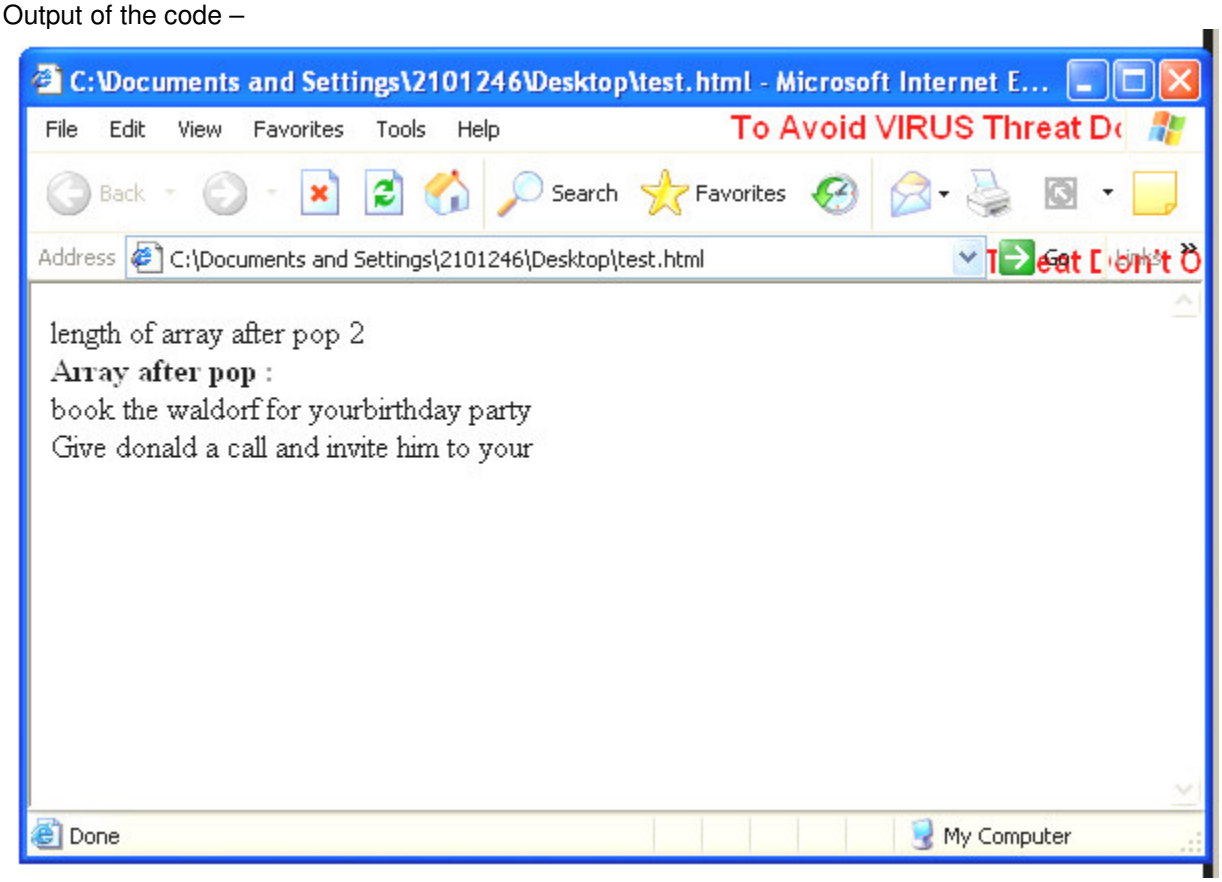


Arrayname.pop() - pop removes the last element from an array and returns that element.

Example-

```
<html>
<body>
<script language="javascript">
var todolist=new Array()
todolist[0]="book the waldorf for yourbirthday party"
todolist[1]="Give donald a call and invite him to your"
todolist[2]="wake up from your dream"
todolist.pop()
document.write("length of array after pop "+todolist.length+"<br>")
document.write("<b>Array after pop  :</b> "+"<br>")
for(i=0;i<todolist.length;i++)
{
document.write(todolist[i]+"<br>")
}
</script>
</body>
</html>
```

Output of the code –

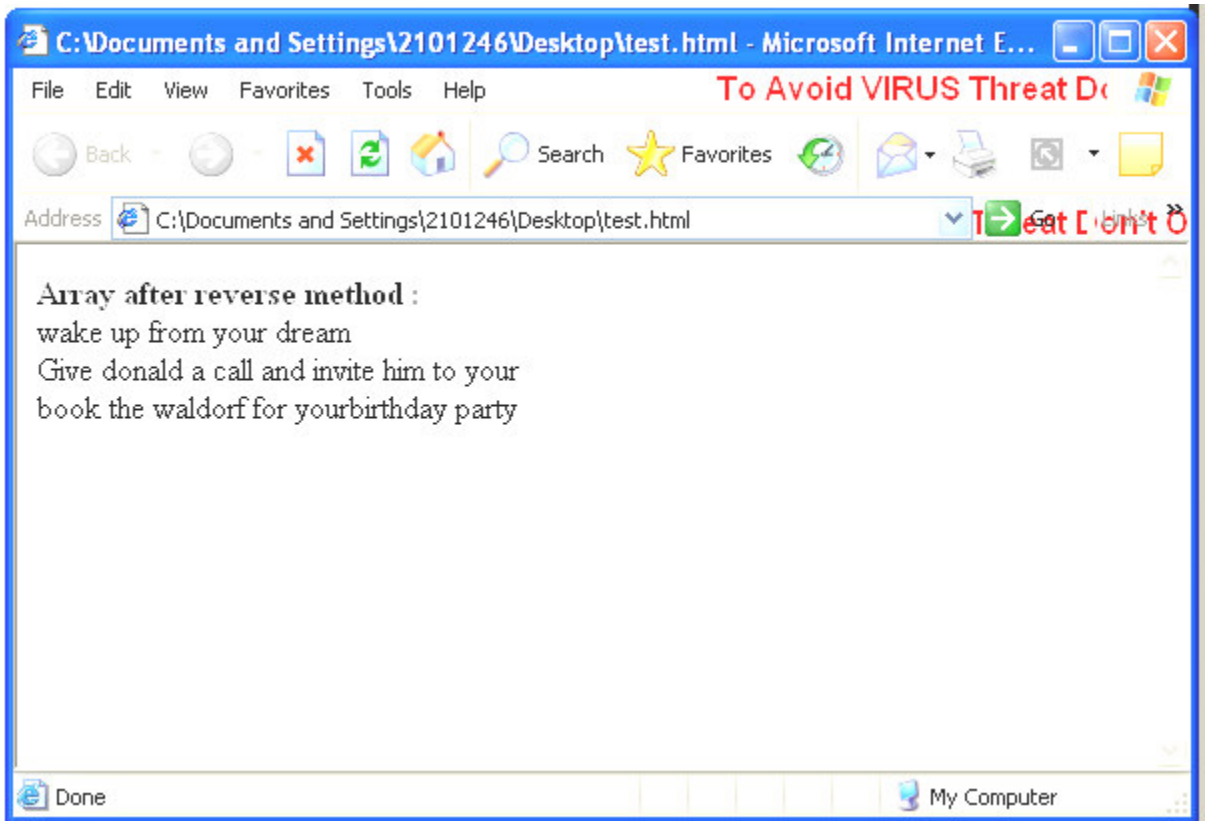


arrayanme.reverse() - reverse transposes the elements of an array: the first array element becomes the last and the last becomes the first.

Example –

```
<html>
<body>
<script language="javascript">
var todolist=new Array()
todolist[0]="book the waldorf for yourbirthday party"
todolist[1]="Give donald a call and invite him to your"
todolist[2]="wake up from your dream"
todolist.reverse()
document.write("<b>Array after reverse method  :</b> "+"<br>")
for(i=0;i<todolist.length;i++)
{
document.write(todolist[i]+"<br>")
}
</script>
</body>
</html>
```

Output of the code –


C:\Documents and Settings\2101246\Desktop\test.html - Microsoft Internet E...
File Edit View Favorites Tools Help
To Avoid VIRUS Threat D
Back
Search
Favorites
Address
C:\Documents and Settings\2101246\Desktop\test.html
Go
Array after reverse method :
wake up from your dream
Give donald a call and invite him to your
book the waldorf for yourbirthday party
Done
My Computer

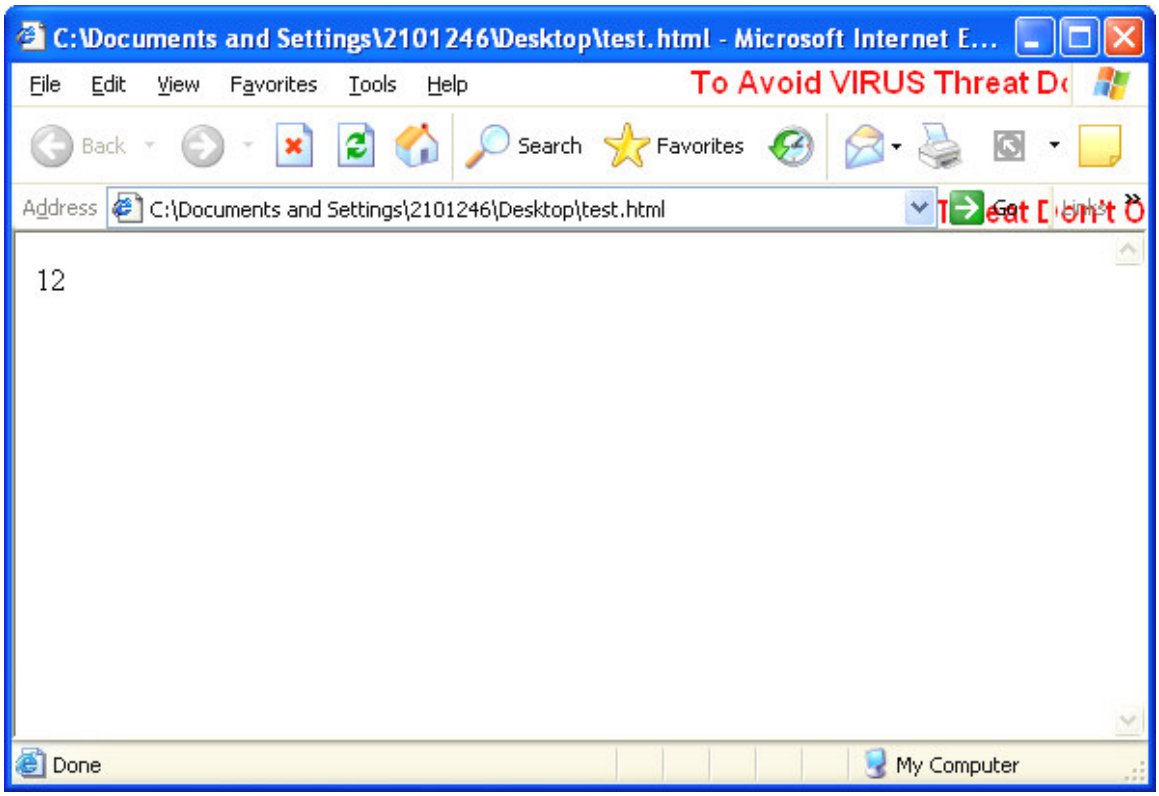
Strings

String Object - The String object is used to manipulate a stored piece of text. A String object has one property, length that indicates the number of characters in the string.

Example –

```
<html>
<body>
<script language=”JavaScript”>
var txt="Hello World!"
document.write(txt.length)
</script>
</body>
</html>
```

Output of the code –



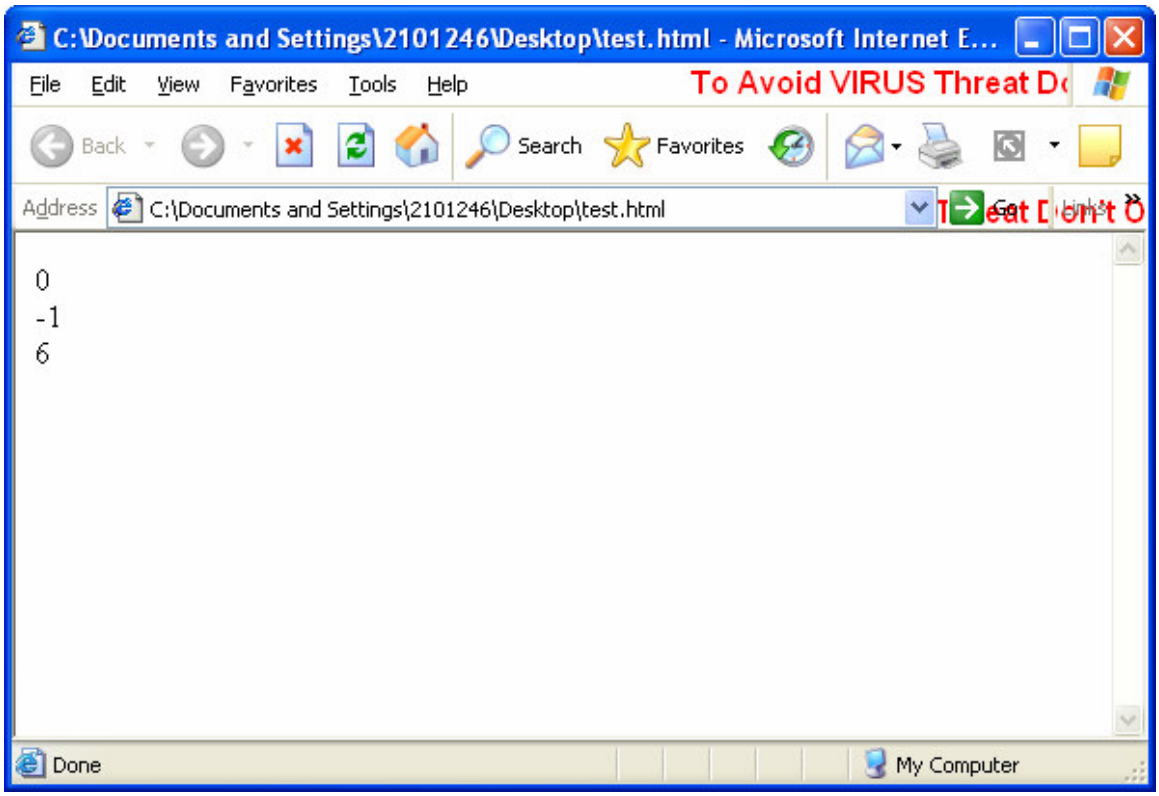
str.indexOf - Return the position of the first occurrence of a text in a

Exmaple –

```
<html>
<body>
<script language=”JavaScript”>
var str="Hello world!"
```

```
        document.write(str.indexOf("Hello") + "<br />")
        document.write(str.indexOf("World") + "<br />")
        document.write(str.indexOf("world"))
</script>
</body>
</html>
```

Output o of the code –

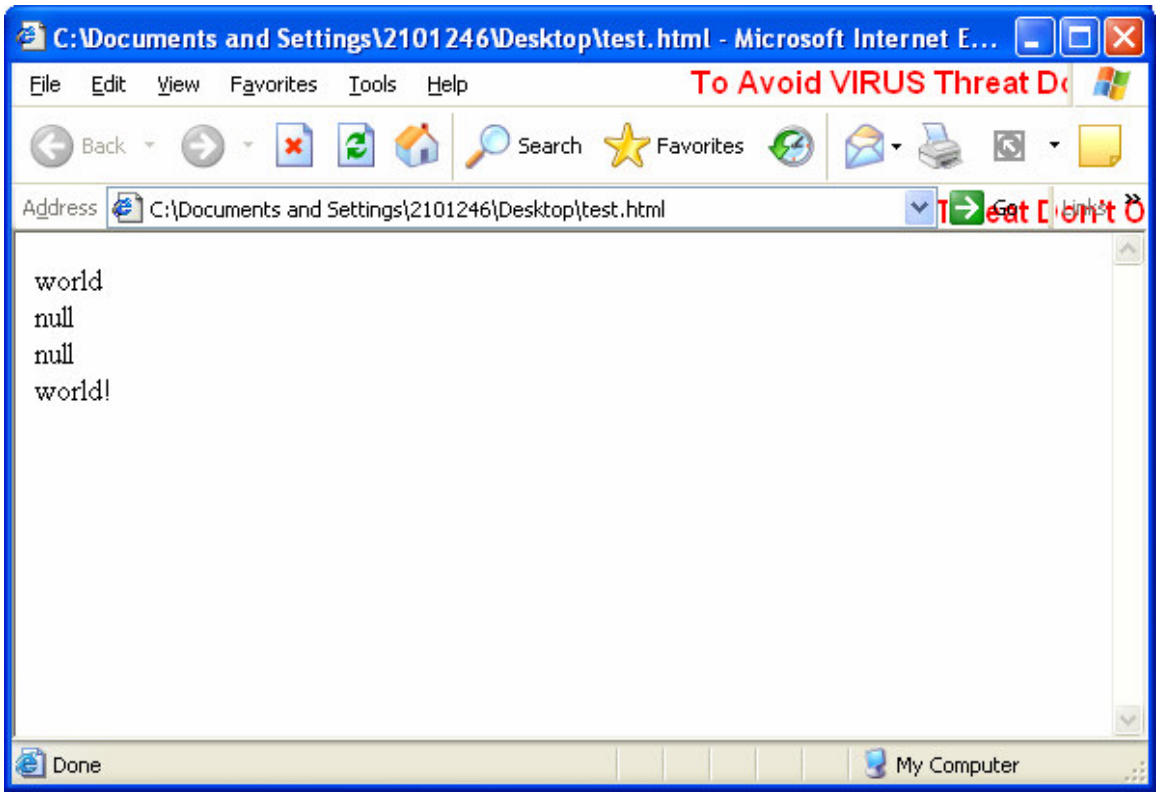


Str.match() - Search for a text in a string and return the text if found - match()

Example –

```
<html>
<body>
<script language=”JavaScript”>
var str="Hello world!"
document.write(str.match("world") + "<br />")
document.write(str.match("World") + "<br />")
document.write(str.match("worlld") + "<br />")
document.write(str.match("world!"))
</script>
</body>
</html>
```

