

MCA First Year Semester – I**Paper - VI****INTRODUCTION TO WEB TECHNOLOGY**

- | | |
|--|---------------|
| 1. Introduction to the Web | 5 Hrs |
| <ul style="list-style-type: none">• History and Evolution• Web development cycle• Web publishing• Web contents• Dynamic Web contents | |
| 2. Languages and technologies for browsers | 5 Hrs |
| <ul style="list-style-type: none">• HTML, DHTML, XHTML, ASP, JavaScript• Features and Applications | |
| 3. Introduction to HTML | 10 Hrs |
| <ul style="list-style-type: none">• HTML Fundamentals• HTML Browsers• HTML tags, Elements and Attributes• Structure of HTML code<ul style="list-style-type: none">◦ Head◦ Body• Lists<ul style="list-style-type: none">◦ Ordered List◦ Unordered List◦ Definition List◦ Nesting List• Block Level Tags<ul style="list-style-type: none">◦ Block formatting, Heading, Paragraph, Comments, Text alignment, Font size• Text Level Tags<ul style="list-style-type: none">◦ Bold, Italic, Underlined, Strikethrough, Subscript, superscript• Inserting graphics, Scaling images• Frameset• Forms• An introduction to DHTML | |

4. Cascading Style Sheets **6 Hrs**

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

5. Introduction to ASP **7 Hrs**

- Working of ASP page
- Variables
- ASP forms
- Data types
- Operators
- Object hierarchies
 - ASP Object model
- Request, Response Object collections
- ASP Applications
 - Creating Active Server Page Application
 - Session Object
 - Session Collections
 - Content Collection
 - Response Object Model

6. JavaScript **7 Hrs**

- Introduction
- Operators, Assignments and Comparisons, Reserved words
- Starting with JavaScript
 - Writing first JavaScript program
 - Putting Comments
- Functions
- Statements in JavaScript
- Working with objects
 - Object Types and Object Instantiation
 - Date object, Math Object, String object, Event object, Frame object, Screen object
- Handling Events
 - Event handling attributes

- Window Events, Form Events
- Event Object
- Event Simulation

7. Website Design Concepts

5 Hrs

- How the website should be
 - Basic rules of Web Page design
 - Types of Website

Reference Books :

1. Web Technologies Achyut S. Godbole, Atul Kahate Tata McGraw Hill
2. Web Tech. & Design C. Xavier New Age
3. Multimedia & Web Technology – Ramesh Bangia
4. HTML : The complete reference – Thomas A. Powel
5. HTML Examples – Norman Smith, Edward
6. ASP 3.0 Programmers Reference – Richard Anderson
7. JavaScript Bible – Danny Goodman

List of Practical :

1. Create Web Page and apply some block level tags, text level tags
2. Create Web Page and apply background color, text color, horizontal rules and special characters.
3. Create Web Page and include Ordered list, Unordered list, Definite list and Nested list.
4. Create Web Page and include links to
 - a) Local page in same folder.
 - b) Page in different folder
 - c) Page on the Web
 - d) Specific location within document
5. Create Web Page and include images with different alignment and wrapped text
6. Create tables and format tables using basic table tags and different attributes.
7. Create a frameset that divides browser window into horizontal and vertical framesets.
8. Create Web Page and apply style rules.
9. Create Web Page including control structures using JavaScript.
10. Programs based on Event Handling.



INTRODUCTION TO WEB TECHNOLOGY

Unit Structure

- 1.1 History and Evolution
- 1.2 Web development cycle
- 1.3 Web publishing
- 1.4 Web contents
- 1.5 Dynamic Web contents

1.1 HISTORY AND EVOLUTION OF WEB

The *World Wide Web* allows computer users to locate and view multimedia-based documents (i.e., documents with text, graphics, animations, audios or videos) on almost any subject. Even though the Internet was developed more than three decades ago, the introduction of the World Wide Web is a relatively recent event. In 1990, *Tim Berners-Lee* of CERN (the European Laboratory for Particle Physics) developed the World Wide Web and several communication protocols that form the backbone of the Web.

The Internet and the World Wide Web surely will be listed among the most important and profound creations of humankind. In the past, most computer applications executed on “stand-alone” computers (i.e., computers that were not connected to one another). Today’s applications can be written to communicate with hundreds of millions of computers. The Internet mixes computing and communications technologies. It makes our work easier. It makes information instantly and conveniently accessible worldwide. Individuals and small businesses can receive worldwide exposure on the Internet. It is changing the nature of the way business is done. People can search for the best prices on virtually any product or service.

Special-interest communities can stay in touch with one another and researchers can learn of scientific and academic breakthroughs worldwide.

➤ **The Internet's origins**

In the late 1960s, one of the authors (HMD) was a graduate student at MIT. His research at MIT's Project Mac (now the Laboratory for Computer Science—the home of the World Wide Web Consortium) was funded by ARPA—the Advanced Research Projects Agency of the Department of Defense. ARPA sponsored a conference at which several dozen ARPA-funded graduate students were brought together at the University of Illinois at Urbana-Champaign to meet and share ideas. During this conference, ARPA rolled out the blueprints for networking the main computer systems of about a dozen ARPA-funded universities and research institutions.

They were to be connected with communications lines operating at a then-stunning 56Kbps (i.e., 56,000 bits per second)—this at a time when most people (of the few who could) were connecting over telephone lines to computers at a rate of 110 bits per second. HMD vividly recalls the excitement at that conference. Researchers at Harvard talked about communicating with the Univac 1108 “supercomputer” at the University of Utah to handle calculations related to their computer graphics research.

Many other intriguing possibilities were raised. Academic research was on the verge of taking a giant leap forward. Shortly after this conference, ARPA proceeded to implement the *ARPAnet*, the grandparent of today's *Internet*.

Things worked out differently from what was originally planned. Rather than the primary benefit of researchers sharing each other's computers, it rapidly became clear that enabling the researchers to communicate quickly and easily among themselves via what became known as *electronic mail* (*e-mail*, for short) was the key benefit of the ARPAnet.

This is true even today on the Internet, as e-mail facilitates communications of all kinds among millions of people worldwide.

One of the primary goals for ARPAnet was to allow multiple users to send and receive information simultaneously over the same communications paths (such as phone lines). The network operated with a technique called *packet-switching*, in which digital data was sent in small packages called *packets*. The packets contained data address, error control and sequencing information.

The address information allowed packets to be routed to their destinations.

The sequencing information helped reassemble the packets (which, because of complex routing mechanisms, could actually arrive out of order) into their original order for presentation to the recipient. Packets from different senders were intermixed on the same lines. This packet-switching technique greatly reduced transmission costs compared with the cost of dedicated communications lines.

The network was designed to operate without centralized control. If a portion of the network should fail, the remaining working portions would still route packets from senders to receivers over alternate paths.

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

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

Initially, Internet use was limited to universities and research institutions; then the military began using the Internet. Eventually, the government decided to allow access to the Internet for commercial purposes. Initially, there was resentment among the research and military communities—these groups were concerned that response times would become poor as “the Net” became saturated with users.

In fact, the exact opposite has occurred. Businesses rapidly realized that they could tune their operations and offer new and better services to their clients, so they started spending vast amounts of money to develop and enhance the Internet. This generated fierce competition among the communications carriers and hardware and software suppliers to meet this demand. The result is that *bandwidth* (i.e., the information carrying capacity) on the Internet has increased tremendously and costs have decreased significantly.

It is widely believed that the Internet has played a significant role in the economic prosperity that the United States and many other industrialized nations have enjoyed recently and are likely to enjoy for many years.

➤ **The creation of World Wide Web**

The *World Wide Web* allows computer users to locate and view multimedia-based documents (i.e., documents with text, graphics, animations, audios or videos) on almost any subject. Even though the Internet was developed more than three decades ago, the introduction of the World Wide Web is a relatively recent event. In 1990, *Tim Berners-Lee* of CERN (the European Laboratory for Particle Physics) developed the World Wide Web and several communication protocols that form the backbone of the Web.

The Internet and the World Wide Web surely will be listed among the most important and profound creations of humankind. In the past, most computer applications executed on “stand-alone” computers (i.e., computers that were not connected to one another). Today’s applications can be written to communicate with hundreds of millions of computers. The Internet mixes computing and communications technologies. It makes our work easier. It makes information instantly and conveniently accessible worldwide. Individuals and small businesses can receive worldwide exposure on the Internet. It is changing the nature of the way business is done. People can search for the best prices on virtually any product or service.

Special-interest communities can stay in touch with one another and researchers can learn of scientific and academic breakthroughs worldwide

➤ **The formation of the W3C**

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

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

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

A document passes through three phases: *Working Draft*—which, as its name implies, specifies an evolving draft, *Candidate Recommendation*—a stable version of the document that industry may begin implementing and *Proposed Recommendation*—a Candidate Recommendation that is considered mature (i.e., has been implemented and tested over a period of time) and is ready to be considered for W3C Recommendation status.

The W3C is comprised of three *hosts*—the Massachusetts Institute of Technology (MIT), Institut National de Recherche en Informatique et Automatique (INRIA) and Keio University of Japan—and over 400 *members*, including Deitel & Associates, Inc. Members provide the primary financing for the W3C and help provide the strategic direction of the Consortium.

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

➤ **The Web Standards Project**

The **Web Standards Project** (WaSP) is a group of professional web developers dedicated to disseminating and encouraging the use of the web standards recommended by the World Wide Web Consortium, along with other groups and standards bodies.

Founded in 1998, The Web Standards Project campaigns for standards that reduce the cost and complexity of development while increasing the accessibility and long-term viability of any document published on the Web. WaSP works with browser companies, authoring tool makers, and peers to encourage them to use these standards, since they "are carefully designed to deliver the greatest benefits to the greatest number of web users".

➤ **The rise of web standards**

In 2000, Microsoft released Internet Explorer 5 Macintosh Edition. This was a very important milestone, it being the default

browser installed with the Mac OS at the time, and having a reasonable level of support for the W3C recommendations too. Along with Opera's decent level of support for CSS and HTML, it contributed to a general positive movement, where web developers and designers felt comfortable designing sites using web standards for the first time.

The WaSP persuaded Netscape to postpone the release of the 5.0 version of Netscape Navigator until it was much more compliant (this work formed the basis of what is now Firefox, a very popular browser). The WaSP also created a Dreamweaver Task Force to encourage Macromedia to change their popular web authoring tool to encourage and support the creation of compliant sites.

The popular web development site A List Apart was redesigned early in 2001 and in an article describing how and why, stated: In six months, a year, or two years at most, all sites will be designed with these standards. We can watch our skills grow obsolete, or start learning standards-based techniques now.

That was a little optimistic—not all sites, even in 2008, are built with web standards. But many people listened. Older browsers decreased in market share, and two more very high profile sites redesigned using web standards: Wired magazine in 2002, and ESPN in 2003 became field leaders in supporting web standards and new techniques.

Also in 2003, Dave Shea launched a site called the CSS Zen Garden. This was to have more impact on web professionals than anything else, by truly illustrating that the entire design can change just by changing the style of the page; the content could remain identical.

Since then in the professional web development community web standards have become de rigeur. And in this series, we will give you an excellent grounding in these techniques so that you can develop websites just as clean, semantic, accessible and standards-compliant as the big companies'.

1.2 WEB DEVELOPMENT CYCLE

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

The exact process will vary slightly from designer to designer, but the basics are generally the same.

1. Information Gathering
2. Planning
3. Design
4. Development
5. Testing and Delivery
6. MaintenancePhase

Phase One: Information Gathering

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

This first step is actually the most important one, as it involves a solid understanding of the company it is created for. **It involves a good understanding of you** – what your business goals and dreams are, and how the web can be utilized to help you achieve those goals.

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

Certain things to consider are:

- **Purpose**
What is the purpose of the site? Do you want to provide information, promote a service, and sell a product... ?
- **Goals**
What do you hope to accomplish by building this web site? Two of the more common goals are either to make money or share information.
- **Target Audience**
Is there a specific group of people that will help you reach your goals? It is helpful to picture the “ideal” person you want to visit your web site. Consider their age, sex or interests – this will later help determine the best design style for your site.
- **Content**
What kind of information will the target audience be looking for on your site? Are they looking for specific information, a particular product or service, online ordering...?

Phase Two: Planning

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

The site map is a list of all main topic areas of the site, as well as sub-topics, if applicable. This serves as a guide as to what content will be on the site, and is essential to developing a consistent, easy to understand navigational system. The end-user of the web site – aka your customer – must be kept in mind when designing your site. These are, after all, the people who will be learning about your service or buying your product. A good user interface creates an easy to navigate web site, and is the basis for this.

During the planning phase, your web designer will also help you decide what technologies should be implemented. Elements such as interactive forms, ecommerce, flash, etc. are discussed when planning your web site.

Phase Three: Design

Drawing from the information gathered up to this point, it's time to determine the look and feel of your site.

Target audience is one of the key factors taken into consideration. A site aimed at teenagers, for example, will look much different than one meant for a financial institution. As part of the design phase, it is also important to incorporate elements such as the company logo or colors to help strengthen the identity of your company on the web site.

Your web designer will create one or more prototype designs for your web site. This is typically a .jpg image of what the final design will look like. Often times you will be sent an email with the mock-ups for your web site, while other designers take it a step further by giving you access to a secure area of their web site meant for customers to view work in progress.

Either way, your designer should allow you to view your project throughout the design and development stages. The most important reason for this is that it gives you the opportunity to express your likes and dislikes on the site design.

In this phase, communication between both you and your designer is crucial to ensure that the final web site will match your needs and taste. It is important that you work closely with your

designer, exchanging ideas, until you arrive at the final design for your web site.

Then development can begin...

Phase Four: Development

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

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

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

This entire time, your designer should continue to make your in-progress web site available to you for viewing, so that you can suggest any additional changes or corrections you would like to have done.

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

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

At this point, your web designer will attend to the final details and test your web site. They will test things such as the complete functionality of forms or other scripts, as well last testing for last minute compatibility issues (viewing differences between different web browsers), ensuring that your web site is optimized to be viewed properly in the most recent browser versions.

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

– this is helpful when checking for issues such as cross-browser compatibility as mentioned above.

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

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

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

A web site driven by a CMS gives you the ability to edit the content areas of the web site yourself. You are given access to a back-end administrative area, where you can use an online text editor (similar to a mini version of Microsoft Word). You'll be able to edit existing content this way, or if you are feeling more adventurous, you can even add new pages and content yourself. The possibilities are endless!

It's really up to you as far as how comfortable you feel as far as updating your own web site. Some people prefer to have all the control so that they can make updates to their own web site the minute they decide to do so. Others prefer to hand off the web site entirely, as they have enough tasks on-hand that are more important for them to handle directly.

That's where the help of a your web designer comes in, once again, as they can take over the web site maintenance for you – one less thing for you to do is always a good thing in these busy times!

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

There are a lot of details involved in optimizing and submitting your web site to the search engines – enough to warrant it's own post. This is a very important step, because even though you now have a web site, you need to make sure that people can find it!

1.3 THE PROCESS OF WEB PUBLISHING

Planning, organizing, and visualizing Web sites and pages may be more important than knowing HTML. Unfortunately, these are very difficult things to teach and tend to be learned only by experience. The biggest mistake in Web development is not having a clear goal for a Web site. Even if the site is launched on time and under budget, how can you understand whether you did a good job if you had no goal in the first place? Often goals are vague. Initially, many corporate Web site projects were fueled by FUD—fear, uncertainty, and doubt. With the hype surrounding the Web, it was important to get on the Web before the competition. If the competition was already online, having a Web site appeared even more crucial to corporate success. This is a dangerous situation to be in. Even if budget is not an issue, the benefit of the site will eventually be questioned. Web professionals may find their jobs on the line. Thus, the first step in the Web publishing process is defining the purpose of a site.

➤ Determining Purpose

Finding a purpose for a Web site isn't necessarily very hard. The Web can be very useful, and many common reasons exist to put up a Web site, a few of which are listed here:

- _ Commerce
- _ Entertainment
- _ Information
- _ Marketing
- _ Personal pleasure
- _ Presence
- _ Promotion
- _ Research and education
- _ Technical support

One problem with Web sites is that they may have multiple purposes. A corporate Web site may include demands for marketing, public relations, investor relations, technical support, commerce, and human resource services such as job recruiting. Trying to meet all of these needs while thinking about the Web site as one entity can be difficult. Much like a large-scale software system with many functions, a Web site with many different goals probably should be broken into modules, or subprojects, that constitute parts of a larger whole. This leads to the idea of a *micro site*—a very specific subsite that is part of a larger site and that may be built separately. Microsites have the advantage of allowing the focus, look, or technology of a portion of a site to change without having to change the site as a whole.

No matter how the site is structured, keeping it cohesive and logical is important. For example, establishing a consistent look and feel for the site as a whole is still important, regardless of the multitude of functions. People should feel comfortable moving from your support pages to your marketing pages to your employment pages. A consistent user interface breeds familiarity and generates a united front. The user doesn't need to know that the site is constructed in modules. An inconsistent interface can lead to a user becoming lost and confused while exploring. It helps to have one person (or at least a small group) designated as the overall decision maker on a Webproject.

The Webmaster, or more appropriately termed *Web manager*, coordinates the work efforts and helps keep the project on track. The Webmanager's role is basically the same as a project manager on a large software project. Without such careful management, a Web site with many goals may quickly become a mess, built to satisfy the needs and desires of its builders rather than its viewers.

➤ **Who Is the Audience?**

Of course, just having a purpose for a site isn't enough: you need to consider a site's audience. Notice how often sites reflect the organizational structure of a company rather than the needs of the customer. The goal is always to keep the user at the center of the discussion. Before building a site, make sure to answer some simple audience questions:

- _ Are the users coming from within your organization, or from outside?
- _ Are they young or old?
- _ What language do they speak?
- _ When do they visit the site?
- _ What technologies do they support?
- _ What browsers do they use?

Figuring out an audience doesn't have to be that hard, but don't assume that your audience is too large. People from South America or the Sudan can visit your Web page—but do they? Should they? It is important to be realistic about the audience of the Web. The Web has millions of users, but they aren't all going to visit a particular Web site. If they did, things probably wouldn't work well. When the idea of a site's audience is discussed, don't think in terms of a nameless, faceless John Q. Cybercitizen with a modem and an America Online account. When thinking about users, try to get as specific as possible, and even ask users, if possible. If you already have a site set up, you have a wealth of information about your users—your server logs. Logs can tell you quite a bit about your user base. Depending on the server and its configuration, you can learn the time of day that you get the most hits, the pages visited the most, the browsers and versions being used, the domains your visitors come from, and even the pages that referred visitors to your site. From the logs, you can even infer connection speeds, based on delivery time between pages. If you do not have a server running yet, begin with your best estimate of the kinds of visitors you expect. Once the site is running, check the logs against your estimates—you may find that your audience is different than you expected. An important point in Web design is that you must be willing to revise your designs, even going as far as throwing away your favorite ideas, if they do not fit with your actual audience.

➤ **Who Will Pay for It?**

Sites cost money to produce, so they generally have to produce some benefit to continue. While people do put up sites for personal enjoyment, even this type of site has limits in terms of an individual's investment of time and money. It is very important to understand the business model of the site. Only a year or two ago, many corporate Web budgets were not always the first concern, due to the novelty of the technology. Today, however, Web sites often have to prove that they're "worth it." The money has to come

from somewhere. A site's creator could pay for everything, but that probably isn't reasonable unless the Web site is for pure enjoyment or is nonprofit. Typically, some funds have to be collected, probably indirectly, to support the site. For example, while a promotional site for a movie may not directly collect revenues, it can influence the audience and have some impact on the success or failure of the film. Interestingly, many Web sites are nearly as indirect as a movie promotion site. Measuring the direct benefit of having such sites can be very difficult. More directly measurable sites are those on which leads are collected or goods are sold. Some value can be put on these transactions, and an understanding of the benefit of the site can be determined.

Harder to track, but no less valuable, are Web sites for customer service and support. Placing product information or manuals online, or posting URLs for Frequently Asked Questions (FAQs) lists on your products, enables your customers to answer many of their own questions. Not only can this directly reduce the load on your customer service and support organizations, it also fosters good will among your customer base. When a customer is shopping around, the vendor who makes it easiest for them to obtain the information they are looking for tends to have an immediate advantage.

Another possible business model for a Web site is to have viewers pay, as in a subscription model. This model's problem is that viewers must be given a convincing reason to pay for the information or service available at the Web site. Making a Web site valuable to a user is tricky, especially considering that value often is both psychological and real. When looking at the value of the information available in an encyclopedia, think about its form. If the encyclopedia's information is in book form, the cost might be as high as Rs 50,000. Put the same information on a CD-ROM, and see if the information can be sold for the same cost. What if the same information is on a Web site? On a CD-ROM, the information probably can be sold for Rs 1500 to Rs 5000. On a Web site, it goes for even less, particularly if the user only wants to buy a specific piece of information. Users often place more value on the *delivery* of a good or service than on the good itself. Consider software, for which the design and production of packaging often costs more than reproducing the software itself.

The bottom line is that packaging does count. It is no wonder that users often mistakenly overvalue the graphic aspect of a site. Another business model involves getting someone other than the owner or the intended audience of the site to pay. This model typically comes in the form of an advertising-driven site. However, what is interesting about advertising is that a good is actually being sold—the audience. Advertisers are interested in reaching a particular audience and are willing to pay for an advertisement based on the effectiveness of that ad reaching the intended audience. The question is, how can an audience be attracted, measured, and then sold to the advertisers? The obvious approach is to provide some reason for an audience to come to a Web site and identify themselves. This is very difficult. Furthermore, the audience must be accurately measured, so that advertisers have a way to compare audience size from one site to the next and know how to spend their advertising dollars. People often discuss the number of visitors to their site as an indication of value to an advertiser. The advertisers, however, may not care about the number of visitors, unless those visitors are in their target audience. Regardless of who is paying for the site, some understanding of the costs and benefits of the site is necessary. How much does each visitor actually cost, and what benefit does he or she produce?

Understand that the number of visitors doesn't count, even when using the advertising model. The value of the site transcends this figure and addresses the effectiveness of the visitation. In other words, many visits don't necessarily mean success. Having many visitors to an online store who nonetheless make few purchases may mean huge losses, particularly if it costs more to reach each visitor. Even the form of the Web site may affect the cost. For example, because the amount of data delivered from a Web site is generally related directly to the site's variable costs, sending video costs more than sending regular HTML text. High costs for Web site development isn't always bad, particularly if it produces a big payoff. Goals must be set to measure success and understand how to budget Web sites.

➤ **Defining Goals**

A goal for a site is not the same as its purpose. A *purpose* gives a general idea of what the site is for, whereas a *goal* is very specific. A goal can help define how much should be spent, but goals must be measurable. What is a measurable goal of the site?

Selling x Rupees worth of product directly via the Web site is a measurable goal, as is selling x Rupees of product or service indirectly through leads. Reaching a certain usage level per day, week, or month can be a goal. So is lowering the number of incoming technical support phone calls by a certain amount. Many ways exist to measure the success or failure of a Web project, but measurements generally come in two categories: soft and hard. Hard measurements are those that are easily measured, such as the number of visitors per day. Soft measurements are a little less clear. For example, with a promotional site for a movie, it might be difficult to understand whether the site had any effect on the box office sales.

➤ **Defining Scope**

After you define a site's goals, you need to define what is necessary to reach your goals. You might call this *defining scope*. One thing to remember, though—scope equals money. Because of the flexible nature of the Web, many developers want to add as much as possible to the Web site. However, more isn't always better. The more that is added to the Web site, the more it costs. Furthermore, having too much information makes finding essential information difficult. To think about scope, return to one of the first steps in the process. What is the main purpose of the site? Shouldn't the information of the site reflect this purpose? Looking at the Web, this doesn't always seem to be the case. Have you ever gone to a site and not understood its point?

Finding the essentials of a Web site might not be easy, particularly if it has many purposes or many parties involved in its development. One approach is to have a brainstorming session, in which users provide ideas. Each idea is then written down on a 3x5 card. After all the cards have been created, ask the users to sort the cards into piles. First, sort the cards into similar piles to see how things are related. Next, sort the piles in order of importance. What is important can eventually be distilled out of the cards. Remember to cut down the number of cards, to make people focus on what is truly important. Instead of coming up with ideas of what should go into a site to meet a particular goal or goals, you may be tempted to take existing materials, such as marketing pieces, and convert them to the Web. Unfortunately, creating the content of the site based solely on text and pictures from manuals, brochures, and other support materials rarely works.

Migrating text from print to the Web is troublesome, because the media are so different. Reading onscreen has been proven to be much slower than reading from paper. In practice, people tend not to read information online carefully. They tend to scan it quickly and then print what they need. In this sense, writing for paper tends to go against screen reading. Think about newspaper or TV news stories: the main point is stated first and then discussed. This goes against the slow buildup of many paper documents, which carefully spell out a point. With visitors skimming the site, key bullet points tend to be read while detailed information is skipped. The main thing is to keep the points obvious and simple. Even if information is presented well, poor organization can ruin all the hard work in preparing the information. If a viewer can't find the information, who cares how great it looks or how well it reads?

1.4 WEB CONTENT

Web content is the textual, visual or aural content that is encountered as part of the user experience on websites. It may include, among other things: text, images, sounds, videos and animations.

Beginnings of web content

While the Internet began with a U.S. Government research project in the late 1950s, the web in its present form did not appear on the Internet until after Tim Berners-Lee and his colleagues at the European laboratory (CERN) proposed the concept of linking documents with hypertext. But it was not until Mosaic, the forerunner of the famous Netscape Navigator, appeared that the Internet become more than a file serving system.

The use of hypertext, hyperlinks and a page-based model of sharing information, introduced with Mosaic and later Netscape, helped to define web content, and the formation of websites. Largely, today we categorize websites as being a particular type of website according to the content a website contains.

The page concept

Web content is dominated by the "page" concept. Having its beginnings in academic settings, and in a setting dominated by type-written pages, the idea of the web was to link directly from one academic paper to another academic paper. This was a completely revolutionary idea in the late 1980s and early 1990s when the best a link could be made was to cite a reference in the midst of a type written paper and name that reference either at the bottom of the page or on the last page of the academic paper.

When it was possible for any person to write and own a Mosaic page, the concept of a "home page" blurred the idea of a page. It was possible for anyone to own a "Web page" or a "home page" which in many cases the website contained many physical pages in spite of being called "a page". People often cited their "home page" to provide credentials, links to anything that a person supported, or any other individual content a person wanted to publish.

Even though "the web" may be the resource we commonly use to "get to" particular locations online, many different protocols are invoked to access embedded information. When we are given an address, such as <http://www.youtube.com>, we expect to see a range of web pages, but in each page we have embedded tools to watch "video clips".

HTML web content

Even though we may embed various protocols within web pages, the "web page" composed of "html" (or some variation) content is still the dominant way whereby we share content. And while there are many web pages with localized proprietary structure (most usually, business websites), many millions of websites abound that are structured according to a common core idea.

A **blog** (a blend of the term "**web log**") is a type of website or part of a website. Blogs are usually maintained by an individual with regular entries of commentary, descriptions of events, or other material such as graphics or video. Entries are commonly displayed in reverse-chronological order. "Blog" can also be used as a verb, meaning *to maintain or add content to a blog*.

Most blogs are interactive, allowing visitors to leave comments and even message each other via widgets on the blogs and it is this interactivity that distinguishes them from other static websites.

Many blogs provide commentary or news on a particular subject; others function as more personal online diaries. A typical blog combines text, images, and links to other blogs, Web pages, and other media related to its topic. The ability of readers to leave comments in an interactive format is an important part of many blogs. Most blogs are primarily textual, although some focus on art (Art blog), photographs (photoblog), videos (Video blogging), music (MP3 blog), and audio (podcasting). Microblogging is another type of blogging, featuring very short posts.

A **web search engine** is designed to search for information on the World Wide Web. The search results are generally presented in a list of results and are often called *hits*. The

information may consist of web pages, images, information and other types of files. Some search engines also mine data available in databases or open directories. Unlike Web directories, which are maintained by human editors, search engines operate algorithmically or are a mixture of algorithmic and human input.

An **Internet forum**, or **message board**, is an online discussion site where people can hold conversations in the form of posted messages. They differ from chat room sin that messages are not shown in real-time, to see new messages the forum page must be reloaded. Also, depending on the access level of a user and/or the forum set-up, a posted message might need to be approved by a moderator before it becomes visible.

Forums have their own language; e.g. A single conversation is called a 'thread'. A forum is hierarchical or tree-like in structure: forum - subforum - topic - thread - reply.

Depending on the forum set-up, users can be anonymous or have to register with the forum and then subsequently login in order to post messages. Usually you do not have to login to read existing messages.

Electronic commerce, commonly known as **e-commerce** or **eCommerce**, or e-business consists of the buying and selling of products or services over electronic systems such as the Internet and other computer networks. The amount of trade conducted electronically has grown extraordinarily with widespread Internet usage. The use of commerce is conducted in this way, spurring and drawing on innovations in electronic funds transfer, supply chain management, Internet marketing, online transaction processing, electronic data interchange (EDI), inventory management systems, and automated data collection systems. Modern electronic commerce typically uses the World Wide Web at least at some point in the transaction's lifecycle, although it can encompass a wider range of technologies such as e-mail as well.

1.5 STATIC AND DYNAMIC WEB CONTENT

Types of Website Content - Static and Dynamic

Static Web Site

A **static web page** (sometimes called a **flat page**) is a web page that is delivered to the user exactly as stored, in

contrast to dynamic web pages which are generated by a web application.

Consequently a static web page displays the same information for all users, from all contexts, subject to modern capabilities of a web server to negotiate content-type or language of the document where such versions are available and the server is configured to do so.

Static web pages are often HTML documents stored as files in the file system and made available by the web server over HTTP. However, loose interpretations of the term could include web pages stored in a database, and could even include pages formatted using a template and served through an application server, as long as the page served is unchanging and presented essentially as stored.

Advantages and disadvantages

Advantages

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

Disadvantages

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

Application areas of Static Website:

Need of Static web pages arise in the following cases.

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

Static Web pages are very simple in layout and informative in context. Creation of static website content requires great level of technical expertise and if a site owner is intended to create static web pages, they must be very clear with their ideas of creating such pages since they need to hire a web designer.

Dynamic Web Sites

A **dynamic web page** is a kind of web page that has been prepared with fresh information (content and/or layout), for each individual viewing. It is not static because it changes with the time (ex. a news content), the user (ex. preferences in a login session), the user interaction (ex. web page game), the context (parametric customization), or any combination of the foregoing.

Two types of dynamic web sites

Client-side scripting and content creation

Using client-side scripting to change interface behaviors *within* a specific web page, in response to mouse or keyboard actions or at specified timing events. In this case the dynamic behavior occurs within the presentation.

Such web pages use presentation technology called rich interfaced pages. Client-side scripting languages like JavaScript or ActionScript, used for Dynamic HTML (DHTML) and Flash technologies respectively, are frequently used to orchestrate media types (sound, animations, changing text, etc.) of the presentation. The scripting also allows use of remote scripting, a technique by which the DHTML page requests additional information from a server, using a hidden Frame, XMLHttpRequests, or a Web service.

The Client-side content is generated on the user's computer. The web browser retrieves a page from the server, then processes the code embedded in the page (often written in JavaScript) and displays the retrieved page's content to the user.

The innerHTML property (or write command) can illustrate the client-side dynamic page generation: two distinct pages, A and B, can be regenerated as `document.innerHTML = A + document.innerHTML = B`; or

"on load dynamic" by `document.write(A)` and `document.write(B)`.

Server-side scripting and content creation

Using server-side scripting to change the supplied page source *between* pages, adjusting the sequence or reload of the web pages or web content supplied to the browser. Server responses may be determined by such conditions as data in a posted HTML form, parameters in the URL, the type of browser being used, the passage of time, or a database or server state.

Such web pages are often created with the help of server-side languages such as PHP, Perl, ASP, ASP.NET, JSP, ColdFusion and other languages. These server-side languages typically use the Common Gateway Interface (CGI) to produce *dynamic web pages*. These kinds of pages can also use, on the client-side, the first kind (DHTML, etc.).

Server-side dynamic content is more complicated:

- (1) The client sends the server the request.
- (2) The server receives the request and processes the server-side script such as [PHP] based on the query string, HTTP POST data, cookies, etc.

The dynamic page generation was made possible by the Common Gateway Interface, stable in 1993. Then Server Side Includes pointed a more direct way to deal with server-side scripts, at the web servers.

Combining client and server side

Ajax is a web development technique for dynamically interchanging content with the server-side, without reloading the web page. Google Maps is an example of a web application that uses Ajax techniques and database.

Application areas of Dynamic Website

Dynamic web page is required when following necessities arise:

- Need to change main pages more frequently to encourage clients to return to site.
- Long list of products / services offered that are also subject to up gradation
- Introducing sales promotion schemes from time to time
- Need for more sophisticated ordering system with a wide variety of functions
- Tracking and offering personalized services to clients.
- Facility to connect Web site to the existing back-end system

The fundamental difference between a static Website and a dynamic Website is a static website is no more than an information sheet spelling out the products and services while a dynamic website has wider functions like engaging and gradually leading the client to online ordering.

But both static web site design and dynamic websites design can be designed for search engine optimization. If the purpose is only to furnish information, then a static website should suffice. Dynamic website is absolutely necessary for e-commerce and online ordering



LANGUAGE AND TECHNOLOGY FOR BROWSERS

Unit Structure

- 2.1 HTML
- 2.2 DHTML
- 2.3 XHTML
- 2.4 ASP
- 2.5 JavaScript
- 2.6 Features and Applications

2.1 HTML

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

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

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

2.1.1 A brief history of HTML

➤ **HTML and SGML**

HTML stands for Hyper-Text Markup Language. It is a coding language, which uses a method called markup, to create hyper-text. HTML is actually a simplified subset of a more general markup language called SGML, which stands for Standard Generalized Markup Language, but is gradually returning to SGML as it evolves. This evolution of HTML is worth knowing at least a little about, since HTML is not set in stone. The changes that are occurring have their reasons, mostly in terms of creating capabilities that previous versions were lacking.

➤ **In the beginning...**

In 1989, Tim Berners-Lee, working at the European particle physics institute known as CERN (Centre Européen pour la Recherche Nucléaire), proposed a system to allow scientists to share papers with other using electronic network in methods. His idea became what is called the World-Wide Web. Since these documents were to be shared, some common method coding them needed to be developed. Tim Berners-Lee suggested that it be based on the already existing SGML. Here are a few quotes from a 1990 CERN memo that Berners-Lee wrote:

Hypertext is a way to link and access information of various kinds as a web of nodes in which the user can browse at will. It provides a single user-interface to large classes of information (reports, notes, data-bases, computer documentation and on-line help).

We propose a simple scheme incorporating servers already available at CERN...

A program which provides access to the hypertext world we call a browser...

It would be inappropriate for us (rather than those responsible) to suggest specific areas, but experiment online help, accelerator online help, assistance for computer center operators, and the dissemination of information by central services such as the user office and CN [Computing & Networks] and ECP [Electronics & Computing for Physics] divisions are obvious candidates.

WorldWideWeb (or W3) intends to cater for these services across the HEP [High Energy Physics] community.

As you can see, Tim Berners-Lee put all of the basic pieces into place.

In 1992, when there were all of 50 web servers in the world, CERN released the portable Web browser as freeware. Marc Andreessen, who was working at the National Center for Supercomputing Applications, created a browser called Mosaic which was released in 1993. Shortly after that, he left NCSA to found Netscape.

The first version of the Netscape browser implemented HTML 1.0.

➤ **HTML 1.0 and 2.0**

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

➤ **Browsers and HTML**

Netscape was just one of a number of browsers available. Mosaic was still offered by NCSA, Lynx was available on Unix machines, and few other companies were creating browsers. One of them, Spyglass, was purchased by Microsoft, and became the basis for Internet Explorer. Each browser contains, in its heart, a *rendering engine*, which is the code that tells it how to take your HTML and turn it into something you can see on the screen. What happened at this point is that each company, most particularly Netscape and Microsoft, started to develop their own "extensions" to HTML, often going in different directions. This problem bedevils us to this day, though the upcoming Netscape 6 browser may resolve this by being 100% compliant with the published HTML standards. We are still waiting to see what this will look like.

➤ **W3C takes over: HTML 3.0 and HTML 3.2**

The World Wide Web Consortium (W3C), which had taken over HTML development, attempted to create some standardization in HTML 3.0. But there was so much argument over what should be included that it never got beyond the draft discussion stage. Finally, in 1996 a consensus version, HTML 3.2, was issued. This added features like tables, and text flowing around images, to the official specification, while maintaining backwards compatibility with HTML 2.0. This also is a convenient place for marking the divergence in practice from the separation that Berners-Lee first made between logical structures and presentational elements. And as the Web took off in popularity, this breakdown became widespread and serious. The main focus of the W3C since then has been to rectify the situation. An example of this is the widespread use of tables and transparent "shim" GIFs to create page layout. While this creates pages that are visually correct, the logical structure of the page is pretty much destroyed, and such pages are frequently useless to anyone using a text browser, or a text-to-speech parser.

➤ **HTML 4.0x**

The W3C released the HTML 4.0 specification at the end of 1997, and followed with HTML 4.01 in 1999, which mostly corrected a few errors in the 4.0 specification. This release attempted to correct some of the more egregious errors that 3.2 had allowed (encouraged?) designers to commit, particularly in introducing Cascading Style Sheets. But in fact the W3C has abandoned HTML as the default standard in favor of a move back towards the root of SGML, a larger and more complex language. There will probably never be another HTML specification.

➤ **XHTML 1.0**

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

➤ **HTML5**

By mid-2004, people started to sense lethargy in W3C's development of web standards. Therefore, a group called WHATWG (Web Hypertext Application Technology Working Group) was formed in June 2004. WHATWG is a small, invitation-only group that was founded by individuals from Apple, Mozilla Foundation and Opera Software. They started working on the specifications in July 2004 under the name Web Applications 1.0. The specifications were submitted to W3C and readily accepted. By 2007, W3C adopted the specifications as a starting point of the new HTML called HTML 5.

By the time the first public draft of HTML 5 was published, the word around was that HTML 5 would redefine the web, obsolescing the likes of Adobe Flash, MS Silverlight and Java FX. The promise was that all browsers would use a standard video codec, which would be based on a more open standard. However, reality could not compete with this common dream. Because of strong opposition from the corporates, like Apple and Nokia, HTML 5 cannot specify a standard video codec for all web development.

The First Public Working Draft of the specification was published January 22, 2008. The specifications will be an ongoing work for many years but there is good news for us. The WHATWG has said that parts of HTML 5 will be incorporated into browsers as and when they are finalized. We won't need to wait until the whole specification is completed and approved to start using some of the features of HTML 5.

➤ **New Features in HTML 5**

Other than elements, HTML 5 also introduces additional capabilities to the browser like working in offline mode, multi-threaded JavaScript, etc. Let's go through some of the features.

OfflineMode

With HTML 5, you can specify what resources your page will require and the browser will cache them so that the user can continue to use the page even if she gets disconnected from the internet. This wasn't a problem before AJAX came into existence as the page could not request for resources after it was loaded. However, today's webpages are designed to be sleek so that they load fast and then the additional resources are fetched asynchronously.

LocalDatabase

HTML 5 has included a local database that will be persistent through your session. The advantage of this is that you can fetch the required data and dump it into the local database. The page there after won't need to query the server to get and update data. It will use the local database. Every now and then, the data from the local database is synced with the server. This reduces the load on the server and speeds up responsiveness of the application.

NativeJSON

JSON, or JavaScript Simple Object Notation is a popular alternative to XML, which was almost the de-facto standard before the existence of JSON. Until HTML 5, you needed to include libraries to encode and decode JSON objects. Now, the JavaScript engine that ships with HTML 5 has built-in support for encoding/decoding JSON objects.

CrossDocumentMessaging

Another interesting addition to HTML 5 is the ability to perform messaging between documents of the same site. A good use of this would be in a blogging tool. In one window, you create your post and in another window, you can see what the post would look like without having to refresh the page. When you save the draft of your post, it immediately updates the view window.

CrossSiteXHR

One of the amazing implications of AJAX was to be able to not only fetch data from the server asynchronously, but to be able to get resources from other websites using the XMLHttpRequest. As this wasn't part of HTML4, you needed to include a library to perform such an action. HTML 5 will have XMLHttpRequest support built-in, so you won't need any library.

Multi-threadedJavaScript

A large portion of most web apps is written in JavaScript as it is the only client-side programming language available. One of the HTML 5 promises is that JavaScript will become a multi-threaded language so that it executes more efficiently. However, that only solves one part of the problem. Multithreading will speed up the processing time of JavaScript once it has loaded, but as you increase the number of lines of JavaScript, the pages take longer to load. To solve that problem, they have introduced an attribute called `async` to the `<script>` element. It tells the browser that this

script is not required when the page loads, so it can be fetched asynchronously even after the page has loaded. The syntax for this is:

```
<script async src="jquery.js"></script>
```

Some Features

Internationalization

This version of HTML has been designed with the help of experts in the field of internationalization, so that documents may be written in every language and be transported easily around the world.

One important step has been the adoption of the ISO/IEC:10646 standard as the document character set for HTML. This is the world's most inclusive standard dealing with issues of the representation of international characters, text direction, punctuation, and other world language issues.

HTML now offers greater support for diverse human languages within a document. This allows for more effective indexing of documents for search engines, higher-quality typography, better text-to-speech conversion, better hyphenation, etc.

Accessibility

As the Web community grows and its members diversify in their abilities and skills, it is crucial that the underlying technologies be appropriate to their specific needs. HTML has been designed to make Web pages more accessible to those with physical limitations. HTML 4 developments inspired by concerns for accessibility include:

- Better distinction between document structure and presentation, thus encouraging the use of style sheets instead of HTML presentation elements and attributes.
- Better forms, including the addition of access keys, the ability to group form controls semantically, the ability to group SELECT options semantically, and active labels.
- The ability to markup a text description of an included object (with the OBJECT element).

- A new client-side image map mechanism (the MAP element) that allows authors to integrate image and text links.
- The requirement that alternate text accompany images included with the IMG element and image maps included with the AREA element.
- Support for the title and lang attributes on all elements.
- Support for the ABBR and ACRONYM elements.
- A wider range of target media (tty, Braille, etc.) for use with style sheets.
- Better tables, including captions, column groups, and mechanisms to facilitate non-visual rendering.
- Long descriptions of tables, images, frames, etc.

Authors who design pages with accessibility issues in mind will not only receive the blessings of the accessibility community, but will benefit in other ways as well: well-designed HTML documents that distinguish structure and presentation will adapt more easily to new technologies.

Tables

The new table model in HTML is based on [\[RFC1942\]](#). Authors now have greater control over structure and layout (e.g., column groups). The ability of designers to recommend column widths allows user agents to display table data incrementally (as it arrives) rather than waiting for the entire table before rendering.

Style sheets

Style sheets simplify HTML markup and largely relieve HTML of the responsibilities of presentation. They give both authors and users control over the presentation of documents -- font information, alignment, colors, etc.

Style information can be specified for individual elements or groups of elements. Style information may be specified in an HTML document or in external style sheets.

The mechanisms for associating a style sheet with a document are independent of the style sheet language.

Before the advent of style sheets, authors had limited control over rendering. HTML 3.2 included a number of attributes and elements offering control over alignment, font size, and text color. Authors also exploited tables and images as a means for laying out pages. The relatively long time it takes for users to upgrade their browsers means that these features will continue to be used for some time. However, since style sheets offer more powerful presentation mechanisms, the World Wide Web Consortium will eventually phase out many of HTML's presentation elements and attributes. Throughout the specification elements and attributes at risk are marked as "deprecated". They are accompanied by examples of how to achieve the same effects with other elements or style sheets.

Scripting

Through scripts, authors may create dynamic Web pages (e.g., "smart forms" that react as users fill them out) and use HTML as a means to build networked applications.

The mechanisms provided to include scripts in an HTML document are independent of the scripting language.

2.2 DHTML

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

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

By contrast, a dynamic web page is a broader concept — any web page generated differently for each user, load occurrence, or specific variable values. This includes pages created by client-side scripting, and ones created by server-side scripting (such

as PHP, Perl, JSP or ASP.NET) where the web server generates content before sending it to the client.

There are four parts to DHTML

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

➤ **DOM**

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

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

Scripts

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

CascadingStyleSheets

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

XHTML

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

Features of DHTML

There are four primary features of DHTML:

1. Changing the tags and properties
2. Real-time positioning
3. Dynamic fonts (Netscape Communicator)
4. Data binding (Internet Explorer)

Changing the tags and Properties

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

Real-time positioning

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

Dynamic Fonts

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

Data binding

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

2.3 XHTML

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

While HTML (prior to HTML5) was defined as an application of Standard Generalized Markup Language (SGML), a very flexible markup language framework, XHTML is an application of XML, a more restrictive subset of SGML. Because XHTML documents need to be well-formed, they can be parsed using standard XML parsers—unlike HTML, which requires a lenient HTML-specific parser.

XHTML 1.0 became a World Wide Web Consortium (W3C) Recommendation on January 26, 2000. XHTML 1.1 became a W3C Recommendation on May 31, 2001. XHTML5 is undergoing development as of September 2009, as part of the HTML5 specification.

XHTML is a family of current and future document types and modules that reproduce, subset, and extend HTML 4. XHTML family document types are XML based, and ultimately are designed to work in conjunction with XML-based user agents. The details of this family and its evolution are discussed in more detail in [XHTMLMOD].

XHTML 1.0 (this specification) is the first document type in the XHTML family. It is a reformulation of the three HTML 4 document types as applications of XML 1.0. It is intended to be used as a language for content that is both XML-conforming and, if some simple guidelines are followed, operates in HTML 4 conforming user agents. Developers who migrate their content to XHTML 1.0 will realize the following benefits:

- XHTML documents are XML conforming. As such, they are readily viewed, edited, and validated with standard XML tools.
- XHTML documents can be written to operate as well or better than they did before in existing HTML 4-conforming user agents as well as in new, XHTML 1.0 conforming user agents.
- XHTML documents can utilize applications (e.g. scripts and applets) that rely upon either the HTML Document Object Model or the XML Document Object Model.
- As the XHTML family evolves, documents conforming to XHTML 1.0 will be more likely to interoperate within and among various XHTML environments. Why the need for XHTML?

The benefits of migrating to XHTML 1.0 are described above. Some of the benefits of migrating to XHTML in general are:

- Document developers and user agent designers are constantly discovering new ways to express their ideas through new markup. In XML, it is relatively easy to introduce new elements or additional element attributes. The XHTML family is designed to accommodate these extensions through XHTML modules and techniques for developing new XHTML-conforming modules (described in the XHTML Modularization specification). These modules will permit the combination of existing and new feature sets when developing content and when designing new user agents.
- Alternate ways of accessing the Internet are constantly being introduced. The XHTML family is designed with general user agent interoperability in mind. Through a new user agent and document profiling mechanism, servers, proxies, and user agents will be able to perform best effort content transformation. Ultimately, it will be possible to develop XHTML-conforming content that is usable by any XHTML-conforming user agent.

The Main Changes

There are several main changes in XHTML from HTML:

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

At a glance, this seems like a huge amount of changes but once you start checking through the list you will find that very little on your site actually needs to be changed. In this tutorial I will go through each of these changes explaining exactly what is different.

2.3.1 The Doctype

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

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

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

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

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

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

2.3.2 Document Formation

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

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



```
<title>PageTitle</title>  
OTHER HEADDATA  
  
</head>  
<body>  
CONTENT  
</body>  
</html>
```

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

2.3.3 XHTML Tags

Introduction

One of the major changes to HTML which was introduced to XHTML is that tags must always be properly formed. With the old HTML specification you could be very sloppy in your coding, with missing tags and incorrect formation without many problems but in XHTML this is very important.

2.3.3.1 Lower Case

Probably the biggest changes in XHTML are that now not only the tags you use but, the way in which you write them must be correct. Luckily the major change can be easily implemented into a normal HTML document without much problem.

In XHTML, tags must always be lower case. This means that:

```
<FONT>  
<Font>  
<FONT>
```

are all incorrect tags and must not be used. The font tag must now be used as follows:

```
<font>
```

If you are not writing your code, but instead use a WYSIWYG editor, you can still begin to migrate your documents to XHTML by setting the editor to output all code in lower case. For example, in Dreamweaver 4 you can do this by going to:

Edit -> Preferences -> Code Format and making sure that **Case For Tags** is set to:

`<lowercase>` and also that **Case For Attributes** is set to:

`lowercase="value"`

2.3.3.2 Nesting

The second change to the HTML tags in XHTML is that they must all be properly nested. This means that if you have multiple tags applying to something on your page you must make sure you open and close them in the correct order. For example if you have some bold red text in a paragraph, the correct nesting would be one of the following:

```
<p><b><font      color="#FF0000">Your      Text</font></b></p>
<b><p><font      color="#FF0000">Your      Text</font></p></b>
<p><font      color="#FF0000"><b>Your      Text<b></font></p>
```

These are only examples, though, and there are other possibilities for these tags. What you must not do, though, is to close tags in the wrong order, for example:

```
<p><b><font      color="#FF0000">Your      Text</p></font></b>
```

Although code in this form would be shown correctly using HTML, this is incorrect in the XHTML specification and you must be very careful to nest your tags correctly.

2.3.3.4 Closing Tags

The previous two changes to HTML should not be a particular problem to you if your HTML code is already well formed. The final change to HTML tags probably will require quite a lot of changes to your HTML documents to make them XHTML compliant.

All tags in XHTML must be closed. Most tags in HTML are already closed (for example `<p></p>`, ``, ``) but there are several which are standalone tags which do not get closed.

The main three are:

```
<br>
<img>
<hr>
```

There are two ways in which you can deal with the change in specification. The first way is quite obvious if you know HTML. You can just add a closing tag to each one, e.g.

```
<br></br>
<img></img>
<hr></hr>
```

Although you must be careful that you do not accidentally place anything between the opening and closing tags as this would be incorrect coding. The second way is slightly different but will be familiar to anyone who has written WML. You can include the closing in the actual tag:

```
<br/>
<img/>
<hr/>
```

This is probably the best way to close your tags, as it is the recommended way by the W3C who set the XHTML standard. You should notice that, in these examples, there is a space before the `/>`. This is not actually necessary in the XHTML specification (you could have `
`) but the reason why I have included it is that, as well as being correct XHTML, it will also make the tag compatible with past browsers. As every other XHTML change is backwards compatible, it would not be very good to have a simple missed out space causing problems with site compatibility.

In case you are wondering how the `` tag works if it has all the normal attributes included, here is an example:

```

```

Again, notice the space before the `/>`

2.3.3.5 Attributes

In this part of the XHTML tutorial, I will show you the changes to HTML attributes in XHTML. HTML attributes are the extra parts you can add onto tags (such as `src` in the `img` tag) to change the way in which they are shown. There are four changes to the way in which attributes are changed.

Lowercase

As with XHTML tags, the attributes for them must be in lowercase. This means that, although in the past, code like:

```
<table Width="100%">
```

would have worked, this must now be given as:

```
<table width="100%">
```

Although this is quite a minor issue, it is important to check your code for this mistake as it is quite a common one.

Correct Quotation

Another change in the HTML syntax is that all attributes in XHTML must be quoted. In HTML you could have used the following:

```
<table width=100%>
```

with absolutely no compatibility problems. This all changes in XHTML. If you use code in this format with XHTML it will be incorrect and must be changed. In future, all attributes must be surrounded by quotes (") so the correct format of this code would be:

```
<table width="100%">
```

2.3.3.6 Attribute Shortening

It has become common practice in HTML to shorten a few of the attributes to save on typing and on transfer times. This method has become almost a standard. As with other common practices in HTML, this has been removed from the XHTML specification as it causes incompatibilities between browsers and other devices.

An example of a commonly shortened tag is:

```
<input type="checkbox" value="yes" name="agree" checked>
```

In this, it is the:

checked part which is incorrect. In XHTML all shortened attributes must be given in their 'long' format. For example:

```
checked="checked"
```

so the checkbox code earlier would now need to be written as:

```
<input type="checkbox" value="yes" name="agree"
checked="checked">
```

There are other attributes (such as noresize) which also must be given in full.

The ID Attribute

Probably the biggest change from HTML to XHTML is the one tag attribute change. All other differences have been just making tags more compatible. This is the only full change.

In HTML, the tag has an attribute 'name'. This is usually used to refer to the image in javascript for doing actions like image rollovers. This attribute has now been changed to the 'id' attribute. So, the HTML code:

```

```

would need to be written in XHTML as:

```

```

Of course, this would not be backward compatible with older browsers, so if you still want your site to work fully in all old browsers (as XHTML is intended to do), you will need to include both id and name attributes (this would also be correct XHTML):

```

```

General Rules for converting HTML to XHTML

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

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

W3C recommends that this declaration be included in all XHTML documents, although it is absolutely required only when the character encoding of the document is other than the default Unicode UTF-8 or UTF-16. I said necessary because there can be problems with older browsers which cannot identify this as a valid HTML tag.

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

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

The declarations for the strict XHTML DTD is:

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

The declarations for the frameset XHTML DTD is:

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

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

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

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

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

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

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

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

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

The commonly used empty tags in XHTML are:

1. `<meta />`: for meta information (contained in the head section).
2. `<base />`: used to specify the base URI and also the target frame for hyperlinks (contained in the head section).
3. `<basefont />`: used to specify a base font for the document. Note that attribute 'size' is mandatory.
4. `<param />`: parameters for applets and objects.
5. `<link />`: to specify external stylesheets and other references.
6. ``: to include images. Attributes 'src' for the source URI and 'alt' for alternate text are mandatory.
7. `
`: used for forced line break.
8. `<hr />`: for horizontal rules.
9. `<area />`: used inside image maps. Attribute 'alt' is mandatory.
10. `<input />`: used inside forms for input form elements like buttons, textboxes, textareas, checkboxes and radio buttons.

Example: `<br clear="all">` is wrong; it should be rewritten as `<br clear="all" />`. `` is wrong; it should be ``

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

Rules for XHTML Attributes

- All XHTML attribute names should be in lower case. Example: `Width="100"` and `WIDTH="100"` are wrong; only `width="100"` is correct.

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

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

- HTML supports certain attributes which have no values. Examples are noshade which appears in the `<hr noshade />` tag. XHTML does not allow such empty or compact attributes. The compact attributes generally found in HTML are compact, nowrap, ismap, declare, noshade, checked, disabled, readonly, multiple, selected, noresize and defer. They should always have a value. In XHTML this is done by giving the attribute name itself as the value! Example: noshade becomes `noshade="noshade"` checked becomes `checked="checked"`.
- The name attribute is deprecated and will be removed in a future version of XHTML and the id attribute will take its place. So, for HTML tags that need the name attribute, an id attribute should also be specified with the same value as that for name. Example: `<frame name="myFrame" >` becomes `<frame name="myFrame" id="myFrame" >`
- All & (ampersand) characters in the source code have to be replaced with `&`, which is the equivalent character entity code. This change should be done in all attribute values and URIs. `
`
Example: Bee&Nee will result in an error if you try to validate it; It should be written as `Bee&Nee`.

`Go` is wrong; it should be coded as `Go`.

XHTML Tables

- For `<table>` tag, attribute height is not supported in XHTML 1.0. Only the width is supported. The `<td>` tag does support the height attribute.
- The `<table>`, `<tr>` and the `<td>` tag does not support the attribute background which is used to specify a background image for the table or the cell. Background images will have to be specified either using the style attribute or using external stylesheet. The attribute bgcolor for background color is however supported by these tags.

XHTML Images

- The alt attribute is mandatory. This value of this attribute will be the text that has to be shown in older browsers, text-only

browsers (like lynx), and in place of the image when it is not available. Note that `` is an empty tag.

Example: ``

XHTML and Javascript

- The `type` attribute is mandatory for all `<script>` tags. This value of `type` is `text/javascript` for Javascript.
- The use of external scripts is recommended.

Example:

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

- If you are using internal scripts, enclose it within the starting tag `<![CDATA[` and the ending tag `]]>`. This will mark it as unparsed character data. Otherwise characters like `&` and `<` will be treated as start of character entities (like ` `;) and tags (like ``) respectively.

Example for XHTML Javascript:

```
<script type="text/javascript" language="Javascript">
<!--
<![CDATA[
    document.write('Hello World!');
]]>

//-->
</script>
```

XHTML and Stylesheets

- The `type` attribute is mandatory for `<style>` tag. The value of `type` is `text/css` for stylesheets.
- The use of external stylesheets is recommended.

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

Enclose internal style definitions within the starting tag `<![CDATA[` and the ending tag `]]>` to mark it as unparsed character data.

Example:

```
<style type="text/stylesheet">
<![CDATA[
  .MyClass { color: #000000; }
]]>
</style>
```

Otherwise the `&` and `<` characters will be treated as start of character entities (like ` `;) and tags (like ``) respectively.

Element Prohibitions in XHTML

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

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

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

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

2.4 ACTIVE SEVER PAGE

Microsoft® Active Server Pages (ASP) is a server-side scripting technology that can be used to create dynamic and interactive Web applications. An ASP page is an HTML page that contains server-side scripts that are processed by the Web server before being sent to the user's browser. You can combine ASP with Extensible Markup Language (XML), Component Object Model (COM), and Hypertext Markup Language (HTML) to create powerful interactive Web sites.

Server-side scripts run when a browser requests an .asp file from the Web server. ASP is called by the Web server, which processes the requested file from top to bottom and executes any script commands. It then formats a standard Web page and sends it to the browser.

It is possible to extend your ASP scripts using COM components and XML. COM extends your scripting capabilities by providing a compact, reusable, and secure means of gaining access to information. You can call components from any script or programming language that supports Automation. XML is a markup language that provides a format to describe structured data by using a set of tags.

ASP files have a file extension of .asp, much like HTML files have file extensions of either .htm or .html. The HTML files that contain the ASP instructions enclose those instructions within tags that look like this: `<% and %>`. Notice that unlike HTML, the ASP ending tag does not include a slash. To send the results of an ASP instruction directly to a browser, you add an equals sign: `<%=`.

ASP programming involves scripting in Visual Basic Script, Jscript, Perl, Python, or other languages. Certain modifications are necessary, but the programmer who has written code in these other languages will find ASP programming to be familiar indeed. The two languages that work the best for ASP programming are VBScript and Jscript

One common use of ASP programming is to gather data from the user and display it at another time. For example, you can use ASP programming to query the user to type in his or her name and then display that name on subsequent pages during the user's visit. Once the name is input, the ASP programming protocols transfer that data to the requisite database, from which it can be accessed by other HTML pages that contain the coding guiding such requests. Such data requests and displays can be as complex as you want to make them.

Written data isn't the only thing that can be uploaded to your website using ASP programming. You can design forms that allow users to upload image files to your site as well. Real estate

websites are perfect examples of sites that can take advantage of this functionality.

ASP programming also comes in handy when your HTML pages and what they display involve accessing large databases containing tons of data. In this case, you will really appreciate the benefits of not having to change HTML files when you update the parameters of your databases. ASP programming makes this process simple.

2.5 JAVASCRIPT

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

What can a JavaScript do?

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

- **JavaScript can be used to detect the visitor's browser** - A JavaScript can be used to detect the visitor's browser, and - depending on the browser - load another page specifically designed for that browser
- **JavaScript can be used to create cookies** - A JavaScript can be used to store and retrieve information on the visitor's computer

2.6 FEATURES AND APPLICATION

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

A few of the more common technologies are listed below.

➤ **ASP**

Active Server Pages are used to perform server-side scripting. This is a way to get things done on the web server, as opposed to, say, JavaScript, which lets you get things done on the client (browser). Although there is a Unix and Linux version of ASP, it is primarily intended for use on Microsoft web server based systems.

ASP is useful for tasks such as maintaining a database, creating dynamic pages and respond to user queries (and many other things as well).

➤ **CGI**

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

➤ **CSS**

You use Cascading Style Sheets to format your web pages anyway that you want. CSS is complicated, but the complication pays off by being able to create web pages that look much better than otherwise. One very nice feature is the ability to define formatting commands in a single file, which is then included in all of your web pages. This let's you make one change to modify the look of your entire site.

➤ **HTACCESS**

The .htaccess file allows you to set parameters for your web site and folders (directories). The most common use is to protect directories by defining usernames and passwords. Htaccess can be used for many other things as well, including denying access to specific addresses, keeping out hostile spiders and redirecting traffic transparently to the user. The downside of htaccess is the language used is often extremely obscure, difficult to understand and extraordinarily precise. A small error in your htaccess file can disable your entire web site until the error is fixed.

➤ **Java**

Java is a client-side (meaning it's executed by the browser not the server) language. It is efficient and very powerful. The primary advantage of Java over ActiveX is Java has a sane security model (called the Sandbox Model), while the ActiveX model is so imbecilic as to defy imagination. Java is also much less likely to crash systems. On the other hand, Java is substantially slower than ActiveX, and there are many tasks that simply cannot be performed in Java because it is denied access to the operating system and disk itself.

➤ **JavaScript**

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

➤ **Office**

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

➤ **Perl**

A great scripting language which makes use of the CGI standard to allow work to be done on the web server. PERL is very easy to learn (as programming languages go) and straightforward to use. It is most useful for guestbooks, email forms and other similar, simple tasks. PERL's primary disadvantage is the overhead on the server is very high, as one process is created each time a routine is called, and the language is interpreted, which means the code is recompiled each time it is run. For complex tasks, a server-side scripting language such as PHP or ASP is much preferred.

➤ **PHP**

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

➤ **SSI**

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

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

➤ **VBScript**

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



INTRODUCTION TO HTML

Unit Structure

3.0 Objectives

3.1 HTML Fundamentals

3.2 HTML Browsers

3.3 HTML tags, Elements and Attributes

3.4 Structure of HTML code

- o Head

- o Body

3.5 Lists

- Ordered List
- Unordered List
- Definition List
- Nesting List

3.0 OBJECTIVES

To learn and understand how to make web pages .

INTRODUCTION (3.1 FUNDAMENTALS AND 3.2 BROWSERS)

- HTML (Hypertext Markup Language) is used to create document on the World Wide Web.
- Its collection of “**TAGS**”, that are used to make web documents that are displayed using browsers on internet.
- HTML is platform independent language
- To display a document in web it is essential to mark-up the different elements (headings, paragraphs, tables, and so on) of the document with the HTML tags.

- To view a mark-up document, user has to open the document in a browser.
- A browser understands and interpret the HTML tags, identifies the structure of the document (which part are which) and makes decision about presentation (how the parts look) of the document.
- We can also make documents look attractive using graphics , fonts size and color using HTML
- User can make a link to the other document or the different section of the same document by creating Hypertext Links also known as Hyperlinks.

3.1.1 How to make HTML pages????

What You Need

You don't need any tools to learn HTML.

- You don't need any HTML editor
- You don't need a web server
- You don't need a web site

Editing HTML

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

We can also use dreamviewer or frontpage

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

HTML gives authors the means to:

- Publish online documents with headings, text, tables, lists, photos, etc.
- Retrieve online information via hypertext links, at the click of a button.
- Design forms for conducting transactions with remote services, for use in searching for information, making reservations, ordering products, etc.
- Include spread-sheets, video clips, sound clips, and other applications directly in their documents.

3.3 TAGS, ELEMENTS, ATTRIBUTE

- HTML is set of instruction.
- These instruction, along with the text to which the instruction apply are called **HTML elements**.

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

HTML Tags

HTML markup tags are usually called HTML tags:

- HTML tags are keywords surrounded by **angle brackets** like `<html>`
- HTML tags normally **come in pairs** like `` and ``
- The first tag in a pair is the **start tag**, the second tag is the **end tag**
- Start and end tags are also called **opening tags** and **closing tags**

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

- **Container Tags:**

- These tags are always paired with closures tags are called *container tags*.
- These tags activate an effect and have a companion tag to close/discontinue the effect.
- Tags which have both the opening and closing i.e. `<TAG>` and `</TAG>`
- For example `` tag starts bold effect for text and its companion tag `` ends the bold effect.
 - Statement like:


```
<B> How </B>
```
 - Will have word **How** in bold.
- The `<HTML>`, `<HEAD>`, `<TITLE>` and `<BODY>` `<SCRIPT>` `` `<A>` etc. tags are all container tags.

- **Empty Tags:**

- Tags, which have only opening and no ending, are called *empty tags/ standalone tag*. The
- `<HR>`, which is used to draw horizontal, rule across the width of the document, and line break `
` tags are empty tags.

- When client request for a page from web server browser fetches. .
- All web pages contain instructions for display called 'tags'.

- Browsers read tags and display page according to tags on client computer

HTML *Attributes*

- HTML elements can have **attributes**.
- Attributes provide **additional information** about an element about how the tag should appear or behave.
- Attributes are always specified in **the start tag** .
- An element's start tag may contain any number of space separated attribute/value pairs.
- Attributes consist of a *name* and a *value* separated by an equals (=) sign (name/value pairs like: **name = "value"**).
- For example, consider the tag BODY, which marks as the beginning (or end) of HTML body.
- This tag can have several attributes, one of them is BGCOLOR, specific the background color of the document.

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

- Attribute values should always be enclosed in quotes.
- Double style quotes are the most common, but single style quotes are also allowed.

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

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

3.4 STRUCTURE OF HTML CODE

- HTML documents are structured into two parts, the **HEAD**, and the **BODY**.
- Both of these are contained within the HTML element – it simply denotes its HTML document
- The head contains information about the document that is not generally displayed with the document, such as its **TITLE**.
- The **BODY** contains the body of the text
- Elements allowed inside the HEAD, such as TITLE, are not allowed inside the BODY, and vice versa.

Example of Document Structure page1.html

```
<HTML>
<HEAD>
  <TITLE> This is my FRIST HTML Page </TITLE>
</HEAD>
<BODY>
  <h1> Global Warming </h1>
  Global warming is when the earth heats up (the temperature
  rises). It happens when greenhouse gases (carbon dioxide,
  water vapor, nitrous oxide, and methane) trap heat and light
  from the sun in the earth's atmosphere, which increases the
  temperature. This hurts many people, animals, and plants.
  Many cannot take the change, so they die.
  <p> What causes global warming?
  <ul>
    <li>Turning on a light
    <li>Riding in a car
    <li>Watching T.V.
    <li>Listening to a stereo
  </ul>
</BODY>
</HTML>
```


- ✓ **ALINK** = "red" - (**Deprecated**) the color of the link currently being visited.
 - ✓ **VLINK** = "green" - (**Deprecated**) the color of visited links.
 - ✓ **BGPROPERTIES** = **FIXED** - if the background image should not scroll
 - ✓ **TOPMARGIN**: size of top and bottom margins
 - ✓ **LEFTMARGIN**: size of left and right margins
 - ✓ **MARGINHEIGHT**: size of top and bottom margins
 - ✓ **MARGINWIDTH**: size of left and right margins
 - ✓ **SCROLL** = **YES | NO** - If the document should have a scroll bar
- ```
<body text = "#000000" bgcolor="#FFFFFF"
link="#0000EF" vlink="#51188E" alink="#FF0000"
background="sunset.gif">
```

**Note:**

- For the depreciated attributes noted above, see the "Setting document style" section for an example of how to set the same attributes using a style sheet.
- The values of color may be a hexadecimal value from 0 through FF which in decimal is 0 through 255. The highest value being the highest strength of the respective color.
- This format is "RRGGBB".

3.	<HEAD>	Container	<ul style="list-style-type: none"> <li>• HEAD tag comes after the HTML start tag.</li> <li>• It contains TITLE tag to give the document a title that displays on the browsers title bar at the top.</li> <li>• The Format is: <pre>&lt;HEAD&gt;   &lt;TITLE&gt;     Your title goes     here   &lt;/TITLE&gt; &lt;/HEAD&gt;</pre> </li> </ul>
<ul style="list-style-type: none"> <li>• Elements allowed in the HTML 4.0 strict HEAD element are: <ul style="list-style-type: none"> <li>◦ <b>BASE</b> - Defines the base location for resources in the current HTML document. Supports the TARGET attribute in frame and transitional document type definitions.</li> </ul> </li> </ul>			

- **LINK** - Used to set relationships of other documents with this document.
- **META** - Used to set specific characteristics of the web page and provide information to readers and search engines.
- **SCRIPT** - Used to embed script in the header of an HTML document.
- **STYLE** - Used to embed a style sheet in the HTML document.
- **TITLE** - Sets the document title.

---

### 3.5 LISTS

---

- HTML defines three different types of lists:
  - i. ordered (commonly known as numbered) lists,
  - ii. unordered (commonly known as bulleted) lists,
  - iii. and definition lists (for term and definition pairs).
- Each HTML list has the following structure:
 

```
<list_tag>
<item_tag> Item text </item_tag>
<item_tag> Item text </item_tag>
...
</list_tag>
```

**Note** Definition lists are slightly different in syntax because they have an item tag (<dt> or “definition term”) and a definition description tag (<dd>).


The ordered and unordered lists have many different display options available:


- ◆ Ordered lists can have their items preceded by the following:
  - Arabic numbers
  - Roman numerals (upper- or lowercase)
  - Letters (upper- or lowercase)
  - Numerous other language-specific numbers/letters
- ◆ Unordered lists can have their items preceded by the following:
  - Several styles of bullets (filled circle, open circle, square, and so on)
  - Images



**Ordered (Numbered) Lists**

- Ordered lists have elements that are preceded by numbers or letters and are meant to provide a sequence of ordered steps for an activity.
- Ordered lists use the ordered list **tag (<ol>)** to delimit the entire list and the list item **tag (<li>)** to delimit each individual list item.

<p><b>Example 1:</b></p> <pre>&lt;html&gt; &lt;head&gt; &lt;title&gt;   Example Ordered List&lt;/title&gt; &lt;/head&gt; &lt;body&gt; &lt;ol type = "1"&gt; &lt;lh&gt; Fruits &lt;li&gt; Mango. &lt;/li&gt; &lt;li&gt; Apple &lt;/li&gt; &lt;li&gt;Banana &lt;/li&gt; &lt;/ol&gt; &lt;/body&gt; &lt;/html&gt;</pre>	 <p>The screenshot shows a web browser window titled "Example Ordered". The browser's address bar and menu bar are visible. The main content area displays a heading "Fruits" followed by an ordered list with three items: "1. Mango.", "2. Apple", and "3. Banana".</p>
---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

<p><b>Example 2:</b></p> <pre>&lt;html&gt; &lt;head&gt; &lt;title&gt;   Example Ordered List &lt;/title&gt; &lt;/head&gt; &lt;body&gt; &lt;ol type = "I"&gt; &lt;lh&gt;   Programming Languages &lt;li&gt; C. &lt;/li&gt; &lt;li&gt; C ++ &lt;/li&gt; &lt;li&gt; JAVA &lt;/li&gt; &lt;li&gt; DOTNET &lt;/li&gt; &lt;/ol&gt; &lt;/body&gt; &lt;/html&gt;</pre>	 <p>The screenshot shows a web browser window titled "Example Ordered List". The browser's address bar and menu bar are visible. The main content area displays a heading "Programming Languages" followed by an ordered list with four items: "I. C.", "II. C++", "III. JAVA", and "IV. DOTNET".</p>
---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

**Unordered (Bulleted) Lists**

- Unordered lists are similar to numbered lists except that they use bullets instead of numbers or letters before each list item.
- Bulleted lists are generally used when providing a list of nonsequential items.
- For example, consider the following list of ice cream flavors:
  - ◆ Vanilla
  - ◆ Chocolate
  - ◆ Strawberry
- Unordered lists use the unordered list tag (<ul>) to delimit the entire list and the list item tag (<li>) to delimit each individual list item.
- In the preceding example, the list has three elements each preceded with a small, round, filled bullet.
- Unordered lists also support the list-style-type property, but with slightly different values:
  - ✓ ◆ disc
  - ✓ ◆ circle
  - ✓ ◆ square
  - ✓ ◆ none

<pre> &lt;html&gt; &lt;head&gt; &lt;title&gt;Example Ordered List&lt;/title&gt; &lt;/head&gt; &lt;body&gt; Example 1: &lt;ul type = "square"&gt; &lt;lh&gt; SQUARE &lt;li&gt; C. &lt;/li&gt; &lt;li&gt; C ++ &lt;/li&gt; &lt;li&gt; JAVA &lt;/li&gt; &lt;li&gt; DOTNET &lt;/li&gt; &lt;/ul&gt;  &lt;ul type = "circle"&gt; &lt;lh&gt; CIRCLE &lt;li&gt; C. &lt;/li&gt; &lt;li&gt; C ++ &lt;/li&gt; &lt;li&gt; JAVA &lt;/li&gt; &lt;li&gt; DOTNET &lt;/li&gt; &lt;/ul&gt;  &lt;ul type = "disc"&gt; &lt;lh&gt; DISC &lt;li&gt; C. &lt;/li&gt; &lt;li&gt; C ++ &lt;/li&gt; &lt;li&gt; JAVA &lt;/li&gt; &lt;li&gt; DOTNET &lt;/li&gt; &lt;/ul&gt;  &lt;ul type = "none"&gt; &lt;lh&gt; DISC &lt;li&gt; C. &lt;/li&gt; &lt;li&gt; C ++ &lt;/li&gt; &lt;li&gt; JAVA &lt;/li&gt; &lt;li&gt; DOTNET &lt;/li&gt; &lt;/ul&gt; &lt;/body&gt; &lt;/html&gt; </pre>	
---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	--

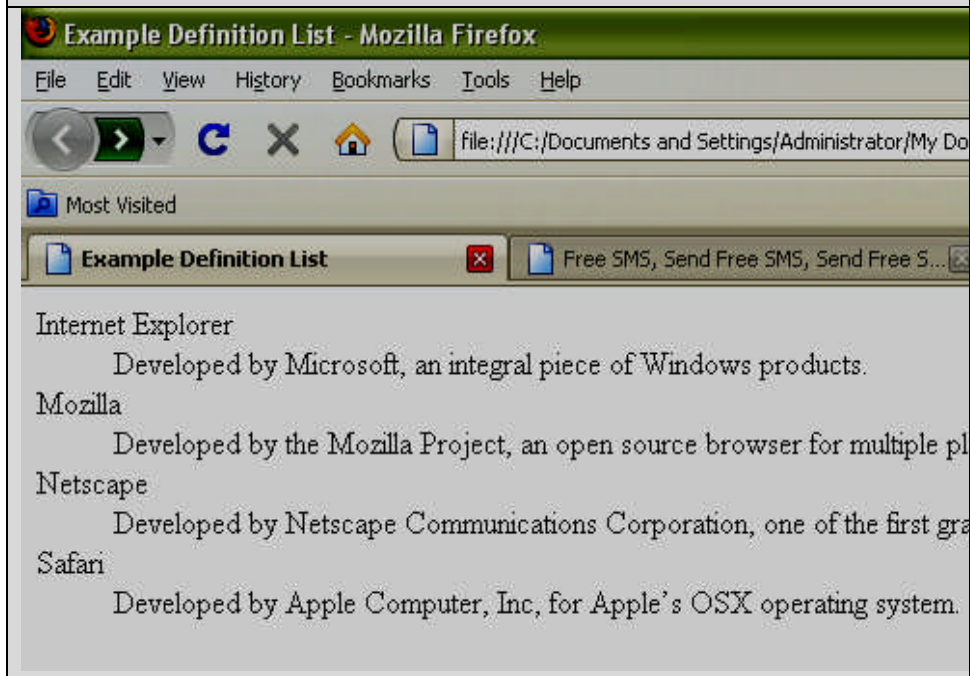
### **Definition Lists**

- Definition lists are slightly more complex than the other two types of lists because they have two elements for each item, a term and a definition.
- However, there aren't many formatting options for definition lists, so their implementation tends to be simpler than that of the other two lists.
- Consider this list of definitions, highlighting popular Web browsers:
  - **Internet Explorer**
    - Developed by Microsoft, an integral piece of Windows products.
  - **Mozilla**
    - Developed by the Mozilla Project, an open source browser for multiple platforms.
  - **Netscape**
    - Developed by Netscape Communications Corporation, one of the first graphical browsers.
  - **Safari**
    - Developed by Apple Computer, Inc., for Apple's OSX operating system.
- The bulleted items can be coded as list terms and their definitions as list definitions, as shown in the following code.

```

<html>
<head>
<title>Example Definition List</title>
</head>
<body>
<dl>
<dt> Internet Explorer</dt>
<dd> Developed by Microsoft, an integral piece of Windows
products < /dd>
<dt> Mozilla </dt>
<dd> Developed by the Mozilla Project, an open source browser for
multiple platforms. </dd>
<dt> Netscape </dt>
<dd> Developed by Netscape Communications Corporation, one of
the first graphical browsers. </dd>
<dt> Safari </dt>
<dd> Developed by Apple Computer, Inc, for Apple's OSX
operating system. </dd>
</dl>
</body>
</html>

```



- You can nest lists of the same or different types.
- For example, suppose you have a bulleted list and need a numbered list beneath one of the items.

## Some few information about tags:

Tags	Attributes
<b>&lt;dd&gt;</b>	Class = "class name" dir = "direction for weak/neutral text" id = "document-wide unique id" lang = "language code" onclick = "script" ondblclick = "script" onkeydown = "script" onkeypress = "script" onkeyup = "script" onmousedown = "script" onmousemove = "script" onmouseout = "script" onmouseover = "script" onmouseup = "script" style = "associated style info" title = "advisory title"
<b>&lt;dl&gt;</b>	lass = "class name" <i>compact</i> dir = "direction for weak/neutral text" id = "document-wide unique id" lang = "language code" onclick = "script" ondblclick = "script" onkeydown = "script" onkeypress = "script" onkeyup = "script" onmousedown = "script" onmousemove = "script" onmouseout = "script" onmouseover = "script" onmouseup = "script" style = "associated style info" title = "advisory title"

<b>&lt;li&gt;</b>	class = "class name" dir = "direction for weak/neutral text" id = "document-wide unique id" lang = "language code" onclick = "script" ondblclick = "script" onkeydown = "script" onkeypress = "script" onkeyup = "script" onmousedown = "script" onmousemove = "script" onmouseout = "script" onmouseover = "script" onmouseup = "script" style = "associated style info" title = "advisory title" <i>type</i> = " <i>content type</i> " <i>value</i> = " <i>value</i> "
<b>&lt;ol&gt;</b>	lass = "class name" <i>compact</i> dir = "direction for weak/neutral text" id = "document-wide unique id" lang = "language code" onclick = "script" ondblclick = "script" onkeydown = "script" onkeypress = "script" onkeyup = "script" onmousedown = "script" onmousemove = "script" onmouseout = "script" onmouseover = "script" onmouseup = "script" <i>start</i> = " <i>starting sequence number</i> " style = "associated style info" title = "advisory title" <i>type</i> = " <i>content type</i> "

## Examples of List:

DEFINITION	
Coding	results
Preceding Text <DL> <LH> List Header </LH> <DT> List item 1 <DD> Description of List item 1. <DT> List item 2 <DD> Description of List item 2. </DL>	Preceding TextList Header List item 1 Description of List item 1. List item 2 Description of List item 2.

WITH IMAGE BULLETS	
Coding	results
Preceding Text <DL> <LH> List Header</LH> <DD> <IMG SRC = "redball.gif" ALT = "*"> List item 1. <DD> <IMG SRC = "redball.gif" ALT = "*"> List item 2. </DL>	Preceding TextList Header • List item 1. • List item 2.

NESTED LISTS	
Coding	Results
Preceding Text <OL TYPE = "1"> <LI> List Item 1 <OL TYPE = "a"> <LI> Nested List Item 1 <LI> Nested List Item 2 </OL> <LI> List Item 2 <UL> <LI> Nested List Item 1 </OL> </OL>	Preceding Text I. List Item 1 a. Nested List Item 1 b. Nested List Item 2 II. List Item 2 o Nested List Item 1



ORDERED		
Coding		Results
Preceding	Text	Preceding Text
<OL>		List Header
<:LH> List Header </LH>		
<LI> List item 1		1. List item 1
<LI> List item 2		2. List item 2
</OL>		

ORDERED : <LI> TYPES	
Coding	results
<OL>	
<LI TYPE = "A"> List item	A. List item
<LI TYPE = "a"> List item	b. List item
<LI TYPE = "I"> List item	III. List item
<LI TYPE = "i"> List item	iv. List item
<LI TYPE = "1"> List item	5. List item
</OL>	

ORDERED : <OL> TYPES	
Example: <OL TYPE = "A">	
Coding	Results
<OL TYPE = "A">	A. List item 1
<LI> List item 1	B. List item 2
<LI> List item 2	
</OL>	

ORDERED : <OL START = "8">	
Coding	Results
<OL START = "8">	8. List item 8
<LI> List item 1	9. List item 9
<LI> List item 2	
</OL>	

UNORDERED		
Coding		Results
Preceding	Text	Preceding Text
<UL>		List Header
<LH> List Header</LH>		
<LI> List item 1		• List item 1
<LI> List item 2		• List item 2
</UL>		

UNORDERED <LI> TYPES	
IMAGE	
Coding	results
<pre>&lt;UL SRC = "redball.gif"&gt; &lt;LI&gt;    List item 1 &lt;LI&gt;    List item 2 &lt;/UL&gt;</pre>	<ul style="list-style-type: none"> <li>○ List item 1</li> <li>○ List item 2</li> </ul>

PLAIN	
Coding	Results
<pre>&lt;UL PLAIN&gt; &lt;LI&gt; List item 1 &lt;LI&gt; List item 2 &lt;/UL&gt;</pre>	<ul style="list-style-type: none"> <li>○ List item 1</li> <li>○ List item 2</li> </ul>

TYPE	
Coding	results
<pre>&lt;UL&gt; &lt;LI TYPE = "CIRCLE"&gt; List item &lt;LI TYPE = "DISC"&gt; List item &lt;LI TYPE = "SQUARE"&gt; List item &lt;/UL&gt;</pre>	<ul style="list-style-type: none"> <li>○ List item</li> <li>○ List item</li> <li>○ List item</li> </ul>

### Summary:

- We have seen what is HTML
  - HTML is a script language.
  - The basic formatting controls are included between the brackets < and >.
- When you save an HTML file, you can use either the **.htm** or **the .html** file extension.
- In this chapter we have also seen 2 type of tag:
  - Container tag
  - Empty tag
- Further we have seen HTML elements has an attributes which is always written in **name : value** format, gives additional information about elements.
- Structure of HTML, always contains <HTLM>, <BODY>.

- List items can contain block and text level items, although headings and address elements are excluded
- A basic *bulleted list* can be produced as follows:
  - Start with an opening `<ul>` tag.
  - Give the items one at a time, each preceded by a `<li>` tag. (There is no closing tag for list items.)
  - End with a closing `</ul>` tag.
  - So, here's an example two-item list:

```

 First item goes here.
 Second item goes here.

```

- For a *numbered list*, do the same thing except use the `ol` directive rather than the `ul` directive. For example:
- ```
<ol>
<li> First item goes here.
<li> Second item goes here.
</ol>
```
- Lists can be arbitrarily nested: any list item can itself contain lists.
 - Also note that no paragraph separator (or anything else) is necessary at the end of a list item; the subsequent `` tag (or list end tag) serves that role. (One can also have a number of paragraphs, each themselves containing nested lists, in a single list item, and so on.)
 - A *description list* usually consists of alternating "description titles" (`dt`'s) and "description descriptions" (`dd`'s). Think of a description list as a glossary: a list of terms or phrases, each of which has an associated definition.



BLOCK LEVEL TAGS , TEXT LEVEL TAGS AND GRAPHICS

Unit Structure

4.1 Block Level Tags

- Block formatting, Heading, Paragraph, Comments, Text alignment, Font size

4.2 Text Level Tags

- Bold, Italic, Underlined, Strikethrough, Subscript, superscript

4.3 Web publishing

4.3 Inserting graphics, Scaling images

4.1 BLOCK LEVEL TAGS

HTML includes several tags to delimit, and hence, format paragraphs of text. These tags include the following:

- `<p>` —Formatted paragraphs
- `
` Break
- `<h1>` through `<h6>` —Headings
- `<blockquote>` —Quoted text
- `<pre>` —Preformatted text
- `<center>` —Centered text
- `<div>` —A division of the document
- `<HR>` horizontal Line

1. **Formatted paragraphs** `<p>`

- Tag Type : Container
- Function:
 - Denotes a paragraph.
- Syntax:

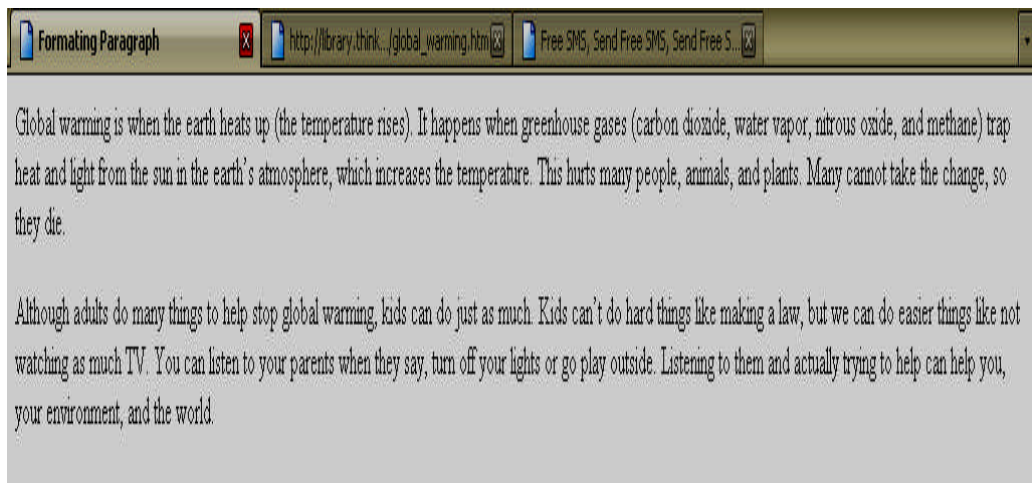

```
<p align = "LEFT|RIGHT|CENTER|JUSTIFY">
...PARAGRAPH... </p>
```

Example:

```

<html>
<head> <Title> Formating Paragraph </title> </head>
<body>
<p> Global warming is when the earth heats up (the temperature
rises). It happens when greenhouse gases (carbon dioxide, water
vapor, nitrous oxide, and methane) trap heat and light from the
sun in the earth's atmosphere, which increases the temperature.
This hurts many people, animals, and plants. Many cannot take
the change, so they die. </p>
<p> Although adults do many things to help stop global warming,
kids can do just as much. Kids can't do hard things like making a
law, but we can do easier things like not watching as much TV.
You can listen to your parents when they say, turn off your lights
or go play outside. Listening to them and actually trying to help
can help you, your environment, and the world. </p>
</body>
</html>

```

**2. Preserving Formatting—The `<pre>` Element**

- Tag type: container
- Function:
 - Denotes text to be treated as preformatted. Browsers render preformatted text in fixed in a fixed with font.
 - Sometimes you will want the client browser to interpret your text literally, including the white space and forced

4 rows in set (0.00 sec) </p>

Not only is this a lot of work, but it also renders the code practically illegible. A

better way is to simply use the <pre> tag, as follows:

```
<pre>
```

```
mysql> select * from settings;
```

```
+-----+-----+
```

```
| name | value |
```

```
+-----+-----+
```

```
| newupdate | 1069455632 |
```

```
| releaseupdate | Tues, 1/28, 8:18pm|
```

```
| rolstatus | 0 |
```

```
| feedupdate | 1069456261 |
```

```
+-----+-----+
```

```
4 rows in set (0.00 sec)
```

```
</pre>
```



```
mysql> select * from settings;
```

```
+-----+-----+
```

```
| name      | value      |
```

```
+-----+-----+
```

```
| newupdate | 1069455632 |
```

```
| releaseupdate | Tue, 1/28, 8:18pm |
```

```
| status     | 0          |
```

```
| feedupdate | 1069456261 |
```

```
+-----+-----+
```

```
4 rows in set (0.00 sec)
```

Not only is this a lot of work, but it also renders the code practically illegible. A better wa

tag, as follows:

```
mysql> select * from settings;
```

```
+-----+-----+
```

```
| name | value |
```

```
+-----+-----+
```

```
| newupdate | 1069455632 |
```

```
| releaseupdate | Tues, 1/28, 8:18pm|
```

```
| rolstatus | 0 |
```

```
| feedupdate | 1069456261 |
```

```
+-----+-----+
```

```
4 rows in set (0.00 sec)
```

3. Headings `<h1>` to `<h6>`

HTML supports six levels of headings. Each heading uses a large, usually bold character-formatting style to identify itself as a heading. The following HTML example produces the output shown in Figure :

```
<html>
<body>
<h1> Heading 1 </h1>
<h2> Heading 2 </h2>
<h3> Heading 3 </h3>
<h4> Heading 4 </h4>
<h5> Heading 5 </h5>
<h6> Heading 6 </h6>
<p> Plain body text:The quick brown fox jumped over the lazy
dog. </p>
</body>
</html>
```

Heading 1

Heading 2

Heading 3

Heading 4

Heading 5

Heading 6

Plain body text: The quick brown fox jumped over the lazy dog

4. Quoted text

The `<blockquote>` tag is used to delimit blocks of quoted text.

For example, the following code identifies the beginning paragraph of the Declaration of Independence as a quote:

```
<body>
```



```

<p> The Declaration of Independence begins with the following
paragraph: </p>
<blockquote>
When in the Course of human events, it becomes necessary for
one people to dissolve the political bands which have connected
them with another, and to assume among the powers of the earth,
the separate and equal station to which the Laws of Nature and of
Nature's God entitle them, a decent respect to the opinions of
mankind requires that they should declare the causes which impel
them to the separation.
</blockquote>
</body>

```

The `<blockquote>` indents the paragraph to offset it from surrounding text, as shown in Figure.



The Declaration of Independence begins with the following paragraph:

When in the Course of human events, it becomes necessary for one people to dissolve the political bands which have connected them with another, and to assume among the powers of the earth, the separate and equal station to which the Laws of Nature and of Nature's God entitle them, a decent respect to the opinions of mankind requires that they should declare the causes which impel them to the separation.

Figure: The `<blockquote>` tag indents the paragraph.

5. Comments `<!-- -->`

Although HTML documents tend to be fairly legible, there are several advantages to adding comments to your HTML code. HTML uses the tag `<!--` to begin a comment and `-->` to end a comment. Note that the comment can span multiple lines, but the browser will ignore anything between the comment tags. For example, the following two comments will both be ignored by the browser:

<!-- This section needs better organization. -->

and

<!-- The following table needs to include these columns:

Age

Marital Status

Employment Date

-->

**6. Line Breaks
**

Inserts the line break in documents.

Example:

```
<html>
```

```
<body>
```

This is the first statement.
 this is the second statements.

```
</body>
```

```
</html>
```

7. Horizontal Rules <hr>

Horizontal rules are used to visually break up sections of a document. The <hr> tag creates a line from the current position in the document to the right margin and breaks the line accordingly.

So basically places horizontal line on page.

Syntax

<HR>

Attribute

Specifications

- ALIGN = [left | center | right] (horizontal alignment)
- NOSHADE (solid line)
- SIZE = *Pixels* (line height)
- WIDTH = *Length* (line width)
- common attributes

Contents


Standalone

Contained in

APPLET, BLOCKQUOTE, BODY, BUTTON, CENTER, DD, DEL, DIV, FIELDSET, FORM, IFRAME, INS, LI, MAP, NOFRAMES, NOSCRIPT, OBJECT, TD, TH

8. Grouping with the <div> Element

The <div> tag defines a division or a section in an HTML document. The <div> tag is often used to group block-elements to format them with styles. For example, consider the following code, which uses styles and paragraph tags:

<pre> <html> <body> <h3> This is a header </h3> <p> This is a paragraph. </p> <div style = "color:#00FF00" > <h3> This is a header </h3> <p> This is a paragraph. </p> </div> </body> </html> </pre>	
--	--

Optional Attributes

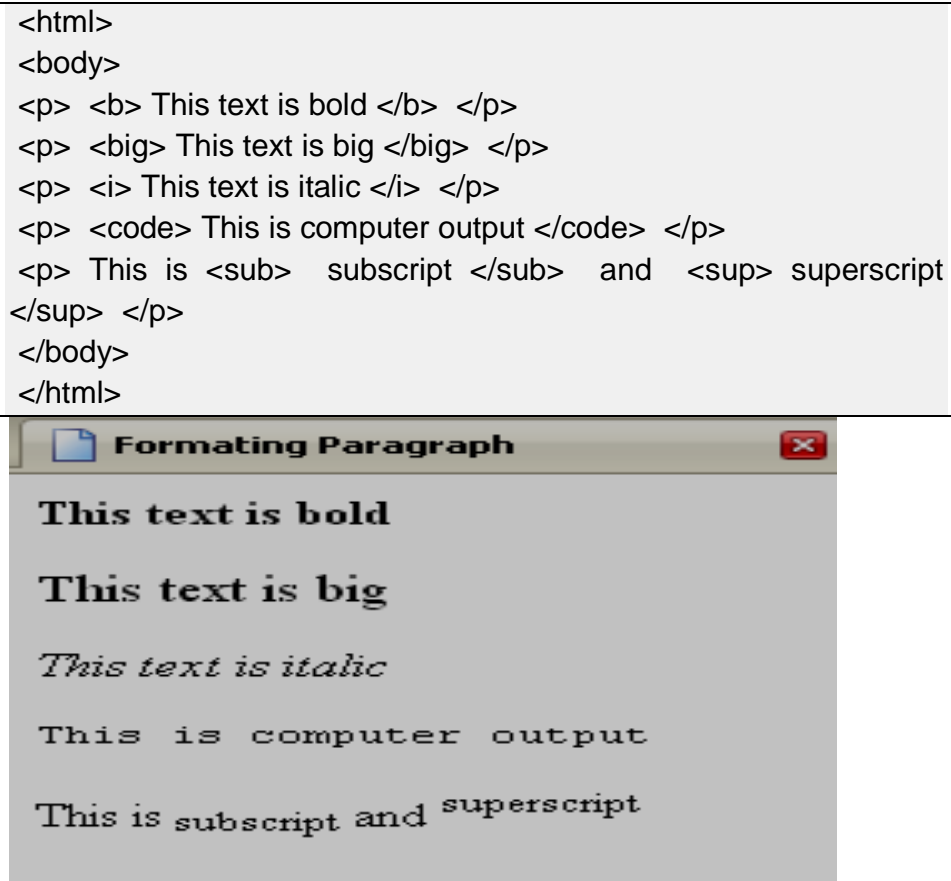
Attribute	Value	Description
align	left right center justify	Deprecated. Use styles instead. Specifies the alignment of the content inside a div element

4.2 TEXT LEVEL TAGS

- **TT** teletype or monospaced text
 - The quick red fox jumped over the lazy brown dog.
- **I** italic text style
 - *The quick red fox jumped over the lazy brown dog.*
- **B** bold text style
 - **The quick red fox jumped over the lazy brown dog.**
- **U** underlined text style
 - The quick red fox jumped over the lazy brown dog.
- **STRIKE** strike-through text style
 - ~~The quick red fox jumped over the lazy brown dog.~~
- **BIG** places text in a large font
 - The quick red fox jumped over the lazy brown dog.
- **SMALL** places text in a small font
 - The quick red fox jumped over the lazy brown dog.

- **SUB** places text in subscript style
 - The fox_{red} jumped over the dog_{brown}.
- **SUP** places text in superscript style
 - The fox^{quick} jumped over the dog^{lazy}.

Example:



4.3 INSERTING IMAGE

-
- Tag type: Stadalone
- Function:
 - Images can be placed in a web page by using tag.
 - The gif format is considered superior to the jpeg format for its clarity and ability to maintain the originality of an image without lowering its quality.

Appearance:

Attributes: **SRC=URL**, ALT=string,
ALIGN=left|right|top|middle|bottom, HEIGHT=n,
WIDTH=n, BORDER=n, HSPACE=n, VSPACE=n,
USEMAP=URL, ISMAP

Syntax:

SRC – Source of the image file

image_URL – represents the image file with its location.

Example:

Here, image_URL = file:///C:/sunset.GIF, it means image is available in the Hard Drive C: on the local hard disk.

This SRC attribute is mandatory for the tag.

Other attributes used with are: -

- **ALIGN:** used to set the alignment of the text adjacent to the image. It takes the following values:
 - **ALIGN = LEFT** - Displays image on left side and the subsequent text flows around the right hand side of that image
 - **ALIGN = RIGHT** - Displays the image on the right side and the subsequent text flows around the left hand side of that image
 - **ALIGN = TOP** - Aligns the text with the top of the image
 - **ALIGN = MIDDLE** - Aligns the text with the middle of the image
 - **ALIGN = BOTTOM** - Aligns the text with the bottom of the image
- By default, the text is aligned with the bottom of the image
- **HEIGHT and WIDTH:** Height and Width of an image can be controlled by using the HEIGHT and WIDTH attributes in the tag as follows:
 - Example:
- **HSPACE and VSPACE:** White space around an image can be provided by using HSPACE (Horizontal Space) and VSPACE (Vertical Space) attributes of the tag. These attributes provide the space in pixels.

Example:

```

<HTML>
  <HEAD>
    <TITLE> Use of IMG Tag With its ALIGN Attribute</TITLE>
  </HEAD>
  <BODY>
    <P>
      <IMG SRC=NOSlogo.GIF ALIGN=TOP>
      Aligns the text with the Top of the image
    </P>
    <P>
      <IMG SRC=NOSlogo.GIF ALIGN=MIDDLE>
      Aligns the text with the Middle of the image
    </P>
    <P>
      <IMG SRC=NOSlogo.GIF ALIGN=LEFT>

      Displays image on left side and the subsequent text flows
      around the right hand side of that image. Displays image
      on left side and the subsequent text flows around the
      right hand side of that image. Displays image on left side
      and the subsequent text flows around the right hand side
      of that image.
    </P>
    <P>
      <IMG SRC=NOSlogo.GIF ALIGN=RIGHT>

      Displays image on right side and the subsequent text flows
      around the left hand side of that image. Displays image
      on right side and the subsequent text flows around the left
      hand side of that image.
    </P>
  </BODY>
</HTML>

```



➤ **ALT (Alternative Text)**

- This attribute is used to give alternative text that can be displayed in place of the image
- This is required when the user needs to stop display of images while retrieving a document in order to make the retrieval faster, or when the browser does not support graphics. It is also used a tool tips – displaying description of the image when the mouse is over the image.
- Example: ``

➤ **BORDER**

- Border around the image can be controlled by using BORDER attribute of `` tag.
- By default image displays with a thin border.
- To change the thickness or turn the border off, the value in pixels should set to BORDER attribute.
- Example: ``
- `BORDER = 0` to Turn the Border off

Example:

Summary:

- Certain HTML elements that may appear in BODY are said to be "block-level" while others are "inline" (also known as "text level").
- In this unit we have covered block level formatting and text level formatting which is used for making HTML page.
- inline elements may contain only data and other inline elements.
- They do not create new lines.
- block-level elements may contain inline elements and other block-level elements.
- They begin on new lines, and will close an unterminated paragraph element.
- This enables you to omit end-tags for paragraphs in many cases.
- Block level tags are: <p>,
 , <h1> through <h6> , <blockquote>, <pre> , <center> , <div> , <HR>
- Text level elements that define character styles can generally be nested.

Logical markup	Physical markup
<ul style="list-style-type: none"> • <u>EM</u> - Emphasized text • <u>STRONG</u> - Strongly emphasized • <u>DFN</u> - Definition of a term • <u>CODE</u> - Code fragment • <u>SAMP</u> - Sample text • <u>KBD</u> - Keyboard input • <u>VAR</u> - Variable • <u>CITE</u> - Short citation 	<ul style="list-style-type: none"> • <u>TT</u> - Teletype • <u>I</u> - Italics • <u>B</u> - Bold • <u>U</u> - Underline • <u>STRIKE</u> - Strikeout • <u>BIG</u> - Larger text • <u>SMALL</u> - Smaller text • <u>SUB</u> - Subscript • <u>SUP</u> - Superscript

- They can contain other text level elements but not block level elements

Question:

1. Write a program to demonstrate Lists.
2. What are the different types of lists?
3. Explain tags with its properties and attributes:
 - a. <HEAD>
 - b. <BODY>
 - c. <TITLE>
 - d.
 - e.
4. Explain all text level tags
5. Explain all Block level tags
6. Explain with properties
 - a. img
 - b. Alt
 - c. border



FRAMESET AND FORMS

Unit Structure

5.1 Frameset

5.2 Forms

5.1 FRAMESET

Syntax	<FRAMESET>...</FRAMESET>
Attribute	<ul style="list-style-type: none"> • ROWS = <i>MultiLengths</i> (row lengths)
Specifications	<ul style="list-style-type: none"> • COLS = <i>MultiLengths</i> (column lengths) • ONLOAD = <i>Script</i> (all frames have been loaded) • ONUNLOAD = <i>Script</i> (all frames have been removed) • <u>core attributes</u>
Contents	One or more FRAMESET and FRAME elements, as well as an optional <u>NOFRAMES</u>
Contained in	<u>HTML</u>

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

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

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

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

<Frame>

Syntax**<FRAME>****Attribute****Specifications**

- NAME = CDATA (name of frame)
- SRC = URI (content of frame)
- LONGDESC = URI (long description of frame)
- FRAMEBORDER = [1 | 0] (frame border)
- MARGINWIDTH = Pixels (margin width)
- MARGINHEIGHT = Pixels (margin height)
- NORESIZE (disallow frame resizing)
- SCROLLING = [yes | no | *auto*] (ability to scroll)
- core attributes

Contents

Empty

Contained inFRAMESET

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

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

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

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

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

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

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

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

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

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

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

```

<FRAMESET ROWS = "*,100">
  <FRAMESET COLS = "40%,*">
    <FRAME NAME = "Menu" SRC = "nav.html" TITLE =
"Menu">
    <FRAME NAME = "Content" SRC = "main.html" TITLE =
"Content">
  </FRAMESET>
  <FRAME NAME = "Ad" SRC = "ad.html" TITLE =
"Advertisement">
<NOFRAMES>
  <BODY>
    <H1>Table of Contents</H1>
    <UL>
      <LI>
        <A HREF = "reference/html40/"> HTML 4 Reference </A>
      </LI>
      <LI>
        <A HREF = "reference/wilbur/"> HTML 3.2 Reference </A>
      </LI>
      <LI>
        <A HREF = "reference/css/"> CSS Guide </A>
      </LI>
    </UL>
    <P>
      <IMG SRC = "ad.gif" ALT = "Ad: Does your bank charge too
much?">
    </P>
  </BODY>
</NOFRAMES>
</FRAMESET>

```

The following example sets up a grid with two rows and three columns:

```
<FRAMESET ROWS = "70%,30%" COLS = "33%,33%,34%">
  <FRAME NAME = "Photo1" SRC = "Row1_Column1.html">
  <FRAME NAME = "Photo2" SRC = "Row1_Column2.html">
  <FRAME NAME = "Photo3" SRC = "Row1_Column3.html">
  <FRAME NAME = "Caption1" SRC = "Row2_Column1.html">
  <FRAME NAME = "Caption2" SRC = "Row2_Column2.html">
  <FRAME NAME = "Caption3" SRC = "Row2_Column3.html">
  <NOFRAMES>
  <BODY>
    <H1>Table of Contents</H1>
    <UL>
      <LI>
        <A HREF = "Row1_Column1.html">Photo 1</A>
        (<A HREF = "Row2_Column1.html">Caption</A>)
      </LI>
      <LI>
        <A HREF = "Row1_Column2.html">Photo 2</A>
        (<A HREF = "Row2_Column2.html">Caption</A>)
      </LI>
      <LI>
        <A HREF = "Row1_Column3.html">Photo 3</A>
        (<A HREF = "Row2_Column3.html">Caption</A>)
      </LI>
    </UL>
  </BODY>
</NOFRAMES>
</FRAMESET>
```

The next example features nested **FRAMESET** elements to define two frames in the first row and one frame in the second row:

```

<FRAMESET ROWS = "*,100">
  <FRAMESET COLS = "40%,*">
    <FRAME NAME = "Menu" SRC = "nav.html" TITLE = "Menu">
    <FRAME NAME = "Content" SRC = "main.html" TITLE =
"Content">
  </FRAMESET>
  <FRAME NAME = "Ad" SRC = "ad.html" TITLE =
"Advertisement">
<NOFRAMES>
  <BODY>
    <H1>Table of Contents</H1>
    <UL>
      <LI>
        <A HREF = "reference/html40/">HTML 4 Reference</A>
      </LI>
      <LI>
        <A HREF = "reference/wilbur/">HTML 3.2 Reference</A>
      </LI>
      <LI>
        <A HREF = "reference/css/">CSS Guide</A>
      </LI>
    </UL>
    <P>
      <IMG SRC = "ad.gif" ALT = "Ad: Does your bank charge too
much?">
    </P>
  </BODY>
</NOFRAMES>
</FRAMESET>

```

The **FRAMESET** element also accepts **ONLOAD** and **ONUNLOAD** attributes to specify client-side scripting actions to perform when the frames have all been loaded or removed.

NOFRAMES - Frames Alternate Content

Syntax **<NOFRAMES>...</NOFRAMES>**

Attribute • common attributes

Specifications

Contents • In HTML 4 Transitional: inline elements, block-level elements
 • In HTML 4 Frameset: one BODY element that must not contain any NOFRAMES elements

Contained in APPLET, BLOCKQUOTE, BODY, BUTTON, CENTER, DD, DEL, DIV, FIELDSET, FORM, FRAMESET, IFRAME, INS, LI, MAP, NOSCRIPT, OBJECT, TD, TH

The **NOFRAMES** element contains content that should only be rendered when *frames are not displayed*. **NOFRAMES** is typically used in a Frameset document to provide alternate content for browsers that do not support frames or have frames disabled.

When used within a **FRAMESET**, **NOFRAMES** must contain a **BODY** element. There must not be any **NOFRAMES** elements contained within this **BODY** element.

A meaningful **NOFRAMES** element should always be provided in a Frameset document and should at the very least contain links to the main frame or frames. **NOFRAMES** should not contain a message telling the user to upgrade his or her browser. Some browsers support frames but allow the user to disable them.

```
<FRAMESET ROWS = "*,100">
  <FRAMESET COLS = "40%,*">
    <FRAME NAME = "Menu" SRC = "nav.html" TITLE = "Menu">
    <FRAME NAME = "Content" SRC = "main.html" TITLE =
"Content">
  </FRAMESET>
  <FRAME NAME = "Ad" SRC = "ad.html" TITLE =
"Advertisement">
```



```

<NOFRAMES>
<BODY>
  <H1>Table of Contents</H1>
  <UL>
    <LI>
      <A HREF = "reference/html40/">HTML 4 Reference</A>
    </LI>
    <LI>
      <A HREF = "reference/wilbur/">HTML 3.2 Reference</A>
    </LI>
    <LI>
      <A HREF = "reference/css/">CSS Guide</A>
    </LI>
  </UL>
  <P>
    <IMG SRC = "ad.gif" ALT = "Ad: Does your bank charge too
much?">
  </P>
</BODY>
</NOFRAMES>
</FRAMESET>

```

5.2 INTRODUCTION TO FORMS

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

Here's a simple form that includes labels, radio buttons, and push buttons (reset the form or submit it):

```

<FORM action = "http://somesite.com/prog/adduser" method =
"post">
  <P>
    <LABEL for = "firstname">First name: </LABEL>
      <INPUT type = "text" id = "firstname"><BR>
    <LABEL for = "lastname">Last name: </LABEL>

```

```

<INPUT type = "text" id = "lastname"><BR>
<LABEL for = "email">email: </LABEL>
  <INPUT type = "text" id = "email"><BR>
  <INPUT type = "radio" name = "sex" value = "Male"> Male<BR>
  <INPUT type = "radio" name = "sex" value = "Female">
Female<BR>
  <INPUT type = "submit" value = "Send"> <INPUT type = "reset">
</P>
</FORM>

```

The screenshot shows a web browser window with the title bar "file:///C:/Docu...HTML/page1.html". The form content is as follows:

First name:

Last name:

email:

Male

Female

Controls

Users interact with forms through named *controls*.

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

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

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

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

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

➤ Text Fields

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

```
<form>
```

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

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

```
</form>
```

How the HTML code above looks in a browser:

First name:

Last name:

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

`<input>`

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

`type`

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

`name`

When you get the results of this form in your email, you're going to need to know which responses were placed in which boxes, as you just get back a load of text. This is where the name attribute comes in. With this, each line in the response will be given a **label** in the email. If you used

name = "firstname", because you were using this box to find out the reader's first name, you would receive firstname = Ross in the email you are sent.

size

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

➤ Password Field

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

`<form>`

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

`</form>`

How the HTML code above looks in a browser:

Password:

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

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

➤ Radio Buttons

These small circular buttons can be used in polls or information forms to ask the user their **preference**. When you set up a group of them, you can only select **one choice**. Try it here:

1. 2. 3.

They're called *radio* buttons because they function like the buttons on a really old car radio. Remember, the guys who came up with this stuff have beards as long as your arm, so you should expect things like that. The code for a radio button is:

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

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

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

How the HTML code above looks in a browser:

- Male
 Female

The code is about the same as the one you've seen before. `type = "radio"` says that this is going to be a radio button. There is a new attribute here too — `value`. This is like the answer to a question. Say you were asking the reader what they liked most about your site. The name of this group of questions would be 'likemost' and there would be three choices, all radio buttons, all with `name = "likemost"` in them, to show that they're all part of the same question. Then the three values could be 'layout', 'content' and 'graphics'. When you receive the results, you'll get something like *likemost = layout* depending on which button was checked. Get it? I should tell you now that you can add the `value` attribute to the single-line text box above to add in some text before the user even starts typing.

➤ Check Boxes

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

1. 2. 3.

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

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

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

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

How the HTML code above looks in a browser:

- I have a bike
- I have a car

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

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

Looks a little silly with the attribute's value being the same as the attribute itself, but that's the way it's done. This checked attribute can also be used on a radio button to set one of the radios as selected by default.

➤ Drop-down Select Boxes

These are a cool way to get a user to select an option. They perform the same thing as radio buttons, it's just the way they look that's different. Most of the options available are not in view until the user gets intimate with the box and clicks on it. The rest of the options will then pop-up below the box.