Output of the code –

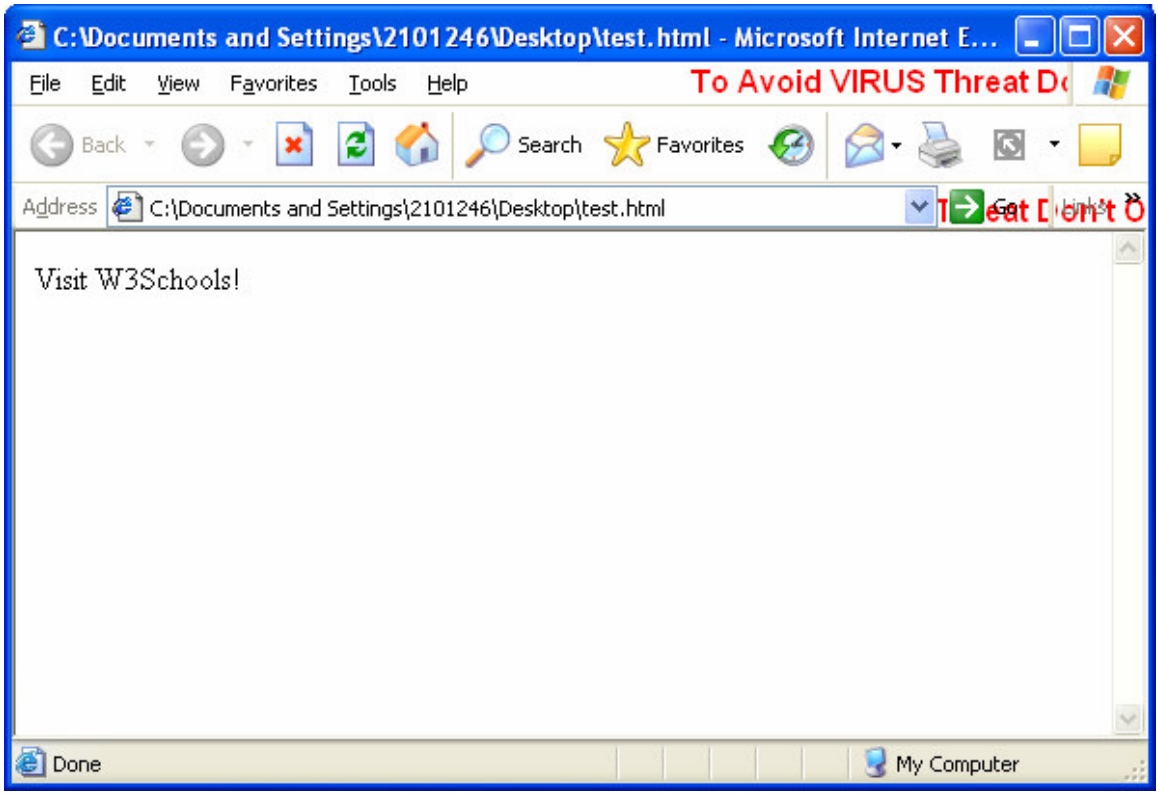


str.replace() – Replace characters in a string - replace()

Example –

```
<html>
<body>
<script language=”JavaScript”>
var str="Visit Microsoft!"
document.write(str.replace("Microsoft","W3Schools"))
</script>
</body>
</html>
```

Output of the code –

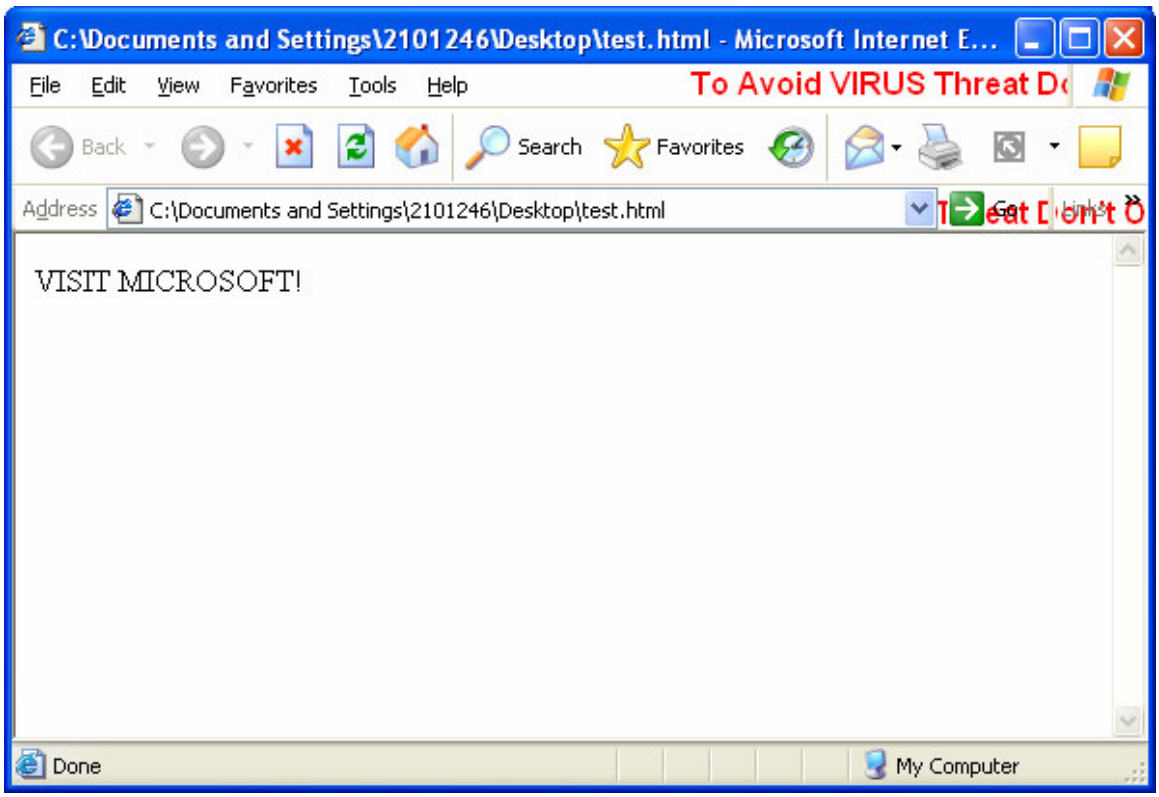


str.toUpperCase – converts a string to uppercase

Example –

```
<html>
<body>
<script language=”JavaScript”>
var str="Visit Microsoft!"
document.write(str.toUpperCase())
</script>
</html>
```

Output of the code –

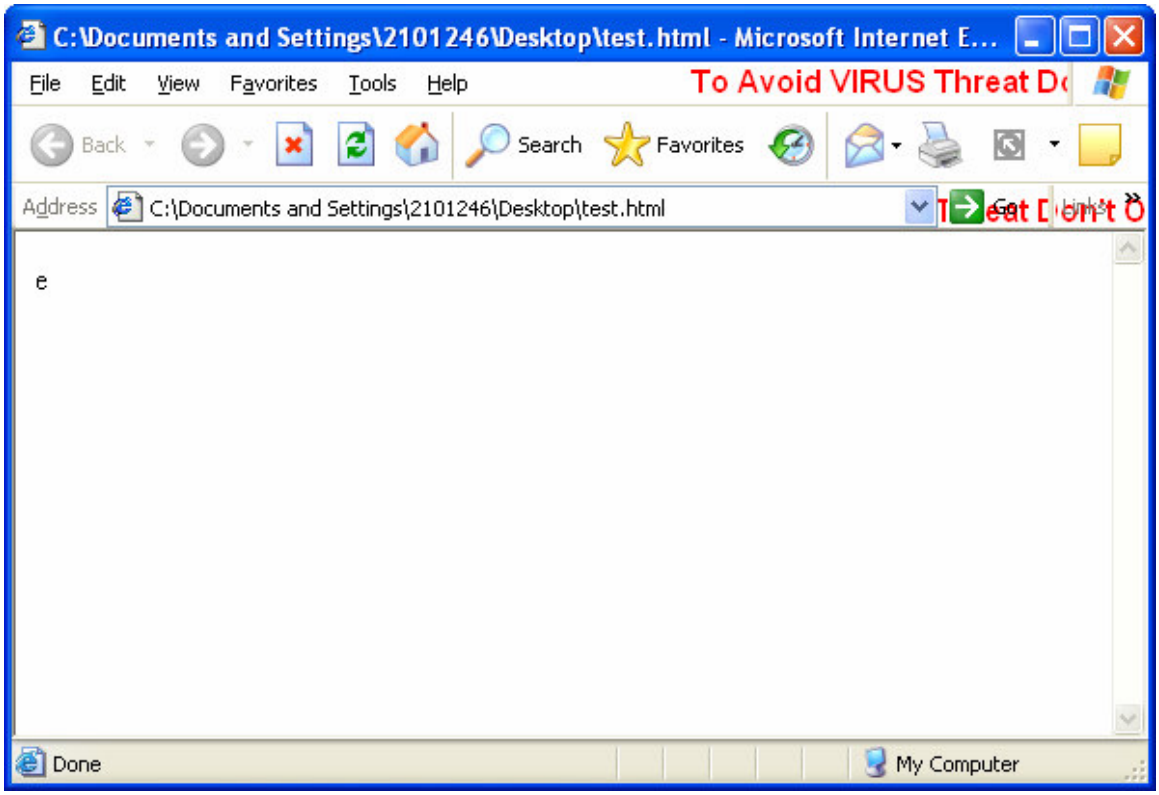


str.charAt() - return the character at position 1:

Example –

```
<html>
<body>
<script language=”JavaScript”>
var str="Hello world!"
document.write(str.charAt(1))
</script>
</body>
</html>
```

Output of the code –



math object

The Math object allows us to perform common mathematical tasks. The Math object includes several mathematical values and functions. No need to define the Math object before using it. The predefined Math object has properties and methods for mathematical constants and functions.

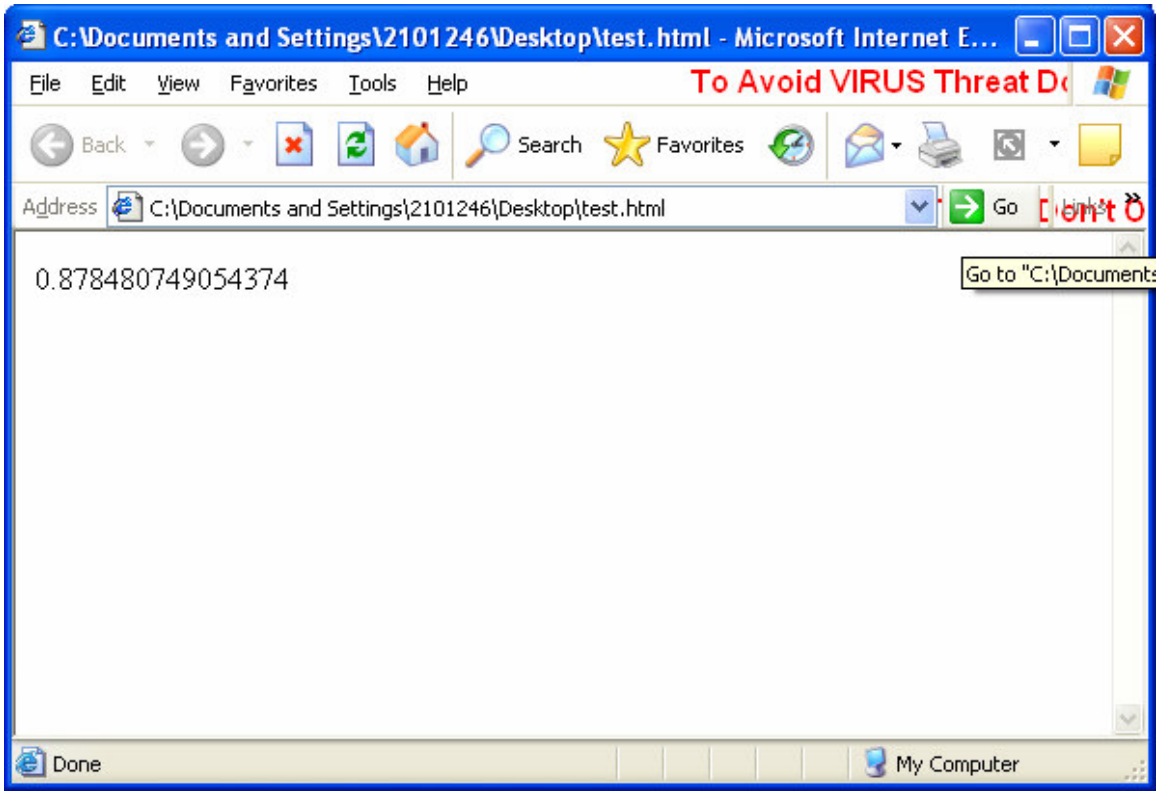
Methods of math object

Math.random() - random() is used to return a random number between 0 and 1.

Example –

```
<html>
<body>
<script type="text/javascript">
document.write(Math.random())
</script>
</body>
</html>
```

Output of the code –



math.round() – use to round number

math.max() - use to return the number with the highest value of two specified numbers.

math.min() - use to return the number with the lowest value of two specified numbers.

math.ceil() – use to return least integer greater than or equal to argument

math.floor() - use to returns greatest integer less than or equal to argument

Test your Progress:

Q. 1 State whether following statement is true or false
This is first element of the product array: product[1]

a) True
b) False

Q. 2 How many elements are there in this array?
Products = new Array('soda','Bread','Pizza')

a) 2
b) 3
c) 4
d) None from the options

Q. 3 What method is used to remove an element from the bottom of an array?

a) push()
b) pop()
c) reverse()
d) shift()

Q. 4 State whether following statement is true or false
The sort() method only places text in sorted order.

a) True
b) False

Q. 5 From the following options which is correct way of declaring date?

a) var today = new Date();
b) var today=new date();
c) var today=Date();
d) today=date()

CHAPTER 19: Functions in JavaScript

Objectives:

- To Understand how to define and call functions
- How to pass parameter trough functions
- How to return values from a function

A function is a reusable code-block that will be executed by an event, or when the function is called.

JavaScript Functions

To keep the browser from executing a script when the page loads, you can put your script into a function.

A function contains code that will be executed 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 can be defined both in the <head> and in the <body> section of a document. However, to assure that the function is read/loaded by the browser before it is called, it could be wise to put it in the <head> section.

Example

```
<html>
<head>
<script language="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 put 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.

You will learn more about JavaScript events in the JS Events chapter.

How to Define a Function

```
function functionname(var1,var2,...,varX)
{
some code
}
```

The syntax for creating a function is:

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.

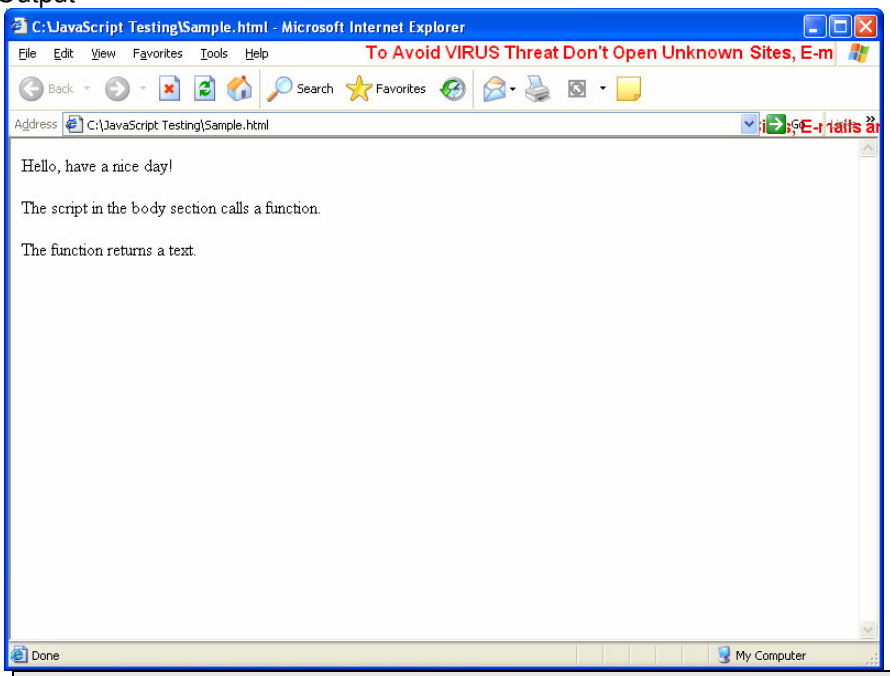
Example: To call a function and pass parameter to function

```
<html>
<head>
<script language="javascript">
function myfunction()
{
        alert("HELLO")
}
</script>
</head>
<body>
<form>
<input type="button" onclick="myfunction()" value="Call function">
</form>

<p>By pressing the button, a function will be called. The function will alert a
message.</p>
</body>
</html>


<html>
<head>
<script language="javascript">
function myfunction(txt)
{
   alert(txt)
}
</script>
</head>
<body>
<form>
<input type="button" onclick="myfunction('Hello')"
value="Call function">
</form>
<p>By pressing the button, a function with an argument will be called. The function
will alert this argument.</p>
</body>
</html>



<head>
<script language="javascript">
function myfunction(txt)
{
   alert(txt)
}
</script>
</head>
<body>
<form>
<input type="button" onclick="myfunction('Good Morning!')"
value="In the Morning">
```

```
<input type="button" onclick="myfunction('Good Evening!')"
value="In the Evening">
</form>
<p>
When you click on one of the buttons, a function will be called. The function will alert
the argument that is passed to it.
</p>
</body>
</html>
```

```
<html>
<head>
<script language="javascript">
function myFunction()
{
    return ("Hello, have a nice day!")
}
</script>
</head>
<body>
<script language="javascript">
      document.write(myFunction())
</script>
<p>The script in the body section calls a function.</p>
<p>The function returns a text.</p>
</body>
</html>
```

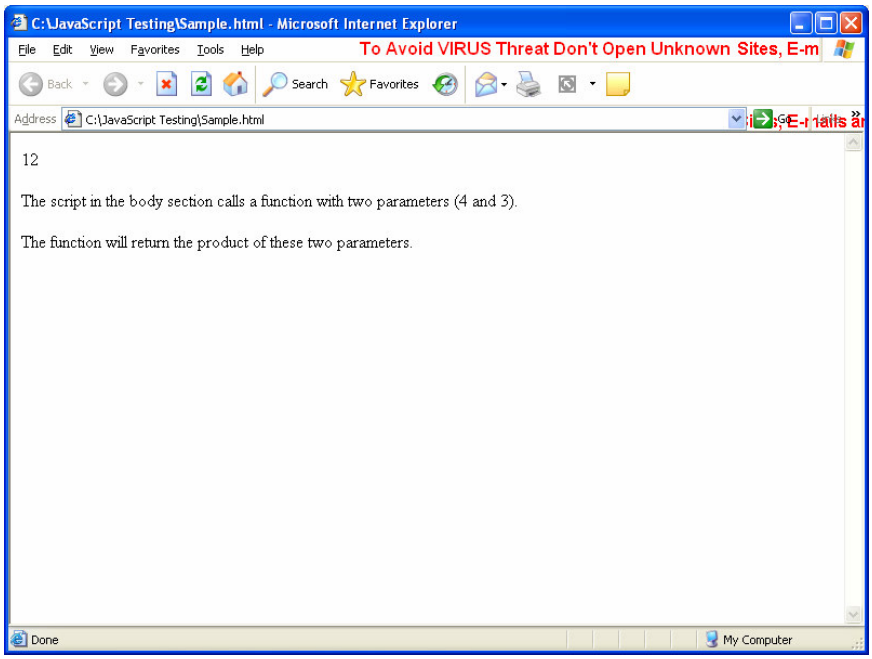
Output



How to let the function find the product of two arguments and return the result.

```
<html>
<head>
<script language="javascript">
function product(a,b)
{
    return a*b
}
</script>
</head>
<body>
<script language="javascript">
      document.write(product(4,3))
</script>
<p>The script in the body section calls a function with two parameters (4 and 3).</p>
<p>The function will return the product of these two parameters.</p>
</body>
</html>
```

Output



Test your Progress:

Q. 1 consider the following JavaScript function:

```
function func(object)
{
for(var i in object)
{
document.write(i;":",object[i],"<br>");
}
}
```

What does the preceding function do?

a) Print all the properties of the obejct along with it's values
b) Prints number from 1 to the number passed as parameter
c) The code will run in infinite loop

Q. 2 consider the following code:

```
<html>
<head>
<script language=""JavaScript"">
function fun()
{
var a=2*(3.14)*this.r;
return a;
}
function c(r)
{
this.r=r;
this.a=fun;
}
var c1=new c(7)
</script>
</head>
<body>
<form>
<input type="button" value="click me" onclick="document.write(c1.a())">
</form>
</body>
</html>
```

What does the preceding code do if user clicks the button?"

a) prints 43.86
b) prints 43.96
c) prints 43.76
d) Code contains error

Q. 3 Consider the following code:

```
function myfunction()
{
var a=10;
var b=5;
var c=8;
result=a/b*c+a-b;
document.write("The result of myfunction :"+result+"<br>");
}
```

What will be the output of the preceding code?

a) 21
b) 5
c) 40
d) 16.5

Q. 4 Which type of variable can be accessed from any function in the code?

a) Numeric variable
b) String variable
c) Boolean variable
d) Global variable

Q. 5 What will the following if statement do?

```
if(i=1)
{
 alert(""Hello"");
}
```

If the variable i has the value 1, it will show an alert.

a) It will always show an alert
b) It will always show an alert
c) It will give an error. The equality comparison operator is ==, not =.
d) None from the options

CHAPTER 20: Document Object Model

Objectives:

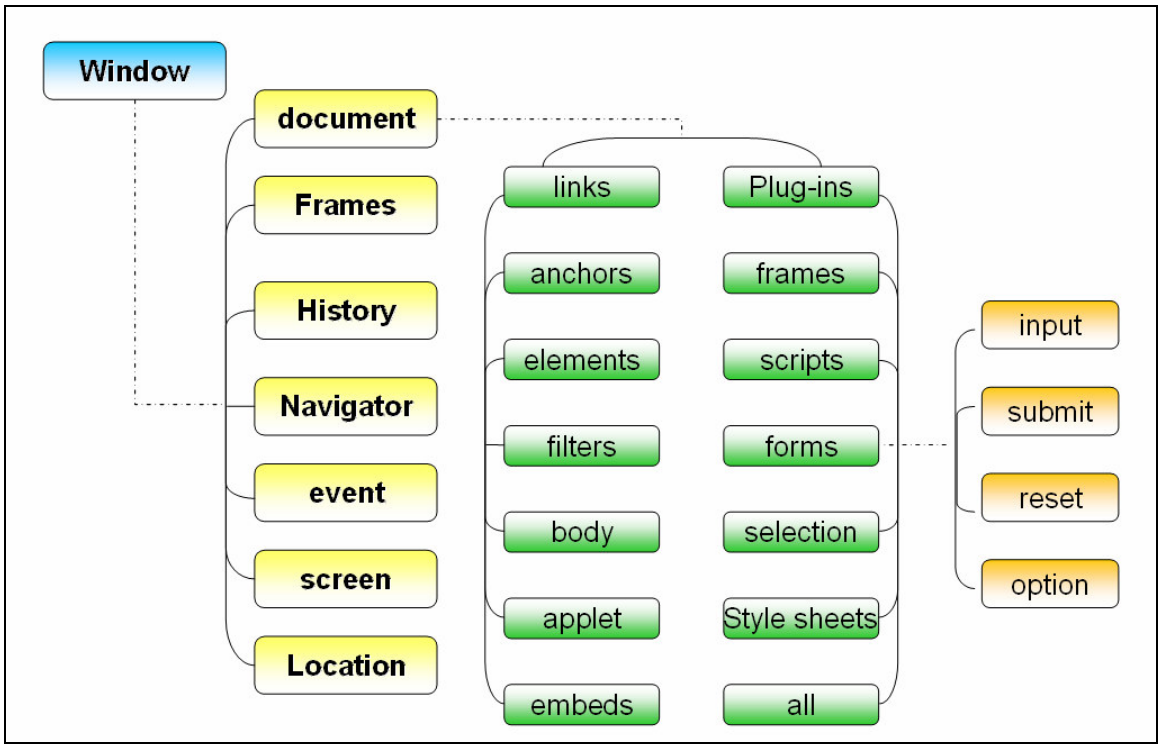
- To describe Document Object Model
- To understand complete Object Model
- To describe how to use window, location, navigator, history, screen and event objects, their methods and properties
- To Use setTimeout() and clearTimeout() functions of window object

What is Document Object Model?

The Document Object Model is a platform- and language-neutral interface that will allow programs and scripts to dynamically access and update the content, structure and style of documents. The HTML Document Object Model (HTML DOM) defines a standard way for accessing and manipulating HTML documents. The DOM presents an HTML document as a tree-structure (a node tree), with elements, attributes, and text. The DOM (Document Object Model) gives generic access to most elements, their styles and attributes in a document To change anything on a page, JavaScript needs access to all elements in the HTML document. This access, along with methods and properties to add, move, change, or remove HTML elements, is given through the Document Object Model (DOM).

According to DOM

- The entire document is a document node
- Every HTML tag is an element node
- The texts contained in the HTML elements are text nodes
- Every HTML attribute is an attribute node



Complete Document Object Model diagram

Window Object

The **top level object** in the JavaScript hierarchy. The Window object represents a browser window. A Window object is created automatically with every instance of a <body> or <frameset> tag. The window object contains everything that is accessible to programs through the object model: the element, frames, images, browser and almost everything that needs to access through the browser.

It contains various method, properties and Events

Method: item, navigator, blur, focus, alert, confirm, setTimeout, clear Timeout

Properties: document, location, history, navigator, event, length, name

Events: onfocus, onload, onunload, onblur, onhelp, onerror

Window.open() - Opens a new browser window

Example –

```
<html>
<head>
<script language="JavaScript">
function winopen()
{
window.open()
}
</script>
</head>
<body>
<input type="button" name="btn" value="open a new window" onclick="winopen()">
</body>
</html>
```

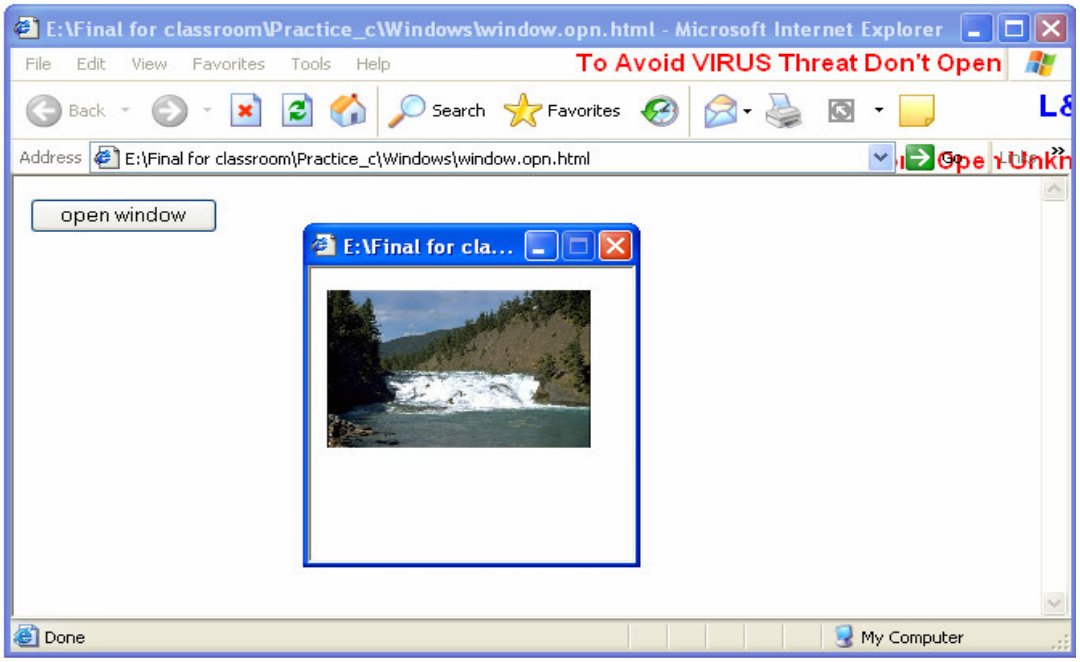
Explanation- In above code when onclick event for button will generate, winopen function will execute and window.open() method will open blank window .

window.open will accept 3 parameter first parameter is name of document or image to be open, second parameter is target of window and third parameter is about features of new window that will open. Third parameter will have properties of window as titlebar, status, tollbar, location, menubar, directories, resizable, height and width, left, top
Example –

```
<html>
<head>
<script language=”JavaScript”>
function open_win()
{
window.open("seen1.jpg","_blank",
"toolbar=0,directories=0,status=0,width=200,height=200,menubar=0,copyhistory=0")
}
</script>
</head>
```

```
<body>
<form>
<input type="button" value="open window" onclick="open_win()">
</form>
</body>
</html>
```

Explanatation – in the above code after clicking on button new window will open which conatin seen1.jpg as image.
Output of the code –



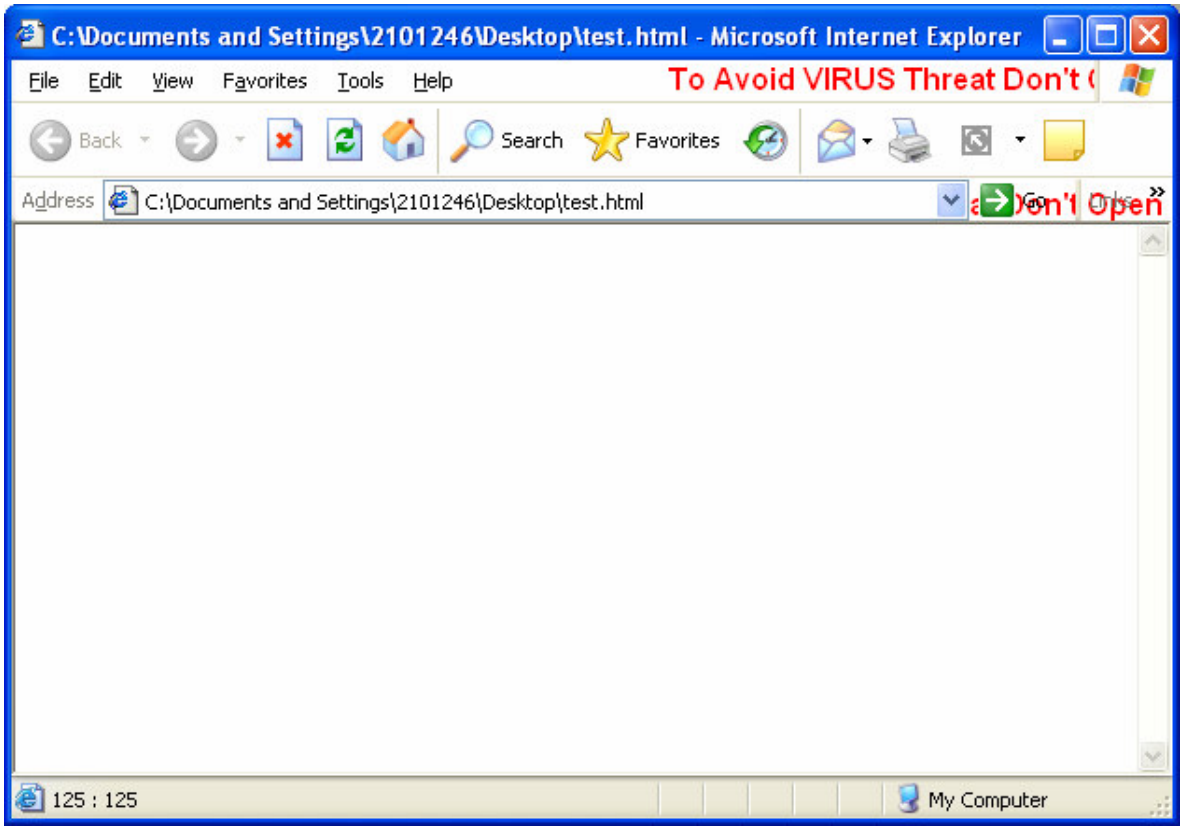
window.close() – Closes the current window

window.status- this property allows to change status bar message on web page.
Example –

```
<html>
<body>
<script type="text/javascript">
function displaycoordIE()
{
window.status=event.clientX+" : "+event.clientY
}
document.onmousemove=displaycoordIE
</script>
</body>
</html>
```

Explanation –
In above example as user move mouse pointer on web page, the status bar displays the current coordinates of the mouse as it moves.

Output of the code –

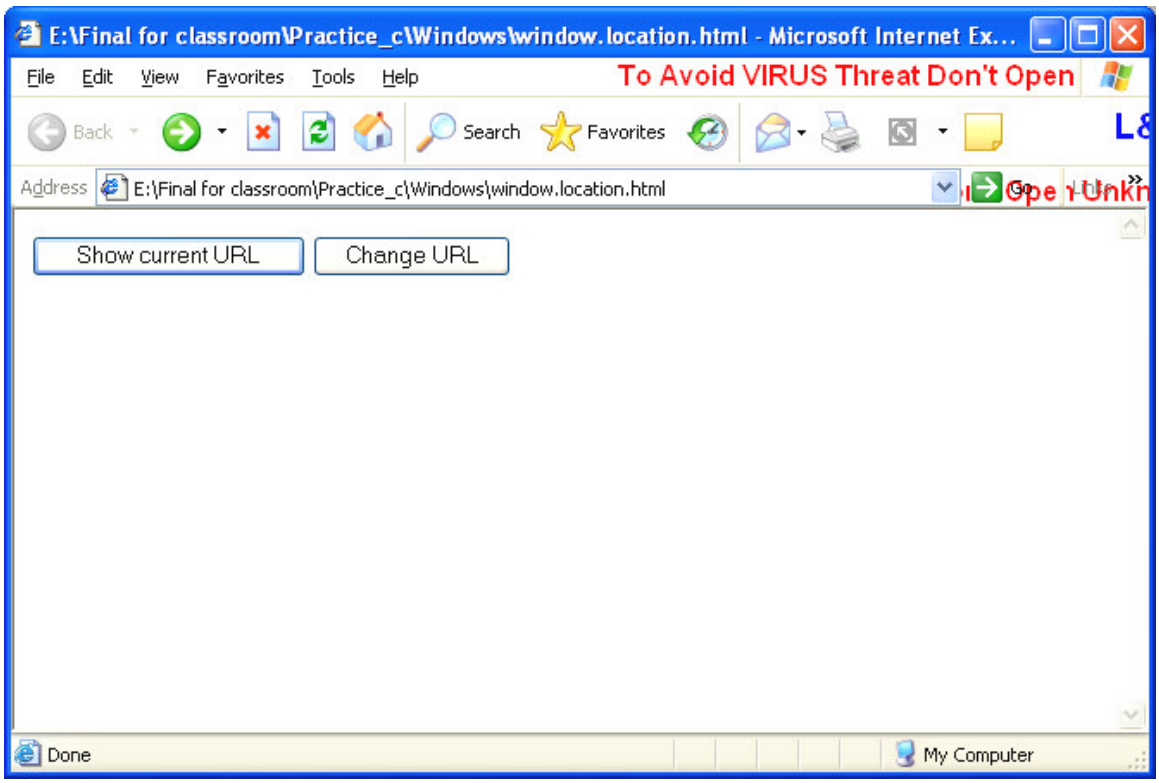


window.location - location is a property of window object. Location property returns current location of a web file. We can transfer location from one web page to another.

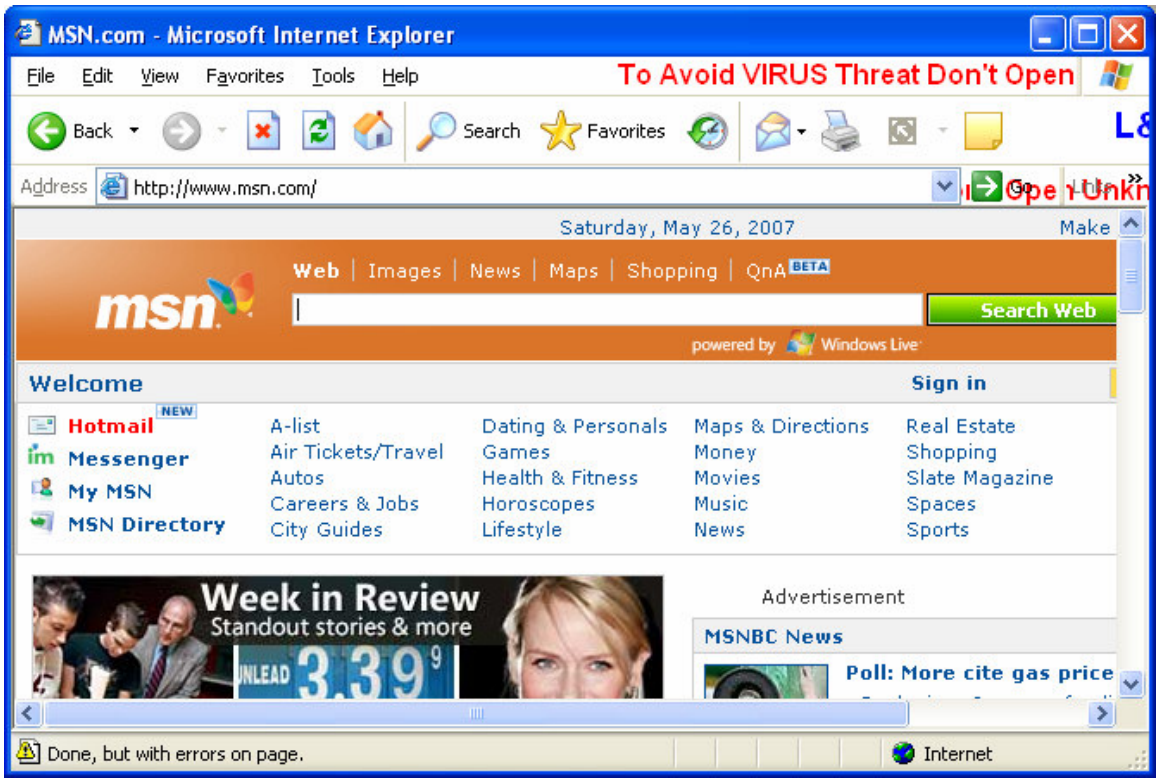
```
<html>
<head>
<script language=”JavaScript”>
function currLocation()
{
alert(window.location)
}
function newLocation()
{
window.location="http://www.msn.com"
}
</script>
</head>
<body>
<input type="button" onclick="currLocation()" value="Show current URL">
<input type="button" onclick="newLocation()" value="Change URL">
</body>
</html>
```

Explanation – on “show current URL” button click we will get current location of file and on “change URL” we will get location of page change to msn.com

Output of the code – on “Show current URL” button click



on “Change URL” button click –



Location object-

The Location object is automatically created by the JavaScript runtime engine and contains information about the current URL. Location object is part of the Window object and is accessed through the window.location property. The location object contains all the information on the location that the window is currently displayed and all the details on that location like port, the protocol.

href-the href property is the entire URL of the current page. Using this property, you can specify the location of the other URL and its equivalent to typing URL in the address bar

host-the hostname is the name of the host machine on which the current URL is located

port-If specific port is specified in the URL, its value is located in the port property

Method: reload() , replace(url)

Location.href –

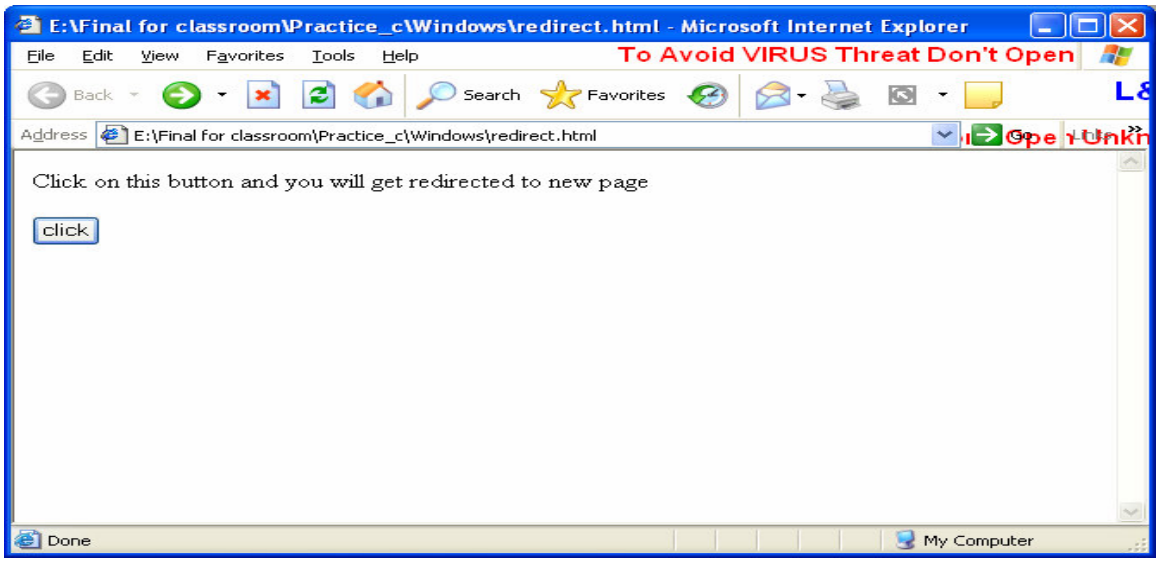
Example –