This box would be used to find out what continent you come from, like I care. The lengthy code is:

```

<select name = "continent" size = "1">
  <option value = "europe">europe</option>
  <option value = "namerica">n. america</option>
  <option value = "samerica">s. america</option>
  <option value = "asia">asia</option>
  <option value = "africa">africa</option>
  <option value = "oz">the other one</option>
</select>

```

select boxes are like textareas — they have their own tag, but these elements hold further tags inside them too. Each choice you give your reader is denoted by an option. The name/value system remains from the tags above. The size attribute sets how many lines of options are displayed. Setting this to anything over 1 (the default) is really defeating the purpose of having the options hidden away.

➤ **Send and Reset Buttons**

Now that you've gotten the reader to fill in all the information you want, you need a finishing button to click on to send it all to your email address (or wherever you've said at the start). You can also clear all the info in the form out with the reset button. They're both real easy to make, and look like this:



The simple tags for these two are:

```

<input type = "submit" value = "Submit">
<input type = "reset">

```

The value attribute in this case sets the text that's displayed on the front of the button. When you click submit all the form information is sent to your target and the page (or following page) is loaded. Done.

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

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

How the HTML code above looks in a browser:

Username:

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

Question:

1. Explain all tags of forms and frame
2. Write an html code to design a resgistration.htm form using all tags.



CASCADING STYLE SHEETS

Unit Structure

- 6.1 The usefulness of style sheets
- 6.2 Creating style sheets
- 6.3 Common tasks with CSS

6.1 WHAT IS CSS STYLE?

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

Using the <style> element

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

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

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

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

```
<style>  
h2 {color:red;font-family:arial;font-size:14pt}  
</style>
```

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

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

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

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

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

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

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

Use of Style Sheet

Understanding the Style Sheet Cascade

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

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

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

mystyles.css

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

index.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<link rel="stylesheet" type="text/css"
href="mystyles.css" />
<style type="text/css">
h1 { color: Red; }
</style>
</head>
<body>
<h1>A Sample Heading</h1>
```

...

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

How do I use CSS?

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

It is actually completely readable – this style sheet defines that all content within the HTML **BODY** element will use font Verdana with size of 9 points and will align it to the right. But, if there's a **DIV** element, the text within that will be written in font Georgia. We're also using a class named "important" (classes use "." notation, which we will cover later on). All elements of this class will have a set background color, a border and will use Franklin Gothic Book font. As you see, style definitions for a certain element or class are written inside curly braces (“{ }”) and each line ends with a semicolon “;”.

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

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

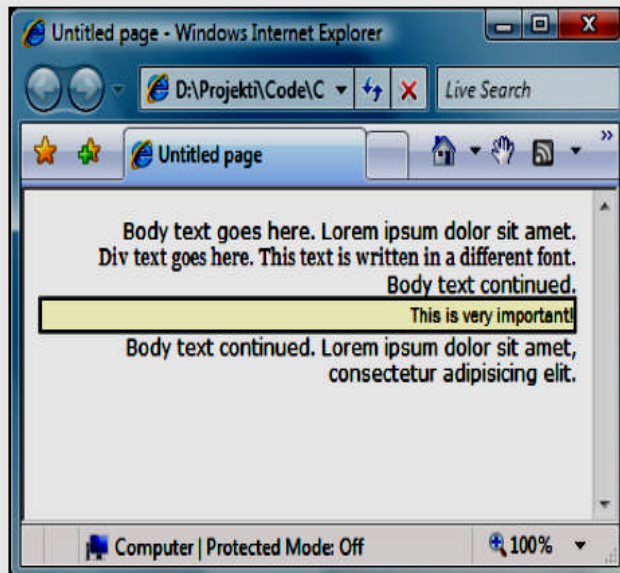
Here are the elements that would be affected by the sample CSS file.

```

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

```

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



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

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

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

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

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

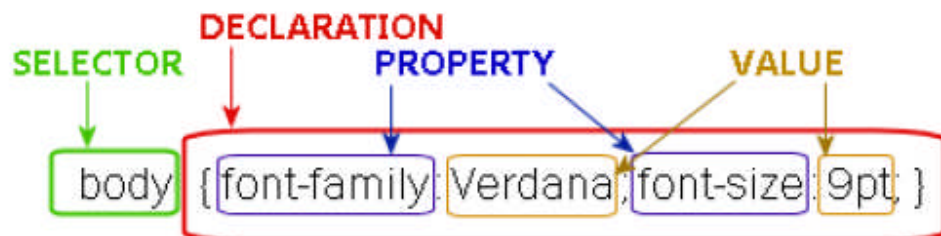
```
        border: thin black ridge;
        font-family: Franklin Gothic Book;
      }
    </style>
  </head>
  <body>
    <div class="important">My content</div>
  </body>
</html>
```

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

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

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

CSS style definition syntax



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

And here are 6 rules of style definitions:

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

4. A property can have multiple values separated with a comma (e.g. "Verdana, Arial, Franklin Gothic Book").
5. Along with a value, can also be a unit of measure (e.g. "9pt", where "pt" stands for *points*). No space is allowed between the value and the unit.
6. When writing CSS code, you can use whitespaces as needed – new lines, spaces, whatever makes your code more readable.

6.2 CREATING STYLE SHEET

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

Inline style

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

Example: ex02-01.htm - Inline CSS Style

```
<html>
  <head><title> Inline CSS Style - ex02-01.htm</title></head>
  <body style="font-family:Times;font-weight: bold; background:
#000088">
    <div style="font-size:20pt;text-align:center;color:#00ffff">
      Inline CSS Style </div>
    <p style="font-family:arial;font-size:16pt;color:#ffff00;
margin-left:20px;margin-right:20px">
      With CSS, we can control the margins of an element.
      This is a paragraph with margin-left:20px and margin-right:20px.
    </p>
  </body>
</html>
```



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

The division element <div> in line 8 has all the CSS properties from <body> with some additional definitions. A division is similar to a paragraph but without an additional line break. Next to this division, there is a paragraph element <p> (line 10). This paragraph changes the default font family to "arial" and adds some margin controls. When an element has two or more of the same CSS definitions, the earlier ones will be overridden by the latest one. That is, the styles will be cascaded into one.

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

➤ **The embedded style element <style>**

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

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

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

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

Example: ex02-02.htm - The Style Element `<style>` I

```
<html>
<head><title> The Style Element <style> I - ex02-
02.htm</title></head>
<style type="text/css">
  h2 {color:#00ffff;font-size:20pt;text-align:center}
  h4 {margin-left:70%}
  p {font-family:arial;font-size:16pt;color:#ffff00;
    margin-left:20px;margin-right:20px}
  body {font-family:arial;font-size:14pt;color:#ffff00;
    background-image: url("backgr01.jpg")}
</style>
</head>
<body>
  <h2>Internal CSS Style</h2>
  <h4>This area was created by CSS margin
margin-left:70% and margin-right:20%</h4>
  <p>With CSS, you can control text font, color, dimension,
position,
margin, background and much more ...</p>
</body>
</html>
```

Lines 6-13 define an internal style. This adds CSS information to a Web page. Line 7 assigns the level 2 heading with color #00ffff, font size 20pt, and text centrally aligned. Line 8 sets the left and right margins of the level 4 heading to be 70% and 20% of the element's width respectively. Line 9 defines the default font typeface "arial," font size, color value, and left and right margins of a paragraph element. The body also has a background image backgr01.jpg. This page has a screen output as shown in [Fig. 6.2](#).

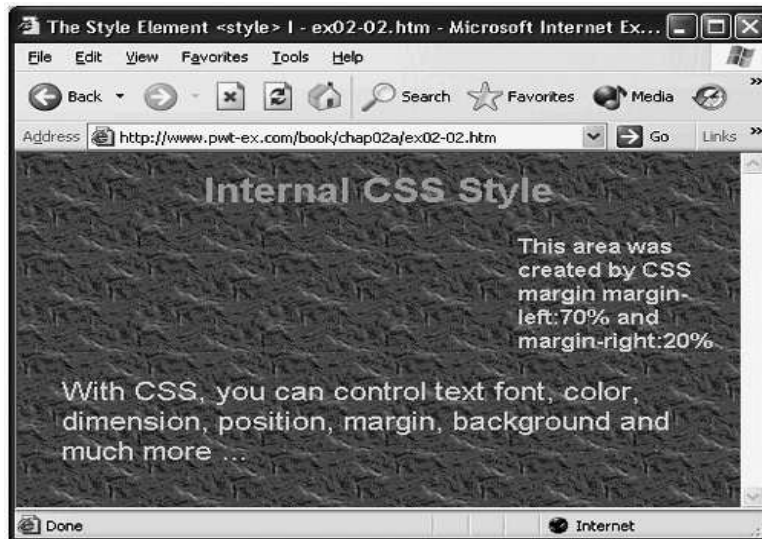


Figure 6.2. ex02-02.htm

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

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

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

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

```
<html>
  <head><title> The Style Element <style> II - ex02-
03.htm</title></head>
  <style type="text/css">
    .txtSt {font-family:arial;color:#ffff00;font-size:20pt;
      font-weight:bold}
```

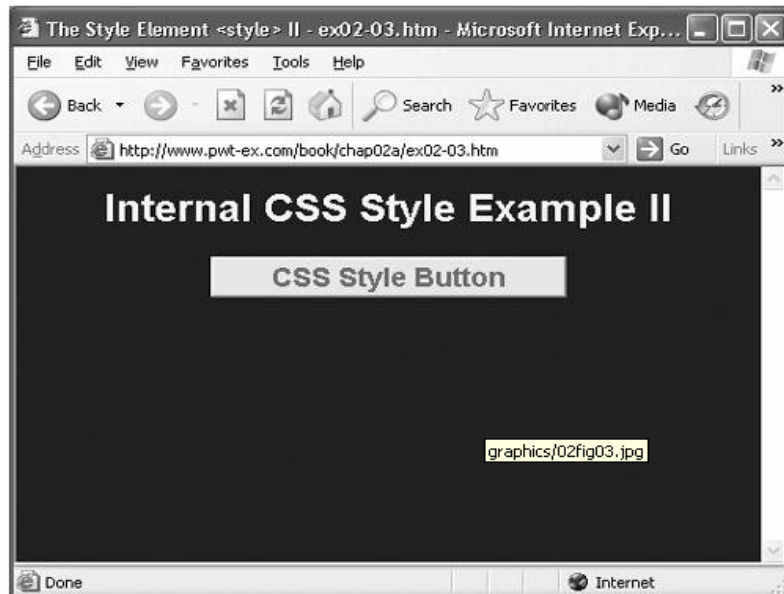
```

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

```

The screen shot is shown in [Fig. 6.3](#). In this example, line 7 defines the CSS class with the unique name `txtSt` with appropriate CSS properties. Lines 910 define another class `butSt` for a button. All elements that you named `class="textSt"` will have the `.txtSt` class attributes. Similarly the `<input>` element that has `class="butSt"` will use the `.butSt` attributes to format the button on the Web

Figure 6.3. ex02-03.htm



➤ External CSS style sheets

An external style sheet is ideal when the style is applied to many pages. The style information is placed in a separate document with the file extension `.css`. With an external style sheet, you can change the look of an entire Web site by changing the corresponding style information file. Each page must link to the

style sheet using the <link> element, which usually goes within the <head> section. For example,

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

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

An external style sheet can be written in any text editor and should be saved with the file extension .css. You should also be sure either that this file is in the root directory with the HTML files that you intend to process or that the link is coded appropriately. An example of a style sheet file is shown below.

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

Example: ex02-04.htm - External CSS Style

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

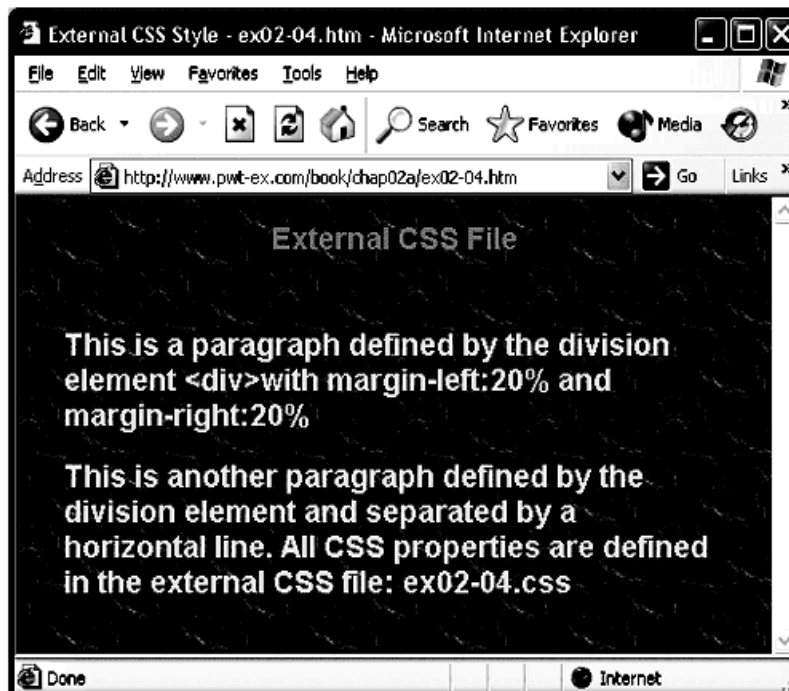
This page includes a link (line 6) to an external style sheet called [ex02-04.css](#). This file defines all the default formatting features used inside the page. The corresponding external CSS style sheet [ex02-04.css](#) is given next:

Example: ex02-04.css - Eternal CSS File For ex02-04.htm

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

Any page containing this link adopts the styles defined in the external style sheet [ex02-04.css](#). In this example, the horizontal rule line `<hr>` is changed to the color sienna. Additional margin control is added to the `<div>` element and the element `<body>` is given a different style definition. Bold "arial" and color value #ffff00 in a font size of 14 points are used as default attributes. A background image backgr01.jpg is also added to specify graphics as background images. This page has a screen output as shown in [Fig. 6.4](#).

Figure 6.4. ex02-04.htm



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

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



FONT FAMILY & PROPERTIES

Unit Structure

7.1 Font Family

- Font Metrics
- Units

7.2 Properties

7.1 FONT FAMILY

7.1.1 Working with Font Styling Attributes

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

7.1.2 Naming font families using CSS

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

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

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

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

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

Listing 18-3 shows a brief example of creating both an inline style and calling a class selector to name a font family.

Listing 7.1: Using Class Selector and Inline Style to Name a Font Family

```
<html>
<head>
<title>Font sizes</title>
<style type="text/css">
<!--
.myFontClass {font-family: "Helvetica Narrow", sans-
serif}
-->
</style>
</head>
<body>
<p>This is an <span style="font-family: 'Helvetica Narrow',
sans-serif">inline</span> style.</p>
<p>This uses a <span class="myFontClass">class
selector</span></p>
</body>
</html>
```

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

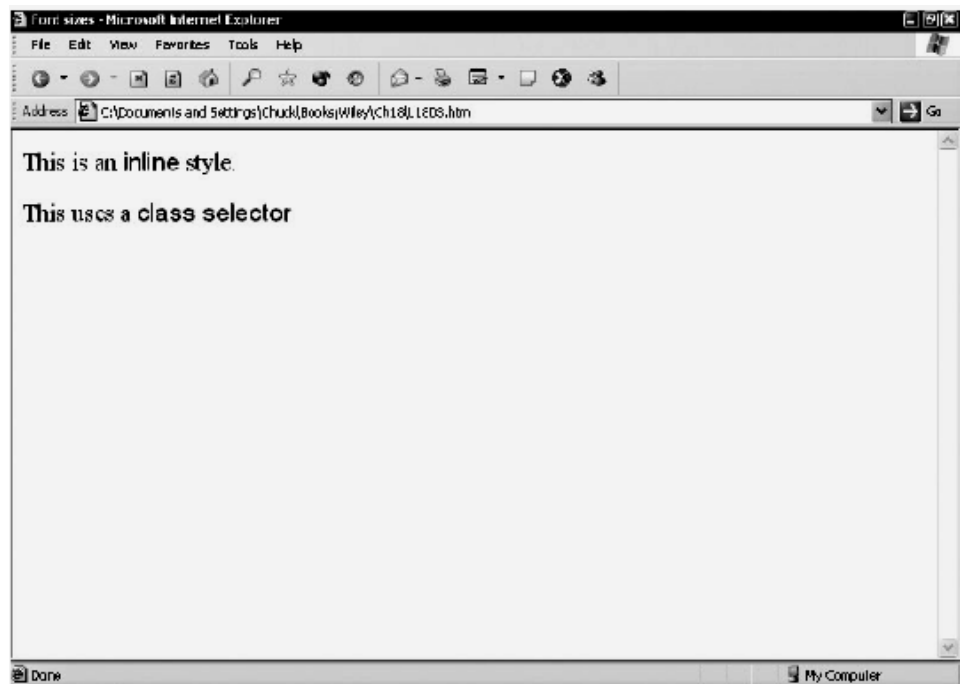


Figure 18-5: Rendering inline and class selector styles in the browser.

7.1.3 Working with font styles

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

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

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

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

7.1.4 Establishing font sizes

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

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

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

Other relative sizes include the following:

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

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

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

xx-small, x-small, small, medium, large, x-large, xx-large

Other absolute sizes include the following:

- ◆ pt for points. This is appropriate for pages that are used for printing, but is not a particularly reliable measure for managing screen-based fonts.
- ◆ in (inches), cm (centimeters), mm (millimeters), and pc (picas) are all rarely used on the Web, because they're designed with print production in mind.

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

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

Attribute

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

```

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

```

7.1.5 Bolding fonts by changing font weight

Font weight refers to the stroke width of a font. If a font has a very thin, or light, stroke width, it will have a weight of 100. If it has a thick, or heavy, stroke width, it will be 900. Everything else is inbetween. To denote font width, you use a numeric set of values from 100 to 900 in increments of 100: 100, 200, 300, 400, and so on. Or, you can use the keywords bold, normal, bolder or lighter to set a value, which will be relative to the font weight of the element containing the font. The keyword bold is equal to the numeric value 700. An example of using font-weight in style rules written for a style element might be as follows:

```

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

```

7.1.6 Making font wider or thinner using font stretch

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

7.1.7 Line height and leading

The CSS line-height property is another one of those nice-in-theory properties that just doesn't pan out in the real world. The syntax is supposed to let you set the space between lines in a process that in the print world is called leading. It works fairly well in Internet Explorer, but is a mess in Netscape 4. The syntax is easy enough:

```
line-height: normal
line-height: 1.1
line-height: 110%
```

The first example in the preceding series of rules makes the line height the same as the base line height of the document. The next line makes the line height 1.1 times greater than the base line height, as does the third, except the third uses percentages as a unit of measure.

7.2 PROPERTIES

- Text Properties
- Color and background
- Box
- Font(we hav already seen that refer topic just before this)

7.2.1 Controlling text properties with style

Some frequently used CSS properties related to font are listed in Table					
<i>. Table 2.1. Font family, size, weight, style, and color</i>					
CSS property	CSS values	NS	IE	Description	CSS version
Color	#rrggbb color name	4.+	4.+	Sets the color of the font in 24-bit red, green, blue mode	CSS1
Font	font-	4.+	4.+	A shorthand property to set all font values	CSS1

Some frequently used CSS properties related to font are listed in Table

. Table 2.1. Font family, size, weight, style, and color

CSS property	CSS values	NS	IE	Description	CSS version
	family, font-size, font-style, font-weight				
Font family	Family name Generic family	4.+	4.+	A prioritized list of font family names	CSS1
Font size	Length fixed % relative	4.+	4.+	Sets the size of font	CSS1
Font style	Normal Italic Oblique	4.+	4.+	Sets the style of the font	CSS1
Font variant	Normal Small caps	4.+	4.+	Displays text in a small-caps font or normal font	CSS1
Font weight	Normal Bold Bolder Lighter	4.+	4.+	Sets the weight of the font	CSS1

7.2.2 Alignment, indent, and margins

In addition to font properties, text formatting elements and margins can also be controlled using the CSS elements, Using these elements, you can specify such things as spacing between words, indentation, alignment, positions of text, and much more. Table 2.2 lists some frequently used CSS properties on margins and text alignments.

Table 2.2. Margins and alignments					
CSS property	CSS values	NS	IE	Description	CSS version
Margin	margin-top margin-right margin-left margin-bottom	4.+	4.+	A shorthand property to set the margin properties in one definition	CSS1
margin-bottom	auto length %	4.+	4.+	Sets the bottom margin of an element	CSS1
margin-left	auto length %	4.+	4.+	Sets the left margin of an element	CSS1
margin-right	auto length %	4.+	4.+	Sets the right margin of an element	CSS1
margin-top	auto length	4.+	4.+	Sets the top margin of an element	CSS1

Table 2.2. Margins and alignments					
CSS property	CSS values	NS	IE	Description	CSS version
	%				
margin-align	left right center justify	4.+	4.+	Aligns the text in an element	CSS1
margin-indent	length %	4.+	4.+	Indents the first line of text in an element	CSS1

CSS can take a specific unit of measurement in length. It can be in points (pt), inches (in), centimeters (cm), or a percentage (%). The left and right margins together with the division element can be used to define a text box with arbitrary length on the browser window.

Consider the following example [ex02-07.htm](#):

Example: ex02-07.htm - Margins and Alignments

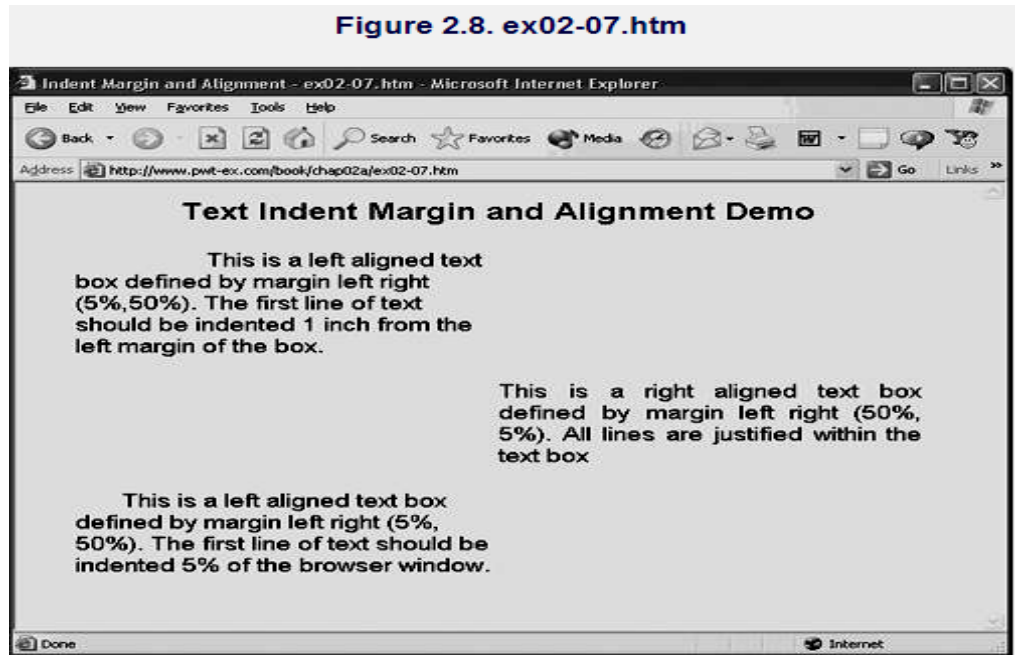
```
<html>
<head>
  <title>Indent Margin and Alignment ex02-07.htm</title>
</head>
<style type="text/css">
  .ins {font-size:14pt;font-weight:normal;text-align:left;
        text-indent:1in;margin-left:5%;margin-right:50%}
  .pts {font-size:14pt;font-weight:normal;text-align:justify;
        margin-left:50%;margin-right:5%}
  .pct {font-size:14pt;font-weight:normal;text-align:left;
        text-indent:5%;margin-left:5%;margin-right:50%}
</style>
```

```

<body style="font-family: arial; font-size: 16pt; font-weight: bold">
  <div style="font-family: arial; font-size: 18pt; text-align: center">
    Text Indent Margin and Alignment Demo</div>
  <br />
  <div class="ins">
    This is a left aligned text box defined by margin left right
(5%,50%). The first
    line of text should be indented 1 inch from the left margin of
the box.</div>
  <br />
  <div class="pts">
    This is a right aligned text box defined by margin left right
(50%, 5%). All lines
    are justified within the text box</div>
  <br />
  <div class="pct">
    This is a left aligned text box defined by margin left right (5%,
50%). The first
    line of text should be indented 5% of the browser
window.</div>
  <br />
</body>
</html>

```

In this example, three classes ins, pts, and pct are defined in lines 616. For example, the class ins sets the 14pt normal text to be indented 1 inch from the left margin of the box and left aligned. The left and right margins of the box are also set to be 5% from the left edge and 50% from the right edge of the browser window respectively. Similarly for the classes pts and pct. A screen shot of this page is shown in [Fig. 2.8](#).

Figure 2.8. ex02-07.htm

The example program [ex02-08.htm](#) shows text-decoration and text-transform in action. The corresponding screen shot is shown in [Fig. 2.9](#).

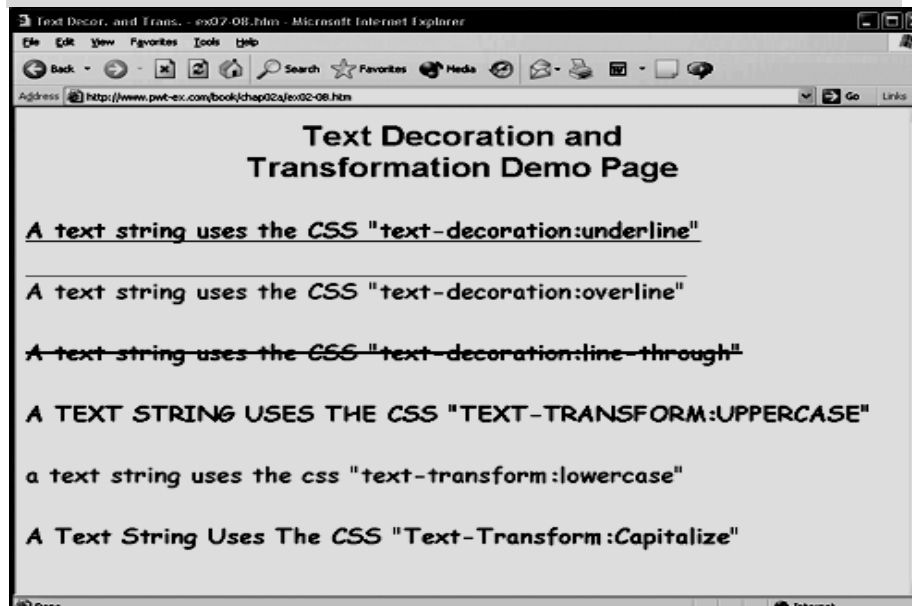
```
<html>
<head>
  <title>Text Decor. and Trans. ex02-08.htm</title>
</head>
<style>
  .bSt
  {
    font-family: 'Comic Sans MS' ,times;
    font-size: 18pt;
    color: #000088;
  }
</style>
<body style="font-family: arial; font-size: 24pt; font-weight: bold">
  <div style="text-align: center; color: #880000; text-align: center">
    Text Decoration and
    <br />
    Transformation Demo Page</div>
  <br />
  <div class="bSt" style="text-decoration: underline; color:
#000088">
```

A text string uses the CSS "text-decoration:underline"</div>

```

<br />
<div class="bSt" style="text-decoration: overline; color:
#008800">
    A text string uses the CSS "text-decoration:overline"</div>
<br />
<div class="bSt" style="text-decoration: line-through; color:
#880000">
    A text string uses the CSS "text-decoration:line-
through"</div>
<br />
<div class="bSt" style="text-transform: uppercase; color:
#000088">
    A text string uses the CSS "text-
transform:uppercase"</div>
<br />
<div class="bSt" style="text-transform: lowercase; color:
#008800">
    A text string uses the CSS "text-
transform:lowercase"</div>
<br />
<div class="bSt" style="text-transform: capitalize; color:
#880000">
    A text string uses the CSS "text-transform:capitalize"</div>
<br />
</body>
</html>

```



7.2.3 Text box dimensions and spacing

With CSS, you can scale the HTML elements it is associated with to fit the specified height and width dimensions. The CSS white-space element is a powerful element that controls the way that white space and carriage returns are handled within a Web page. It allows you to add plenty of visual space to enhance the clarity of your Web pages.

Some of the most frequently used CSS properties relating to line and character spacing are given in Table 2.4. They are all CSS1 elements and therefore fully supported by both the IE6+ and NS6+ browsers.

Table 2.4. Line and character spacing					
CSS property	CSS values	NS	IE	Description	CSS version
Height	auto length %	6.+	4.+	Sets the height of an element	CSS1
Width	auto length %	4.+	4.+	Sets the width of an element	CSS1
line-height	normal number length %	4.+	4.+	Sets the distance between lines	CSS1
white-space	normal pre nowrap	4.+	4.+	Sets how white space inside an element is handled	CSS1
letter-spacing	normal length	6.+	4.+	Increases or decreases the space between characters	CSS1
word-spacing	normal length	6.+	6.+	Increases or decreases the space between words	CSS1

These CSS properties provide you with yet more control over how your text should be displayed by the browser. For example, the CSS element word-spacing can be used to set the spacing distance between words on a Web page. Wide values can make your text easier to read, or achieve some visual effects.

The example [ex02-09.htm](#) demonstrates some of these CSS properties.

```
<html>
<head>
  <title>Line-height and Spacing ex02-09.htm</title>
</head>
<style>
  div
  {
    font-size: 14pt;
    color: #000088;
    padding: 2ex;
    margin-left: 1in;
    font-weight: normal;
    width: 6in;
  }
  .line01
  {
    line-height: 150%;
    letter-spacing: 0.2em;
  }
  .line02
  {
    line-height: 200%;
    word-spacing: 1.5em;
  }
</style>
<body>
  <div style="font-family: arial; font-size: 20pt; font-
weight: bold; color: #880000">
    CSS Line-height, Letter and Word Spacing Demo
  </div>
  <div>
    This paragraph should be leading of 100% i.e., the
    default leading produced by the
    CSS line-height property.</div>
```

```

<div class="line01">
  This paragraph should be leading of 150% produced
  by the CSS line-height property.
  The letter-spacing feature is supported in IE4 but not
  NS4. You should have no problem
  if you are using NS6+</div>
<div class="line02">
  This should be leading of 200% produced by the CSS
  line-height property. Word spacing
  is not supported by IE4 or NS4. You have no
  problems if you are using the latest
  browsers</div>
</body>
</html>

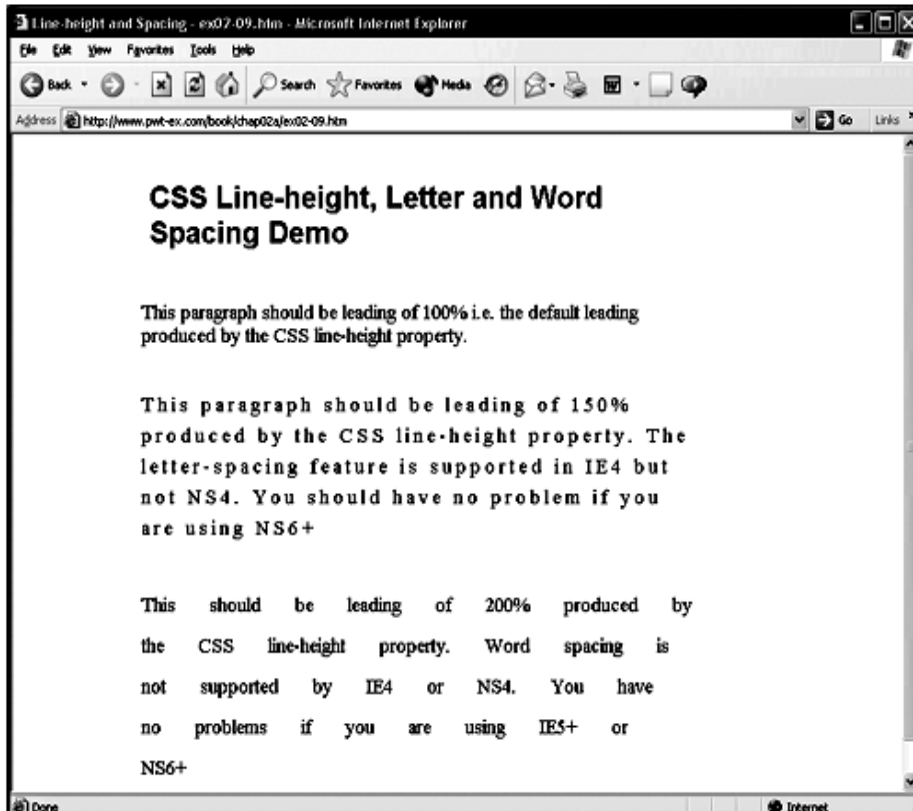
```

Three CSS properties are defined within the internal CSS style sheet in lines 613. Line 7 has an attribute `padding:2ex` which is used to add padding (of 2ex units) equally to the top, bottom, and sides of the division element. This will add visual space to the text. Line 10 defines a class `line01` that sets the spacing of 0.2em between characters. The distance between two lines is 150% in relation to the size of the font in use. The unit `em` is a measure relative to the height of the current font used. The unit `ex`, on the other hand, refers to the relative height of a lower case "x." Line 11 is another class `line02`. It sets the distance between two lines to be 200% and the spacing distance between words is 1.5em.

An interesting element is the `<div>` element in line 16. This line has an inline style that redefines the font-size (20pt) and font-weight (bold). The browser will use this new set of CSS properties to format the text that is associated with this division element. This is an example of cascading styles in practice.

A screen shot of this example is shown in [Fig. 2.10](#).

Figure 2.10. ex02-09.htm



7.2.4 Background and border

The background family of CSS style elements is used to set the background characteristics on your Web page. For example, you could apply some CSS background elements to highlight an area on a page, or just simply to enhance the contrasts of the display and the background. Another useful CSS element is border. The border properties set the display of borders around its associated CSS element. All these, together with the dimensioning and positioning CSS elements, give a variety of controls down to pixel level to help you design your pages. The dimensioning and positioning CSS elements will be discussed in more detail in [section 2.4](#).

Some frequently used background CSS elements are shown in Table 2.5.

Table 2.5. Background CSS elements				
CSS property	CSS values	NS	IE	Description
background	#rrggbb	4.+	4.+	Sets the background color or image
background-color	#rrggbb transparent	4.+	4.+	Sets the background color for an element, or sets it to transparent
background-image	image_file_name	4.+	4.+	Specifies the image_file_name as a background image
background-repeat	repeat repeat-x repeat-y no-repeat	4.+	4.+	Specifies how the background image is repeated
background-attachment	scroll fixed	4.+	4.+	Specifies background image movement when the browser window is scrolled
background-position	x y % % left/center/right top/center/bottom	4.+	4.+	Indicates the coordinates in which the background image first appears

Note that the background CSS element is the father of all the other background CSS elements, all of which share similar CSS properties for adding special background effects to your Web page. Some CSS elements like background-repeat, background-attachment, and background-position will not work unless the CSS element background-image is specified first.

Let's now have a look at the background CSS elements.

7.2.4.1 Background color and image

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

The CSS element `<background-color>` takes the general format

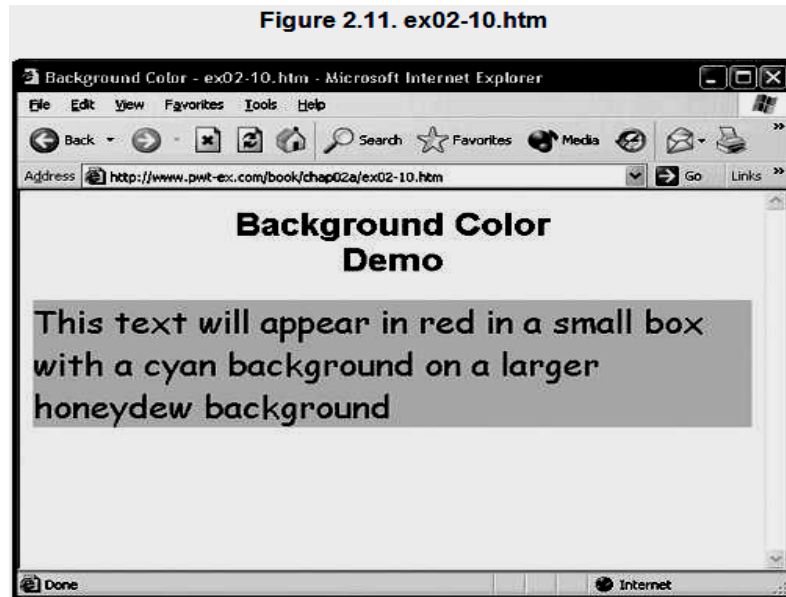
```
<b style="background-color:#rrggb">
  your body text here ...
</b>
```

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

Example: ex02-10.htm - Background Color

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

Line 7 sets a general background color for the whole page. With the CSS background-color element, you can have additional control over the background color that is associated with this element. The division element in lines 1317 uses a different color (cyan) from that of the background (honeydew color) in order to emphasize a string of text. This page has the screen output shown in [Fig. 2.11](#).

Figure 2.11. ex02-10.htm

You can also use a small picture, a photograph, or a graphic design to form a background pattern. With the `background-image` CSS properties, your small picture is tiled repeatedly in the horizontal and/or vertical directions to form the image background. If carefully arranged, this type of background can have both an unusual and original effect.

In [Chapter 1](#) we have already discussed adding images to the background of your Web page. This is a very straightforward process with the CSS element. The code `<body style="background-image:url (bg_image.gif)">` will repeatedly insert the image `bg_image.gif` into the body of the page to create a background picture. Note that once this element is specified, you can further modify the behavior of the background by using the related CSS elements such as `background-repeat`, `background-attachment`, and `background-position`.

The following example shows how to create a background consisting of the image "Practical Web" logo:

Example: ex02-09.htm - Line-height, Letter and Word Spacing

```
<html>
<head><title>Background Image ex02-11.htm</title></head>
<style type="text/css">
.txtSt {background-color:#000000;font-family:arial; color:ffffff;
font-size:30pt; font-weight:bold; text-align:center}
</style>
```

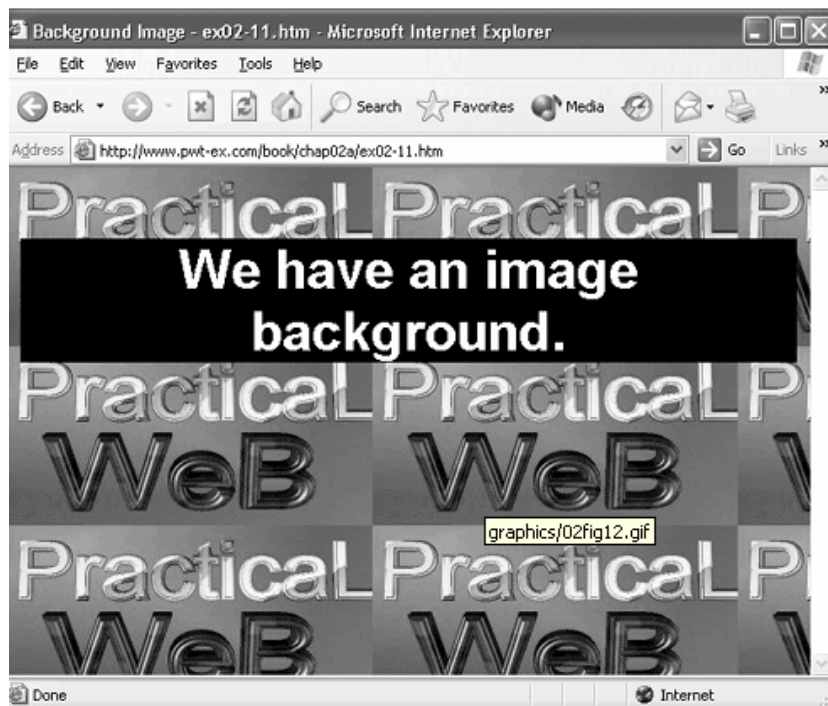
```
<body style="background-image: url(logo_web.jpg);  
background-repeat:repeat">  
<br /><br />  
<div class="txtSt">We have an image background.  
</div>  
</body>  
</html>
```

In this example, the CSS element `background-repeat:repeat` tiles the image `web_logo.jpg` both horizontally and vertically to create the image background. If the `background-repeat:repeat-x` is set, then the image is tiled horizontally only and can be used to create a graphical edge effect for your Web page. The `background-repeat` element is always used in conjunction with the `background-image` element and modifies the way the background image is displayed.

A screen display of this example is shown in [Fig. 2.12](#).

Figure 2.12. ex02-11.htm

Figure 2.12. ex02-11.htm



7.2.4.2 Positioning a background image

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

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

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

"Nailing" a background image

The CSS style element `background-attachment` allows you to control whether the background image moves when the browser window is scrolled. Similar to the `background-repeat` element, the `background-attachment` CSS element only works when the `background-image` element is set. It takes the general format

```
<body style="background-image:url (bg_image.gif) background-attachment: fixed">
```

If the `background-attachment` is set to `fixed`, then the background image will be fixed with respect to the viewing area and therefore not affected by any scrolling action. This has the effect of "nailing" the background image in place and may be a useful function if, for instance, you want to create a watermark feature using your own logo. Example [ex02-13.htm](#) illustrates this action.

Example: [ex02-13.htm](#) - Fixing A Background Image Position

```
<html>
<head><title>Positioning a Background Image ex02-13.htm</title></head>
<style type="text/css">
.txtSt1 {font-family:arial; color:#000000;
font-size:20pt; font-weight:bold}
.txtSt2 {font-family:'Comic Sans MS'; color:#000088;
font-size:20pt; font-weight:bold}
.txtSt3 {font-family:Times New Roman; color:#dd8800;
font-size:20pt; font-weight:bold}
</style>
```

```

<body style="background-image: url(title4.gif); background-
position:center; background-repeat:no-repeat;background-
attachment:fixed">
<div style="font-family:arial;font-size:24pt;color:#8b0000;
font-weight:bold;text-align:center">Fixing A Background Image
<br /><br /></div>
<div class="txtSt1">
The background-image is fixed<br /><br />
</div>
<div class="txtSt2">
and therefore <br /><br />
</div>
<div class="txtSt3">
will not be affected
<br /><br />
</div>
<div class="txtSt1" style="font-size:10pt">
by any scrolling action<br /><br />
</div>
<div class="txtSt2" style="font-size:25pt">
The default scroll attribute makes the background-image
scroll when the user scrolls the page <br /><br />
</div>
</body>
</html>

```

Color, width, and style of element borders

One of the most powerful CSS properties is positioning. This property gives you total, pixel-level control over the location of every element on your Web page. The remainder of this section is devoted to a discussion of the CSS border element and its associated properties.

The CSS border property sets the display of borders around the CSS element that it is associated with. Every border has three aspects: width, style, and color. These properties allow you to have full control as to how you want the borders to be displayed on the Web page.

Some frequently used CSS border elements are shown in [Table 2.6](#).

CSS property	CSS values	NS	IE	Description
border-style	none dashed solid dotted inset outset ridge double groove	4.+	4.+	Sets the style of borders
border-color	#rrggbb color name	4.+	4.+	Sets the color of border sides
border-width	length thin medium thick	4.+	4.+	Specifies the thickness of each border side
border-top	border-top-width border-style color	4.+	4.+	Sets the display values of the top border
border-right	border-right-width border-style color	4.+	4.+	Sets the display values of the right border
border-bottom	border-bottom-width border-style color	4.+	4.+	Sets the display values of the bottom border
border-left	border-left-width border-style color	4.+	4.+	Sets the display values of the left border

There are a total of nine different border styles defined in the CSS1 standard. However, only the support of the solid border style is required for CSS1 compliance. For example,


```
<div style="border-style:double border-color:red">
  Double bordered texts</div>
```

will create a double-line red border around "Double bordered texts."
The nine different border styles are demonstrated in the example [ex02-14.htm](#).

Example: ex02-14.htm - Border With CSS

```
<html>
<head><title>Borders with Styles ex02-14.htm</title></head>
<style>
.bSt {font-family:'Comic Sans MS',times;font-size:12pt}
</style>
<body style="font-family:arial;font-size:20pt;font-weight:bold">
<div style="text-align:center;color:#880000;text-align:center">
  Border Styles and Colors Demo Page</div><br />
<div class="bSt" style="border-style:none;border-color:#000088">
The paragraph has no border style </div><br />
<div class="bSt" style="border-style:double;border-color:#008800">
The paragraph has a DOUBLE border style </div><br />
<div class="bSt" style="border-style:dashed;border-color:#ff0000">
The paragraph has a DASHED border style </div><br />
<div class="bSt" style="border-style:dotted;border-color:#ffd700">
The paragraph has a DOTTED border style </div><br />
<div class="bSt" style="border-style:inset;border-color:#fff4e1">
The paragraph has an INSET border style </div><br />
<div class="bSt" style="border-style:outset;border-color:#ffa500">
The paragraph has an OUTSET border style </div><br />
<div class="bSt" style="border-style:groove;border-color:#006400">
The paragraph has a GROOVE border style </div><br />
<div class="bSt" style="border-style:ridge;border-color:#00ffff">
The paragraph has a RIDGE border style </div><br />
<div class="bSt" style="border-style:solid;border-color:#0000ff">
The paragraph has a SOLID border style </div><br />
</body> </html>
```

In this example, the internal CSS style in lines 68 defines the display text properties. Lines 1540 define all nine different border-style attributes. The none attribute is the default style. Also the support of only the solid style is required for CSS1 compliance. As an example, the lines 3031

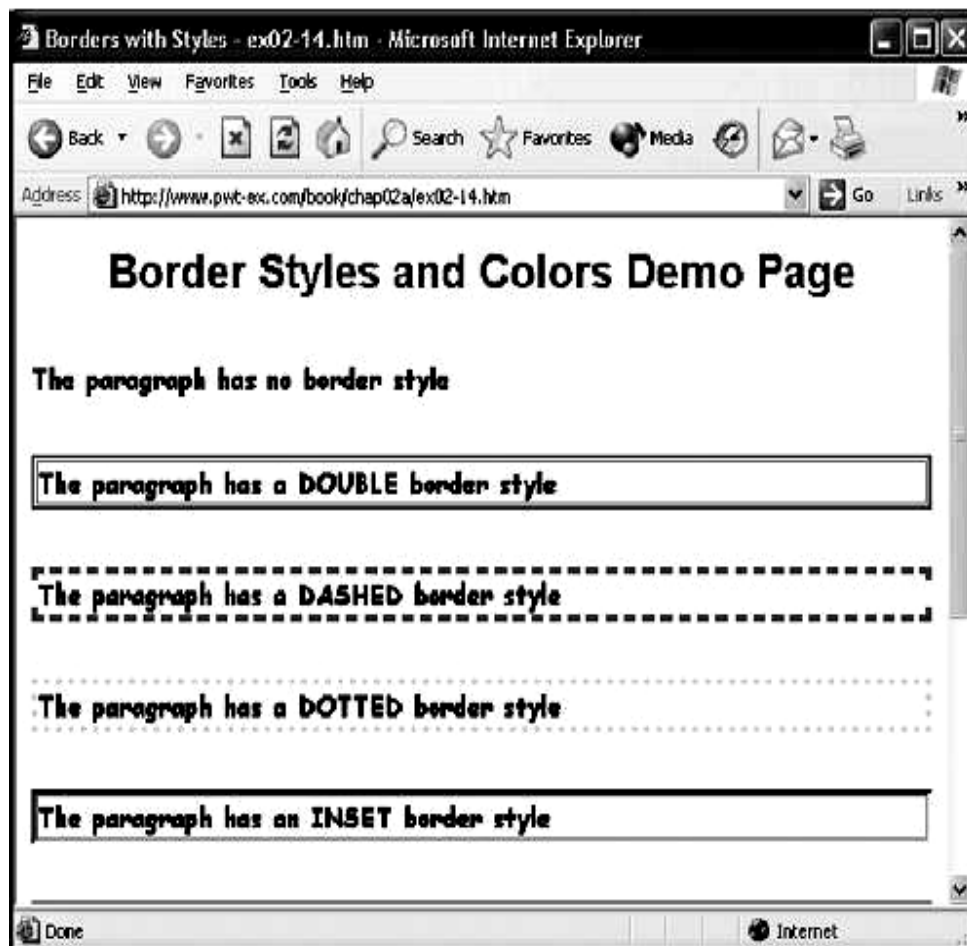
```
<div class="bSt" style="border-style:outset;border-color:#ffa500">
  The paragraph has an OUTSET border style </div><br />
```

set an orange-colored outset border.

The screen shot of example [ex02-14.htm](#) is shown in [Fig. 2.15](#).

Figure 2.15. ex02-14.htm

Figure 2.15. ex02-14.htm



CLASSES AND CSS TAG

Unit Structure

8.1 Classes and Pseudo Classes

8.2 CSS Tag.

8.1 CSS PSEUDO-CLASSES

Syntax

The syntax of pseudo-classes:

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

CSS classes can also be used with pseudo-classes:

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

8.1.1 Anchor Pseudo-classes

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

Example

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

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

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

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

8.1.2 Pseudo-classes and CSS Classes

Pseudo-classes can be combined with CSS classes:

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

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

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

CSS - The :first-child Pseudo-class

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

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

Match the first <p> element

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

Example

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

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

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

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

Example

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

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

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

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

Example

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

CSS - The :lang Pseudo-class

The :lang pseudo-class allows you to define special rules for different languages.

Note: Internet Explorer 8 (and higher) supports the :lang pseudo-class if a <!DOCTYPE> is specified.

In the example below, the :lang class defines the quotation marks for q elements with lang="no":

Example:

```
<html>
<head>
<style type="text/css">
q:lang(no) {quotes: "~" "~";}
</style>
</head>

<body>
<p>Some text <q lang="no">A quote in a
paragraph</q> Some text.</p>
</body>
</html>
```

Pseudo-classes

The "CSS" column indicates in which CSS version the property is defined (CSS1 or CSS2).

Pseudo name	Description	CSS
<u>:active</u>	Adds a style to an element that is activated	1
<u>:first-child</u>	Adds a style to an element that is the first child of another element	2
<u>:focus</u>	Adds a style to an element that has keyboard input focus	2
<u>:hover</u>	Adds a style to an element when you mouse over it	1
<u>:lang</u>	Adds a style to an element with a specific lang attribute	2

:link	Adds a style to an unvisited link	1
:visited	Adds a style to a visited link	1

8.2 CSS ID AND CLASS

The id and class Selectors

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

8.2.1 The id Selector

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

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

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

Example

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

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

8.2.2 The class Selector

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

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

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

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

Example

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

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

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

Example

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

Example:

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



INTRODUCTION TO WEB TECHNOLOGY

Unit Structure

- 9.1 Working of ASP page
- 9.2 Variables
- 9.3 Data types
- 9.4 Operators
- 9.5 Object hierarchies
 - a. ASP Object model

9.1 INTRODUCTION

Active Server Pages are Microsoft's solution to creating dynamic Web pages. With the explosion of the Internet and the World Wide Web into our everyday lives, Web site creation is quickly becoming one of the fastest growing sectors. In the early days of the World Wide Web, Web site design consisted primarily of creating fancy graphics and nice-looking, easy-to-read Web pages. As today's Web sites have become user interactive, the steps in Web site design have changed. Although creating a pleasant-looking Web site is still important, the primary focus has shifted from graphical design to programmatic design. For example, imagine that you wanted to create a Web site from which you could sell widgets. The programmatic design, creating the Web pages that will collect and store user billing information, for example, is more pressing than deciding what background color to use.

Enter Active Server Pages. If you need to build a dynamic Web site—one that can interact with users—Active Server Pages are an easy-to-use solution. Today, you take your first step into the world of Active Server Pages!

➤ **What Are Active Server Pages?**

Over the past couple of years, we have seen some major changes concerning the Internet.

Initially, the Internet served as a medium for members of government and education institutions to communicate. With the advent of the World Wide Web, the Internet became a multimedia, user-friendly environment. Originally, the Internet served as a place for enthusiasts to create personal home pages, but as more people began going "online," the Internet transformed into an informational resource for the common man. When the number of people online reached a critical mass, companies that sold products and services began to spring up. These companies had no physical presence, only a virtual one. For example, you can buy a book from Amazon.com's Internet site, but you won't be able to find an Amazon.com bookstore in your neighborhood. As the Internet has matured into a viable marketplace, Web site design has changed in step.

In the early days of the World Wide Web, HTML was used to create static Web pages. Today, though, static Web pages are quickly becoming obsolete. Imagine if Amazon.com was composed of nothing but static Web pages—you couldn't search its inventory; you couldn't place an order online; you couldn't read other users' comments. It is a safe bet that Amazon.com wouldn't sell many books if it didn't use dynamic Web pages. You can create dynamic Web pages in many ways. Microsoft's solution to building dynamic Web pages is through the use of Active Server Pages, commonly abbreviated ASP.

NOTE

Many large Web sites use Active Server Pages to serve dynamic Web content. For example, Buy.com, HotBot.com, and Dell.com use Active Server Pages to build their interactive, dynamic Web sites. Active Server Pages contain two parts: programmatic code and embedded HTML. The programmatic code can be written in a number of *scripting languages*. A *scripting language* is a particular syntax used to execute commands on a computer. A program composed of commands from a particular scripting language is referred to as a *script*.

Some popular Web-related scripting languages include VBScript and JavaScript. When creating an ASP page, you can use one of four programming languages:

- VBScript—Similar to Visual Basic's syntax, the most commonly used scripting language for Active Server Pages

- JScript—Similar to JavaScript
- PerlScript—Similar to Perl
- Python—A powerful scripting language commonly used for Web development

Most ASP pages are created using VBScript. VBScript has the most English-like syntax of the four scripting languages and is similar to Visual Basic's syntax, which many Web developers have experience with.

NOTE***An ASP page must contain an .ASP extension*****➤ Understanding the Client-Server Model**

Have you ever wondered what, exactly, happens when you type a URL into your browser's Address window? The Internet operates on a *client-server model*. In a *client-server model*, two computers work together to perform a task. A client computer requests some needed information from a server computer. The server returns this information, and the client acts on it. Many everyday activities mimic the client-server model. For example, a map at a large mall performs the role of the server, whereas those strolling through the mall are the clients. If one of these clients wants to know how to reach Sears, he would consult this map, requesting a particular piece of information—namely, "How do I get to Sears from here?" After the client (the mall shopper) has received the information from the server (the map), he leaves, headed in the correct direction. The client-server model typically has many more clients than servers. For example, many mall shoppers are requesting information from just a few maps spread throughout the mall.

The Internet runs on a client-server model as well. With the Internet, the server is a particular *Web server*.

NOTE

A *Web server* is a computer that contains all the Web pages for a particular Web site and has special software installed to send these Web pages to Web browsers that request them. The client, on the Internet, is a Web browser.

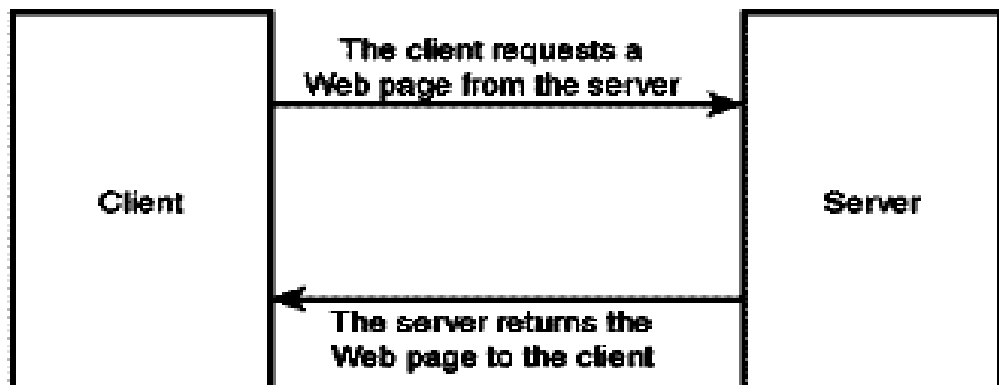
When you visit a static Web page through a Web browser, the following steps occur:

1. The client (the Web browser) locates the Web server specified by the first part of the URL (<http://www.Something.com>).

2. The client then requests the static Web page specified by the second part of the URL (/index.htm).
3. The Web server sends the contents of that particular file to the client in HTML format.
4. The client receives the HTML sent by the server and renders it for you, the user.

Figure 1.1 illustrates this transaction.

Figure 1.1. The Internet is based on a client-server model.



In this transaction, the Web server acts passively, like the mall map in the previous example. The Web server sits around idly, waiting for a client to request a static Web page. After such a page is requested, the Web server sends that page to the client and then returns to idly wait for the next request. With this series of steps, only static Web pages can be sent to the client.

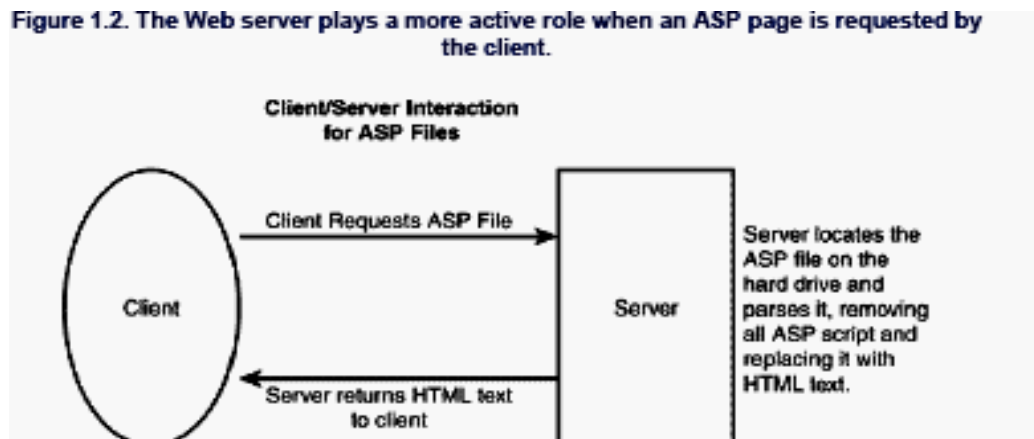
To allow for dynamic Web pages, the Web server must play a more active role. code. This code, which can be written in many different languages, allows ASP pages to be dynamic; however, the Web server has to process this programmatic code *before* sending the HTML to the client. When a Web browser requests an ASP page, the following steps occur:

1. The client (the Web browser) locates the Web server specified by the first part of the URL (http://www.Something.com).
2. The client then requests the ASP page specified by the second part of the URL (/default.asp).
3. The Web server reads the ASP file and *processes* the code.
4. After the ASP page has been completely *processed* by the Web server, the output is sent in HTML format to the client.

5. The client receives the HTML sent by the server and renders it for you, the user.

The client cannot tell the difference between an ASP page and a static Web page because, in both cases, it receives just HTML. When the Web server *processes* an ASP page, all the programmatic code is interpreted on the server—none of it is sent to the client. Figure 1.2 graphically represents this transaction.

Figure 1.2. The Web server plays a more active role when an ASP page is requested by the client.



We've just looked at the two ways a Web server responds to a client's request. If the request is for a static HTML page, the server simply sends back the contents of the Web page. If, however, the request is for an ASP page, the Web server first processes the ASP page and then sends the resulting HTML output to the client. How, though, does the Web server determine whether the client is requesting a static HTML page or an ASP page? The Web server determines this by the extension of the Web page being requested. This is why when you create an ASP page you must give it an .ASP extension. This way, the Web server knows to process the programmatic code *before* sending the output to the client.

Let's briefly look at an example ASP page. Contains code that displays the current date and time. To execute the code in, you first need to install a Web server on your computer. We will discuss how to do this later today in "Running ASP Pages." For now, just examine the code to get a feeling for what an ASP page looks like.

Example 1.1. An ASP Page Displaying the Current Date and Time