```
<html>
<head>
<script language=”JavaScript”>
function redirect()
{
 location.href="seen1.jpg"
 }
</script>
</head>
<body>
Click on this button and you will get redirected to new page
<form>
```

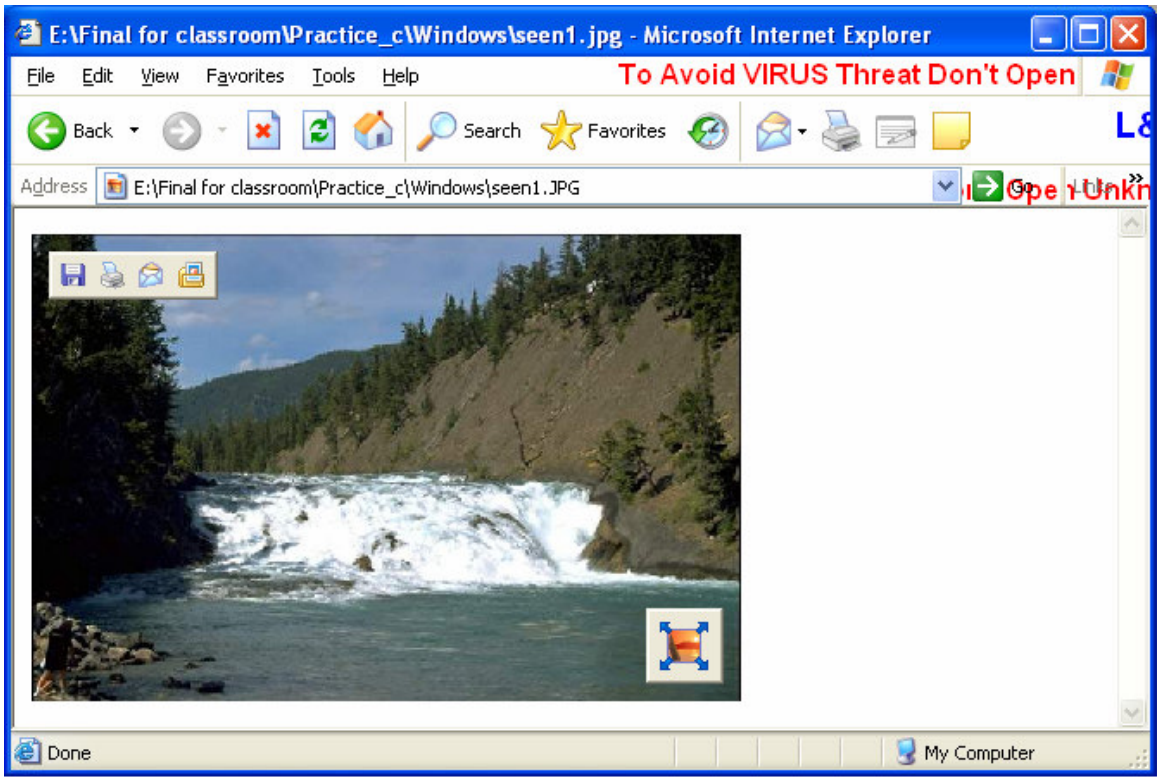
```
<input type=button value=click onclick="redirect()">
</form>
</body>
</html>
```

output of the code - Image file will open in same window as follows:

Original window



Window after button click



window.location.replace()

Example –

```
<html>
<head>
<script language=”JavaScript”>
function replaceDoc()
  {
  window.location.replace("http://www.msn.com")
  }
</script>
</head>
<body>
<input type="button" value="Replace document"
onclick="replaceDoc()" />
</body>
</html>
```

window.location.relaod()

Example –

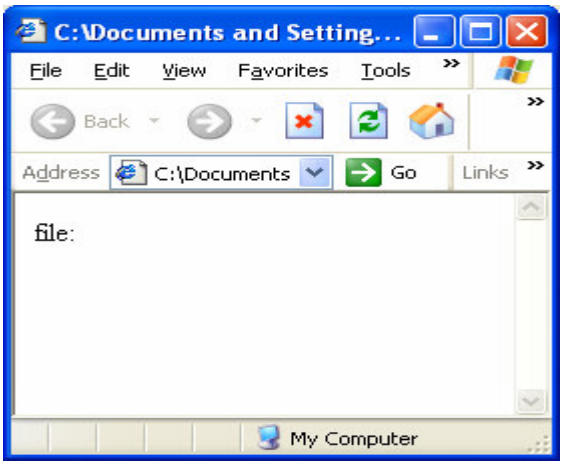
```
<html>
<head>
<script language=”JavaScript”>
function reloadPage()
  {
  window.location.reload()
  }
</script>
</head>
<body>
<input type="button" value="Reload page"
onclick="reloadPage()" />
</body>
</html>
```

location.protocol –

Example -

```
<html>
<body>
<script language=”JavaScript”>
document.write(location.protocol);
</script>
</body>
</html>
```

Output of the code-

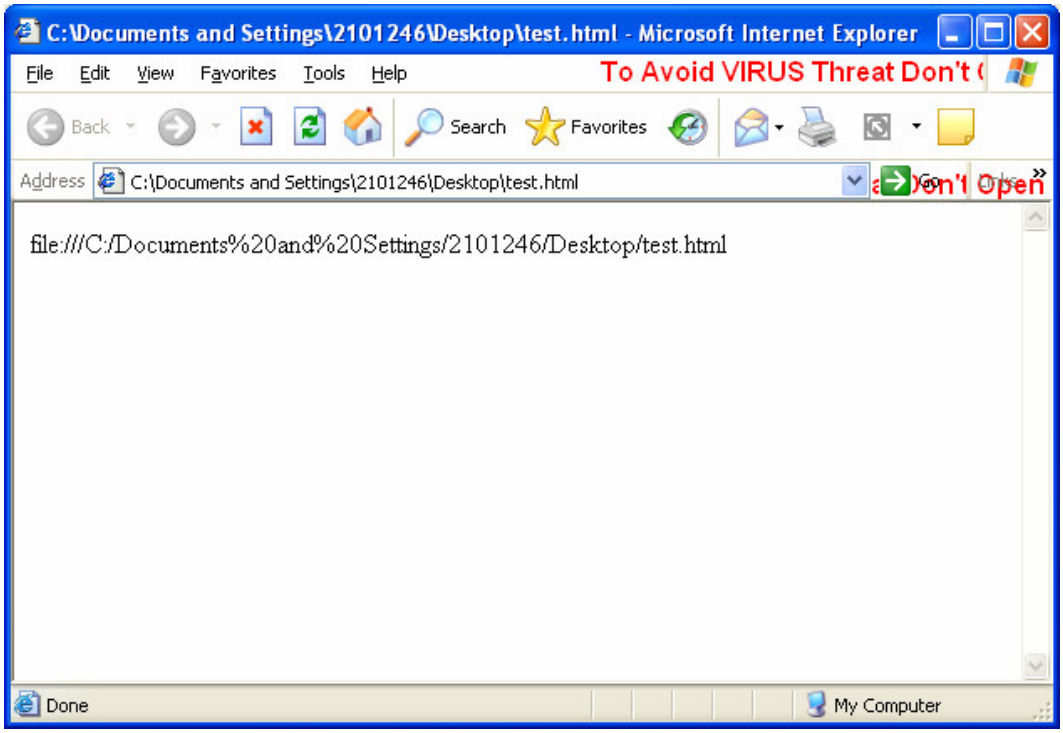


Loation.href- Sets or returns the entire URL

Example -

```
<html>
<body>
<script language="JavaScript">
document.write(location.href);
</script>
</body>
</html>
```

Output of the code -



Navigator- The navigator object enables to access general information about the browser program. – What the browser does and does not support.The Navigator object is automatically created by the JavaScript runtime engine and contains information about the client browser.
appName – returns name of the browser that is processing the script

appVersion – returns version number of the browser

appCodeName -The appCodeName property supplies the application code name, for compatibility reason to show that the browser is compatible with Netscape Navigator, which was the dominant browser for several years.

userAgent -The userAgent property supplies the user agent, which is the exact string that is sent via HTTP as user-agent header when communicating with a web server: browser Language - Returns the current browser language

cookieEnabled - Returns a Boolean value that specifies whether cookies are enabled in the browser

cpuClass - Returns the CPU class of the browser's system

onLine - Returns a Boolean value that specifies whether the system is in offline mode

platform - Returns the operating system platform

systemLanguage - Returns the default language used by the OS

userAgent - Returns the value of the user-agent header sent by the client to the server

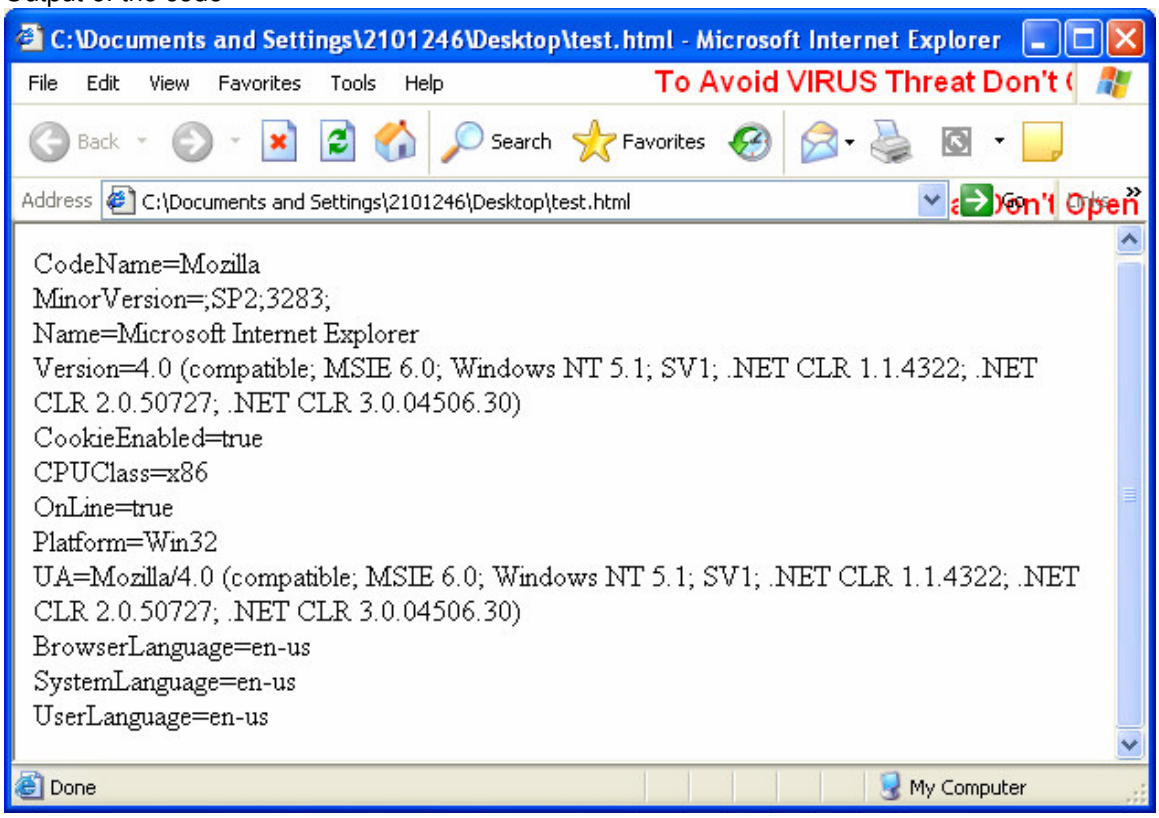
userLanguage - Returns the OS' natural language setting

Example –

```
<html>
<body>
<script language=”JavaScript”>
var x = navigator
document.write("CodeName=" + x.appCodeName)
document.write("<br />")
document.write("MinorVersion=" + x.appMinorVersion)
document.write("<br />")
document.write("Name=" + x.appName)
document.write("<br />")
document.write("Version=" + x.appVersion)
document.write("<br />")
document.write("CookieEnabled=" + x.cookieEnabled)
document.write("<br />")
document.write("CPUClass=" + x.cpuClass)
document.write("<br />")
document.write("OnLine=" + x.onLine)
document.write("<br />")
document.write("Platform=" + x.platform)
document.write("<br />")
document.write("UA=" + x.userAgent)
document.write("<br />")
document.write("BrowserLanguage=" + x.browserLanguage)
document.write("<br />")
```

```
document.write("SystemLanguage=" + x.systemLanguage)
document.write("<br />")
document.write("UserLanguage=" + x.userLanguage)
</script>
</body>
</html>
```

Output of the code –



JavaScript Timing Obejct

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. The two key methods that are used are:

- setTimeout() - executes a code some time in the future
- clearTimeout() - cancels the setTimeout()

Note: The setTimeout() and clearTimeout() are both methods of the HTML DOM Window object.

setTimeout()

Syntax

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.

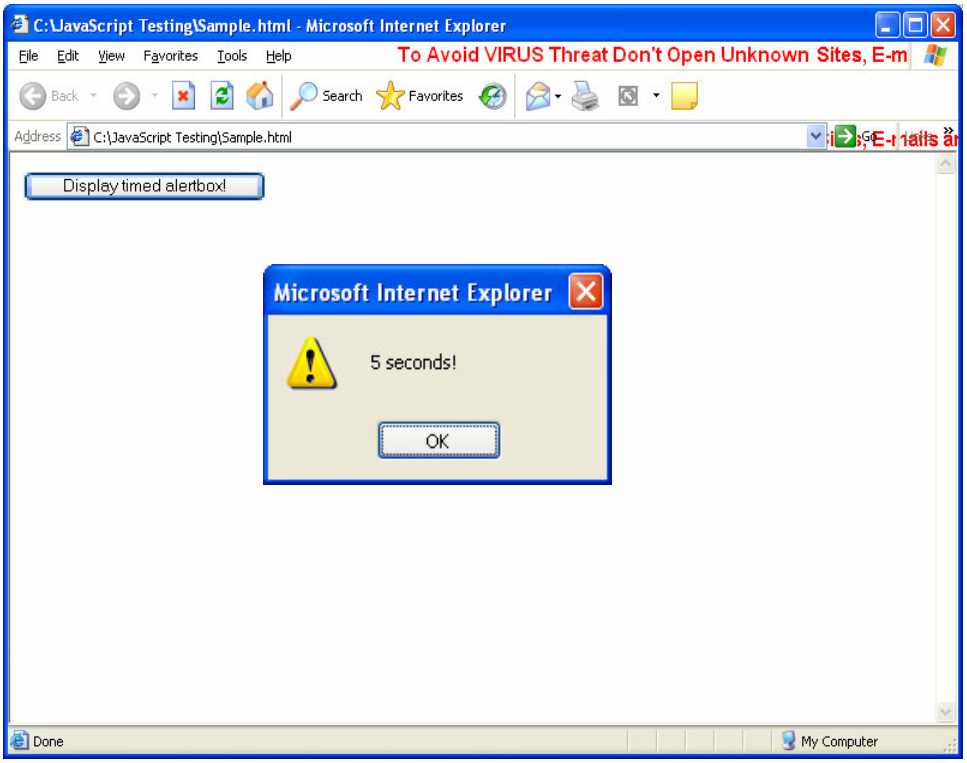
Note: There are 1000 milliseconds in one second.

Example

When the button is clicked in the example below, an alert box will be displayed after 5 seconds.

```
<html>
<head>
<script language="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>
```

Output of the above code will be



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 language="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()

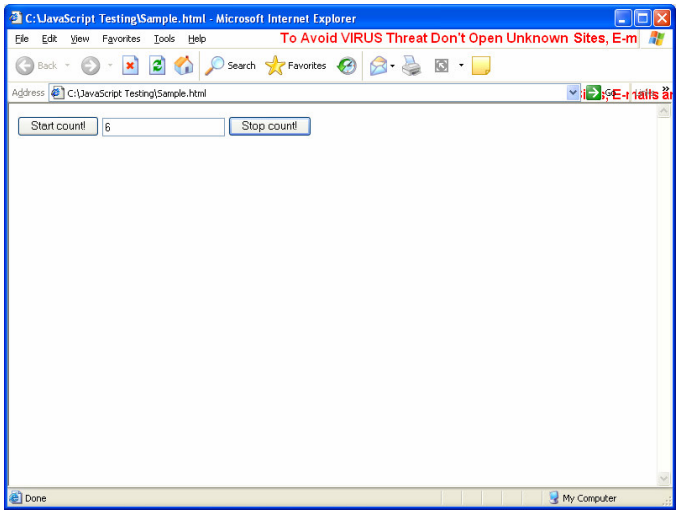
Syntax

```
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 language="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>
```


C:\JavaScript Testing\Sample.html - Microsoft Internet Explorer
File Edit View Favorites Tools Help
To Avoid VIRUS Threat Don't Open Unknown Sites, E-m
Back Search Favorites
Address C:\JavaScript Testing\Sample.html
Start count!
6
Stop count!
Done
My Computer

History Object

The History object is automatically created by the JavaScript runtime engine and consists of an array of URLs. These URLs are the URLs the user has visited within a browser window. The History object is part of the Window object and is accessed through the window.history property.
history.length – The length property specified how many URLs are contained in the current history object. The URLs saved are identical to those shown in browser history list

Example –

```
<html>
<body>
<script language="JavaScript">
document.write(history.length);
</script>
</body>
</html>
```

Methods of history object

history.forward() – Loads the next URL in the history list
history.back() - Loads the previous URL in the history list
history.go(URL or no.) - Loads a specific page in the history list

Screen Object

Screen Object – The Screen object is automatically created by the JavaScript runtime engine and contains information about the client's display screen.

Properties of Screen Object

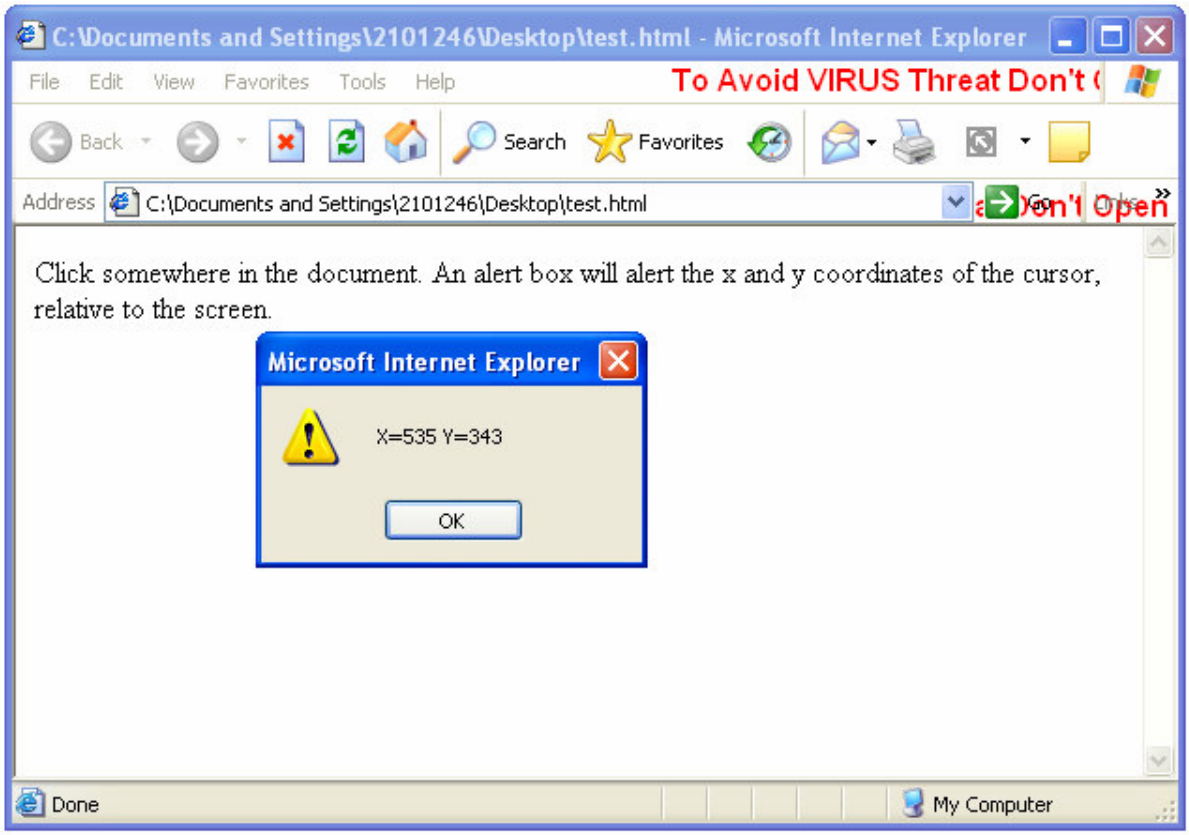
bufferDepth - Sets or returns the bit depth of the color palette in the off-screen bitmap buffer
colorDepth - Returns the bit depth of the color palette on the destination device or buffer
deviceXDPI - Returns the number of horizontal dots per inch of the display screen deviceYDPI - Returns the number of vertical dots per inch of the display screen
fontSmoothingEnabled - Returns whether the user has enabled font smoothing in the display control panel
height - The height of the display screen
logicalXDPI - Returns the normal number of horizontal dots per inch of the display screen
logicalYDPI - Returns the normal number of vertical dots per inch of the display screen
pixelDepth - Returns the color resolution (in bits per pixel) of the display screen
updateInterval - Sets or returns the update interval for the screen
width - Returns width of the display screen

Example –

```
<html>
<head>
<script language="JavaScript">
function coordinates(event)
{
```

```
        x=event.screenX
        y=event.screenY
        alert("X=" + x + " Y=" + y)
}
</script>
</head>
<body onmousedown="coordinates(event)">
<p>
Click somewhere in the document. An alert box will alert the x and y  coordinates of the cursor, relative to the screen.
</p>
</body>
</html>
```

Explanation –
In the above code as soon as user clicks anywhere on document event will generate. Event will call for function, function will calculate x and y coordinates of cursor relative to screen. This value is displayed in alert box.
Output of the code after clicking inside page



Event Object

An event is a browser way of telling user that user is interacting with browser

altKey -this property returns true if the ALT key is pressed when the event was fired , false otherwise

button-the mouse button that had been pressed. It returns 0 if no button was pressed, 1 if left button was pressed, 2 if right button was pressed, and 4 if the middle button was pressed

clientX -The X position of the mouse relative to the client area of the window

clientY -the Y position of the mouse relative to the client area of the window

event.keyCode- keyCode property of event object returns Unicode of the key pressed

ctrKey-this property returns true if the ctr key is pressed when the event was fired , false otherwise

keyCode-the code of the key that was pressed when the event was fired

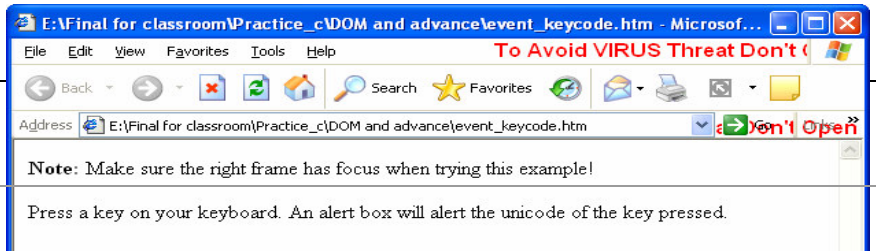
x-The X position or horizontal coordinates of the mouse object when the event was fired to the nearest parent object that was position with CSS positioning

y- The Y position or Vertical coordinates of the mouse object when the event was fired to the nearest parent object that was position with CSS positioning

Example –

```
<html>
<head>
<script language=”JavaScript”>
var i = 0
function whichButton(event)
{
i=event.keyCode;
alert(i);
}
</script>
</head>
<body onkeypress="whichButton(event)">
<p><b>Note:</b> Make sure the right frame has focus when trying this example!</p>
<p>Press a key on your keyboard. An alert box will alert the unicode of the key pressed.</p>
</body>
</html>
```

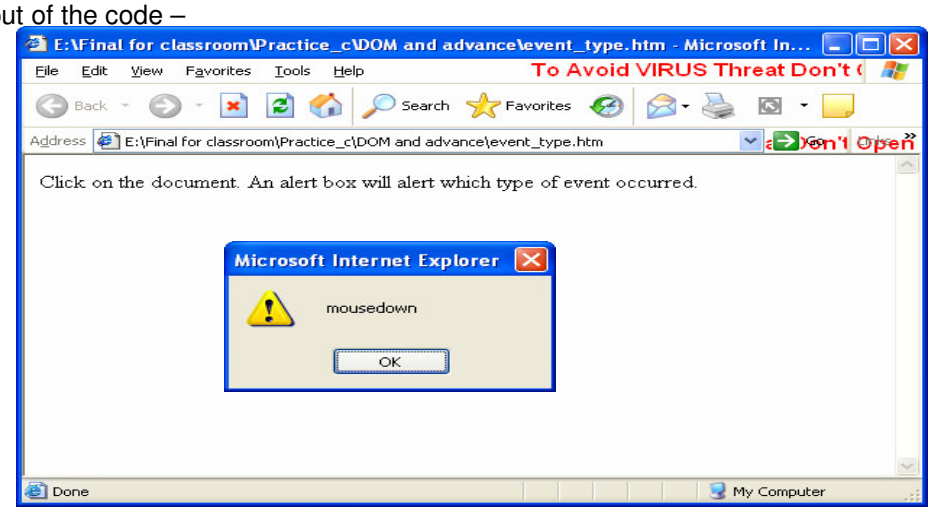
Output of the code – on pressing of key for letter “a”



event.type- returns type of event fired on a web page by user.
Example –

```
<html>
<head>
<script language=”JavaScript”>
function whichType(event)
{
alert(event.type)
}
</script>
</head>
<body onmousedown="whichType(event)">
<p>
Click on the document. An alert box will alert which type of event occurred.
</p>
</body>
</html>
```

Output of the code –

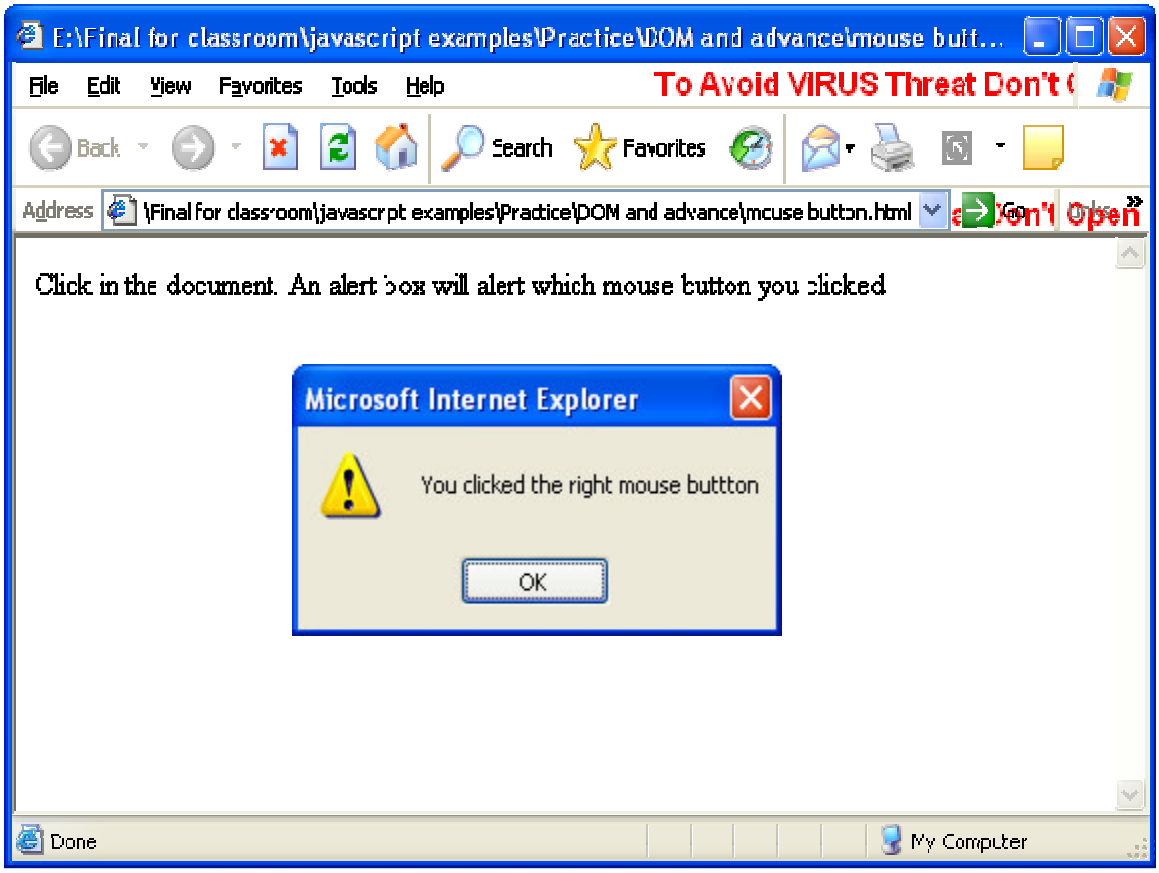


event.button –

Example –

```
<html>
<head>
<script language=”JavaScript”>
function whichButton(event)
{
if(event.button==2)
{
alert("You clicked the right mouse buttton");
}
else
{
alert("You clicked the left mouse button")
}
}
</script>
</head>
<body onmousedown="whichButton(event)">
<p>Click in the document. An alert box will alert which mouse button you clicked</p>
</body>
</html>
```

Output of the code on clicking right mouse button by user



Test your Progress:

Q. 1 State whether following statement is true or false
All images on web page are reflected in the document .image array

a) True
b) False

Q. 2 What object is used to load a web page from within a JavaScript?

a) location
b) upload
c) dnload
d) None from the options

Q. 3 The foreground color of document is a type of

a) Object
b) Method
c) Property
d) Variable

Q. 4 Amongst the following option which is the correct way of changing status bar message?

a) window.status=”My new message”;
b) browser.staus=” My new message”
c) window.staus(My new message);
d) None from the options

Q. 5 Which property of the window object stores the name of the window that contains the active document?

a) frames
b) Name
c) Parent
d) Self

CHAPTER 21: Event handling

Objectives:

- To understand what are events
- To describe various types of events
- To Work with different types of events
- To understand how to implement events on practical applications in a web page

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

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

Now we will see some of the events in JavaScript

onabort Event

Definition and Usage

The onabort event occurs when loading of an image is aborted.
Syntax

```
onabort="SomeJavaScriptCode"
```

Parameter	Description
SomeJavaScriptCode	Required. Specifies a JavaScript to be executed when the event occurs.

Supported by the following HTML tags:

```
<img>
```

Supported by the following JavaScript objects:

```
Image
```

Example 1

In this example an alert box will be displayed if the loading of the image is aborted:

```
<img src="imgName.jpg"
onabort="alert('Error: Loading of the image was aborted')">
```

Example 2

```
<html>
<head>
<script language="javascript">
function abortImage()
{
    alert("Error: Loading of the image was aborted")
}
</script>
</head>
<body>
<img src="gateWay.jpg" onabort="abortImage()">
</body>
</html>
```

In this example we will call a function if the loading of the image is aborted:

onblur Event

Definition and Usage

The onblur event occurs when an object loses focus.

Syntax

```
onblur="SomeJavaScriptCode"
```

Parameter	Description
SomeJavaScriptCode	Required. Specifies a JavaScript to be executed when the event occurs.

Supported by the following HTML tags:

<a>, <acronym>, <address>, <area>, <b>, <bdo>, <big>, <blockquote>, <button>, <caption>, <cite>, <dd>, <del>, <dfn>, , <dl>, <dt>, , <fieldset>, <form>, <frame>, <frameset>, <h1> to <h6>, <hr>, <i>, <iframe>, <img>, <input>, <ins>, <kbd>, <label>, <legend>, <li>, <object>, <ol>, <p>, <pre>, <q>, <samp>, <select>, <small>, , , <sub>, <sup>, <table>, <tbody>, <td>, <textarea>, <tfoot>, <th>, <thead>, <tr>, <tt>, <ul>, <var>

Supported by the following JavaScript objects:

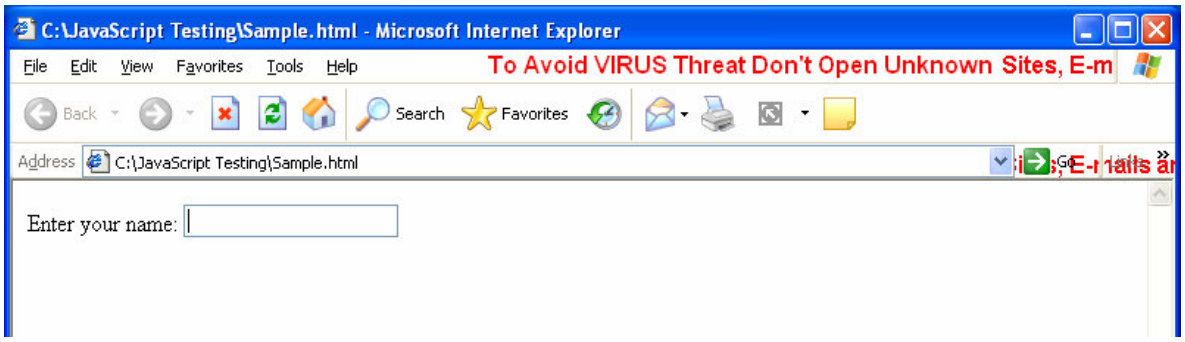
button, checkbox, fileUpload, layer, frame, password, radio, reset, submit, text, textarea, window

Example

In this example we will execute some JavaScript code when a user leaves an input field:

```
<html>
<head>
<script language="javascript">
function upperCase()
{
 var x=document.getElementById("fname").value
document.getElementById("fname").value=x.toUpperCase()
}
</script>
</head>
<body>
Enter your name:
<input type="text" id="fname" onblur="upperCase()">
</body>
</html>
```

The output of the code above will be:



If you enter name in text box and click outside the text box, all characters will be converted into upper case.

onchange Event

Definition and Usage

The onchange event occurs when the content of a field changes.

Syntax