```
1: <%@ Language=VBSCRIPT %>
2: <HTML>
3: <BODY>
4: The current time is
5: <% Response.Write Time() %>
6: </BODY>
7: </HTML>
```

Note that the ASP code is surrounded by a `<%` and `%>`. When an ASP page is requested from a Web server, the Web server fully processes all the code between `<%` and `%>` before sending the output to the client. The code in Listing 1.1 probably looks a lot like a regular HTML file. This embedded HTML (lines 2, 3, 6, and 7) makes it easy to create ASP pages from existing HTML documents. In fact, the only ASP code is on lines 1 and 5. Line 1 informs the Web server what scripting language this particular ASP page is using. Recall that an ASP page can use one of four scripting languages. If you wanted to use JScript instead of VBScript in this example, you could change line 1 to the following:

```
<%@ LANGUAGE=JScript %>
```

The second line of ASP code (line 5) displays the current date and time. The `Time()` function is a VBScript function. The `Response.Write` outputs the results of the `Time()` function to the client. This `Response` object and the `Response.Write` method "Using the Response Object." If you have a Microsoft Web server already running on your computer, you can test the code

Create a file named `CurrentTime.asp` and place it in your Web site's root directory. Next, load your favorite browser and visit the ASP page you just created. The URL you want to type in is <http://machineName/CurrentTime.asp> where *machineName* is the name of your computer.

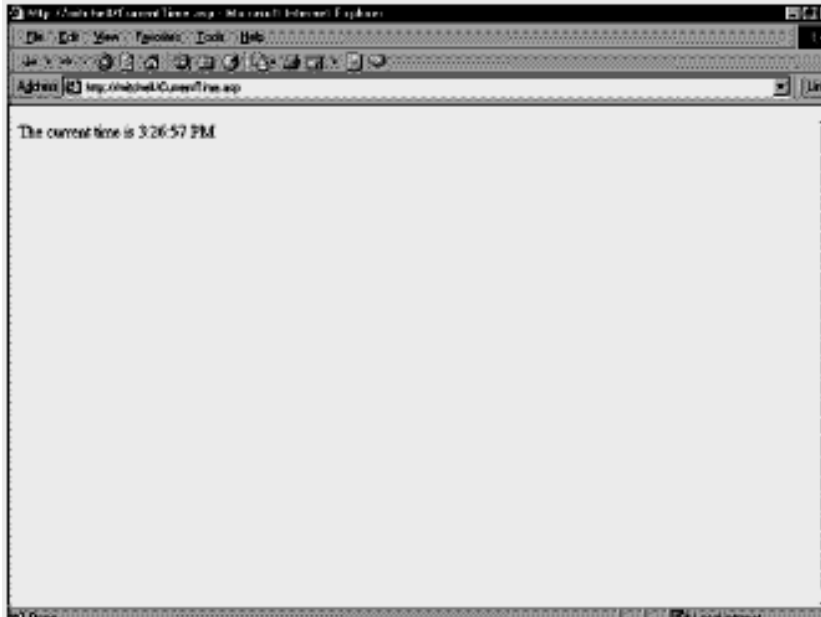
TIP

The following URL will also work:

```
http://localhost/CurrentTime.asp
```

Figure 1.3 displays the output of Listing 1.1 when viewed through a browser.

Figure 1.3. The current date and time is displayed.



Remember that the browser just receives HTML text from the Web server—it does not receive any of the ASP code that was between the `<%` and `%>` delimiters. You can see exactly what the browser received from the client by viewing the HTML source code the browser received. To see this in Internet Explorer, select View, Source from the menu. This opens up Notepad and shows you the source code received. the source code received by the browser when visiting CurrentTime.asp.

Example 1.2. The Browser Receives Only HTML

```
1: <HTML>
2: <BODY>
3: The current time is
4: 3:26:57 PM
5: </BODY>
6: </HTML>
```

➤ How ASP Differs from Client-Side Scripting Technologies

When using ASP, it is vitally important to understand that ASP code exists on the server only. ASP code, which is code surrounded by the `<%` and `%>` delimiters, is processed completely on the server. The client cannot access this ASP code. If you've created Web pages before, you might be familiar with *client-side scripting*. *Client-side scripting* is programmatic code in an HTML file

that runs on the browser. Client-side scripting code is simply HTML code and is denoted by the `<SCRIPT>` HTML tag. Client-side scripting is commonly written using the JavaScript programming language due to the fact that Netscape Navigator only supports the JavaScript scripting language for clientside scripting. Listing 1.3 contains a static HTML page that contains client-side scripting code.

Example 1.3. The Browser Receives Only HTML

```
1: <HTML>
2: <HEAD>
3: <SCRIPT LANGUAGE="JavaScript">
4: <!--
5: alert("Hello world!");
6: // -->
7: </SCRIPT>
8: </HEAD>
9: <BODY>
10: Welcome to my web page!
11: </BODY>
12: </HTML>
```

The code includes raw HTML (lines 1 through 3, and lines 7 through 12) and client-side JavaScript code (lines 4 through 6). It is nothing more than a static HTML file. If the contents of Listing 1.3 were entered into a Web page named `ClientSideScripting.htm`, the entire contents would be sent to the browser when the client requested the Web page. The browser, when rendering the HTML, would display a message box when the `alert` method was reached (line 5). Figure 1.4 shows the output

Figure 1.4. Use client-side scripting to display message boxes on the client's computer.



You can have client-side scripting code in an ASP page because client-side scripting is HTML code, as far as the Web

server is concerned. When developing ASP pages, though, it is important to remember that client-side scripting and ASP code are two different things and cannot interact with one another. ASP scripts are *server-side scripts*. *Server-side scripts* are scripts that execute on the Web server. These scripts are processed and their output is sent to the client.

Table 1.1 outlines the differences between client-side scripting and server-side ASP scripting.

Table 1.1. Differences Between Client-Side Scripting and Server-Side ASP Code	
Method	Differences
Client-side scripting	A client-side script is not processed at all by the Web server, only by the client. It is the client's responsibility to execute any and all client-side scripts.
Server-side scripting	Server-side scripts are processed completely on the Web server. The client does not receive any code from server-side scripts; rather, the client receives just the output of the server-side scripts. Client-side scripts and server-side scripts cannot interact with one another because the client-side scripts are executed on the client after the server-side scripts have finished processing completely.

1. Working With asp Page

➤ Creating Your First ASP Pages

To create ASP pages, you need access to a computer with a Web server that supports Active Server Pages technology. In "Running ASP Pages," we showed how to set up and install two free Microsoft Web servers: Personal Web Server and Internet Information Server. At this point, you should either have an ASP-enabled Web server installed on your computer, or have access to a computer that has such a Web server already installed. After you have a Web server installed, you can create ASP pages in your Web site's root physical directory, or in subdirectories of the root physical directory, and view the result of these ASP pages through a standard Web browser. Because ASP pages are processed completely on the server-side and only return HTML to the client, any Web browser can be used to view ASP pages. There are no restrictions on the client-side. You now have the elements necessary to create and visit ASP pages. Over the next four days, you will learn the ins and outs of the VBScript scripting language, the most commonly used scripting language for ASP pages.

Although at this point you may not be familiar with VBScript, let's look at an example ASP page. This will help you become familiar with the notation and VBScript syntax. Furthermore, it will show you some neat things you can do with Active Server Pages. Imagine that, depending on the time of the day, you want a Web page to display a different message. For example, if the time is 11:00 AM, you want to display Good Morning!, whereas if the time is 5:00 PM, you want to display Good Evening! Using static HTML pages, you would have to edit the HTML page twice a day—once before noon and once after, altering the Web page and changing its message. With ASP pages, however, you can use programmatic code to determine the current time and display a custom message based on the time. Example 1.5 contains the code for an ASP page that displays a custom message based on the current time.

Example 1.5. Displaying a Different Message Depending on the Time of Day

```
1: <%@ Language=VBScript %>
2: <% Option Explicit %>
3:
4: <HTML>
5: <BODY>
6: The current time is <%=Time()%>
7: <P>
8: <%
9: If DatePart("h",Time()) >= 12 then
10: 'Is is after noon
11: Response.Write "Good Evening!"
12: Else
13: 'Is is before noon
14: Response.Write "Good Morning!"
15: End If
16: %>
17:
18: </BODY>
19: </HTML>
```

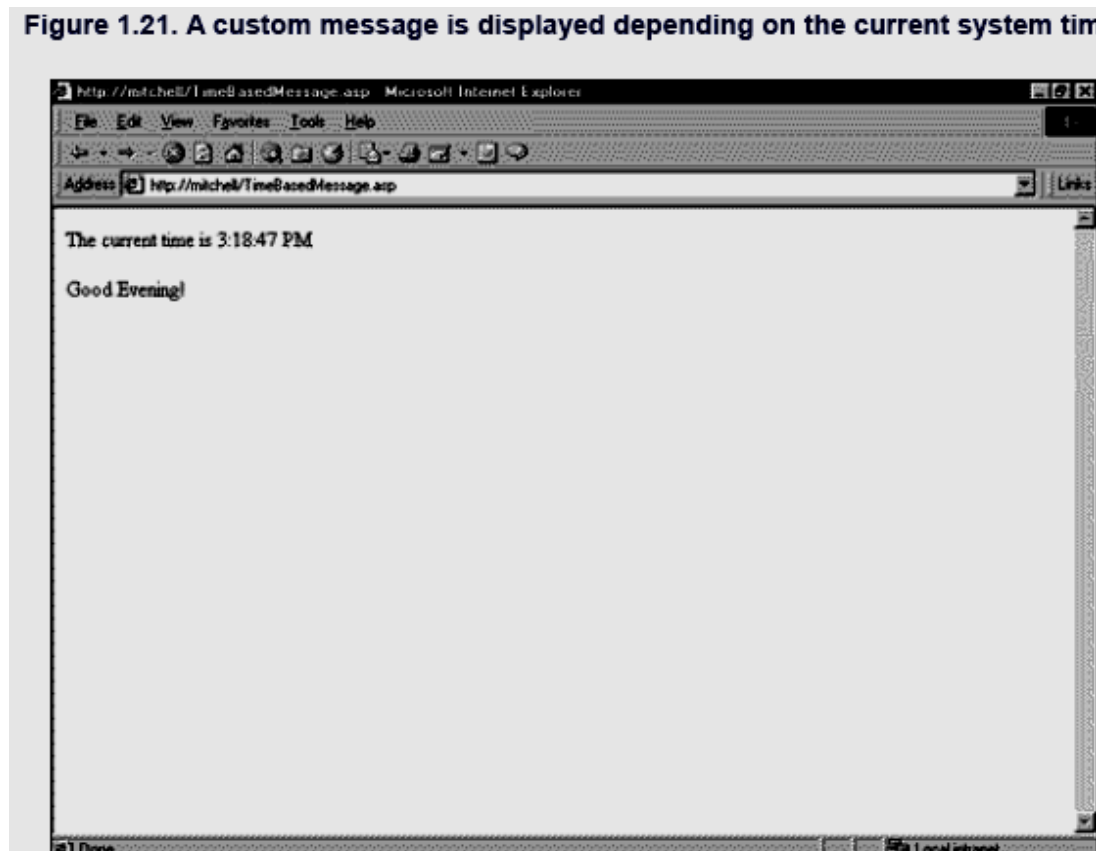
To view the output, create an ASP page named TimeBasedMessage.asp and save this file in your Web's root physical directory. Enter the code in Listing 1.5, save the file, and then view it through your browser of choice using the following URL:

<http://localhost/TimeBasedMessage.asp>

Let's look over the code in Listing 1.5. Line 1 begins with the `@LANGUAGE` directive, which informs the Web server what scripting language the current ASP page is using. Option Explicit, is another line of code that should *always* be used in *every* ASP page you create. When Option Explicit is used, all variables must be explicitly declared. We'll discuss Option Explicit in greater

Line 6 displays the current system time using the `Time()` function. The notation for displaying the results of the function, `<%=...%>`. For the time being, realize that `<%=...%>` shares the same functionality as `Response.Write`, which outputs information to the client. Lines 8 through 16 are an ASP code block, denoted by the `<%` and `%>` delimiters. An If statement is used on line 9 to determine whether the current time is after or at noon or before noon. The `DatePart`, which is used here to get just the hour portion of the current time. Figure 1.21 shows the output of Listing 1.5 when viewed through a browser.

Figure 1.21. A custom message is displayed depending on the current system time



9.2 VARIABLE

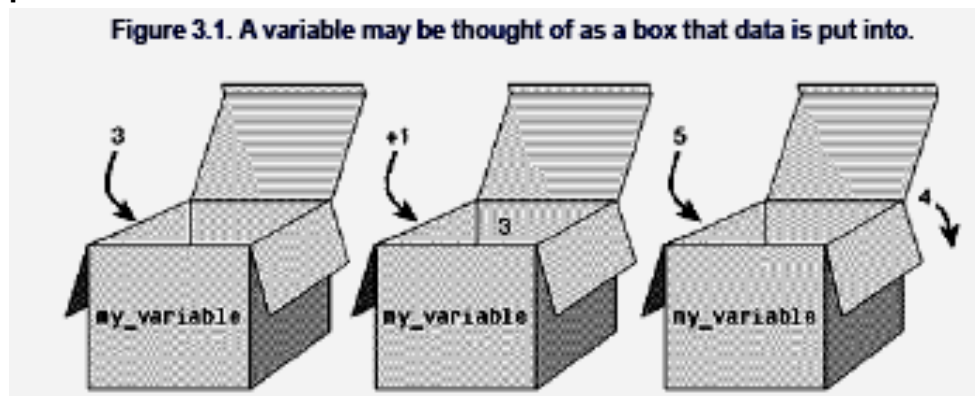
Now that you know what ASP is and what an ASP page looks like, it is time to get into some programming topics. Today, you will learn about variables and operators. These are fundamental to moving and manipulating data in ASP and VBScript. Without these, your pages would have no way of remembering the information they are told and would be unable to perform basic computations using that information. Today, you will learn the following:

- What a variable is
- What the different types of variables are
- Good programming techniques with variables
- What operators are found in VBScript and how to use them
- What operator precedence is and how it affects programmers

What Is a Variable?

If you have never programmed before, you may wonder what the term *variable* means. A variable is a small section of a computer's memory that you give a name to. Think of a variable as a box into which you can put numbers, letters, dates, and more. This information can now be carried around and manipulated by referring to the name you gave it. For example, as illustrated in Figure 3.1, you might put the number 3 into a variable. Then you might add 1 to it. Or you could decide that you do not want that number after all, and so you replace it with 5. All these things, and more, are possible with variables.

Figure 3.1. A variable may be thought of as a box that data is put into.



9.3 DATA TYPES

There are many different types of data that you might want to be able to store into a variable: numbers, words, dates, and many more. In this section, you will look at a few of the most important data types that can be stored in variables and discuss the way in which VBScript does this.

- **Integer**

- An integer is a whole number—that is, a number with no fractional portion.
- For example, 1, 3, 9, and -4 are all integers, but 1.2, 0.9, and -5.5 are not.
- Two other data types are related to the integer: byte and long.
- A long can store a larger range of numbers than an integer.
- A byte stores fewer.
- You do not have to worry too much about which range your numeric values fall into.
- VBScript handles the issue for you.

- **Floating-point Numbers**

- Floating-point numbers may have a decimal. 1.5, -3.4, 4.1, and even 5.0 are all floating-point numbers.
- It is important to note, however, that although integers are stored exactly, that is not necessarily true of floating-point numbers.
- Floating-point numbers are often rounded or truncated to fit into the space allotted for them.
- Single and Double** data types are associated with floating-point numbers.
- The difference between the two has to do with the precision used to store the number.
- Doubles require twice as much memory as singles, but can obviously hold a much greater range of numbers and to greater precision than can singles.
- Again, in VBScript, you do not have to worry about the distinction too much.

- **String**

- A string can hold any sequence of letters, numbers, and symbols.

- ii. Strings are distinguished from code, variable names, and numbers by putting them between double quotation marks. "My name is Fred.", "20mph", and "14" are all possible strings.
- iii. Even the empty string "" can be treated like a string in most cases.
- iv. For example: When we would use a statement like Response.Write "Hello", the "Hello" is a string value.
- v. It is not stored in a variable, but it is the same kind of data. String values will be used often in sending output like this.

Do Don't

DO remember when using digits in strings that there is a difference between a string of digits and a number. The string "14" is handled differently than the integer value 14.

DON'T confuse a string variable's name with the value it contains. A string variable called black might contain a value of "white".

- **Date**

- i. A nice feature of VBScript that is missing in other programming languages is its date handling.
- ii. Although it is possible to represent the date using strings and/or integers, this variable type simplifies things.
- iii. A date variable can hold either a date or a time, and VBScript's various date functions and operators make the formatting and printing of date-related information easy.

- **Boolean**

- i. A Boolean variable may hold a value of either True or False.
- ii. Boolean variables are generally used when a decision needs to be made.
- iii. The value of the variable can determine which of two actions should be taken.

- **Currency**

- i. A single precision number would work fine for storing monetary values, but VBScript provides a special data type for money that works with several special VBScript functions and displays nicer.

- **Object**

- i. This refers to special objects. It is used a lot in performing database operations.

Q. What Are Variant Variables?

- In most programming languages, a distinction must be made between variables of different types.
- A variable used to contain a string cannot later be used to contain an integer.
- This is not true in VBScript.
- VBScript uses *variant variables*, which are variables that may contain values of any type.

Q. What Does It Mean to Declare a Variable?

- Many programming languages require that, before you use a variable, you tell the system what type of data you intend to put into the variable and what you want it to be called. For example, in the C++ programming language, you might say
- `int my_variable;`
- `my_variable = 2;`
- The first line tells the system that you want to use a variable that you will call `my_variable`, and that you want to be able to put integer data into it
- This is an example of an explicit declaration.
- You explicitly tell the system what variable you want to create.
- The second line begins to use that variable by putting the value 2 into it.
- If you are familiar with C or C++, this should look familiar to you.
- If you are not, you need not be concerned with it. VBScript makes things a little easier.
- In VBScript, it is not necessary to specify integer, real, char, or whatever when you create a variable.
- In VBScript, all variables are declared using the keyword `dim`.
- This is because VBScript uses the aforementioned variant variables.
- Therefore, the VBScript equivalent of the preceding statements would be as follows:
- `dim my_variable`
- `my_variable = 2`
- Here the first line declares `my_variable` without specifying that `my_variable` will represent an integer.
- Further, it is not even necessary to include the first line at all.
- In VBScript, it would be acceptable to simply use the second line with no prior mention of `my_variable` whatsoever.

- This is called an implicit declaration. That is, the system figures out on its own that you want to create a variable named `my_variable`.

Q. Why Use Explicit Declarations in VBScript?

- Now that you can see that explicit declarations are not necessary in VBScript, you may well be wondering why anyone would want them.
- Try putting Listing 3.1 into your editor of choice. Name the file `Typo1.asp`.

Example 3.1. Mistyping a Variable Without Using Option Explicit

```

1: <%@ Language=VBScript %>
2: <% myfirstvariable = 2 %>
3: <HTML>
4: <BODY>
5: The variable named "myfirstvariable" has a value of
6: <%
7: Response.Write(myfirtvariable)
8: %>
9: </BODY>
10: </HTML>

```

- Notice that in line 7, the variable name has been misspelled.
- This is deliberate. Now try viewing this page. You get no error message, but the result is not what the programmer had intended.
- In this simple example, you could probably find the problem with no great difficulty. Imagine, though, a longer page with 200 lines.
- Finding the typo might be a bit more of a challenge!
- That is why the following line is often included in active server pages, especially ones of any great length:

```
<% Option Explicit %>
```

- Adding this immediately following the `<%@ Language=...%>` line will cause VBScript to require explicit declarations of all variables.
- So let's try adding this line into the previous example.
- Name the new file `Typo2.asp`. It should now look like Listing 3.2.

Example 3.2. Declaring Variables with Option Explicit

```

<%@ Language=VBScript %>
<% Option Explicit %>
<% myfirstvariable = 2 %>
<HTML>
<BODY>
The variable named "myfirstvariable" has a value of
<%
Response.Write(myfirtvariable)
%>
</BODY>
</HTML>

```

- You get an error message when you try to view this in your Web browser.
- It says something like, "Variable is undefined: 'myfirstvariable'."
- Do you know what is causing this error?
- It's the work of that Option Explicit.
- You did not declare the variable "myfirstvariable" explicitly.
- So, let's do that, and call this Typo3.asp. This version can be found in Listing 3.3.

Example 3.3. Finding the Typo with Option Explicit

```

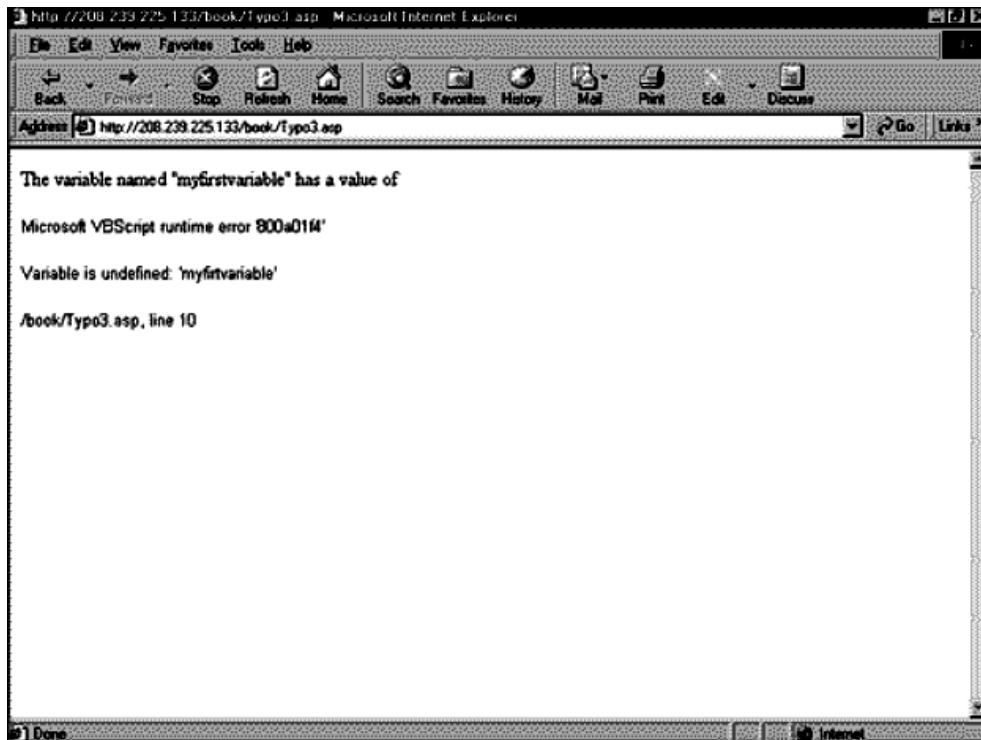
<%@ Language=VBScript %>
<% Option Explicit %>
<% Dim myfirstvariable
myfirstvariable = 2
%>
<HTML>
<BODY>
The variable named "myfirstvariable" has a value of
<%
Response.Write(myfirtvariable)
%>
</BODY>
</HTML>

```

- So that is fixed now.
- Try viewing it again. You will still get an error message.
- Now it says "Variable is undefined: 'myfirtvariable'."
- This makes fixing the problem easy because you know you didn't mean to have any variables named 'myfirtvariable'.

- Also notice that below that it tells you what file the error is in and what line number it is on. You can see the full error message in Figure 3.2.
- This is much better than when you had a page with incorrect results, but no error message. Now all you have to do is go down to line 10 and fix your typo!

Figure 3.2. Because we used Option Explicit, we see our error.



3. Constants

- A constant is a little like a variable in that you give it a name and store data in it.
- Unlike variables, however, constants are assigned a value when they are declared, and that value cannot be changed.
- VBScript has several constants built into it. For example, the constant `vbInteger` is declared to be equivalent to the number 2.
- In your code, you could either refer to the number directly or using the constant name.
- Typing the number directly may involve fewer keystrokes, but using the constant name makes your code a little easier to read.
- It also helps save you from having to memorize many numbers

- For example, if you were creating a page to sell merchandise, you might find you need to use the sales tax rate several times within the page.
- If you did not make a constant for the tax rate, you would need to refer to a strange decimal number like 0.0625 a lot.
- Plus, if the tax rate ever changed, you would have to update several lines of code.
- Put a simple declaration like this at the top of your page, though, and you can refer to TAXRATE throughout the rest of the page and make updates easily:
 - Const TAXRATE = 0.0625
- It is suggested that you use all capital letters to refer to constants to help distinguish them from

9.4 VBSCRIPT OPERATORS

Operators allow us to work with data, combining, changing, or replacing it. There are five major classes of operators we will deal with. The first is assignment, which you have already seen a little of.

i. Assignment Operator

- So far today we have discussed variables and mentioned that data may be stored in them without explaining how to do so.
- The most common way to accomplish this is through the assignment operator.
- The assignment operator, in VBScript, is the equals sign (=).
- The assignment operator takes whatever is on the right-hand side of it and stores it in the variable on the left-hand side of it.

For example, consider the following code:

```
<% Dim iVar  
iVar = 3  
iVar = 5  
%>
```

If you have not done any programming before, you may be wondering how iVar could equal 3 and 5 at the same time. It does not. You must be careful not to confuse the assignment operator with mathematical equality. The third statement does not say iVar equals 5. Rather, it says 5 is stored in iVar. For a little review, we shall step through this code line-by-line. The first line is the variable

declaration, as discussed previously. At this point, a value of Empty is stored in iVar. The next line assigns a value of 3 to iVar. So in the box labeled iVar, there is now a 3. Finally, a value of 5 is assigned to iVar. The 3 that was in there before is gone now. Be careful with this when you are programming. Do not overwrite variables with new values until you are sure that you are finished with the previous value.

If it is absolutely necessary to overwrite a value you plan to use later, you might create a second variable to hold onto it:

```
<% Dim iVar, iOldVar
iVar = 3
iOldVar = iVar
iVar = 5
%>
```

Here, the first line declares two variables now: iVar and iOldVar. Both begin with an empty value.

The next line assigns a value of 3 into iVar. At this point, iOldVar still has an empty value. The third line might be a bit confusing if you are new to programming. Assignment may be done not just with explicit values such as 3 or 5 on the right, but also with variables. Following this statement, both iOldVar and iVar will have values of 3 stored in them. Finally, iVar is assigned a value of 5. This does not affect iOldVar. Line 3 copied what was in iVar into iOldVar, but it did not establish any kind of permanent connection between iVar and iOldVar. This way, iVar can be used with its new value, but if you still need to use the old value of iVar, it is available. Now look at one last version of this code before going on to a new topic:

```
<% Dim iVar
iVar = 3
5 = iVar
%>
```

Do you think this code will work? Reread the first paragraph on the assignment operator if you are not sure. The answer is No. This code is not valid. The first two lines are carried out as expected, but the third is meaningless. The assignment operator copies what is on the right into the variable on the left. In this case, the number 5—not a variable—is on the left. 5 cannot be the name of a variable since all variable names must begin with letters

I have been using integers in these examples, but I needn't have. The assignment operator works with singles, doubles, strings, Booleans, and so on, just as well.

```
strName = "John Smith"  
bol_The_Assignment_Operator_Is_Powerful = True  
dtJills_Birthday = #03/06/1946#
```

NOTE

Surrounding the date with #s keeps it from being evaluated as three divided by six divided by 1946. You might have noticed that Listing 3.5 used the assignment operator. Listing 3.6 is a modified version of that code to show off the assignment operator a bit more. The file is called AssignmentDemo.asp.

Example 3.6. Demonstration of the Assignment Operator

```
1: <%@ Language=VBScript %>  
2: <% Option Explicit  
3: Dim strName, iAge  
4: %>  
5: <HTML>  
6: <BODY>  
7: <%  
8: Response.Write("Before assigning a value, strName has value  
9: ")  
9: Response.Write(strName)  
10: %>  
11: <BR>  
12: <%  
13: strName = "James"  
14: iAge = 21  
15: Response.Write("Now strName has value ")  
16: Response.Write(strName)  
17: %>  
18: <BR>  
19: <%  
20: Response.Write("Now iAge has value ")  
21: Response.Write(iAge)  
22: %>  
23: </BODY>  
24: </HTML>
```

Line 3 declares the two variables we will use: `strName` and `iAge`. Lines 8 and 9 write a message that demonstrates the value that `strName` has before we use the assignment operator to set it. Lines 13 and 14 set the values of the two variables. Lines 15 and 16 show the new value given to `strName`. Looking at the output, you can now verify that the assignment was successful. Lines 20 and 21 similarly display the new value given to `iAge`.

ii. Mathematical Operators

Now that you can put values into variables, it is time to start using those values. We begin with the operations VBScript can perform that are classified as mathematical operators. They include addition, subtraction, negation, multiplication, division, and exponentiation, all of which you have probably seen before, as well as integer division, modulus, and string concatenation, which may be new.

Addition

Addition takes the form *argument + argument* where each argument may be a number, a numerical variable, or another numerical expression:

```
<% Dim iSum
iSum = 3 + 5
%>
```

This is one of the simplest cases, where both arguments are numbers. When the code is run, the variable `iSum` ends with a value of 8. The next example demonstrates how a variable may be used as an argument:

```
<% Dim sngSum, sngLeft
sngLeft = 3.2
sngSum = sngLeft + 1.1
%>
```

In this case the result, as you may have guessed, is 4.3. The value in `sngLeft` is added to 1.1 and stored into `sngSum`. `sngLeft` is unaffected by the addition. Let's take a look at another tricky example.

```
<% Dim iCount
iCount = 2
iCount = iCount + 1
%>
```

This case may seem strange to you at first. How can `iCount` be equal to `iCount` plus 1? First, remember that the equals sign

refers to the assignment operator. Let's step through this line-by-line. The first line, as you have seen before, is the declaration. The variable `iCount` is created, with an empty value. The second line assigns a value of 2 to `iCount`. In the third line, the 2 is retrieved from `iCount`. Then 1 is added to that value, giving 3. 3 is then sent to the assignment operator, which stores it in `iCount`. So, at the end of this code, `iCount` holds a value of 3. The 2 formerly in `iCount` has been overwritten. Notice that the 1 is added to `iCount` before the assignment is carried out. Everything on the right side of an assignment operator is executed before the assignment. Table 3.4 lists the order in which operations are carried out.

Table 3.4. Operator Precedence

Table 3.4. Operator Precedence	
<i>Precedence</i>	<i>Operators</i>
Highest (done first)	Anything in parentheses
	Exponentiation (^)
	Negation (-)
	Multiplication, Division (*, /)
	Integer Division (\)
	Modulus (Mod)
	Addition, Subtraction (+, -)
	String Concatenation (&)
	Equality (=)
	Inequality (<>)
	Less than (<)
	Greater than (>)
	Less than or equal to (<=)
	Greater than or equal to (>=)
	Not
	And
	Or
	Xor
	Eqv
Lowest (done last)	Imp

Subtraction

Now that you know how to add, you may well wonder how to subtract. Subtraction works much like addition, taking the form *argument - argument*. Any combinations you might have used with addition will again work for subtraction. One difference, however, is that with subtraction the order of arguments is important. For example, $3 - 5$ is certainly much different from $5 - 3$. In the following

code, can you tell what value iCount will have after the code is finished?

```
<% Dim iCount  
iCount = 3  
iCount = iCount + 1  
iCount = iCount - 2  
%>
```

The answer you should come up with is 2. iCount begins as 3, adding 1 makes it 4, and subtracting 2 makes it 2.

Multiplication

Multiplication should be easy for you. The symbol for multiplication is the asterisk (*). Multiplication follows the same form as the other operations we have covered so far. Multiplication does introduce a new complication, though. Think about the expression $3 * 5 + 2$. What should this expression evaluate to? If you perform the addition first, the result is 21. If you perform the multiplication first, the result is 17. Which is correct? You should remember from math class that multiplication is performed before addition, making the correct answer 17. VBScript knows this, too. This understanding that multiplication comes before addition is an example of *precedence*. Precedence is a set of rules for the order in which operations should be performed. Multiplication is said to have higher precedence than addition. Table 3.4 offers a complete listing of operator precedence in VBScript.

If you wanted to evaluate the addition first, there is a way. Perhaps you remember from math class that whatever is in parentheses is carried out first. VBScript works the same way. Putting parentheses into the previous expression gives you $3 * (5 + 2)$, which evaluates to 21. Parentheses can be used with any operation to force it to be evaluated in a certain order.

Division

VBScript has two different kinds of division. The first is the kind you are probably most familiar with. Standard division is represented with the slash (/). It takes two numerical values and returns their floating-point quotient. For example, $5 / 2$ returns 2.5, and $4 / 5$ returns 0.8. Some divisions will result in decimals that do not terminate. In these cases, the best approximation the system can store is used. For example, in the case of $1 / 3$, the computer cannot store an infinitely repeating decimal like 0.33333.... Also, be

careful to avoid division by zero. Dividing by zero, or a number so close to zero that the computer thinks it is zero, results in an error.

Integer Division

Chances are, before you knew what decimals were, you learned division like this: 5 divided by 3 is 1 with a remainder of 2. Together, the integer division and modulus operators allow you to do this kind of division in VBScript. The integer division operator, represented with the backslash (`\`), returns the quotient. So, for example

```
5 \ 3 returns 1
4 \ 2 returns 2
0 \ 8 returns 0
1 \ 2 returns 0
```

Unlike most other programming languages, integer division is even defined when the terms are floating-point numbers. When a term in an integer division operator is a floating-point number, it is rounded to the nearest integer and then integer division is applied. For example `4 \ 2.2` returns 2 `8.3 \ 2.6` returns 2

Modulus

Going along with integer division is the modulus operator. Whereas integer division returns the quotient when the two numbers are divided, modulus returns instead the remainder. For example

```
5 Mod 3 returns 2
4 Mod 2 returns 0
0 Mod 8 returns 0
1 Mod 2 returns 1
4 Mod 2.2 returns 0
8.3 Mod 2.6 returns 2
```

The usefulness of these last two operators may not be apparent right now, but, in truth, they are very powerful. Notice that the Mod operator repeats.

```
0 Mod 3 returns 0
1 Mod 3 returns 1
2 Mod 3 returns 2
3 Mod 3 returns 0
4 Mod 3 returns 1
5 Mod 3 returns 2
```

and so forth. This can make the Mod operator useful when you need something to behave in a cyclical manner. Mod can also help you check whether one number divides evenly into another. If a Mod b returns 0, b divides evenly into a.

Exponentiation

In VBScript, the exponentiation operator is represented by the carat symbol (^). If you do not recall much about exponentiation, $a^b = a * a * a * \dots * a$ (b times). For example

$$3^3 = 3 * 3 * 3 = 27$$

$$5^2 = 5 * 5 = 25$$

$$6^3 = 6 * 6 * 6 = 216$$

Also, note that exponentiation is executed from left to right. This means that if you have an expression such as 2^3^2 , the 2^3 is carried out first and then raised to the second power.

$$2^3^2 = 8^2 = 64$$

Negation

Negation is the operation that converts a positive number to a negative number and vice versa. It is equivalent to multiplying by -1. Negation is denoted using the same symbol, the dash (-), as subtraction. The difference between negation and subtraction is that subtraction—like addition, multiplication, and the other operations discussed so far—is a *binary operation*. A binary operation is one that takes two arguments. Negation, on the other hand, is a *unary operation*. It takes only one argument. So the dash means subtraction when it is between two numerical values and negation when it is just in front of one. Listing 3.7 puts together a few of the arithmetic operations covered so far. Take a good look at it and make sure that you can follow what is happening.

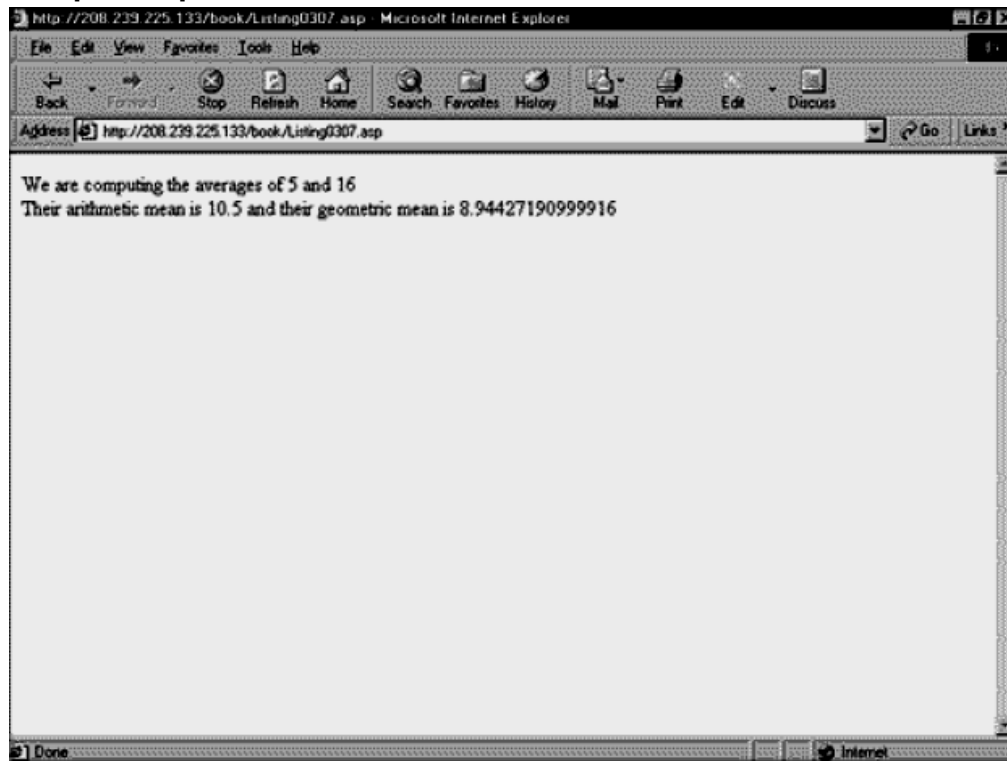
Example 3.7. Putting the Arithmetic Operators Together

```
1: <%@ Language=VBScript %>
2: <% Option Explicit
3: Dim iTerm1, iTerm2, sngArithmetic, sngGeometric
4: %>
5: <HTML>
6: <BODY>
7: <%
8: iTerm1 = 5
9: iTerm2 = 16
```

```
10: Response.Write("We are computing the averages of ")
11: Response.Write(iTerm1)
12: Response.Write(" and ")
13: Response.Write(iTerm2)
14: %>
15: <BR>
16: <%
17: sngArithmetic = iTerm1 + iTerm2
18: sngArithmetic = sngArithmetic / 2
19: sngGeometric = (iTerm1 * iTerm2)^0.5
20: Response.Write("Their arithmetic mean is ")
21: Response.Write(sngArithmetic)
22: Response.Write(" and their geometric mean is ")
23: Response.Write(sngGeometric)
24: %>
25: </BODY>
26: </HTML>
```

Line 3 declares the variables this script will use. Line 8 initializes one of the variables with the number 5. Line 9 initializes the other with 16. Lines 10 through 13 print a message that tells the user what numbers are being used for the calculations. Lines 17 and 18 proceed to calculate the arithmetic mean of the two numbers. The arithmetic mean is simply the standard average you learned in school. Line 17 takes the two numbers, adds them together, and stores the result in `sngArithmetic`. Then line 18 divides `sngArithmetic` by 2 and stores the result in `sngArithmetic`. Notice that the slash is used, indicating that we are performing floating-point division rather than integer division. `sngArithmetic` now holds the average of the two numbers. Line 19 computes the geometric mean. If you have not seen the geometric mean of two numbers before, it is simply the square root of their products. To calculate it, we first multiply the two numbers. Then, the result is raised to the power 0.5. Raising a number to the power 0.5 is the same as finding its square root. Notice the parentheses that are used on line 19. They are necessary. Without them, the exponentiation would be performed first, and then the multiplication. Lines 20 through 23 print out some closing messages. The values of `sngArithmetic` and `sngGeometric` are displayed. You can see the output of this listing in Figure 3.5.

Figure 3.5. Operators may be put together to form more complex expressions.



Concatenation

The arithmetic operations we have discussed have been operations on numbers. Concatenation, though, is an operation between two strings. The two strings are joined together, becoming one string. Concatenation may be represented by either the plus sign (+) or the ampersand (&), but the ampersand is preferred to avoid confusion with addition. Look at a few examples:

"Hello" & "World" becomes "HelloWorld"

"Hello " & "World" becomes "Hello World"

"My name is " & "John Smith" becomes "My name is John Smith"

Like the numerical operations, concatenation may be used several times in one statement, as in the following:

"Welcome," & " John Smith, " & "to the wonderful world of strings"

Becomes "Welcome, John Smith, to the wonderful world of strings"

Listing 3.8 demonstrates how string concatenation can make life a little easier. Instead of constantly using Response.Write as was done in Listing 3.7, you can collect data, put it together with the concatenation operator, and write it out together.

Example 3.8. Simplifying Things with String Concatenation

```

1: <%@ Language=VBScript %>
2: <% Option Explicit
3: Dim iTerm1, iTerm2, sngArithmetic, sngGeometric, strOut
4: %>
5: <HTML>
6: <BODY>
7: <%
8: iTerm1 = 5
9: iTerm2 = 16
10: strOut = "We are computing the averages of "& iTerm1 &_
11: " and " & iTerm2 & "<BR>"
12: Response.Write(strOut)
13: sngArithmetic = iTerm1 + iTerm2
14: sngArithmetic = sngArithmetic / 2
15: sngGeometric = (iTerm1 * iTerm2)^0.5
16: strOut = "Their arithmetic mean is " & sngArithmetic &_
17: " and their geometric mean is " & sngGeometric
18: Response.Write(strOut)
19: %>
20: </BODY>
21: </HTML>

```

This listing does the same thing as Listing 3.7. The only difference is that we were able to cut down on calls to `Response.Write` by using the string concatenation operator. Lines 10 and 11 take all the output that was displayed on lines 10 through 13 of Listing 3.7, and concatenate them together. Then, line 12 needs only one `Response.Write` to send the entire message to the output at once. Lines 13 through 15 perform the same computations as before. Now, lines 16 and 17 concatenate the output strings. Once again, `strOut` is used to hold the result. Then, line 18 simply needs to write `strOut` to output.

Comparison Operators

The comparison operators make comparisons between two arguments and evaluate to either `True` or `False`. The VBScript comparison operators are equality (`=`), inequality (`<>`), less than (`<`), greater than (`>`), less than or equal to (`<=`), and greater than or equal to (`>=`). You probably worked with all these in math class, but in case you are a bit unsure about them, Table 3.5 provides a little review.

Table 3.5. Comparison Operators

Table 3.5. Comparison Operators		
Operator	True When...	False When...
A = B	A and B are the same	A and B are different
A >B	A larger than B	A same as or smaller than B
A <B	A smaller than B	A same as or bigger than B
A >= B	A bigger than or same as B	B bigger than A
A <= B	A smaller than or same as B	A bigger than B
A <>B	A and B different	A and B the same
A few examples		
3 >4	False	
5 >4	True	
4 >= 3	True	
4 >= 4	True	
3 <3	False	
2 <>9	True	

Comparison operators can also be used to compare strings. In this case, it compares them alphabetically. The string that comes first alphabetically is treated like it is less than the one that comes later. It treats uppercase letters like they come before lowercase letters. So "Alligator" comes before "aardvark," which comes before "alligator." This means that "Alligator" < "aardvark" would return true, and "alligator" < "aardvark" would return false.

Logical Operators

The last set of operators allows you to join together and manipulate Boolean expressions such as those in the "Comparison Operators" section. They are And, Or, Not, the exclusive or (XOR), equivalence (EQV), and implication (IMP). If you have had some logic courses, you should find using these operators comes naturally. All these, except NOT, take two Boolean values and return a Boolean value. AND returns true when both of its arguments are true. Table 3.6 describes when each logical operator evaluates to True and to False.

Table 3.6. Logical Operators

Table 3.6. Logical Operators		
<i>Operator</i>	<i>True When...</i>	<i>False When...</i>
A AND B	A and B both true	Either A or B is false
A OR B	One of A or B is true	Both A and B are false
NOT A	A is false	A is true
A XOR B	A or B true, but not both	Both true or both false
A EQV B	Both false or both true	One is false; other is true
A IMP B	A is false or B is true	A is true and B is false
Logical operators are usually put together with comparison operators in the same line. A few examples		
(3 <4) AND (4 <5)		evaluates to True
(4 <>4) OR (6 <7)		evaluates to True
(4 <2) AND bolExpr		evaluates to False

Notice in that last case, we do not know based on the information shown what the value of `bolExpr` is. However, because it is an AND statement and we know the first part is false, the whole statement is false regardless.

9.5 . WORKING WITH OBJECTS

A popular buzzword in programming is "object-oriented programming." Today, you will get a high-level overview of objects, including what they are and how they can help you in your programming. Today, you will learn the following:

- What objects are
- Components of objects
- Actions that can be performed on objects
- Available ASP built-in objects
- What a collection is

What Are Objects?

Think about your car. You know that when you want to start your car, you put the key into the ignition and turn it. You probably have not thought too much about everything that happens when you turn that key. You do, however, know the result you expect. Your car executes a sequence of steps to get ready for your next instruction. Imagine if you had to specify each step yourself. It would be difficult to keep it straight. You might occasionally forget a step and cause serious damage. Further, if you ever wanted to drive a different car, you would have to learn a whole new set of instructions. It is much easier to simply remember to put the key in the ignition and turn it. This is an instruction you can remember, and

you can apply to any car. This is the object-oriented way of thinking about your car. You think about a couple of general things: the things that describe your car and the things you can tell it to do. An object is a reusable piece of software that contains related data and functions that represent some real-life thing. Why would objects be useful in the pages you write? Objects help increase the level of abstraction in our pages. Say that you want to display a randomly chosen banner. You could read in the list of banners to choose from, run the random number generator, and write the code for the `` tag. You could do all these things every time you wanted to display a banner. But wouldn't it be easier to write all that into an object that represents a random banner? From then on, you would only need to write something like `RandomBanner.Display` to display a banner. Then, if you wanted to change or add to your banner displaying system, you would only need to change one piece of code instead of many. And wouldn't it be even better to find out that someone else had already written an object that would do it all for you? Using functions helps to improve the simplicity and readability of your code, and objects take that to the next level. Pages can then be built from objects rather than low-level statements.

The Building Blocks of Objects

Like your car, programming objects are composed of the things that describe them and the things that can be done with them. The things that describe the object are called properties. The things you can do with an object are called methods.

Properties

Properties describe an object. If you were treating your car like an object, some of the properties might be

- Color
- Manufacturer
- Model
- What year it was made

These are things about your car that tend not to change. This is not all you need to completely describe your car, though. Probably even more important is describing the things that change often:

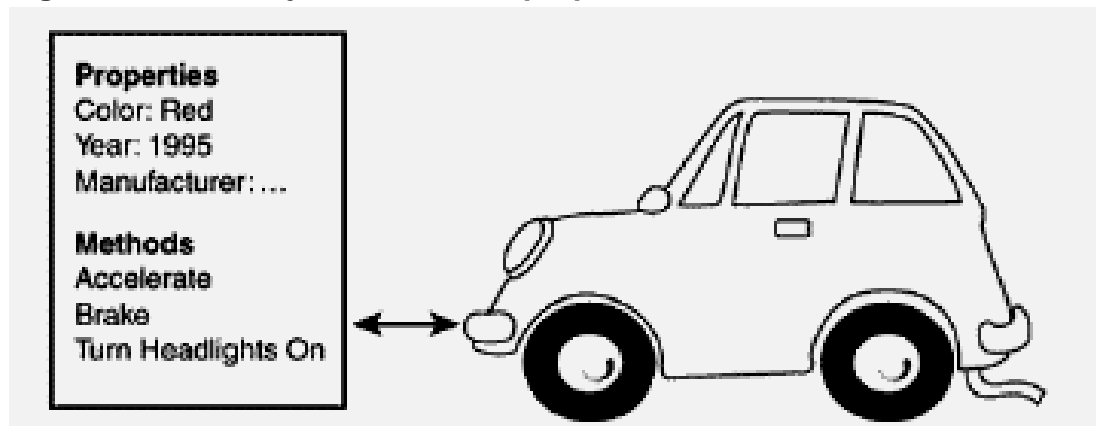
- Is it stopped or moving?
- How fast is it moving?

- Is it in forward or reverse?
- Are the headlights on?

All this information and more is needed to fully describe your car. Examining your car at particular moment in time, we might be able to describe it like this:

- Manufacturer: Ford
- Model: Explorer
- Year: 1995
- Color: Silver
- Speed: 55 mph
- Headlights: On

Figure 9.5.1. An object consists of properties and methods.



If you had an object to represent a random banner, you might have properties to represent things such as the URL a user is taken to when that banner is displayed, which would change for different banners. You might also have properties to represent the height and width of the image, which would probably be the same for all your banners. In programming, properties work pretty much the same as variables. You access properties of an object in the following way:

ObjectVariableName.Property

So if you had an object variable called `objLesson`, with a property called `Name`, you would set a value for the property `Name` like this:

```
<% objLesson.Name = "Joe" %>
```

And you would write the value of `Name` like this:

```
<% Response.Write(objLesson.Name) %>
```

Some properties are hidden. Just like you do not know all of Ford's trade secrets for making your car, you will not know everything that goes into building most of the objects you will use. Some properties may be hidden from you, just as parts inside your car are hidden from you. That does not matter, though. As long as you follow the documentation, you do not need to see everything.

Methods

Methods are the things you can do with an object. A few of your car's methods might be:

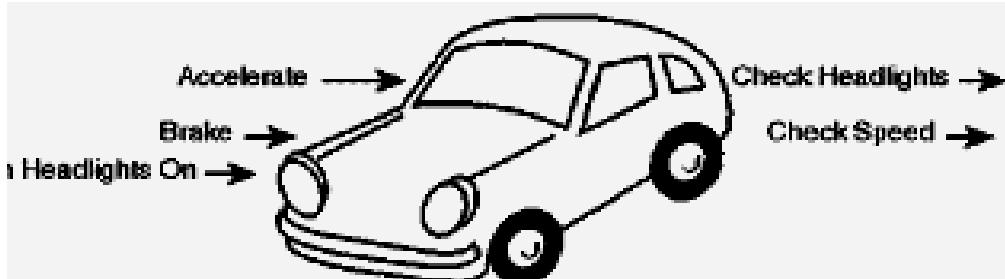
- Accelerate
- Brake
- Change gears
- Turn on headlights
- Check speed

You may notice that the method "Change gears" is, by itself, pretty meaningless. Change to which gear? That method requires more information. It receives that information based on how you move the shift stick. Accelerate needs more information and gets it based on how you press the pedal.

Methods associated with programming objects may also need information. They receive it in much the same way as the functions and subroutines we have discussed during the last two days did. Functions and subroutines receive arguments. So, too, do methods. Methods, like functions, may have zero, one, or more arguments. Methods often affect the values of properties. For example, when you use "accelerate," the value of the property "speed" should change. Methods are also often used to retrieve the values of properties. You might want to know what your current speed is. To find out, you would look at the speedometer. "Displaying the speed on the speedometer" is another example of a method. Unlike "accelerate," it does not change the speed. It merely tells you what the speed is.

Figure 9.2 shows what some of the methods of a car would be. You can see that some methods, such as "Accelerate" change properties of the car. Other methods, such as "checkspeed" give you information about the properties of the car.

Figure 9.5.2. Methods are often used to set and retrieve property values.



Methods can also tell you the value of properties. They do this by returning values, the same way functions return values. Methods are accessed like this:

ObjectVariableName.Method

Most methods behave just like the functions and subroutines discussed in the last two days. If you had a method called `Go` in `objLesson` and it returned a value, you could store the value in a variable like this:

```
MyVariable = objLesson.Go
```

or write it to the browser like this:

```
Response.Write(objLesson.Go)
```

If another method called `Compute` took a numerical argument, you would write the result to the screen like this:

```
Response.Write(objLesson.Compute(4.5))
```

or

```
Response.Write(objLesson.Compute(sngMyNumber))
```

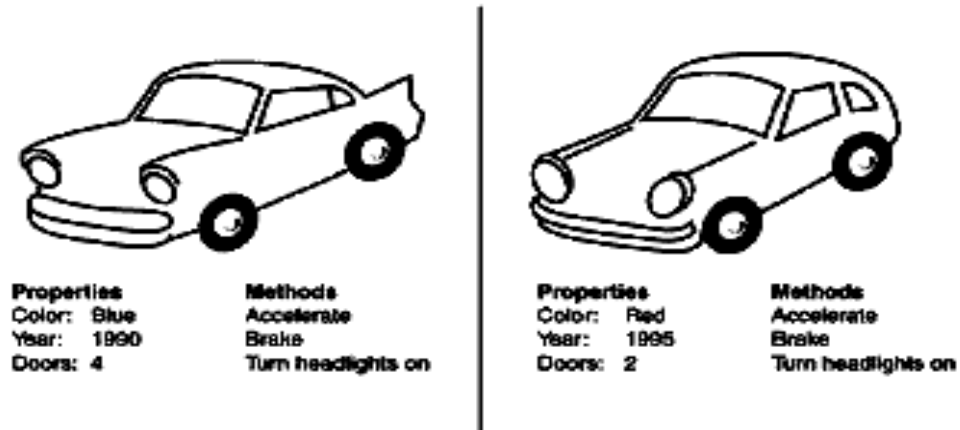
So again, there is not much difference between an object's method and a function.

Like properties, methods may be hidden.

Instances of Objects

One important thing to understand is the difference between an *instance* of an object and the object itself. In the car analogy, the object is "car." Your specific car would be one instance of "car." Your neighbor's car would be another instance of "car." The properties of "car" would be manufacturer, model, year, and so on. All instances of "car" have some value for each of these things, but they may be different values. Your car is a Ford, and your neighbor's is a Chevrolet. Both have the property "Manufacturer," but they each have different values for that property. Figure 9.5.3 illustrates how two separate instances can have different values for the properties.

Figure 9.5.3. Two instances of the same object have the same properties but different values.



Built-in ASP Objects

You may not realize it, but you have already been using an ASP object. During the last few days, you have been using `Response.Write` without really knowing what it is. `Response` is one of the six objects built-in to ASP.

1. Response Object

`Response` is used to send output. The `Write` method sends output to the user's Web browser. `Response` can also control how and when data is sent and write cookies to store information.

2. Request Object

`Request` is used to retrieve data from the client. When the client's Web browser makes a request for a particular page, it sends some information along to the server. That data is packaged together in the `Request` object. Some of it may be useful to the requested page; some of it may not be. `Request` allows the page to retrieve what it needs—cookie information, information from a form, query string data, and more. *Query string* data is the extra stuff sometimes attached at the end of a URL. It might look like `"?firstname=John&lastname=Smith"`. You have probably seen query string data before, but may not know much about it.

3. Application Object

`Application` is used to share information among several clients visiting the same group of pages. In ASP, the term *application* refers to all the `.asp` pages in a directory and its

subdirectories. Only one instance of the Application object is created per application. It is shared among all the clients accessing that application

4. Session Object

A session, on the other hand, refers to a single client accessing an application. Therefore, a new instance of the Session object is created for each session. Session is important to carrying information as a client travels between pages because Session variables persist for the entire session.

5. Server Object

The Server object provides a few basic properties and methods. Probably the most important of these is the CreateObject method. CreateObject is used to create an instance of a server component. Components are packages of related objects that you can use in your pages. They make common ASP tasks easier, and add a great deal of power to your pages. CreateObject is used in conjunction with the Set statement like this:

```
<% Set objInstance = Server.CreateObject("Class.Component") %>
```

You will see CreateObject more throughout this book. The property ScriptTimeout can be used to specify the length of time the script may be allowed to execute before an error occurs.