```
onchange="SomeJavaScriptCode"
```

Parameter	Description
SomeJavaScriptCode	Required. Specifies a JavaScript to be executed when the event occurs.

Supported by the following HTML tags:

<input type="text">, <select>, <textarea>

Supported by the following JavaScript objects:

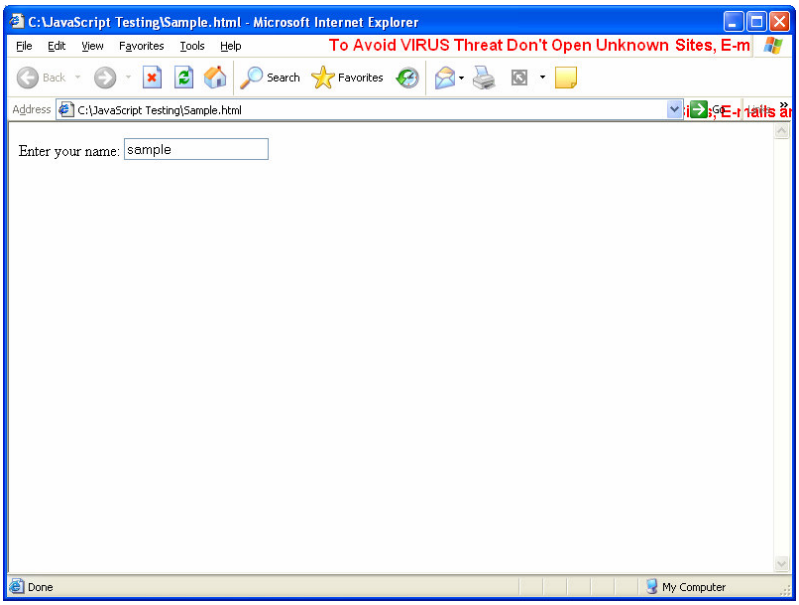
fileUpload, select, text, textarea

Example

In this example we will execute some JavaScript code when a user changes the content of an input field:

```
<html>
<head>
<script language="javascript">
function upperCase(x)
{
var y=document.getElementById(x).value
document.getElementById(x).value=y.toUpperCase()
}
</script>
</head>
<body>
Enter your name:
<input type="text" id="fname" onchange="upperCase(this.id)">
</body>
</html>
```

The output of the code above will be:



Onclick Event

Definition and Usage
The onclick event occurs when an object gets clicked.

Syntax

```
onclick="SomeJavaScriptCode"
```

Parameter	Description
SomeJavaScriptCode	Required. Specifies a JavaScript to be executed when the event occurs.

Supported by the following HTML tags:

<a>, <address>, <area>, <b>, <bdo>, <big>, <blockquote>, <body>, <button>, <caption>, <cite>, <code>, <dd>, <dfn>, , <dl>, <dt>, , <fieldset>, <form>, <h1> to <h6>, <hr>, <i>, <img>, <input>, <kbd>, <label>, <legend>, <li>, <map>, <object>, <ol>, <p>, <pre>, <samp>, <select>, <small>, , , <sub>, <sup>, <table>, <tbody>, <td>, <textarea>, <tfoot>, <th>, <thead>, <tr>, <tt>, <ul>, <var>

Supported by the following JavaScript objects:

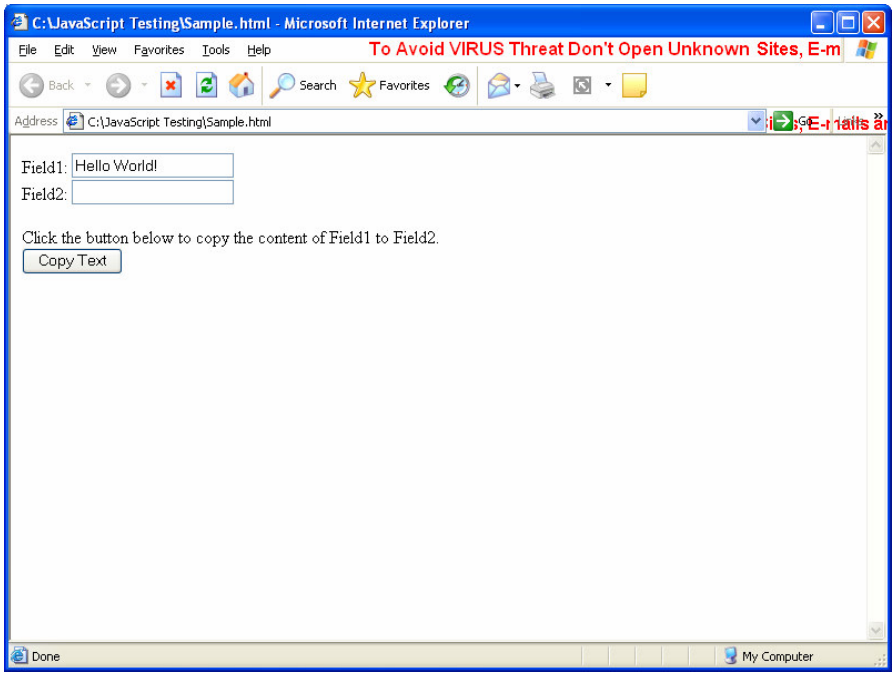
button, document, checkbox, link, radio, reset, submit

Example

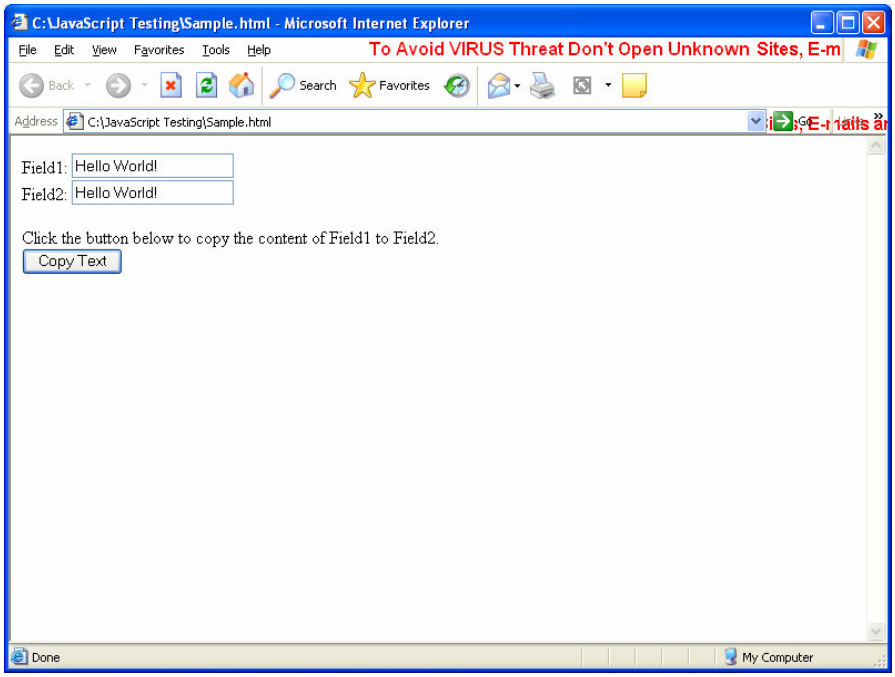
In this example the text in the first input field will be copied to the second input field when a button is clicked:

```
<html>
<body>
Field1: <input type="text" id="field1" value="Hello World!">
<br />
Field2: <input type="text" id="field2">
<br /><br />
Click the button below to copy the content of Field1 to Field2.
<br />
<button onclick="document.getElementById('field2').value=
document.getElementById('field1').value">Copy Text</button>
</body>
</html>
```

The output of the code above will be:



As soon as we click on “Copy Text” button It will give output as


C:\JavaScript Testing\Sample.html - Microsoft Internet Explorer
File Edit View Favorites Tools Help
To Avoid VIRUS Threat Don't Open Unknown Sites, E-m
Back Search Favorites
Address C:\JavaScript Testing\Sample.html
Field1: Hello World!
Field2: Hello World!
Click the button below to copy the content of Field1 to Field2.
Copy Text
Done
My Computer

onerror Event

Definition and Usage

The onerror event is triggered when an error occurs loading a document or an image.

Syntax

```
onerror="SomeJavaScriptCode"
```

Parameter	Description
SomeJavaScriptCode	Required. Specifies a JavaScript to be executed when the event occurs.

Supported by the following HTML tags:

```
<img>, <object>, <style>
```

Supported by the following JavaScript objects:

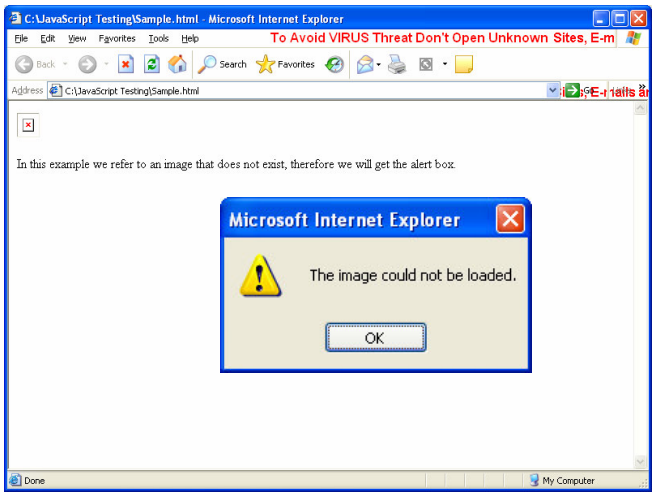
```
window, image
```

Example

In this example an alert box will be displayed if an error occurs when loading an image:

```
<html>
<body>
<img src="image.gif"
onerror="alert('The image could not be loaded.')">
<p>In this example we refer to an image that does not exist, therefore we will get the
alert box.</p>
</body>
</html>
```

output :



onfocus Event

Definition and Usage

The onfocus event occurs when an object gets focus.

Syntax

```
onfocus="SomeJavaScriptCode"
```

Parameter	Description
SomeJavaScriptCode	Required. Specifies a JavaScript to be executed when the event occurs.

Supported by the following HTML tags:

<a>, <acronym>, <address>, <area>, <b>, <bdo>, <big>, <blockquote>, <button>, <caption>, <cite>, <dd>, <del>, <dfn>, , <dl>, <dt>, , <fieldset>, <form>, <frame>, <frameset>, <h1> to <h6>, <hr>, <i>, <iframe>, <img>, <input>, <ins>, <kbd>, <label>, <legend>, <li>, <object>, <ol>, <p>, <pre>, <q>, <samp>, <select>, <small>, , , <sub>, <sup>, <table>, <tbody>, <td>, <textarea>, <tfoot>, <th>, <thead>, <tr>, <tt>, <ul>, <var>

Supported by the following JavaScript objects:

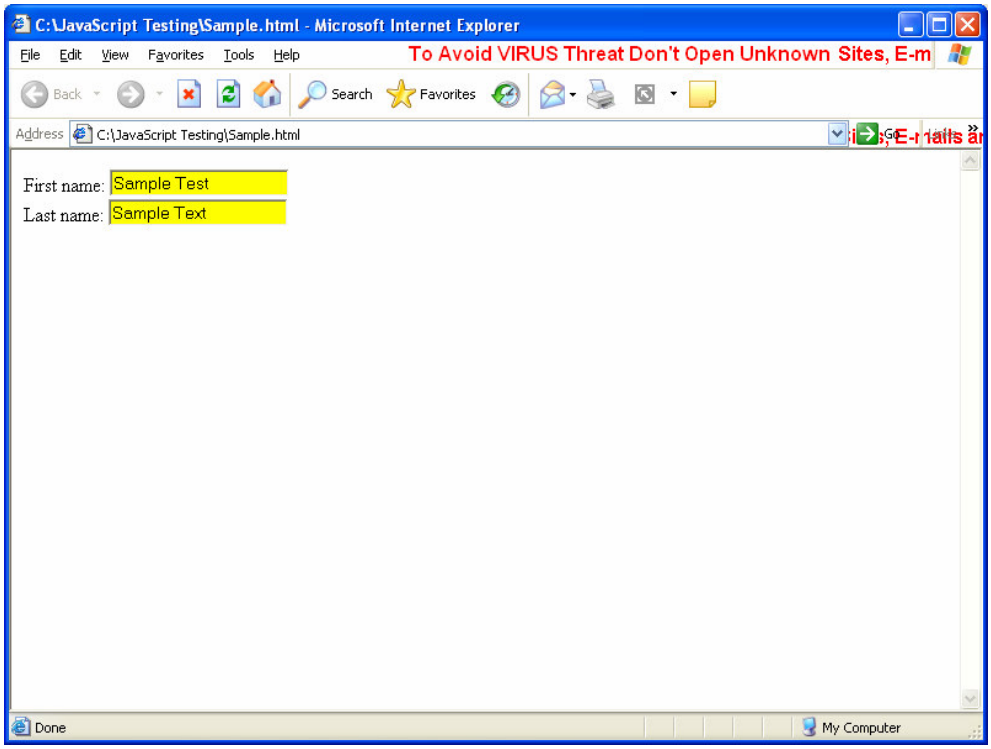
button, checkbox, fileUpload, layer, frame, password, radio, reset, select, submit, text, textarea, window

Example

In this example the background color of the input fields change when they get focus:

```
<html>
<head>
<script language="javascript">
function setStyle(x)
{
    document.getElementById(x).style.background="yellow"
}
</script>
</head>
<body>
First name: <input type="text" onfocus="setStyle(this.id)" id="fname">
<br/>
Last name: <input type="text" onfocus="setStyle(this.id)" id="lname">
</body>
</html>
```

The output of the code above will be:



As soon as user clicks inside the text box, color of text box will change to “yellow”.

onkeydown Event

Definition and Usage

The onkeydown event occurs when a keyboard key is pressed.

Syntax

```
onkeydown="SomeJavaScriptCode"
```

Parameter	Description
SomeJavaScriptCode	Required. Specifies a JavaScript to be executed when the event occurs.

Supported by the following HTML tags:

<a>, <acronym>, <address>, <area>, <b>, <bdo>, <big>, <blockquote>, <body>, <button>, <caption>, <cite>, <code>, <dd>, <del>, <dfn>, , <dt>, , <fieldset>, <form>, <h1> to <h6>, <hr>, <i>, <input>, <kbd>, <label>, <legend>, <li>, <map>, <object>, <ol>, <p>, <pre>, <q>, <samp>, <select>, <small>, , , <sub>, <sup>, <table>, <tbody>, <td>, <textarea>, <tfoot>, <th>, <thead>, <tr>, <tt>, <ul>, <var>

Supported by the following JavaScript objects:

document, image, link, textarea

Tips and Notes

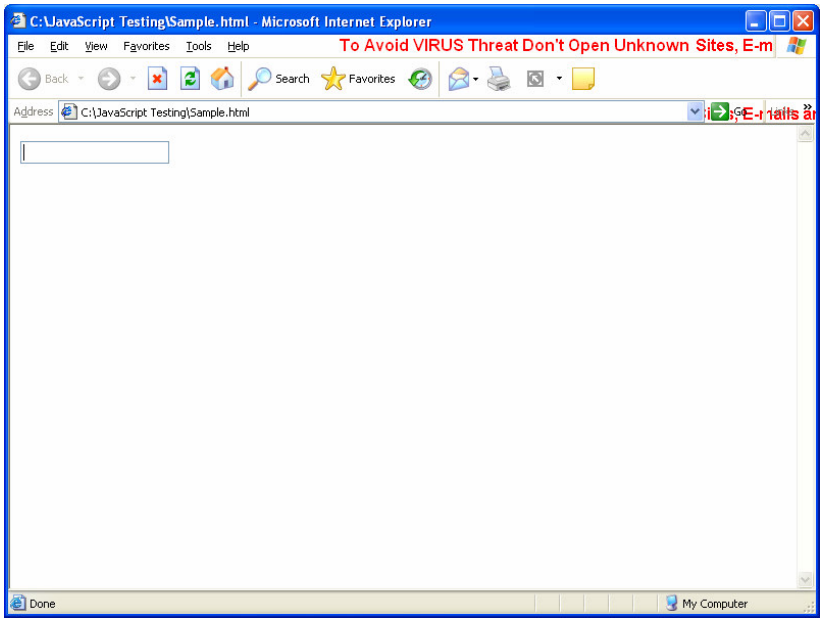
Browser differences: Internet Explorer uses event.keyCode to retrieve the character that was pressed and Netscape/Firefox/Opera uses event.which.

Example

In this example the user cannot type numbers into the input field:

```
<html>
<body>
<script language="javascript">
function noNumbers(e)
{
var keynum
var keychar
var numcheck
if(window.event)
{
keynum = e.keyCode
}
keychar = String.fromCharCode(keynum)
numcheck = /\d/
return !numcheck.test(keychar)
}
</script>
<form>
<input type="text" onkeydown="return noNumbers(event)" />
</form>
</html>
```

The output of the code above will be: Here User cannot enter number in input box.



onkeypress Event

Definition and Usage

The onkeydown event occurs when a keyboard key is pressed or held down.

Syntax

```
onkeypress="SomeJavaScriptCode"
```

Parameter	Description
SomeJavaScriptCode	Required. Specifies a JavaScript to be executed when the event occurs.

Supported by the following HTML tags:

<a>, <acronym>, <address>, <area>, <b>, <bdo>, <big>, <blockquote>, <body>, <button>, <caption>, <cite>, <code>, <dd>, <del>, <dfn>, , <dt>, , <fieldset>, <form>, <h1> to <h6>, <hr>, <i>, <input>, <kbd>, <label>, <legend>, <li>, <map>, <object>, <ol>, <p>, <pre>, <q>, <samp>, <select>, <small>, , , <sub>, <sup>, <table>, <tbody>, <td>, <textarea>, <tfoot>, <th>, <thead>, <tr>, <tt>, <ul>, <var>

Supported by the following JavaScript objects:

document, image, link, textarea

Tips and Notes

Browser differences: Internet Explorer uses event.keyCode to retrieve the character that was pressed and Netscape/Firefox/Opera uses event.which.

Example

In this example the user cannot type numbers into the input field:

```
<html>
<body>
<script language="javascript">
function noNumbers(e)
{
var keynum
var keychar
var numcheck
if(window.event)
{
keynum = e.keyCode
}
keychar = String.fromCharCode(keynum)
numcheck = /\d/
return !numcheck.test(keychar)
}
</script>
<form>
<input type="text" onkeypress="return noNumbers(event)" />
</form>
</html>
```

onKeyUp Event

Definition and Usage

The onkeyup event occurs when a keyboard key is released.

Syntax

```
onkeyup="SomeJavaScriptCode"
```

Parameter	Description
SomeJavaScriptCode	Required. Specifies a JavaScript to be executed when the event occurs.

Supported by the following HTML tags:

<a>, <acronym>, <address>, <area>, <b>, <bdo>, <big>, <blockquote>, <body>, <button>, <caption>, <cite>, <code>, <dd>, <del>, <dfn>, , <dt>, , <fieldset>, <form>, <h1> to <h6>, <hr>, <i>, <input>, <kbd>, <label>, <legend>, <li>, <map>, <object>, <ol>, <p>, <pre>, <q>, <samp>, <select>, <small>, , , <sub>, <sup>, <table>, <tbody>, <td>, <textarea>, <tfoot>, <th>, <thead>, <tr>, <tt>, <ul>, <var>

Supported by the following JavaScript objects:

document, image, link, textarea

Example

When typing letters in the input field in the following example, the letters will change to uppercase (one by one):

```
<html>
<head>
<script language="javascript">
function upperCase(x)
{
    var y=document.getElementById(x).value
    document.getElementById(x).value=y.toUpperCase()
}
</script>
</head>
<body>
Enter your name: <input type="text"
id="fname" onkeyup="upperCase(this.id)">
</body>
</html>
```

onload Event

Definition and Usage

The onload event occurs immediately after a page or an image is loaded.

Syntax

```
onload="SomeJavaScriptCode"
```

Parameter	Description
SomeJavaScriptCode	Required. Specifies a JavaScript to be executed when the event occurs.

Supported by the following HTML tags:

```
<body>, <frame>, <frameset>, <iframe>, <img>, <link>, <script>
```

Supported by the following JavaScript objects:

```
image, layer, window
```

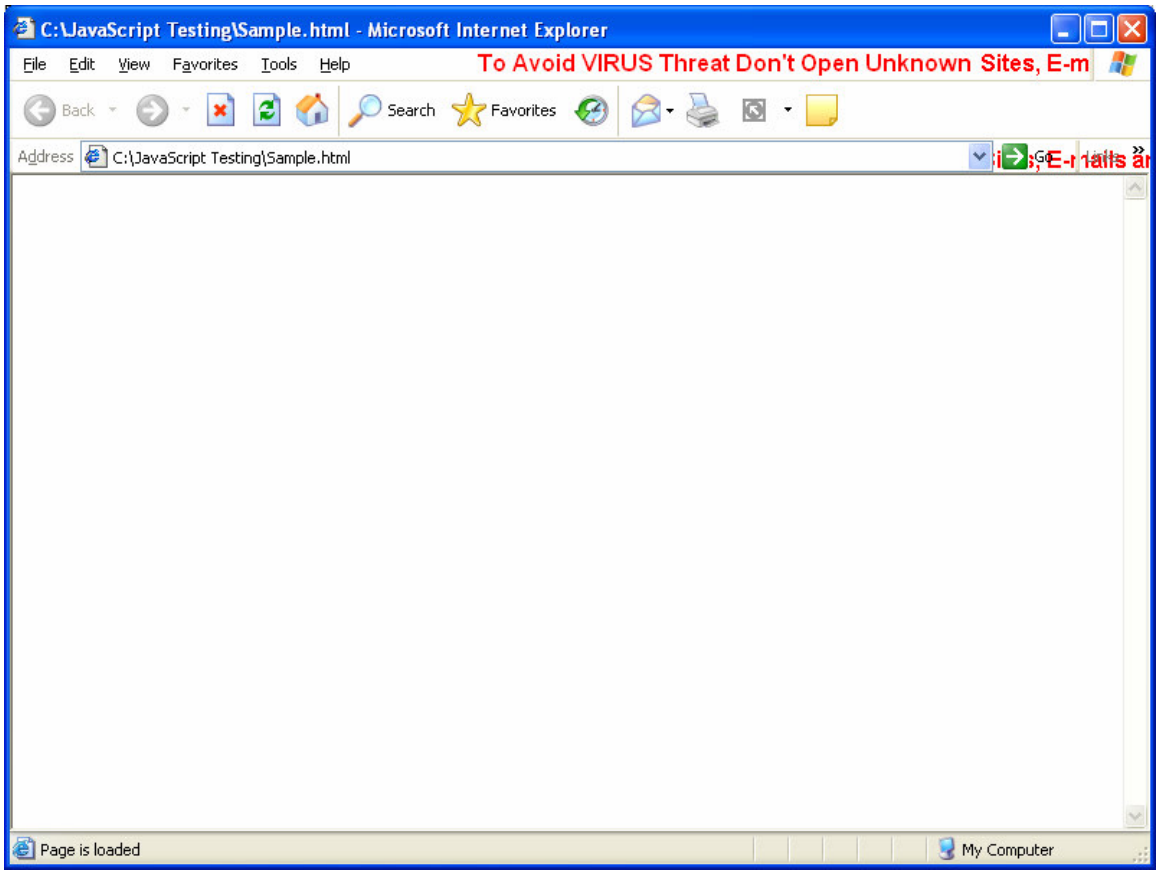
Example

In this example the text "Page is loaded" will be displayed in the status bar:

```
<html>
<head>
<script language="javascript">
```

```
function load()
{
    window.status="Page is loaded"
}
</script>
</head>
<body onload="load()">
</body>
</html>
```

Output: Look towards status bar to see the effect.



onmousedown Event

Definition and Usage

The onmousedown event occurs when a mouse button is clicked.

Syntax

```
onmousedown="SomeJavaScriptCode"
```

Parameter	Description
SomeJavaScriptCode	Required. Specifies a JavaScript to be executed when the event occurs.

Supported by the following HTML tags:

```
<a>, <address>, <area>, <b>, <bdo>, <big>, <blockquote>, <body>, <button>, <caption>, <cite>, <code>, <dd>, <dfn>, <div>, <dl>, <dt>, <em>, <fieldset>, <form>, <h1> to <h6>, <hr>, <i>, <img>, <input>, <kbd>, <label>, <legend>, <li>, <map>, <ol>, <p>, <pre>, <samp>, <select>, <small>, <span>, <strong>, <sub>, <sup>, <table>, <tbody>, <td>, <textarea>, <tfoot>, <th>, <thead>, <tr>, <tt>, <ul>, <var>
```

Supported by the following JavaScript objects:

```
button, document, link
```

Example 1

In this example an alert box is displayed when clicking on the picture:

```
<img src="imageXYZ.jpg"
onmousedown="alert('You clicked the picture!')">
```

Example 2

In this example an alert box will alert the tag name of the element you clicked on:

```
<html>
<head>
<script language="javascript">
function whichElement(e)
{
    var targ
    if (!e) var e = window.event
    if (e.target) targ = e.target
    else if (e.srcElement) targ = e.srcElement
    if (targ.nodeType == 3) // defeat Safari bug
    targ = targ.parentNode
    var tname
    tname=targ.tagName
    alert("You clicked on a " + tname + " element.")
}
</script>
</head>
<body onmousedown="whichElement(event)">
<h2>This is a header</h2>
<p>This is a paragraph</p>
<img border="0" src="hrs.gif" alt="Ball">
</body>
</html>
```

onselect event

Definition and Usage

The onselect event occurs when text is selected in a text or textarea field.

Syntax

```
onselect="SomeJavaScriptCode"
```

Parameter	Description
SomeJavaScriptCode	Required. Specifies a JavaScript to be executed when the event occurs.

Supported by the following HTML tags:

```
<input type="text">, <textarea>
```

Supported by the following JavaScript objects:

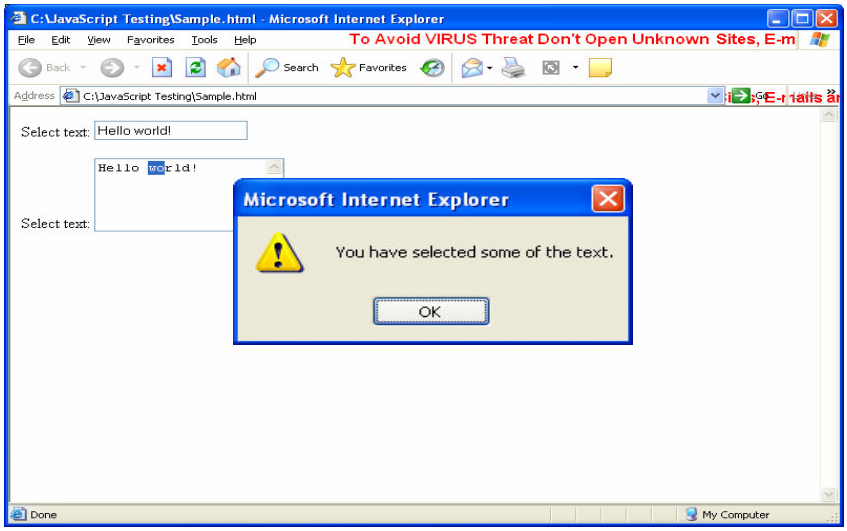
```
text, textarea
```

Example

In this example an alert box will be displayed if some of the text is selected:

```
<html>
<body>
<form>
Select text:
<input type="text" value="Hello world!"
onselect="alert('You have selected some of the text.')">
<br><br>
Select text:
<textarea cols="20" rows="5"
onselect="alert('You have selected some of the text.')">
Hello world!
</textarea>
```

The output of the code above will be:



onsubmit event

Definition and Usage

The onsubmit event occurs when the submit button in a form is clicked.

Syntax

```
onsubmit="SomeJavaScriptCode"
```

Parameter	Description
SomeJavaScriptCode	Required. Specifies a JavaScript to be executed when the event occurs.

Supported by the following HTML tags:

```
<form>
```

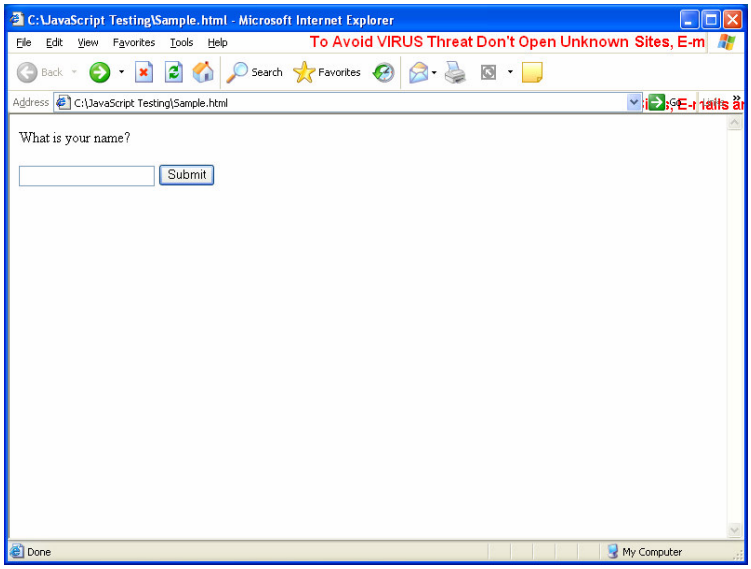
Supported by the following JavaScript objects:

```
Form
```

Example

In this example an alert box displays when a submit box is used:

```
<html>
<body>
What is your name?<br />
<form name="testform" action="otherPageName_onsubmit.html"
onSubmit="alert('Hello ' + testform.inputfield.value +'!')">
<input type="text" name="inputfield" size="20">
<input type="submit" value="Submit">
</form>
</body>
</html>
```



onunload Event

Definition and Usage

The onunload event occurs when a user exits a page.

Syntax

```
onunload="SomeJavaScriptCode"
```

Parameter	Description
SomeJavaScriptCode	Required. Specifies a JavaScript to be executed when the event occurs.

Supported by the following HTML tags:

```
<body>, <frameset>
```

Supported by the following JavaScript objects:

```
Window
```

Example

In this example an alert box will be displayed when the page is closed:

```
<body onunload="alert('The onunload event was triggered')">
</body>
```

Test your Progress:

Q. 1 What event occurs when a person highlights text in a text field?

a) Onblur
b) Onfouc
c) Onselect
d) Onchange

Q. 2 What event occurs when person leaves text in a text field?

a) Onblur
b) Onfocus
c) Onselect
d) Onchange

Q. 3 onsubmit event is an attribute of which HTML element?

a) Form
b) submit
c) button
d) Body

Q. 4 Allen has created a web document that has 15 images, out of 15 images in a web document developed by him, 5 images are used as links. Which event of the image object in the web document will be triggered when the user click on any of the image used as link?

a) onload
b) onclick
c) onmouseover
d) onabort

Q. 5 From the following which is not a valid event?

a) onmouseclick
b) onclick
c) onmouseover
d) onmouseout

CHAPTER 22: JavaScript and cookies

Objectives:

- To describe cookies
- To Understand the use of cookies in web pages
- To implement writing cookies, retrieving values from cookies and deleting cookies

Practical application of cookies

A great feature of JavaScript is that it lets you set and retrieve browser cookies. In this section we are going to see how this is done, along with a simple example that remembers your name and displays it on every page.

What are cookies?
Cookies are small amounts of data stored by the web browser. They allow us to store particular information about a user and retrieve it every time they visit your pages. Each user has their own unique set of cookies.
Cookies are typically used by web servers to perform functions such as tracking your visits to websites, enabling you to log in to sites, and storing your shopping cart. However we don't need fancy web server programming to use cookies. We can use them in JavaScript, too!

How to work with Cookies?

The document.cookie property

Cookies in JavaScript are accessed using the cookie property of the document object. In simple terms, we create a cookie like this:

```
document.cookie = "name=value; expires=date; path=path; domain=domain; secure";
```

and to retrieve all our previously set cookies like this we can use:

```
var x = document.cookie;
```

Let's look more closely at how to set and retrieve cookies.

Setting a cookie

To set a cookie, we set the document.cookie property to a string containing the properties of the cookie that we want to create:

```
document.cookie = "name=value; expires=date; path=path;domain=domain; secure";
```

These properties are explained in the table below:

Property	Description	Example
name=value	This sets both the cookie's name and its value.	username=matt
expires=*date*	This optional value sets the date that the cookie will expire on. The date should be in the format returned	expires= 13/06/2003 00:00:00

	by the toGMTString() method of the Date object. If the expires value is not given, the cookie will be destroyed the moment the browser is closed.	
path=*path*	This optional value specifies a path within the site to which the cookie applies. Only documents in this path will be able to retrieve the cookie. Usually this is left blank, meaning that only the path that set the cookie can retrieve it.	path=/folderName/
domain=*domain*	This optional value specifies a domain within which the cookie applies. Only websites in this domain will be able to retrieve the cookie. Usually this is left blank, meaning that only the domain that set the cookie can retrieve it.	domain=Intinfotech.com
secure	This optional flag indicates that the browser should use SSL when sending the cookie to the server. This flag is rarely used.	Secure

Let's look at a few examples of cookie setting:

```
document.cookie = "username=John; expires=15/02/2003 00:00:00";
```

This code sets a cookie called username, with a value of "John", that expires on Feb 15th, 2003 (note the European time format!).

```
var cookie_date = new Date ( 2003, 01, 15 );
document.cookie = "username=John; expires=" + cookie_date.toGMTString();
```

This code does exactly the same thing as the previous example, but specifies the date using the Date.toGMTString() method instead. Note that months in the Date object start from zero, so February is 01.
document.cookie = "logged_in=yes";

This code sets a cookie called logged_in, with a value of "yes". As the expires attribute has not been set, the cookie will expire when the browser is closed down.

```
var cookie_date = new Date ( );  // current date & time
cookie_date.setTime ( cookie_date.getTime() - 1 );
document.cookie = "logged_in=;
expires=" + cookie_date.toGMTString();
```

This code sets the logged in cookie to have an expiry date one second before the current time - this instantly expires the cookie. A handy way to delete cookies!

There's no escape!

Strictly speaking, we should be *escaping* our cookie values -- encoding non-alphanumeric characters such as spaces and semicolons. This is to ensure that our browser can interpret the values properly. Fortunately this is easy to do with JavaScript's escape() function. For example:

```
document.cookie = "username=" + escape("John Smith")
                + "; expires=15/02/2003 00:00:00";
```

A function to set a cookie

Setting cookies will be a lot easier if we can write a simple function to do stuff like escape the cookie values and build the document.cookie string. Let's see this example

```
function set_cookie ( name, value, exp_y, exp_m, exp_d, path, domain, secure )
{
  var cookie_string = name + "=" + escape ( value );
  if ( exp_y )
  {
    var expires = new Date ( exp_y, exp_m, exp_d );
    cookie_string += "; expires=" + expires.toGMTString();
  }
  if ( path )
        cookie_string += "; path=" + escape ( path );
  if ( domain )
        cookie_string += "; domain=" + escape ( domain );
  if ( secure )
        cookie_string += "; secure";
  document.cookie = cookie_string;
}
```

This function expects the cookie data to be passed to it as arguments; it then builds the appropriate cookie string and sets the cookie.
For example, to use this function to set a cookie with no expiry date:

```
set_cookie ( "username", "John Smith" );
```

To set a cookie with an expiry date of 15 Feb 2003:

```
set_cookie ( "username", "John Smith", 2003, 01, 15 );
```

To set a secure cookie with an expiry date and a domain of Intinfotech.com, but no path:

```
set_cookie ( "username", "John Smith", 2003, 01, 15, "",          "Intinfotech.com", "secure" );
```

A function to delete a cookie

Another useful cookie-handling function is provided below. This function will "delete" the supplied cookie from the browser by setting the cookie's expiry date to one second in the past.

```
function delete_cookie ( cookie_name )
{
  var cookie_date = new Date ( );  // current date & time
  cookie_date.setTime ( cookie_date.getTime() - 1 );
  document.cookie = cookie_name += "=; expires=" +
  cookie_date.toGMTString();
}
```

To use this function, just pass in the name of the cookie you would like to delete - for example:

```
delete_cookie ( "username" );
```

Retrieving cookies

To retrieve all previously set cookies for the current document, you again use the document.cookie property:

```
var x = document.cookie;
```

This returns a string comprising a list of name/value pairs, separated by semi-colons, for *all* the cookies that are valid for the current document. For example:

```
"username=John; password=abc123"
```

In this example, 2 cookies have been previously set: username, with a value of "John", and password, with a value of "abc123".

A function to retrieve a cookie

Usually we only want to read the value of one cookie at a time, so a string containing all our cookies is not that helpful! So here's another useful function that parses the document.cookies string, and returns just the cookie we're interested in:

```
function get_cookie ( cookie_name )
{
  var results = document.cookie.match ( cookie_name + '=(.*?)(;|$)' );
  if ( results )
    return ( unescape ( results[1] ) );
  else
```

```
    return                                                    null;
}
```

The function uses a regular expression to find the cookie name and value we're interested in, then returns the value portion of the match, passing it through the unescape() function to convert any escaped characters back to normal. (If it doesn't find the cookie, it returns a null value.)
Using the function is easy. For example, to retrieve the value of the username cookie:

```
var x = get_cookie ( "username" );
```

A complete simple example

In this example, we've created a page that prompts you for your name the first time you visit it, then stores your name in a cookie and displays your name in the page on subsequent visits.
The first time you visit the page, it should ask you for your name and store it in a cookie. If you visit the page again at any point, it will get your name from the cookie and display it within the page.
Try closing the window that pops up, then opening it again in a new window. Notice that this time it still displays the user name that it retrieved from the cookie!
The cookie is given an expiry date of 1 year from the current date, which means that the browser will remember your name even if you close it down and re-open it.
You can clear the cookie by clicking on the **Forget about me!** link, which calls our delete_cookie() function and then refreshes the page to prompt you for a name again.