```
<% Server.ScriptTimeout = 90 %>
```

This specifies that if the script is still executing after 90 seconds, it should give up and produce an error message.

HTMLEncode and URLEncode are two methods that apply encoding to a string. HTMLEncode goes through the string and replaces the character "<" with "<" and ">" with ">". This causes the Web browser to display the text literally rather than interpret it as HTML tags. For example, `<%=Server.HTMLEncode("<P align=right>")%>` returns the string "<P align=right>", which the Web browser displays as `<P align=right>` rather than applying the tag. This is useful if you need to display HTML source code on your page. URLEncode applies URL encoding. Often, you may want to pass data to another page as part of the URL. This is done through the query string. Certain characters, such as the ampersand (&) have special meanings to the query string and can cause problems if you try to use them in your data. URLEncode can help encode that data so it can be

safely passed as part of the query string. The `MapPath` method converts a virtual path into a physical path. So, if your script is in `C:\mypage\www\`, `Server.MapPath("scripts/test.asp")` would return `C:\mypage\www\scripts\test.asp`. Various objects, such as the `FileSystemObject`, may require physical paths rather than virtual paths.

6. ObjectContext Object

The `ObjectContext` object is used to link ASP and the Microsoft Transaction Server. MTS is used to make web sites more scalable and improve the performance of other components.

7. ASPError Object

The `ASPError` object is new to ASP. It allows you to obtain information about script errors in your pages. It will not be covered further in this book.

Collections

Sometimes, values need to be grouped together. For example, a query string may contain any number of names and values. In these cases, a *collection* is used to store the data. A collection is a set of name/value pairs. The query string `"?firstname=John&lastname=Smith"` contains two name/value pairs. The first pair has the name "firstname" and the value "John". The second has the name "lastname" and the value "Smith". When this data is stored in the Request object, it is stored in `QueryString`, which is a collection. Suppose that you have an object instance named `Texas` that contains a collection named `Cities`. Further suppose that one of the pairs in `namedCities` has the name "Capital". The value that corresponds to "Capital" could be found using the `Item` method, like this:

```
Texas.Cities.Item("Capital")
```

This retrieves the value that corresponds to the name "Capital". To print out each name/value pair in `Cities`, you would use the `For Each...Next` statement as follows:

```
For Each varItem in Texas.Cities
Response.Write(varItem & " = " & Texas.Cities.Item(varItem) &
"<BR>")
Next
```

`Item` is the default method for collections, so `Texas.Cities("Capital")` is equivalent to `Texas.Cities.Item("Capital")`.

You can also access a value in a collection using an index number. Index numbers in collections work about like index numbers for arrays. For example, `Texas.Cities(2)` would refer to one element of the collection, and `Texas.Cities(3)` would be another. However, this is usually not useful because a pair's index number might change. For example, `Texas.Cities("LargestCity")` might have index number 5 at one point and index number 4 later. So, depending on where it is in your code, `Texas.Cities(5)` might or might not be the same as `Texas.Cities("LargestCity")`.

Be careful about using the index number. However, if all you need to do is iterate through the pairs, the index number works fine:

```
For iCount = 1 to 5
Response.Write(iCount & "th value = ")
Response.Write(Texas.Cities(iCount) & "<BR>")
Next
```

You may recall that array index numbers start at zero. For collections, index numbers start at one. This code will work if there are five pairs in `Cities`. If you do not know how many pairs there are, use the `Count` property.

```
For iCount = 1 to Bob.Frank.Count
Response.Write(iCount & "th value = ")
Response.Write(Bob.Frank(iCount) & "<BR>")
Next
```

Example 9.1. Using Collections

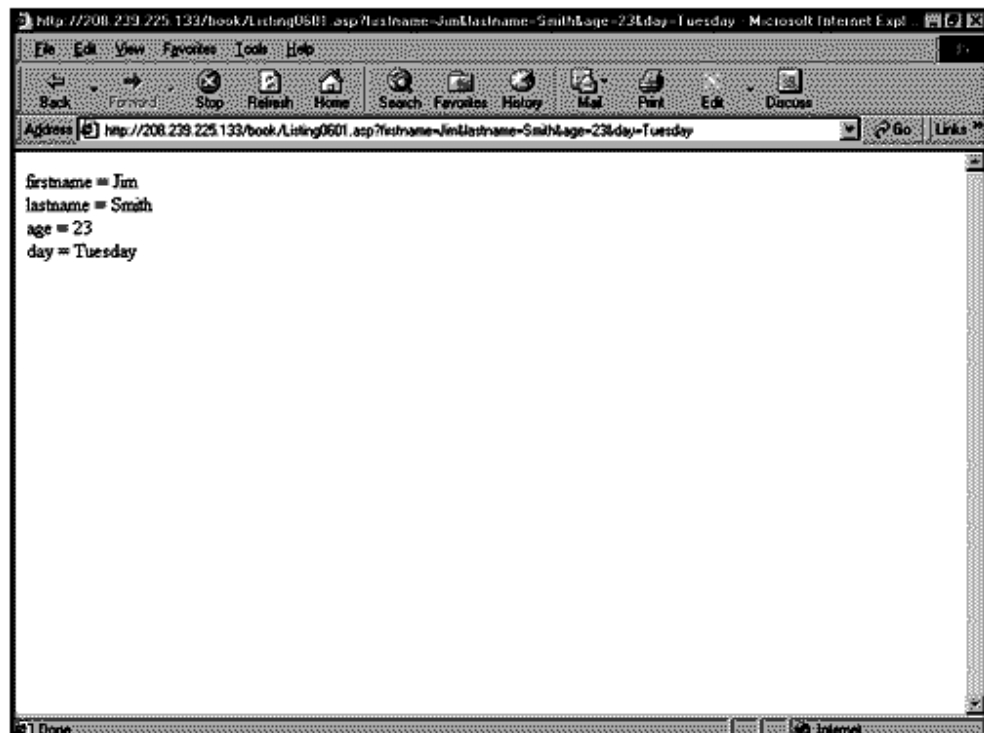
```
1: <%@ Language=VBScript %>
2: <% Option Explicit %>
3: <HTML>
4: <BODY>
5: <%
6: Dim varItem
7: For Each varItem in Request.QueryString
8: Response.Write(varItem & " = ")
9: Response.Write(Request.QueryString(varItem) & "<BR>")
10: Next
11: %>
12: </BODY>
13: </HTML>
```

This listing uses For Each (line 7) to iterate through all the name/value pairs in the QueryString collection. For each one, the name (line 8) and the value (line 9) is printed. Try viewing this script in your browser, passing in some values on the QueryString. For example, you might call it like this: listing0601.asp?firstname=Jim&lastname=Smith&age=23&day=Tuesday

This passes in four name/value pairs. firstname has value "Jim", lastname has value "Smith", age has value "23", and day has value "Tuesday". Figure 6.4 shows the results of passing these into listing9.1.

Looping through a collection (QueryString).

Figure 6.4. Looping through a collection (QueryString).



INTRODUCTION TO ASP

Unit Structure

- 10.1 Request, Response Object collections
- 10.2 ASP Applications
 - 10.2.1 Creating Active Server Page Application
 - 10.2.2 Session Object
 - 10.2.3 Session Collections
 - 10.2.4 Content Collection
 - 10.2.5 Response Object Model

10.1 WORKING WITH OBJECTS

- Now, we will write some code to demonstrate how objects are used.
- The most powerful objects you will use generally are either the built-in ASP objects or separate ASP components.
- It is possible to create simple objects within your VBScript code.
- We will do this using the VBScript Class statement.

- Listing 10.2 shows a simple class.

Example : Simple Object with No Methods

```
<%  
Class Car  
public Color  
public CurrentSpeed  
public HeadlightsOn  
End Class  
%>
```

- Above creates an object called Car, which contains three properties and no methods.
- Its three properties are Color, CurrentSpeed, and HeadlightsOn.
- The keyword public specifies that all three of them can be accessed from outside the class.
- Notice that we have not yet created any instances of Car.
- Creating an instance of an object is called *instantiation*.
- In ASP, creating an instance of an object is a two-step process.
- First, declare a variable normally, using **Dim**.
- Normal variable naming requirements apply, and it's a good idea to begin the name with obj to indicate what it is.
- Second, use the Set statement to make your variable an instance of the appropriate object.

Set *variablename* = *objectexpression*

- *objectexpression* is either the name of an object, another instance of the same object type, or the keyword New followed by a class name.
- Because we are using a class name, we will use the last form. So, if you want to create an instance of Car called objMyCar, you would do this:

```
Dim objMyCar
```

```
Set objMyCar = New Car
```

- This creates a single instance of Car. You could now set the Color property of objMyCar like this:

```
objMyCar.Color = "Blue"
```

It would not, however, make any sense to write a statement like

```
Car.Color = "Blue"
```

The class Car defines what cars will look like and what guidelines they follow. It does not actually create any cars.

If you wanted a second instance of Car called objJoesCar, and set its color as "Black", you would do it like this:

```
Dim objJoesCar
```

```
Set objJoesCar = New Car
```

```
objJoesCar.Color = "Black"
```

objMyCar would still have a value of "Blue".

- Changing the value of a property of one instance of Car does not affect the value of that property in any other instance of Car. The fact that Joe's car is black does not change the color of your car. Let's expand Car a bit. So far, to change the value of a property, we have been doing it directly, as in `objMyCar.Color = "Silver"`
- However, this is not always the best way to change the value of a property. Often, objects have properties that should not be changed, should only be changed in special circumstances, or require some other kind of special attention. Rather than leaving it to the user to understand those special circumstances, properties can be changed through methods included in the object.
- We can add such methods to Car, as shown in Example

Example: Simple Object with Methods

```
<%
Class Car
public Model
public CurrentSpeed
public HeadlightsOn
public Sub Accelerate(PercentAccel)
CurrentSpeed = CurrentSpeed * (1 + PercentAccel * 0.01)
End Sub
End Class
%>
```

This adds a method called `Accelerate` that increases `CurrentSpeed` by the percentage specified by `PercentAccel`.

Now there are two ways to change the current speed. The first is the direct approach:

```
objMyCar.CurrentSpeed = 26
```

The second way uses the new method:

```
objMyCar.Accelerate(50)
```

Either way, save the file as `CarDefinition.asp`.

Example: Car Object's Definition

```
1: <%
2: Class Car
3: private internal_color
4: private internal_speed
5: private internal_headlights
6: private Sub Class_Initialize()
7: internal_speed = 0
8: internal_color = "WHITE"
9: internal_headlights = FALSE
10: End Sub
11: public Property Get CurrentSpeed
12: CurrentSpeed = internal_speed
13: End Property
14: public Property Let CurrentSpeed(ByVal iSpeedIn)
15: internal_speed = iSpeedIn
16: End Property
17: public Property Get Color
18: Color = internal_color
19: End Property
20: public Property Let Color(ByVal strColorIn)
21: internal_color = Ucase(strColorIn)
22: End Property
23: public Sub TurnHeadlightsOn
24: internal_headlights = True
25: End Sub
26: public Sub TurnHeadlightsOff
27: internal_headlights = False
28: End Sub
29: public Function CheckHeadlights
30: if internal_headlights then
31: CheckHeadlights = "ON"
32: else
33: CheckHeadlights = "OFF"
34: end if
35: End Function
```

```
36: public Sub Accelerate(PercentAccel)
37: Dim sngMultiplier
38: sngMultiplier = (1 + PercentAccel * 0.01)
39: internal_speed = internal_speed * sngMultiplier
40: End Sub
41: End Class
42: %>
```

Above example defines the car object.

(Explanation of above example)

As mentioned before, you do not need to worry too much about how this listing works, but here is a quick look at it. Basically, the Class statement on line 2 tells VBScript what we are defining. Lines 3 through 5 define three properties. The private keyword is used with each, so they are only accessible within the object. If you want to modify them, you have to go through special methods. Lines 6 through 10 define a method. It is also private. When you create a new instance of the car object, this method will run. It initializes the values of the three private properties. This will be discussed more in the section "Events". Lines 11 through 13 and 14 through 16 define two related methods. These two methods, together, allow us to pretend we have a property called CurrentSpeed. Lines 11 through 13 set how we can read the value of CurrentSpeed using the Property Get statement. Lines 14 through 16 set how we can assign the value of the property. With these two methods together, we can treat CurrentSpeed as if it were a normal property of the object. Lines 17 through 22 define a similar "property" called Color. You can use Property Get and Property Let to control how values are given to your data. If you only want to allow certain values, you can control this. Line 21 converts the user's data into upper case before saving it, for example.

Lines 23 through 25 and 26 through 28 create two more traditional methods, for turning the headlights on or off. Lines 29 through 35 create a method that checks the headlights and returns "ON" or "OFF". Lines 36 through 40 create one more method, which increases the speed by a specified percentage. This is probably all a bit confusing for you. That is okay; writing objects and components is a pretty advanced topic. For now, what you should know is that we have a Car object. For all our purposes, it has two

properties, CurrentSpeed and Color. It has two methods for controlling the headlights, TurnHeadlightsOn and Turn Headlights Off, and one for finding out the current status of the headlights, called CheckHeadlights. Finally, there is an Accelerate method.

Then type in Listing 10.5 and call it UsingCar.asp.

If you want to find out more about VBScript's class statement, you can read about it on the Web:

<http://www.4guysfromrolla.com/webtech/092399-1.shtml>

Example 10.5. More Complex Object with Methods

```

1: <%@ Language=VBScript %>
2: <% Option Explicit %>
3: <HTML>
4: <BODY>
5: <!--#include file="CarDefinition.asp"-->
6: <%
7: Dim objMyCar
8: Set objMyCar = New Car
9: Response.Write("The car is " & objMyCar.Color)
10: Response.Write("<BR>")
11: Response.Write("The headlights are ")
12: Response.Write(objMyCar.CheckHeadlights)
13: Response.Write("<BR>")
14: objMyCar.CurrentSpeed = 44
15: Response.Write("My car is currently travelling at ")
16: Response.Write(objMyCar.CurrentSpeed & ".")
17: Response.Write("<BR>")
18: objMyCar.Accelerate(25)
19: Response.Write("My car is currently travelling at ")
20: Response.Write(objMyCar.CurrentSpeed & ".")
21: Response.Write("<BR>")
22: objMyCar.TurnHeadlightsOn
23: Response.Write("The headlights are ")
24: Response.Write(objMyCar.CheckHeadlights)
25: objMyCar.Color = "blue"
26: Response.Write("<BR>")
27: Response.Write("The car is " & objMyCar.Color)
28: Set objMyCar = Nothing
29: %>
30: </BODY>
31: </HTML>

```

Line 5 of Listing 10.5 is a server-side include. Server-side includes will be discussed more in Day 13. For now, all you need to know is that the server will insert `CarDefinition.asp` into `UsingCar.asp`. This breaks up our script to keep it from getting too long and difficult to follow.

Rather than worrying about how `CarDefinition.asp` works at the moment, let's look at what it can do. You still declare an instance of car like before, as you can see in lines 7 and 8 of Listing 10.5.

Now if you want to set `CurrentSpeed`, you may do so directly, as before. This is done in line 14. You can also affect the current speed using `Accelerate`, as in line 18. You write the current speed to the screen the same way, too, as in lines 16 and 20. If you want to turn on the headlights, use `objMyCar.TurnHeadlightsOn`, as in line 22. To turn them off, use `objMyCar.TurnHeadlightsOff`. To find out whether they are on or off, use `objMyCar.CheckHeadlights`, as in lines 12 and 24. Line 25 shows how to set the color property of the car. Finding out what color the car is works like you would expect, too, as in line 27. Notice, though, that no matter how you typed "blue", it is stored all in capital letters.

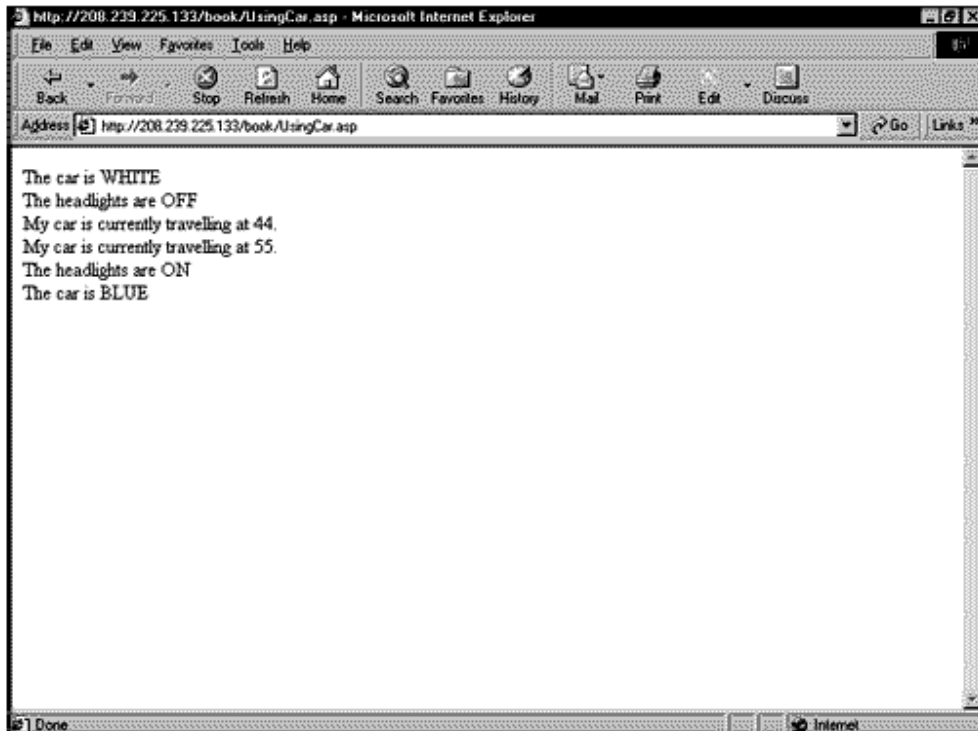
Normally, if you typed something like

```
variablename = "blue"
```

you would expect it to be stored exactly as you typed it. That is not the case here because `objMyCar.Color` is not really a variable. The data is being stored in a property that is hidden from you. `Color` acts as sort of an interface between you and the data hidden in this object. For some reason, when this object was written, it was decided not to allow lowercase letters in that particular value. Why? It does not matter. Whatever the reason, the writer chose not to make you remember this rule. Instead, he built barriers into the object. This is not to restrict your freedom. It is to help you, so that you do not have to know anything about the interior working of this object. In fact, when you set `CurrentSpeed`, the same thing happened. It appeared that `CurrentSpeed` was a simple property of `Car`, as in the earlier versions. In truth, though, `CurrentSpeed` is now another interface between you and the hidden data. Does `CurrentSpeed` do anything special to protect the data? In this case, no, but it does not matter. Again, the important thing is to know how to use the object, not

what is going on "behind the scenes" of the object. Figure 10.5 shows the results of viewing Listing 10.5. Compare the output with the code to make sure you understand how to set and change properties, and call methods.

Figure 10.5. Using the Car object.



In this version of Car, we have three hidden properties: `internal_color`, `internal_speed`, and `internal_headlights`. They are hidden because the keyword `private` is used instead of `public`. Try changing line 14 of Listing 10.5 to `objMyCar.internal_speed = 44`

This gives you an error message. Because of the keyword `private`, you cannot modify `internal_speed` directly. In fact, you cannot retrieve `internal_speed` directly either. Try changing line 16 to `Response.Write(objMyCar.internal_speed & ".")`

The only way to access the properties of Car is to go through the methods `Accelerate`, `TurnHeadlightsOn`, `TurnHeadlightsOff`, and `CheckHeadlights`, or the special subroutines `CurrentSpeed` and `Color`. The special subroutines are created using `Property Let` and `Property Get`.

In line 23 of Listing 10.5, `Set` is used again. This time, it frees up the memory used by the `objMyCar` instance of Car. Following line

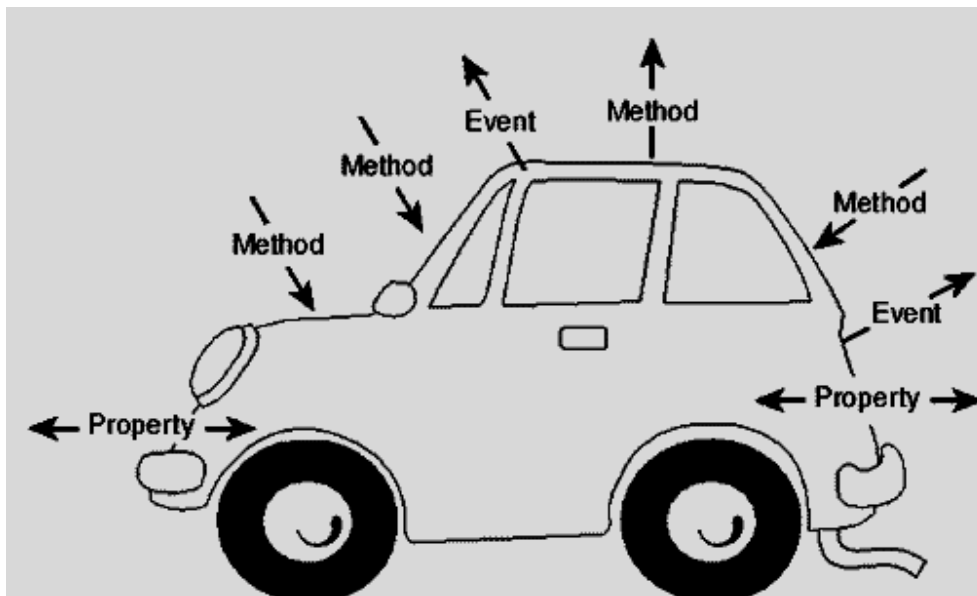
28, objMyCar is back to being an empty variable. Doing this frees up memory the system may need. A general guideline to follow is, "instantiate late, free early." You should wait as late in your code as possible before you use the first Set statement to create an instance. You should also free the memory as soon as possible. As soon as you know you are finished using an object, Set it to Nothing. Although memory is getting cheaper, a Web server may deal with thousands of visitors. It does not hurt to minimize memory use in your pages!

B. Events

- A third part of the object is the capability of the object to alert you that something has happened.
- If you have a fairly new car, it may beep at you to bring your attention to a problem: if your door is not closed, if your seat belt is not buckled, and so on. Some programming objects do something similar.
- Now if your car wants to warn you about something, it could just light up a warning on your dashboard.
- Then it is up to the driver to check the dashboard and notice the problem.
- This is fine, if it is not something too urgent.
- If it is urgent, though, it is important to get the driver's attention immediately. This is why the car beeps. This is also why we need *events*.
- Events are what objects use to let the user know that something important just happened. Some possible events your car might generate include "Driver not buckled in", or "Engine overheating". If you have a car alarm, "Person touching car" might be another.
- Unlike methods and properties, events are different from anything we have discussed before.
- When an event is generated, special code called an *event handler* can be executed.
- For example, with the event "Engine Overheating" your car would automatically start taking actions to try to cool it down. Generating events of your own is difficult and will not be discussed in this book. Writing event handlers to deal with existing events is not too difficult, though.
- Two commonly handled events are Initialize and Terminate. Initialize is the event generated when an instance of the object is created.

- Terminate is the event generated when an instance of the object is destroyed (set to Nothing). If you will recall Listing 6.4, lines 6 through 10 created a private method called Class_Initialize. This name designates the method as an event handler for the Initialize event. When an instance of Car is created, Class_Initialize is called.
- Below figure adds events to complete our picture of what an object is.

Figure: . Events send out alerts that something important has happened.



- For example, in **Example: Car Object's Definition**, (snap shot of above example)

```
6: private Sub Class_Initialize()
7: internal_speed = 0
8: internal_color = "WHITE"
9: internal_headlights = FALSE
10: End Sub
```

create an event handler for the Initialize event of the Car object.

- It sets the car's color to white, the speed to 0, and the headlights to off before you do anything with it.
- These become the default values for the properties of Car.

10.1.1 Response Object

The Response object is used to send output to the client from the web server. The syntax, collections, properties and methods of the ASP Response object are as follows:

Syntax:**Response.collection|property|method**

A.1 Collections

Collections	Description										
Cookies	<p>i. The Cookies collection sets the value of a cookie.</p> <p>ii. If you attempt to set the value of a cookie that does not exist, it is created.</p> <p>iii. If it already exists, the new value you set overwrites the old value already written to the client machine.</p> <p><i>Response.Cookies(name)[(key)].attribute=value variablename=Request.Cookies(name)[(key)].attribute]</i></p> <table border="1" data-bbox="512 1037 1279 1742"> <thead> <tr> <th>Parameter</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>name</td> <td>Required. The name of the cookie</td> </tr> <tr> <td>value</td> <td>Required for the Response.Cookies command. The value of the cookie</td> </tr> <tr> <td>attribute</td> <td>Optional. Specifies information about the cookie. Can be one of the following parameters: <ul style="list-style-type: none"> ◆ Domain - Write-only. The cookie is sent only to requests to the specified domain. ◆ Expires - Write-only. The date when the cookie expires. If not specified, the cookie will expire when the session ends. ◆ HasKeys - Read-only. Specifies whether the cookie has keys (true) or not (false). This is the only attribute that can be used with the Request.Cookies command. ◆ Path - Write-only. If set, the cookie is sent only to requests to the specified path. If not set, the application path is used. ◆ Secure - Write-only. Indicates if the cookie is secure. </td> </tr> <tr> <td>key</td> <td>Optional. Specifies the key to where the value is assigned</td> </tr> </tbody> </table> <p>Examples</p> <ul style="list-style-type: none"> • The "Response.Cookies" command is used to create a cookie or to set a cookie value: 	Parameter	Description	name	Required. The name of the cookie	value	Required for the Response.Cookies command. The value of the cookie	attribute	Optional. Specifies information about the cookie. Can be one of the following parameters: <ul style="list-style-type: none"> ◆ Domain - Write-only. The cookie is sent only to requests to the specified domain. ◆ Expires - Write-only. The date when the cookie expires. If not specified, the cookie will expire when the session ends. ◆ HasKeys - Read-only. Specifies whether the cookie has keys (true) or not (false). This is the only attribute that can be used with the Request.Cookies command. ◆ Path - Write-only. If set, the cookie is sent only to requests to the specified path. If not set, the application path is used. ◆ Secure - Write-only. Indicates if the cookie is secure. 	key	Optional. Specifies the key to where the value is assigned
Parameter	Description										
name	Required. The name of the cookie										
value	Required for the Response.Cookies command. The value of the cookie										
attribute	Optional. Specifies information about the cookie. Can be one of the following parameters: <ul style="list-style-type: none"> ◆ Domain - Write-only. The cookie is sent only to requests to the specified domain. ◆ Expires - Write-only. The date when the cookie expires. If not specified, the cookie will expire when the session ends. ◆ HasKeys - Read-only. Specifies whether the cookie has keys (true) or not (false). This is the only attribute that can be used with the Request.Cookies command. ◆ Path - Write-only. If set, the cookie is sent only to requests to the specified path. If not set, the application path is used. ◆ Secure - Write-only. Indicates if the cookie is secure. 										
key	Optional. Specifies the key to where the value is assigned										

```
<%
Response.Cookies("firstname")="Alex"
%>
```

- In the code above, we have created a cookie named "firstname" and assigned the value "Alex" to it.
- It is also possible to assign some attributes to a cookie, like setting a date when a cookie should expire:

```
<%
Response.Cookies("firstname")="Alex"
Response.Cookies("firstname").Expires=#May
10,2002#
%>
```

- Now the cookie named "firstname" has the value of "Alex", and it will expire from the user's computer at May 10, 2002.
- The "Request.Cookies" command is used to get a cookie value.
- In the example below, we retrieve the value of the cookie "firstname" and display it on a page:

```
<%
fname=Request.Cookies("firstname")
response.write("Firstname="      &      fname)
%>
```

Output:

```
Firstname=Alex
```

- A cookie can also contain a collection of multiple values.
- We say that the cookie has Keys.
- In the example below, we will create a cookie-collection named "user".
- The "user" cookie has Keys that contains information about a user:

```
<%
Response.Cookies("user")("firstname")="John"
Response.Cookies("user")("lastname")="Smith"
Response.Cookies("user")("country")="Norway"
Response.Cookies("user")("age")="25"
%>
```

- The code below reads all the cookies your server has sent to a user.
- Note that the code checks if a cookie has Keys with the HasKeys property:

```
<html>
<body>
<%
dim x,y
for each x in Request.Cookies
  response.write("<p>")
  if Request.Cookies(x).HasKeys then
    for each y in Request.Cookies(x)
      response.write(x & ":" & y & "=" &
Request.Cookies(x)(y))
      response.write("<br />")
    next
  else
    Response.Write(x & "=" & Request.Cookies(x) &
"<br />")
  end if
  response.write "</p>"
next
%>
</body>
</html>
%>
```

Output:

```
firstname=Alex
user:firstname=John
user:lastname=Smith
user:
country=Norway
user:
age=25
```

10.1.1.2 *Properties*

Properties	Description				
Buffer	<p data-bbox="608 409 1267 741"> i. The Buffer property determines whether to buffer page output or not. ii. If set to True, then output from the page is not sent to the client until the script on that page has been processed, or until the Response object Flush or End methods are called. iii. Note: If this property is set, it should be before the <html> tag in the .asp file </p> <p data-bbox="608 775 727 808">Syntax</p> <div data-bbox="608 813 1267 880" style="background-color: #e0e0e0; padding: 2px;"> <p data-bbox="608 813 1267 880">response.Buffer[=flag]</p> </div> <table border="1" data-bbox="608 936 1267 1653"> <thead> <tr> <th data-bbox="608 936 826 1003">Parameter</th> <th data-bbox="829 936 1267 1003">Description</th> </tr> </thead> <tbody> <tr> <td data-bbox="608 1003 826 1653">flag</td> <td data-bbox="829 1003 1267 1653"> <p data-bbox="829 1003 1267 1115">A boolean value that specifies whether to buffer the page output or not.</p> <p data-bbox="829 1149 1267 1395">False indicates no buffering. The server will send the output as it is processed. False is default for IIS version 4.0 (and earlier). Default for IIS version 5.0 (and later) is true.</p> <p data-bbox="829 1429 1267 1653">True indicates buffering. The server will not send output until all of the scripts on the page have been processed, or until the Flush or End method has been called.</p> </td> </tr> </tbody> </table> <p data-bbox="608 1686 770 1720">Examples</p> <p data-bbox="608 1731 775 1765">Example 1</p> <ul data-bbox="608 1776 1267 1989" style="list-style-type: none"> <li data-bbox="608 1776 1267 1877">• In this example, there will be no output sent to the browser before the loop is finished. <li data-bbox="608 1888 1267 1989">• If buffer was set to False, then it would write a line to the browser every time it went through the loop. 	Parameter	Description	flag	<p data-bbox="829 1003 1267 1115">A boolean value that specifies whether to buffer the page output or not.</p> <p data-bbox="829 1149 1267 1395">False indicates no buffering. The server will send the output as it is processed. False is default for IIS version 4.0 (and earlier). Default for IIS version 5.0 (and later) is true.</p> <p data-bbox="829 1429 1267 1653">True indicates buffering. The server will not send output until all of the scripts on the page have been processed, or until the Flush or End method has been called.</p>
Parameter	Description				
flag	<p data-bbox="829 1003 1267 1115">A boolean value that specifies whether to buffer the page output or not.</p> <p data-bbox="829 1149 1267 1395">False indicates no buffering. The server will send the output as it is processed. False is default for IIS version 4.0 (and earlier). Default for IIS version 5.0 (and later) is true.</p> <p data-bbox="829 1429 1267 1653">True indicates buffering. The server will not send output until all of the scripts on the page have been processed, or until the Flush or End method has been called.</p>				

	<pre data-bbox="619 277 1254 658"><%response.Buffer=true%> <html> <body> <% for i=1 to 100 response.write(i & "
") next %> </body> </html></pre> <p data-bbox="619 696 778 730">Example 2</p> <pre data-bbox="619 734 1254 1151"><%response.Buffer=true%> <html> <body> <p>I write some text, but I will control when the text will be sent to the browser.</p> <p>The text is not sent yet. I hold it back!</p> <p>OK, let it go!</p> <%response.Flush%> </body> </html></pre> <p data-bbox="619 1189 778 1223">Example 3</p> <pre data-bbox="619 1227 1254 1599"><%response.Buffer=true%> <html> <body> <p>This is some text I want to send to the user.</p> <p>No, I changed my mind. I want to clear the text.</p> <%response.Clear%> </body> </html></pre>
CacheControl	<ol style="list-style-type: none"> <li data-bbox="635 1615 1259 1749">i. The CacheControl property determines whether proxy servers are able to cache the output generated by ASP or not. <li data-bbox="635 1756 1259 1890">ii. If your page's content is large and doesn't change often, you might want to allow proxy servers to cache the page. <li data-bbox="635 1897 1259 1930">iii. In this case set this property to Public. <li data-bbox="635 1937 1259 1971">iv. Otherwise set it to Private.

	<p>Syntax</p> <pre>response.CacheControl[=control_header]</pre> <table border="1"> <thead> <tr> <th>Parameter</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>control_header</td> <td> <ul style="list-style-type: none"> i. A cache control header that can be set to "Public" or "Private". ii. Private is default and indicates that only private caches may cache this page. Proxy servers will not cache pages with this setting. iii. Public indicates public caches. iv. Proxy servers will cache pages with this setting. </td> </tr> </tbody> </table> <p>Examples</p> <pre><%response.CacheControl="Public"%> or <%response.CacheControl="Private"%></pre>	Parameter	Description	control_header	<ul style="list-style-type: none"> i. A cache control header that can be set to "Public" or "Private". ii. Private is default and indicates that only private caches may cache this page. Proxy servers will not cache pages with this setting. iii. Public indicates public caches. iv. Proxy servers will cache pages with this setting.
Parameter	Description				
control_header	<ul style="list-style-type: none"> i. A cache control header that can be set to "Public" or "Private". ii. Private is default and indicates that only private caches may cache this page. Proxy servers will not cache pages with this setting. iii. Public indicates public caches. iv. Proxy servers will cache pages with this setting. 				
Charset	<ul style="list-style-type: none"> i. The Charset property appends the name of the character set to the Content-Type header in the Response object. Default character set is ISO-LATIN-1. <p>Note: This property will accept any string, regardless of whether it is a valid character set or not, for the name.</p> <p>Syntax</p> <pre>response.Charset(charsetname)</pre> <table border="1"> <thead> <tr> <th>Parameter</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>charsetname</td> <td>A string that specifies a character set for the page</td> </tr> </tbody> </table> <p>Examples</p> <p>If an ASP page has no Charset property set, the content-type header would be:</p> <pre>content-type:text/html</pre>	Parameter	Description	charsetname	A string that specifies a character set for the page
Parameter	Description				
charsetname	A string that specifies a character set for the page				

	<p>If we included the Charset property:</p> <pre><%response.Charset="ISO-8859-1"%></pre> <p>the content-type header would be:</p> <pre>content-type:text/html; charset=ISO-8859-1</pre>
--	---

ContentType	<p>The ContentType property specifies the HTTP content type for the response. If not specified, the default is "text/html".</p> <p>Syntax</p> <pre>response.ContentType[=contenttype]</pre> <p>response.ContentType[=contenttype]</p> <table border="1"> <thead> <tr> <th>Parameter</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>contenttype</td> <td> <p>A string describing the content type.</p> <p>For a full list of content types, see your browser documentation or the HTTP specification.</p> </td> </tr> </tbody> </table> <p>Examples</p> <p>If an ASP page has no ContentType property set, the default content-type header would be:</p> <pre>content-type:text/html</pre> <p>Some other common ContentType values:</p> <pre><%response.ContentType="text/HTML"%> <%response.ContentType="image/GIF"%> <%response.ContentType="image/JPEG"%> <%response.ContentType="text/plain"%> <%response.ContentType="image/JPEG"%></pre> <p>This example will open an Excel spreadsheet in a browser (if the user has Excel installed):</p> <pre><%response.ContentType="application/vnd.ms-excel"%> <html> <body> <table> <tr> <td>1</td></pre>	Parameter	Description	contenttype	<p>A string describing the content type.</p> <p>For a full list of content types, see your browser documentation or the HTTP specification.</p>
Parameter	Description				
contenttype	<p>A string describing the content type.</p> <p>For a full list of content types, see your browser documentation or the HTTP specification.</p>				

	<pre> <td>2</td> <td>3</td> <td>4</td> </tr> <tr> <td>5</td> <td>6</td> <td>7</td> <td>8</td> </tr> </table> </body> </html> </pre>						
Expires	<p>i. The Expires property specifies the length of time (in minutes) that the client machine will cache the current page.</p> <p>ii. If the user returns to the page before it expires, the cached version will be displayed.</p> <p>Syntax</p> <table border="1" data-bbox="619 1064 1276 1288"> <tr> <td colspan="2" data-bbox="619 1064 1276 1126">response.Expires[=number]</td> </tr> <tr> <th data-bbox="619 1126 821 1189">Parameter</th> <th data-bbox="821 1126 1276 1189">Description</th> </tr> <tr> <td data-bbox="619 1189 821 1288">number</td> <td data-bbox="821 1189 1276 1288">The time in minutes before the page expires</td> </tr> </table> <p>Examples</p> <p>Example 1 The following code indicates that the page will never be cached:</p> <pre data-bbox="619 1480 1276 1543"><%response.Expires=-1%></pre> <p>Example 2 The following code indicates that the page will expire after 1440 minutes (24 hours):</p> <pre data-bbox="619 1688 1276 1751"><%response.Expires=1440%></pre>	response.Expires[=number]		Parameter	Description	number	The time in minutes before the page expires
response.Expires[=number]							
Parameter	Description						
number	The time in minutes before the page expires						
ExpiresAbsolute	<p>i. The ExpiresAbsolute property specifies a date and time on which a page cached on a browser will expire.</p>						

	<p>ii. If the user returns to the same page before that date and time, the user will view the cached version of the page.</p> <p>iii. If no time is specified, the page expires at midnight on the date specified.</p> <p>iv. If a date is not specified, the page expires at the given time on the day that the script is run.</p> <p>Syntax</p> <table border="1" data-bbox="624 701 1273 1323"> <tr> <td colspan="2" data-bbox="624 701 1273 763">response.ExpiresAbsolute=[date][time]</td> </tr> <tr> <th data-bbox="624 763 821 831">Parameter</th> <th data-bbox="821 763 1273 831">Description</th> </tr> <tr> <td data-bbox="624 831 821 1077">date</td> <td data-bbox="821 831 1273 1077"> <p>Specifies the date on which the page will expire.</p> <p>If this parameter is not specified, the page will expire at the specified time on the day that the script is run.</p> </td> </tr> <tr> <td data-bbox="624 1077 821 1323">time</td> <td data-bbox="821 1077 1273 1323"> <p>Specifies the time at which the page will expire.</p> <p>If this parameter is not specified, the page will expire at midnight of the specified day.</p> </td> </tr> </table> <p>Examples</p> <p>The following code indicates that the page will expire at 4:00 PM on October 11, 2009:</p> <table border="1" data-bbox="624 1469 1273 1565"> <tr> <td data-bbox="624 1469 1273 1565"> <pre><%response.ExpiresAbsolute=#October 11,2009 16:00:00#%></pre> </td> </tr> </table>	response.ExpiresAbsolute=[date][time]		Parameter	Description	date	<p>Specifies the date on which the page will expire.</p> <p>If this parameter is not specified, the page will expire at the specified time on the day that the script is run.</p>	time	<p>Specifies the time at which the page will expire.</p> <p>If this parameter is not specified, the page will expire at midnight of the specified day.</p>	<pre><%response.ExpiresAbsolute=#October 11,2009 16:00:00#%></pre>
response.ExpiresAbsolute=[date][time]										
Parameter	Description									
date	<p>Specifies the date on which the page will expire.</p> <p>If this parameter is not specified, the page will expire at the specified time on the day that the script is run.</p>									
time	<p>Specifies the time at which the page will expire.</p> <p>If this parameter is not specified, the page will expire at midnight of the specified day.</p>									
<pre><%response.ExpiresAbsolute=#October 11,2009 16:00:00#%></pre>										
<p>Is Client Connected</p>	<p>i. The IsClientConnected property is a read-only property that indicates if the client has disconnected from the web server since the last use of the Response object's Write method.</p> <p>Syntax</p> <table border="1" data-bbox="624 1821 1273 1883"> <tr> <td data-bbox="624 1821 1273 1883">response.IsClientConnected</td> </tr> </table>	response.IsClientConnected								
response.IsClientConnected										

	<p>Examples</p> <pre><% If response.IsClientConnected=true then response.write("The user is still connected!") else response.write("The user is not connected!") end if %></pre>														
<p>Pics</p>	<p>The PICS (Platform for Internet Content Selection) property adds a value to the pics-label field of the response header.</p> <p>The PICS property appends a value to the PICS label response header.</p> <p>Note: This property will accept any string value, regardless of whether it is a valid PICS label or not.</p> <p>What is PICS?</p> <p>The PICS (Platform for Internet Content Selection) rating system is used to rate the content in a web site. It looks something like this:</p> <p>PICS-1.1 "http://www.rsac.org/ratingsv01.html" by "your@name.com" for "http://www.somesite.com" on "2002.10.05T02:15-0800" r (n 0 s 0 v 0 l 0)</p> <table border="1" data-bbox="619 1330 1273 1973"> <thead> <tr> <th>Part</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>PICS-1.1</td> <td>PICS version number</td> </tr> <tr> <td>"http://www.rsac.org/ratingsv01.html"</td> <td>Rating organization</td> </tr> <tr> <td>by "your@name.com"</td> <td>Author of the label</td> </tr> <tr> <td>for "http://www.somesite.com"</td> <td>The URL or the document that has been rated</td> </tr> <tr> <td>on "2002.10.05T02:15-0800"</td> <td>Expiration date</td> </tr> <tr> <td>r (n 0 s 0 v 0 l 0)</td> <td>Rating</td> </tr> </tbody> </table>	Part	Description	PICS-1.1	PICS version number	"http://www.rsac.org/ratingsv01.html"	Rating organization	by "your@name.com"	Author of the label	for "http://www.somesite.com"	The URL or the document that has been rated	on "2002.10.05T02:15-0800"	Expiration date	r (n 0 s 0 v 0 l 0)	Rating
Part	Description														
PICS-1.1	PICS version number														
"http://www.rsac.org/ratingsv01.html"	Rating organization														
by "your@name.com"	Author of the label														
for "http://www.somesite.com"	The URL or the document that has been rated														
on "2002.10.05T02:15-0800"	Expiration date														
r (n 0 s 0 v 0 l 0)	Rating														

One of the most popular rating system is RSACi (Recreational Software Advisory Council on the Internet). RSACi rating system uses four categories: violence, nudity, sex, and language. A number between 0 to 4 is assigned to each category. 0 means that the page does not contain any potentially offensive content and 4 means that the page contains the highest levels of potentially offensive content.

Level	Violence Rating	Nudity Rating	Sex Rating	Language Rating
0	None of the below or sports related	None of the below	None of the below or innocent kissing; romance	None of the below
1	Injury to human being	Revealing attire	Passionate kissing	Mild expletives
2	Destruction of realistic objects	Partial nudity	Clothed sexual touching	Moderate expletives or profanity
3	Aggressive violence or death to humans	Frontal nudity	Non-explicit sexual acts	Strong language or hate speech
4	Rape or wanton, gratuitous violence	Frontal nudity (qualifying as provocative display)	Explicit sexual acts or sex crimes	Crude, vulgar language or extreme hate speech

There are two ways you can obtain rating for your site. You can either rate your site yourself or use a rating provider, like RSACi. They'll ask you fill out some questions. After filling out the questions, you will get the rating label for your site.

Microsoft IE 3.0 and above and Netscape 4.5 and above support the content ratings. You can set the ratings in IE 5, by selecting Tools and Internet Options. Select the Content tab and click the Enable. When the rating exceeds the defined levels the Content Advisor will block the site. You can set the ratings in Netscape 4.7, by selecting Help and NetWatch.

	<p>We can use the META tag or the response.PICS property to add a rating to our site.</p> <p>Syntax</p> <table border="1" data-bbox="624 450 1278 678"> <tr> <th colspan="2">response.PICS(piclabel)</th> </tr> <tr> <th>Parameter</th> <th>Description</th> </tr> <tr> <td>piclabel</td> <td>A properly formatted PICS label</td> </tr> </table> <p>Examples For an ASP file that includes:</p> <p>Note: Because PICS labels contain quotes, you must replace quotes with " & chr(34) & ".</p> <pre data-bbox="624 920 1278 1339"> <% response.PICS("(PICS-1.1 <http://www.rsac.org/ratingv01.html> by " & chr(34) & "your@name.com" & chr(34) & " for " & chr(34) & "http://www.somesite.com" & chr(34) & " on " & chr(34) & "2002.10.05T02:15-0800" & chr(34) & " r (n 2 s 0 v 1 2))") %> </pre> <p>the following header is added:</p> <pre data-bbox="624 1413 1278 1653"> PICS-label:(PICS-1.1 <http://www.rsac.org/ratingv01.html> by "your@name.com" for "http://www.somesite.com" on "2002.10.05T02:15-0800" r (n 2 s 0 v 1 2)) </pre>	response.PICS(piclabel)		Parameter	Description	piclabel	A properly formatted PICS label
response.PICS(piclabel)							
Parameter	Description						
piclabel	A properly formatted PICS label						
Status	<p>The Status property specifies the value of the status line that is returned to the client machine from the web server.</p> <p>Tip: Use this property to modify the status line returned by the server.</p>						

Syntax	
response.Status=statusdescription	
Parameter	Description
statusdescription	A three-digit number and a description of that code, like 404 Not Found. Note: Status values are defined in the HTTP specification.
Examples	
<pre><% ip=request.ServerVariables("REMOTE_ADDR") if ip<>"194.248.333.500" then response.Status="401 Unauthorized" response.Write(response.Status) response.End end if %></pre>	

10.1.1.3 Methods

Methods	Description
AddHeader	<p>i. The AddHeader method allows you to add your own HTTP header with a specified value.</p> <p>ii. If you add an HTTP header with the same name as a previously added header, the second header will be sent in addition to the first; adding the second header does not overwrite the value of the first header with the same name.</p> <p>iii. Also, once the header has been added to the HTTP response, it cannot be removed.</p> <p>Note: Once a header has been added, it cannot be removed.</p> <p>Note: In IIS 4.0 you have to call this method before any output is sent to the browser. In IIS</p>

	<p>5.0 you can call the AddHeader method at any point in the script, as long as it precedes any calls to the response.Flush method.</p> <p>Syntax</p> <table border="1" data-bbox="579 479 1267 882"> <tr> <td colspan="2" data-bbox="579 479 1267 546">response.AddHeader name,value</td> </tr> <tr> <th data-bbox="579 546 775 613">Parameter</th> <th data-bbox="775 546 1267 613">Description</th> </tr> <tr> <td data-bbox="579 613 775 770">name</td> <td data-bbox="775 613 1267 770">Required. The name of the new header variable (cannot contain underscores)</td> </tr> <tr> <td data-bbox="579 770 775 882">value</td> <td data-bbox="775 770 1267 882">Required. The initial value of the new header variable</td> </tr> </table> <p>Examples</p> <pre data-bbox="579 965 1267 1070"><%Response.AddHeader "WARNING","Error message text"%></pre>	response.AddHeader name,value		Parameter	Description	name	Required. The name of the new header variable (cannot contain underscores)	value	Required. The initial value of the new header variable
response.AddHeader name,value									
Parameter	Description								
name	Required. The name of the new header variable (cannot contain underscores)								
value	Required. The initial value of the new header variable								
AppendToLog	<p>i. The AppendToLog method adds a string to the end of the Web server log entry for the current page request.</p> <p>ii. You can only add up to 80 characters at a time, but you can call this method multiple times.</p> <p>Syntax</p> <table border="1" data-bbox="579 1408 1267 1700"> <tr> <td colspan="2" data-bbox="579 1408 1267 1476">response.AppendToLog string</td> </tr> <tr> <th data-bbox="579 1476 775 1543">Parameter</th> <th data-bbox="775 1476 1267 1543">Description</th> </tr> <tr> <td data-bbox="579 1543 775 1700">string</td> <td data-bbox="775 1543 1267 1700">Required. The text to append to the log file (cannot contain any comma characters)</td> </tr> </table> <p>Examples</p> <pre data-bbox="579 1783 1267 1865"><%Response.AppendToLog "My log message"%></pre>	response.AppendToLog string		Parameter	Description	string	Required. The text to append to the log file (cannot contain any comma characters)		
response.AppendToLog string									
Parameter	Description								
string	Required. The text to append to the log file (cannot contain any comma characters)								
BinaryWrite	The BinaryWrite method writes information directly to the output without any character								

	<p>conversion.</p> <p>Syntax</p> <table border="1" data-bbox="579 421 1267 674"> <tr> <td colspan="2" data-bbox="579 421 1267 488">response.BinaryWrite data</td> </tr> <tr> <th data-bbox="579 488 775 555">Parameter</th> <th data-bbox="775 488 1267 555">Description</th> </tr> <tr> <td data-bbox="579 555 775 674">data</td> <td data-bbox="775 555 1267 674">Required. The binary information to be sent</td> </tr> </table> <p>Example</p> <p>If you have an object that generates an array of bytes, you can use BinaryWrite to send the bytes to an application:</p> <pre data-bbox="579 880 1267 1115"><% Set objBinaryGen=Server.CreateObject("MyComponents.BinaryGenerator") pic=objBinaryGen.MakePicture response.BinaryWrite pic %></pre>	response.BinaryWrite data		Parameter	Description	data	Required. The binary information to be sent
response.BinaryWrite data							
Parameter	Description						
data	Required. The binary information to be sent						
Clear	<p>i. The Clear method erases any buffered HTML output. It does so without sending any of the buffered response to the client.</p> <p>ii. Calling Clear will cause an error if Buffer property of the Response object is not set to True.</p> <p>Note: This method does not clear the response headers, only the response body.</p> <p>Note: If response.Buffer is false, this method will cause a run-time error.</p> <p>Syntax</p> <pre data-bbox="579 1693 1267 1760">response.Clear</pre> <p>Examples</p> <pre data-bbox="579 1845 1267 2013"><% response.Buffer=true %> <html></pre>						

	<pre><body> <p>This is some text I want to send to the user.</p> <p>No, I changed my mind. I want to clear the text.</p> <% response.Clear %> </body> </html></pre> <p>Output: (nothing)</p>
End	<p>i. The End method stops the processing of the script in the current page and sends the already created content to the client.</p> <p>ii. Any code present after the call to the End method is not processed.</p> <p>Note: This method will flush the buffer if Response.Buffer has been set to true. If you do not want to return any output to the user, you should call Response.Clear first.</p> <p>Syntax</p> <pre>Response.End</pre> <p>Examples</p> <pre><html> <body> <p>I am writing some text. This text will never be <% Response.End %> finished! It's too late to write more!</p> </body> </html></pre> <p>Output: I am writing some text. This text will never be</p>

Flush	<p>i. The Flush method sends buffered output to the client immediately.</p> <p>ii. Flush will cause a run-time error if Buffer property of the Response object is not set to True.</p> <p>Note: If response.Buffer is false, this method will cause a run-time error.</p> <p>Syntax</p> <pre>Response.Flush</pre> <p>Example</p> <pre><% Response.Buffer=true %> <html> <body> <p>I write some text, but I will control when the text will be sent to the browser.</p> <p>The text is not sent yet. I hold it back!</p> <p>OK, let it go!</p> <% Response.Flush %> </body> </html></pre> <p>Output:</p> <p>I write some text, but I will control when the text will be sent to the browser.</p> <p>The text is not sent yet. I hold it back!</p> <p>OK, let it go!</p>
Redirect	<p>i. The Redirect method redirects the user to a different URL</p>

	<p>Syntax</p> <table border="1" data-bbox="579 353 1270 629"> <tr> <th colspan="2">Response.Redirect URL</th> </tr> <tr> <th>Parameter</th> <th>Description</th> </tr> <tr> <td>URL</td> <td>Required. The URL that the user (browser) is redirected to</td> </tr> </table> <p>Examples</p> <pre data-bbox="579 712 1270 857"><% Response.Redirect "http://www.w3schools.com" %></pre>	Response.Redirect URL		Parameter	Description	URL	Required. The URL that the user (browser) is redirected to
Response.Redirect URL							
Parameter	Description						
URL	Required. The URL that the user (browser) is redirected to						
Write	<p>The Write method writes a specified string to the output.</p> <p>Syntax</p> <table border="1" data-bbox="579 1016 1270 1218"> <tr> <th colspan="2">Response.Write variant</th> </tr> <tr> <th>Parameter</th> <th>Description</th> </tr> <tr> <td>variant</td> <td>Required. The data to write</td> </tr> </table> <p>Examples</p> <p>Example 1</p> <pre data-bbox="579 1346 1270 1630"><% Response.Write "Hello World" %></pre> <p>Output: Hello World</p> <p>Example 2</p> <pre data-bbox="579 1715 1270 1973"><% name="John" Response.Write(name) %></pre> <p>Output: John</p>	Response.Write variant		Parameter	Description	variant	Required. The data to write
Response.Write variant							
Parameter	Description						
variant	Required. The data to write						

	<p>Example 3</p> <pre><% Response.Write("Hello
World") %></pre> <p>Output: Hello World</p>
--	---

10.1.2 Request Object

- The Request object makes available all the values that client browser passes to the server during an HTTP request. It includes client browser info, cookie details (of this domain only), client certificates (if accessing through SSL) etc.

Syntax Request.collection property method (variable)	
Collections	Description
ClientCertificate	<ol style="list-style-type: none"> i. Makes us available a collection of values stored in the client certificate that is sent to the HTTP request. ii. It is usually of interest when the client is requesting secure pages through SSL connection. iii. But before using this collection the server should be configured to request client certificates.
Cookies	<ol style="list-style-type: none"> i. Makes us available all the cookies stored in the client browser for this domain. ii. the Cookies collection is used to set or get cookie values. If the cookie does not exist, it will be created, and take the value that is specified. <p>Note: The Response.Cookies command must appear before the <html> tag.</p> <p>Syntax</p> <pre>Response.Cookies(name)[(key)].attribute]=value variablename=Request.Cookies(name)[(key)]. attribute]</pre>

Parameter	Description
name	Required. The name of the cookie
value	Required for the Response.Cookies command. The value of the cookie
attribute	<p>Optional. Specifies information about the cookie. Can be one of the following parameters:</p> <ul style="list-style-type: none"> • Domain - Write-only. The cookie is sent only to requests to this domain • Expires - Write-only. The date when the cookie expires. If no date is specified, the cookie will expire when the session ends • HasKeys - Read-only. Specifies whether the cookie has keys (This is the only attribute that can be used with the Request.Cookies command) • Path - Write-only. If set, the cookie is sent only to requests to this path. If not set, the application path is used • Secure - Write-only. Indicates if the cookie is secure
key	Optional. Specifies the key to where the value is assigned
<p>Examples</p> <p>The "Response.Cookies" command is used to create a cookie or to set a cookie value:</p> <pre data-bbox="576 1720 1294 1868"><% Response.Cookies("firstname")="Alex" %></pre> <p>In the code above, we have created a cookie named "firstname" and assigned the value "Alex" to it</p>	

It is also possible to assign some attributes to a cookie, like setting a date when a cookie should expire:

```
<%  
Response.Cookies("firstname")="Alex"  
Response.Cookies("firstname").Expires=#May  
10,2002#  
%>
```

Now the cookie named "firstname" has the value of "Alex", and it will expire from the user's computer at May 10, 2002.

The "Request.Cookies" command is used to get a cookie value.

In the example below, we retrieve the value of the cookie "firstname" and display it on a page:

```
<%  
fname=Request.Cookies("firstname")  
response.write("Firstname=" & fname)  
%>
```

Output:

```
Firstname=Alex
```

A cookie can also contain a collection of multiple values. We say that the cookie has Keys.

In the example below, we will create a cookie-collection named "user". The "user" cookie has Keys that contains information about a user:

```
<%  
Response.Cookies("user")("firstname")="John"  
Response.Cookies("user")("lastname")="Smith"  
Response.Cookies("user")("country")="Norway"  
Response.Cookies("user")("age")="25"  
%>
```

	<p>The code below reads all the cookies your server has sent to a user. Note that the code checks if a cookie has Keys with the HasKeys property:</p> <pre data-bbox="576 427 1302 1413"> <html> <body> <% dim x,y for each x in Request.Cookies response.write("<p>") if Request.Cookies(x).HasKeys then for each y in Request.Cookies(x) response.write(x & ":" & y & "=" & Request.Cookies(x)(y)) response.write("
") next else Response.Write(x & "=" & Request.Cookies(x) & "
") end if response.write "</p>" next %> </body> </html> %> </pre> <p>Output:</p> <pre data-bbox="576 1496 1302 1809"> firstname=Alex user:firstname=John user:lastname=Smith user: country=Norway user: age=25 </pre>
Form	Collection of all the values of Form element in the HTTP request.

Syntax

Request.Form(element)[(index)].Count

Parameter	Description
element	Required. The name of the form element from which the collection is to retrieve values
index	Optional. Specifies one of multiple values for a parameter. From 1 to Request.Form(parameter).Count.

Examples**Example 1**

You can loop through all the values in a form request. If a user filled out a form by specifying two values - Blue and Green - for the color element, you could retrieve those values like this:

```
<% for i=1 to Request.Form("color").Count
  Response.Write(Request.Form("color")(i) & "<br
/>")
next
%>
```

Output:

```
Blue
Green
```

Example 2

Consider the following form:

```
<form action="submit.asp" method="post">
<p>First name: <input name="firstname"></p>
<p>Last name: <input name="lastname"></p>
<p>Your favorite color:
<select name="color">
<option>Blue</option>
<option>Green</option>
<option>Red</option>
<option>Yellow</option>
<option>Pink</option>
```

	<pre> </select> </p> <p><input type="submit"></p> </form> </pre> <p>The following request might be sent:</p> <pre> firstname=John&lastname=Dove&color=Red </pre> <p>Now we can use the information from the form in a script:</p> <pre> Hi, <%=Request.Form("firstname")%>. Your favorite color is <%=Request.Form("color")%>. </pre> <p>Output:</p> <pre> Hi, John. Your favorite color is Red. </pre> <p>If you do not specify any element to display, like this:</p> <pre> Form data is: <%=Request.Form%> </pre> <p>the output would look like this:</p> <pre> Form data is: firstname=John&lastname=Dove&color=Red </pre>
QueryString	<p>Collection of variables which are stored in the HTTP query string. Name / value pairs can also be appended to the URL after the end of page name e.g.</p> <p><i>"http://www.stardeveloper.com/asp_request.asp?author=Faisal+Khan"</i> contains one variable <i>'author'</i> with a value of <i>'Faisal Khan'</i>.</p> <p>The QueryString collection is used to retrieve the variable values in the HTTP query string. The HTTP query string is specified by the values following the question mark (?), like this:</p> <pre> Link with a query string </pre> <p>The line above generates a variable named txt with the value "this is a query string test".</p>

Query strings are also generated by form submission, or by a user typing a query into the address bar of the browser.

Note: If you want to send large amounts of data (beyond 100 kb) the Request.QueryString cannot be used.

Syntax

Request.QueryString(variable)[(index)].Count

Parameter	Description
variable	Required. The name of the variable in the HTTP query string to retrieve
index	Optional. Specifies one of multiple values for a variable. From 1 to Request.QueryString(variable).Count

Examples

Example 1

To loop through all the n variable values in a Query String:

The following request is sent:

```
http://www.w3schools.com/test/names.asp?n=John&n=Susan
```

and names.asp contains the following script:

```
<%
for i=1 to Request.QueryString("n").Count
  Response.Write(Request.QueryString("n")(i) &
"<br />")
next
%>
```

The file names.asp would display the following:

```
John
Susan
```

	<p>Example 2 The following string might be sent:</p> <pre>http://www.w3schools.com/test/names.asp?name=John&age=30</pre> <p>this results in the following QUERY_STRING value:</p> <pre>name=John&age=30</pre> <p>Now we can use the information in a script:</p> <pre>Hi, <%=Request.QueryString("name")%>. Your age is <%= Request.QueryString("age")%>.</pre> <p>Output:</p> <pre>Hi, John. Your age is 30.</pre> <p>If you do not specify any variable values to display, like this:</p> <pre>Query string is: <%=Request.QueryString%></pre> <p>the output would look like this:</p> <pre>Query string is: name=John&age=30</pre>								
ServerVariables	<p>Contains a collection of predetermined environment variables plus a collection of all the HTTP header values sent from the client browser to the server.</p> <p>The ServerVariables collection is used to retrieve the server variable values.</p> <p>Syntax</p> <pre>Request.ServerVariables (server_variable)</pre> <table border="1"> <thead> <tr> <th>Parameter</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>server_variable</td> <td>Required. The name of the <u>server variable</u> to retrieve</td> </tr> </tbody> </table> <p>Server Variables</p> <table border="1"> <thead> <tr> <th>Variable</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>ALL_HTTP</td> <td>Returns all HTTP headers sent by the client. Always prefixed with HTTP_ and capitalized</td> </tr> </tbody> </table>	Parameter	Description	server_variable	Required. The name of the <u>server variable</u> to retrieve	Variable	Description	ALL_HTTP	Returns all HTTP headers sent by the client. Always prefixed with HTTP_ and capitalized
Parameter	Description								
server_variable	Required. The name of the <u>server variable</u> to retrieve								
Variable	Description								
ALL_HTTP	Returns all HTTP headers sent by the client. Always prefixed with HTTP_ and capitalized								

ALL_RAW	Returns all headers in raw form
APPL_MD_PATH	Returns the meta base path for the application for the ISAPI DLL
APPL_PHYSICAL_PATH	Returns the physical path corresponding to the meta base path
AUTH_PASSWORD	Returns the value entered in the client's authentication dialog
AUTH_TYPE	The authentication method that the server uses to validate users
AUTH_USER	Returns the raw authenticated user name
CERT_COOKIE	Returns the unique ID for client certificate as a string
CERT_FLAGS	bit0 is set to 1 if the client certificate is present and bit1 is set to 1 if the cCertification authority of the client certificate is not valid
CERT_ISSUER	Returns the issuer field of the client certificate
CERT_KEYSIZE	Returns the number of bits in Secure Sockets Layer connection key size
CERT_SECRETKEYSIZE	Returns the number of bits in server certificate private key
CERT_SERIALNUMBER	Returns the serial number field of the client certificate
CERT_SERVER_ISSUER	Returns the issuer field of the server certificate

CERT_SERVER_SUBJECT	Returns the subject field of the server certificate
CERT_SUBJECT	Returns the subject field of the client certificate
CONTENT_LENGTH	Returns the length of the content as sent by the client
CONTENT_TYPE	Returns the data type of the content
GATEWAY_INTERFACE	Returns the revision of the CGI specification used by the server
HTTP_<HeaderName>	Returns the value stored in the header <i>Header Name</i>
HTTP_ACCEPT	Returns the value of the Accept header
HTTP_ACCEPT_LANGUAGE	Returns a string describing the language to use for displaying content
HTTP_COOKIE	Returns the cookie string included with the request
HTTP_REFERER	Returns a string containing the URL of the page that referred the request to the current page using an <a> tag. If the page is redirected, HTTP_REFERER is empty
HTTP_USER_AGENT	Returns a string describing the browser that sent the request
HTTPS	Returns ON if the request came in through secure channel or OFF if the request came in through a non-secure channel

HTTPS_KEYSIZE	Returns the number of bits in Secure Sockets Layer connection key size
HTTPS_SECRETKEYSIZE	Returns the number of bits in server certificate private key
HTTPS_SERVER_ISSUER	Returns the issuer field of the server certificate
HTTPS_SERVER_SUBJECT	Returns the subject field of the server certificate
INSTANCE_ID	The ID for the IIS instance in text format
INSTANCE_META_PATH	The meta base path for the instance of IIS that responds to the request
LOCAL_ADDR	Returns the server address on which the request came in
LOGON_USER	Returns the Windows account that the user is logged into
PATH_INFO	Returns extra path information as given by the client
PATH_TRANSLATED	A translated version of PATH_INFO that takes the path and performs any necessary virtual-to-physical mapping
QUERY_STRING	Returns the query information stored in the string following the question mark (?) in the HTTP request
REMOTE_ADDR	Returns the IP address of the remote host making the request
REMOTE_HOST	Returns the name of the host making the request

REMOTE_USER	Returns an unmapped user-name string sent in by the user
REQUEST_METHOD	Returns the method used to make the request
SCRIPT_NAME	Returns a virtual path to the script being executed
SERVER_NAME	Returns the server's host name, DNS alias, or IP address as it would appear in self-referencing URLs
SERVER_PORT	Returns the port number to which the request was sent
SERVER_PORT_SECURE	Returns a string that contains 0 or 1. If the request is being handled on the secure port, it will be 1. Otherwise, it will be 0
SERVER_PROTOCOL	Returns the name and revision of the request information protocol
SERVER_SOFTWARE	Returns the name and version of the server software that answers the request and runs the gateway
URL	Returns the base portion of the URL
<p>Examples</p> <p>You can loop through all of the server variables like this:</p> <pre><% for each x in Request.ServerVariables response.write(x & "
") next %></pre>	

The following example demonstrates how to find out the visitor's browser type, IP address, and more:

```
<html>
<body>
<p>
<b>You are browsing this site with:</b>
<%Response.Write(Request.ServerVariables("http_user_agent"))%>
</p>
<p>
<b>Your IP address is:</b>
<%Response.Write(Request.ServerVariables("remote_addr"))%>
</p>
<p>
<b>The DNS lookup of the IP address is:</b>
<%Response.Write(Request.ServerVariables("remote_host"))%>
</p>
<p>
<b>The method used to call the page:</b>
<%Response.Write(Request.ServerVariables("request_method"))%>
</p>
<p>
<b>The server's domain name:</b>
<%Response.Write(Request.ServerVariables("server_name"))%>
</p>
<p>
<b>The server's port:</b>
<%Response.Write(Request.ServerVariables("server_port"))%>
</p>
<p>
<b>The server's software:</b>
<%Response.Write(Request.ServerVariables("server_software"))%>
</p>
</body>
</html>
```

Properties	Description
TotalBytes	An <i>integer</i> read-only value which gives the total number of bytes the client browser is to the server with each request.
Methods	Description
BinaryRead	Retrieves data sent to the server from the client in raw format as part of the POST request. It saves all this data in a <i>SafeArray</i> .

10.2 APPLICATION OBJECT

- i. The Application object is created when the first .asp page is requested after starting the IIS and remains there until the server shuts down.
- ii. All the variables created with application object have application level scope meaning there by that they are accessible to all the users.
- iii. All .asp pages in a virtual directory and its subdirectories come under the application scope. So application level variables are shared by more than one user at a time. The Application object is used to store and access variables from any page, just like the Session object.
- iv. The difference is that ALL users share ONE Application object (with Sessions there is ONE Session object for EACH user).
- v. The Application object holds information that will be used by many pages in the application (like database connection information).
- vi. The information can be accessed from any page. The information can also be changed in one place, and the changes will automatically be reflected on all pages.

Syntax : Application.method

response.ContentType[=contenttype]

Collections	Description				
Contents	<p data-bbox="536 344 1243 421">A collection of all the items that have been added to the Application object.</p> <p data-bbox="536 454 644 488">Syntax</p> <div data-bbox="536 495 1243 600" style="background-color: #e0e0e0; padding: 5px;"> <pre data-bbox="536 495 900 573">Application.Contents(Key) Session.Contents(Key)</pre> </div> <table border="1" data-bbox="536 600 1243 786"> <thead> <tr> <th data-bbox="536 600 724 667">Parameter</th> <th data-bbox="729 600 1243 667">Description</th> </tr> </thead> <tbody> <tr> <td data-bbox="536 667 724 786">key</td> <td data-bbox="729 667 1243 786">Required. The name of the item to retrieve</td> </tr> </tbody> </table> <p data-bbox="536 831 1078 864">Examples for the Application Object</p> <p data-bbox="536 871 691 904">Example 1</p> <p data-bbox="536 911 1243 987">Notice that both name and objtest would be appended to the Contents collection:</p> <div data-bbox="536 994 1243 1272" style="background-color: #e0e0e0; padding: 5px;"> <pre data-bbox="552 1001 1214 1238"><% Application("name")="W3Schools" Set Application("objtest")=Server.CreateObject("AD ODB.Connection") %></pre> </div> <p data-bbox="536 1312 695 1346">Example 2</p> <p data-bbox="536 1352 1091 1386">To loop through the Contents collection:</p> <div data-bbox="536 1393 1243 1984" style="background-color: #e0e0e0; padding: 5px;"> <pre data-bbox="552 1400 1031 1637"><% for each x in Application.Contents Response.Write(x & "=" & Application.Contents(x) & "
") next %></pre> <p data-bbox="552 1688 592 1722">or:</p> <pre data-bbox="552 1729 1078 1966"><% For i=1 to Application.Contents.Count Response.Write(i & "=" & Application.Contents(i) & "
") Next %></pre> </div>	Parameter	Description	key	Required. The name of the item to retrieve
Parameter	Description				
key	Required. The name of the item to retrieve				

Example 3

```
<%
Application("date")="2001/05/05"
Application("author")="W3Schools"

for each x in Application.Contents
  Response.Write(x & "=" &
Application.Contents(x) & "<br />")
next
%>
```

Output:

```
date=2001/05/05
author=W3Schools
```

Examples for the Session Object**Example 1**

Notice that both name and object would be appended to the Contents collection:

```
<%
Session("name")="Hege"
Set
Session("objtest")=Server.CreateObject("ADOD
B.Connection")
%>
```

Example 2

To loop through the Contents collection:

```
<%
for each x in Session.Contents
  Response.Write(x & "=" & Session.Contents(x)
& "<br />")
next
%>
```

or:

```
<%
For i=1 to Session.Contents.Count
```

	<pre>Response.Write(i & "=" & Session.Contents(i) & "
") Next %></pre> <p>Example 3</p> <pre><% Session("name")="Hege" Session("date")="2001/05/05" for each x in Session.Contents Response.Write(x & "=" & Session.Contents(x) & "
") next %></pre> <p>Output:</p> <pre>name=Hege date=2001/05/05</pre>				
<p>Static Objects</p>	<p>Collection of all the items that have been added to the Application object through <i><object></i> tag.</p> <p>Syntax</p> <pre>Application.StaticObjects(Key) Session.StaticObjects(Key)</pre> <table border="1"> <thead> <tr> <th>Parameter</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>key</td> <td>Required. The name of the item to retrieve</td> </tr> </tbody> </table> <p>Examples for the Application Object:</p> <p>Example 1</p> <p>To loop through the StaticObjects collection:</p> <pre><% for each x in Application.StaticObjects Response.Write(x & "
") next %></pre>	Parameter	Description	key	Required. The name of the item to retrieve
Parameter	Description				
key	Required. The name of the item to retrieve				

Example 2

In Global.asa:

```
<object runat="server" scope="application"
id="MsgBoard" progid="msgboard.MsgBoard">
</object>

<object runat="server" scope="application"
id="AdRot" progid="MSWC.AdRotator">
</object>
```

In an ASP file:

```
<%
for each x in Application.StaticObjects
  Response.Write(x & "<br />")
next
%>
Output:

MsgBoard
AdRot
```

Examples for the Session Object**Example 1**

To loop through the StaticObjects collection:

```
<%
for each x in Session.StaticObjects
  Response.Write(x & "<br />")
next
%>
```

Example 2

In Global.asa:

```
<object runat="server" scope="session"
id="MsgBoard" progid="msgboard.MsgBoard">
</object>

<object runat="server" scope="session"
id="AdRot" progid="MSWC.AdRotator">
</object>
```

	<p>In an ASP file:</p> <pre><% for each x in Session.StaticObjects Response.Write(x & "
") next %></pre> <p>Output: MsgBoard AdRot</p>						
Methods	Description						
Contents.Remove	<p>Deletes the specified item from the <i>Application.Contents</i> collection.</p> <p>The Contents.Remove method deletes an item from the Contents collection.</p> <p>Syntax</p> <pre>Application.Contents.Remove(name index) Session.Contents.Remove(name index)</pre> <table border="1" data-bbox="636 1234 1252 1469"> <thead> <tr> <th>Parameter</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>name</td> <td>The name of the item to remove</td> </tr> <tr> <td>index</td> <td>The index of the item to remove</td> </tr> </tbody> </table> <p>Examples for the Application Object</p> <p>Example 1</p> <pre><% Application("test1")="First test" Application("test2")="Second test" Application("test3")="Third test" Application.Contents.Remove("test2") for each x in Application.Contents</pre>	Parameter	Description	name	The name of the item to remove	index	The index of the item to remove
Parameter	Description						
name	The name of the item to remove						
index	The index of the item to remove						

```

Response.Write(x & "=" &
Application.Contents(x) & "<br />")
next
%>

```

Output:

```

test1=First test
test3=Third test

```

Example 2

```

<%
Application("test1")="First test"
Application("test2")="Second test"
Application("test3")="Third test"

Application.Contents.Remove(2)

for each x in Application.Contents
  Response.Write(x & "=" &
Application.Contents(x) & "<br />")
next
%>

```

Output:

```

test1=First test
test3=Third test

```

Examples for the Session Object**Example 1**

```

<%
Session("test1")="First test"
Session("test2")="Second test"
Session("test3")="Third test"

```


	<pre> Session.Contents.Remove("test2") for each x in Session.Contents Response.Write(x & "=" & Session.Contents(x) & "
") next %> Output: test1=First test test3=Third test </pre> <hr/> <p>Example 2</p> <pre> <% Session("test1")="First test" Session("test2")="Second test" Session("test3")="Third test" Session.Contents.Remove(2) for each x in Session.Contents Response.Write(x & "=" & Session.Contents(x) & "
") next %> Output: test1=First test test3=Third test </pre>
<p>Contents.RemoveAll</p>	<p>Deletes all the items from the <i>Application.Contents</i> collection.</p> <p>The Contents.RemoveAll method deletes all items from the Contents collection.</p> <hr/> <p>Syntax</p> <pre> Application.Contents.RemoveAll() Session.Contents.RemoveAll() </pre>

	<table border="1"> <tr> <td data-bbox="632 315 1262 365">Example for the Application Object</td> </tr> <tr> <td data-bbox="632 365 1262 506"> <pre><% Application.Contents.RemoveAll() %></pre> </td> </tr> <tr> <td data-bbox="632 555 1262 604">Example for the Session Object</td> </tr> <tr> <td data-bbox="632 604 1262 745"> <pre><% Session.Contents.RemoveAll() %></pre> </td> </tr> <tr> <td data-bbox="632 768 1262 801"></td> </tr> <tr> <td data-bbox="632 801 1262 835"></td> </tr> </table>	Example for the Application Object	<pre><% Application.Contents.RemoveAll() %></pre>	Example for the Session Object	<pre><% Session.Contents.RemoveAll() %></pre>				
Example for the Application Object									
<pre><% Application.Contents.RemoveAll() %></pre>									
Example for the Session Object									
<pre><% Session.Contents.RemoveAll() %></pre>									
Lock	<p>Locks the application object so that only one user at a time can modify the values.</p> <table border="1"> <tr> <td data-bbox="632 969 1262 1019">Lock Method</td> </tr> <tr> <td data-bbox="632 1019 1262 1189"> <p>The Lock method prevents other users from modifying the variables in the Application object (used to ensure that only one client at a time can modify the Application variables).</p> </td> </tr> <tr> <td data-bbox="632 1238 1262 1288">Unlock Method</td> </tr> <tr> <td data-bbox="632 1288 1262 1458"> <p>The Unlock method enables other users to modify the variables stored in the Application object (after it has been locked using the Lock method).</p> </td> </tr> <tr> <td data-bbox="632 1507 1262 1556">Syntax</td> </tr> <tr> <td data-bbox="632 1556 1262 1637"> <pre>Application.Lock Application.Unlock</pre> </td> </tr> <tr> <td data-bbox="632 1709 1262 1758">Example</td> </tr> <tr> <td data-bbox="632 1758 1262 2007"> <p>The example below uses the Lock method to prevent more than one user from accessing the variable visits at a time, and the Unlock method to unlock the locked object so that the next client can increment the variable visits:</p> </td> </tr> </table>	Lock Method	<p>The Lock method prevents other users from modifying the variables in the Application object (used to ensure that only one client at a time can modify the Application variables).</p>	Unlock Method	<p>The Unlock method enables other users to modify the variables stored in the Application object (after it has been locked using the Lock method).</p>	Syntax	<pre>Application.Lock Application.Unlock</pre>	Example	<p>The example below uses the Lock method to prevent more than one user from accessing the variable visits at a time, and the Unlock method to unlock the locked object so that the next client can increment the variable visits:</p>
Lock Method									
<p>The Lock method prevents other users from modifying the variables in the Application object (used to ensure that only one client at a time can modify the Application variables).</p>									
Unlock Method									
<p>The Unlock method enables other users to modify the variables stored in the Application object (after it has been locked using the Lock method).</p>									
Syntax									
<pre>Application.Lock Application.Unlock</pre>									
Example									
<p>The example below uses the Lock method to prevent more than one user from accessing the variable visits at a time, and the Unlock method to unlock the locked object so that the next client can increment the variable visits:</p>									

	<pre><% Application.Lock Application("visits")=Application("visits")+1 Application.Unlock %> This page has been visited <%=Application("visits")%> times!</pre>
UnLock	Unlocks the application object allowing other users to modify application level variables.
Events	Description
Application_OnEnd	This event occurs when the IIS is shut down after Session_OnEnd event. All the variables are destroyed after that.
Application_OnStart	This event occurs when the first .asp page is called after starting the IIS. Application level variables can be declared here.

<i>Application_OnStart Event</i>
The Application_OnStart event occurs before the first new session is created (when the Application object is first referenced).
This event is placed in the Global.asa file.
Note: Referencing to a Session, Request, or Response objects in the Application_OnStart event script will cause an error.
<i>Application_OnEnd Event</i>
The Application_OnEnd event occurs when the application ends (when the web server stops).
This event is placed in the Global.asa file.
Note: The MapPath method cannot be used in the Application_OnEnd code.
<i>Syntax</i>
<pre><script language="vbscript" runat="server"> Sub Application_OnStart . . . End Sub Sub Application_OnEnd . . . End Sub </script></pre>
<i>Examples</i>
Global.asa:
<pre><script language="vbscript" runat="server"> Sub Application_OnEnd() Application("totvisitors")=Application("visitors") End Sub Sub Application_OnStart Application("visitors")=0 End Sub</pre>

<pre> Sub Session_OnStart Application.Lock Application("visitors")=Application("visitors")+1 Application.Unlock End Sub Sub Session_OnEnd Application.Lock Application("visitors")=Application("visitors")-1 Application.Unlock End Sub </script> </pre>
To display the number of current visitors in an ASP file:
<pre> <html> <head> </head> <body> <p> There are <%response.write(Application("visitors"))%> online now! </p> </body> </html> </pre>

10.2.2 Session Object

- i. When you are working with an application on your computer, you open it, do some changes and then you close it.
- ii. This is much like a Session. The computer knows who you are. It knows when you open the application and when you close it. However, on the internet there is one problem: the web server does not know who you are and what you do, because the HTTP address doesn't maintain state.
- iii. ASP solves this problem by creating a unique cookie for each user.

- iv. The cookie is sent to the user's computer and it contains information that identifies the user.
- v. This interface is called the Session object.
- vi. The Session object stores information about, or change settings for a user session.
- vii. Variables stored in a Session object hold information about one single user, and are available to all pages in one application.
- viii. Common information stored in session variables are name, id, and preferences.
- ix. The server creates a new Session object for each new user, and destroys the Session object when the session expires.
- x. The Session object's collections, properties, methods, and events are described below:

10.2.2.1 Collections

Syntax: Session.collection|property|method

A.1 Collections	Description
Contents	Contains all the items which have been added to the <i>Session</i> object. You can iterate through the <i>Contents</i> collection and retrieve a list of all the items added or you can retrieve a specific item. Note that it contains all the <i>Session</i> items except the ones created using <i><object></i> element.
StaticObjects	Contains all the items which have <i>Session</i> scope created using <i><object></i> element. As with <i>Session.Contents</i> collection you can iterate through the <i>StaticObjects</i> collection to get a list of items or you can get a specific item out of the <i>StaticObjects</i> collection.

A.2 Properties	Description				
CodePage	<p>An <i>integer</i> which defines the code page to be used to display content to the client browser e.g. code page 1252 is used to for American english and most European languages and 932 is used for Japanese kanji.</p> <p>Example of some code pages:</p> <ul style="list-style-type: none"> • 1252 - American English and most European languages • 932 - Japanese Kanji <p>Syntax</p> <pre>Session.CodePage(=Codepage)</pre> <table border="1" data-bbox="507 949 1353 1128"> <thead> <tr> <th>Parameter</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>codepage</td> <td>Defines a code page (character set) for the system running the script engine</td> </tr> </tbody> </table> <p>Examples</p> <pre><% Response.Write(Session.CodePage) %></pre> <p>Output:</p> <p>1252</p>	Parameter	Description	codepage	Defines a code page (character set) for the system running the script engine
Parameter	Description				
codepage	Defines a code page (character set) for the system running the script engine				
LCID	<p>Stands for locale identifier. It is a standard international abbreviation that uniquely identifies the locale e.g. LCID 2057 stands for British locale.</p> <p>Syntax</p> <pre>Session.LCID(=LCID)</pre> <table border="1" data-bbox="507 1792 1353 1928"> <thead> <tr> <th>Parameter</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>LCID</td> <td>A locale identifier</td> </tr> </tbody> </table>	Parameter	Description	LCID	A locale identifier
Parameter	Description				
LCID	A locale identifier				

Examples

```
<%
response.write("<p>")
response.write("Default LCID is: " & Session.LCID & "<br />")
response.write("Date format is: " & date() & "<br />")
response.write("Currency format is: " &
FormatCurrency(350))
response.write("</p>")
```

Session.LCID=1036

```
response.write("<p>")
response.write("LCID is now: " & Session.LCID & "<br />")
response.write("Date format is: " & date() & "<br />")
response.write("Currency format is: " &
FormatCurrency(350))
response.write("</p>")
```

Session.LCID=3079

```
response.write("<p>")
response.write("LCID is now: " & Session.LCID & "<br />")
response.write("Date format is: " & date() & "<br />")
response.write("Currency format is: " &
FormatCurrency(350))
response.write("</p>")
```

Session.LCID=2057

```
response.write("<p>")
response.write("LCID is now: " & Session.LCID & "<br />")
response.write("Date format is: " & date() & "<br />")
response.write("Currency format is: " &
FormatCurrency(350))
response.write("</p>")
%>
```

Output:

```
Default LCID is: 2048
Date format is: 12/11/2001
Currency format is: $350.00
```


	<p>LCID is now: 1036 Date format is: 11/12/2001 Currency format is: 350,00 F LCID is now: 3079 Date format is: 11.12.2001 Currency format is: öS 350,00 LCID is now: 2057 Date format is: 11/12/2001 Currency format is: £350.00</p>
SessionID	<p>A <i>long</i> which returns the session identifier for this client browser.</p> <p>The SessionID property returns a unique id for each user. The unique id is generated by the server.</p> <p>Syntax</p> <pre>Session.SessionID</pre> <p>Examples</p> <pre><% Response.Write(Session.SessionID) %></pre> <p>Output: 772766038</p>
Timeout	<p>An <i>integer</i> which specifies a time out period in minutes. If the client doesn't refresh or request any page of your site within this time out period then the server ends the current session. If you do not specify any time out period then by default time out period is 20 minutes.</p> <p>The Timeout property sets or returns the timeout period for the Session object for this application, in minutes. If the user does not refresh or request a page within the timeout period, the session will end.</p>

Syntax	
Session.Timeout[=nMinutes]	
Parameter	Description
nMinutes	The number of minutes a session can remain idle before the server terminates it. Default is 20 minutes
Examples	
<pre><% response.write("<p>") response.write("Default Timeout is: " & Session.Timeout) response.write("</p>") Session.Timeout=30 response.write("<p>") response.write("Timeout is now: " & Session.Timeout) response.write("</p>") %></pre> <p>Output:</p> <p>Default Timeout is: 20</p> <p>Timeout is now: 30</p>	

10.2.3.3 Methods

Method	Description
<u>Abandon</u>	<p>Destroys the current session object and releases its resources, meaning there by that if the client requests any page of your site after <i>Session.Abandon</i> method has been called then a separate session will be started.</p> <p>Note: When this method is called, the current Session object is not deleted until all of the script on the current page have</p>

	<p>been processed. This means that it is possible to access session variables on the same page as the call to <code>Abandon</code>, but not from another Web page.</p> <p>Syntax</p> <pre>Session.Abandon</pre> <p>Examples</p> <p>File1.asp:</p> <pre><% Session("name")="Hege" Session.Abandon Response.Write(Session("name")) %></pre> <p>Output:</p> <p>Hege</p> <p>File2.asp:</p> <pre><% Response.Write(Session("name")) %></pre> <p>Output:</p> <p>(none)</p>
<u>Contents.Remove</u>	Deletes the given item from the <i>Session.Contents</i> collection.
<u>Contents.RemoveAll()</u>	Destroys all the items in the <i>Session.Contents</i> collection.

10.2.3.4 Events

Event	Description
<u>Session_OnEnd</u>	This event occurs when the session is abandoned or times out for a specific user.

<u>Session_OnStart</u>	Occurs when a new session is started. All the ASP objects are available for you to use. You can define your session wide variables here.
------------------------	--

10.3 SERVER OBJECT

The ASP Server object is used to access properties and methods on the server. Its properties and methods are described below:

Syntax

Server.property|method

Properties	Description				
ScriptTimeout	<p>An <i>integer</i> which specifies time in seconds until which the script can run after that the server aborts the script and displays an <u>error message</u>.</p> <p>The ScriptTimeout property sets or returns the maximum number of seconds a script can run before it is terminated.</p> <p>Syntax</p> <pre>Server.ScriptTimeout[=NumSeconds]</pre> <table border="1"> <thead> <tr> <th>Parameter</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>NumSeconds</td> <td>The maximum number of seconds a script can run before the server terminates it. Default is 90 seconds</td> </tr> </tbody> </table> <p>Examples</p> <p>Example 1</p> <p>Set the script timeout:</p> <pre><% Server.ScriptTimeout=200 %></pre>	Parameter	Description	NumSeconds	The maximum number of seconds a script can run before the server terminates it. Default is 90 seconds
Parameter	Description				
NumSeconds	The maximum number of seconds a script can run before the server terminates it. Default is 90 seconds				

	<p>Example 2 Retrieve the current value of the ScriptTimeout property:</p> <pre data-bbox="568 398 1254 546"><% response.write(Server.ScriptTimeout) %></pre>						
<p>Methods</p>	<p>Description</p>						
<p>CreateObject</p>	<p>Creates an instance of the object (a component, application or a scripting object). The component can be instantiated by giving its <i>CLSID</i> or <i>ProgID</i> in the <i>Server.CreateObject</i> method e.g. <i>Server.CreateObject ("MSWC.MyInfo")</i>.</p> <p>Note: Objects created with this method have page scope. They are destroyed when the server are finished processing the current ASP page. To create an object with session or application scope, you can either use the <object> tag in the Global.asa file, or store the object in a session or application variable.</p> <p>Syntax</p> <table border="1" data-bbox="568 1285 1254 1509"> <tr> <td colspan="2" data-bbox="568 1285 1254 1348">Server.CreateObject(progID)</td> </tr> <tr> <th data-bbox="568 1348 756 1406">Part</th> <th data-bbox="756 1348 1254 1406">Description</th> </tr> <tr> <td data-bbox="568 1406 756 1509">progID</td> <td data-bbox="756 1406 1254 1509">Required. The type of object to create</td> </tr> </table> <p>Example 1 This example creates an instance of the server component MSWC.AdRotator:</p> <pre data-bbox="568 1711 1254 1935"><% Set adrot=Server.CreateObject("MSWC.AdRotator") %></pre>	Server.CreateObject(progID)		Part	Description	progID	Required. The type of object to create
Server.CreateObject(progID)							
Part	Description						
progID	Required. The type of object to create						

	<p>Example 2</p> <p>An object stored in a session variable is destroyed when the session ends. However, you can also destroy the object by setting the variable to Nothing or to a new value:</p> <pre data-bbox="568 495 1254 840"> <% Session("ad")=Nothing %> or <% Session("ad")="a new value" %> </pre> <p>Example 3</p> <p>You cannot create an instance of an object with the same name as a built-in object:</p> <pre data-bbox="568 1037 1254 1218"> <% Set Application=Server.CreateObject("Application") %> </pre>						
<p>Execute</p>	<p>Executes the given .asp page and then returns the control to the page who called the method. Very useful method which can be used to execute another .asp page without leaving the current page and then control is passed back to the calling page.</p> <p>After executing the called .asp file, the control is returned to the original .asp file.</p> <p>Syntax</p> <table border="1" data-bbox="568 1673 1254 1933"> <tr> <td colspan="2">Server.Execute(path)</td> </tr> <tr> <th>Parameter</th> <th>Description</th> </tr> <tr> <td>path</td> <td>Required. The location of the ASP file to execute</td> </tr> </table>	Server.Execute(path)		Parameter	Description	path	Required. The location of the ASP file to execute
Server.Execute(path)							
Parameter	Description						
path	Required. The location of the ASP file to execute						

	<p>Example</p> <pre>File1.asp: <% response.write("I am in File 1!
") Server.Execute("file2.asp") response.write("I am back in File 1!") %></pre> <p>File2.asp:</p> <pre><% response.write("I am in File 2!
") %></pre> <p>Output: I am in File 1! I am in File 2! I am back in File 1!</p> <p>Also look at the Server.Transfer method to see the difference between the Server.Execute and Server.Transfer methods.</p>
GetLastError	<p>Returns an <i>ASPErr</i> object which can be used to get information about the last error occurred in the ASP script.</p> <p>The GetLastError method returns an <i>ASPErr</i> object that describes the error condition that occurred.</p> <p>By default, a Web site uses the file <code>\iishelp\common\500-100.asp</code> for processing ASP errors. You can either use this file, or create your own. If you want to change the ASP file for processing the 500;100 custom errors you can use the IIS snap-in.</p> <p>Note: A 500;100 custom error will be generated if IIS encounters an error while processing either an ASP file or the application's Global.asa file.</p> <p>Note: This method is available only before the ASP file has sent any content to the browser.</p> <p>Syntax</p> <pre>Server.GetLastError()</pre>

	<p>Examples</p> <p>Example 1</p> <p>In the example an error will occur when IIS tries to include the file, because the include statement is missing the file parameter:</p> <pre data-bbox="568 510 1254 689"> <!--#includef="header.inc"--> <% response.write("sometext") %></pre> <p>Example 2</p> <p>In this example an error will occur when compiling the script, because the "next" keyword is missing:</p> <pre data-bbox="568 913 1254 1178"> <% dim i for i=1 to 10 nxt %></pre> <p>Example 3</p> <p>In this example an error will occur because the script attempts to divide by 0:</p> <pre data-bbox="568 1361 1254 1823"> <% dim i,tot,j i=0 tot=0 j=0 for i=1 to 10 tot=tot+1 next tot=tot/j %></pre>
HTMLEncode	Provides HTML encoding to a given string. All non-legal HTML characters are converted to their equivalent HTML entity e.g. "<" is converted to < .

	<p>Syntax</p> <table border="1" data-bbox="568 322 1252 506"> <tr> <th colspan="2" data-bbox="568 322 1252 383">Server.HtmlEncode(string)</th> </tr> <tr> <th data-bbox="568 383 762 443">Parameter</th> <th data-bbox="762 383 1252 443">Description</th> </tr> <tr> <td data-bbox="568 443 762 506">string</td> <td data-bbox="762 443 1252 506">Required. The string to encode</td> </tr> </table> <p>Example</p> <div data-bbox="568 613 1252 1084" style="background-color: #e0e0e0; padding: 5px;"> <p>The following script:</p> <pre data-bbox="580 703 1240 860"><% response.write(Server.HtmlEncode("The image tag: ")) %></pre> <p>Output: The image tag: &lt;img&gt; Web browser output: The image tag: </p> </div>	Server.HtmlEncode(string)		Parameter	Description	string	Required. The string to encode
Server.HtmlEncode(string)							
Parameter	Description						
string	Required. The string to encode						
MapPath	<p>Maps the specified virtual or relative path into physical path of the server.</p> <p>Note: This method cannot be used in Session.OnEnd and Application.OnEnd.</p> <p>Syntax</p> <table border="1" data-bbox="568 1357 1252 1868"> <tr> <th colspan="2" data-bbox="568 1357 1252 1417">Server.MapPath(path)</th> </tr> <tr> <th data-bbox="568 1417 719 1518">Parameter</th> <th data-bbox="719 1417 1252 1518">Description</th> </tr> <tr> <td data-bbox="568 1518 719 1868">path</td> <td data-bbox="719 1518 1252 1868">Required. A relative or virtual path to map to a physical path. If this parameter starts with / or \, it returns a path as if this parameter is a full virtual path. If this parameter doesn't start with / or \, it returns a path relative to the directory of the .asp file being processed</td> </tr> </table>	Server.MapPath(path)		Parameter	Description	path	Required. A relative or virtual path to map to a physical path. If this parameter starts with / or \, it returns a path as if this parameter is a full virtual path. If this parameter doesn't start with / or \, it returns a path relative to the directory of the .asp file being processed
Server.MapPath(path)							
Parameter	Description						
path	Required. A relative or virtual path to map to a physical path. If this parameter starts with / or \, it returns a path as if this parameter is a full virtual path. If this parameter doesn't start with / or \, it returns a path relative to the directory of the .asp file being processed						

	<p>Examples</p> <p>Example 1</p> <p>For the example below, the file test.asp is located in C:\inetpub\Wwwroot\Script.</p> <p>The file Test.asp (located in C:\inetpub\Wwwroot\ Script) contains the following code:</p> <pre data-bbox="566 600 1254 1279"> <% response.write(Server.MapPath("test.asp") & "
") response.write(Server.MapPath("script/test.asp") & "
") response.write(Server.MapPath("/script/test.asp") & "
") response.write(Server.MapPath("\script") & "
") response.write(Server.MapPath("/") & "
") response.write(Server.MapPath("") & "
") %> Output: c:\inetpub\wwwroot\script\test.asp c:\inetpub\wwwroot\script\script\test.asp c:\inetpub\wwwroot\script\test.asp c:\inetpub\wwwroot\script c:\inetpub\wwwroot c:\inetpub\wwwroot </pre> <p>Example 2</p> <p>How to use a relative path to return the relative physical path to the page that is being viewed in the browser:</p> <pre data-bbox="566 1518 1254 1823"> <% response.write(Server.MapPath("../")) %> or <% response.write(Server.MapPath("../\")) %> </pre>
Transfer	<p>Transfers the control of the page to another page specified in the URL. Note that unlike <i>Execute</i>, control of the page is not returned to the page calling the <i>Server.Transfer</i> method.</p>

	<p>Syntax</p> <table border="1" data-bbox="568 322 1251 629"> <tr> <th colspan="2" data-bbox="568 322 1251 383">Server.Transfer(path)</th> </tr> <tr> <th data-bbox="568 383 703 483">Parameter</th> <th data-bbox="703 383 1251 483">Description</th> </tr> <tr> <td data-bbox="568 483 703 629">path</td> <td data-bbox="703 483 1251 629">Required. The location of the ASP file to which control should be transferred</td> </tr> </table> <p>Example</p> <div data-bbox="568 734 1251 1532" style="border: 1px solid black; padding: 5px;"> <p>File1.asp:</p> <pre data-bbox="579 831 1117 1025"><% response.write("Line 1 in File 1
") Server.Transfer("file2.asp") response.write("Line 2 in File 1
") %></pre> <p>File2.asp:</p> <pre data-bbox="579 1160 1117 1317"><% response.write("Line 1 in File 2
") response.write("Line 2 in File 2
") %></pre> <p>Output:</p> <pre data-bbox="579 1406 790 1518">Line 1 in File 1 Line 1 in File 2 Line 2 in File 2</pre> </div> <p>Also look at the <code>Server.Execute</code> method to see the difference between the <code>Server.Transfer</code> and <code>Server.Execute</code> methods.</p>	Server.Transfer(path)		Parameter	Description	path	Required. The location of the ASP file to which control should be transferred
Server.Transfer(path)							
Parameter	Description						
path	Required. The location of the ASP file to which control should be transferred						
URLEncode	<p>Provides URL encoding to a given string e.g. <code>Server.URLEncode ("http://www.stardeveloper.com")</code> returns <code>http%3A%2F%2Fwww%2Estardeveloper%2Ecom</code>.</p> <p>The <code>URLEncode</code> method applies URL encoding rules to a specified string.</p>						

Syntax	
Server.URLEncode(string)	
Parameter	Description
string	Required. The string to encode
Example	
<pre><% response.write(Server.URLEncode("http://www.w3schools.com")) %></pre> <p>Output:</p> <pre>http%3A%2F%2Fwww%2Ew3schools%2Ecom</pre>	

Question:

1. What are objects of ASP.
2. Explain different type of onjects of ASP?

Or

3. Short Note on
 - a. Request Object
 - b. Response Object
 - c. Session Object
 - d. Application object
 - e. Server Object



ASP FORMS

Unit Structure

11.1 ASP Forms

11.1 WHAT ARE FORMS?

1. Recall from Day 1, "Getting Started with Active Server Pages," that the Internet is based upon the client-server model.
2. When you visit a Web page, your browser, the client, makes a request to the Web server, asking for a particular Web page.
3. The Web server responds by sending the requested document to the client. When requesting an ASP page, the web server first processes the ASP code *before* sending the resulting web page back to the client.
4. What, though, if we want our ASP page to make decisions based upon a user's input?
5. To accomplish this, we need to use forms.
6. A *form* has two duties: to collect information from the user and to send that information to a separate Web page for processing.
7. Through the use of a form, an ASP page can acquire the user's input, and make programmatic decisions based on that input.
8. Forms also allow for the user to enter detailed information using a variety of input controls, such as text boxes, list boxes, check boxes, and radio buttons.

11.1.1 Creating Forms

Creating a form is straightforward and simple. It requires as little as two lines of HTML, as shown in Listing 11.1.

Example 11.1. A Form is Created Using the <FORM> Tag

```
1: <FORM METHOD=POST ACTION="somePage.asp">
2: </FORM>
```

Listing 11.1 uses the HTML tag <FORM> to create a simple form. The <FORM> tag has two properties: METHOD and ACTION.

- **METHOD**—The METHOD tag can be set to either GET or POST.
- **ACTION**—The ACTION tag specifies what page will be called when the form is submitted. Usually, this is an ASP page that will process the information entered by the user.

- A form is *submitted* when the user confirms that he is finished entering the information, usually by clicking a button.
- If there is just one input field in the form, such as a text box, the user can simply press Enter to submit the form.

Using Form Fields

The form in Listing 11.1 serves no function. It has no text boxes for users to enter information into. It has no list boxes, radio buttons, or check boxes either. On a Web site, this form would be useless; however, it does demonstrate how a <FORM> tag is used. To be useful, a form must contain one or more *form fields*: objects inside a form that are used to collect information from the user. Each text box, list box, check box, or radio button in a form is a form field. You need a way to create form fields within your form.

To create text boxes, check boxes, and radio buttons, use the <INPUT> tag. The <INPUT> tag has a number of properties, but we will only concentrate on the following three:

- **NAME**—The NAME tag uniquely identifies each element in the form. You will use the NAME tag in tomorrow's lesson when you use ASP to process the user's input.
- **TYPE**—The TYPE tag determines what type of form field is displayed. To display a text box, set TYPE equal to TEXT. To create a check box, assign TYPE equal to CHECKBOX.

- **VALUE**—The **VALUE** tag determines the default value for the form field. This property is important when processing the information submitted by list boxes, check boxes, and radio buttons.

To create list boxes, use the `<SELECT>` tag in conjunction with the `<OPTION>` tag. Each option in the list box needs an `<OPTION>` tag. The `<SELECT>` tag is only used once, encompassing many `<OPTION>` tags. Let's say that you want a list box that lists the months of the year. You would need 12 `<OPTION>` tags enclosed by a `<SELECT>` tag, like so:

```
<SELECT NAME=Month>
<OPTION VALUE="January">January
<OPTION VALUE="February">February
<OPTION VALUE="March">March
...
<OPTION VALUE="November">November
<OPTION VALUE="December">December
</SELECT>
```

The **VALUE** property in the `<OPTION>` tags serves as a unique identifier for each separate option in the list box. We will discuss list boxes in "List Boxes."

Putting it All Together

Now that you know how to create forms and form fields, let's create a form that asks for the user's name, age, and sex. This form would need a number of form fields. First, we would need a text box for the user's name. We could also use a text box for the user's age; however, if we were only interested in what age group our user fit in, we could use a list box.

Finally, to obtain the user's sex, we will use two radio buttons, one labeled Male and the other labeled Female. Listing 11.2 contains the HTML code that will generate these form fields.

Example 11.2. A Form to Collect Generic User Information

```
1: <FORM METHOD=POST ACTION="somePage.asp">
2: What is your name?
3: <INPUT TYPE=TEXT NAME=Name>
```

```

4: <P>
5:
6: How old are you?
7: <SELECT NAME=Age>
8: <OPTION VALUE="Under 21">Under 21
9: <OPTION VALUE="21 - 50">21 – 50
10: <OPTION VALUE="Over 50">Over 50
11: </SELECT>
12: <P>
13:
14: Sex:<BR>
15: <INPUT TYPE=RADIO NAME=Sex VALUE=Male>
16: Male
17: <BR>
18: <INPUT TYPE=RADIO NAME=Sex VALUE=Female>
19: Female
20: <BR>
21:
22: <INPUT TYPE=SUBMIT VALUE="Send us your Information!">
23: </FORM>

```

The code in Listing 11.2 creates a form that contains a number of form fields. These form fields are used to collect information from the user. Each form field is created using either the `<INPUT>` tag, for text boxes, check boxes, and radio buttons, or the `<SELECT>` and `<OPTION>` tags, for list boxes.

For example, our users will be presented with a text box to enter their names into. This text box was created using the `<INPUT>` tag (line 3) with its `TYPE` property set to `TEXT`. The list box that contains the various age ranges is created on line 7 with the `<SELECT>` tag. Each option for the list box is created using the `<OPTION>` tag (lines 8, 9, and 10). Finally, the two radio buttons are created on lines 15 and 18. These are both created using the `<INPUT>` tag with the `TYPE` property set to `RADIO`.

Line 22 in Listing 11.2 also demonstrates the use of a valid submit button. All forms you create should contain a submit button. A *submit button*, when clicked, submits the form. To create a submit button, you use the `<INPUT>` tag with `TYPE` set equal to the keyword `SUBMIT`. The `VALUE` tag determines the submit button's label. If you do not include a `VALUE` for your submit button, the

browser will decide what to label the submit button. (Internet Explorer 5.0, for example, will label the submit button as Submit Query.)

Submitting Forms

Using a standard Web browser, a user can surf to a Web page with a form on it and enter information. When the user does this, the information he is typing in has not yet been sent to the Web server. This information is not available for the Web server to process until the user submits the form by clicking the form's submit button.

It would be nice to be able to send this information to an ASP page, which could then determine what the user entered into the form and act on that information. The `<FORM>` tag offers two properties that allow you to send form information to an ASP page for processing: the `ACTION` property and the `METHOD` property.

Using the ACTION Property

The `ACTION` property of a form can be set to any valid URL. When a user submits the form, the URL specified in the `ACTION` property is called, and the values in the form fields are passed. In Listing 11.3, the form's `ACTION` property is set to `catalog.asp`. When the user clicks the submit button, the form field values are sent to `catalog.asp` as the user's browser is redirected to `catalog.asp`.

NOTE

The `ACTION` property does not have to be set to an ASP page. The `ACTION` property can be set to any Web page name on your Web server (such as a CGI script) or to a script on another server altogether, or it could be left out completely.

Note that if you do not specify the `ACTION` property in a form, when a user submits the form, the current page is reloaded.

In the examples in this book, we will *always* specify the `ACTION` property in our forms. Also, because this book deals with ASP, the `ACTION` property will always be set to an ASP page.

The second property of the `<FORM>` tag is called `METHOD` and can be set to either `GET` or `POST`. The `METHOD`

determines how the form field values are passed to the ASP page specified in the form's ACTION property.

The Difference Between GET and POST

There are, not surprisingly, two ways through which information can be passed from a form to an ASP page. The first method uses the querystring and is the method used when a form's METHOD property is set to GET. The other method, POST, hides the user's information by not using the querystring.

The *querystring* is additional information sent to a Web page appended to the end of the URL.

The querystring is made up of name/value pairs, in the following form:

VariableName=ValueOfVariable

For example, if a URL were to appear as

`http://www.yourserver.com/someFile.asp?name=Scott`

the querystring would be

`?name=scott`

Note that the start of the querystring is denoted by a question mark (?).

The querystring can contain multiple name/value pairs. When more than one name/value pairs is in the querystring, each name/value pair is separated by an ampersand (&). For example, if both the name and age were stored in the querystring, the querystring might look like this:

`?name=scott&age=21`

Remember that the querystring is always appended to the URL, so in your browser's address pane, the full URL of the page would appear as follows:

`http://www.yourserver.com/someFile.asp?name=Scott&age=21`

CAUTION

If you have a vast number of form fields in your form, it quickly becomes apparent that the querystring will become very long! Strive to keep the length of the total URL fewer than 255 characters. This 255 character limitation was a shortcoming of older browsers. Today's web browsers do not have this limitation; however, you have no guarantee that all of your Web site's visitors will be using up to date browsers, and therefore you should strive to keep the length of the total URL fewer than 255 characters. That

means if you are going to have a vast number of form fields, it might be wise to set the form's METHOD to POST. We'll discuss how POST differs from GET shortly.

In Listing 11.4, you'll find a simple form that has its METHOD property set to GET. This form doesn't really do anything complex; it simply demonstrates how form values can be passed through the querystring.

Example 11.4. Creating a Form Where METHOD=GET