```
<html>
<head>
<title>Cookies Example</title>
</head>
<body>
<script language="JavaScript">
<!--
function set_cookie( name, value, expires_year, expires_month, expires_day, path, domain,
secure )
{
  var cookie_string = name + "=" + escape ( value );
  if ( expires_year )
  {
    var expires = new Date ( expires_year, expires_month,
    expires_day );
    cookie_string += "; expires=" + expires.toGMTString();
  }
  if ( path )
          cookie_string += "; path=" + escape ( path );
  if ( domain )
          cookie_string += "; domain=" + escape ( domain );
  if ( secure )
          cookie_string += "; secure";
  document.cookie = cookie_string;
}
```

```
function delete_cookie ( cookie_name )
{
  var cookie_date = new Date();  // current date & time
  cookie_date.setTime(cookie_date.getTime() - 1 );
  document.cookie = cookie_name += "=; expires=" +
  cookie_date.toGMTString();
}

function get_cookie ( cookie_name )
{
  var results = document.cookie.match ( cookie_name +
'=(.*?)(;|$)' );
  if ( results )
    return ( unescape ( results[1] ) );
  else
    return null;
}

if ( ! get_cookie("username" ) )
{
  var username = prompt ( "Please enter your name", "" );
  if ( username )
 {
    var current_date = new Date();
    var cookie_year = current_date.getFullYear() + 1;
    var cookie_month = current_date.getMonth ( );
    var cookie_day = current_date.getDate ( );
    set_cookie ( "username", username, cookie_year,
    cookie_month, cookie_day );
    document.location.reload( );
  }
}
else
{
  var username = get_cookie("username" );
  document.write( "Hi " + username + ", welcome to my
  website!" );
  document.write ( "<br> <a
  href=\"javascript:delete_cookie('username');
  document.location.reload();\">Forget about me!</a>" );
}
// -->
</script>
</body>
</html>
```

Test your Progress:

Q. 1 The Expiration date is stored in cookie as _____

a) A GMT string
b) A Date data type
c) A digital sequence type
d) A sequential numeric type

Q. 2 A cookie is ____

a) A variable
b) A date variable
c) A text variable
d) An object

Q. 3 State whether below statement is true or false
We can delete a cookie

a) True
b) False

Q. 4 A cookie takes a format of a _____

a) Pair-name value
b) Pair-value name
c) Value-name pair
d) Name-value pair

Q. 5 Information in cookie identifies _____

a) The person who is visiting your web site
b) The computer used by the person who is visiting your web site
c) The Internet service provider used by a person who is visiting your web site
d) The visitor's browser

CHAPTER 23: Regular expressions in JavaScript

Objectives:

- To describe regular expressions
- To examine how to use regular expressions to perform pattern-matching operations
- To Understand Special characters in regular expression
- To use match(), replace() functions in regular expressions

JavaScript is useful for a lot more than opening pop-ups. If you use HTML forms on your website, and want to make sure that your visitors submit valid data on those forms, you might want to consider using some regular expressions in JavaScript.

Introduction

If you've ever programmed in Perl, or have had in your hands a UNIX system, then maybe you are pretty familiar with what regular expressions are. If you think that JavaScript is not well suited for your client-side Web programming needs, only useful to open pop-ups and other fancy windowing-related stuff, you don't know the whole story. JavaScript includes full support for Perl-style regular expressions, and it's extremely useful for string matching processes, as I will try to demonstrate in the following lines.
We'll start out explaining what, exactly, regular expressions are, and what they can do for you when used in JavaScript. Finally, We'll show some practical examples, giving a quick taste of how powerful they can be when they are utilized for client-side data validation.

What are regular expressions?

One of the most common situations that come up is having an HTML form for users to enter data. Normally, we might be interested in the visitor's name, phone number and email address, and so forth. However, even if we are very careful about putting some hints next to each required field, some visitors are going to get it wrong, either accidentally or for malicious purposes.
Coding a script to check sensitive user data is sometimes pretty straightforward. But most of the time this process is not so easy. With current websites, we'll need to check the proper standardized format of an email address or a URL. That would be a nightmare for programmers, not to mention a confusing and inefficient error-prone process for checking data validity.
Here's where regular expressions come in handy. A regular expression is a way of describing a pattern in a piece of text. In fact, it's an easy way of matching a string to a pattern. We could write a simple regular expression and use it to check, quickly, whether or not any given string is a properly formatted user input. This saves us from difficulties and allows us to write clean and tight code. Let's see in detail how to work with regular expressions.

Regular expressions in JavaScript - The Basics

At times, regular expressions can look fairly complex, but when it comes right down to it, they are actually text string themselves.
As we can see, in the following example, we are coding a regular expression that searches for the text "JavaScript" (with no quotes):
JavaScript
Pretty simple, isn't it? Any string containing the text "JavaScript" matches this regular expression. In fact, this is more than an equal comparison, because it's matching a string somewhere within another string. It can be anywhere, unless specified otherwise.
But it's not always so simple. As real needs arise, finding more complex string patterns can be a lot more difficult. Special characters might be needed, among other things. So, let's begin working our way through a few examples to explain the basic regular expression syntax.

Anchoring

It's very useful to represent that a string must start with a specific character or many of them, and end with another. That's known as string anchoring. For anchoring strings, a caret (^) may be used to indicate the beginning of a string. A dollar sign ($) is used to indicate the end.

For example:

JavaScript // Matches "JavaScript is great", "The power of JavaScript" and "What is JavaScript?"

Using anchoring characters:

^JavaScript // Matches "JavaScript shines", but not "I love JavaScript"

JavaScript$ // Matches "I like JavaScript", but certainly not "JavaScript is powerful"

^JavaScript$ // matches only "JavaScript" and nothing else.

Regular expressions in JavaScript - Character Escaping

Sometimes, meta-meaning characters, such as (^) or ($) and other special ones need to be included within the string to be searched for, representing the corresponding character instead of having the special meaning in the context of regular expressions syntax. To do so, we need to escape them properly in the string, with a backslash. If a backslash has to be represented too, it must be escaped with another backslash (two slashes \\).

Let's see it in action:

\^ is used to mark the beginning of the string // Matches any string with a caret (^) in it.

\$ is used to mark the end of the string // Matches any string with the dollar sign ($) in it.

Character Sets
Anything enclosed in the special square brace brackets [and] is a character class, a set of characters to which a matched character must belong. Please note that the expression in the square brackets matches only a simple character.

We can list a set, such as:

[aeiou]

which means any vowel.

Or something like this:

[12345]

which matches "1" and "3" but not "a" or "6".

We can also describe a range, or set of ranges with the special hyphen character:

[1-5] // Same as previous example.

[a-z] // Matches any lowercase letter.

[a-zA-Z] // Matches any alphabetic character in lowercase or uppercase.

[0-9a-zA-Z] // Matches any letter or digit.

Besides, we can use sets to specify that a character cannot be a member of a set.

For example:

[^a-z] // Matches any character that is not between a and z.

The caret symbol means "not" when it is placed inside the square brackets. As we have seen previously, it has a different meaning when it's used outside, anchoring the beginning of a string.

Regular expressions in JavaScript - Repetition

Often, it's useful to specify that there might be multiple occurrences of a particular string. We can represent this using the following special characters: "?", "+" and "*". Specifically, "?" means that the preceding character is optional, "+" means one or more of the previous character, while "*" means zero or more of the previous character.

Let's see some examples to clarify this concept:

comput?er // Matches "computer" and "compuer", but not "computter".

comput+er // Matches "computer" and "computter", but not "compuer".

comput*er // Matches "compuer" and "computter" but not "coputer".

^[a-zA-Z]+$ // Matches any string of one or more letters and nothing else.

Subexpressions

Sometimes, it's good to be able to split an expression into subexpressions, so, it's possible, for example, to represent "at least one of these strings followed by exactly one of those." We can achieve this using parentheses and combinations of the special characters ?,+ and *, exactly as we would do in an arithmetic expression.

For example,

(good)?computer // Matches "good computer" and "computer", but not "good good computer".

(good)+computer // Matches "good computer" and "good good computer" but not "computer".

(good)*computer // Matches "computer" and "good good computer" but not "good computers".

Regular expressions in JavaScript - Counted Subexpressions

We can specify how many times something can be repeated by using a numerical expression in curly braces ({ }). We can define an exact number of repetitions ({3} means exactly 3 repetitions), a range of repetitions ({2,4} means from 2 to 4 repetitions), or an open-ended range of repetitions ({2,} means at least two repetitions).

For example,

computer{1,3} // Matches “computer”, “computer computer” and “computer computer computer”.

Branching
Another useful option in building regular expressions is to represent choices for a string. This is done with a vertical pipe (|).

For example, if we want to match several domains, such as com, edu or net, the following expression would be used:

(com)|(edu)|(net)

Summary of special characters

Here are a few special characters that can be used for matching characters in regular expressions:

\n // a newline character

. // any character except a newline

\r // a carriage return character

\t // a tab character

\b // a word boundary (the start or end of a word)

\B // anything but a word boundary.

\d // any digit (same as [0-9])

\D // anything but a digit (same as [^0-9])

\s // single whitespace (space, tab, newline, etc.)

\S // single nonwhitespace.

\w // A “word character” (same as [0-9a-zA-Z_])

\W // A “nonword character (same as [^0-9a-zA-Z_]

Of course, there are more special characters and tips for regular expressions, generally well covered in any complete reference. For the sake of brevity, this list is good enough for this article. Since JavaScript has the same support that Perl for regular expressions, any full guide focused on Perl regular expressions will be applicable to JavaScript too. Now, with all of the basics

covered, we'll see how we can add the power of regular expressions to our JavaScript code, making our developer life a lot easier and expanding our background a little bit more.

Regular expressions in JavaScript - Using regular expressions in JavaScript

Using regular expressions in JavaScript is very easy, often being passed over by people who don't know that it can be done, or by developers arguing that parsing regular expressions slows down client-side applications. Whatever the reasons are, let's show how we can create a regular expression in JavaScript:

```
var re = /regexp/;
```

where regexp is the regular expression itself. Extending the concept to our first example presented in the basics section, let's build one that detects the string "JavaScript":

```
var re = /JavaScript/;
```

As default behavior, JavaScript regular expressions are case sensitive and only search for the first match in any given string. But we can add more functionality by adding the g and i modifiers (g for global and i for insensitive). Annexing the modifiers after the last /, we can make a regular expression search for all matches in the string and ignore case. Once again, let's see some examples to properly understand these concepts.

Given the string "example1 Example2 EXAMPLE3", the following regular expressions match as listed below:

```
/Example[0-9]+/   // Matches "Example2"

/Example[0-9]+/i   // Matches "example1"

/Example[0-9]+/gi   // Matches "example1", "Example2" and "EXAMPLE3"
```

As seen in the previous examples, the use of "i" and "g" modifiers increases noticeably the matching capabilities of regular expressions. So don't forget they exist when you code your next script.

Applying methods to JavaScript strings

Using a regular expression is easy. Every JavaScript variable containing a text string is able to support four main methods (in Object Oriented parlance) or functions to work with regular expressions. They are match(), replace(), search() and test(), the last one being an object method rather than a string method.We'll see in turn how they work.

Regular expressions in JavaScript - The match() method

The match() method takes a regular expression as a parameter and returns an array of all the matching strings found in the string given. If no matches are found, then match() returns false. Let's say we want to check the proper format for a phone number entered by a user, with the form of (XXX) XXX-XXXX. The code listed below does that:

```
function checkPhone( phone ) {  phoneRegex = /^\(\d\d\d\) \d\d\d-\d\d\d\d$/;
 if( !phone.match( phoneRegex ) ) {
  alert( 'Please enter a valid phone number' );
```

```
  return false;
 }
 return true;
}
```

Let's break down the code to understand how it works. First, we define a function that will check if the phone number entered has a valid format. Next, we declare the regular expression to define our pattern. It begins with ^, to indicate that any match must begin at the start of the string. Then we have \(, which will match the opening parenthesis. As seen previously, the character is escaped with a backslash to remove its special meaning in regular expression syntax. As mentioned, \d is a special code that matches any digit. The expression \d\d\d matches any three digits (same effect is achieved with [0-9] [0-9] [0-9]).
The rest of the pattern is pretty easy to understand. \) matches the closing parenthesis, the space matches the proper space for the phone number, then \d\d\d-\d\d\d\d matches three any digits followed by a dash, and then followed by four any digits.Finally, the $ indicates that any match must end at the end of the string.

It's possible to short the regular expression as follows:

```
phoneRegex = /^\(\d{3}\) \d{3}-\d{4}$/;
```

Once we have seen in detail the regular expression pattern, let's see how our function works. It checks whether or not the string contained in phone, passed as a parameter, matches our regular expression. If it does, then an array will be returned which JavaScript will evaluate as true. Otherwise it will return false, displaying the proper error message to the user.This kind of function is commonly used to validate user input data coming from HTML forms, chaining several specific functions to check if data entered is valid or not.

Here is an example:

First, the JavaScript code located in the HEAD section (or even better, in a separate .js file)

```
<script language="javascript">
validateForm = function() {
 if ( checkPhone( this.phone, 'Please enter a valid phone number' ) ) {
  return true;
 }
 return false;
}
checkPhone= function( field, errorMsg) {
  phoneRegex = /^\(\d{3}\) \d{3}-\d{4}$/;
  if( !field.match( phoneRegex ) ) {
  alert( errorMsg );
  field.focus();
  field.select();
  return false;
 }
 return true;
}
signupForm = document.forms[0];   // assumes that it's the first form present in the document
signupForm.onsubmit = validateForm;
</script>
```

And the HTML form code is the following:

```
<form action="signup.htm">
<p>Phone number ( e.g. (123) 456-7890):<input type="text" name="phone" /></p>
<p><input type="submit" value="send" /></p>
</form>
```

The user will be unable to submit this form unless a valid phone number has been entered. If the number format is not valid, an error message will be displayed (generated by our validateForm function).
As stated above, it's easy to add more functionality to our validateForm() function. If we want to apply more than one check to the form, we can embed several calls to specific functions to perform particular validation, achieving something like this:

```
validateForm=function ()
{
 if ( checkPhone( this.phone, 'Please enter a valid phone number' ) && checkEmail( this.email,
'Please enter a valid email address' ) ) {
  return true;
 }
 return false;
```

The code is very compact and is separated completely from the HTML.

Next, it's time to see another useful JavaScript method for working with regular expressions: the replace() method.

Regular expressions in JavaScript - The replace() method

As you might suppose, replace() method replaces matches to a given regular expression with some new string. For a simple example, let's say we want to replace every newline character (\n) with a break
 tag, within a form field used for comments, by formatting the content for proper displaying.
For example:

```
comment = document.forms[0].comments.value;
  // assumes that our HTML form the first one present in the document, and it has a field named
"comments"
comment = comment.replace(/\n/g, "<br />");
```

Pretty simple, right?, The first parameter taken is the regular expression we're searching for (please note the g modifier indicating that it will do a global search, so it will find all of the occurrences in the string, not just the first). The second argument or parameter is the string with which we want to replace any matches (in this case, the
 tag).
Let's wrap the above code into a simple function:

```
function formatField( fieldValue ) {
 return fieldValue = fieldValue. replace(/\n/g, "<br />");
}
```

The function accepts any string as a parameter, and returns the new string with all of the newline characters replaced by
 tags.

In a moment, we'll see another useful method used with regular expressions: the search() method.

The search() method

The search() method is very similar to the indexOf() method, with the difference being that it takes a regular expression as a parameter instead of a string. Then it searches the string for the first match to the given regular expression and returns an integer that indicates the position in the string (strings in JavaScript are zero-indexed elements, so it would return 0 if the match is at the start of the string, 5 if the match begins with the 5th character in the string, and so on). If no match is found, the method will return –1.

Let's say we wish to know the location of the first absolute link within a HTML document. We might code something like this:

```
pos = htmlString.search(/^<a href="http:\/\/$/i);
if ( pos != -1) {
 alert( 'First absolute link found at' + pos +'position.');
}
else {
 alert ( 'Absolute links not found');
}
```

It's very simple and not quite useful, but good enough for example purposes.

So far, all methods described here work by accepting a regular expression as the parameter.

Now, let's take a detailed look at our final method: test(), which is by far the most used to perform client-side validation when using regular expressions in JavaScript.

Regular expressions in JavaScript - The test() method

The test() method is somewhat particular and different from the rest, as we'll see shortly. Within the JavaScript context, when a pattern is defined following the syntax previously described, we are actually defining a new object, called a "regular expression object". I don't intend to go deeply into object programming concepts here. All we need to know is that this object owns the proprietary test() method, which allows us to perform string matching according to a given string.

The test() method takes a given string as a parameter and looks for matches according to the pattern defined within the regular expression object itself. If any matches are found, it will return true. If no matches are found, then it will return false. Let's see an example to explain how this method works:

```
emailpat = /^([a-zA-Z0-9])+([\.a-zA-Z0-9_-])*@([a-zA-Z0-9])+(\.[a-zA-Z0-9_-]+)+$/;
if( !mailpat.test( emailString ) ) {
```

```
 alert( 'Please enter a valid email address' );
}
```

First, we have defined a regular expression object that represents the standardized format of an email address. Then, we use the test() method to check for any matches to the email string passed as a parameter. If there are no matches, the error message will be displayed to the user. We can easily build a function to check for email address validity, as we have seen so many times:

```
function validateEmail ( emailField, errorMsg ) {
 emailpat = /^([a-zA-Z0-9])+([\.a-zA-Z0-9_-])*@([a-zA-Z0-9])+(\.[a-zA-Z0-9_-]+)+$/;
 if( !emailpat.test( emailField.value ) ) {
  alert( errorMsg);
  emailField.focus();
  emailField.select();
  return false;
 }
 return true;
}
```

To validate an email address, we should call the function as:
validateEmail(this.email , 'Please enter a valid email address');
where "this.email" is representing the form field named "email".

Summary

Having described the most common methods used with regular expressions, we can appreciate that they are not as intimidating as they seem. What's more, we took a deeper look at their powerful capabilities for client-side validation, since they are an invaluable tool for verifying user input. By taking advantage of regular expressions in JavaScript, that verification can be done without making any requests to the server.

Validating user input prior to its being submitted is a good way to make sure that data are, at least, well formatted. However, JavaScript cannot be used on its own for complete validation, since it can be disabled in most browsers and offers limited control on user data. Server-side validation is always the best resource for complete and effective control.

Test your Progress

Q.1 State true or false
A regular expression begins with the special character \b.

a) True
b) False

Q.2 Which special character is used to tell the browser to start at the beginning of a string?

a) $
b) *
c) ^
d) []

Q.3 What special character do you use to search for any letter, number, or the underscore?

a) \w
b) \W
c) w
d) W

Q.4 What special character you use to tell the browser to search all occurances of a character?

a) *
b) i
c) g
d) a

Q.5 What special character would you use to specify any nondigit?

a) \d
b) \D
c) $
d) $*