```
1: <FORM METHOD=GET ACTION="GetMethodExample.asp">
2: Name: <INPUT TYPE=TEXT NAME=Name>
3: <BR>
4: Age: <INPUT TYPE=TEXT NAME=Age>
5: <BR>
6: <INPUT TYPE=SUBMIT>
7: </FORM>
```

For this example, the ASP file `GetMethodExample.asp` does nothing useful; it simply prints a message about the querystring. The complete code for `GetMethodExample.asp` is shown in Listing 11.5. Note the Address bar in Figure 11.3, which shows how the querystring is appended to the URL and contains two name/value pairs:
`?Name=Scott&Age=21`

Figure 11.3. The form field values are stored as name/value pairs in the querystring.



In Listing 11.4, we examined how to create a form using the GET method. When this form is submitted, the ASP page specified by the form's ACTION property is called, and values the user entered are passed to this page. How they are passed depends upon the form's METHOD property. In Listing 11.4, in line 1, the form's METHOD is set to GET. When the form is submitted, the user's inputs are passed through the querystring to GetMethodExample.asp, and the output is shown in Figure 11.3.

Let's take a moment to examine the code for GetMethodExample.asp. Note that it outputs the contents of the querystring. Listing 11.5 shows the code for GetMethodExample.asp:

Example 11.5. Outputting the Contents of the querystring

- 1: Note the A<U>d</U>ress bar above.
- 2: <P>
- 3: The querystring is set to:

- 4: <%=Request.QueryString%>

Listing 11.5, the code for GetMethodExample.asp, is fairly straightforward. The only non-HTML code used is on line 4, where the contents of the Request.QueryString collection are outputted. The Request.QueryString is discussed in detail in tomorrow's lesson, "Collecting the Form Information." As you'll learn tomorrow, the Request.QueryString collection contains the information passed in the querystring. There are three ways to pass information to an ASP page through the querystring. You've already seen the first way, which is to set a form's METHOD property as GET. Another way is to use the HREF tag, which creates a link in your Web page. If you wanted to allow a user to click a link that would take him directly to someFile.asp, passing in a querystring of ?Name=Scott&age=21 you would simply need to create an HREF tag:<AHREF="someFile.asp?Name=Scott&age=21">Click Me!

This method proves useful when you want to provide a quick link that passes some predetermined information to a Web page. The third and final way to pass information to an ASP page through the querystring is to simply type in the full URL with the querystring into your browser. For example, you could enter <http://www.yourserver.com/someFile.asp?Name=Scott&age=21>

into your browser's Address bar, which would be equivalent to using either of the other two methods.

CAUTION

Although passing form information through the querystring might seem harmless, imagine if you have a form where you want the user to enter his password or some other type of sensitive information. When the information is passed through the querystring, this sensitive information appears on the screen. Furthermore, the browser might save the full URL in the Histories folder, which could be a security threat if multiple individuals used the computer.

If you are going to ask for private information, it is best *not* to set METHOD to GET.

When the METHOD property is set to POST, the information being passed is hidden, and, therefore, setting METHOD=POST is preferred when collecting sensitive information from your users. One disadvantage with using GET is that the form field values are exposed through the Address bar. Also, some older clients do not support URLs longer than 255 characters, which means that you should always make sure that the Web page's URL plus the querystring is less than 255 characters. You can hide these variables being passed and get rid of the 255 character limitation by simply setting the METHOD property to POST instead of GET. When using POST, the data is still passed in name/value pairs to the ASP page specified by the form's ACTION property. Instead of being appended to the URL, however, the name/value pairs are hidden from the Address bar. Therefore, if you plan to ask your users to enter private information, it is best to use POST.

When using a form whose METHOD is set to POST, the form field values are sent through the HTTP headers. These headers are bits of information that the web browser sends to the web server when requesting a Web page.

Reading Form Values from an ASP Page

Now that you know how to send form field values to an ASP page, you may find yourself wondering how to read these form field values in your ASP pages. Day 9 is dedicated to this discussion, but I think it only fair that I give you a sneak peek today!

You read form field values by using the Request object. The Request object contains two *collections* used to read form data:

- QueryString collection—The QueryString collection is used to access the name/value pairs passed through the querystring.
- Form collection—The Form collection is used to access the name/value pairs passed by a form that has its METHOD property set to POST.

A *collection* is a set of name/value pairs, similar to a two-column matrix. Suppose that you wanted to list the months of the year and how many days were in each month. Table 11.1 shows such a two-column matrix. In a collection, you refer to each row by the left column of the matrix, which gives you the value, or the right column of the matrix. For example, say that you named your collection DaysInMonths. If you requested the value of DaysInMonth("August") you would get the value 31.

Table 11.1. DaysInMonths Collection Illustrated as a Matrix
Name Value

... ..
June 30
July 31
August 31
September 30
... ..

We will discuss collections in more detail during Day 9. For now, just remember that a collection is a set of name/value pairs. To refer to a particular value, you need to know its name and refer to it as:

NameOfCollection(Name)

Because you want to read the values from a form in an ASP page, you will need two Web pages: one page that contains the HTML code for the form and another page, an ASP page, that processes the information entered into the form.

Let's start by creating the page that will have the form on it. For this example, we'll create a form that asks for the user's name and date of birth. We will name this page BirthdateForm.htm.

Create this HTML page and enter the code in Listing 11.6. You can see the output of BirthdateForm.htm in Figure 11.4.

Example 11.6. The HTML Code for BirthdateForm.htm

```
1: <HTML>
2: <BODY>
3: <B>Please enter your name and birthdate:</B><BR>
4:
5: <FORM METHOD=POST ACTION="YourAge.asp">
6: Your name: <INPUT TYPE=TEXT NAME=Name>
7: <BR>
8:
9: Your Birthdate (in MM/DD/YY format):<BR>
10: <INPUT TYPE=TEXT NAME=Birthdate>
11:
12: <P>
13: <INPUT TYPE=SUBMIT VALUE="Send!">
14: </FORM>
15: </BODY>
16: </HTML>
```

The form created in Listing 11.6 is fairly straightforward. The form consists of two form fields: a text box for the user to enter his name (line 6) and a text box for the user to enter his date of birth (line 10). In line 5, we set the form's ACTION property to YourAge.asp. This is the ASP page that will be called when the user submits the form.

You need to create YourAge.asp in the same directory as BirthdateForm.htm. The code for YourAge.asp can be found in Listing 11.7. Although some of the ASP code present in YourAge.asp might be new, we will cover it in detail during Day 9. The output for YourAge.asp can be seen in Figure 11.5.

It is important to understand how the form's ACTION property works. If you specify a simple filename, such as ACTION="someFile.asp", it is important that someFile.asp exists in the same directory as the page that displays the form. In our example with BirthdateForm.htm, we set the form's ACTION property as ACTION="YourAge.asp" (line 5). This requires that both YourAge.asp and BirthdateFile.htm exist in the same directory. If YourAge.asp does not exist in the same directory as

BirthdateForm.htm, when the user submits the form she will receive a 404 Error. A 404 Error occurs when the browser asks the Web server for a Web page that does not exist. Although YourAge.asp may exist, if it does not exist in the same directory as BirthdateForm.htm, a 404 Error will occur.

With a slight modification of the form's ACTION property, you can have these two files exist in different directories, or even on different Web servers. This topic will be discussed in "Revisiting the ACTION property."

Example 11.7. The Code for YourAge.asp

```
1: <%@ Language=VBScript %>
2: <% Option Explicit %>
3: <%
4: 'Read in the form field variables
5: Dim strName, dtBirthDate
6: strName = Request.Form("Name")
7: dtBirthDate = Request.Form("Birthdate")
8:
9: Dim idaysOld
10: idaysOld = DateDiff("d", dtBirthDate, Date)
11: %>
12: <HTML>
13: <BODY>
14:
15: Hello <%=strName%>!
16: You were born on <%=dtBirthDate%>.
17: <BR>
18: That makes you <%=iDaysOld%> days old.
19: </BODY>
20: </HTML>
```

Although the code in Listing 11.7 uses the Request object (lines 6 and 7), which we have not formerly discussed, you should be familiar with the rest of the ASP code. We've used the shorthand notation for Response.Write (<%= ... %>) before (lines 15, 16, and 18), and on Day 5, "Using VBScript's Built-in Functions," we discussed how to use DateDiff (line 10), one of the many powerful date functions available with VBScript. YourAge.asp simply reads in the name and birth date that the user entered into the form in BirthdateForm.htm (lines 6 and 7). It then takes this information and

uses `DateDiff` to calculate the number of days that have passed since the user's date of birth (line 10). Finally, `YourAge.asp` prints out a welcome message with the user's name and birth date, and then displays the number of days since the user's date of birth (lines 15 through 18).

Revisiting the ACTION Property

Up until this point we've specified the `ACTION` property as a simple filename, such as `YourAge.asp` or `GetMethodExample.asp`. Although there is nothing wrong with this, it restricts you to placing your form creation Web pages and form processing scripts in the same directory. What do you do, though, if you want to have your form creation Web page and form processing scripts in different directories? What if you want your form processing script on another Web server altogether? The `ACTION` property can be used to allow for either of these two scenarios.

Assume that you have a directory called `scripts`, which you create in the root directory. In the `scripts` directory, you place all your form processing scripts.

Client-Side Form Validation

In "Reading Form Values from an ASP Page" you saw an example of creating an HTML page to display a form to collect the user's name and birth date. This HTML page was named `BirthdateForm.htm` and is presented in Listing 11.6. We also created an ASP page that read the form field values, calculated the number of days between the user's date of birth and the current date, and displayed this calculation, as well as the user's name and birth date. This ASP page, named `YourAge.asp`, is available for review in Listing 11.6. Suppose that in the Birth date form field, the user entered an ill-formatted date. For the moment, let's assume that the user entered Thirty years ago as the value for the Birth date field. What would happen? What should happen?

The form would submit normally and would pass the "Thirty years ago" as the value of the Birth date field. The ASP page would read in the form field value with no problem but would generate an error on line 10 of Listing 11.6 when trying to use the `DateDiff` function. The `DateDiff` function, which we discussed on Day 5, expects properly formatted dates as its parameters and

generates an error if improperly formatted dates are used. Now that we know what *will* happen, what *should* happen instead? It would be nice to be able to determine whether any form field values were improperly formatted *before* the form was submitted. If any form field values were incorrectly formatted, you would want to alert the user and not allow the user to submit the form until the form field values were properly formatted. Checking the user's form fields for validity *before* the form is submitted is known as *client-side form validation*: the process of using a client-side scripting language, such as JavaScript, to validate form fields.

Client-side form validation can be bewildering for those who do not have a solid programming background. Client-side form validation scripting requires the use of a client-side scripting language. The only client-side scripting language supported by both Internet Explorer and Netscape Navigator is JavaScript, which is similar in syntax to the C programming language.

Using the Different Form Fields

The examples in today's lesson have used all four types of form fields. Although we used the various types of form fields, we skimmed over the details of each form field type. This section covers the more intricate details of each type of form field.

CAUTION

Whenever you want to place a form field within your Web page, make sure that you place it *after* a <FORM> tag and *before* that <FORM> tag's associated closing tag (</FORM>). Netscape Navigator will not display form fields that are not encapsulated by <FORM> ... </FORM>.

There are times when a list box makes more sense than a text box, or when it's easier to use a series of radio buttons as opposed to a series of check boxes. Table 11.2 lists each form field type and discusses what circumstances each form field type is best suited for.

Choosing the Best Form Field Type

Form

Field

Type

When to Use

Text box

When you need to allow the user to enter a string of characters or a number, the text box is the best option. If, however, the input is restricted, such as in the case with a user choosing his state of residency, it may be wiser to use a list box or a series of radio buttons.

List box

If you want to restrict the user to selecting an item among a set of acceptable answers, a list box is usually the best choice, especially when the set of acceptable answers is large.

Check box

Anytime you have multiple, related, Yes/No options that can be mixed and matched in any number of ways, you will want to use a series of check boxes. Also, anytime you have a simple Yes/No type question, such as, "Do you want to receive updates on our products via email?", a check box will do nicely.

Radio button

Whenever you have a set of options that are mutually exclusive—that is, only none or one of the options out of the set of options can be selected, radio buttons are the way to go. Radio buttons can also be used in place of list boxes when the number of unique options is not too great.

Text Boxes

Text boxes can be created using the INPUT tag. Text boxes are not the only form field type that is created via the INPUT tag: check boxes and radio buttons are created similarly. To specify that you want to create a text box and not a check box or radio button, you must set the INPUT's TYPE property to TEXT. Other important properties are the NAME, SIZE, and VALUE properties. The NAME property, common among all form field types, uniquely identifies the form field.

The form processing script also uses the value of the NAME property. Recall that the form processing script interprets the form field values as a set of name/value pairs. That is, to retrieve the information the user entered into a specific form field, the form processing script must know the form field's name. This name is

specified by the NAME property in the INPUT tag in the form creation Web page.

This probably sounds a little confusing. Perhaps an example will clear things up. For this example, we will create a new ASP page that will contain a form. We'll name this ASP page SimpleForm.asp, and set the form's ACTION property to collectInfo.asp. The code for SimpleForm.asp is shown in Listing 11.8. Figure 11.6 displays SimpleForm.asp when viewed through a browser. Note that in Figure 11.6 the user has entered two values into the text boxes.

Example 11.8. SimpleForm.asp Creates a Form with Two Form Fields

```
1: <%@ Language=VBScript %>
2: <% Option Explicit %>
3:
4: <HTML>
5: <BODY>
6: <FORM METHOD=POST ACTION="collectInfo.asp">
7: TextBox1:
8: <INPUT TYPE=TEXT NAME=TextBox1>
9: <BR>
10: TextBox2:
11: <INPUT TYPE=TEXT NAME=TextBox2>
12: <P>
13: <INPUT TYPE=SUBMIT>
14: </FORM>
15: </BODY>
16: </HTML>
```

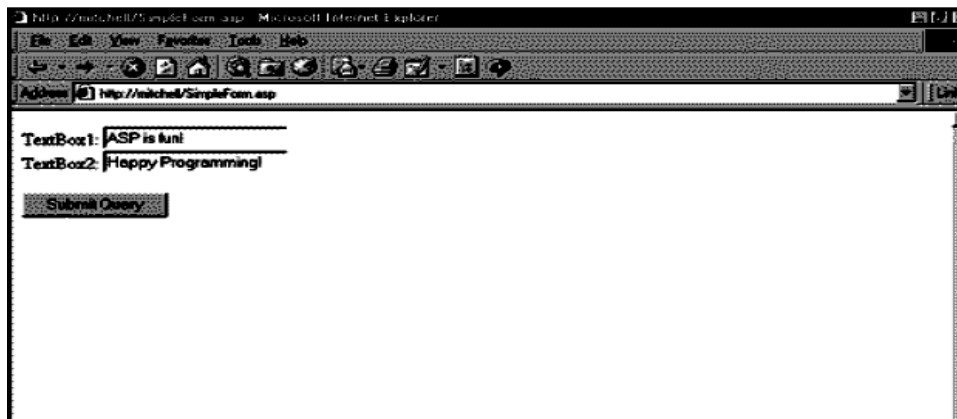
Listing 11.8 demonstrates how to create form fields within a form. We start by creating a form using the <FORM> tag (line 6). Whenever you use the <FORM> tag, you should have a closing form tag as well (line 14). The closing form tag, </FORM>, identifies the end of a form. All of your form fields should be placed between the <FORM> and </FORM> tags.

In Listing 11.8 we create two text boxes on lines 8 and 11. Text boxes are created using the <INPUT> tag with its TYPE property set to TEXT. The NAME property is responsible for uniquely identifying each text box. Finally, we add a submit button

to our form (line 13). The form created in Listing 11.8 contains two text boxes and a submit button; Figure 11.6 displays the output of Listing 11.8 when viewed through a browser.

When this form is submitted, the user will be redirected to the ASP page

Figure 11.6. SimpleForm.asp when viewed through a browser. The user has entered two values for the text boxes.



collectInfo.asp. collectInfo.asp can easily read the form variables by using the Request object. If we wanted to store the user's two text box entries into two separate variables, we could do so with some simple code. The code for collectInfo.asp, shown in Listing 11.9, demonstrates how to use the name of the form field to obtain the form field's value.

Example 11.9. Store the Values of the Form Fields into Two Variables

```

1: <%@ Language=VBScript %>
2: <% Option Explicit %>
3: <%
4: 'Create two variables to hold the values from
5: 'TextBox1 and TextBox2
6: Dim strTextBox1, strTextBox2
7:
8: 'Read in the form field values using the Request object
9: strTextBox1 = Request.Form("TextBox1")
10: strTextBox2 = Request.Form("TextBox2")
11: 'Output the values of strTextBox1 and strTextBox2
12: Response.Write "TextBox1 = " & strTextBox1
13: Response.Write "<BR>TextBox2 = " & strTextBox2
14: %>

```

The values of `strTextBox1` and `strTextBox2` in `collectInfo.asp` depend on what the user entered into the text boxes in `SimpleForm.asp`. These two variables are declared on line 6, and are assigned the values passed in through the `Request.Form` collection on lines 9 and 10. In Figure 11.6, the user entered the value `ASP is fun!` into the first text box (whose `NAME` property is set to `TextBox1`), and `Happy Programming!` into the second (whose `NAME` property is set to `TextBox2`). What are the values of the variables in `collectInfo.asp`? In this example, `strTextBox1` would be equal to `Happy Programming`, and `strTextBox2` would be equal to `Happy Programming!`, exactly the values entered into the form fields.

Besides the `NAME` and `TYPE` properties, there are a couple of other interesting text box properties. The first is the `VALUE` property, which allows you to set the default response in the text box. For example, if you have a text box where you want the user to type in the country she lives in, you might decide to have the text box show `United States` by default. To do so, all you need to do is create the text box and set the `VALUE` property correctly:

```
<INPUT TYPE=TEXT NAME=Country VALUE="United States">
```

Another neat property is the `SIZE` property, which determines how many characters long the text box is. If you create a text box for a user to enter his email address, that text box might need to be 20 characters long. If you want the user to enter a shorter string, perhaps an area code, a text box of length 3 would suffice. If you want to create a small text box to inquire for the user's area code, you might do so with the following code:

```
<INPUT TYPE=TEXT NAME=AreaCode SIZE=3>
```

NOTE

The `SIZE` property does not force the user's input to be less than a certain length.

The `SIZE` property only determines how many characters wide the text box will be.

In Listing 11.10, you'll find the code for an HTML page that creates a form and displays three text boxes. One text box has its `VALUE` property set. The other two text boxes have differing `SIZE` property

values. Figure 11.7 illustrates how the browser renders the user of the VALUE and SIZE properties.

Example 11.10. Working with the VALUE and SIZE Properties

```
1: <HTML>
2: <BODY>
3: <FORM METHOD=GET ACTION="/scripts/someFile.asp">
4: This INPUT box has its VALUE property set to "United
States":
5: <BR>
6: <INPUT TYPE=TEXT NAME=Country VALUE="United States">
7: <P>
8: This INPUT box has its SIZE property set to 3:<BR>
9: <INPUT TYPE=TEXT NAME=AreaCode SIZE=3>
10: <P>
11: This INPUT box has its SIZE property set to 25
12: and its VALUE property set to "Hi, mom!"<BR>
13: <INPUT TYPE=TEXT NAME=Hi VALUE="Hi, mom!" SIZE=25>
14: </FORM>
15: </BODY>
16: </HTML>
```

In Listing 11.10 we create three text boxes. Recall that text boxes are created using the `<INPUT>` tag. In line 6 we create our first text box, setting the `VALUE` property to `United States`. The `VALUE` property, when used with text boxes, indicates the entry a text box will contain by default. When a user visits this page, the first text box will already contain the words `United States`. The text box created on line 13 also makes use of the `VALUE` property.

The text boxes created on lines 9 and 13 demonstrate the use of the `SIZE` property. The `SIZE` property determines how many characters wide the text box will be. If you expect that your users will only enter a few characters into a given text box, it is a good idea to use the `SIZE` property to make the text box smaller. When a user sees a small text box, they instantly realize that the text box is to only contain a few characters. On line 9 we query the visitor for their age. Since it is ludicrous to expect this to be more than three characters long, we create the text box with `SIZE=3`.

NOTE

The order with which you place properties in a form field is unimportant. For example, these two lines are functionally identical:

```
<INPUT NAME=City SIZE=20 VALUE="Chicago" TYPE=TEXT>
```

```
And <INPUT VALUE="Boston" TYPE=TEXT SIZE=20 NAME=City>
```

Check the text boxes' differing sizes and default values.**List Boxes**

There are times when a text box just won't cut it. Perhaps you want to restrict the user to a specific set of choices. For example, if you want users to specify their states of residency, you don't want to use a text box because someone might misspell a state's name or enter 41 as his state of residency. When you need the user to choose a response to a particular set of valid options, it is best to use a list box. Of all the form field types, the list box is the oddball, being the only one that isn't created via the `<INPUT>` tag. Rather, the list box uses two tags, the `<SELECT>` and the `<OPTION>` tags.

The `<SELECT>` tag indicates that a list box will be created, whereas each `<OPTION>` tag represents a unique choice for the list box.

The `<SELECT>` tag has two properties that we will discuss: the `NAME` property and the `SIZE` property. The `NAME` property serves the same purpose as with the text box form field type—it uniquely identifies the particular list box. The `SIZE` property determines how many list box options are shown at one time. The default value for the `SIZE` property is 1, which means that a list box, by default, will show only one option at a time. Figure 8.8 displays a list box with the default `SIZE` property and a list box with a `SIZE` property of 5. The `<OPTION>` tag has two important properties. The first is the `VALUE` property, which uniquely identifies each separate list box option. When the user selects a list box option and submits the form, the form processing script is passed the string in the `VALUE` property of the selected list box item. The `VALUE` property does not determine what is displayed in the list box. Rather, the text that appears *after* the `<OPTION>` tag is displayed in the list box.

Examine the code in Listing 11.11. This is the code that, when viewed through a browser, appears in `OPTION tag>`Figure 11.8.

Example 11.11. Understanding the <OPTION> Tag

```
1: <HTML>
2: <BODY>
3: <FORM METHOD=GET ACTION="/scripts/someFile.asp">
4: This list box has its SIZE property set to the default:
5: <BR>
6: <SELECT NAME=ASPOpinion>
7: <OPTION VALUE="5">I like ASP a lot!
8: <OPTION VALUE="4">ASP sure is neat.
9: <OPTION VALUE="3">It's Interesting.
10: <OPTION VALUE="2">ASP is difficult!
11: <OPTION VALUE="1">Ah!
12: </SELECT>
13: <P>
14: This list box has its SIZE property set to 5:
15: <BR>
16: <SELECT NAME=Experience SIZE=5>
17: <OPTION VALUE="10">10+ Years of ASP Experience
18: <OPTION VALUE="9">9 Years of ASP Experience
19: <OPTION VALUE="8">8 Years of ASP Experience
20: <OPTION VALUE="7">7 Years of ASP Experience
21: <OPTION VALUE="6">6 Years of ASP Experience
22: <OPTION VALUE="5">5 Years of ASP Experience
23: <OPTION VALUE="4">4 Years of ASP Experience
24: <OPTION VALUE="3">3 Years of ASP Experience
25: <OPTION VALUE="2">2 Years of ASP Experience
26: <OPTION VALUE="1">1 Year of ASP Experience
27: <OPTION VALUE="0">Less than a year of ASP Experience
28: </SELECT>
29: </FORM>
30: </BODY>
31: </HTML>
```

If you compare the code in Listing 11.11 to the output shown in Figure 11.8, you'll notice that the text after the <OPTION> tag is displayed in the list box. Look at line 7 in Listing 11.11 and the first list box in Figure 11.8. Notice how the text in the list box reads: I like ASP a lot! This is, not coincidentally, the text that follows after the <OPTION> tag. The text that is displayed and the text that is sent to the form processing script when the form is submitted can be two completely different values. The form processing script is sent the value in the <OPTION> tag's VALUE

property, not what is displayed as the list box's text in the browser. To help you understand this concept, let's look at an example. The following code creates a simple list box that contains two options, Yes and No. The <OPTION> tags that define these two options each have a unique VALUE.

```
<FORM METHOD=POST ACTION="/scripts/processListBox.asp">
<SELECT NAME=YesOrNo SIZE=1>
<OPTION VALUE="YesChoice">Yes
<OPTION VALUE="NoChoice">No
</SELECT>
<P>
<INPUT TYPE=SUBMIT VALUE="Voice your Opinion!">
</FORM>
```

The form processing script, /scripts/processListBox.asp, could determine what list box option was selected with Request.Form("YesOrNo"). YesOrNo is the value of the <SELECT> tag's NAME property and is what the form processing script would use to refer to the list box. The value of Request.Form("YesOrNo") would depend on what list box option the user chooses. If the user chooses Yes, Request.Form("YesOrNo") would be equal to YesChoice. If, on the other hand, the user chooses the option labeled No, Request.Form("YesOrNo") would be equal to NoChoice.

NOTE

When you want to read the value of a list box in the form processing script, you need to refer to it by the NAME property in the <SELECT> tag. The value of the list box is equivalent to the VALUE property of the selected list box option.

One other interesting property for <OPTION> tags deserves mentioning. Notice that in a list box, the option selected by default is the first <OPTION> tag. You can change that by using the SELECTED property of the <OPTION> tag. Just place the word SELECTED within the <OPTION> tag that you want to have selected by default. If you created a list box like the following:

```
<FORM>
<SELECT NAME=DefaultTest>
<OPTION VALUE="1">1GuyFromRolla
<OPTION VALUE="2">2GuysFromRolla
```

```
<OPTION VALUE="3">3GuysFromRolla  
<OPTION VALUE="4" SELECTED>4GuysFromRolla  
</SELECT>  
</FORM>
```

the preceding code, if viewed through a browser, would have the last list box option selected by default.

Check Boxes

Suppose that you were asked to create a way for a large eCommerce Web site to ascertain the interests of its visitors. You might want to know what product lines customers are interested in. Rather than asking the user to type into a text box what product lines they are interested in, it would make more sense to use a series of check boxes to limit the choices to those that make sense for your business. Further, check boxes are useful if there is a group of related Yes/No type questions that are not mutually exclusive. That is, there are a number of Yes/No type questions, and the user should be able to answer each question in the affirmative or negative.

For the user interests example, you might ask the users to choose what product lines they are interested in. You then might list several product lines such as Home Electronics, Major Appliances, and Stereos. It would be nice to have a series of check boxes next to each product line name, so that the user could check the product lines that interest them. In this example, we'd have three check boxes, one for each of three product lines. Check boxes are created with the `<INPUT>` tag. For a check box, you need to set the `TYPE` property to `TYPE=CHECKBOX`. The `NAME` property is slightly different for a check box. Rather than having a unique `NAME` for each check box, you can group check boxes by giving them all the same `NAME`. In the product line example, you would want to give all three check boxes the same `NAME`. The `VALUE` property needs to be unique among check boxes that have the same `NAME`. The `VALUE` property is what the form processing script will receive when referring to the check box group. Let's write some HTML to create a form to query users about their interests in the three product lines. The code will create three check boxes. When creating a check box, the `<INPUT>` tag only creates the check box. You need to use HTML to label the check box.

Listing 11.12, when viewed through a browser, can be seen in Figure 11.9.

Example 11.12. Users Can Select Their Interests Via a Series of Checkboxes.

```

1: <HTML>
2: <BODY>
3: <FORM METHOD=POST ACTION="/scripts/someFile.asp">
4: What Product Lines are you Interested in?<BR>
5: <INPUT TYPE=CHECKBOX NAME=ProductLine
   VALUE=HomeElectronics>
6: Home Electronics
7: <BR>
8: <INPUT TYPE=CHECKBOX NAME=ProductLine
   VALUE=MajorAppliances>
9: Major Appliances
10: <BR>
11: <INPUT TYPE=CHECKBOX NAME=ProductLine
    VALUE=Stereos>
12: Stereos
13: <P>
14: <INPUT TYPE=SUBMIT>
15: </FORM>
16: </BODY>
17: </HTML>

```

Listing 11.12 creates three checkboxes. Recall that checkboxes are created using the `<INPUT>` tag with the `TYPE` property set to `CHECKBOX`. In lines 5, 8, and 11 we create our three related checkboxes, each with the same `NAME`.

Each checkbox in the group of related checkboxes will be uniquely identified in the form processing script via the `INPUT` tag's `VALUE` property.

The `INPUT` tag simply creates a checkbox; we have to supply our own label for the checkbox.

A checkbox's label is denoted by the text that follows the `INPUT` tag used to create the checkbox. In Listing 8.12, the three checkbox labels are Home Electronics, Major Appliances, and Stereos. Line 5 creates our first checkbox, which, on line 6, is

labeled Home Electronics. Note how the other two labels follow each of their respective checkboxes.

Radio Buttons

Radio buttons and check boxes are a lot alike. Both radio buttons and check boxes are used to group options that users can choose from. Looking back at the last example, we created a form that listed three product lines and asked users to select which ones they were interested in. Because we wanted to let the users select one, two, or three product lines that they were interested in, we used check boxes to facilitate our information gathering. However, what if we wanted to let the user select only one product line? Perhaps we want to ask users what is their most favorite product line. Check boxes wouldn't work in this situation, but radio buttons would.

Check boxes allow users to select none, one, or many of the available options among a group of related options. Radio buttons, on the other hand, only allow none or one of the options to be selected from a group of options. Let's examine how you can use radio buttons to query users for their favorite product line. The code in Listing 11.13 creates a form with three radiobuttons, and the output can be seen in Figure 11.10.

Example 11.13. Radio Buttons Allow the User to Select One Option, at Most

```
1: <HTML>
2: <BODY>
3: <FORM METHOD=POST ACTION="/scripts/someFile.asp">
4: What Product Lines are you most interested in?<BR>
5: <INPUT TYPE=RADIO NAME=ProductLine
   VALUE=HomeElectronics>
6: Home Electronics
7: <BR>
8: <INPUT TYPE=RADIO NAME=ProductLine
   VALUE=MajorAppliances>
9: Major Appliances
10: <BR>
11: <INPUT TYPE=RADIO NAME=ProductLine VALUE=Stereos>
12: Stereos
13: <P>
14: <INPUT TYPE=SUBMIT>
```

```
15: </FORM>
16: </BODY>
17: </HTML>
```

The three radio buttons in Listing 11.13 are related radio buttons; that is, they each share the same NAME (ProductLine). Each radio button is created with an <INPUT> tag with its TYPE set to RADIO. Lines 5, 8, and 11 contain the three INPUT tags responsible for creating our three radio buttons. The text that follows the radio button will be the radio button's label. Our first radio button's label is Home Electronics, and is created in line 6. Major Appliances and Stereos are the other two labels, shown on lines 9 and 12 respectively. Note the syntax of the <INPUT> tag when creating a radio button. It looks a lot like the syntax when we created a check box. The only change is the value of the TYPE property from TYPE=CHECKBOX to TYPE=RADIO. Again, with radio buttons, as with check boxes, related radio buttons have their NAME properties equal but different values for their VALUE properties. You can also use the CHECKED keyword to have a radio button selected by default. Again, its syntax is identical to the syntax for having a check box selected by default. The following line would create a radio button that is checked by default:

```
<INPUT TYPE=RADIO NAME=ProductLine VALUE=Stereos
CHECKED>
```

Summary

1. Explains how to collect information from your users using forms.
2. We discussed a number of useful HTML tags in depth and examined how to use <FORM>, <INPUT>, and <SELECT> tags to create forms. A form, when submitted, sends the values entered into its form fields to a form processing script, which can be an ASP page.
3. This form processing script reads the form field values entered by the user and makes decisions based on these values.
4. Today's lesson also detailed the four types of form fields: text boxes, list boxes, check boxes, and radio buttons. Although each form field type is fairly similar, there are some minor differences in creating and correctly using each of them. Each form field type is also best suited for a specific role. Text boxes are useful when the user needs to enter a string or number.

List boxes are needed when there is a certain set of information the user must select from. Check boxes and radio buttons are a must when there is a related group of options you want to have your users select from.

Question: (Sample of ASP)

A. Short Note on:

- a. Text Box
- b. List box
- c. Radio button
- d. Checkbox

B. Give the output for the following :

(i)

Response.Write(LTRIM(LEFT("####Congratulations", 7))
(where # denotes a blank space)

(ii) Response.Write((3 * 5 > 4 + 5) AND (2 ^ 3 + 9 \ 2))

(iii) Response.Write(ABS(3 - 11 * 4 ^ 2))5.

C. Questions given below are based on ASP:

- a) Name and specify the usage of any two ASP components.
- b) Differentiate between Properties and Methods with the help of an example.
- c) Underline the errors in the following code and write the corrected script.

```

%>
dim fname
fname=Request.Query("fname")
If fname<>"" Then
    Response.Output("Hello " fname "!<br />")
    Response.Output("How are you today?")
End
<%
  
```

d) Give the output for the following code segment:

```

%>
Arr=Array(25,14,20,45,25,4,1,31)
max=ubound(Arr)
For i=max to 1 step -2
    Arr[i]= 10*Arr[i]
Response.write (Arr(i) & "<BR>")
Next
%>
  
```

- D. What is a variable?
- E. What are the methods in Session Object?
- F. What is Global.asa file?
- G. What is the difference between Cookies collection and Form/Query string collection?
- H. What are the properties of Session Object?
- I. Explain the difference between POST and GET Method.
- J. Why do we use Option Explicit?
- K. What is Querystring collection?
- L. What are LOCAL and GLOBAL variables?
- M. What is the difference between ASP and HTML? Or Why ASP is better than HTML?
- N. What is ServerVariables collection?
- O. What are the ASP Scripting Objects?
- P. What is a Form collection?
- Q. What is IIS?
- R. What is the difference between Querystring collection and Form collection?
- S. What are the attributes of the tags? What are their functions?
- T. What is application Object?



INTRODUCTION TO JAVASCRIPT

Unit Structure

12.1 Introduction

12.2 Operators, Assignments and Comparisons, Reserved words

12.3 Starting with JavaScript

- o Writing first JavaScript program

- o Putting Comments

12.4 Functions

Client-Side versus Server-Side Scripting

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

Client-side scripting

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

Server-side scripting

Server-side scripts are run by the Web server. Typically, these scripts are referred to as CGI scripts, CGI being an acronym for Common Gateway Interface, the first interface for server-side Web scripting. Server-side scripts impose more load on the server, but generally don't influence the client—even output to the client is optional; the client may have no idea that the server is running a

script. Perl, Python, PHP, and Java are all examples of server-side scripting languages. The script typically resides only on the server, but is called by code in the HTML document. Although server-side scripts cannot read or write to the client's file system, they usually have some access to the server's file system. As such, it is important that the system administrator takes appropriate measures to secure server-side scripts and limit their access.

12.1 INTRODUCING JAVASCRIPT

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

JavaScript is object-oriented. An *Object* in JavaScript is a resource that has specific characteristics known as *properties* and provides several services known as *methods* and *events*. An example of an object is **document**, which represents the current web page and has properties such as **location** (which stores the URL location of the document) and methods such as **writeln**, which writes dynamically created html text to the document.

A *variable* stores a value. It can be thought of as a labeled box, with the name of the variable as the label and the value as the contents. The JavaScript statement:

```
var x= "hello";  
assigns to the variable named x the String value "hello".  
var x=1;
```

This line of JavaScript assigns to the variable **x** the integer value 1. As you can see, a JavaScript variable can refer to a value of any type; this can be integer, string, or even any type of

object. You don't have to specify the type of variable before creating it. Note that object *properties* can be thought of as variables that belong to the object.

A *method* is basically a collection of statements that does something. For example, a *method* “**writeln()**” exists in the **document** object that can be used to write html to your document. *Methods* are predefined in JavaScript. It is possible for you to define functions, which can be thought of as methods you define outside of any object. When you have the syntax *object.method* as you do with **document.writeln**, the method operates on the object given. In this case, the **writeln** method operates (the operation is writing) to the **document** (the browser window that you see). This syntactic structure is often used in JavaScript.

What java script can do???

Getting your Web page to respond or react directly to user interaction with form elements (input fields, text areas, buttons, radio buttons, checkboxes, selection lists) and hypertext links—a class of application I call the *serverless CGI*

- a. Distributing small collections of database-like information and providing a friendly interface to that data
- b. Controlling multiple-frame navigation, plug-ins, or Java applets based on user choices in the HTML document
- c. Preprocessing data on the client before submission to a server
- d. Changing content and styles in modern browsers dynamically and instantly in response to user interaction

12.1.1 Writing JavaScript Code

JavaScript follows a fairly basic syntax that can be outlined with a few simple rules:

1. With few exceptions, code lines should end with a semicolon (;). Notable exceptions to the semicolon rule are lines that end in a block delimiter
2. ({ or }). Blocks of code (usually under control structures such as functions, if
3. statements, and so on) should be enclosed in braces ({ and }).
4. Although not necessary, explicit declaration of variables is a good idea.

The use of functions to delimit code fragments is highly advised and increases the ability to execute those fragments independently from one another.

12.2.2 Javascript and Comments

Some older browsers do not recognize JavaScript. These browsers would sometimes display JavaScript code in the page as if it were part of the contents of the page

Therefore, it is conventional to place

JavaScript code between comment tags as follows:

```
<script>
<!--
..JavaScript code goes here..
//-->
</script>
```

Older browsers would just ignore the Javascript code between the <!-- and --> comment tags, while new browsers would recognize it as JavaScript code. The // just before the end comment tag --> is a JavaScript comment symbol, and tells the browser not to execute the end comment tag --> as JavaScript. Using comment tags makes a webpage more accessible to older browsers.

12.2.3 Entering Your First Script

It's time to start creating your first JavaScript script. Launch your text editor and browser. Next, follow these steps to enter and preview your first JavaScript script:

1. Activate your text editor and create a new, blank document.
2. Type the script into the window exactly as shown in Listing.
3. Save the document with the name script1.htm. (This is the lowest common denominator filenaming convention for Windows 3.1—feel free to use an .html extension if your operating system allows it.)
4. Switch to your browser.
5. Choose Open (or Open File on some browsers) from the File menu and select script1.htm. (On some browsers, you have to click a Browse button to reach the File dialog box.)

Listing 3-1: Source Code for script1.htm

```

<HTML>
<HEAD>
<TITLE>My First Script</TITLE>
</HEAD>

<BODY>
<H1>Let's Script...</H1>
<HR>
<SCRIPT LANGUAGE="JavaScript">
<!-- hide from old browsers
document.write("This browser is version " + navigator.appVersion)
document.write(" of <B>" + navigator.appName + "</B>.")
// end script hiding -->
</SCRIPT>
</BODY>
</HTML>

```

12.2.4 Calculations and operators

JavaScript supports the usual range of operators for both arithmetic and string values. Tables 5-1 through 5-4 list the various operators supported by JavaScript.

Table 25-1
JavaScript Arithmetic Operators

<i>Operator</i>	<i>Use</i>
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus (division remainder)
++	Increment
--	Decrement

Table 25-2
JavaScript Assignment Operators

<i>Operator</i>	<i>Use</i>
=	Assignment
+=	Increment assignment
-=	Decrement assignment
*=	Multiplication assignment
/=	Division assignment
%=	Modulus assignment

Table 25-3
JavaScript Comparison Operators

<i>Operator</i>	<i>Use</i>
--	Is equal to
!=	Is not equal to
>	Is greater than
<	Is less than
>=	Is greater than or equal to
<=	Is less than or equal to

Table 25-4
Logical Operators

<i>Operator</i>	<i>Use</i>
&&	And
	Or
!	Not

12.2.5 Handling strings

Strings are assigned using the standard assignment operator (=). You can concatenate two strings together using the concatenate operator (+). For example, at the end of this code, the `full_name` variable will contain “Terri Moore”:

```
first_name = "Terri";
last_name = "Moore";
full_name = first_name + " " + last_name;
```

12.3 FUNCTIONS

Functions are a means of grouping code fragments together into cohesive pieces. Typically, those pieces perform very specific tasks—receiving values to execute upon and returning values to indicate their success, failure, or result. There are essentially two types of functions, built-in JavaScript functions and user-defined functions.

12.3.1 Built-in functions

JavaScript has quite a few built-in functions to perform a variety of tasks. Augmenting the functions are a bunch of properties and methods that can be used with just about any object, from

browser function to variable. The scope of built-in JavaScript functions, methods, and properties is too vast to adequately convey here. However, comprehensive references can be found on the Internet, including the following:

◆ Netscape Devedge JavaScript 1.5 Guide

(<http://devedge.netscape.com/library/manuals/2000/javascript/1.5/guide/>)

◆ DevGuru JavaScript Quick Reference

(http://www.devguru.com/Technologies/ecmascript/quickref/javascript_intro.html)

12.3.2 User-defined functions

Like any other robust programming language, JavaScript allows for user-defined functions. User-defined functions allow you to better organize your code into discrete, reusable chunks. User-defined functions have the following syntax:

```
function function_name (arguments) {
...code of function...
return value_to_return;
}
```

For example, the following function will spacefill any string passed to it to 25 characters and return the new string:

```
function spacefill (text) {
while ( text.length < 25 ) {
text = text + " ";
}
return text;
}
```

Elsewhere in your code you can call a function similar to the following:

```
address = spacefill(address);
```

This would cause the variable `address` to be spacefilled to 25 characters:

- a. The `spacefill` function is called with the current value of `address`.
- b. The `spacefill` function takes the value and assigns it to the local variable `text`.
- c. The local variable `text` is spacefilled to 25 characters.
- d. The local variable `text` (now spacefilled) is returned from the function.
- e. The original calling assignment statement assigns the returned value to the `address` variable.

Note The arguments passed to a function can be of any type. If multiple arguments are passed to the function, separate them with commas in both the calling statement and function definition, as shown in the following examples:

Calling statement:

```
spacefill(address, 25)
```

Function statement:

```
function spacefill (text, spaces)
```

Note that the number of arguments in the calling statement and in the function definition should match. The variables used by the function for the arguments and any other variables declared and used by the function are considered local variables—they are inaccessible to code outside the function and exist only while the function is executing.

12.4 DATA TYPES AND VARIABLES

Variables are storage containers where you can temporarily store values for later use. JavaScript, like most scripting languages, supports a wide range of variable types (integer, float, string, and so on) but incorporates very loose variable type checking. That means that JavaScript doesn't care too much about what you store in a variable or how you use the variable's value later in the script. JavaScript variable names are case-sensitive but can contain alphabetic or numeric characters. The following are all valid JavaScript variable names:

```
Rose  
rose99  
total  
99_password
```

Although JavaScript doesn't require that you declare variables before their use, declaring variables is a good programming habit to develop. To declare a variable in JavaScript, you use the `var` keyword. For example, each of the following lines declares a variable:

```
var name = "Hammond";  
var total;  
var tax_rate = .065;
```



WORKING WITH OBJECTS

Unit Structure

13.1 Working with Objects

- o Object Types and Object Instantiation
- o Date object, Math object, String object, Event object, Frame object, Screen object

13.1 WHAT DEFINES AN OBJECT?

When an HTML tag defines an object in the source code, the browser creates a slot for that object in memory as the page loads. But an object is far more complex internally than, say, a mere number stored in memory. The purpose of an object is to represent some “thing.” Because in JavaScript you deal with items that appear in a browser window, an object may be an input text field, a button, or the whole HTML document. Outside of the pared-down world of a JavaScript browser, an object can also represent abstract entities, such as a calendar program’s appointment entry or a layer of graphical shapes in a drawing program.

Every object is unique in some way, even if two or more objects look identical to you in the browser. Three very important facets of an object define what it is, what it looks like, how it behaves, and how scripts control it. Those three facets are properties, methods, and event handlers.

13.1.1 Strings Objects

A *string* is any text inside a quote pair. A quote pair consists of either double quotes or single quotes. This allows one string to nest inside another, as often happens in event handlers. In the following example, the `alert()` method requires a quoted string as a parameter, but the entire method call also must be inside quotes.

```
onClick="alert('Hello, all!')
```

JavaScript imposes no practical limit on the number of characters that a string can hold. However, most older browsers have a limit of 255 characters in length for a script statement. You have two ways to assign a string value to a variable. The simplest is a basic assignment statement:

```
var myString = "Howdy"
```

You can also create a string object using the more formal syntax that involves the **new** keyword and a constructor function (that is, it “constructs” a new object):

```
var myString = new String("Howdy")
```

Whichever way you use to initialize a variable with a string, the variable receiving the assignment can respond to all String object methods.

➤ Joining strings

Bringing two strings together as a single string is called *concatenating* strings. String concatenation requires one of two JavaScript operators. The addition operator (+) linked multiple strings together to produce the text dynamically written to the loading Web page:

```
document.write(" of <B>" + navigator.appName + "</B>.")
```

As valuable as that operator is, another operator can be even more scripter friendly. This operator is helpful when you are assembling large strings in a single variable. The strings may be so long or cumbersome that you need to divide the building process into multiple statements. The pieces may be combinations of *string literals* (strings inside quotes) or variable values. The clumsy way to do it (perfectly doable in JavaScript) is to use the addition operator to append more text to the existing chunk:

```
var msg = "Four score"
msg = msg + " and seven"
msg = msg + " years ago,"
```

But another operator, called the *add-by-value operator*, offers a handy shortcut. The symbol for the operator is a plus and equal sign together (+=). This operator means “append the stuff on the right of me to the end of the stuff on the left of me.” Therefore, the preceding sequence is shortened as follows:

```
var msg = "Four score"
msg += " and seven"
msg += " years ago,"
```

You can also combine the operators if the need arises:

```
var msg = "Four score"
msg += " and seven" + " years ago"
```

➤ String methods

Of all the core JavaScript objects, the String object has the most diverse collection of methods associated with it. Many methods are designed to help scripts extract segments of a string. Another group, rarely used in my experience, wraps a string with one of several style-oriented tags (a scripted equivalent of tags for font size, style, and the like). To use a string method, the string being acted upon becomes part of the reference followed by the method name. All methods return a value of some kind. Most of the time, the returned value is a converted version of the string object referred to in the method call—but the original string is still intact. To capture the modified version, you need to assign the results of the method to a variable:

```
var result = string.methodName()
```

➤ Changing string case

Two methods convert a string to all uppercase or lowercase letters:

```
var result = string.toUpperCase()
var result = string.toLowerCase()
```

Not surprisingly, you must observe the case of each letter of the method names if you want them to work. These methods come in handy when your scripts need to compare strings that may not have the same case (for example, a string in a lookup table compared with a string typed by a user). Because the methods don't change the original strings attached to the expressions, you can simply compare the evaluated results of the methods:

```
var foundMatch = false
if (stringA.toUpperCase() == stringB.toUpperCase()) {
  foundMatch = true
}
```

➤ **String searches**

You can use the `string.indexOf()` method to determine if one string is contained by another. Even within JavaScript's own object data, this can be useful information. For example, another property of the navigator object (`navigator.userAgent`) reveals a lot about the browser that loads the page. A script can investigate the value of that property for the existence of, say, "Win" to determine that the user has a Windows operating system. That short string might be buried somewhere inside a long string, and all the script needs to know is whether the short string is present in the longer one—wherever it might be. The `string.indexOf()` method returns a number indicating the index value (zero based) of the character in the larger string where the smaller string begins.

The key point about this method is that if no match occurs, the returned value is -1. To find out whether the smaller string is inside, all you need to test is whether the returned value is something other than -1. Two strings are involved with this method: the shorter one and the longer one. The longer string is the one that appears in the reference to the left of the method name; the shorter string is inserted as a parameter to the `indexOf()` method. To demonstrate the method in action, the following fragment looks to see if the user is running Windows:

```
var isWindows = false
if (navigator.userAgent.indexOf("Win") != -1) {
  isWindows = true
}
```

The operator in the if construction's condition (`!=`) is the inequality operator.

You can read it as meaning "is not equal to."

➤ **Extracting copies of characters and substrings**

To extract a single character at a known position within a string, use the `charAt()` method. The parameter of the method is an index number (zero based) of the character to extract. When I say *extract*, I don't mean delete, but rather grab a snapshot of the character. The original string is not modified in any way. For example, consider a script in a main window that is capable of

inspecting a variable, `stringA`, in another window that displays map images of different corporate buildings. When the window has a map of Building C in it, the `stringA` variable contains “Building C.” The building letter is always at the tenth character position of the string (or number 9 in a zero-based counting world), so the script can examine that one character to identify the map currently in that other window:

```
var stringA = "Building C"
var bldgLetter = stringA.charAt(9)
// result: bldgLetter = "C"
```

Another method—`string.substring()`—enables you to extract a contiguous sequence of characters, provided you know the starting and ending positions of the substring of which you want to grab a copy. Importantly, the character at the ending position value is not part of the extraction: All applicable characters, up to but not including that character, are part of the extraction. The string from which the extraction is made appears to the left of the method name in the reference. Two parameters specify the starting and ending index values (zero based) for the start and end positions:

```
var stringA = "banana daiquiri"
var excerpt = stringA.substring(2,6)
// result: excerpt = "nana"
```

13.1.2 Math Objects

JavaScript provides ample facilities for math. The `Math` object contains all of these powers. This object is unlike most of the other objects in JavaScript in that you don’t generate copies of the object to use. Programmers call this kind of fixed object a *static object*. That `Math` object (with an uppercase M) is part of the reference to the property or method. Properties of the `Math` object are constant values, such as `pi` and the square root of two:

```
var piValue = Math.PI
var rootOfTwo = Math.SQRT2
```

`Math` object methods cover a wide range of trigonometric functions and other math functions that work on numeric values already defined in your script. For example, you can find which of two numbers is the larger:

```
var larger = Math.max(value1, value2)
Or you can raise one number to a power of ten:
var result = Math.pow(value1, 10)
```

More common, perhaps, is the method that rounds a value to the nearest integer value:

```
var result = Math.round(value1)
```

Another common request of the Math object is a random number. The Math.random() method returns a floating-point number between 0 and 1. To generate a random integer between zero and any top value, use the following

formula:

```
Math.floor(Math.random() * (n + 1))
```

where n is the top number. (Math.floor() returns the integer part of any floating-point number.) To generate random numbers between one and any higher number, use this formula:

```
Math.floor(Math.random() * n) + 1
```

where n equals the top number of the range. For the dice game, the formula for each die is

```
newDieValue = Math.floor(Math.random() * 6) + 1
```

13.1.3 Dates Objects

Working with dates beyond simple tasks can be difficult business in JavaScript. A lot of the difficulty comes with the fact that dates and times are calculated internally according to *Greenwich Mean Time (GMT)*—provided the visitor's own internal PC clock and control panel are set accurately. A scriptable browser contains one global Date object (in truth, one Date object per window) that is always present, ready to be called upon at any moment. The Date object is another one of those static objects. When you wish to work with a date, such as displaying today's date, you need to invoke the Date object constructor to obtain an instance of a Date object tied to a specific time and date. For example, when you invoke the constructor without any parameters, as in `var today = new Date()` the Date object takes a snapshot of the PC's internal clock and returns a date object for that instant. Notice the distinction between the static Date object and a date object

instance, which contains an actual date value. The variable, `today`, contains not a ticking clock, but a value that you can examine, tear apart, and reassemble as needed for your script.

Internally, the value of a date object instance is the time, in milliseconds, from zero o'clock on January 1, 1970, in the Greenwich Mean Time zone—the world standard reference point for all time conversions. That's how a date object contains both date and time information.

You can also grab a snapshot of the `Date` object for a particular date and time in the past or future by specifying that information as parameters to the `Date` object constructor function:

```
var someDate = new Date("Month dd, yyyy hh:mm:ss")
var someDate = new Date("Month dd, yyyy")
var someDate = new Date(yy,mm,dd,hh,mm,ss)
var someDate = new Date(yy,mm,dd)
var someDate = new Date(GMT milliseconds from 1/1/1970)
```

If you attempt to view the contents of a raw date object, JavaScript converts the value to the local time zone string as indicated by your PC's control panel setting. To see this in action, use The Evaluator Jr.'s top text box to enter the following:
`new Date()`

Your PC's clock supplies the current date and time as the clock calculates them (even though JavaScript still stores the date object's millisecond count in the GMT zone). You can, however, extract components of the date object via a series of methods that you apply to a date object instance. Table shows an abbreviated listing of these properties and information about their values.

Table Some Date Object Methods

Table 10-1 Some Date Object Methods		
Method	Value Range	Description
<code>dateObj.getTime()</code>	0-...	Milliseconds since 1/1/70 00:00:00 GMT
<code>dateObj.getYear()</code>	70-...	Specified year minus 1900; four-digit year for 2000+
<code>dateObj.getFullYear()</code>	1970-...	Four-digit year (Y2K-compliant); version 4+ browsers
<code>dateObj.getMonth()</code>	0-11	Month within the year (January = 0)
<code>dateObj.getDate()</code>	1-31	Date within the month
<code>dateObj.getDay()</code>	0-6	Day of week (Sunday = 0)
<code>dateObj.getHours()</code>	0-23	Hour of the day in 24-hour time
<code>dateObj.getMinutes()</code>	0-59	Minute of the specified hour
<code>dateObj.getSeconds()</code>	0-59	Second within the specified minute
<code>dateObj.setTime(val)</code>	0-...	Milliseconds since 1/1/70 00:00:00 GMT
<code>dateObj.setYear(val)</code>	70-...	Specified year minus 1900; four-digit year for 2000+
<code>dateObj.setMonth(val)</code>	0-11	Month within the year (January = 0)
<code>dateObj.setDate(val)</code>	1-31	Date within the month
<code>dateObj.setDay(val)</code>	0-6	Day of week (Sunday = 0)
<code>dateObj.setHours(val)</code>	0-23	Hour of the day in 24-hour time
<code>dateObj.setMinutes(val)</code>	0-59	Minute of the specified hour
<code>dateObj.setSeconds(val)</code>	0-59	Second within the specified minute

Be careful about values whose ranges start with zero, especially the months. The `getMonth()` and `setMonth()` method values are zero based, so the numbers are one less than the month numbers you are accustomed to working with (for example, January is 0, December is 11).

13.1.4 Screen Object

In JavaScript 1.2, the `screen` property of a `Window` object refers to a `Screen` object. This `Screen` object provides information about the size of the user's display and the number of colors available. The `width` and `height` properties specify the size of the display in pixels. The `availWidth` and `availHeight` properties specify the display size that is actually available: they exclude the space required by features like the Windows 95 taskbar. You can use these properties to help you decide what size images to include in a document, for example, or in a program that creates multiple browser windows, what size windows to create.

The `colorDepth` property specifies the base-2 logarithm of the number of colors that can be displayed. Often, this value is the same as the number of bits per pixel used by the display. For example, an 8-bit display can display 256 colors, and if all of these colors were available for use by the browser, the `screen.colorDepth` property would be 8. In some circumstances, however, the browser may restrict itself to a subset of the available colors, and you might find a `screen.colorDepth` value that is lower than the bits-per-pixel value of the screen. If you have several versions of an image that were defined using different numbers of colors, you can test this `colorDepth` property to decide which version to include in a document.

➤ **Javascript Window Screen Object Properties**

Following are the commonly used properties of javascript screen object that are also accessible in almost all the modern browsers:

1. **availWidth**: returns the available width of the display screen of the computer monitor based on resolution of the screen. `availWidth` property of screen object gives approximate value of width available to display the content in a browser along x-axis. **availWidth property** returns the width in pixels without excluding the width of window's taskbar coz in Windows operation system, according to default settings, position of taskbar is bottom of display screen.
2. **availHeight**: returns the available height of the display screen of the computer monitor based on resolution of the screen. `availHeight` property of screen object gives approximate value of height available to display the content in a browser along y-axis. **availHeight property** returns the height in pixels excluding the height of window's taskbar coz in Windows operation system, according to default settings, position of taskbar is bottom of display screen that reduces the available height for the web browsers.
3. **colorDepth**: returns the bit depth of the color palette retrieved from the graphic properties of the computer system. `colorDepth` property of the javascript screen object returns the color quality property of the client's computer such as 16 bit, 24 bit or 32 bit that depends upon graphics card and its memory (buffer).

4. **height**: returns the height of display screen. Height property of javascript screen object actually returns the number of pixels in vertical direction i.e. vertical value for screen resolution along y-axis.

5. **width**: returns the width of display screen. Width property of javascript screen object actually returns the number of pixels in horizontal direction i.e. horizontal value for screen resolution along x-axis.

Eg:

```
<html>
<head>
  <title>Javascript Window Screen availWidth</title>
</head>
<body>
  <script type="text/javascript" language="javascript">
    document.write(screen.availWidth);
    document.write("<br />");
    document.write(window.screen.availWidth);
  </script>
</body>
</html>
```

13.1.5 Frame Object Properties

The JavaScript Frame object is the representation of an HTML FRAME which belongs to an HTML FRAMESET. The frameset defines the set of frame that make up the browser window. The JavaScript Frame object is a property of the window object.

The frame object is a browser object of JavaScript used for accessing HTML frames. The user can use frames array to access all frames within a window. Using the indexing concept, users can access the frames array.

NOTE:

- The frames array index always starts with zero and not 1.
- The frame object is actually a child of the window object. These objects are created automatically by the browser and help users to control loading and accessing of frames.

- The properties and methods of frame object are similar to that of Window object in JavaScript.
- The frame object does not support close() method that is supported by window object.
- Using the <FRAMESET> document creates frame objects and each frame created is thus a property of window object.

➤ **Properties of frame object:**

- frames
- name
- length
- parent
- self

frames:

The frames property of frame object denotes a collection or array of frames in a window and also in a frame set.

self:

As the name implies, the self property of frames object denotes the current frame. Using self property, the user can access properties of the current frame window.

name:

The name property of frame object denotes the name of the frame. The method of denoting the name attribute is performed by using the name attribute of the <frame> tag.

For example it can be written as:

```
exforsys=window.frames(2).name
```

The above statement would store the name of the third window frame (as the frames array start with index 0) in a frameset document in the variable exforsys.

length:

The frames array has all the frames present within a window and the length property of the frame object denotes the length of the frames array or gives the number of frames present in a window or a frames array.

parent:

As the name implies, the parent property of frames object denotes the parent frame of the current frame.

➤ Methods of frame object:

- blur()
- focus()
- setInterval()
- clearInterval()
- setTimeout(expression, milliseconds)
- clearTimeout(timeout)

blur():

blur() method of frame object removes focus from the object.

focus():

focus() method of frame object gives focus to the object.

setInterval():

setInterval() method of frame object is used to call a function of JavaScript or to evaluate an expression after the time interval specified in arguments has expired. The time interval in arguments is always specified in milliseconds. **For example:**

```
setInterval=exforsys(test(),2000)
```

In the above statement, the function test() executes after 2000 milliseconds (2 seconds), specified in the argument.

clearInterval():

clearInterval method of frame object is used to cancel the corresponding defined setInterval method. This is written by referencing the setInterval method using its ID or variable. General syntax for the method clearInterval() is as below:

```
clearInterval (Interval_ID)
```

setTimeout(expression, milliseconds):

setTimeout method of frame object can be used to execute any function, or access any method or property after a specified time interval given to this method as argument. General syntax for the method setTimeout() is as below:

```
setTimeout(expression, milliseconds)
```

For example:

```
exforsys=setTimeout ("test()", 3000)
```

The time is always specified in milliseconds and in the above statement, the function test() is called after the specified time of 3000 milliseconds (3 seconds). This is stored in variable named exforsys. There is confusion about the similarity between setTimeout() method and setInterval() method. The main difference between the two methods is the setInterval method will repeatedly call the referenced function or evaluate the expression until the user leaves the document. In the setTimeout method, the call executes only once after the specified time interval given as argument.

clearTimeout():

clearTimeout method of frame object is used to clear a specified setTimeout method. This is written by referencing the setTimeout method using its ID or variable.

General syntax for the method clearTimeout is as below:

```
clearTimeout ID_of_setTimeout
```

The above statement clears the setTimeout associated with the ID named as exforsys, created in the earlier example.

➤ **Events associated with frame object:**

Though the frame object and frames array have no event handlers associated directly with them, the following event handlers are used to access and control frame objects and frames array:

- onBlur
- onFocus
- OnLoad
- OnUnload

13.1.6 Form Object

The JavaScript Form Object is a property of the document object. This corresponds to an HTML input form constructed with the FORM tag. A form can be submitted by calling the JavaScript submit method or clicking the form submit button.

➤ Form Object Properties

- action - This specifies the URL and CGI script file name the form is to be submitted to. It allows reading or changing the ACTION attribute of the HTML FORM tag.
- elements - An array of fields and elements in the form.
- encoding - This is a read or write string. It specifies the encoding method the form data is encoded in before being submitted to the server. It corresponds to the ENCTYPE attribute of the FORM tag. The default is "application/x-www-form-urlencoded". Other encoding includes text/plain or multipart/form-data.
- length - The number of fields in the elements array. I.E. the length of the elements array.
- method - This is a read or write string. It has the value "GET" or "POST".
- name - The form name. Corresponds to the FORM Name attribute.
- target - The name of the frame or window the form submission response is sent to by the server. Corresponds to the FORM TARGET attribute.

➤ Form Objects

Forms have their own objects.

- button - An GUI pushbutton control. Methods are click(), blur(), and focus(). Attributes:
 - name - The name of the button
 - type - The object's type. In this case, "button".
 - value - The string displayed on the button.
- checkbox - An GUI check box control. Methods are click(), blur(), and focus(). Attributes:
 - checked - Indicates whether the checkbox is checked. This is a read or write value.
 - defaultChecked - Indicates whether the checkbox is checked by default. This is a read only value.

- name - The name of the checkbox.
- type - Type is "checkbox".
- value - A read or write string that specifies the value returned when the checkbox is selected.

FileUpload - This is created with the INPUT type="file". This is the same as the text element with the addition of a directory browser. Methods are blur(), and focus().

- name - The name of the FileUpload object.
- type - Type is "file".
- value - The string entered which is returned when the form is submitted.
- hidden - An object that represents a hidden form field and is used for client/server communications. No methods exist for this object. Attributes:
 - name - The name of the Hidden object.
 - type - Type is "hidden".
 - value - A read or write string that is sent to the server when the form is submitted.
- password - A text field used to send sensitive data to the server. Methods are blur(), focus(), and select(). Attributes:
 - defaultValue - The default value.
 - name - The name of the password object."
 - type - Type is "password".
 - value - A read or write string that is sent to the server when the form is submitted.
- radio - A GUI radio button control. Methods are click(), blur(), and focus(). Attributes:
 - checked - Indicates whether the radio button is checked. This is a read or write value.
 - defaultChecked - Indicates whether the radio button is checked by default. This is a read only value.
 - length - The number of radio buttons in a group.
 - name - The name of the radio button.
 - type - Type is "radio".
 - value - A read or write string that specifies the value returned when the radio button is selected.
- reset - A button object used to reset a form back to default values. Methods are click(), blur(), and focus(). Attributes:
 - name - The name of the reset object.
 - type - Type is "reset".
 - value - The text that appears on the button. By default it is "reset".

- select - A GUI selection list. This is basically a drop down list. Methods are blur(), and focus(). Attributes:
 - length - The number of elements contained in the options array.
 - name - The name of the selection list.
 - options - An array each of which identifies an options that may be selected in the list.
 - selectedIndex - Specifies the current selected option within the select list
 - type - Type is "select".
- submit - A submit button object. Methods are click(), blur(), and focus(). Attributes:
 - name - The name of the submit button.
 - type - Type is "submit".
 - value - The text that will appear on the button.
- text - A GUI text field object. Methods are blur(), focus(), and select(). Attributes:
 - defaultValue - The text default value of the text field.
 - name - The name of the text field.
 - type - Type is "text".
 - value - The text that is entered and appears in the text field. It is sent to the server when the form is submitted.
- textarea - A GUI text area field object. Methods are blur(), focus(), and select(). Attributes:
 - defaultValue - The text default value of the text area field.
 - name - The name of the text area.
 - type - Type is textarea.

value- The text that is entered and appears in the text area field. It is sent to the server when the form is submitted.

➤ **Form Object Methods**

- reset() - Used to reset the form elements to their default values.
- submit() - Submits the form as though the submit button were pressed by the user.

➤ **Events**

- onReset
- onSubmit

Onsubmit and Onchange

The **onchange()** event handler is triggered whenever the content of a form field is changed. The form field where this is most useful is for drop down selection lists since by using onchange instead of onblur the field can be tested immediately rather than waiting for a different field to be selected.

The **onsubmit()** event handler is attached to the form tag itself. Whenever a submit button is selected (or the submit method for the form is called from within your Javascript code) this event will be triggered.

Here is a sample form to demonstrate how these events are triggered:



Here is the code for the above form with the tag names and the event handlers shown in bold.

```
<form name="ex" method="POST"
onsubmit="alert('onsubmit');return false;">
<div align="center">
<select name="sel" size="1"
onchange="alert('onchange')">
<option value="1" selected="selected">1</option>
<option value="2">2</option>
<option value="3">3</option>
</select>
<input type="submit" value="submit" />
</div></form>
```

Onreset

The **onreset()** event handler (like onsubmit) is attached to the form itself. This event is triggered if the form contains a reset button and that button is pressed.



HANDLING EVENTS

Unit Structure

14.1 Handling Events

- o Event handling attributes
- o Window Events, Form Events
- o Event Object
- o Event Simulation

14.1 HANDLING EVENTS

In this JavaScript tutorial, you will learn about handling events in JavaScript, what is event handling in JavaScript? events in JavaScript, events associated with mouse - onmousemove, onclick, ondblclick, onmouseout, onmouseover, events associated with keyboard - onkeydown, onkeyup, onkeypress. onerror, onfocus, onblur, onsubmit, onload and onunload.

What is Event Handling in JavaScript?

This is a very vital concept of JavaScript because without events there would be no code. Event handling is the execution of code for the user's reaction. In other words, when a user performs some action, the associated event fires or executes. For example, if a programmer want a piece of code to execute when a user presses a button, then the code is placed in the onclick event of the button, executing the code when the user clicks that button. In addition, there are events in JavaScript that are automatically fired without the intervention of the user. The load event fires when the page loads. Thus, there are various scenarios for firing events such as:

- When the user performs some action based on which event fires
- When the page load event fires
- When some fields change, the associated events fire.

14.1 **Events in JavaScript:**

There are numerous events in JavaScript, some of which are listed below. A particular object has numerous events associated with it, depending on the action taken by the user.

For example, the object mouse has numerous events associated with it which depend on the user's actions.

➤ **Events associated with Mouse:**

a) onmousemove:

If the user moves a button, then the events associated with onmousemove fire.

b) onclick:

onclick events fire when the mouse button clicks.

c) ondblclick:

The event ondblclick fires when the mouse button is double clicked.

There are also some events associated with the mouse pointer position such as:

d) onmouseout:

onmouseout event fires if the mouse pointer position is out of focus from the element.

e) onmouseover:

onmouseover event fires if the mouse pointer position is in focus of the element position.

The above are some of the various mouse events available in JavaScript.

➤ **Events associated with Keyboard:**

- onkeydown
- onkeyup
- onkeypress

a) onkeydown:

onkeydown event fires when key is pressed.

b) onkeyup:

onkeyup event fires when key is released.

c) onkeypress:

The event onkeypress fires if the onkeydown is followed by onkeyup.

There are many additional events available in JavaScript. A few are listed below:

d) onerror:

onerror event fires when an error occurs.

e) onfocus:

When a element gains focus, onfocus event fires or executes.

f) onblur:

In contrast to an onfocus event, this event fires when the element loses its focus. Both onfocus and onblur may be used for handling validation of forms.

g) onsubmit:

The event onsubmit fires when the command form submit is given. This event is used for validating all fields in the form before submitting the form.

h) onload:

onload event automatically executes as soon as the document fully loads. This loads when the user enters the page. This is a commonly used event. This event is used to check compatibility with the browser version and type. Based on this compatibility, the appropriate version of a page will then load.

i) onunload:

In contrast to onload event, the onunload event fires when the user leaves the page.

14.1.1 JavaScript Event Handler

In this JavaScript tutorial, you will learn about using event handlers along with events for each HTML tag.

➤ **Using Event Handler in JavaScript:**

Event Handlers are used in JavaScript by placing the name of the event handler inside the HTML tag associated with object. This is followed by ='JavaScript code', the code in JavaScript which must execute when the event fires.

The events for each HTML tag are as follows:

<A>

click(onClick)
mouseover(onMouseOver)
mouseout (onMouseOut)

<AREA>

mouseover(onMouseOver)
mouseout (onMouseOut)

<BODY>

blur(onBlur)
error(onError)
focus(onFocus)
load(onLoad)
unload (onUnload)

<FORM>

submit(onSubmit)
reset (onReset)

<FRAME>

blur(onBlur)
focus (onFocus)

<FRAMESET>

blur(onBlur)
error(onError)
focus(onFocus)
load(onLoad)
unload (onUnload)

abort(onAbort)
 error(onError)
 load (onLoad)

<INPUT TYPE = "button">

click (onClick)

<INPUT TYPE = "checkbox">

click (onClick)

<INPUT TYPE = "reset">

click (onClick)

<INPUT TYPE = "submit">

click (onClick)

<INPUT TYPE = "text">

blur(onBlur)
 focus(onFocus)
 change(onChange)
 select (onSelect)

<SELECT>

blur(onBlur)
 focus(onFocus)
 change (onChange)

<TEXTAREA>

blur(onBlur)
 focus(onFocus)
 change(onChange)
 select (onSelect)

For example, consider a button placed in a form named *PressButton*. The following code placed in the click event of the button named *PressButton* would be written:

```
<input type="button"
name="PressButton"
value="Press the Button to output value!!!"
onClick="outputvalue();">
```

In the above example, when the user clicks the button, the *onclick* event of the button fires and the message assigned to value displays:

Press the Button output value!!!

The block of code written in the function *outputvalue()* in JavaScript fires or calls.

14.1.2 JavaScript Event Object

In this JavaScript tutorial you will learn about JavaScript event object, properties of event object, altKey, ctrlKey and shiftKey, button, integer value action representation, cancelBubble, clientX and clientY, fromElement and toElement, height and width.

➤ **event Object:**

The event object is a browser object used to get information about a particular event specified. Using event object, users can access information about event happenings. The difference between event object and Event object is such that the latter gives constants that can be used to identify events, while the former is used to get information about events.

Properties of event Object:

- altKey,ctrlKey and shiftKey
- button
- cancelBubble
- clientX
- clientY
- fromElement
- toElement
- height
- width
- keycode
- layerX
- layerY
- modifiers
- offsetX
- offsetY
- pageX
- pageY
- reason
- returnValue

- screenX
- screenY
- srcElement
- srcFilter
- target
- type
- which
- x and y

In the above list, some of the events are supported with Internet Explorer Browser and some with Navigator browser, which will be detailed. Some of the properties of event object are explained below.

altKey, ctrlKey and shiftKey:

The above property is used to indicate whether the Alt Key, Control Key or Shift Key was pressed by the user when the event occurred. The indication is known by appropriately setting the value as true or false. This property takes a boolean value as its return type. Both Internet Explorer and Navigator support this property. In Navigator, there is an extra property called metaKey which is not supported by Internet Explorer. This indicates if the user has pressed Meta key when the event occurred.

button:

The button property of event object is used to denote whether the mouse button was pressed or released when the event occurred. The indication is performed by means of an integer value where the values listed below are returned as per the action when the event occurred. The return value from button property is an integer.

Integer Value Action Representation

0 no Button was pressed

1 Left Mouse Button was pressed

2 Right Mouse Button was pressed

4 Middle Mouse Button was pressed

Sometimes, user presses more than one button at the same time when the event occurs. To indicate such a situation, the integer representation would take the sum of integer representations of buttons pressed by the user.

For example, if a user presses both the right and the middle button, then addition of the integer representations of right and middle button gives $2+4=6$ is returned by the property. This property is supported both by Internet Explorer and Navigator. In Navigator, the integer representation values differ:

Integer Value Action Representation

0 Left Mouse Button was pressed

2 Right Mouse Button was pressed

1 Middle Mouse Button was pressed

cancelBubble:

`cancelBubble` property of event object is used for enabling or disabling the event bubbling concept of an event object. Event bubbling concept will be discussed in the next section of this tutorial. This property enables and disables event bubbling by setting the boolean value appropriately as true or false as needed. The value of the property `cancelBubble` is set to true to prevent the event from bubbling and if it is set to false, the event bubbling is enabled. Only Internet Explorer supports this property. In Navigator, to implement the same property, the programmer has to use the method associated with event object (to be discussed in the methods of event object section).

clientX and clientY:

This property of event object indicates the cursor's horizontal and vertical position when the event occurs relative to the upper-left corner of the window. The denoted value would be in terms of pixels. Both by Internet Explorer and Navigator support this property.

Syntax for using this property is as follows:

```
event.clientX
```

```
event.clientY
```

fromElement and toElement:

These properties are supported only in Internet Explorer and denote the HTML element. The event moves from or to, respectively, for `fromElement` and `toElement`. The properties `fromElement` and `toElement` each denote, respectively, the

elements the mouse is leaving from and moving onto. In Navigator, the property named `relatedTarget` is used to achieve the same result. In case of a mouseover event, the `relatedTarget` property of event object denotes the element that the mouse has left. In case of mouseout event, the `relatedTarget` property of event object denotes the element that the mouse has entered.

height and width:

The height and width property of event object is only supported by Navigator and it indicates the height and width of window or frame, respectively. The value is denoted in pixels.

JavaScript Window Object

The JavaScript Window Object is the highest level JavaScript object which corresponds to the web browser window.

PROPERTIES

closed Property

This property is used to return a Boolean value that determines if a window has been closed. If it has, the value returned is true.

Syntax: **window.closed**

The following code opens a new window and then immediately closes it. The **onClick** event of the button then calls a function which uses the **window.closed** property to display the status (open or closed) of the window.

Code:

```
<INPUT TYPE="Button" NAME="winCheck" VALUE="Has window
been closed?" onClick=checkIfClosed(>
```

```
newWindow=window.open("",',toolbar=no,scrollbars=no,width=300,
height=150')
newWindow.document.write("This is 'newWindow'")
newWindow.close()
```

```
function ifClosed() {
document.write("The window 'newWindow' has been closed")
}
```

```
function ifNotClosed() {
document.write("The window 'newWindow' has not been closed")
}
```

```
function checkIfClosed() {
if (newWindow.closed)
    ifClosed()
else
    ifNotClosed()
}
```

defaultStatus Property

This property is used to define the default message displayed in a window's `status` bar.

Syntax: **window.defaultStatus(= "message") document Property**

Code:

```
window.defaultStatus = "This is the default status bar message."
```

document Property

This property's value is the document object contained within the window.

Syntax: **window.document**

frames Property

This property is an array containing references to all the named child frames in the current window.

Syntax: **window.frames (= "frameID")**

history Property

This property's value is the window's History object, containing details of the URL's visited from within that window.

Syntax: **window.history**

innerHeight / innerWidth Properties

These properties determine the inner dimensions of a window's content area.

Syntax: **window.innerHeight = pixelDimensions**
window.innerWidth = pixelDimensions

length Property

This property returns the number of child frames contained within a window, and gives identical results as using the length property of the **frames** array.

Syntax: **window.length**

location Property

This property contains details of the current URL of the window and its value is always the Location object for that window.

Syntax: **window.location**

locationbar Property

This property relates to the area of a browser's window that contains the details of the URL or bookmark (this is where you physically enter URL details). The locationbar property has its own property, **visible**, that defaults to true (visible) and can be set to false (hidden).

Syntax: **window.locationbar[.visible = false]**

menubar Property

This property relates to the area of a browser's window that contains the various pull-down menus (File, Edit, View, etc.). The menubar property has its own property, **visible**, that defaults to true (visible) and can be set to false (hidden).

Syntax: **window.menubar[.visible = false]**

name Property

This property is used to return or set a window's name.

Syntax: **window.name**

opener Property

When opening a window using `window.open`, use this property from the destination window to return details of the source window. This has many uses, for example, `window.opener.close()` will close the source window.

Syntax: **`window.opener`**

outerheight / outerwidth Property

These properties determine the dimensions, in pixels, of the outside boundary, including all interface elements, of a window.

Syntax: **`window.outerheight`**

Syntax: **`window.outerwidth`**

pageXOffset / pageYOffset Property

These properties return the X and Y position of the current page in relation to the upper left corner of a window's display area.

Syntax: **`window.pageXOffset`**

Syntax: **`window.pageYOffset`**

parent Property

This property is a reference to the window or frame that contains the calling child frame.

Syntax: **`window.parent`**

personalbar Property

This property relates to the browser's personal bar (or directories bar). The `personalbar` property has its own property, `visible`, that defaults to `true` (visible) and can be set to `false` (hidden).

Syntax: **`window.personalbar[.visible = false]`**

scrollbars Property

This property relates to the browser's scrollbars (vertical and horizontal). The `scrollbars` property has its own property, `visible`, that defaults to `true` (visible) and can be set to `false` (hidden).

Syntax: **`window.scrollbars[.visible = false]`**

self Property

This property is a reference (or synonym) for the current active window or frame.

Syntax: **self.property or method**

status Property

This property, which can be set at any time, is used to define the transient message displayed in a window's status bar such as the text displayed when you **onMouseOver** a link or anchor.

Syntax: **window.status(= "message")**

statusbar Property

This property relates to the browser's status bar. The statusbar property has its own property, visible, that defaults to true (visible) and can be set to false (hidden).

Syntax: **window.statusbar[.visible = false]**

toolbar Property

This property sets or returns a Boolean value that defines whether the browser's tool bar is visible or not. The default is true (visible). False means hidden. It can only be set before the window is opened and you must have UniversalBrowserWrite privilege.

Syntax: **window.toolbar[.visible = false]**

top Property

This property is a reference (or synonym) for the topmost browser window.



Syntax: **top.property or method**

window Property


This property is a reference (or synonym) for the current window or frame.



Syntax: **window.property or method**

METHODS

alert Method	<p>Syntax: window.alert("message")</p> <p>This method is used to display an alert box containing a message and an o.k. button. Use this method to convey a message that does not require a decision from the user.</p> <p>Code: <code>window.alert("Welcome to google.com")</code> If you wish to have the text appear on more than one line, you use the <code>\n</code> as a line break. <code>window.alert("Welcome to\ngoogle.com")</code></p> <p>Output:</p> 
back Method	<p>Using this method is the same as clicking the browser's Back button, i.e. it undoes the last navigation step performed from the current top-level window.</p> <p>Syntax: window.back()</p> <p>The following example creates a button on the page that acts the same as the browser's back button.</p> <p>Code: <code><input type="button" value="Go back" onClick="window.back()"></code></p> <p>Output: </p>
blur Method	<p>This method is used to remove focus from the current window.</p> <p>Syntax: window.blur()</p>
captureEvents Method	<p>This method instructs the window to capture all events of a particular type. See the event object for a list of event types.</p> <p>Syntax: window.captureEvent(eventType)</p>

clearInterval Method	This method is used to cancel a timeout previously set with the setInterval method. Syntax: window.clearInterval(intervalID)
clearTimeout Method	This method is used to cancel a timeout previously set with the setTimeout method. Syntax: window.clearTimeout(timeoutID)
close Method	This method is used to close a specified window. If no window reference is supplied, the close() method will close the current active window. Note that this method will only close windows created using the open() method; if you attempt to close a window not created using open(), the user will be prompted to confirm this action with a dialog box before closing. The single exception to this is if the current active window has only one document in its session history. In this case the closing of the window will not require confirmation. Syntax: window.close()
confirm Method	This method brings up a dialog box that prompts the user to select either 'o.k.' or 'cancel', the first returning true and the latter, false. Syntax: window.confirm("message") The following example opens a new window, creates a button in the original window and assigns the closeWindow() function to its onClick event handler. This function prompts the user to confirm the closing of the new window. Code: <pre> <form action="" method="POST" id="myForm"> <input type="Button" name="" value="Close" id="myButton" onClick="closeWindow()"> <script type="" language="JavaScript"> myWindow = window.open("", "tinyWindow", 'toolbar, width=150, height=100') function closeWindow() { myWindow.document.write("Click 'O.K.' to close me and 'Cancel' to leave me open.") if (confirm("Are you sure you want to close this </pre>

	<pre> window?")) { myWindow.close() } } </script></form> Output: </pre> 
disableExternalCapture Method	<p>This method disables the capturing of events previously enabled using the enableExternalCapture method below.</p> <p>Syntax: window.disableExternalCapture()</p>
enableExternalCapture Method	<p>This method allows a window that contains frames to capture events in documents loaded from different servers.</p> <p>Syntax: window.enableExternalCapture()</p>
find Method	<p>This method allows the searching of the contents of a window for a specified string. The caseSensitive and backward arguments are Booleans and to use either of these you must also specify the other. If a search string is not supplied, JavaScript will display a Find dialog box which prompts the user for a string to search for, and also provides the facility to set the other two (caseSensitive and backward) arguments.</p> <p>Syntax: window.find([string[, caseSensitive, backward]])</p> <p>The first example below uses the onClick event handler of the <A> tag to call the findhello() function that searches the contents of the window for the strings "hello" and "goodbye". The results of these searches are displayed, true or false, as JavaScript alerts. The second example shows the "Find" dialog box that is displayed if no search string is supplied.</p> <p>Code:</p>

	<pre><SCRIPT> function findhello () { alert("FIND hello = " + window.find("hello")) alert("FIND goodbye = " + window.find("goodbye")) } </SCRIPT> Find hello Hello Code: self.find() Output:</pre> 
focus Method	<p>This method is used to give focus to the specified window. This is useful for bringing windows to the top of any others on the screen.</p> <p>Syntax: window.focus()</p>
forward Method	<p>Using this method is the same as clicking the browser's Forward button, i.e. it goes to the next URL in the history list of the current top-level window.</p> <p>Syntax: window.forward()</p> <p>The following example creates a button on the page that acts the same as the browser's Forward button.</p> <p>Code:</p> <pre><input type="button" value="Go back" onClick="window.forward()"></pre> <p>Output: </p>
handleEvent Method	<p>This method is used to call the handler for the specified event.</p> <p>Syntax: window.handleEvent("eventID")</p>

home Method	Using this method has the same effect as pressing the Home button in the browser, i.e. the browser goes to the URL set by the user as their home page. Syntax: window.home()
moveBy Method	This method is used to move the window a specified number of pixels in relation to its current co-ordinates. Syntax: window.moveBy(horizPixels, vertPixels)
moveTo Method	This method moves the window's left edge and top edge to the specified x and y co-ordinates, respectively. Syntax: window.moveTo(Xposition, Yposition)
open Method	Syntax: window.open(URL, name [, features]) This method is used to open a new browser window. Note that, when using this method with event handlers, you must use the syntax window.open() as opposed to just open() . Calling just open() will, because of the scoping of static objects in JavaScript, create a new document (equivalent to document.open()), not a window. The available parameters are as follows: URL - this is a string containing the URL of the document to open in the new window. If no URL is specified, an empty window will be created. name - this is a string containing the name of the new window. This can be used as the 'target' attribute of a <FORM> or <A> tag to point to the new window. features - this is an optional string that contains details of which of the standard window features are to be used with the new window. This takes the form of a comma-delimited list. Most of these features require yes or no (1 or 0 is also o.k.) and any of these can be turned on by simply listing the feature (they default to yes). Also, if you don't supply any of the feature arguments, all features with a choice of yes or no are enabled; if you do specify any feature parameters, titlebar and hotkeys still default to yes but all others are no.


Note that many of the values for the **features** parameter are Netscape only. Further, with the exception of **dependent** and **hotkey**, these Netscape only values represent potential sources of security problems and therefore require signed script (and user's permission) if they are to be used.

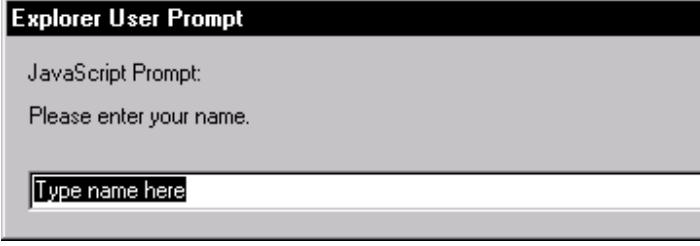
Details of the available values are given below:

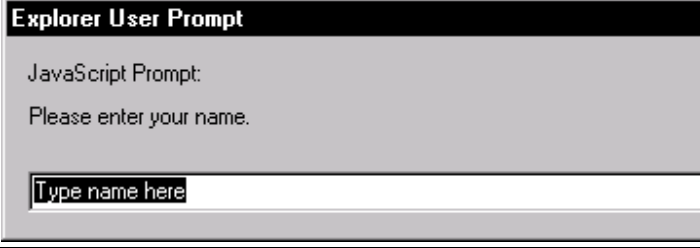
features Value	Description
alwaysLowered	When set to yes, this creates a window that always floats below other windows.
alwaysRaised	When set to yes, this creates a window that always floats above other windows.
dependent	When set to yes, the new window is created as a child (closes when the parent window closes and does not appear on the task bar on Windows platforms) of the current window.
directories	When set to yes, the new browser window has the standard directory buttons.
height	This sets the height of the new window in pixels.
hotkeys	When set to no, this disables the use of hotkeys (except security and quit hotkeys) in a window without a menubar.

innerHeight	This sets the inner height of the window in pixels.
innerWidth	This sets the inner width of the window in pixels.
location	When set to yes, this creates the standard Location entry feild in the new browser window.
menubar	When set to yes, this creates a new browser window with the standard menu bar (File, Edit, View, etc.).
outerHeight	This sets the outer height of the new window in pixels.
outerWidth	This sets the outer width of the new window in pixels.
resizable	When set to yes this allows the resizing of the new window by the user.
screenX	This allows a new window to be created at a specified number of pixels from the left side of the screen.
screenY	This allows a new window to be created at a specified number of pixels from the top of the screen.
scrollbars	When set to yes the new window is created with the standard horizontal and vertical scrollbars, where needed

status	When set to yes, the new window will have the standard browser status bar at the bottom.
titlebar	When set to yes the new browser window will have the standard title bar.
toolbar	When set to yes the new window will have the standard browser tool bar (Back, Forward, etc.).
width	This sets the width of the new window in pixels.
z-lock	When set to yes this prevents the new window from rising above other windows when it is made active (given focus).
These features may only be used with IE4:	
channelmode	sets if the window appears in channel mode.
fullscreen	the new window will appear in full screen.
Left	same as screenX, allows a new window to be created at a specified number of pixels from the left side of the screen.
top	same as screenY, allows a new window to be created at a specified number of pixels from the top of the screen.

	<p>The following example creates a new window of the specified dimensions complete with toolbar, changes the background color and writes a message to it.</p> <p>Code:</p> <pre>myWindow = window.open("", "tinyWindow", 'toolbar, width=150, height=100') myWindow.document.write("Welcome to this new window!") myWindow.document.bgColor="lightblue" myWindow.document.close()</pre> <p>Output:</p> 
print Method	<p>This method is used to print the contents of the specified window.</p> <p>Syntax: window.print()</p>
prompt Method	<p>This method displays a dialog box prompting the user for some input. Syntax: window. Prompt (message [,defaultInput])</p> <p>This method displays a dialog box prompting the user for some input. The optional defaultInput parameter specifies the text that initially appears in the input field.</p> <p>The following example prompts the user for their name and then writes a personalized greeting to the page.</p> <p>Code:</p> <pre><body onload=greeting()> <script type="text/javascript"> function greeting() {</pre>

	<pre> y = (prompt("Please enter your name.", "Type name here")) document.write("Hello " + y) } </script> </pre> <p>Output:</p> 
releaseEvents Method	<p>This method is used to release any captured events of the specified type and to send them on to objects further down the event hierarchy</p> <p>Syntax: window.releaseEvents("eventType")</p>
resizeBy Method	<p>This method is used to resize the window. It moves the bottom right corner of the window by the specified horizontal and vertical number of pixels while leaving the top left corner anchored to its original co-ordinates.</p> <p>Syntax: window.resizeBy(horizPixels, vertPixels)</p>
resizeTo Method	<p>This method is used to resize a window to the dimensions supplied with the outerWidth and outerHeight (both integers, in pixels) parameters.</p> <p>Syntax: window.resizeTo(outerWidth, outerHeight)</p>
routeEvent Method	<p>This method is used to send a captured event further down the normal event hierarchy; specifically, the event is passed to the original target object unless a sub-object of the window (a document or layer) is also set to capture this type of event, in which case the event is passed to that sub-object.</p> <p>Syntax: window.routeEvent(eventType)</p>
scroll Method	<p>This method is used to scroll the window to the supplied co-ordinates. This method is now deprecated; use the scrollTo method detailed below instead.</p> <p>Syntax: window.scroll(coordsPixels)</p>

scrollBy Method	<p>This method is used to scroll the window's content area by the specified number of pixels. This is only useful when there are areas of the document that cannot be seen within the window's current viewing area, and the visible property of the window's scrollbar must be set to true for this method to work.</p> <p>Syntax: window.scrollBy(horizPixels, vertPixels)</p>
scrollTo Method	<p>This method scrolls the contents of a window, the specified co-ordinate becoming the top left corner of the viewable area.</p> <p>Syntax: window.scrollTo(xPosition, yPosition)</p> <p>This method displays a dialog box prompting the user for some input. The optional defaultInput parameter specifies the text that initially appears in the input field.</p> <p>The following example prompts the user for their name and then writes a personalized greeting to the page.</p> <p>Code:</p> <pre><body onload=greeting()> <script type="text/javascript"> function greeting() { y = (prompt("Please enter your name.", "Type name here")) document.write("Hello " + y) } </script></pre> <p>Output:</p> 
setInterval Method	<p>This method is used to call a function or evaluate an expression at specified intervals, in milliseconds.</p> <p>Syntax: indow.setInterval(expression/function, milliseconds)</p>

	<p>This method is used to call a function or evaluate an expression at specified intervals, in milliseconds. This will continue until the clearInterval method is called or the window is closed. The ID value returned by setInterval is used as the parameter for the clearInterval method. Note that if an expression is to be evaluated, it must be quoted to prevent it being evaluated immediately.</p> <p>The following example uses the setInterval method to call the clock() function which updates the time in a text box.</p> <p>Code:</p> <pre><form name="myForm" action="" method="POST"> <input name="myClock" type="Text"> <script language=javascript> self.setInterval('clock()', 50) function clock() { time=new Date() document.myForm.myClock.value=time } </script> </form></pre>
<p>setTimeout Method</p>	<p>This method is used to call a function or evaluate an expression after a specified number of milliseconds.</p> <p>Syntax:</p> <p>window.setTimeout(expression/function, milliseconds)</p> <p>This method is used to call a function or evaluate an expression after a specified number of milliseconds. If an expression is to be evaluated, it must be quoted to prevent it being evaluated immediately. Note that the use of this method does not halt the execution of any remaining scripts until the timeout has passed, it just schedules the expression or function for the specified time.</p>

The following example opens a new window and uses the **setTimeout** method to call the `winClose()` function which closes it after five seconds (5000 milliseconds).

Code:

```
function winClose()
{
  myWindow.close()
}
myWindow = window.open("", "tinyWindow",
'width=150, height=110')
myWindow.document.write("This window will
close automatically after five seconds. Thanks for
your patience")
self.setTimeout('winClose()', 5000)
```

In this example, the **setTimeout** method is used with the **onClick** core attribute in an **input** tag within the **body** element to call a function after five seconds (5000 milliseconds):

```
<html>
<head>
<script type="text/javascript">
function displayAlert()
{
  alert("The GURU sez hi!")
}
</script>
</head>
<body>
<form>
Click on the button.
<br>
After 5 seconds, an alert will appear.
<br>
<input type="button"
onclick="setTimeout('displayAlert()',5000)"
value="Click Me">
</form>
</body>
</html>
```

stop Method	<p>This method is used to cancel the current download. This is the same as clicking the browser's Stop button.</p> <p>Syntax: window.stop()</p>
--------------------	---

EVENT HANDLERS

<u>onBlur</u> Event handler	<p>This event handler executes some specified JavaScript code on the occurrence of a Blur event (when an window loses focus).</p> <p>Syntax: window.onBlur="myJavaScriptCode" Event handler for Button, Checkbox, FileUpload, Layer, Password, Radio, Reset, Select, Submit, Text, TextArea, Window.</p> <p>The onBlur event handler executes the specified JavaScript code or function on the occurrence of a blur event. This is when a window, frame or form element loses focus. This can be caused by the user clicking outside of the current window, frame or form element, by using the TAB key to cycle through the various elements on screen, or by a call to the window.blur method.</p> <p>The onBlur event handler uses the following Event object properties.</p> <p>type - this property indicates the type of event.</p> <p>target - this property indicates the object to which the event was originally sent.</p> <p>The following example shows the use of the onBlur event handler to ask the user to check that the details given are correct. Note that the first line is HTML code.</p> <p>Code: Enter email address <INPUT TYPE="text" VALUE="" NAME="userEmail"</p>
------------------------------------	---

	<pre>onBlur=addCheck(> <script type="text/javascript" language="JavaScript"> function addCheck() { alert("Please check your email details are correct before submitting") } </script></pre>
<p><u>onDragDrop</u> Event handler</p>	<p>This event handler executes some specified JavaScript code on the occurrence of a DragDrop event.</p> <p>Syntax: window.onDragDrop="myJavaScriptCode"</p> <p>Event handler for Window.</p> <p>The onDragDrop event handler executes the specified JavaScript code or function on the occurrence of a DragDrop event. This is when an object, such as a shortcut or file, is dragged and dropped into the browser window. If the event handler returns true, the browser will attempt to load the dropped item into its window, and if false the drag and drop process is cancelled.</p> <p>The onDragDrop event handler uses the following Event object properties.</p> <p>data - this property returns the URLs of any dropped objects as an Array of Strings.</p> <p>type - this property indicates the type of event.</p> <p>target - this property indicates the object to which the event was originally sent.</p> <p>screenX - the cursor location when the click event occurs.</p> <p>screenY - the cursor location when the click event occurs.</p> <p>modifiers - lists the modifier keys (shift, alt, ctrl, etc.) held down when the click event occurs.</p>

<u>onError</u> handler	Event	<p>This event handler executes some specified JavaScript code on the occurrence of an Error event</p> <p>Syntax: window.onError="myJavaScriptCode" Event handler for Image, Window.</p> <p>The onError event handler executes the specified JavaScript code or function on the occurrence of an error event. This is when an image or document causes an error during loading. The distinction must be made between a browser error, when the user types in a non-existent URL, for example, and a JavaScript runtime or syntax error. This event handler will only be triggered by a JavaScript error, not a browser error.</p> <p>As well as the onError handler triggering a JavaScript function, it can also be set to onError="null" which suppresses the standard JavaScript error dialog boxes. To suppress JavaScript error dialogs when calling a function using onError, the function must return true (example 2 below demonstrates this).</p> <p>There are two things to bear in mind when using window.onerror. Firstly, this only applies to the window containing window.onerror, not any others, and secondly, window.onerror must be spelt all lower-case and contained within <script> tags; it cannot be defined in HTML (this obviously doesn't apply when using onError with an image tag, as in example 1 below).</p> <p>The onFocus event handler uses the following Event object properties.</p>
---	--------------	---

	<p>type - this property indicates the type of event.</p> <p>target - this property indicates the object to which the event was originally sent. The first example suppresses the normal JavaScript error dialogs if a problem arises when trying to load the specified image, while example 2 does the same, but applied to a window, by using return true in the called function, and displays a customized message instead.</p> <p>Code: <pre></pre></p> <p>Code: <pre><script type="text/javascript" language="JavaScript"> s1 = new String(myForm.myText.value) window.onerror=myErrorHandler function myErrorHandler() { alert('A customized error message') return true } </script> <body onload=nonexistantFunc(></pre></p>
<p><u>onFocus</u> handler</p>	<p>Event</p> <p>This event handler executes some specified JavaScript code on the occurrence of a Focus event.</p> <p>Syntax: window.onFocus="my JavaScriptCode"</p> <p>Event handler for <u>Button</u>, <u>Checkbox</u>, <u>FileUpload</u>, <u>Layer</u>, <u>Password</u>, <u>Radio</u>, <u>Reset</u>, <u>Select</u>, <u>Submit</u>, <u>Text</u>, <u>TextArea</u>, <u>Window</u>.</p> <p>The onFocus event handler executes the specified JavaScript code or function on the occurrence of a focus event. This</p>

is when a window, frame or form element is given focus. This can be caused by the user clicking on the current window, frame or form element, by using the TAB key to cycle through the various elements on screen, or by a call to the **window.focus** method. Be aware that assigning an alert box to an object's **onFocus** event handler will result in recurrent alerts as pressing the 'o.k.' button in the alert box will return focus to the calling element or object. The **onFocus** event handler uses the following **Event** object properties

type - this property indicates the type of event.

target - this property indicates the object to which the event was originally sent. The following example shows the use of the **onFocus** event handler to replace the default string displayed in the text box. Note that the first line is HTML code and it is accepted that the text box resides on a form called 'myForm'.

Code:

```
<input type="text" name="myText"
value="Give me focus" onFocus =
"changeVal()">
<script type="text/javascript"
language="JavaScript">
s1 = new String(myForm.myText.value)
function changeVal() {
    s1 = "I'm feeling focused"
    document.myForm.myText.value =
s1.toUpperCase()
}
</script>
```


<p><u>onload</u> Event handler</p>	<p>This event handler executes some specified JavaScript code on the occurrence of a Load event.</p> <p>Syntax: window.onload="myJavaScriptCode" Event handler for <u>Image</u>, <u>Layer</u> and <u>Window</u>.</p> <p>The onload event handler executes the specified JavaScript code or function on the occurrence of a Load event. A Load event occurs when the browser finishes loading a window or all the frames in a window.</p> <p>The onload event handler uses the following Event object properties.</p> <p>type - this property indicates the type of event.</p> <p>target - this property indicates the object to which the event was originally sent.</p> <p>width - when the event is over a window, not a layer, this represents the width of the window.</p> <p>height - when the event is over a window, not a layer, this represents the height of the window</p> <p>The following example shows the use of the onload event handler to display a message in the text box.</p> <p>Code:</p> <pre><body onload = "changeVal()" > <form action="" method="POST" id="myForm" > <input type="text" name="myText" > <script type="text/javascript" language="JavaScript"> s1 = new String(myForm.myText.value) function changeVal() {</pre>
---	---

	<pre>s1 = "Greetings!" myForm.myText.value = s1.toUpperCase() } </script> </form> </body></pre>
<p><u>onMove</u> handler</p>	<p>Event</p> <p>This event handler executes some specified JavaScript code on the occurrence of a Move event.</p> <p>Syntax: window.on Move=" my Java Script Code"</p> <p>Event handler for <u>Window</u></p> <p>The onMove event handler is used to execute specified Javascript code whenever the user or the script moves a window or frame. It uses the following properties of the <u>Event</u> object:</p> <p>type - indicates the type of event.</p> <p>target - indicates the target object to which the event was sent.</p> <p>screenX, screenY - indicates the position of the top left corner of the window or frame.</p>
<p><u>onResize</u> handler</p>	<p>Event</p> <p>This event handler executes some specified JavaScript code on the occurrence of a Resize event.</p> <p>Syntax:</p> <p>window.onResize="myJavaScript Code"</p> <p>Event handler for <u>Window</u></p> <p>The onResize even handler is use to execute specified code whenever a user or script resizes a window or frame. This allows you to query the size and position of window elements, dynamically reset SRC properties etc. It uses the following properties of the <u>Event</u> object:</p> <p>type - indicates the type of event.</p>

	<p>target - indicates the target object to which the event was sent.</p> <p>width, height - indicates the width or height of the window or frame</p>
<p><u>onUnload</u> handler Event</p>	<p>This event handler executes some specified JavaScript code on the occurrence of an Unload event.</p> <p>Syntax: window.onUnload="myJavaScriptCode"</p> <p>Event handler for <u>Window</u></p> <p>The onUnload event handler is used to run a function or JavaScript code whenever the user exits a document. The onUnload event handler is used within either the <BODY> or the <FRAMESET> tag, and uses the following properties of the <u>Event</u> object:</p> <p>type - indicates the type of event target - indicates the target object to which the event was sent.</p> <p>The following example shows the onUnload event handler being used to execute the 'finishOff' function:</p> <p>Code: <BODY onUnload="finishOff()"></p> <p>Compare to the <u>onload</u> event handler.</p>



WEBDESIGN CONCEPT

Unit Structure

15.1 How the website should be

- Basic rules of Web Page design
- Types of Website

What Is Good Web Design?

Before you read about the process of building Web pages, this section helps you define your goal clearly. What, exactly, is good Web design? Some people discuss what *isn't* good Web design (www.webpagesthatsuck.com), but this really doesn't demonstrate how to create good Web sites. Others like to discuss aesthetics and layout (www.highfive.com). This may be appropriate on a superficial level, but beauty is often in the eye of the beholder. Looks aren't everything. Function is important, too, and some people even claim that the answer to what constitutes good Web design is purely a matter of function. If it isn't usable (www.useit.com), then it isn't reasonable—but function without motivating form is boring. Some talk too much about success, citing numerous visitors as true validation of a site's design. This assumes that the Web is primarily about popularity. Who cares how many visitors come to a page, unless it has some benefit? Think about quality and success. If serving the most burgers says anything about making good hamburgers, then McDonald's makes the world's best hamburger. This kind of logic gets people in trouble on the Web all the time. Consider whether economically successful or trendy Web pages are well designed. Characterizing good Web design is not easy, especially because it depends largely on your target audience. Most Web discussions lose sight of the big picture, placing too much emphasis on how pages look, and not enough emphasis on their content, purpose, functionality, or the user's experience. Web design is not just graphic design. Web design

includes graphic design. Other important aspects of the Web design process may include such areas as the following:

- _ Artistic style, color theory, typography, and other visual concerns
- _ Information design, which specifies how information should be organized and linked
- _ Hypertext theory
- _ Technical writing
- _ System design
- _ Programming
- _ Network and server design
- _ Business issues and project management

Obviously, many disciplines are part of Web design. The first requirement, however, is a clear understanding of the site's ultimate purpose. The goal of a Web designer is to produce a usable and appealing visual design for a software system, in the form of a Web site that helps a user fulfill some goal. In other words, the goal is to develop a site that can be delivered to the user in a satisfactory manner, be interpreted correctly by the user, and induce the desired outcome. Web design should be concerned not only with the aesthetic qualities of a Web site, but also with the user's overall experience in the context of a specific task or problem. The focus is on how something can be done, not just on how it looks. It is easy to throw out expressions like "perception is reality" or "content is king" as arguments for or against focusing on the visual nature of the Web. However, the reality is a balance between these extreme points of view. If you skimp on graphics, the site may seem boring. If you provide a wonderful interface, but skimp on content, the user may leave to find a site with more information. If you forget to debug, you may send the user angrily away, facing error dialog boxes. Remember: experience is vital. Always consider what feeling the user will take away after visiting your site. A sense of accomplishment? Frustration? Understanding? Disgust? The best approach to Web design is a holistic one, in which content, presentation, and interactivity work in harmony. So, how can you make a Web site that is both functional and visually appealing, without exceeding the constraints of the Internet and Web technologies?

Putting a page together with HTML and then sprucing it up with a few colored balls, a rainbow-color bar, and animated clip art

doesn't help. The page looks slapped together, and the graphics provide little more than extra eye-catching glitz. In this case, the background plain interferes with the user's ability to read the text. On the other hand, focusing too much on the visual aspects leads to online brochures with slow-downloading, full-screen images. Everything is created with graphic composition tools, such as Photoshop, which provide nearly absolute layout control, but result in huge files. Text on such a page can't be changed without a graphic designer, let alone be indexed by a Web search engine. This design also excludes those who surf with images turned off, use a text browser, or are disabled and simply can't see your images. Even worse, the site may not scale on a high-resolution monitor, causing it to be so small that it's unreadable. Many large sites fall into this trap because they never test their pages over a dial-in link. A page that seems to work well over the local Ethernet network may take ages to load over a 56Kbps modem connection. The average modem user still connects at that speed (or a lower speed), and most users are not willing to wait forever for your page to load before they give up and move on to less-bandwidth-intensive sites. Again, balance is the issue. Sometimes, stark pages are okay. Other times, full-screen images make sense. The form of a site depends on its goals. Figuring out the site's audience and what its ultimate goals are before diving into HTML coding seems obvious, but it isn't always the approach adopted. Unfortunately, once the simplicity of HTML is revealed, many eager authors quickly mark up pages and then try to improve them by adding graphics. At the other extreme, designers may ruin the site just by thinking more about the user interface than about what is actually delivered. Creating Web sites requires a process, not an ad hoc decision to focus more on visuals or more on content.

5 Basic Rules of Web Page Design and Layout

Your Web Site Should Be Easy to Read

The most important rule in web design is that your web site should be easy to read. What does this mean? You should choose your text and background colors very carefully. You don't want to use backgrounds that obscure your text or use colors that are hard to read. Dark-colored text on a light-colored background is easier to read than light-colored text on a dark-colored background.

You also don't want to set your text size too small (hard to read) or too large (it will appear to shout at your visitors). All capitalized letters also give the appearance of shouting at your visitors.

Keep the alignment of your main text to the left, not centered. Center-aligned text is best used in headlines. You want your visitors to be comfortable with what they are reading, and most text (in the West) is left aligned.

Your Web Site Should Be Easy to Navigate

All of your hyperlinks should be clear to your visitors. Graphic images, such as buttons or tabs, should be clearly labeled and easy to read. Your web graphic designer should select the colors, backgrounds, textures, and special effects on your web graphics very carefully. It is more important that your navigational buttons and tabs be easy to read and understand than to have "flashy" effects.

Link colors in your text should be familiar to your visitor (blue text usually indicates an unvisited link and purple or maroon text usually indicates a visited link), if possible. If you elect not to use the default colors, your text links should be emphasized in some other way (boldfaced, a larger font size, set between small vertical lines, or a combination of these). Text links should be unique -- they should not look the same as any other text in your web pages. You do not want people clicking on your headings because they think the headings are links.

Your visitors should be able to find what they are looking for in your site within three clicks. If not, they are very likely to click off your site as quickly as they clicked on.

Your Web Site Should Be Easy to Find

How are your visitors finding you online? The myth, "If I build a web site, they will come," is still a commonly held belief among companies and organizations new to the Internet. People will not come to your web site unless you promote your site both online and offline.

Web sites are promoted online via search engines, directories, award sites, banner advertising, electronic magazines (e-zines)

and links from other web sites. If you are not familiar with any of these online terms, then it is best that you have your site promoted by an online marketing professional. (See our section, What to Look for in an Online Marketing Company, for some general guidelines.)

Web sites are promoted offline via the conventional advertising methods: print ads, radio, television, brochures, word-of-mouth, etc. Once you have created a web site, all of your company's printed materials including business cards, letterhead, envelopes, invoices, etc. should have your URL printed on them.

Your Web Pages' Layout Should Be Consistent Throughout the Site

Just as in any document formatted on a word processor or as in any brochure, newsletter, or newspaper formatted in a desktop publishing program, all graphic images and elements, typefaces, headings, and footers should remain consistent throughout your web site. Consistency and coherence in any document, whether it be a report or a set of web pages, project a professional image.

For example, if you use a drop shadow as a special effect in your bullet points, you should use drop shadows in all of your bullets. Link-colors should be consistent throughout your web pages. Typefaces and background colors, too, should remain the same throughout your site.

Color-coded web pages, in particular, need this consistency. Typefaces, alignment in the main text and the headings, background effects, and the special effects on graphics should remain the same. Only the colors should change.

Overall Web Page Size Should be 75K or Less

Studies have indicated that visitors will quickly lose interest in your web site if the majority of a page does not download within 15 seconds. (Artists' pages should have a warning at the top of their pages.) Even web sites that are marketed to high-end users need to consider download times. Sometimes, getting to web site such as Microsoft or Sun Microsystems is so difficult and time consuming that visitors will often try to access the sites

during non-working hours from their homes. If your business does not have good brand name recognition, it is best to keep your download time as short as possible.

A good application of this rule is adding animation to your site. Sure, animation looks "cool" and does initially catch your eye, but animation graphics tend to be large files. Test the download time of your pages first. If the download time of your page is relatively short and the addition of animation does not unreasonably increase the download time of your page, then and ONLY then should animation be a consideration.

Finally, before you consider the personal preferences of your web page design, you should consider all of the above rules FIRST and adapt your personal preferences accordingly. The attitude "I don't like how it looks" should always be secondary to your web site's function. Which is more important: creative expression corporate image or running a successful business?

Types of websites

There are three website types:

- Content (information)
- E-Commerce (online sales)
- Interaction (Blogs, Bulletin Boards, Chat Rooms, and gaming sites).

Website types are implemented as dynamic or static:

- Dynamic websites have frequently changing content or interact with the visitor. Dynamic websites typically use server side programming to generate HTML code as requested.
- Static websites are written in pure HTML perhaps with a bit of JavaScript and only change when manually updated.

It's common to see combinations of the three types as well as combinations of dynamic and static. It's important to understand what they are and what works for you!

Content or information websites may be dynamic or static and the implementation depends upon how frequently the website information changes. News sites and search engines are dynamic

database driven websites to allow rapid information update. Many corporate websites are static but that is changing rapidly.

E-commerce sites are almost always dynamic allowing for frequent product changes, pricing changes, sales and inventory updates. Simple e-commerce transactions like membership applications and online payment may be interactive while the main website is still static.

Interaction sites (Blogs, Bulletin Boards, Chat Rooms, and gaming sites) are dynamic.

Websites can be a combination of Content, E-Commerce and Interactive as well as a combination of dynamic and static. It's common to see a combination of dynamic and static implementations implementations and and combination of types. Because of this, more website owners are moving toward dynamic pages.

Pictures and graphics are always good to liven up a website. You should have at least some because the phrase "one picture is worth a thousand words" is as true now as when it was coined.

